Trusted Execution Environment with Silicon Level Root-of-Trust based on RISC-V Computer System

TRONG-THUC HOANG

Department of Computer and Network Engineering

The University of Electro-Communications

A dissertation submitted for the degree of
Doctor of Engineering

March 24, 2022

This page intentionally left blank.

Trusted Execution Environment with Silicon Level Root-of-Trust based on RISC-V Computer System

APPROVED

This page intentionally left blank.

This page intentionally left blank.

I would like to dedicate this dissertation to my entire family, who encouraged me to pursue higher education.

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

— Alan Turing

This page intentionally left blank.

Acknowledgements

First of all, I would like to express my deep respect and appreciation to my advisors, Prof. Cong-Kha PHAM and Prof. Koichiro ISHIBASHI, for their mentorship throughout the research process. I would also like to express my sincere appreciation to the committee members, including Prof. Akashi SATOH, Prof. Kazuo SAKIYAMA, and Assoc. Prof. Takeshi SUGAWARA, for their critical questions and valuable comments on this dissertation.

Second, the completion of this dissertation is due largely to the support of my friends in the PHAM laboratory. I would like to thank CKRISTIAN Duran for his sincere assistance. I also thank Ba-Anh DAO, Anh-Tien LE, Binh Kieu-Do-NGUYEN Khai-Duy NGUYEN, DANG Tuan Kiet, Marco SARMIENTO, and Ronaldo SERRANO for their countless hours of brainstorming and helping. I also thank all of my Vietnamese friends, Japanese friends, as well as other international friends for keeping my spirits up.

Third, I would like to thank Dr. Kuniyasu SUZAKI and Mr. Akira TSUKAMOTO, researchers at the National Institute of Advanced Industrial Science and Technology (AIST), for their guidance and support. I also thank all Cyber-Physical Security Research Center (CPSEC) members at the AIST for their cooperation.

Fourth, the research was supported by the New Energy and Industrial Technology Development Organization (NEDO). It was also supported by the VLSI Design and Education Center (VDEC), the University of Tokyo, collaborating with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc.

Fifth, I sincerely thank the University of Electro-Communications and the Ministry of Education, Culture, Sports, Science, and Technology for providing me with an excellent opportunity to experience my study and life in Japan.

Last but not least, I am grateful to my entire family, especially my father, Hoang Van Manh, my mother, Phan Hoang Yen Phuong, and my little sister, Hoang Ngoc Thanh Truc, for offering their love and sharing their wisdom with me throughout my life.

This page intentionally left blank.

和文要旨

RISC-Vコンピュータシステムに基づくシリコンレベルの信頼ルートを有する信頼できる実行環境

ホン　チョン　トゥク

信頼できる実行環境（TEE）は、認証されていないプログラム等の実行を防ぐための安全な環境を提供することを目的としている。TEEの強度は、一連のハッシュ、署名、および検証によって作成された信頼の鎖（CoT）に依存する。CoTの開始点は、Root-of-Trust（RoT）と呼ばれる。今日、RISC-Vのオープンソースの命令セットアーキテクチャ（ISA）は、セキュリティの目的でコンピュータシステムを再検討する機会を提供している。RISC-Vを使用することでシリコンレベルまで前述のような安全なハードウェアRoTを作成できる。このRoTにおいて、理論的には、ルートキーをハッキングすることは、チップ製造プロセスに干渉することを意味する。

本論文の目的は、安全なハードウェアRoTを備えたRISC-VベースのTEEコンピュータシステムを提案する。提案したSoCはLinuxで起動可能であり、TEEの構築に必要な複数のハードウェア暗号化アクセラレータが含まれている。本SoCの目標は、代替ハードウェアを使用してTEEを完全に起動し、ブートローダーソフトウェアから暗号化機能を保護することでである。提案アーキテクチャは、Linux対応プロセッサと32ビットの隠しMCUを組み合わせたヘテロジニアス・システム・アーキテクチャ（複数の種類のプロセッサ）である。32ビットMCUはTEE側から分離され、ルートキーなどの機密データ、およびZero Stage BootLoader（ZSBL）やキー生成プログラムなどの機密動作を処理する。分離されたサブシステム内にRoTが実装されているため、起動後にTEE側からRoTにアクセスすることはできなくなる。さらに、MCUがブートシーケンスを実行するため、キーの生成は柔軟なアルゴリズムを使用するため、任意のセキュリティスキームに合わせて更新可能となる。

本論文は、安全性の高いコンピュータシステムを提案することだけでなく、将来使用するためのフレームワークにも注目する。提案フレームワークは使いやすく、長期的に将来の脅威に適応するためのある程度の柔軟性を有している。 RISC-Vアーキテクチャの柔軟性により、ユーザーはセキュリティ機能をオンまたはオフにできるため、コストとセキュリティの懸念のバランスをとることが可能となる。また、フレームワークをオープンソースプロジェクトとして維持することにより、本提案は、今後数年間、コンピュータセキュリティ開発者にとって強力な開発ツールになると期待できる。

This page intentionally left blank.

Abstract

Trusted Execution Environment (TEE) aims to provide a secure environment to prevent unauthenticated code execution. Currently, this is the most common security extension for a secure Operating System (OS). Several major vendors have developed and released TEEs. The strength of TEE depends on the Chain-of-Trust (CoT) created by a series of hashes, signatures, and verifications. The starting point of CoT is called the Root-of-Trust (RoT), and it is often hard-coded or locally generated before boot. TEE uses multiple cryptographic functions during boot to authenticate the bootloader and then validate other sensitive data and applications such as OS-related software. The certification of this layer is signed by the previous lower layer and verified by the next upper layer. By this, the generated certificates of all layers are linked together to create the CoT. As a result, only a trusted application with a valid authentication can gain the intended privileges after boot. Untrusted codes and infected codes will lose their valid signatures, making them impossible to achieve any special right.

Typical TEEs nowadays are based on closed-source systems tailored for a specific hardware of the relevant manufacturer. Thus, many innovative ideas are stuck with limited options; and many of them have to work around the given limitations. Nowadays, the open-source Instruction Set Architecture (ISA) of RISC-V has created a chance to revisit the computer system for security purposes. RISC-V ISA is developed based on the idea that it needs only a minimal integer instruction set to operate. And then, many extensions can be added to the basic set to satisfy any system requirement. The RISC-V Foundation and the open-source community have developed a new toolset, some of which uses the Scala language with the Chisel library to generate a Register Transfer Level (RTL) code. The generated RTL code can be used for Very Large Scale Integrated (VLSI) implementations. This approach allows developers to share and reuse each other's designs. The processors and computer systems can be shared and reconfigured as we wanted, thus significantly reducing the research and development time compared to the conventional approach. Furthermore, the open-source community continuously researches and releases new ISA extensions for many different applications. With RISC-V, it is possible to create such a secure hardware RoT, even down to the silicon level. With this RoT, in theory, hacking the root key means interfering with the chip manufacturing process.

Although the secure boot process is not new, it needs to be constantly reviewed and updated to fight newly found attack vectors. TEE is just an isolated environment, and it shouldn't be the RoT or does the secure boot. Hence, secure boot with hardware RoT in TEE is still an ongoing problem that needs research and development. In this dissertation, the primary goal is to present a RISC-V-based architecture that can do the secure boot for TEE with a flexible boot program while wholly isolated from the TEE processors. The proposed design supports various cryptographic accelerators necessary for the secure boot process. The main idea is a heterogeneous design by combining 64-bit or 32-bit Linux-capable processors with a 32-bit hidden Micro-Controller Unit (MCU). The 32-bit MCU is isolated from the TEE-side, and it will take care of the sensitive data, like the root key, and sensitive activities, such as Zero Stage BootLoader (ZSBL) and keys generation program. With RoT implemented inside the isolated sub-system, the RoT is inaccessible from the TEE-side after boot by any means. Furthermore, with the MCU to do the boot sequence, the keys generation is flexible and could be updated to fit any security scheme. Two chips were made for the proposed implementation, including one 32-bit version and one 64-bit version; both are ROHM-180nm.

Besides the presented architectures, the secondary goal of this proposal is the framework that is easy to use and has a certain degree of flexibility to adapt to future threats in the long term. By using such a framework, security developers can increase the security level of their systems by continuing to add state-of-the-art protection methods. And the flexibility of the RISC-V architecture allows users to turn on or off any security feature, resulting in balancing between cost and security concerns. Finally, by maintaining the framework as an open-source project, the framework is expected to be a good development tool for computer security developers over the years to come.

Contents

This page intentionally left blank.

List of Abbreviations

ABI	**A**pplication **B**inary **I**nterface
ADC	**A**nalog-to-**D**igital **C**onverter
AEAD	**A**uthenticated **E**ncryption with **A**ssociated **D**ata
AEE	**A**pplication **E**xecution **E**nvironment
AES	**A**dvanced **E**ncryption **S**tandard
ALM	**A**daptive **L**ogic **M**odule
AMT	**A**ctive **M**anagement **T**echnology
APB	**A**dvance **P**eripheral **B**us
API	**A**pplication **P**rogramming **I**nterface
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
AXI4	**A**dvanced e**X**tensible **I**nterface **4**
BBL	**B**erkeley **B**oot**L**oader
BIOS	**B**asic **I**nput/**O**utput **S**ystem
BMC	**B**aseboard **M**anagement **C**ontroller
BMT	**B**onsai **M**erkle **T**ree
BOOM	**B**erkeley **O**ut-of-**O**rder **M**achine
BTB	**B**ranch **T**arget **B**uffer
CA	**C**ertificate **A**uthority
CAU	**C**ode **A**uthentication **U**nit
CLINT	**C**ore **L**ocal **INT**errupt
CMOS	**C**omplementary **M**etal **O**xide **S**emiconductor
CORDIC	**CO**ordinate **R**otation **DI**gital **C**omputer
CoT	**C**hain-**o**f-**T**rust
CPU	**C**entral **P**rocessing **U**nit
CURE	**CU**stomizable and **R**esilient **E**nclaves
CSR	**C**ontrol **S**tatus **R**egister
DC	**D**esign **C**ompiler
DDR	**D**ouble **D**ata **R**ate
DIMM	**D**ual **I**n-line **M**emory **M**odule
DMA	**D**irect **M**emory **A**ccess
DMIPS	**D**hrystone **M**illion **I**nstructions **P**er **S**econd
D-mode	**D**ebug **mode**
DoS	**D**enial-**o**f-**S**ervice
DSP	**D**igital **S**ignal **P**rocessing
EC	**E**lliptic **C**urve
ECDSA	**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
EdDSA	**Ed**wards-curve **D**igital **S**ignature **A**lgorithm
EPC	**E**nclave **P**age **C**ache
FD-SOI	**F**ully-**D**epleted **S**ilicon-**O**n-**I**nsulator

FIRRTL	**F**lexible **I**ntermediate **R**epresentation for **RTL**
FLL	**F**requency **L**ocked **L**oop
$\mathbf{F_{Max}}$	Maximum operating frequency
FPGA	**F**ield-**P**rogrammaple **G**ate **A**rray
FPU	**F**loating-**P**oint **U**nit
FSBL	**F**irst **S**tage **B**oot**L**oader
FSM	**F**inite **S**tate **M**achine
GCM	**G**alois **C**ounter **M**ode
GIC	**G**eneric **I**nterrupt **C**ontroller
GIDL	**G**ate-**I**nduced **D**rain **L**eakage
GPIO	**G**eneral-**P**urpose **I**n/**O**ut
HBI	**H**ypervisor **B**inary **I**nterface
HDL	**H**ardware **D**escription **L**anguage
HEE	**H**ypervisor **E**xecution **E**nvironment
HMAC	**H**ashed **M**essage **A**uthentication **C**ode
I/O	**I**n/**O**ut
IBBE	**I**dentity-**B**ased **B**roadcasting **E**ncryption
IBus	**I**solated **Bus**
IC	**I**ntegrated **C**ircuit
ICC	**I**ntegrated **C**ircuit **C**ompiler
IDE	**I**ntegrated **D**evelopment **E**nvironment
IEC	**I**nternational **E**lectrotechnical **C**ommission
IoT	**I**nternet-**o**f-**T**hings
IP	**I**ntellectual **P**roperty
ISA	**I**nstruction **S**et **A**rchitecture
ISO	**I**nternational **O**rganization for **S**tandardization
JTAG	**J**oint **T**est **A**ction **G**roup
KDF	**K**ey **D**erivation **F**unction
KMU	**K**ey **M**anagement **U**nit
LLC	**L**ast **L**evel **C**ache
LOC	**L**ines **O**f **C**ode
MAC	**M**essage **A**uthentication **C**ode
MBus	**M**emory **Bus**
MCU	**M**icro-**C**ontroller **U**nit
MIPS	**M**icroprocessor without **I**nterlocked **P**ipelined **S**tages
M-mode	**M**achine **mode**
MMC	**M**ulti**M**edia **C**ard
MMU	**M**emory **M**anagement **U**nit
MPU	**M**emory **P**rotection **U**nit
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NS	**N**on-**S**ecure
OpenOCD	**Open** **O**n-**C**hip **D**ebugger
OS	**O**perating **S**ystem
OTP	**O**ne-**T**ime **P**rogrammable
PC	**P**ersonal **C**omputer
PBus	**P**eripheral **Bus**
PCB	**P**rinted **C**ircuit **B**oard

PCH	**P**latform **C**ontroller **H**ub
PGA	**P**in **G**rid **A**rray
PLIC	**P**latform **L**evel **I**nterrupt **C**ontroller
PLL	**P**hase **L**ocked **L**oop
PMA	**P**hysical **M**emory **A**ttributes
PMP	**P**hysical **M**emory **P**rotection
PnR	**P**lace a**n**d **R**oute
PRM	**P**rocessor **R**eserved **M**emory
PSP	**P**latform **S**ecurity **P**rocessor
PTW	**P**age **T**able **W**alker
PUF	**P**hysical **U**nclonable **F**unction
PULP	**P**arallel **U**ltra-**L**ow **P**ower
QEMU	**Q**uick **EMU**lator
QFP	**Q**uad **F**lat **P**ackage
RAM	**R**andom **A**ccess **M**emory
REE	**R**ich **E**xecution **E**nvironment
RISC-V	**R**educded **I**nstruction **S**et **C**omputer - Five
RNG	**R**andom **N**umber **G**enerator
ROM	**R**ead-**O**nly **M**emory
RoT	**R**oot-**o**f-**T**rust
RSA	**R**ivest-**S**hamir-**A**dleman
RTL	**R**egister **T**ransfer **L**evel
RVSCP	RISC-V **S**ecure **C**o**P**rocessor
SAR	**S**uccessive **A**pproximation **R**egister
SBI	**S**upervisor **B**inary **I**nterface
SBus	**S**ystem **Bus**
SCA	**S**ide-**C**hannel **A**ttack
SD	**S**ecure **D**igital
SDK	**S**oftware **D**evelopment **K**it
SDRAM	**S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory
SEE	**S**upervisor **E**xecution **E**nvironment
SerDes	**Ser**ializer/**Des**erializer
SEV	**S**ecure **E**ncrypted **V**irtualization
SEV-ES	**SEV** **E**ncrypted **S**tate
SEV-SNP	**SEV** **S**ecure **N**ested **P**aging
SGX	**S**oftware **G**uard e**X**tensions
SHA	**S**ecure **H**ashing **A**lgorithm
SM	**S**ecure **M**onitor
SMA	**S**ub**M**iniature version **A**
S-mode	**S**upervisor **mode**
SoC	**S**ystem-**o**n-**C**hip
SOTB	**S**ilicon **O**n **T**hin **B**uried oxide
SPI	**S**erial **P**arallel **I**nterface
SRAM	**S**tatic **R**andom **A**ccess **M**emory
STI	**S**hallow **T**rench **I**solation
TCB	**T**rusted **C**omputing **B**ase
TEE	**T**rusted **E**xecution **E**nvironment

TEE-HW	**T**rusted **E**xecution **E**nvironment **H**ard**W**are
TF	**T**rusted **F**irmware
TLB	**T**ranslation **L**ookaside **B**uffer
TPM	**T**rusted **P**latform **M**odule
TRNG	**T**rue **R**andom **N**umber **G**enerator
TZASC	**T**rust**Z**one **A**ddress **S**pace **C**ontroller
TZMA	**T**rust**Z**one **M**emory **A**dapter
TZPC	**T**rust**Z**one **P**rotection **C**ontroller
UART	**U**niversal **A**synchronous **R**eceiver/**T**ransmitter
UEFI	**U**nified **E**xtensible **F**irmware **I**nterface
ULP	**U**ltra **L**ow **P**ower
U-mode	**U**ser **mode**
UVM	**U**niversal **V**erification **M**ethodology
V_{BB}	Back-gate bias voltage
V_{DD}	Power supply
VLSI	**V**ery **L**arge **S**cale **I**ntegrated circuit
VM	**V**irtual **M**achine
V_{TH}	Threshold voltage
WG	**W**orld**G**uard
ZSBL	**Z**ero **S**tage **B**oot**L**oader

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What Is Trusted Execution Environment?

A typical boot process in a computer system usually starts with the Baseboard Management Controller (BMC) or Platform Controller Hub (PCH) on the motherboard. BMC or PCH prepares the necessary hardware configurations and allows the Central Processing Unit (CPU) to boot. The first thing that the CPU does is load the boot firmware (often from the flash outside) like in the Unified Extensible Firmware Interface (UEFI) boot flow [17]. The boot firmware then prepares the boot sector and loads the bootloader into the main memory. Initial Operating System (OS) settings are ready by the bootloader, and then the actual OS image is loaded into the system memory. Finally, the machine boots into the loaded OS. The whole process is like "a relay race where one team member passes a baton to another to win the race" [18], where you trust each member in the team to do their part correctly. However, trust in the computer world needs more than that. As a result, a trust mechanism is necessary. A mechanism that you can verify in each step of the boot sequence that nothing is compromised. Then, a Trusted Execution Environment (TEE) can be built based on that trust.

TEE models, commonly known as the Trusted Computing Base (TCB), were proposed and improved from time to time. The most common TEE implementations are the Intel Software Guard eXtensions (SGX) [1, 19–21] and its variations (i.e., Haven [22], Graphene [23], and Scone [24]), ARM TrustZone [2, 25] and its modifications (i.e., Komodo [26], OP-TEE [27], and Sanctuary [28]), and AMD Secure Encrypted Virtualization (SEV) [3] with its descendants (i.e., AMD SEV-ES [29] and AMD SEV-SNP [30]). However, they are closed-source TEEs; thus, we cannot modify their boot flow or the hardware primitives. Recently, with RISC-V emerging, many new RISC-V-based TEE models were proposed, such as Hex-Five MultiZone [4], Sanctum [5], TIMBER-V [6], Keystone [7], and CUstomizable and Resilient Enclaves (CURE) [8]. Nowadays, nearly all smartphones have a TEE-like feature, and multiple companies, from software to hardware, market their products with built-in security features. In academics, many major security conferences publish their works related to TEE every year. Clearly, the demand for TEE is ever-increasing, especially in the Internet-of-Things (IoT) era.

The design philosophy of TEE is to isolate trusted codes and untrusted codes by a divide-and-conquer approach. In most cases, the isolation was done using privilege separation, thus creating a barrier between programs. Nowadays, recent TEE models are gearing up with barrier enforcers implemented in software and hardware at many architectural levels. The end goal of TEE is to allow only the authenticated codes to run; unauthenticated code cannot run on the trusted side or gain any privilege.

The strength of TEE relies on the Chain-of-Trust (CoT), which is a series of cryptographic functions such as hashing, signing, verification, and encryption/decryption processes. The CoT is created by authenticating a layer signed by the previous lower layer (higher privilege) and making the subsequent signature for the next higher layer (lower privilege) to verify. The foundation of CoT is the very first authentication of the system at reset, and it is called the Root-of-Trust (RoT). For security reasons, the RoT should be inaccessible from the Rich Execution Environment (REE) or even the TEE processors after boot. RoT could be anything from a randomly generated value, an asymmetric secret key, to a device's certificate pre-signed by a root Certificate Authority (CA). For simplicity, in most crypto-systems, the RoT is often elaborated from a hard-coded Read-Only Memory (ROM) or generated locally before boot. As a result, the confidentiality of the secret root key and the integrity of the whole TEE system rely on the secure boot process to establish the RoT.

1.2 What Is RISC-V?

From the beginning of the 21st century, the Reduced Instruction Set Computer (RISC) architecture was already dominant in the mobile marketplace because of its low-power and low-cost characteristics [31]. There were RISC-based CPUs like ARM CPUs in most of the hand-held devices [32], Microprocessor without Interlocked Pipelined Stages (MIPS) based CPUs in most of the gaming consoles [33]. And very recently, an open-source community about RISC-V Instruction Set Architecture (ISA) was emerged [13, 16]. The development of RISC-V was expanding and turning the silicon industry more efficiently than ever. Compared to the conventional Integrated Circuit (IC) development flow, the RISC-V ecosystem is like "one barbarian is at the gates with a refurbished siege engine" [34]. Up to now, there are plenty of RISC-V processors that have been presented in both academic and industrial forums [35–39]. Some worth-mention works are the highly customizable Rocket cores of the Berkeley architecture group [40], the high-performance 32-bit E-core series [41] and 64-bit U-core series [42] of the SiFive Inc., and the 32-bit RI5CY cores [43] and 64-bit Ariane cores [44] of the PULP-platform research group.

RISC-V is an open-source ISA, the interface between software and hardware developers. It was first presented by the Berkeley architecture group in 2014 [45], and now it is maintained by the RISC-V Foundation group [46]. The primary goal of the RISC-V Foundation is to provide a completely open ISA to support the research, development, and education in both academia and industry areas. The idea of RISC-V is that it needs only the base integer instruction set to operate. Then, depending on the requirements, more ISA extensions can be added. This is to avoid the "over-architecting" in micro-architecture [16]. Currently, the ISA can support 32-bit, 64-bit, and 128-bit addressing spaces. The most used ISA extensions are the "M" for multiplication and division, the "A" for atomic, the "F" for floating-point, the "D" for double floating-point, and the "C" for compressed instruction sets. Together, the default RISC-V's ISA is the RV64IMAFDC, also called the RV64GC.

The RISC-V ecosystem is rich with many development tools for software and hardware designers. Among those tools is the novel idea of using Scala language with the Chisel library to generate a Register Transfer Level (RTL) code. The chisel generator first generates an intermediate RTL representation named Flexible Intermediate Representation for RTL (FIRRTL) [47] and then converts the FIRRTL to conventional Verilog Hardware Description Language (HDL). The generated Verilog file can be used for Field-Programmable Gate Array (FPGA) and Very Large Scale Integrated circuit (VLSI) implementations [48, 49]. This approach created a new playground for developers to share and reuse each other designs with virtually no effort [50]. As a result, the whole computer system can be easily modified to satisfy consumers' specific needs. With RISC-V, the development procedure of a computer can be reduced significantly.

1.3 Motivation and Key Contributions

1.3.1 Motivation

The secure boot process is not a new idea, and the typical secure boot standard is the UEFI [17] that was being used widely in commercial Personal Computers (PCs). In UEFI, the processor verifies each stage's integrity by checking its cryptographic signature. If any integrity in the flow fails, the boot process will be abandoned. If the secure boot succeeds, the computer is expected to be run in a trusted state [51]. However, recently, many attack vectors have been discovered that can corrupt the secure boot flow or reveal the root key [52], thus making the whole TEE system vulnerable. The TEE is just an isolated environment; it cannot be the RoT. As a general rule of thumb, a secure boot process with RoT is recommended to be run by hardware primitives. Many TEE models begin with the assumption that their trusted firmware was securely loaded into the stack by a hardware-provided secure boot procedure. In practice, common TEEs use extra hardware or third-party Intellectual Properties (IPs). For instance, Intel SGX relies on the Intel Active Management Technology (AMT) [53], ARM TrustZone uses ARM CryptoCell [54], AMD SEV requires Platform Security Processor (PSP) [55], and for RISC-V, many RoT IPs are presented such as Rambus

CryptoManager [56] and OpenTitan [57]. To summarize, the secure boot with RoT in TEE is still an ongoing problem that needs research and development.

On the current development trend in the cryptography world, open-source is inevitable. The open-source ISA of RISC-V with an open-source TEE is the match in heaven. With RISC-V architecture, custom hardware can be tailored for custom TEE, thus creating many potentials for solving long-lasting problems. As mentioned earlier, the secure boot process with RoT is one of the TEE issues. The TEE's remote attestation has to be done based on the RoT, and the RoT must be established by the secure boot process. However, the secure boot program itself should not be done by the TEE but by a hardware module/primitive or another party inaccessible for the TEE processors after boot. Furthermore, the RoT is also the recommended place for storing and managing the device's root key and certificates. According to the standard of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) [58], such a hardware platform that can provide secure boot with RoT must have the following features:

- Can generate cryptographic keys.
- Can wrap and bind keys.
- Can seal and unseal keys.
- Have a True Random Number Generator (TRNG).
- Have an integrity measurement.
- Can do keys attestation.

Now, with RISC-V, we have the chance to revisit the hardware architecture for a better TEE and bring the RoT down to the lowest possible level, the silicon level, while maintaining a flexible and adaptable secure boot program. As a result, in theory, hacking the silicon RoT means interfering with the chip fabrication process. Moreover, with the developing open-source mindset, the strength of CoT can be enhanced from time to time with state-of-the-art protections. To summarize, the main contribution in this dissertation is about introducing a way to seal the RoT off the TEE processors but still provide the flexibility for updating the boot sequence.

The secondary object of this dissertation is about developing a TEE HardWare (TEE-HW) framework. Because cyber-security is a race between attacker and defender, by having an open-source TEE-HW framework, we can quickly gear up the system for future fights. Moreover, the RISC-V foundation is a solid open-source community that constantly updates and shares new ideas and designs [50]. With RISC-V, we have the freedom to shape the processors and the computer architecture as our will. And that is the most potent weapon for cyber-security developers in the fight against various threats.

1.3.2 Key Contributions

The work of this dissertation can be categorized into three major contributions as follows:

1. **The TEE-HW framework.** The goal is not just to develop a high-security computer system but a framework, an open-source TEE-HW framework to match up with the open-source TEE software framework, named Keystone [7]. The proposed TEE-HW framework must satisfy these criteria: secure, easy to use, flexible for various security purposes, and most importantly, easy to update a new protection method. Many architecture aspects are kept optional and can be quickly reconfigured by changing parameters in the Makefile system. The source codes of this TEE-HW framework [59] is kept open to the RISC-V open-source community for reuse purposes. Such open-source TEE hardware is beneficial for security developers in future improvements.

2. **TEE-HW with cryptographic accelerators.** Based on the proposed TEE-HW framework, a custom system was made exclusively for accelerating the TEE. Multiple crypto-cores were introduced, including Advanced Encryption Standard (AES), Secure Hash Algorithm 3 (SHA-3), Ed25519, and TRNG. The Ed25519 crypto-core also has a secret write-only memory that can be read only by other crypto-cores, not by the TEE processors. This write-only memory will store the keys generated by the Ed25519 module. The performances of the proposed TEE hardware with crypto-cores were explored with the FPGA and VLSI implementation. The TEE boot performance was also investigated.

3. **TEE-HW with isolated RoT.** For the secure boot process with RoT, a heterogeneous architecture was proposed in this dissertation. The idea is to combine 64-bit Linux-capable TEE processors with a 32-bit isolated Micro-Controller Unit (MCU). The TEE side executes the TEE's software stack as usual, while the hidden MCU takes care of the secure boot process, keys generation, and keeping safe the root key. At reset, the isolated MCU boots first, does the first authentication, and generates the subsequent keys elaborated from the root key pre-stored in ROM. After that, the TEE processors are woken up by the MCU to follow the conventional TEE boot flow. The MCU can access all the resources in the system, but the TEE processors cannot access the peripherals in the isolated bus of the MCU. By taking the RoT and the secure boot process out of the TEE domain, the RoT is safe, and the secure boot process is flexible, providing the ability to adapt to future threats. The proposed architecture was developed and tested on both FPGA and VLSI implementations. Two ROHM-180nm chips were made, and the measurements were given.

1.4 Dissertation Layout

The dissertation is divided into six chapters as follows.

Chapter 1: Introduction

In this chapter, the definition and a brief history of TEE are described. Several legacy TEE models and recent publications are discussed together with the trending of development. The ever-increasing need for TEE in the IoT era is mentioned, and some insights into the recent implementations are given. The current problem of RoT in the existing TEE models is explained, together with the potential solutions.

This chapter also introduces the novelty of the open-source RISC-V ISA with current state-of-the-art developments. On the current open-source trending in the cryptography world, hardware RISC-V is the match in heaven for the next step in the evolution of TEE. Therefore, the potential of solving long-lasting problems in TEEs, old and new alike, can be done by leveraging the open-source RISC-V architecture.

Finally, this chapter summarizes this dissertation's motivation and key contributions and presents the dissertation layout.

Chapter 2: Literature Review

In this chapter, all critical or within-five-year publications related to TEE, RISC-V, or RoT are sorted into one single Venn diagram, presented together with their one-line description.

Additionally, deep-dive analysis of the most common TEE models, RISC-V-based or not-RISC-V-based alike, is done for each model. Then, a comparison table is presented comparing those TEEs regarding their security-related features.

Similarly, the same deep-dive analysis is made for each of the most recent security-driven RISC-V-based computer architectures. And then, a comparison table is made based on the investigation.

Chapter 3: Background Research

In this chapter, the goal of the open-source RISC-V is defined, and the key ideas in the ISA are depicted together with its ecosystem. The novelty of using Chisel and FIRRTL to generate a VLSI-friendly Verilog code is also explained. Then, the definition, the typical structure, and the usage of TEE are mentioned together with its advantages and disadvantages. Additionally, the implementation of Keystone, a TEE framework, is described in detail.

Two RISC-V computer systems are also investigated in this chapter. They are the 32-bit MCU named VexRiscv and the 64-bit Linux-capable processor called Rocket-chip. Both architectures are described, and their implementations are shown in FPGA and VLSI. Two silicon-proofs were made for the VexRiscv MCU, one for the ROHM-180nm process and one for the SOTB-65nm process. For the Rocket-chip computer, one ROHM-180nm chip was also made. All three chips' measurement results were reported in this chapter.

Chapter 4: TEE Hardware Computer Systems

In this chapter, the proposed TEE-HW framework is shown. And based on the framework, two developments are made. They are the quick boot for TEE using cryptographic accelerators and the TEE's secure boot procedure using isolated RoT.

The architecture of TEE-HW with crypto-cores is also shown in this chapter. The design and implementation of each crypto-core, including SHA-3, AES, Ed25519 base-point multiplier, and Ed25519 sign algorithm, are described. The boot procedure using the aforementioned crypto-cores is also given.

This chapter also depicts the heterogeneous TEE-HW design with 64-bit TEE processors and an isolated 32-bit MCU. The complete isolation between the hidden MCU and the TEE domain is explained. By storing the root key inside the isolated MCU and having the MCU does the secure boot process, the RoT and the boot-flow are inaccessible to the TEE processors after boot. Furthermore, by introducing the MCU, the boot flow is flexible and adaptable for future changes. The detail of the secure boot procedure is also described.

Chapter 5: Performance Analysis

In this chapter, the FPGA and VLSI implementations of the TEE-HW are given, including TEE-HW with crypto-cores and TEE-HW with isolated RoT. By alternating the boot-flow with cryptographic accelerators, the boot performance is improved, and its result is shown.

Two ROHM-180nm chips were made for the proposed TEE-HW architecture (crypto-cores plus isolated MCU), and their implementations are given. The two chips are 5×5-mm^2 and 5×7.5-mm^2 for the 32-bit and 64-bit TEE-HW versions, respectively. The chips' measurement results are done and shown in the chapter. Finally, the comparison with recent works is discussed.

Chapter 6: Conclusion and Future Work

This chapter summarizes the achievements and limitations of the dissertation. Several ideas are discussed based on the limits to improve the current design. And on top of that, potential developments are given to conclude the dissertation.

This page intentionally left blank.

Chapter 2

Literature Review

2.1 Overview

Figure 2.1 show the summary of the related publications in RISC-V, Trusted Execution Environment (TEE), and Root-of-Trust (RoT). In the figure, the white eclipses are the references, and the yellow squares are the content of this dissertation. The numbers in the white eclipses are the references' numbers, and the numbers in the yellow squares are the subsections' numbers. The references in Figure 2.1 are either major publications or recent publications (less than five years).

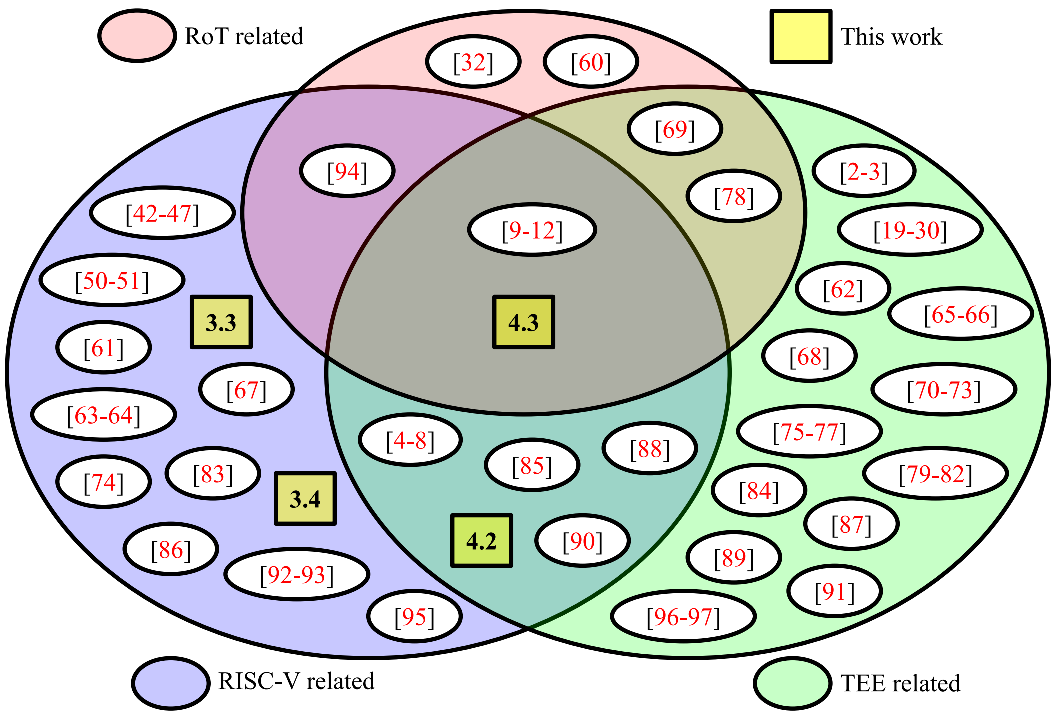


FIGURE 2.1: Literature review summary.

- ARM TrustZone (Apr. 2009) [2]: is TEE implementation explicitly designed for ARM processors.
- A. Furtak *et al.* (2014) [52]: describes many attack vectors that target the Basic Input/Output System (BIOS) and the conventional secure boot process in Personal Computers (PCs).
- Haven (Oct. 2014) [22]: introduces the prototype named Haven that can provide the shielded execution feature for Intel Software Guard eXtensions (SGX) to protect the confidentiality and integrity of a program and its data. It can be used on unmodified legacy applications.
- S. Zhao *et al.* (Nov. 2014) [60]: provides the RoT for ARM TrustZone using on-chip Static Random Access Memory (SRAM).
- V. Costan and S. Devadas (Jan. 2016) [19]: explains the structure, mechanism, and applications of the Intel SGX. It also discusses about advantages and disadvantages of the Intel SGX implementation.
- Rocket-chip generator (Apr. 2016) [40]: guides how to use the Rocket-chip generator to generate the RISC-V rocket core with the desired features.
- C. Duran *et al.* (Apr. 2016) [61]: introduces a 32-bit RISC-V Micro-Controller Unit (MCU) with 10-bit Successive-Approximation-Register (SAR) Analog-to-Digital Converters (ADC).
- F. McKeen *et al.* (Jun. 2016) [62]: presents a method for dynamic memory management inside an Intel SGX's enclave.
- Sanctum (Aug. 2016) [5]: is a RISC-V-based TEE implementation for user-space enclaves with some methods for preventing basic Side-Channel Attacks (SCAs).
- Parallel Ultra-Low-Power (PULP) (Sep. 2016) [63]: is an Ultra-Low-Power (ULP) processor based on RISC-V architecture that promises high energy efficiency and flexible embedded applications.
- Scone (Nov. 2016) [24]: describes a secure Linux container mechanism, named Scone, for Intel SGX's docker to protect container processes from outside attacks.
- RI5CY (Feb. 2017) [43]: presents a ULP RISC-V core that can operate at near-threshold voltage. It also has Digital Signal Processing (DSP) extensions for Internet-of-Things (IoT) devices.
- C. Duran *et al.* (Feb. 2017) [64]: presents the chip measurement results for the 32-bit RISC-V MCU in [61].
- Intel SGX book I & II (Jul. 2017) [20,21]: describes the Intel SGX architecture in detail. The book also compares the similar approach of Sanctum, a RISC-V-based TEE.

- Yogesh Swami *et al.* (Jul. 2017) [65]: reports the findings of enclaves' security in Intel SGX architecture, including positive and negative results.
- Graphene (Jul. 2017) [23]: presents a port of Graphene to SGX with some improvements to make the security benefits of SGX more usable.
- Sancus (Aug. 2017) [66]: describes the design of Sancus with Field-Programmable Gate Array (FPGA) evaluation. The prototype extends an MSP430 processor with hardware support for memory access control and cryptographic functionalities.
- Komodo (Oct. 2017) [26]: proposes an alternative approach to attested, on-demand, user-mode, concurrent isolated execution. The implementation is a prototype in verified assembly code on ARM TrustZone.
- PULPv2 (Oct. 2017) [67]: is the next evolution of PULP in [63], targeting a wide range of emerging near-sensor processing tasks for IoT applications.
- D. K. Dennis *et al.* (Dec. 2017) [38]: shows a development of a fully synthesizable 32-bit processor based on RISC-V, targeting low-cost embedded devices.
- Identity-Based Broadcasting Encryption (IBBE) SGX (Jun. 2018) [68]: constructs an encryption scheme for group access control over the cloud secured by Intel SGX.
- M. Staffa *et al.* (Aug. 2018) [69]: presents a study of the security treats over humanoid robots and the first hardware-based solution using TEEs.
- Y. Fan *et al.* (Aug. 2018) [70]: implements TEEs on mobile devices.
- K. Patsidis *et al.* (Sep. 2018) [39]: proposes a method that allows 32-bit RISC-V processors to execute compressed 16-bit instructions.
- Z. Ning *et al.* (Oct. 2018) [71]: investigates the typical hardware-assisted TEEs and evaluates the performance of these TEEs to help analyze the feasibility of deploying them on the IoT edge platforms.
- E. Gür *et al.* (Nov. 2018) [37]: presents a 32-bit open-source RISC-V processor with web-based assembler/disassembler tools. The tools allow users to generate and download their machine codes over the web.
- A. Munir *et al.* (Dec. 2018) [36]: proposes a reusable framework for the end-to-end verification of RISC-V cores against the Instruction Set Architecture (ISA) specs using the Universal Verification Methodology (UVM).
- Sanctuary (Feb. 2019) [28]: presents an architecture, named Sanctuary, that allows unconstrained TEEs in the ARM TrustZone ecosystem, enabling the execution of security-sensitive applications within strongly isolated compartments.

- TIMBER-V (Feb. 2019) [6]: is a RISC-V-based TEE implementation with a "Tag" procedure across all layers to isolate one enclave.
- M. A. Mukhtar *et al.* (Mar. 2019) [72]: analyses popular TEEs, including Intel SGX and ARM TrustZone, for better protection insights regarding functionality, implementation, and security.
- L. Guan *et al.* (May 2019) [73]: shields applications over compromised operative systems and physical attacks using a lightweight run-time system based on ARM TrustZone.
- Y. Wang and N. Tan (Jun. 2019) [35]: proposes novel instructions for energy accumulation of energy metering. The prototype was realized in a 32-bit RISC-V microprocessor.
- R. Höller *et al.* (Jun. 2019) [74]: evaluates open-source 32-bit Central Processing Unit (CPU) Intellectual Property (IP) cores suitable for FPGAs and supports the upcoming RISC-V ISA extensions.
- M. Bailleu *et al.* (Jun. 2019) [75]: offers performance measurement tools for TEEs, injecting profiling code, and tracing the program execution.
- A. Gollamudi *et al.* (Jun. 2019) [76]: presents a core calculus for secure decentralized distributed applications using standard cryptographic mechanisms and the TEE platforms.
- Ariane (Jul. 2019) [44]: proposes a 64-bit RISC-V ISA variant called Ariane with a detailed analysis of power, performance, and applications efficiency.
- SPEED (Jul. 2019) [77]: proposes a secure and generic deduplication system in Intel SGX, improving performance by reusing computation results by identifying redundant computation.
- E. M. Benhani *et al.* (Aug. 2019) [78]: highlights the security issue of complex System-on-Chips (SoCs) and presents six efficient attacks on the ARM TrustZone.
- DER-TEE (Aug. 2019) [79]: presents a smart inverter using a TEE-based architecture.
- HyperMI (Aug. 2019) [80]: presents Virtual Machine (VM) protection, featuring security against compromised hypervisors by isolating guests in a secure execution environment.
- MicroTEE (Aug. 2019) [81]: designs a TEE on a microkernel software architecture with the necessary services for the application layer.
- R. Ladjel *et al.* (Aug. 2019) [82]: evaluates the use of TEE-based computing for personal data in a large number of participants.
- OP-TEE [27]: was proposed in Jun. 2014, now been transferred to TrustedFirmware.org since Sep. 2019. It is an open-source TEE implementation designed to companion a non-secure Linux kernel running

on ARM processors. OP-TEE implements TEE Internal Core Application Programming Interface (API) defined in the GlobalPlatform API specifications.

- ITUS (Mar. & Sep. 2019) [11,12]: attempts to solve the RoT for TEE by a hardware approach. It uses mainly two hardware modules of the Key Management Unit (KMU) and Code Authentication Unit (CAU) to do the keys generation, keys provisioning, and enclaves authentication.
- MI6 (Oct. 2019) [83]: proposes a method to secure RISC-V-based Out-of-Order processors by modifying the queue structure of the Last Level Cache (LLC).
- S. Pinto and N. Santos (Nov. 2019) [25]: presents an in-depth study of ARM TrustZone technology and a comprehensive survey of relevant works.
- AMD Secure Encrypted Virtualization (SEV) (Nov. 2019) [3]: is a TEE implementation designed for AMD secure processors aiming for the cloud computing market.
- AMD SEV Encrypted State (SEV-ES) (Feb. 2017) [29]: adds the encryption for core register states, thus blinding the hypervisor from seeing the data used by VMs.
- AMD SEV Secure Nested Paging (SEV-SNP) (Jan. 2020) [30]: introduces some new hardware-based security enhancements for protecting memory integrity.
- J. V. Bulck *et al.* (Nov. 2019) [84]: analyzes the vulnerability space arising in TEEs when interfacing a trusted enclave with untrusted applications and OS, exposing several sanitizations at the level of Application Binary Interface (ABI) and API.
- T.-T. Hoang *et al.* (Apr. 2020) [85]: presents a RISC-V system compatible with TEEs featuring security algorithm accelerators, including Secure Hash Algorithm 3 (SHA-3) hash and Ed25519 elliptic curve algorithms.
- Keystone (Apr. 2020) [7]: presents Keystone, the first open-source framework for building customized TEEs.
- BOOMv3 (SonicBOOM) (May 2020) [86]: presents SonicBOOM, the third generation of the open-source Berkeley Out-of-Order Machine (BOOM) based on RISC-V architecture.
- D. Cerdeira *et al.* (May 2020) [87]: presents a security analysis of popular TrustZone-assisted TEE systems (targeting Cortex-A processors) developed by Qualcomm, Trustonic, Huawei, Nvidia, and Linaro. The goal is to the main challenges to build them correctly.
- Elasticlave (Jul. 2020) [88]: presents Elasticlave, a new TEE memory model which allows sharing, aiming to balance security and flexibility in managing access permissions.

- RIPTE (Sep. 2020) [89]: proposes a novel and practical scheme, named RIPTE, for trusted software execution based on lightweight trust, using the combination of dynamic measurement and control flow integrity with Physical Unclonable Function (PUF).
- S. Moritz *et al.* (Oct. 2020) [90]: proposes new security properties relevant for platforms that enclaves can utilize a dynamic hardware Trusted Computing Base (TCB).
- J. Jang and B. B. Kang (Oct. 2020) [91]: constructs a TEE defense framework using a partially privileged process that can communicate with the hypervisor and TrustZone without depending on the kernel.
- VexRiscv [92]: is an open-source 32-bit RISC-V core developed on SpinalHDL language.
- T.-T. Hoang *et al.* (Oct. 2020) [93]: presents chip measurement results of the 65-nm Silicon-On-Thin-BOX (SOTB) MCU based on the 32-bit VexRiscv core in [92].
- Hex-Five MultiZone [4]: is a RISC-V-based TEE development that is small, portable, multi-purpose, and requires only Physical Memory Protection (PMP) to work. It promises a lightweight and robust security solution with multiple domains.
- SiFive WorldGuard [10]: is a RISC-V-based computer system with security primitives developed for strengthening the isolation in TEEs.
- M. Boubakri *et al.* (Feb. 2021) [94]: explores the feasibility of implementing a software-only Trusted Platform Module (TPM) on RISC-V hardware.
- HECTOR-V (Jun. 2021) [9]: is a RISC-V-based SoC system aiming for TEE purposes. Its goal is to separate the TEE processor from the Rich Execution Environment (REE) domain.
- K.-D. Nguyen *et al.* (Aug. 2021) [95]: presents a 32-bit RISC-V MCU with a COordinate Rotation DIgital Computer (CORDIC) accelerator.
- CUstomizable and Resilient Enclaves (CURE) (Aug. 2021) [8]: is a RISC-V-based TEE architecture. Its goal is to provide multiple types of enclaves to fit in various situations, i.e., kernel-space enclaves, user-space enclaves, and self-contained sub-space enclaves.
- K. Suzaki *et al.* (Sep. 2021) [96]: proposes a compiler-based performance measurement for TEE and REE on Intel SGX, ARM TrustZone, and RISC-V Keystone.
- SGX-FPGA (Dec. 2021) [97]: presents SGX-FPGA, a trusted hardware isolation path enabling the first FPGA-based TEE by bridging SGX enclaves and FPGAs in the heterogeneous CPU-FPGA architecture.

2.2 TEE Implementations

2.2.1 Intel SGX

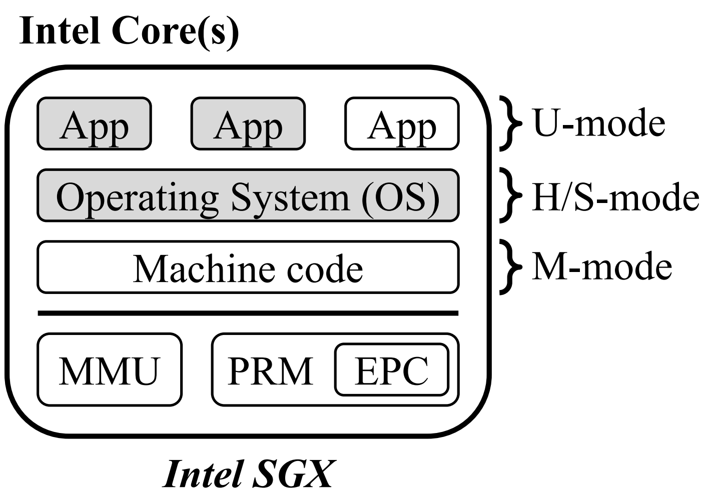


FIGURE 2.2: Intel SGX TEE implementation [1] (clear-box: trusted; gray-box: untrusted).

Intel SGX is a set of extensions used to provide TEE for Intel architecture [1], and its implementation is given in Figure 2.2. The idea is to create an isolated virtual address space called enclave to execute a user's code [19]. The program and data of each enclave are isolated from other enclaves and normal programs, including Operating System (OS) and hypervisor applications, thus protecting that enclave's confidentiality and integrity. On top of that, the conventional software stack in the Intel architecture remains the same; hence the standard OS kernel and hypervisor are compatible with the SGX [20].

The life cycle of an enclave begins with the creation of an Enclave Page Cache (EPC) inside the Processor Reserved Memory (PRM), as shown in Figure 2.2. The SGX protects the PRM against non-enclave, Direct Memory Access (DMA), and peripherals accesses [19]. An EPC has 4-KB and contains only one enclave's data and program. During the initiation process, the enclave's initial code is loaded from outside the PRM into its corresponding EPC inside the PRM, and the processor hashes its content. That hash value will become the enclave's measurement hash and will be used for attestation later. The enclave is always executed in the protected mode with address translation set up by the OS kernel and hypervisor [19]. A user can do the remote attestation to make sure that he/she is communicating with a specific enclave running in trusted hardware. In addition, the remote computation service will refuse to load the data if the hash does not match the expected value [20].

The drawbacks of Intel SGX are because the enclaves have to run on top of an untrusted OS and host applications, relying on the potentially malicious system software stack for its services and resources [21]. For example, the 4-KB EPC is assigned to an enclave by the untrusted system software, and the enclave's initial code is loaded to its EPC from an untrusted memory region and by an untrusted system service.

2.2.2 ARM TrustZone

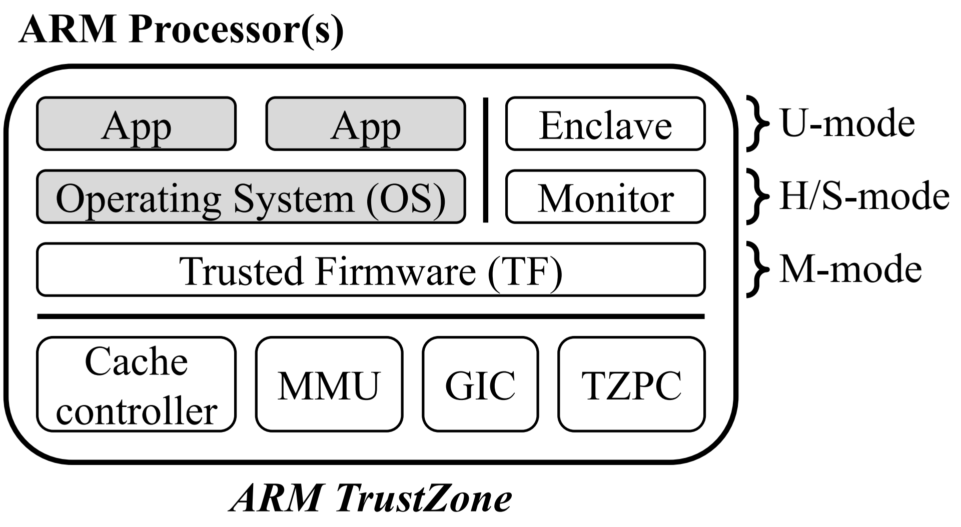


FIGURE 2.3: ARM TrustZone TEE implementation [2] (clear-box: trusted; gray-box: untrusted).

TrustZone is a TEE explicitly designed for ARM processors [2, 25], and its architecture is described in Figure 2.3. ARM TrustZone has many versions with slight differences depending on which version of the ARM core is being used (i.e., ARMv7-A, ARMv7-M, ARMv8-A). But the core idea is the same with the two "worlds" implemented, as shown in Figure 2.3. As seen in the figure, sensitive applications called "enclaves" are run in the secure world and based on the monitor, isolated from the normal world based on an OS. Both worlds are built upon the Trusted Firmware (TF) at Machine-mode (M-mode). The underlying mechanism is the isolation enforced by the Not-Secure (NS) bit in all operations, including memory bus and peripheral accesses [87]. TF is aware of the NS bit and can allow or disallow access across the two worlds accordingly. The hardware features needed for ARM TrustZone are many and can be varied depending on the processor's version. But the note-worthy mentions are the TrustZone Address Space Controller (TZASC) for partitioning Advanced eXtensible Interface (AXI) bus's memory-mapped devices, the TrustZone Memory Adapter (TZMA) for dividing on-chip memories, the Generic Interrupt Controller (GIC) to handle prioritized interrupts, and the TrustZone Protection Controller (TZPC) to protect control signals.

The limitation of ARM TrustZone is that it has only two hardware-enforced partitions. Hence, multiple TEEs system is impossible to be implemented. Furthermore, because the secure side cannot use the normal world's services directly, it must provide and manage its resources. And on top of that, executing multiple enclaves requires multiplexing via a dedicated secure-world OS.

2.2.3 AMD SEV

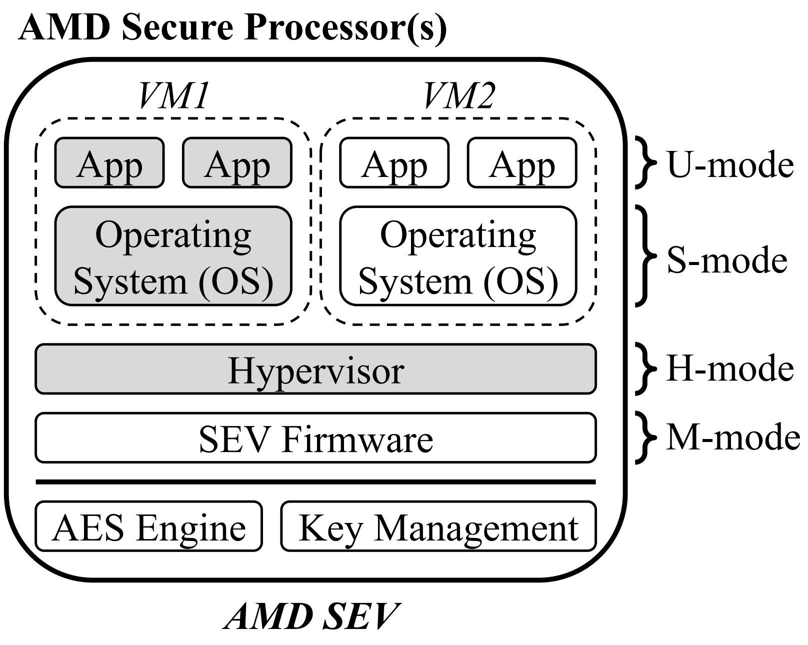


FIGURE 2.4: AMD SEV TEE implementation [3] (clear-box: trusted; gray-box: untrusted).

AMD SEV [3] is designed for AMD secure processors, and its architecture is shown in Figure 2.4. The target market of the SEV is cloud computing, where each user often has his/her VMs running on the hardware infrastructure provided by a cloud administrator via a share hypervisor layer, as depicted in Figure 2.4. As a result, AMD proposed their solution to isolating an entire VM from its untrusted hypervisor and other VMs that may coexist on one physical server [3]. The underlying mechanism of SEV uses Advanced Encryption Standard (AES) and key management modules to assign a unique key to each VM and then encrypts that VM's main memory to prevent higher-privileged accesses. This approach protects an entire VM rather than a specific user-level code. In addition, SEV also provides a remote attestation method for users to verify their VMs' integrity.

Because SEV is developed based on an untrusted hypervisor layer, it is subjected to software and physical adversaries. To overcome this issue, AMD introduced the SEV-ES [29] in 2017, and then the SEV-SNP [30] in 2020 as the next evolution of the original design. The SEV-ES added the encryption for core register states, thus blinding the hypervisor from seeing the data that is being used by VMs [29]. Moreover, SEV-ES extends the support for memory encryption, thus protecting the memory confidentiality [98]. If the memory confidentiality is protected in SEV-ES, then the SEV-SNP [30] aims for memory integrity by introducing some new hardware-based security enhancements. SEV-SNP states that it could prevent malicious hypervisor-based attacks such as data replay and memory re-mapping [30]. Furthermore, SEV-SNP also addressed the recent SCAs by offering some protections related to interrupting behavior.

2.2.4 Hex-Five MultiZone

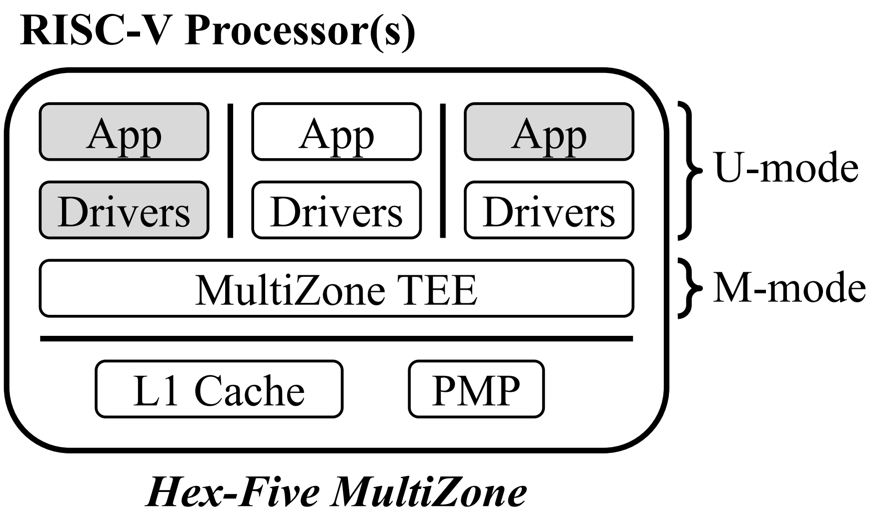


FIGURE 2.5: Hex-Five MultiZone TEE implementation [4] (clear-box: trusted; gray-box: untrusted).

MultiZone is a RISC-V-based TEE developed by the Hex-Five, Inc. [4], and its architecture is given in Figure 2.5. The source codes of MultiZone are available on the Github repository [99]. The requirements to implement MultiZone are the PMP feature in hardware and 32/64-bit RISC-V processors with the User-mode (U-mode) privilege. MultiZone promises a lightweight and strong security solution with multiple domains. The philosophy in design is to separate the software stack into zones by the firmware at M-mode, as shown in the figure. Each domain has complete control over its data and resources but cannot overstep on the other domains. Hex-File MultiZone also inherits the open-source mindset with its source codes are being open on Github, and its license is free for evaluation and commercial use [4]. Furthermore, it supports real-time monitoring, secure boot, and remote firmware updates [4].

The "zone" in Figure 2.5 could be U-mode-only applications or Supervisor-mode (S-mode) to U-mode applications with a Unix-like OS. In theory, one zone or a part of one zone could be hacked, but that couldn't affect the neighboring zones because the barriers are guaranteed by the trusted firmware below. The underlying mechanism is that each application block will be written, compiled, and linked separately. Then, the MultiZone will set the desired Random Access Memory (RAM), Read-Only Memory (ROM), Input/Output (I/O), and interrupt isolation for each zone [4]. Finally, the MultiZone configurator is invoked to combine the zones' elf/hex files with the runtime into one single signed firmware image [4].

Although the MultiZone provided strong security over many zones, for practical implementations, it falls under the category of embedded application or MCUs rather than rich OS applications own to the fact that the sharing resources between them zones are restricted.

2.2.5 Sanctum

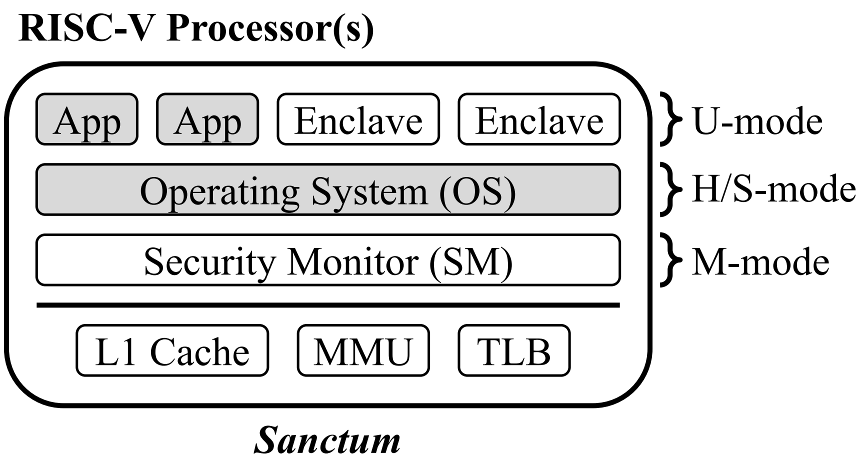


FIGURE 2.6: Sanctum TEE implementation [5] (clear-box: trusted; gray-box: untrusted).

Sanctum is a RISC-V based TEE developed by V. Costan *et al.* [5], and its implementation is given in Figure 2.6. The source codes of Sanctum are available on the Github repository [100]. As shown in the figure, enclaves are isolated at U-mode with the assumption of a compromised OS layer underneath. This software stack of Sanctum resembles that of the Intel SGX, but with RISC-V processors instead of Intel cores [21]. In Sanctum, the integrity of an enclave is ensured by a prior verification process using local or remote attestation [5]. The isolation in Sanctum is enforced by the customized Page Table Walker (PTW) in the Memory Management Unit (MMU). The changes in PTW can deny access to one enclave's memory from the OS or other enclaves. Furthermore, it could prevent the addresses translation from virtual space to physical space if the current execution context does not have the right to access. As seen in Figure 2.6, a Security Monitor (SM) is deployed at M-mode to increase the security level, especially for the purpose of preventing SCAs. The SM is verified from the beginning by a secure boot process.

To combat with SCAs, Sanctum utilized two mechanisms of L2 partitioning and L1 flushing (including the Translation Lookaside Buffer, the TLB). However, because its cache partitioning method is based on the memory page coloring scheme, it is quite impractical since we have to rearrange the complete OS memory layout according to their colors at run time. Furthermore, due to the struggle against SCAs, Sanctum cannot provide a direct secure peripheral connection to its enclaves because it would need a privileged driver code to run under an untrusted environment, thus introducing a potential security threat. This limits the Sanctum's range of applications. To summarize, similar to Intel SGX, Sanctum has to rely on the untrusted OS for managing resources (like memory) and providing services (e.g., interrupt and I/Os); hence some adversaries and attack vectors could be realized in the future with the compromised OS.

2.2.6 TIMBER-V

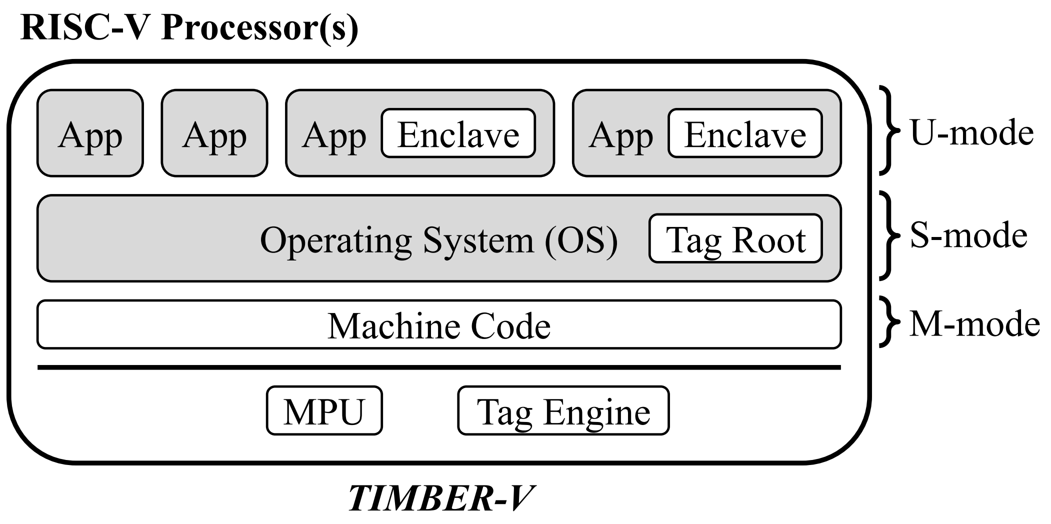


FIGURE 2.7: TIMBER-V TEE implementation [6] (clear-box: trusted; gray-box: untrusted).

TIMBER-V is a RISC-V-based TEE design by S. Weiser *et al.* [6], and its software stack is given in Figure 2.7. The source codes of TIMBER-V are available on the Github repository [101]. The idea of TIMBER-V is to utilize a "Tag" procedure across all layers, as shown in the figure, for the isolation of one enclave's operation from other enclaves and the usual OS domain. The underlying mechanism is the memory tagging process done by the Memory Protection Unit (MPU) and the tag engine in hardware [6]. The memory tagging is used for assigning memory regions, and the tag engine enforces the isolation by checking every memory access rights by its tags during the current execution program. Additionally, each DMA-capable peripheral must include a tag engine inside to defend against DMA-based peripheral attacks. At the S-mode layer, a trusted TagRoot is introduced, as seen in Figure 2.7, for setting up the U-mode enclaves and providing several services such as cross-domain communication, sealing, and attestation. TagRoot is also responsible for assigning tags and sub-process enclaves separation.

Compared to similar approaches of Sanctum and Intel SGX, the TIMBER-V allows more fine-grained settings for its enclaves and sub-process enclaves. However, it still suffers from some SCAs due to the untrusted OS. For example, TIMBER-V does not have a protection mechanism against cache-based and interrupt-based SCAs because the untrusted OS does the interrupts handling, and the cache states are still shared in the system. In addition, TIMBER-V can not provide a secure Enclave-to-peripheral communication because introducing peripheral drivers into the TagRoot would increase the overall attack surface. Finally, TIMBER-V needs custom instructions with 2-bit tags for every 32-bit memory, which would increase the area overhead and decrease the performance.

2.2.7 Keystone

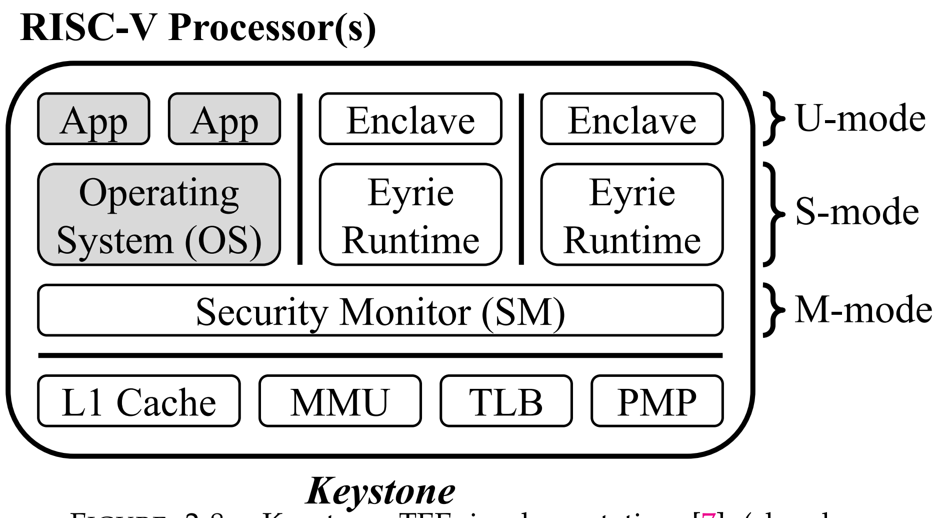


FIGURE 2.8: Keystone TEE implementation [7] (clear-box: trusted; gray-box: untrusted).

Keystone is a RISC-V-based TEE developed by D. Lee *et al.* [7], and its architecture is described in Figure 2.8. The source codes of Keystone are available on the Github repository [102]. The goal of Keystone is not to present one TEE implementation but to propose a TEE framework that is open-sourced, portable, and flexible with a modular design approach [7]. According to Figure 2.8, Keystone's enclaves comprise not only the user code but also the runtime at S-mode as well. Multiple enclaves with the untrusted OS share the same SM at the M-mode layer. In the figure, the Eyrie runtime at S-mode provides the OS-equivalent services such as memory management and interrupts handling, and the SM at M-mode enforces the isolation by using the PMP feature of RISC-V [13]. SM uses PMP to assign memory regions and define the memory access rights for each active enclave and the OS. Similar to Sanctum and MultiZone, the M-mode SM is verified by a prior authentication during the secure boot procedure. Because Keystone's enclaves have a complete software stack from U-mode to M-mode, they are not scheduled like OS programs and do not rely on the OS for critical functions. As a result, Keystone can defense against strong software adversary and controlled SCA that exploit the sharing states across domains, like interrupt handlers and table paging.

For preventing SCAs, Keystone supports a list of features that can be included as plugins. Currently, the supported plugins are memory isolation, memory encryption, enclave encryption, enclave self-paging, enclave's memory dynamic resizing, edge call and syscall services, and cache partitioning [7]. The L2 cache partitioning technique used in Keystone is a way-based method, which can be an under-utilization implementation in practice. Therefore, the communication between an enclave and an untrusted application might harm the system's performance. Finally, although Keystone allows peripheral drivers to be included in the Eyrie runtime, thus making an Enclave-to-peripheral connection, it is not a two-way binding process, allowing DMA-capable peripherals attacks.

2.2.8 CURE

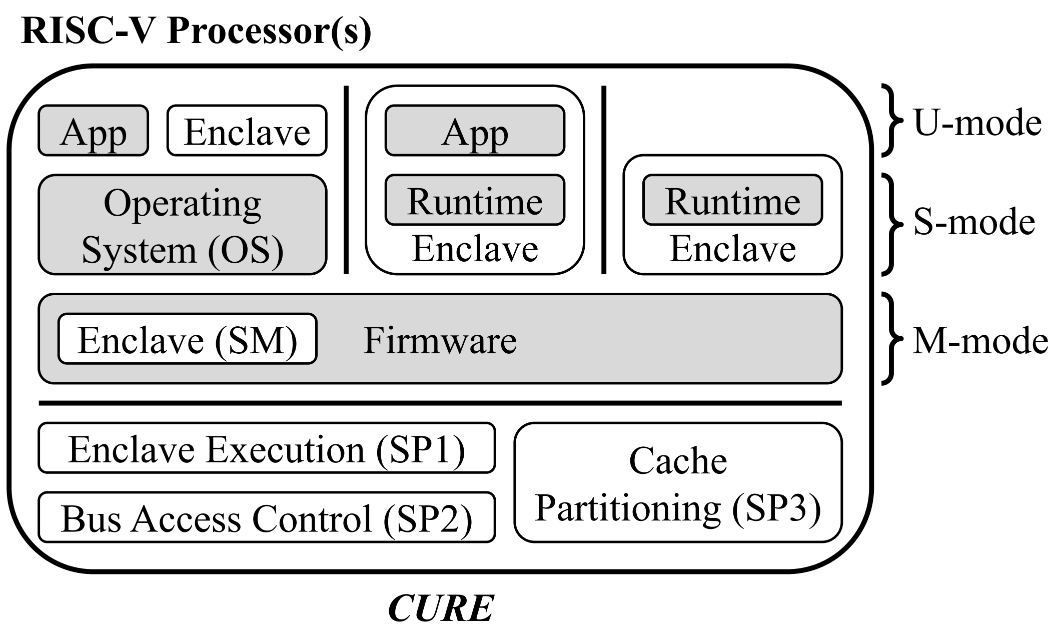


FIGURE 2.9: CURE TEE implementation [8] (clear-box: trusted; gray-box: untrusted).

CURE is a RISC-V-based TEE architecture introduced by R. Bahmani *et al.* [8], and its implementation is shown in Figure 2.9. CURE presented a strong isolation solution for its enclaves across multiple layers and use cases. Its goal is to provide multiple types of enclaves to fit in various situations [8], i.e., kernel-space enclaves (S-mode only), user-space enclaves (U-mode only), and self-contained sub-space enclaves (S-to-U-mode), as seen in the figure. Although CURE shows an interesting structure and implementation of TEE, its source codes are not available in public.

The underlying mechanism of the CURE is based on the new hardware security primitives developed at three different privileged levels of computer architecture. As depicted in Figure 2.9, they are enclave execution (SP1) added in core's register files, bus access control (SP2) added in the system bus's arbitrator, and cache partitioning (SP3) added in the shared cache of L2. Newly developed hardware primitives can be configured to fit different security requirements, and they were proven to be small sizes with a reasonable performance overhead [8]. With three proposed SPs, CURE can assign system resources from caches to peripherals exclusively for a single enclave.

The SM manages all enclaves in CURE at M-mode, as shown in Figure 2.9. After being called by the host application, the life-cycle of an enclave begins with the OS loading its binary and configuration file, then does a context switch to the SM. The SM then verifies the enclave code by checking its signature and certificate. If the enclave's signature is valid, the SM continues to configure hardware primitives according to the configuration file, including exclusively physical memory regions, caches, and peripherals. At this step, the enclave's authentication and encryption keys are also created by the SM for attestation purposes. SM also handles all interruptions in and out of the enclaves at run time, thus providing supervisor-level services. In contrast, enclaves can still use services provided by the untrusted OS as long as they do not require direct sensitive data access. Furthermore, the enclave can use its encryption key, generated by the SM at the enclave setup time, to protect its sensitive data and perform communication over the untrusted OS layer.

To combat with SCAs, CURE disables the hyper-threading during an enclave execution to prevent leaks via shared resources between threads. All caches, including L1, TLB, and Branch Target Buffer (BTB), are flushed by the SM to prevent leakage across execution programs at every setup and teardown of an enclave or an enclave's context switch. Furthermore, the shared cache of L2 is protected due to the way-based cache partitioning method that allows cache lines assignment exclusively. For the limitations, because user-space enclaves don't have a trusted runtime, they can not securely execute device drivers to bind with peripherals. For the kernel-space enclaves, the trusted runtime needs to be added, which increases the binary size and memory consumption. Furthermore, to set up a runtime enclave, the selected cores need to be freed first, then detached from the OS, and booted to the desired runtime; this would introduce performance overhead for the system. Finally, although CURE guarantees the secure enclave-to-peripheral connection that can defend DMA-based peripheral attacks, it assumes that the underlying hardware is trusted and bug-free. Therefore, the SCAs that exploit hardware flaws such as fault injection and physical attacks are not considered.

2.2.9 Comparison

To summarize, TEEs nowadays have various designs and purposes. For example, some TEEs are based on open-source RISC-V (MultiZone [4], Sanctum [5], TIMBER-V [6], Keystone [7], CURE [8]), and some TEEs are for conventional closed-source processors (Intel SGX [1], ARM TrustZone [2], AMD SEV [3]). The purposes of TEE are also different, from aiming for embedded market and IoT (ARM TrustZone [2], Hex-Five MultiZone [4]) to providing a cloud computing service (AMD SEV [3]), from wanting to preserve the traditional OS software stack (Intel SGX [1], Sanctum [5]) to multi-type enclaves for multi-purposes (CURE [8]). Different goals make different developing mindsets. Therefore, a fair and complete comparison would be impossible. The comparison in Table 2.1 considers security-related and flexible features of the commonly used TEEs. The comparison keys are:

- **Open-source.** The source codes are available for modification.
- **Enclave type.** Type(s) of enclaves that the TEE supported.
- **Software adversary.** Can protect against a software attacker that can control host applications, OS services, network communications, unprotected memory regions, and replay messages.

- **Physical adversary.** Can protect against a physical attacker that can intercept, modify, and replay messages in and out of the chip.
- **SCA resilience.** The ability to prevent a particular type of SCA, including cache-based attacks, controlled-channel attacks, and DMA-based peripheral attacks.
- **Secure enclave-to-peripheral.** Could bind a peripheral to an enclave to make a direct secure connection.
- **Small trusted firmware.** Require a small firmware to make a trusted base; millions Line-Of-Codes (LOCs) are large, thousands LOCs are medium, and less than a thousand LOC is small.
- **Hardware modification.** The TEE implementation needs hardware modifications to operate. The best is no modifications, the medium is several modules included, and the worst is intensive modifications, including micro-architecture.
- **Resource management.** The enclaves can manage their assigned resources.
- **Wide-range applications.** The proposed TEE can be applied for many purposes.
- **High expressiveness.** The TEE allows forking, syscalls, multi/hyper-threading, shared memory, and so on.
- **Low porting effort.** The development effort for adapting/porting the proposed TEE structure to another design/hardware.

In Table 2.1, the speculative execution attacks and timing-based SCAs are deemed out of scope. The Denial-of-Service (DoS) attacks are not considered because the compromised OS can always starve enclaves or shut down the system entirely. The attacks that exploit hardware flaws such as memory-bit flipping, dynamic clock/power management, and physical SCAs like power analysis are also out of scope.

TABLE 2.1: TEE implementations comparison regarding the security-related features; ●, ◐, and ○ rank the performance from best/supported to worst/not-supported, respectively.

		Intel				ARM				AMD			RISC-V				
		SGX [1]	Haven [22]	Graphene [23]	Scone [24]	TrustZone [2]	Komodo [26]	OP-TEE [27]	Sanctuary [28]	SEV [3]	SEV-ES [29]	SEV-SNP [30]	MultiZone [4]	Sanctum [5]	TIMBER-V [6]	Keystone [7]	CURE [8]
Open-source		○	○	●	○	○	●	●	○	○	○	○	◐	●	●	●	○
Enclave type	User-space	●	●	●	●	○	○	○	○	○	○	○	○	●	●	○	●
	Kernel-space	○	○	○	○	●	●	●	●	●	●	●	●	○	○	●	●
Adversary	Software	●	●	●	●	●	●	●	●	◐	●	●	●	●	●	●	●
	Physical	●	●	●	●	○	●	●	○	◐	●	●	●	○	●	●	●
SCA resilience	Cache-based	○	○	○	○	○	◐	◐	◐	○	○	◐	●	●	○	●	●
	Ctrl-channel	○	○	○	○	○	●	○	○	○	○	○	●	●	○	●	◐
	DMA-based	○	○	○	○	●	●	●	●	○	○	○	●	○	●	○	●
Secure enclave-to-peripheral		○	○	○	○	◐	◐	◐	◐	○	○	○	●	○	○	○	●
Small trusted firmware		●	○	○	◐	○	◐	○	○	●	●	●	◐	◐	●	◐	◐
Hardware modification		○	●	●	●	○	●	●	●	○	○	○	●	○	○	●	○
Resource management		○	◐	◐	○	●	◐	●	◐	●	●	●	○	◐	●	●	●
Wide-range applications		○	◐	◐	◐	●	●	●	●	●	●	●	○	◐	◐	●	●
High expressiveness		○	●	●	●	●	◐	●	●	●	●	●	○	◐	●	◐	◐
Low porting efforts		○	●	●	◐	○	●	●	●	●	●	●	◐	●	◐	●	○

2.3 Security-driven RISC-V Computer Systems

2.3.1 CURE

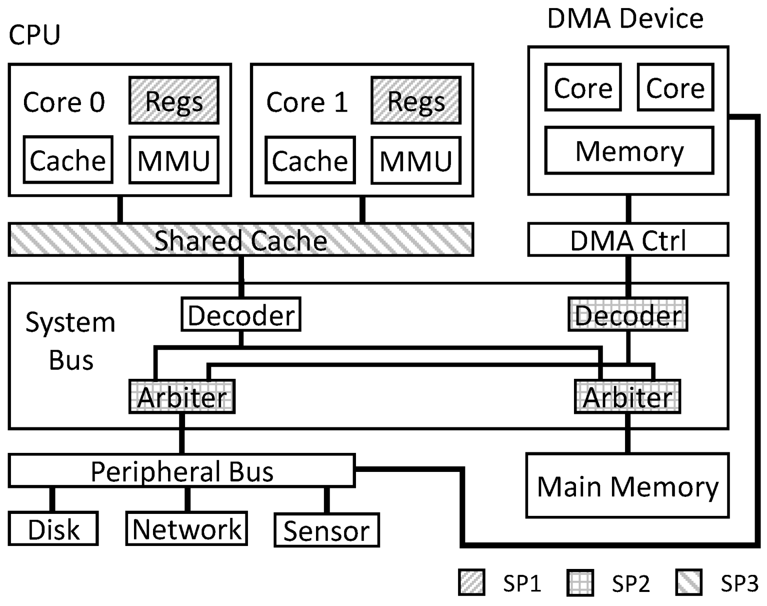


FIGURE 2.10: CURE hardware security primitives at core register files (SP1), system bus (SP2), and shared cache (SP3) [8].

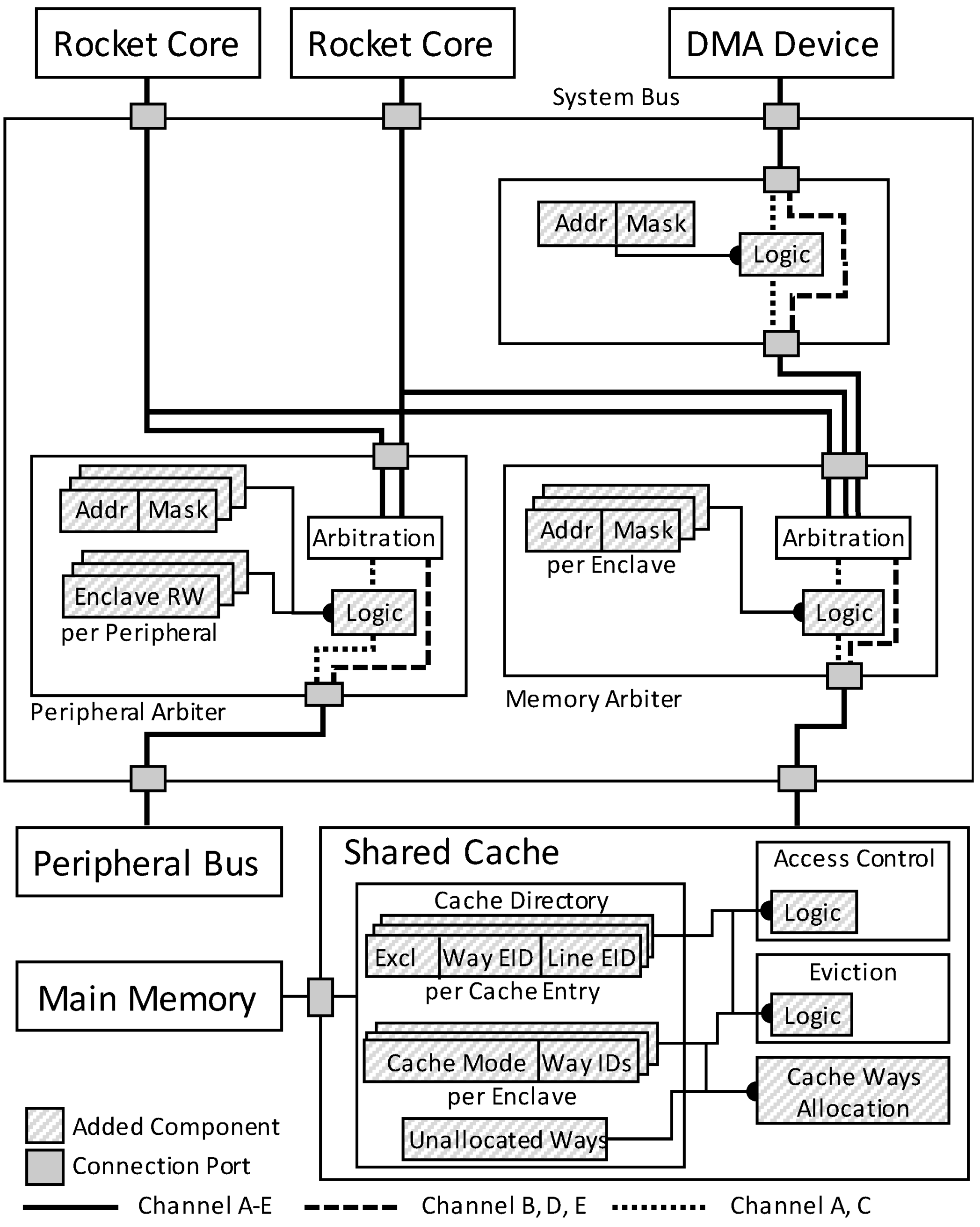


FIGURE 2.11: Detail CURE hardware implementation based on Rocket cores [8].

CURE [8] proposes a strong enclaves isolation for TEE based on new hardware security primitives using the RISC-V architecture. In CURE, we can have many types of enclaves coexisted in the same system, i.e., kernel-space enclaves, user-space enclaves, and sub-space enclaves. To achieve that, three main hardware modifications, named SP1, SP2, and SP3, are added. As shown in Figure 2.10, the SP1 is registers added in the core for enclave execution, the SP2 is in the system bus's arbitrator for controlling bus' accesses, and the SP3 is related to the partitioning in the shared cache. With three proposed SPs, CURE hardware can strengthen the isolation of TEE and can assist in the fight with SCAs. A prototype was realized based on Rocket cores, and Figure 2.11 describes its implementation.

In Figure 2.10, SP1 is used to store enclave IDs, thus indicating which enclave is currently executed by which core. The ID values are set during enclave setup/teardown and enclave's context switch. Inside the Rocket core, the enclave IDs are also considered during the address translation process in the PTW. Therefore, even if a compromised enclave tries to modify its page table, the PTW transactions will be blocked by the system bus' access control due to ID mismatch. The conventional TileLink bus is extended by a 4-bit ID signal to carry the IDs throughout the system. Only the A and C channels are modified because they are the transactions from CPU/DMA to the memory; the other channels remain the same.

In Figure 2.11, SP2 is the added components in the peripheral and memory arbiters. They are used for bus access control. Whenever a memory access request happens, the arbiters will check the access rights based on the enclave's ID signal that came together with the transaction. If there is a violation, the transaction will be redirected to an unused, forbidding the transaction to continue. In CURE, a DMA device can be assigned exclusively to a single enclave. Therefore, to protect an enclave from DMA-based peripheral attacks, a register is added in front of every DMA port, and a DMA device can access the main memory but not other peripherals. The added register in DMA ports defines which memory regions the DMA can access. The register will be updated accordingly whenever a DMA device is assigned to an enclave. For the enclave-to-peripheral binding, since no party can access the memory regions without permission, no encryption or authentication was done on the communication between the enclave and the peripheral.

For mitigating SCAs, two main methods were used in CURE. They are the L1 flushing at every enclave's context switch and L2 cache partitioning. The L2 cache partitioning has two options of strict partitioning across all enclaves or randomization-based scheme; both options are way-based partitioning. Each cache is allocated exclusively to an enclave, and it will be flushed at the enclave's context switch. In Figure 2.11, the added components of SP3 in the shared cache enforce cache access rights based on the enclave's ID signal.

About the RoT, the CURE doesn't do the RoT but assumes that the secure boot process was done on reset. And the first bootloader stored in ROM would verify the firmware (including the SM) and load the firmware to the designated RAM.

2.3.2 HECTOR-V

HECTOR-V [9] is a RISC-V-based SoC tailoring for the isolation effect of the existing TEEs, and its architecture is given in Figure 2.12. Unlike CURE, HECTOR-V is not proposing a new TEE model; it enhances the security strength for the conventional TEEs at the architectural level. The HECTOR-V design approach comes from two main ideas, the heterogeneous multicore architecture and the security-hardened RISC-V Secure CoProcessor (RVSCP). The RVSCP (the ⚓ symbol in Figure 2.12) is designed exclusively for TEE with many SCA resilient features, and the heterogeneous design is for separating the TEE processor from the application processors (the 🚀 symbol in Figure 2.12). As seen in the figure, the two processors of TEE and REE are coupled tightly together with many hardware primitives, enforcing strong isolation between secure and non-secure domains.

The threat model of HECTOR-V considers a strong software adversary that directly exploits architectural weaknesses, and a strong physical adversary that can corrupt the peripherals and memory communication. It also considers that a malicious enclave is trying to attack the system. Therefore, a trusted I/O path mechanism is the central element of the HECTOR-V [9]. The trusted I/O paths were done by the identifier-based strategy and executed by the communication fabric (the AXI4 buses of ①, ②, ③, and Ⓒ in Figure 2.12), thus making fine-grain protection between cores, peripherals, and DMA devices. To carry the IDs, the AXI4 bus protocol was extended by a 16-bit user signal with 1-bit core ID, 4-bit process ID, and 10-bit peripheral ID. Based on the IDs, every transaction is checked by the SM module (the 🛡 symbol in Figure 2.12), and any illegitimate access will be turned down. The core ID is permanently fixed in hardware, and the process and peripheral IDs are assigned at run time. Since the core IDs are hard-coded directly into the bus interface, no attacker can change those IDs.

The key difference of HECTOR-V from the CURE is that a hardware module does the SM, and the concept of SM ownership is introduced, which can be dynamically transferred between participants, thus providing a variety of uses. Only one SM owner was allowed at a time, and only the SM owner could define the access rights of resources. When the SM grants a peripheral, the corresponding participant (OS or enclave) will do the peripheral claiming process, and then it should do the peripheral releasing process afterward. Furthermore, to prevent the abuse of enclave-to-peripheral binding (such as in DoS attacks), HECTOR-V also allows the peripheral access withdrawal procedure to be initialized by other parties, as long as they also have valid IDs. When peripheral access withdrawal is issued and approved by the SM owner, the SM starts a timer and notifies the peripheral's owner accordingly, via the dedicated interrupt line (the 💣 symbol in Figure 2.12). After receiving the notification, the peripheral's owner should start the cleanup function to clear secrets. When the timeout is reached, the SM will force release the peripheral by removing the ID field in the firewall.

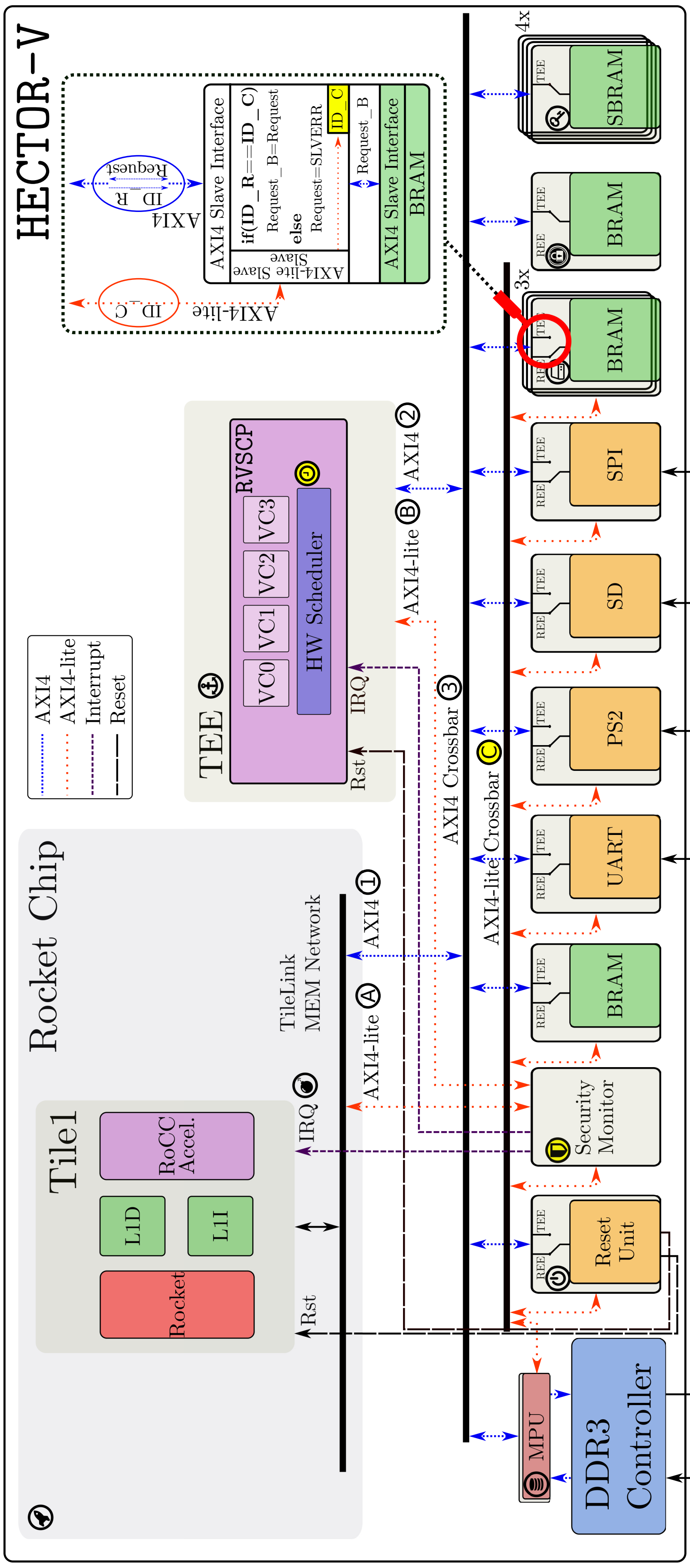


FIGURE 2.12: HECTOR-V architecture [9].

For the RVSCP design, two main ideas were deployed. They are the control-flow integrity unit with secure I/O to prevent control-flow hijack attacks, and the hardware scheduling (the 🕘 symbol in Figure 2.12) that allows TEE processor to multi-tasking even with bare-metal codes. The hardware scheduling module is also responsible for securing the enclave's context switch. For SCA resilience, due to the clean separation of TEE and REE processors, all cache-based and micro-architecture-based SCAs are prevented because there are no sensitive components (i.e., caches, branch predictors, and execution pipelines) are shared between the two domains. However, the attack surface still exists in the system. For example, a malicious SM owner could influence the availability of the system by permanently withdrawing peripherals, thus starving the enclaves [9].

In HECTOR-V, the secure boot process is achieved by only allowing the virtual core of VC0 to access the secure storage element and executes codes from the secure code storage (the 🔑 and 🔒 symbols in Figure 2.12, respectively). The access rights of these two elements are permanently hard-coded and exclusively owned by the VC0. The other virtual processors of VC1, V2, and VC3 can only fetch the codes from the claimable Block RAM (BRAM), the 🖴 symbol in Figure 2.12. At reset, the reset unit (the ⏻ symbol in Figure 2.12) configures the SM owner to be VC0 and starts the VC0 while keeping the REE processors halted. Then, the VC0 executes the Zero Stage BootLoader (ZSBL) in the secure code storage 🔒, thus making the first authentication of the system; this is the RoT of HECTOR-V. Subsequently, the code in ZSBL configures the MPU (the ⛁ symbol in Figure 2.12) for external memory access rights, like the Secure Digital card (SD-card) or Double Data Rate (DDR) memory. VC0 compares the hash value of the Berkeley BootLoader (BBL) with the expected one in secure storage element 🔑. If the BBL is successfully verified, it will be loaded to the main memory, and the VC0 will release the SD-card driver together with claimed DDR memory regions. Finally, the VC0 transfers the SM owner privilege over the REE processors and triggers the reset unit ⏻ to start the REE. Compared to CURE, HECTOR-V has a clear secure boot procedure, while in CURE, the RoT is assumed by reset. Furthermore, HECTOR-V guarantees that the secrets stored in secure storages (🔑 and 🔒) are still protected after boot.

2.3.3 SiFive WorldGuard

WorldGuard [10] was developed by SiFive for strengthening TEE isolation, and its hardware architecture is shown in Figure 2.13. WorldGuard resembles CURE and HECTOR-V in many ways because it relies too on ID implementation across the entire system for strong isolation at many levels. In WorldGuard, each core has an assigned world ID, and each process on the core has a process ID. Similar to CURE and HECTOR-V, these IDs are propagated throughout the system, including cores, caches, buses, memories, peripherals, and DMA devices. The difference is that WorldGuard leverages the PMP and Physical Memory Attributes (PMA) supported in the RISC-V ISA, while CURE and HECTOR-V are not. In HECTOR-V, an exclusive processor is reserved only for TEE, wholly separated from the application processors, while in WorldGuard, TEE and REE share the same processor complex.

About the secure boot process, WorldGuard uses a similar approach with HECTOR-V that stores the first bootloader and root keys in ROM at the time manufactured. The bootloader then verifies and loads the SM into RAM for further processing. The difference is, in WorldGuard, the bootloader's source code is open for examination. Furthermore, because WorldGuard doesn't have an exclusive processor for the boot process, it also doesn't have secure storages exclusively for TEE that cannot be accessed by an application processor (REE processor) after boot.

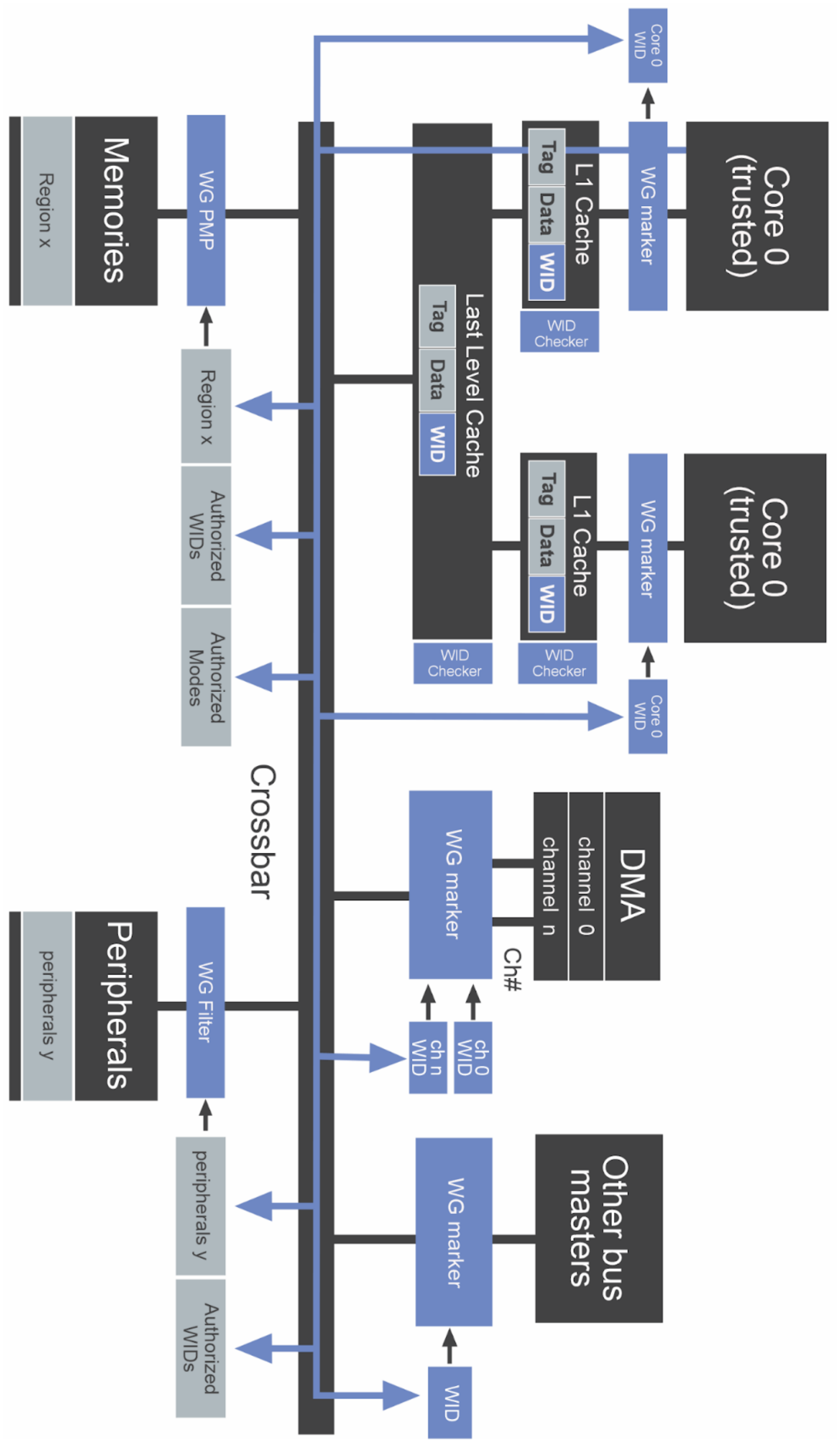


FIGURE 2.13: SiFive WorldGuard architecture [10].

2.3.4 ITUS

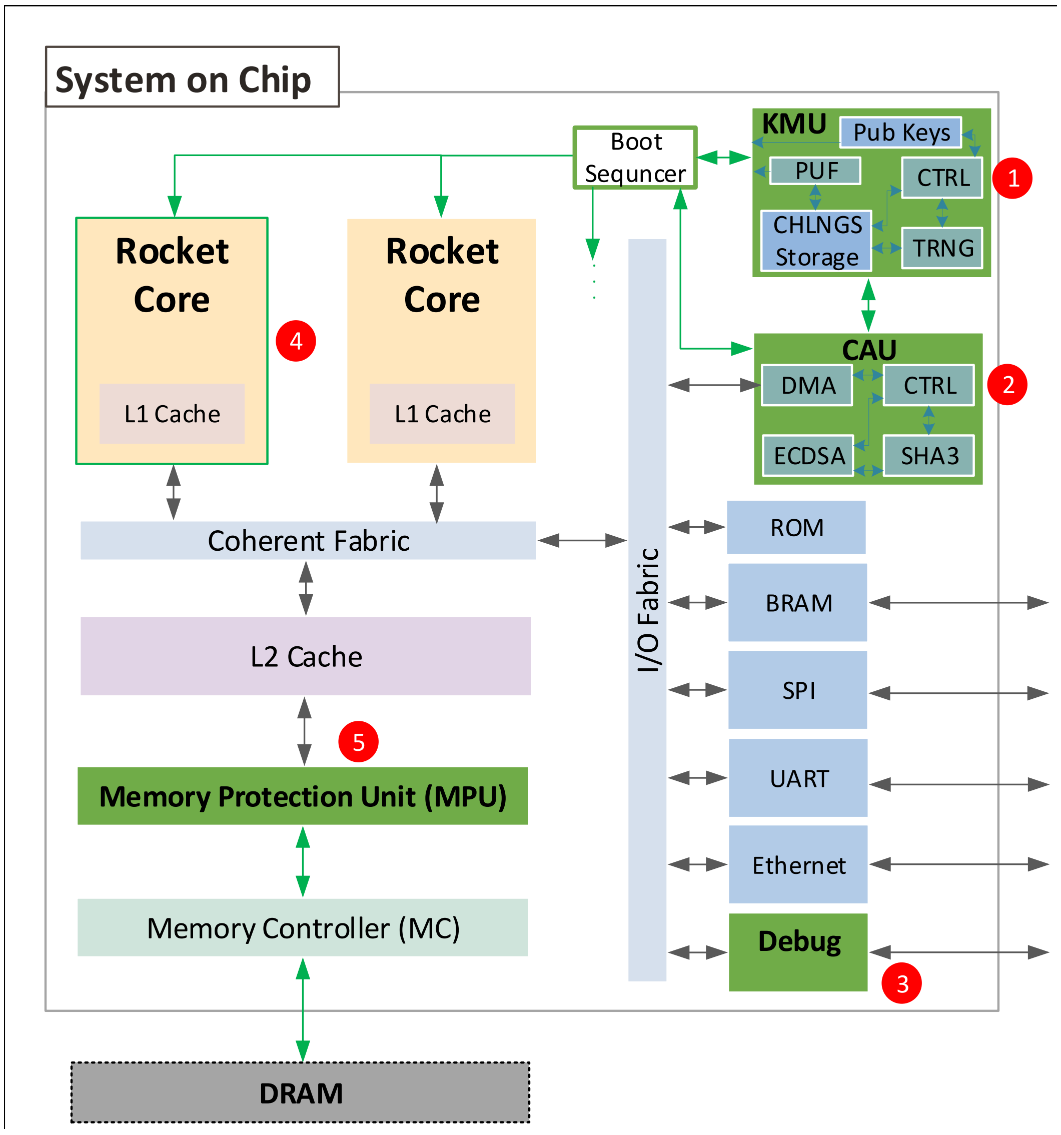


FIGURE 2.14: ITUS architecture [11,12].

ITUS is a RISC-V-based SoC aiming for the secure boot process with RoT in TEE. The new hardware primitives were first introduced by V. B. Y. Kumar *et al.* in [11], and then the complete secure boot procedure was realized by J. H.-Yajya *et al.* in [12]. Figure 2.14 shows the ITUS architecture [11, 12]. ITUS is not proposing a new TEE model like CURE or focusing on isolating environments and SCA resilience like HECTOR-V and WorldGuard. ITUS attempts to solve the RoT problem in TEEs by a pure hardware approach. As seen in Figure 2.14, the developed hardware modules are the KMU (number ①) for keys generation and keys distribution, CAU (number ②) for the authentication process, secure debug module (number ③) for some SCA protections over the debug channel, and MPU (number ⑤) for enforcing isolation and providing off-chip communication encryption. All the ①, ②, ③, and ⑤ are for providing a secure boot flow completely out of touch of the

TEE processors in number ④. In the secure boot process, the CAU is used for verifying the Chain-of-Trust (CoT) integrity based on the Elliptic Curve Digital Signature Algorithm (ECDSA) and SHA3 accelerators. The keys are generated in KMU using PUF and True Random Number Generator (TRNG) based on the One-Time Programmable (OTP) memory seed. In MPU, the AES Galois/Counter Mode (AES-GCM) is used to guarantee the confidentiality of the external memory. The secure memory-range registers are added at the MPU's bus interface for the authentication process. Furthermore, a simplified Bonsai Merkle Tree (BMT) is used in MPU as a countermeasure against memory replay attacks. At boot, the boot sequencer, a Finite State Machine (FSM), first activates the KMU to retrieve the root key, then passes it to the CAU to generate the subsequent asymmetric keys. Finally, if the BBL authentication is corrected, the boot sequencer wakes up the TEE cores. In summary, ITUS provides a TCB for TEEs with many hardware features for secure boot, encryption and authentication of the off-chip memory, key management, and cryptographic accelerators [12].

By using all hardware modules for boot and keys provisioning, the root keys are inaccessible to the eyes of the TEE processors after boot, thus achieving the set goal. However, there are two major disadvantages to this approach. The first one is due to the fixed hardware solution, the boot procedure is not flexible and unadaptable compared to other approaches, especially the HECTOR-V architecture. The second disadvantage is that because of the complete hardware implementation, all the cryptographic functions used in the boot process will need to be realized in hardware, thus increasing the resources significantly if the complexity of those cryptographic functions is high.

2.3.5 Comparison

Up to now, the major publications that has a security-driven TEE-related RISC-V-based computer system are the CURE [8], HECTOR-V [9], SiFive WorldGuard [10], and ITUS [11, 12]. Although their designs have some ideas that resemble each other, their implementations and set goals were quite different. For instance, the main goal of the CURE [8] is to propose a new TEE model with multi-type enclaves to fit multi-purposes while maintaining strong isolation enforced by hardware modifications at many levels throughout the entire system. On the other hand, the goal of HECTOR-V [9] is to make a clear separation between secure and non-secure domains by using a security-harden co-processor designed exclusively for TEE. WorldGuard [10] was being developed nearly at the same time as the HECTOR-V, and its goal was nearly the same. The differences are, WorldGuard didn't have a heterogeneous architecture, and it relied on the PMP and PMA features supported by the RISC-V ISA for their implementations. For the ITUS [11, 12], the achievement was implementing a secure boot process executed purely by hardware, removing TEE processors completely from the boot scheme. Table 2.2 shows the comparison between the implementations as mentioned earlier regarding their security features and flexibilities. The comparison keys are:

- **Open-source.** The source codes are available for modification.
- **Secure boot.** Has a clear secure boot process with RoT.
- **Flexible boot process.** The boot process is flexible and can be updated.
- **TEE and secure boot isolation.** The secure boot process is done by the TEE processor or by another party?
- **Exclusive TEE processor.** TEE is executed by an exclusive processor or sharing processors with the REE?
- **Exclusive secure storage.** The proposed system has a safe place to store the root key(s) and the first bootloader. The storage could be ROM or otherwise, as long as it is inaccessible for the application processors after boot.
- **Secure I/O paths.** The proposed architecture could bind a peripheral to an isolated environment and make a direct secure connection.
- **Cryptographic accelerators.** Hardware accelerators for cryptographic functions are provided in the system.
- **SCA resilience.** The ability to prevent some SCAs.
- **Hardware cost.** The amount of resources that the proposed architecture needs, compared in the overhead ratio. Low is around 5-10%, medium is around 20%-25%, and high is around 50%.
- **High expressiveness.** The proposed architecture allows multi/hyper-threading, flexible isolation enforcement, memory encryption, cache partitioning, and so on.
- **Low porting effort.** The development effort for adapting/porting the proposed design.

TABLE 2.2: Security-driven RISC-V computer systems comparison regarding the security and flexibility features; ●, ◐, and ○ rank the performance from best to worst, respectively.

	CURE [8]	HECTOR-V [9]	WorldGuard [10]	ITUS [11,12]
Open-source	○	○	◐	○
Secure boot	◐	●	◐	●
Flexible boot process	●	●	●	○
TEE & secure boot iso.	○	○	○	●
Exclusive TEE processor	◐	●	◐	○
Exclusive secure storage	○	●	○	●
Secure I/O paths	●	●	◐	○
Crypto. accel.	○	○	◐	●
SCA resilience	●	●	◐	○
Hardware cost	●	◐	●	○
High expressiveness	◐	●	◐	○
Low porting efforts	○	○	◐	●

Chapter 3

Background Research

3.1 RISC-V Architecture

3.1.1 Overview

RISC-V is an open-source Instruction Set Architecture (ISA) with the goal to support research and education, also aiming for a free and open standard for industry implementations [13, 16]. RISC-V was initially developed by the Berkeley architecture group in 2014 [45]. Now it is maintained by the RISC-V Foundation, and the latest released versions can be found at their website [46]. The key features of the RISC-V ISA are:

- Completely open for both academic and industry.
- Suitable for direct native hardware implementation.
- Avoids "over-architecting."
- Separated into a small base integer ISA with optional extensions.
- Support IEEE-754 floating-point standard [103].
- Support extensive extensions with specialized variants.
- 32-bit/64-bit address spaces have variants for custom applications.
- Support highly-parallel multicore/manycore.
- Optional variable-length instructions.
- Fully virtualizable to ease hypervisor development.
- Simplifies experiments with new privileged architecture designs.

With the benefits of open-source ISA, many cores and System-on-Chips (SoCs) were released under the free license. Currently, on the foundation website [50], there are over a hundred RISC-V cores and a couple of dozen RISC-V SoCs to be chosen.

The key innovation of RISC-V is that it needs only the base integer instruction set to operate. Then many extensions can be added depending on needs. According to the specifications [16], the base integer ISAs and extension ISAs are listed in Table 3.1.

TABLE 3.1: List of base and extension ISAs [16].

RV32I	32-bit base integer
RV32E	reduced version of RV32I, designed for embedded systems
RV64I	64-bit base integer
RV128I	128-bit base integer
M	multiplication and division extension
A	atomic instructions extension
F	single-precision floating-point extension
D	double-precision floating-point extension
Q	quad-precision floating-point extension
L	decimal floating-point extension
C	compressed instructions extension
B	bit manipulation instructions extension
J	dynamically translated languages extension
T	transactional memory extension
P	packed-SIMD instructions extension
V	vector operations extension
Zam	misaligned atomics extension
Ztso	total store ordering extension

Besides the keywords listed in Table 3.1, the "G" keyword (shorted for Generic) is frequently used to represent the combination of "IMAFD." For example, RV32GC means RV32IMAFDC.

The base integer and extension ISAs in Table 3.1 are called the unprivileged ISAs. The privileged instructions sets are also standardized in [13]. Three privilege levels are currently used as shown in Table 3.2.

TABLE 3.2: RISC-V privilege levels [13].

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	*Reserved*	
3	11	Machine	M

Privilege levels are used to provide protection between different software stack components. The machine-level has the highest privileges and is the only mandatory privilege level for a RISC-V hardware platform. Code run in Machine-mode (M-mode) is usually inherently trusted, as it has low-level access to the machine implementation. M-mode can be used to manage secure execution environments on RISC-V. User-mode (U-mode) and Supervisor-mode (S-mode) are intended for conventional application and operating system usage, respectively. Each privilege level has a core set of privileged ISA extensions with optional extensions and variants. For example, M-mode supports an optional standard extension for Physical Memory Protection (PMP).

Besides the M, S, and U modes, there is also the Debug-mode (D-mode) that basically can be counted as a privilege because it has more access than even the M-mode. Implementations might provide anywhere from 1 to 3 privilege modes trading off reduced isolation for lower implementation cost, as shown in Table 3.3.

TABLE 3.3: Supported combinations of privilege modes [13].

No. of levels	Supported modes	Intended usage
1	M	Simple embedded systems
2	M, U	Secure embedded systems
3	M, S, U	Unix-like systems

Figure 3.1 shows some of the possible software stacks supported by the RISC-V architecture. The left-hand side shows a simple system that supports only a single application running on an Application Execution Environment (AEE). The Application Binary Interface (ABI) includes the supported U-mode plus a set of ABI calls to interact with the AEE. The ABI hides the AEE from the application to allow greater flexibility in implementing the AEE.

The middle configuration shows a conventional Operating System (OS) that can support multiple applications. Each application communicates over an ABI with the OS, which provides the AEE. Just as applications interface with an AEE via an ABI, RISC-V operating systems interface with a Supervisor Execution Environment (SEE) via a Supervisor Binary Interface (SBI), which comprises U-mode and S-mode.

Finally, the rightmost configuration shows a virtual machine monitor configuration where a single hypervisor supports multiple OSs. Each OS communicates via an SBI with the hypervisor, which provides the SEE. The hypervisor communicates with the Hypervisor Execution Environment (HEE) using a Hypervisor Binary Interface (HBI) to isolate the hypervisor from details of the hardware platform. Hardware implementations of the RISC-V ISA will generally require additional features beyond the privileged ISA to support the various execution environments (AEE, SEE, or HEE).

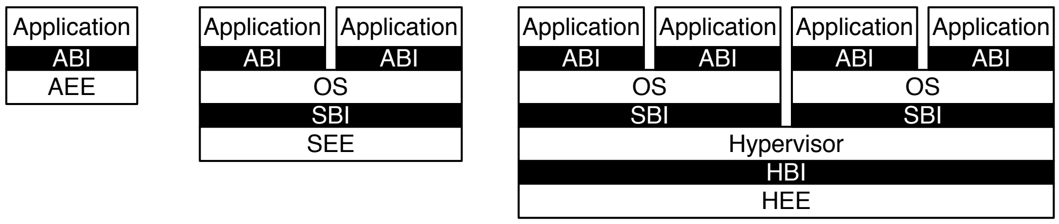


FIGURE 3.1: Different implementation stacks supporting various forms of privileged execution [13].

3.1.2 Ecosystem

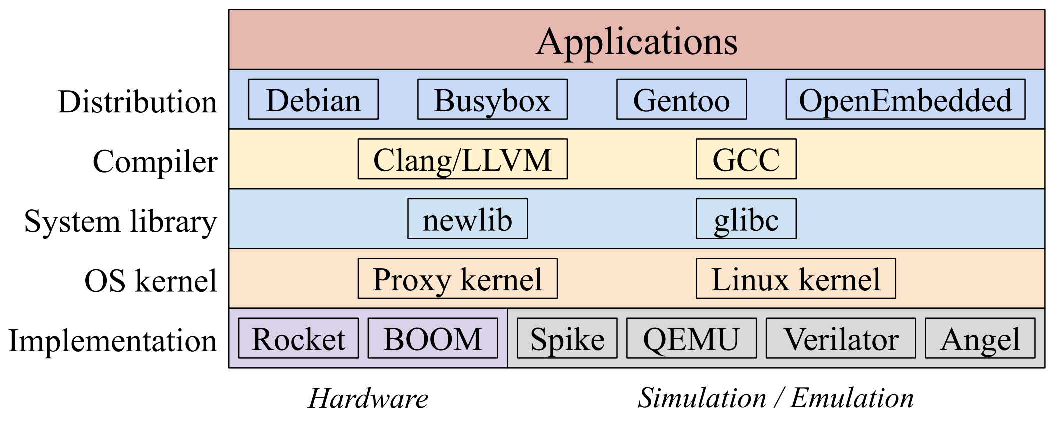


FIGURE 3.2: RISC-V software stack.

Figure 3.2 describes the software stack for the RISC-V ecosystem. Most of the above components are built and provided free by the open-source community. It will be an excellent motivation for scientists and start-up companies to participate in research in the field of computer architecture.

RISC-V GCC is a C compiler program for RISC-V architecture. The toolchain is open-source at [104]. It supports many ISA configurations with different ABIs. For example, the architecture could be RV32I, RV32E, RV64I, and RV128I with the option for all standard extensions. When compiled, a user must specify the toolchain configuration. RISC-V GCC is mandatory for creating a system library and Linux kernel. Currently, the toolchain supports GCC version 11.1 (commit 5964b5c). To use an older GCC version, just check out the corresponding commit:

- GCC 7.2: commit 36e932c
- GCC 7.3: commit 87fb575
- GCC 8.1: commit 3c148a7
- GCC 8.2: commit be9abee
- GCC 8.3: commit bdf3ad8
- GCC 9.1: commit 1df2a6b
- GCC 9.2: commit 3e6d81b
- GCC 10.1: commit 602fad9

System libraries are special functions or programs using which application programs or system utilities access Kernel's features. These libraries implement most of the functionalities of the OS do not require kernel module's code access rights. The system libraries are essential components in building a full image of the Linux OS. The most used libraries are Glibc and Newlib:

- **Glibc:** system library for the Linux kernel. The GNU C Library project provides the core libraries for the GNU system and GNU/Linux systems, and many other Linux systems. Github repository of RISC-V Glibc can be found at [105].
- **Newlib:** system library for the proxy kernel. Newlib is a C standard library implementation intended for use on embedded systems. It is a conglomeration of several library parts under free software licenses, making them easily usable on embedded products. Github repository of RISC-V Newlib can be found at [106].

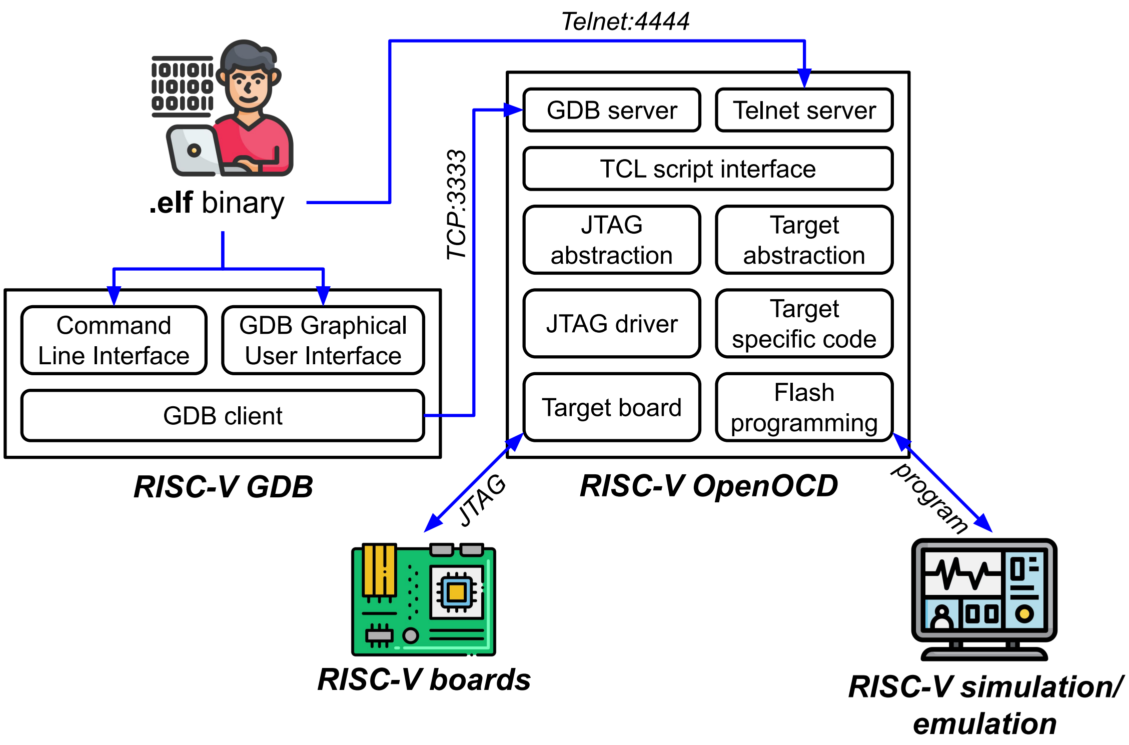


FIGURE 3.3: RISC-V debug architecture.

Figure 3.3 shows the RISC-V debug architecture. The most common tool used for debugging is the Open On-Chip Debugger (OpenOCD). OpenOCD is open-source software that interfaces with a hardware debugger's Joint Test Action Group (JTAG) port. It provides debugging and in-system programming for embedded target devices. It also provides the ability to access NAND/NOR flash memory devices attached to the processor on the target system. RISC-V OpenOCD source codes are open at [107].

Besides the OpenOCD, GDB is also an irreplaceable part of the debug architecture. GDB allows you to see what is going on 'inside' another program while it executes – or what another program was doing at the moment it crashed. Four crucial tasks that GDB allows developers to do: start your program, add break-points (either soft or hard) to your program, examine what has happened, and change things in your program. RISC-V GDB is included in the toolchain [104].

For the simulation/emulation tools, the followings are currently the most common ones:

- **Spike.** Spike is an interpreting simulator that provides an instruction-by-instruction trace accurate simulation of a RISC-V processor. Spike is the "golden reference" simulator for the RISC-V ISA, and its behavior is the reference for hardware and software. The focus of an interpreter is typically behavioral accuracy for verification. The input of the spike simulation can be RISC-V Linux image or bare-metal machine code. Spike program communicates with host machine via command line. The binary load into spike memory via command line or proxy kernel. The output of the simulation is a file consist value of register and memory. Spike allows debugging programs over GDB. Spike program is published on a Github repository [108].
- **QEMU.** Quick EMUlator (QEMU) is a fast binary translator and offers Linux U-Mode simulation and full system emulation. Given that it is the fastest RISC-V simulator up to now, QEMU is the most frequently used tool for tasks that would be too costly to run on simulated hardware—for instance, testing GCC, Binutils, and Glibc library. Furthermore, QEMU is fast enough to simulate a Linux environment containing a build environment for self-hosted builds. The input of QEMU is a RISC-V Linux binary image. The image passes to QEMU via the command line parameter. The simulation output is shown in the command line or QEMU simulation interface. Source codes of QEMU can be found at [109].
- **Verilator.** Verilator is a cycle-accurate software simulation based on Register Transfer Level (RTL) implementations. Hence, Verilator can provide the exact cycle-by-cycle behavior of one particular RISC-V implementation. Upon the simulated hardware, the software can be simulated with the input of a binary file. The output can be a file containing register values or a waveform. Necessary parameters can be passed to the simulated program over the command line. Verilator Github repository is at [110].
- **Angel.** Angel is a Javascript RISC-V ISA simulator often used to run RISC-V Linux with busybox. The primary goal of the Angel is to create an interactive Linux session with minimal work. Angel does not require the RISC-V toolchain to be installed. Sources of Angel are open at [111].

Table 3.4 shows a brief comparison between RISC-V simulations tools.

TABLE 3.4: Comparison of RISC-V simulation tools.

	Verilator	**Spike**	**QEMU**	**Angel**
Simulation level	Cycle-accurate	Trace-accurate	Functional	Functional
Speed (inst./s)	10k $\sim$ 100k	10mil. $\sim$ 100mil.	100mil. $\sim$ 1bil.	$\sim$13.5mil.

3.1.3 Chisel and FIRRTL

Chisel is a hardware construction language embedded in the high-level programming language of Scala. In other words, Scala is the language, and Chisel is the library. Chisel was first developed at UC Berkeley to support advanced hardware design and circuit generation. The latest iteration of Chisel is Chisel3, which uses Flexible Intermediate Representation for RTL (FIRRTL) as an intermediate hardware representation language [47, 112]. Figure 3.4 shows the workflow of the Chisel program, which can create C++ code to simulate, Verilog code to demo in Field-Programmable Gate Array (FPGA) board or even synthesis in Application Specific Integrated Circuit (ASIC) layout.

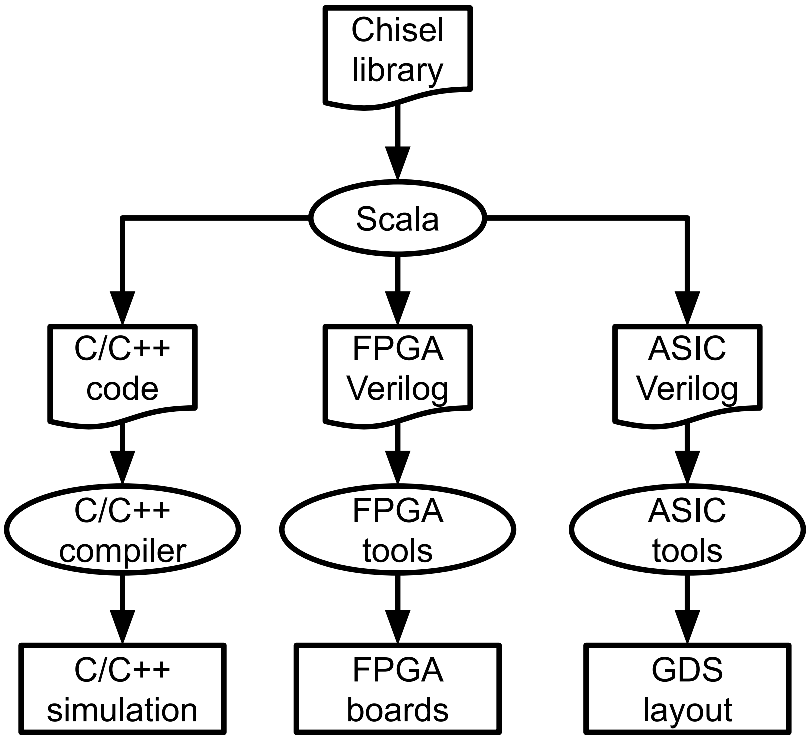


FIGURE 3.4: Typical Chisel workflow.

FIRRTL is an intermediate representation for digital circuits designed as a platform for writing circuit-level transformations. A FIRRTL compiler is constructed by chaining together transformations such as simplification, verification, transmittance, and emittance of the input circuit. Furthermore, the FIRRTL compiler can support custom user-defined circuit transformations. Therefore, either in simulation, FPGA, or VLSI, as long as the primitive circuits are defined, FIRRTL-based Chisel can generate the final RTL sources.

3.2 Trusted Execution Environment

3.2.1 Overview

Typically, security issues cannot be solved with remote computing systems. For example, users cannot control the physical components running on their computers. Data can be dumped, and malicious programs can be run on your computer from another computer in the system or over the internet. The efforts of hardware manufacturers to overcome these issues are to get a trusted mechanism. Thus, the idea of a Trusted Execution Environment (TEE) is presented. Traditionally, TEEs provide the following three guarantees:

- **Integrity.** The code and data cannot be tampered with (e.g., by running arbitrary code within a partition).
- **Confidentiality.** The attacker cannot learn the runtime content of the application (e.g., secret keys and code control flow).
- **Attestation.** The integrity proof is provided to a remote party that the environment has not been tampered with and is safe.

The idea of TEE is to provide isolation between applications, thus creating a barrier between programs. The barrier is often done by a privilege separation approach and enforced by hardware primitives such as memory isolation. To isolate low-privilege codes (user's applications) from high-privilege codes (OS's services) or vice versa, the earlier version of TEEs simply encrypts the want-to-be-protected codes and has some form of authentication between involving parties. Modern TEEs nowadays have much more complex than that for the trusting mechanism. However, the main idea stays the same. A typical setup for an isolated program in TEE, called enclave, will need a True Random Number Generator (TRNG) for keys generation and several cryptographic functions for creating the crypto-keys, hashing, signing, verification, and even cipher encryption/decryption. To protect an enclave completely, modern TEEs often have a Trusted Firmware (TF) at M-mode for exclusive services, i.e., not relying on the OS's services. Common TF's services are the dedicated memory allocation, flushing caches at every enclave's context switch, and message encryption in and out of an enclave. Furthermore, TF also plays the role of Trusted Computing Base (TCB) for setting up the trusted domain and ensuring enclaves' barriers. Because of the vital role of TF, checking TF integrity must be done by a secure boot process, and the TF authentication is often called the Root-of-Trust (RoT) in a TEE system.

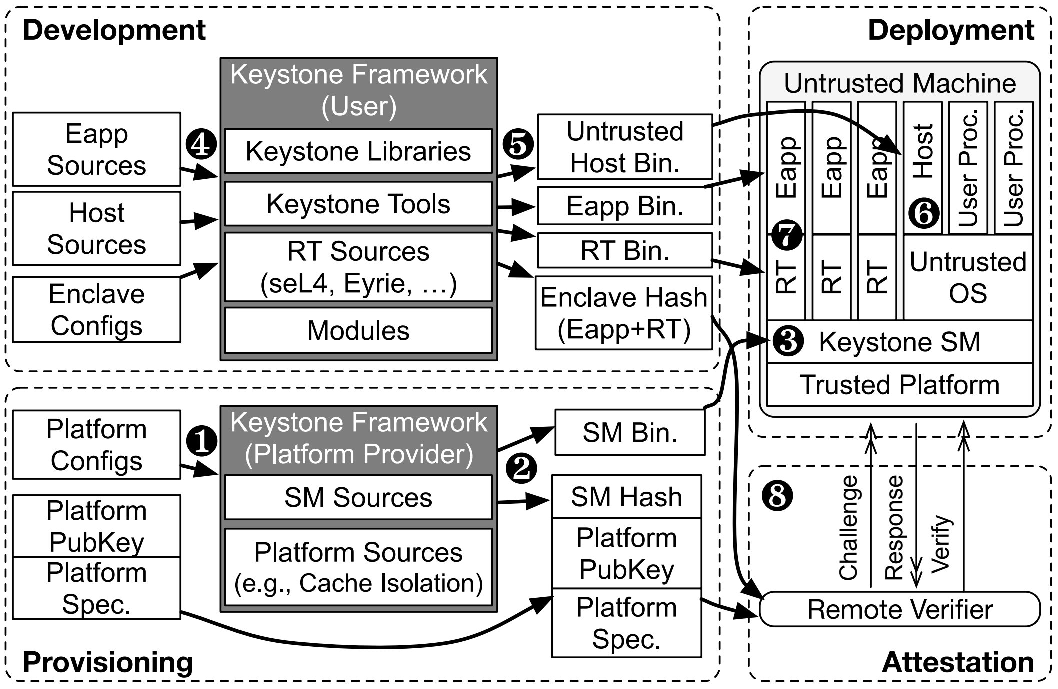


FIGURE 3.5: Keystone TEE implementation [7].

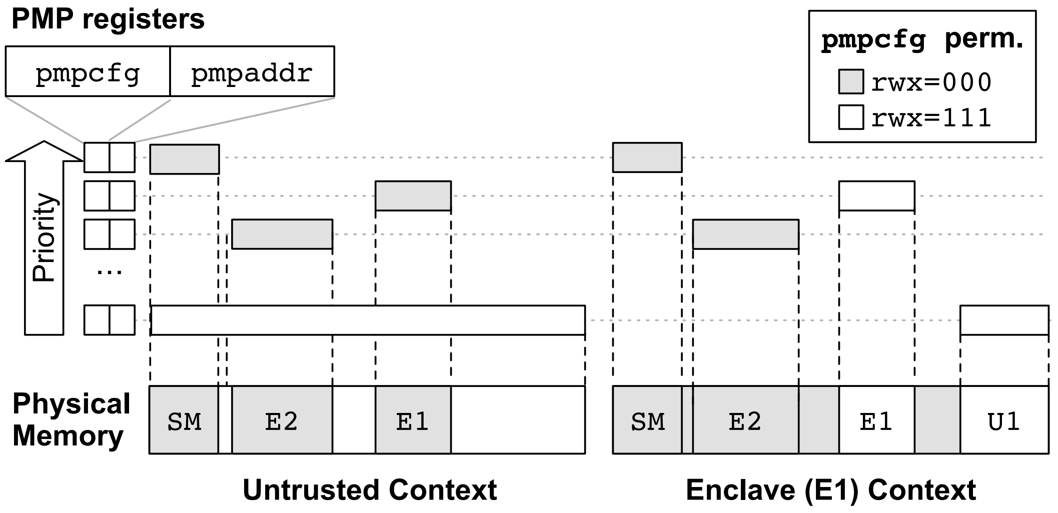


FIGURE 3.6: RISC-V PMP architecture [7].

3.2.2 Secure Boot Procedure for TEE

The boot procedure and keys generation described in this sub-section is based on the Keystone framework [7]. Keystone is not a specific TEE implementation but a framework for customizing TEE depending on needs. Furthermore, it is currently the most completed TEE based on RISC-V architecture.

To establish a secure boot process, two things need to be trusted in the beginning:

- **Hardware manufacturer:** For legal reasons, chip manufacturers must be held responsible for their products. Thus, we can trust the silicon manufacturing process to create the right hardware such as RoT.
- **Software provider:** Similarly, for legal reasons and brand reputation, software providers also need to meet security standards to protect their users.

What we can't trust are the data transmission environment and the infrastructure:

- **Infrastructure:** There are many factors that the infrastructure providers cannot control. For example, security vulnerability on virtualization software allow hackers to attack other virtual machines directly from the hacked virtual machine.
- **Data transmission environment:** It is best to assume that a hacker can capture the data on the transmission line, like the Internet, for example.

The RoT in a TEE system is the very first authentication. That means self-authentication at the manufacturer level. As a result, each manufacturer has to create their pair keys, called the manufacturer's secret-key SK_M and public-key PK_M, as shown in Figure 3.7. The pair keys should be asymmetric like Rivest-Shamir-Adleman (RSA) pair keys [113], Elliptic Curve Digital Signature Algorithm (ECDSA) pair keys [114], or Edwards-curve Digital Signature Algorithm (EdDSA) pair keys [115]. In this case, the used algorithm is the Ed25519 [116] as seen in the figure. The manufacturer publishes the PK_M and conceals the SK_M.

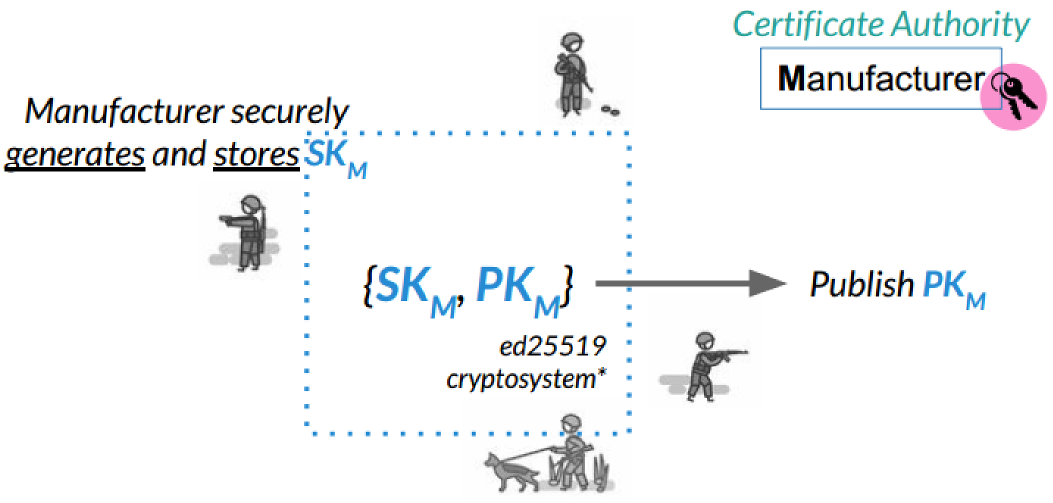


FIGURE 3.7: Manufacturer's pair keys.

The next step in the trust flow is the integrity of the silicon products. The chip fabricated at the trustworthy manufacturer needs to have its pair keys, named device's secret-key SK_D and public-key PK_D. These pair keys are also created by the manufacturer and stored inside the chip at the time manufactured, as described in Figure 3.8. Finally, the manufacturer uses its SK_M

to sign and endorse the chip, thus creating the device's certificate $Cert_D$, as shown in Figure 3.9. Based on this $Cert_D$, we can now determine whether the content in the boot Read-Only Memory (ROM) can be trusted or not, thus ensuring the chip's integrity and its Zero Stage BootLoader (ZSBL).

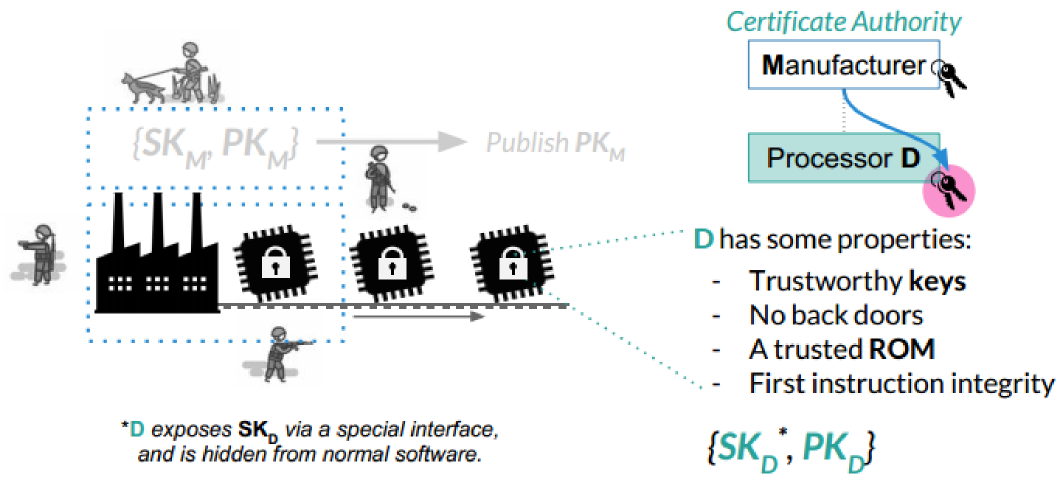


FIGURE 3.8: SoCs are manufactured with built-in device's pair keys.

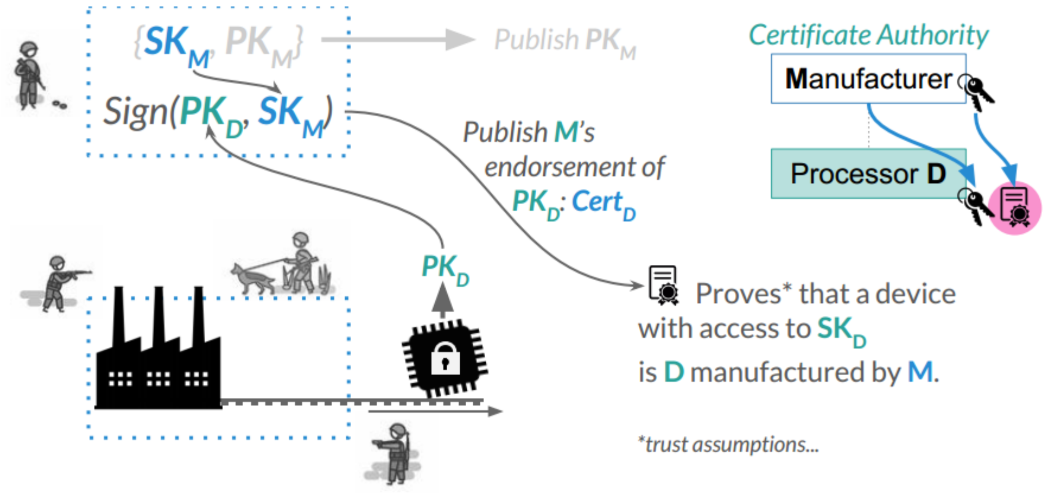


FIGURE 3.9: Manufacturer signs and creates the device's certificate.

With the secure SK_D and PK_D pair keys, the next step in the CoT is about software, starting with the bootloaders. At this point, we have the trusted chip and the trusted boot ROM. Any software after this point is considered vulnerable to attack because it is located on easy-to-attack physical memories such as flash, Secure Digital card (SD-card), and Double Data Rate (DDR) memory. Therefore, a trust mechanism with hash and encryption algorithms is required.

First of all, the RoT (secure chip with trusted boot ROM) hashes the software binary to create the H_S, as shown in Figure 3.10. The programs that need to be hashed are the sensitive programs like OS-related applications and those that need a special privilege after boot. Each software has its own generated H_S. After the H_S is generated, the SK_D together with the H_S are used to create the software pair keys of secret-key SK_S and public-key PK_S via a Key Derivation Function (KDF), as described in Figure 3.11.

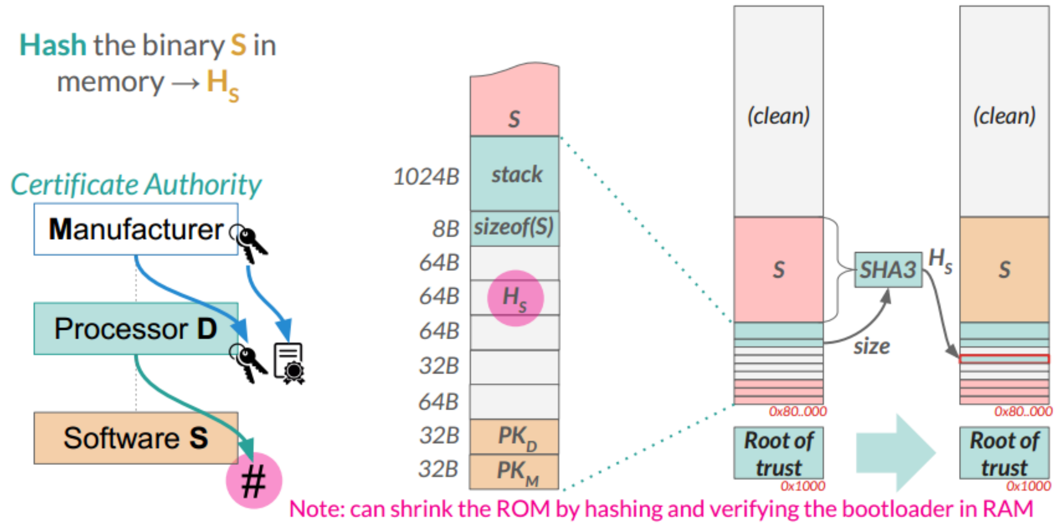


FIGURE 3.10: The RoT hashes the software binary.

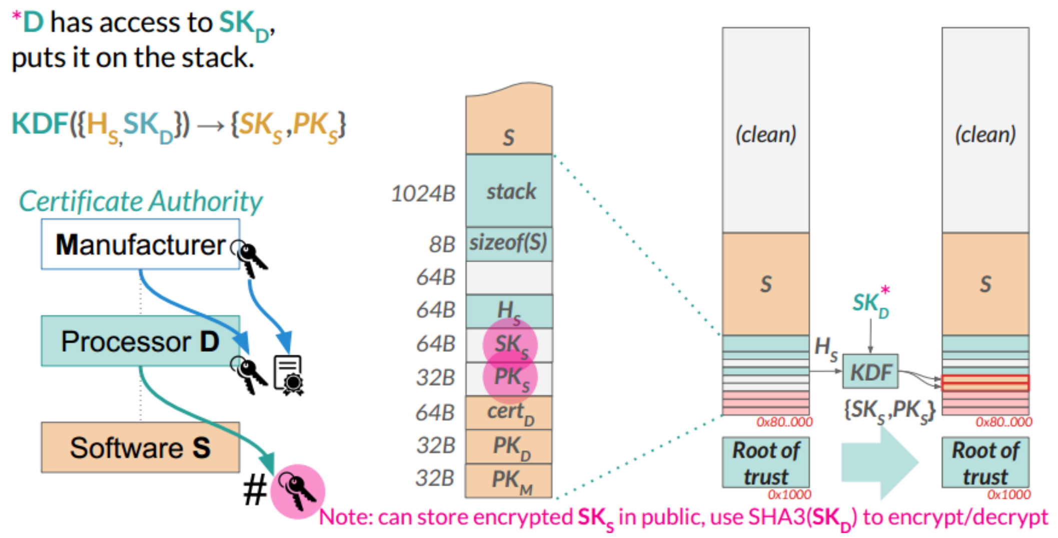


FIGURE 3.11: Software pair keys are created.

After the SK_S and PK_S are created, the SK_D is used to sign and endorse the PK_S together with its H_S, thus creating a software's certificate $Cert_S$, as shown in Figure 3.12. Now the $Cert_S$ can be used to verify the integrity of the software because it is bound by the H_S and signed by the device. And with the chain of certificates, we can do the attestation report down to the manufacturer. Finally, when all the necessary $Cert_S$ are generated, the machine can boot to the OS space. However, because the boot image S is untrusted, all the sensitive data must be cleaned up beforehand, like stack and SK_D, as seen in Figure 3.13.

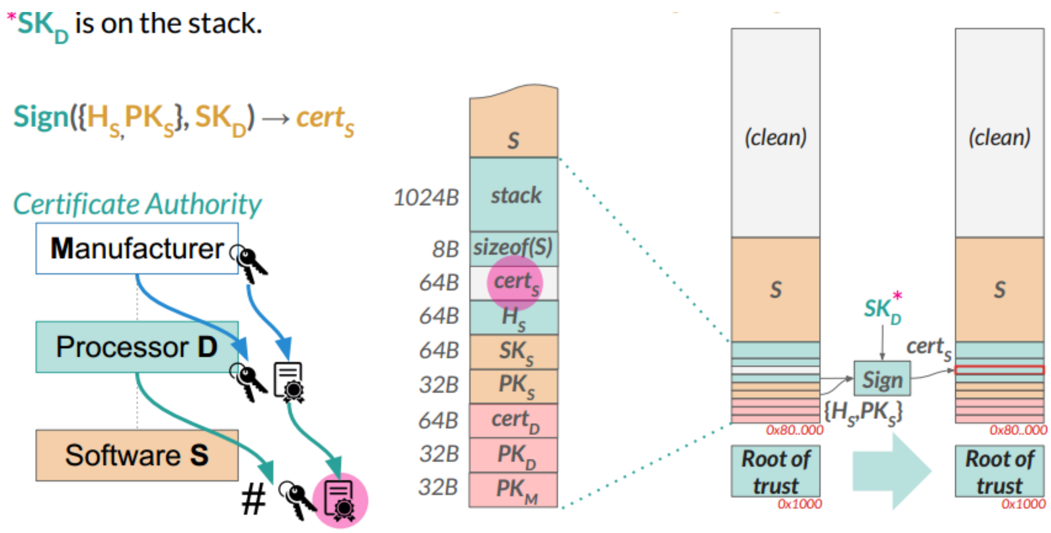


FIGURE 3.12: Software's certificate is created.

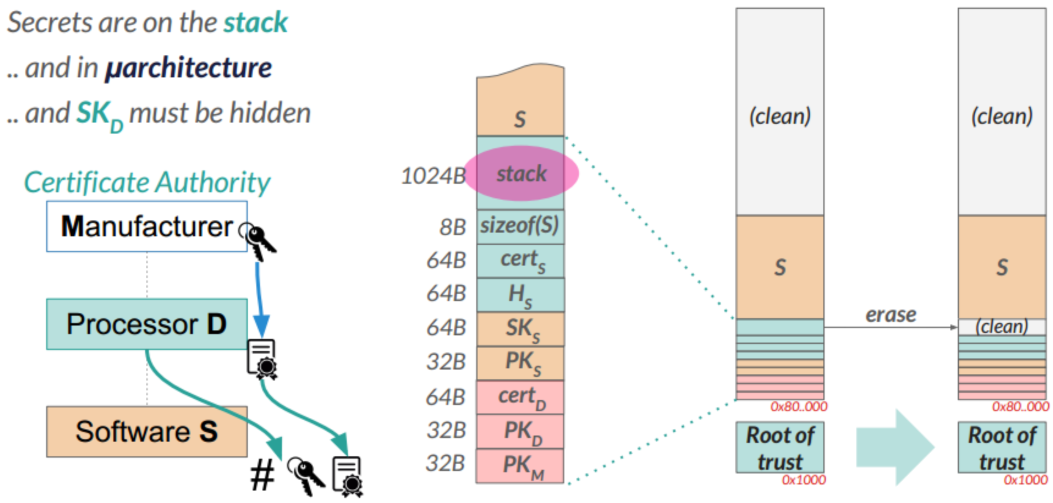


FIGURE 3.13: RoT cleans up SK_D and stack then boot to the OS space.

3.3 VexRiscv: A 32-bit RISC-V Microcontroller

3.3.1 The Architecture

In this section, a 32-bit RISC-V MCU is presented. The core processor is the VexRiscv CPU, originally from the SpinalHDL research group [92]. VexRiscv is a 5-stage pipeline in-of-order processor with the RV32IM ISA extensions. It has many options and complimentary plugins to provide various functionalities. According to the site [92], some parts of the core can be turned on or off as wished via the plugin system. For instance, new functions or instructions can be added without modifying any of the CPU sources, thus making the CPU design completely parameterizable. If no plugin is defined, the VexRiscv core contains only the 5-stage pipeline as the definition.

To make the first step into understanding the RISC-V, this VexRiscv was implemented on both FPGA and VLSI. An MCU based on this architecture was made, and Figure 3.14 shows the architecture. The core processor shown in the figure was generated with the full option, including cache trashing, cache exceptions, single cycle barrel shifter, debug module via JTAG, dynamic branching, and MMU [92]. Compared to the original design from the SpinalHDL research group [92], the caches sizes were modified to fit the chips better, and the SPI controller was added for the usage of the SD-card. The GPIO has 16 LEDs and 16 switches. The on-chip memory was used in chips, while the off-chip SRAM controller was replaced for the FPGA versions. The boot ROM has 8-KB with 7-KB of hard-code in combinational logics and 1-KB in SRAM for the stack. The 1-KB of SRAM stack can be used later after boot. The 7-KB hard-code boot ROM inits the CSRs in the CPU, prints the initial text to the UART, starts the SD-card, loads the program from the SD-card to the on-chip memory, jumps to the on-chip memory, and executes there. The MCU can self-boots to run any desired software in the SD-card for an embedded application with this boot flow. For both FPGA and VLSI implementations, if the MCU is successfully booted, the terminal should show the content, as seen in Figure 3.15. The source codes and guide for replicating this proposed MCU are published in the given repository [117]. The architecture presented in this section was later used to develop trigonometric hardware with custom instruction in [95] (2021).

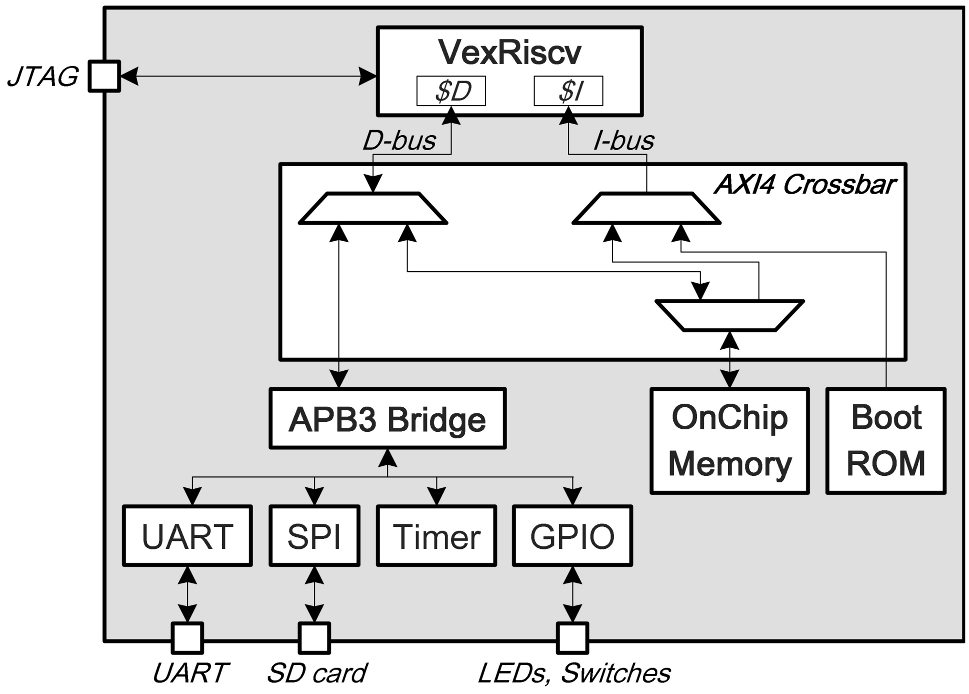
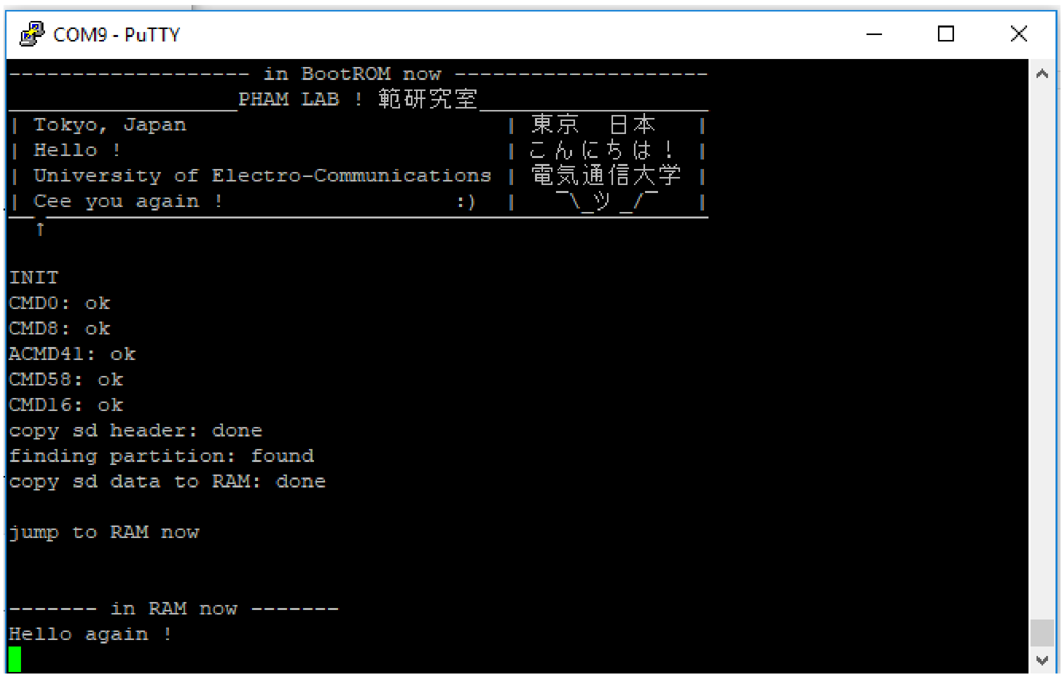


FIGURE 3.14: Block diagram of the implemented VexRiscv MCU.



```
------------------- in BootROM now --------------------
__________________PHAM LAB ! 範研究室_________________
| Tokyo, Japan                            | 東京  日本   |
| Hello !                                 | こんにちは！ |
| University of Electro-Communications   | 電気通信大学 |
| Cee you again !                      :) |  ¯\_ツ_/¯   |
  ↑

INIT
CMD0: ok
CMD8: ok
ACMD41: ok
CMD58: ok
CMD16: ok
copy sd header: done
finding partition: found
copy sd data to RAM: done

jump to RAM now


------- in RAM now -------
Hello again !
```

FIGURE 3.15: Terminal shows the VexRiscv core is booting.

3.3.2 FPGA Implementation

For FPGA versions, the on-chip memory in Figure 3.14 was replaced by an off-chip SRAM controller. Depending on the FPGA chip being implemented, the off-chip SRAM controller could be SDRAM, DDR2, or DDR3 memory controller. The available FPGA demos in the repository [117] are Arrow-SoCKit, DE0-Nano, DE2-115, DE1-SoC, DE4, and TR4 FPGA boards. The FPGA build reports are collected and given in Table 3.5. Because SRAM controller IPs in the boards are vastly different, the results in Table 3.5 were reported without them.

TABLE 3.5: VexRiscv MCU build reports in various FPGAs.

FPGA board	Arrow-SoCKit	DE0-Nano	DE2-115
FPGA chip	Cyclone V (5CSXFC6D6)	Cyclone IV (EP4CE22)	Cyclone IV (EP4CE115)
Combinational logic	2,858	4,800	4,810
Register	3,054	2,751	2,766
ALM	1,891	N/A	N/A
Memory (bit)	106,752	106,752	123,136
DSP block	4	8	8

FPGA board	DE1-SoC	DE4	TR4
FPGA chip	Cyclone V (5CSEMA5)	Stratix IV (EP4SGX230)	Stratix IV (EP4SGX230)
Combinational logic	2,832	2,874	2,863
Register	2,974	2,750	2,734
ALM	1,850	2,392	2,403
Memory (bit)	123,136	106,752	106,752
DSP block	4	8	8

3.3.3 VLSI Implementation

For small and low-power 32-bit RISC-V microprocessors, although there were plenty of IP cores presented in FPGAs as reviewed in [74] (2019), silicon-proof publications were still limited. The worth-mention 32-bit RISC-V chip measurement publications can be listed are the PULP SoC in [63] (2016), PULPv2 SoC in [67] (2017), the low-power IoT MCUs in [61] (2016) and in [64] (2017), the FE310-G000 in [118] (2017), and the FE310-G002 in [119] (2019). As a result, a further step was made for the VexRiscv MCU implementation to realize the MCU into VLSI circuits.

The SOTB-65nm process is an FD-SOI technology that has the ultra-low-power feature [120]. It can provide the chip with the back-gate biasing technique, further enhancing its performance [121]. The measurement result of the SOTB-65nm chip was reported in [93] (2020). And its completed MCU with PCB was also served as the platform for the investigation of countermeasures against power analysis attacks in [122] (2021). Figure 3.16 shows the layout of the chip, and its barechip image is given in Figure 3.17. The implemented SoC has 1,436.24-μm in width and 921.6-μm in height and occupies the whole 1.5$\times$1.0-mm^2 die as seen in Figure 3.17. According to the figure, it can be seen that the four 16-KB SRAM macros made up a total of 64-KB on-chip memory for the system. The 16-KB SRAM macro was chosen because it is the largest SRAM macro available in the SOTB-65nm process. The 8-KB of boot ROM also contains one 1-KB SRAM macro for the stack. As shown in Figure 3.17, one VexRiscv core is placed at the bottom with two caches of instruction and data that sit right next to its left and right. Each cache contained four 1-KB SRAM macros and one 512-B SRAM macro, thus 4.5-KB in total.

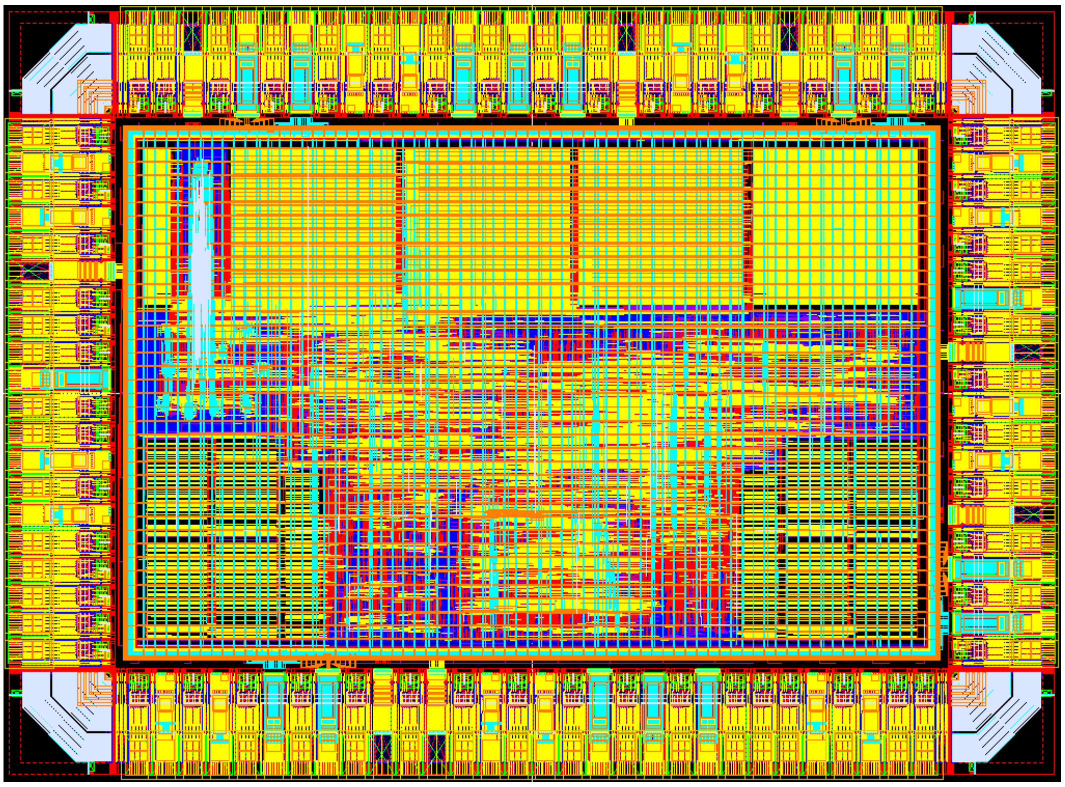


FIGURE 3.16: Layout of the SOTB-65nm VexRiscv SoC.

ROHM-180nm is a conventional bulk process that is popular and close to the industrial standard. The same version of VexRiscv MCU was also done in this technology. Figure 3.18 and Figure 3.19 show the layout and the barechip images, respectively. Compared to the last VLSI implementation, due to lack of space, the 64-KB of on-chip memory was reduced to 16-KB SRAM, as shown in the figure. The 8-KB of boot ROM was kept (as we can see that the 1-KB of SRAM stack was still there), but two caches of instruction and data were removed. As seen from the figure, the ROHM-180nm version of the VexRiscv MCU has a size of 1,933.44$\times$1,933.2-μm^2 and sits on the whole 2.5$\times$2.5-mm^2 die.

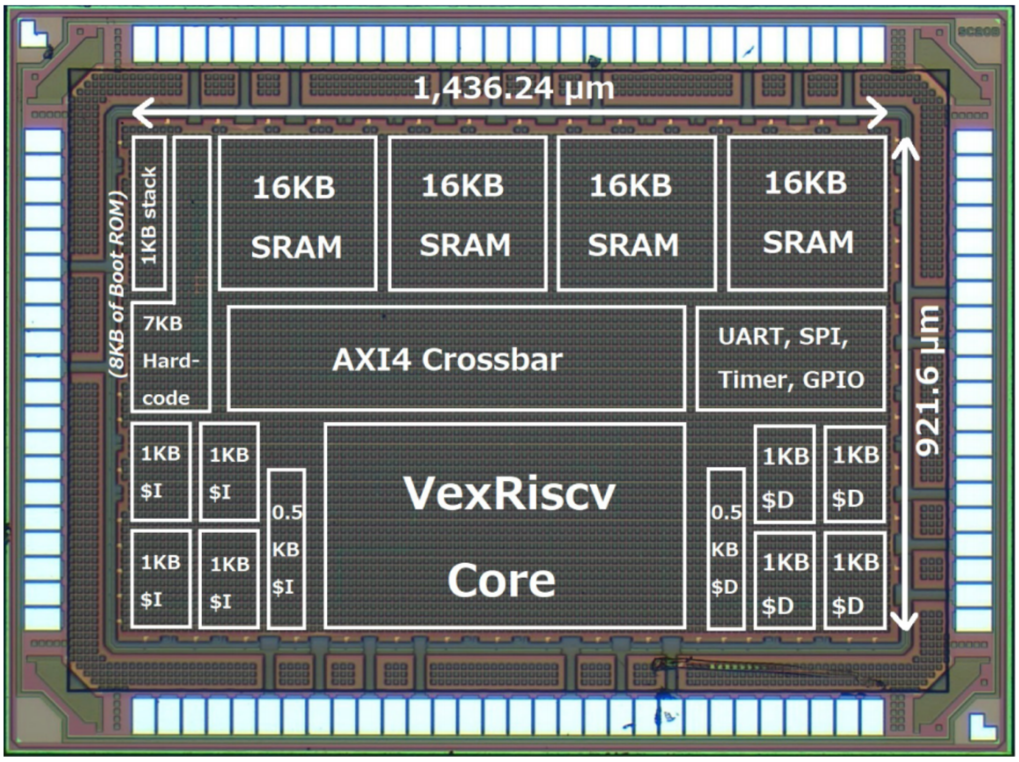


FIGURE 3.17: Barechip image of the SOTB-65nm VexRiscv SoC.

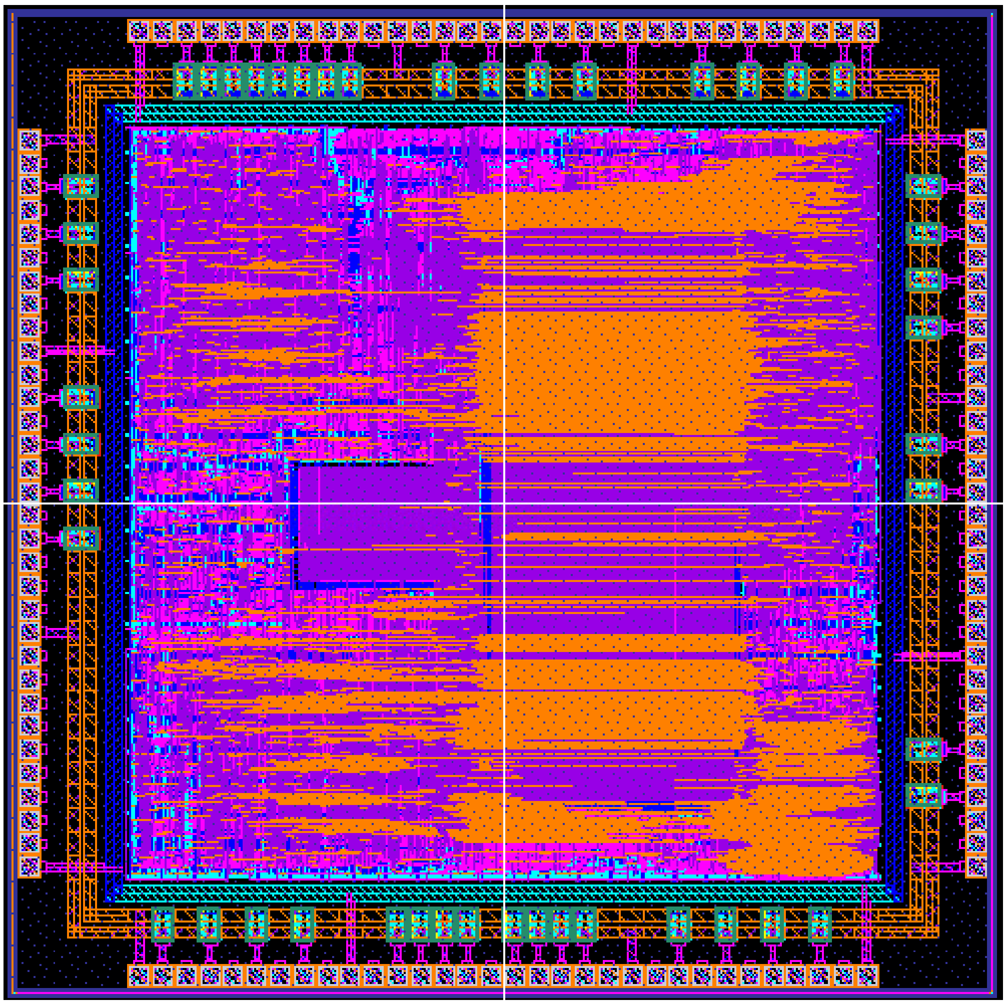


FIGURE 3.18: Layout of the ROHM-180nm VexRiscv SoC.

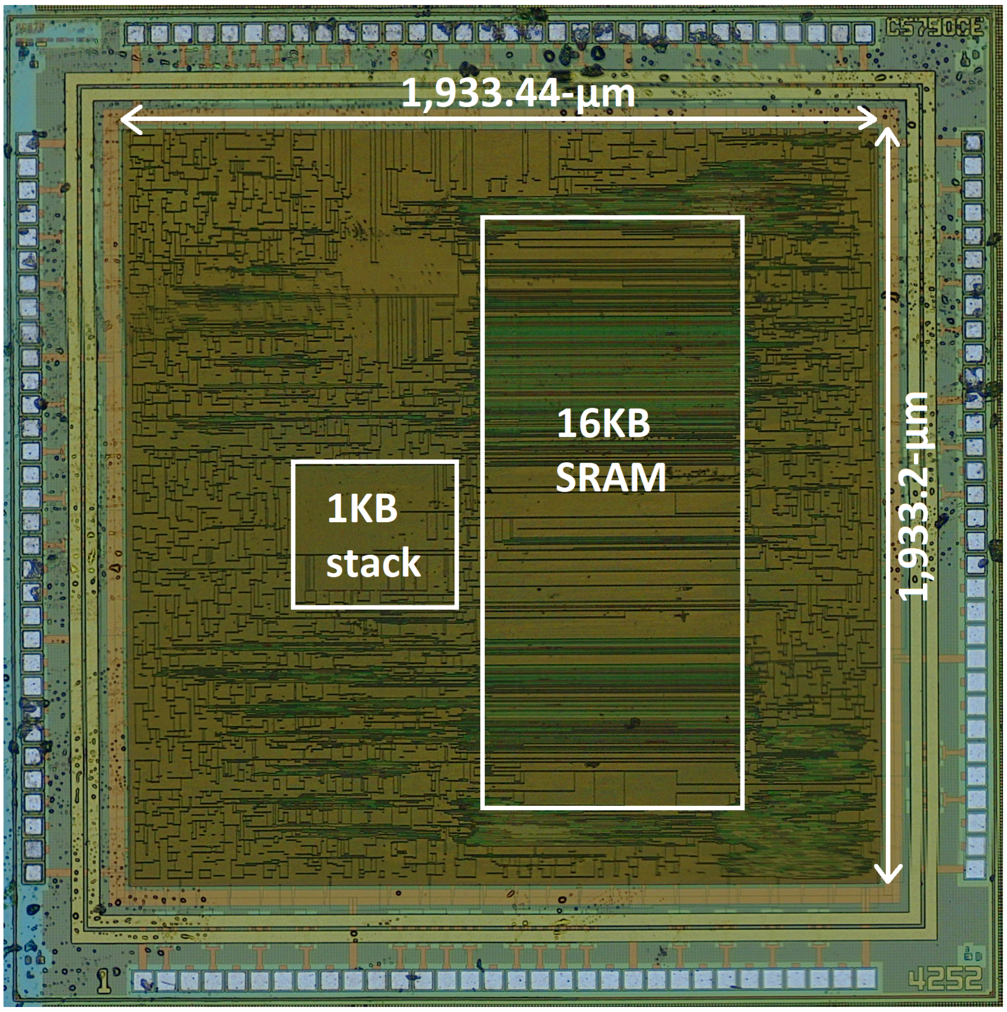


FIGURE 3.19: Barechip image of the ROHM-180nm VexRiscv SoC.

Both two chips were done by using the Synopsys tools of Design Compiler (DC) for synthesis and IC Compiler (ICC) for Place-and-Route (PnR). The results reported by the Synopsys tools were collected and given in Table 3.6 for the two VLSI implementations.

TABLE 3.6: VexRiscv MCU VLSI synthesis reports.

Process		SOTB-65nm (1.2-V @ 100-MHz)	ROHM-180nm (1.8-V @ 100-MHz)
Memory	**On-chip RAM**	64-KB	16-KB
	Inst. cache	4.5-KB	none
	Data cache	4.5-KB	none
Area	**Die**	2.0×1.5-mm^2	2.5×2.5-mm^2
	Core	1,436.24×921.6-μm^2 = 1.324-mm^2 ≈ 349,061-NAND2	1,933.44×1,933.2-μm^2 = 3.74-mm^2 ≈ 289,692-NAND2
	Cell	33,935	48,541
Power (mW)	**@Synthesis**	13.39	66.92
	@PnR	15.74	89.94
F_{Max} (MHz)	**@Synthesis**	94	2
	@PnR	88	102

3.3.4 Chip Measurement Result

Both SOTB-65nm and ROHM-180nm chips were fabricated with the 160-pin QFP packaging. Two PCB test boards were developed to test the chips. Figure 3.20 and Figure 3.21 show the testing PCBs for the SOTB-65nm and ROHM-180nm chips, respectively. Both PCBs have a built-in USB-to-UART interface, SD-card socket, JTAG header, LEDs, switches, and a programable clock circuit. The power supplies can be drawn directly from the USB interface or external power sources using power jumpers. The clock can be provided by on-board clock circuit or external sources via the SMA connector.

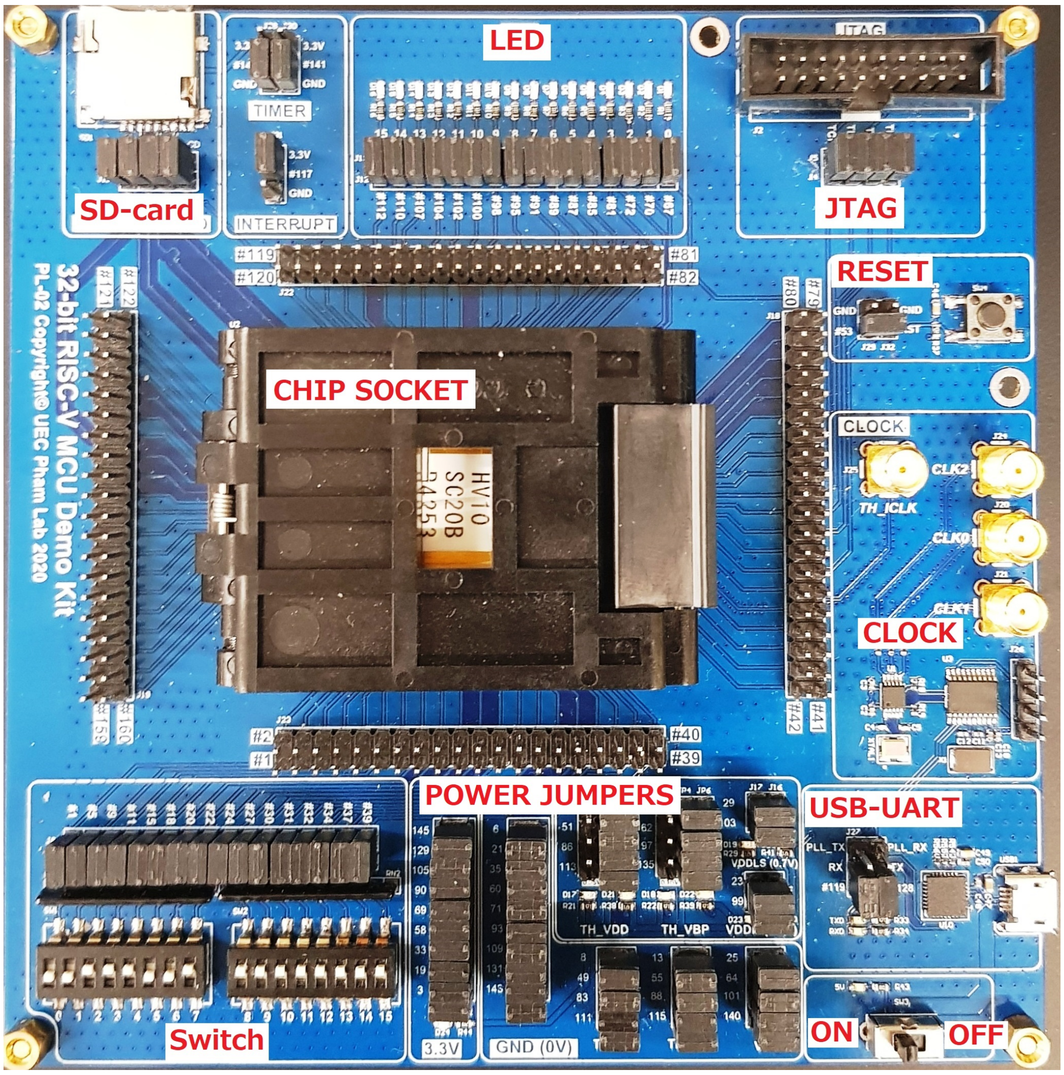


FIGURE 3.20: PCB for testing the SOTB-65nm VexRiscv MCU.

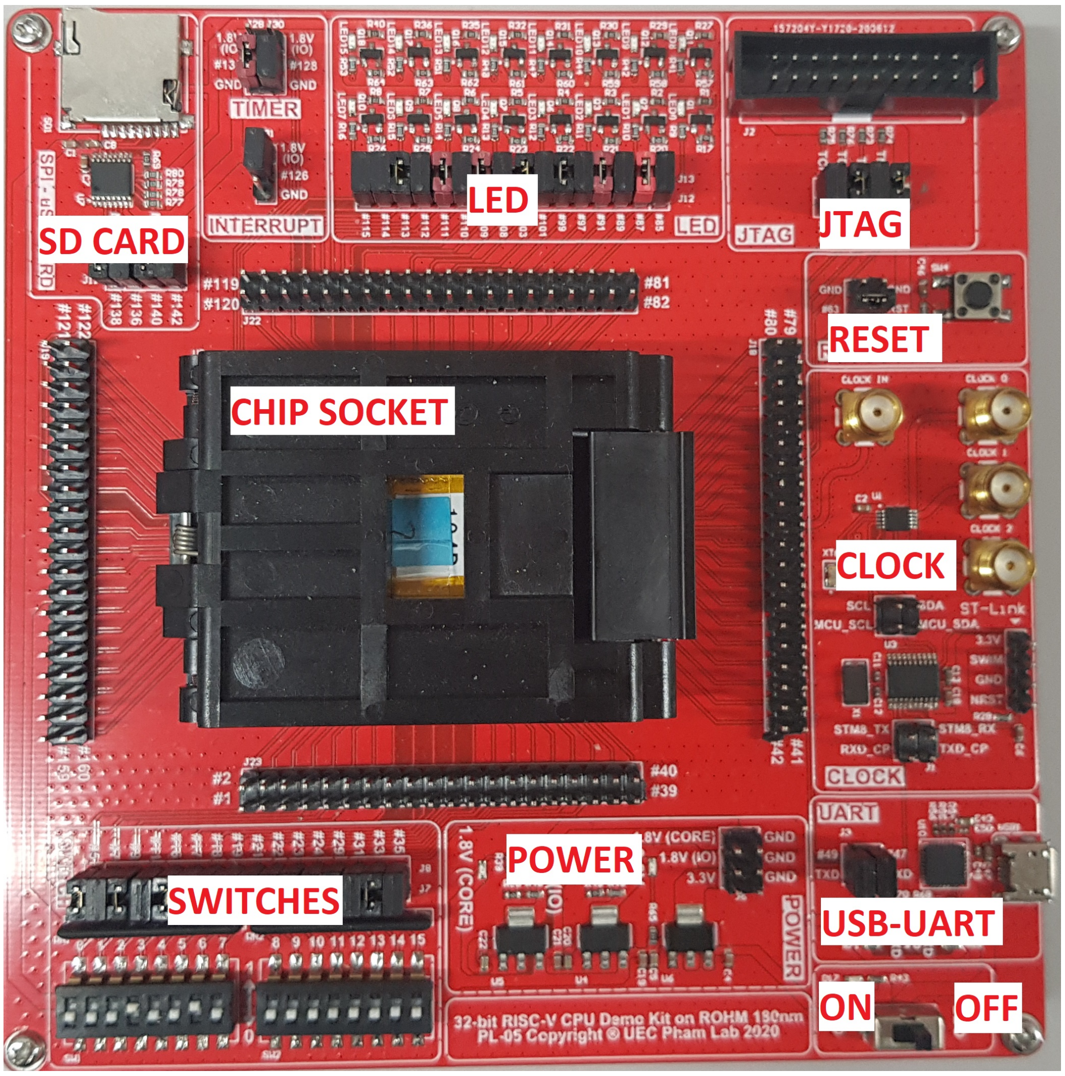

SD CARD
TIMER
INTERRUPT
LED
JTAG
RESET
CHIP SOCKET
CLOCK
SWITCHES
POWER
USB-UART
ON
OFF
32-bit RISC-V CPU Demo Kit on ROHM 180nm
PL-05 Copyright © UEC Pham Lab 2020

FIGURE 3.21: PCB for testing the ROHM-180nm VexRiscv MCU.

In SOTB-65nm technology, the default threshold voltage (V_{TH}) is about 0.4-V to 0.5V, and the recommended operating V_{DD} is 0.75-V. Therefore, the SOTB-65nm chip measurement was done in the condition of 0.4-V to 1.2V V_{DD} and -1.6-V to $+1.6$-V back-gate bias voltage (V_{BB}). Figure 3.22 shows the changes in maximum operating frequency (F_{Max}) corresponding to the power supplies (V_{DD} and V_{BB}). In general, the F_{Max} performances increased almost linear with the increment of V_{DD}. Specifically, with no bias (i.e., $V_{BB} = 0$-V), the F_{Max} values ranged from 12-MHz at 0.6-V V_{DD} to 104-MHz at 1.2-V V_{DD}; the changes were about 15-MHz per 0.1-V of V_{DD}. For V_{BB} from -1.6-V to $+0.8$-V, F_{Max} values also increased nearly linear; there was about 18-MHz F_{Max} improvement for each 0.4-V V_{BB} increment. However, when a $V_{BB} \geq$ 0.8-V was applied, the F_{Max} increment became little to none, as shown in Figure 3.22. The highest F_{Max} value of 156-MHz was achieved at 1.2-V V_{DD} with $+1.6$-V V_{BB}. At -1.6-V V_{BB}, the MCU can function only with $V_{DD} \geq$ 0.9-V.

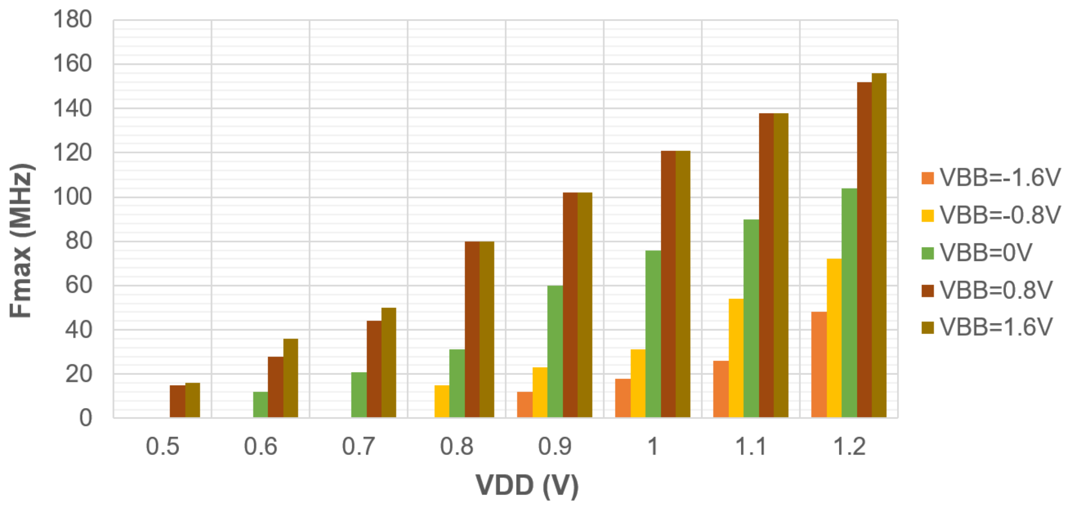


FIGURE 3.22: SOTB-65nm VexRiscv MCU: F_{Max} vs. supply voltages.

Continue with the SOTB-65nm measurement, Figure 3.23 and Figure 3.24 give the variations in power consumption (P_{Active}) and power density (P_{Active}/F_{Max}) corresponding to power supplies. Overall, the changes were almost linear with the V_{DD} increment. For no bias, the power consumption values ranged from 0.49-mW (40.8-μW/MHz) at 0.6-V V_{DD} to 17.58-mW (169.04-μW/MHz) at 1.2-V V_{DD}; the changes were about 2.85-mW (21-μW/MHz) for each 0.1-V increment of V_{DD}. At reversed back-gate bias, there were only slight reductions in consumption, as seen in the figures. In contrast, P_{Active} values increased quite a lot with forwarding back-gate bias. The lowest power consumption of 0.49-mW has been achieved at 0.6-V V_{DD} with no bias, but the best power density of 33.4-μW/MHz was at 0.5-V V_{DD} with +0.8-V V_{BB}.

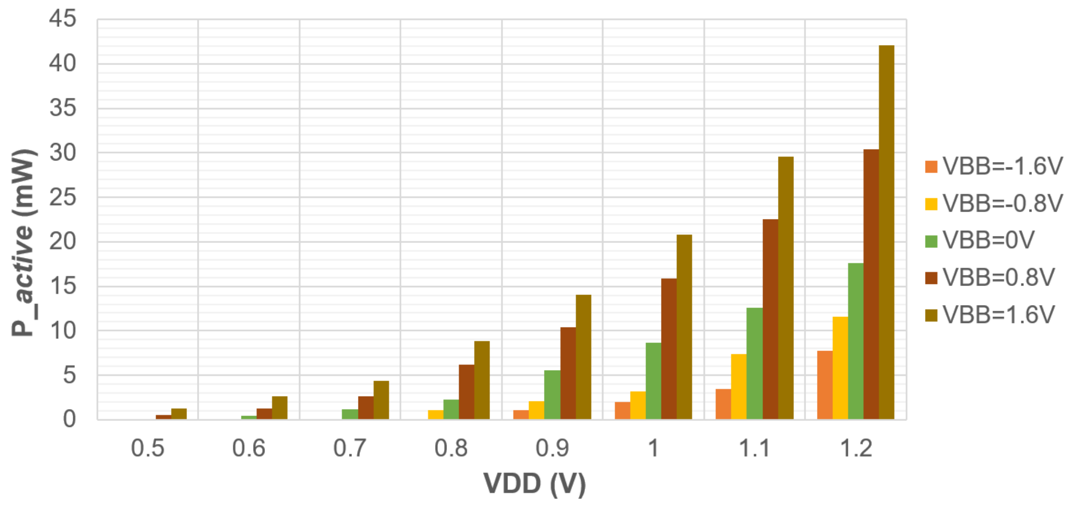


FIGURE 3.23: SOTB-65nm VexRiscv MCU: P_{Active} vs. supply voltages.

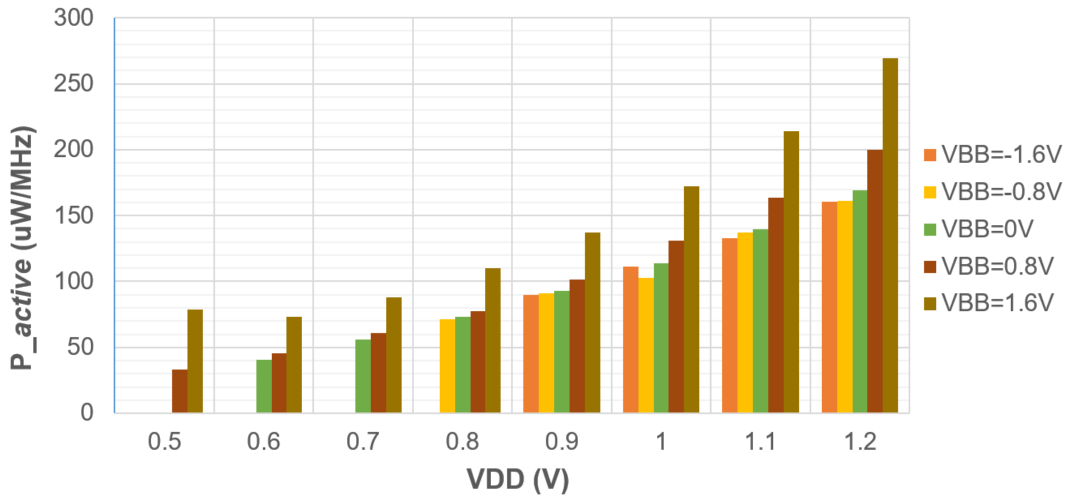


FIGURE 3.24: SOTB-65nm VexRiscv MCU: P_{Active}/F_{Max} vs. supply voltages.

Figure 3.25 shows the leakage power of the SOTB-65nm chip in the changes of power supplies. These values were measured at sleep mode when the clock was cut off. At no bias, P_{sleep} values ranged from 2.165-μW at 0.5-V V_{DD} to 30-μW at 1.2-V V_{DD}. From -0-8 to $+0.8$-V V_{BB}, the P_{sleep} values reduced roughly about one order of magnitude per 0.4-V V_{BB} reduction. However, the $V_{BB} \leq -1.6$-V lines can not result in further reduction due to the Gate-Induced Drain Leakage (GIDL) phenomenon [121]. The best leakage power was 2.43-nW with 0.5-V V_{DD} and -1.6-V V_{BB}.

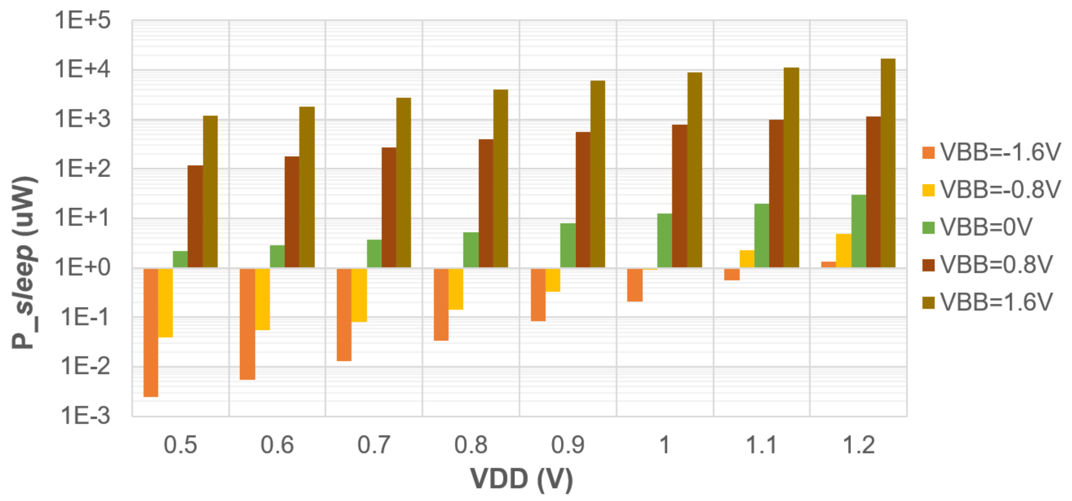


FIGURE 3.25: SOTB-65nm VexRiscv MCU: P_{Leak} vs. supply voltages.

In ROHM-180nm technology, the default V_{TH} is about 1.0-V and the recommended operating V_{DD} is 1.8-V. Therefore, the ROHM-180nm chip measurement was done with the V_{DD} range of 1.0-V to 2.0-V. Figure 3.26 shows the changes in F_{Max} with V_{DD}. We can see that the F_{Max} performances increased almost linear with V_{DD} increment, about 8-MHz per 0.1-V of V_{DD} increment. The lowest and highest F_{Max} were 35-MHz and 99-MHz achieved at 1.2-V and 2.0-V V_{DD}, respectively. The MCU failed to function with $V_{DD} < 1.2$-V.

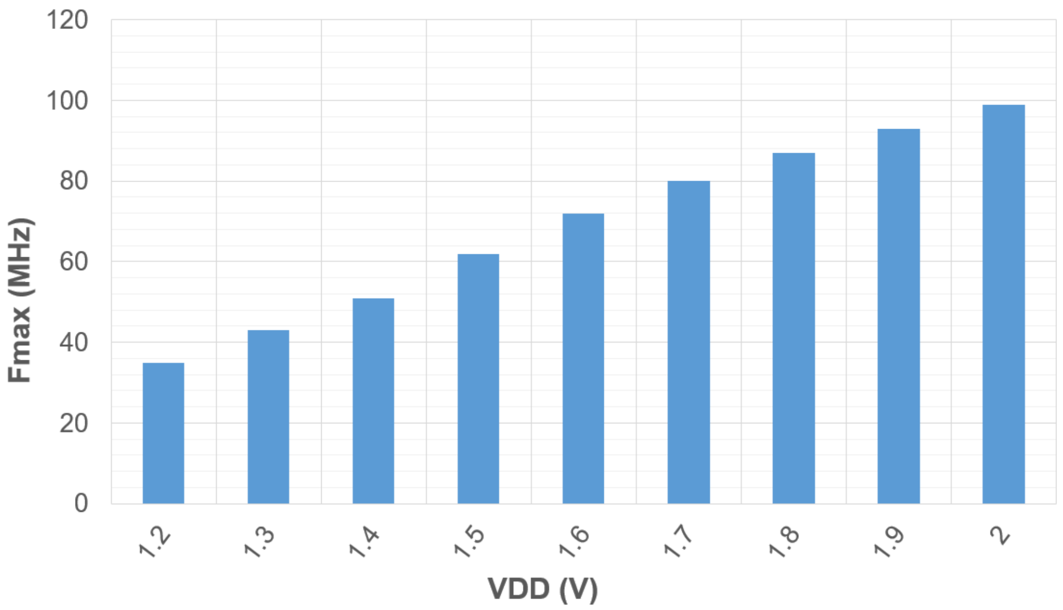


FIGURE 3.26: ROHM-180nm VexRiscv MCU: F_{Max} vs. supply voltages.

Figure 3.27 and Figure 3.28 show the P_{Active} and $P_{\text{Active}}/F_{\text{Max}}$ of the ROHM-180nm chip, respectively. In the figures, the idle-state is when the chip was held at reset, and the active-state is when it was continuously running the Dhrystone test. In general, their changes were almost linear, with roughly about 15.44-mW (113-μW/MHz) and 12.42-mW (90.8-μW/MHz) per 0.1-V V_{DD} for the active-state and idle-state, respectively. The lowest and highest P_{Active} were 18.61-mW (531.77-μW/MHz) and 142.14-mW (1,435.76-μW/MHz) at 1.2-V and 2.0-V of V_{DD}, respectively. The lowest and highest P_{Idle} were 15.01-mW (428.91-μW/MHz) and 114.38-mW (1,155.35-μW/MHz) at 1.2-V and 2.0-V of V_{DD}, respectively.

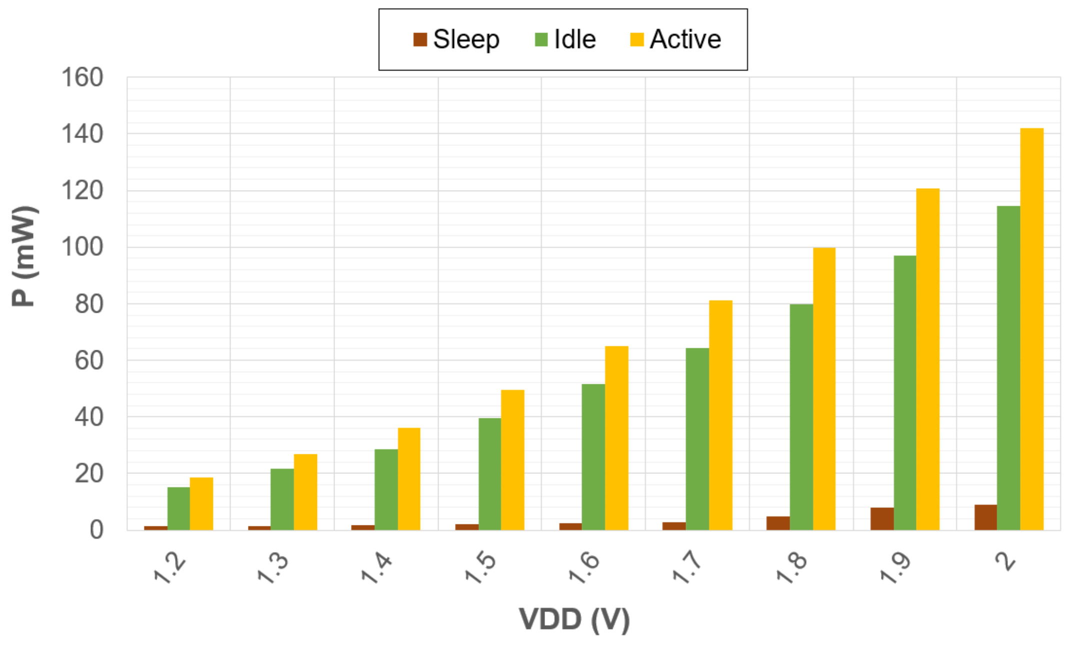


FIGURE 3.27: ROHM-180nm VexRiscv MCU: P_{Active} vs. supply voltages.

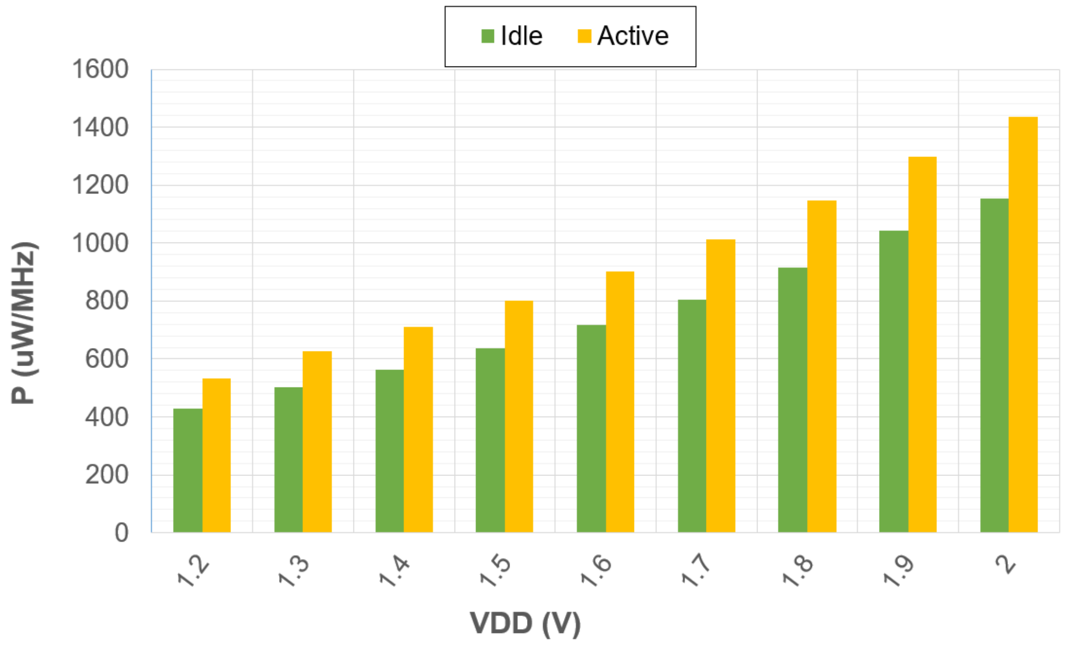


FIGURE 3.28: ROHM-180nm VexRiscv MCU: P_{Active}/F_{Max} vs. supply voltages.

Figure 3.29 shows the leakage power of the ROHM-180nm chip in the changes with V_{DD}. The sleep-state was measured when the clock was cut off. From the figure, we can see a "break-point" at V_{DD} = 1.7-V. This is because when the V_{DD}-core $\geq$ V_{DD}-I/O (the V_{DD}-I/O is fixed at 1.8-V), currents from the I/Os were drawn back to the core, thus increasing the P_{Sleep} of the core. In the range of V_{DD} = 1.2-V to 1.7-V, the changes were about 0.29-mW per 0.1-V V_{DD}. And in the range of V_{DD} = 1.7-V to 2.0-V, the changes were about

1.55-mW per 0.1-V V_{DD}. The leakage power consumption at 1.2-V, 1.7-V, and 2.0-V were 1.3-mW, 2.74-mW, and 8.92-mW, respectively.

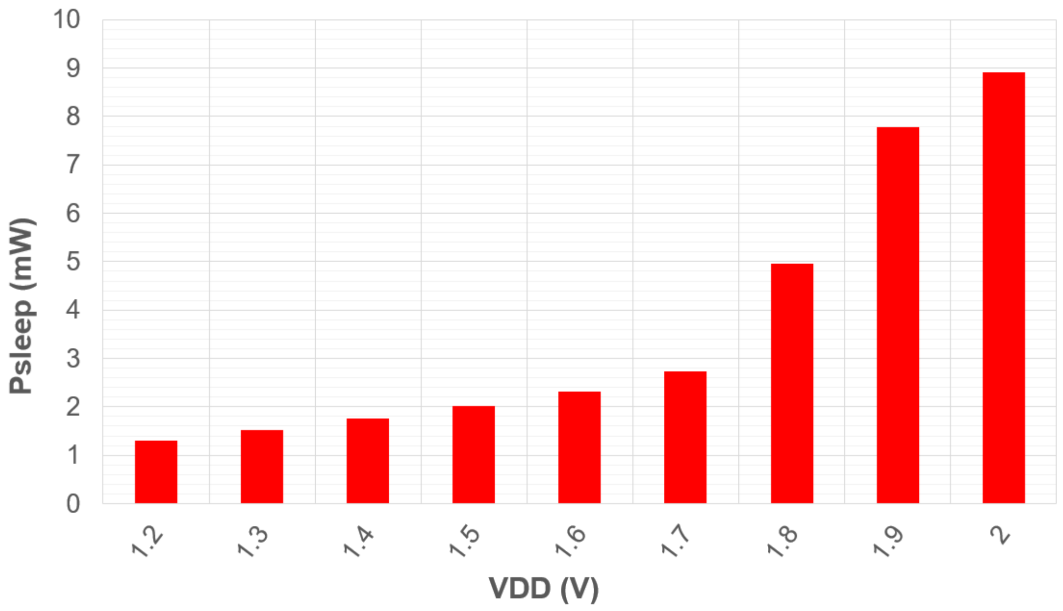


FIGURE 3.29: ROHM-180nm VexRiscv MCU: P_{Leak} vs. supply voltages.

TABLE 3.7: VexRiscv MCU chips' features summary.

Process		SOTB-65nm	ROHM-180nm
Memory	**On-chip**	64-KB	16-KB
	Inst. cache	4.5-KB	none
	Data cache	4.5-KB	none
Area	**Die**	2.0×1.5-mm^2	2.5×2.5-mm^2
	Core	$1{,}436.24\times921.6$-μm^2 = 1.324-mm^2 $\approx$ 349,061-NAND2	$1{,}933.44\times1{,}933.2$-$\mu$m^2 = 3.74-mm^2 $\approx$ 289,692-NAND2
	Cell	33,935	48,541
V_{DD}	**I/O**	3.3-V	1.8-V
	Core	0.5-V to 1.2-V	1.2-V to 2.0-V
Benchmarks		1.27-DMIPS/MHz 2.4-Coremark/MHz	1.02-DMIPS/MHz 1.9-Coremark/MHz
Peak performance		at 1.2-V V_{DD} & +1.6-V V_{BB} F_{Max}=156-MHz P_{Active}=269.54-μW/MHz	at 2.0-V V_{DD} F_{Max}=99-MHz P_{Active}=1,435.76-μW/MHz
Best power density		at 0.5-V V_{DD} & +0.8-V V_{BB} F_{Max}=15-MHz P_{Active}=33.4-μW/MHz	at 1.2-V V_{DD} F_{Max}=35-MHz P_{Active}=531.77-μW/MHz

Key features of the two chips are summarized in Table 3.7. The completed SoCs were also benchmarked with the Dhrystone and Coremark tests. The

SOTB-65nm chip achieved 1.27-DMIPS/MHz and 2.4-Coremark/MHz, while the results of the ROHM-180nm chip were 1.02-DMIP/MHz and 1.9-Coremark/MHz. Due to the decisive advantage of the back-gate biasing, the SOTB-65nm chip got overwhelming better metrics than its ROHM-180nm equivalent in both peak performance and best power density, as shown in Table 3.7.

3.3.5 Comparison and Discussion

Table 3.8 gives the results of the two chips in comparison with other recent 32-bit RISC-V MCUs. To provide a better point-of-view, the results of PULPv2 [67], Duran *et al.* [64], and the ROHM-180nm were scaled to the equivalent results of a 65-nm node by using the equations from [123]. It is noted that because the equations in [123] did not have the parameters for the 28-nm process, the scaled values of PULPv2 [67] were calculated by using the settings of the 32-nm process instead.

Although the F_{Max} of the VexRiscv MCUs are the lowest value in the table, the comparison may not reflect the true nature of the architecture. The reason is that for those designs without integrated Phase-Locked Loop (PLL) or Frequency-Locked Loop (FLL), the operating frequencies heavily depended on the I/O circuits. For example, the chip in [67] had integrated FLL while the others had not. Therefore, the operating frequency in [67] could quickly go higher than 500-MHz, while the other works were limited by the general digital I/Os, as seen in the table.

TABLE 3.8: Comparison between the VexRiscv chips with other 32-bit RISC-V MCUs.

Design	Duran *et al.* (2017) [64]	PULPv2 (2017) [67]	This work	
ISA	RV32IM	RV32IMC	RV32IM	
Number of cores	1	4	1	
Core V_{DD} (V)	1.2	0.32 to 1.15	0.5 to 1.2	1.2 to 2.0
Process	130-nm	28-nm	65-nm	180-nm
F_{Max} (MHz)	160	825	156	99
P_{Active} (μW/MHz)	167	20.7	33.4	531.77
Leakage power (mW)	N/A	0.37*	0.4*	4.97**
Scaled to 65-nm by using equations [123]				
Process	65-nm			
F_{Max} (MHz)	304.88	388.68	156	235.63
P_{Active} (μW/MHz)	23.05	126.6	33.4	31.57
Leakage power (mW)	N/A	2.22*	0.4*	0.32**

* measured at 0.6-V V_{DD} & no bias.
** measured at 1.8-V V_{DD} & no bias.

For the power consumption of P_{Active}, the VexRiscv results were measured while running the Dhrystone test, while the result in [64] was measured while running three while loops. Therefore, if the MCU in [64] was running the Dhrystone test when being measured, the value of 23.05-μW/MHz should be a bit higher. For the result of [67], it can be argued that if with a single-core processor, its power consumption will be much less. Hence, the power density of a single-core PULPv2 can be roughly approximated by $126.6/4 = 31.65$-μW/MHz, close to the value of 33.4-μW/MHz and 31.57-μW/MHz of the VexRiscv MCUs.

For leakage power comparison, the PULPv2 chip [67] reported 0.37-mW while running at 0.6-V V_{DD} with no bias (i.e., V_{BB} = 0-V). Scaling to the equivalent result of the 65-nm node, 0.37-mW became 2.22-mW. With a similar argument about single-core versus multi-core, the leakage power of a single-core could be roughly estimated to be about $2.22/4 = 0.555$-mW. Thus, the 0.4-mW and 0.32-mW results of the VexRiscv MCUs still yield the best performances in the table. It is also noted that the best values of leakage powers in this work were not brought to the comparison table because the results with reserved back-gate bias voltages were not reported in those papers [64, 67].

To conclude, a truly fair comparison between implementations was hard to achieve due to the complex nature of computer architecture. Table 3.8 has already brought a proper perspective for the comparison, but it may not ultimately reflect all of the pros and cons of all implementations. However, it can be said that the implemented chips have achieved the average performances of F_{Max} and P_{Active} with a genuinely good leakage power consumption. With the powerful tool of back-gate biasing, the SOTB-65nm chip can be used for a wide range of embedded applications in both means of low-power and high-performance settings.

3.4 Rocket-chip Computer System

3.4.1 The Architecture

In this section, the Linux-bootable Rocket-chip computer system is investigated. The system originally came from the SiFive-Freedom repository [124]. The Rocket-chip is a highly customizable core processor with a 5-stage in-of-order architecture. Its default ISA is RV64IMAFDC (also called RV64GC); that means 64-bit RISC-V with **I**nteger, **M**ultiplication, **A**tomic, **F**loating-point, **D**ouble floating-point, and **C**ompressed instruction sets. However, own to the fact that it is highly customizable [40], any sub-set in the RV64IMAFDC can be turned on or off, including the 64-bit/32-bit addressing.

Studying this Rocket-chip generator and implementing its computer system set a foundation for future developments in the following chapters. Figure 3.30 shows the original system architecture demonstrated on various FPGA boards given by the site [124]. Following the tutorial on the site, a new version was slightly modified to better fit with VLSI implementation, and Figure 3.31 gives its architecture. As seen in Figure 3.31, the PCIe connection was removed for simplicity. About the DDR memory, because we can not include the DDR IP into our chip, the memory bus was exported out to the chip's I/O. The DDR controller with its TileLink-to-AXI4 bridge will be kept in the FPGA, while the rest will become an SoC and go to the chip. When the chip is done, it will be connected back to an FPGA and use the FPGA's DDR IP for its memory. Tutorial for replicating this architecture is published in the given repository [125]. The Keystone boot flow was followed to boot the system into Linux; its publication is provided in [7], and its tutorial is given in [102].

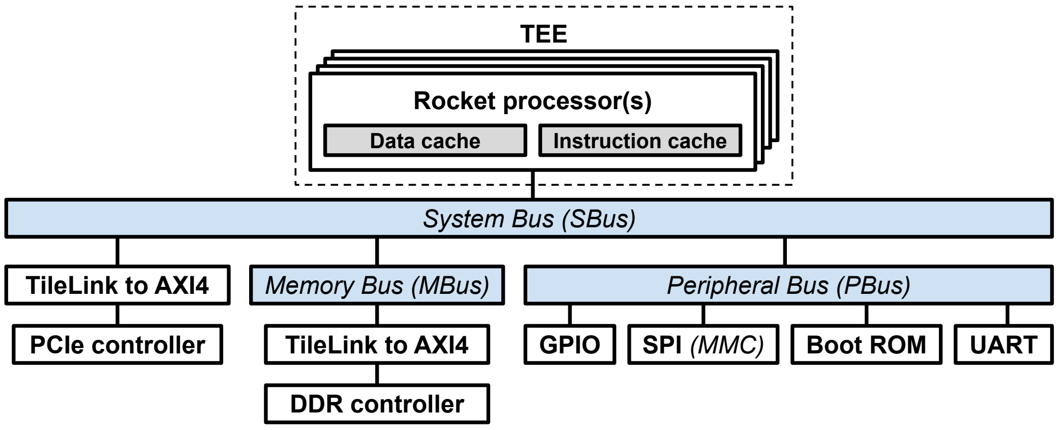


FIGURE 3.30: The Rocket-chip computer system, original architecture.

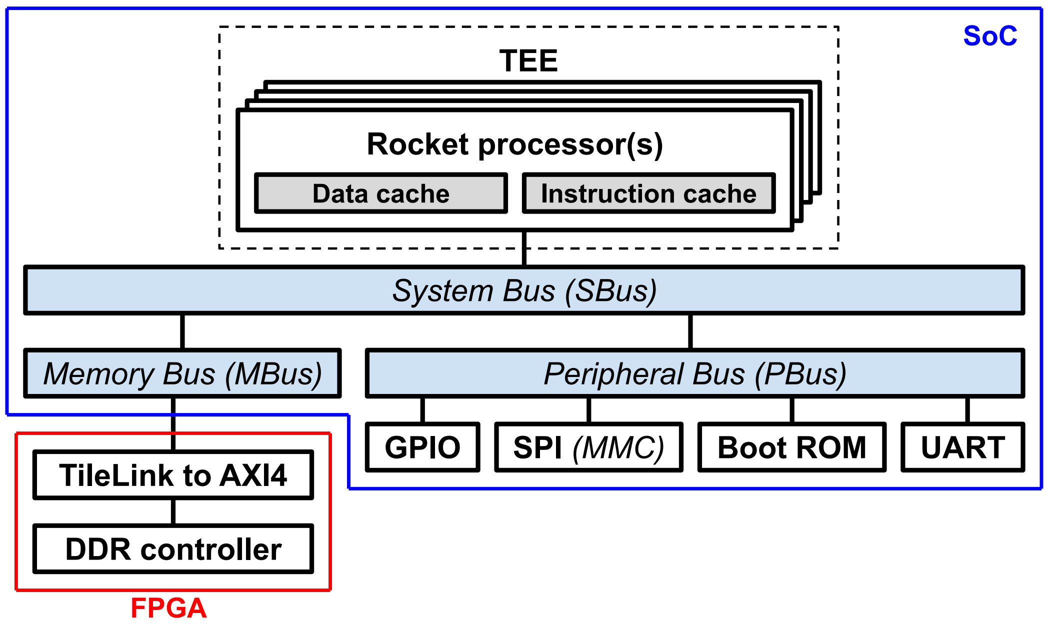


FIGURE 3.31: The Rocket-chip computer system, modified for SoC implementation.

3.4.2 FPGA and VLSI Implementation

The FPGA implementation followed the original architecture in Figure 3.30, with PCIe and DDR controllers included. The tested FPGA was the Altera TR4 board; the tutorial for the build can be found on the given site [125]. The FPGA build report is shown in Table 3.9. To be consistent with the VLSI implementation, the results in the table omitted the PCIe and DDR controller IPs resources.

TABLE 3.9: 64-bit quad-core Rocket-computer build report in Altera TR4 FPGA.

FPGA	TR4: Stratix IV (EP4SGX230)
Processor	Rocket-chip $\times 4$
Cache	Inst.: 16-KB; Data: 16-KB
ISA	RV64IMAFDC
Combinational logic	106,702
Register	68,295
ALM	88,475
Memory (bit)	202,422
DSP block	86

For VLSI implementation, one ROHM-180nm chip was made with the architecture given in Figure 3.31. Figure 3.32 shows the layout of the chip, and its barechip image is provided in Figure 3.33. This chip is a quad-core Rocket-chip computer system implemented in a whole 5.0×7.5-mm^2 die. The Synopsys tools of DC and ICC were also used to make this chip. Table 3.10 gives the results reported by the synthesis tools.

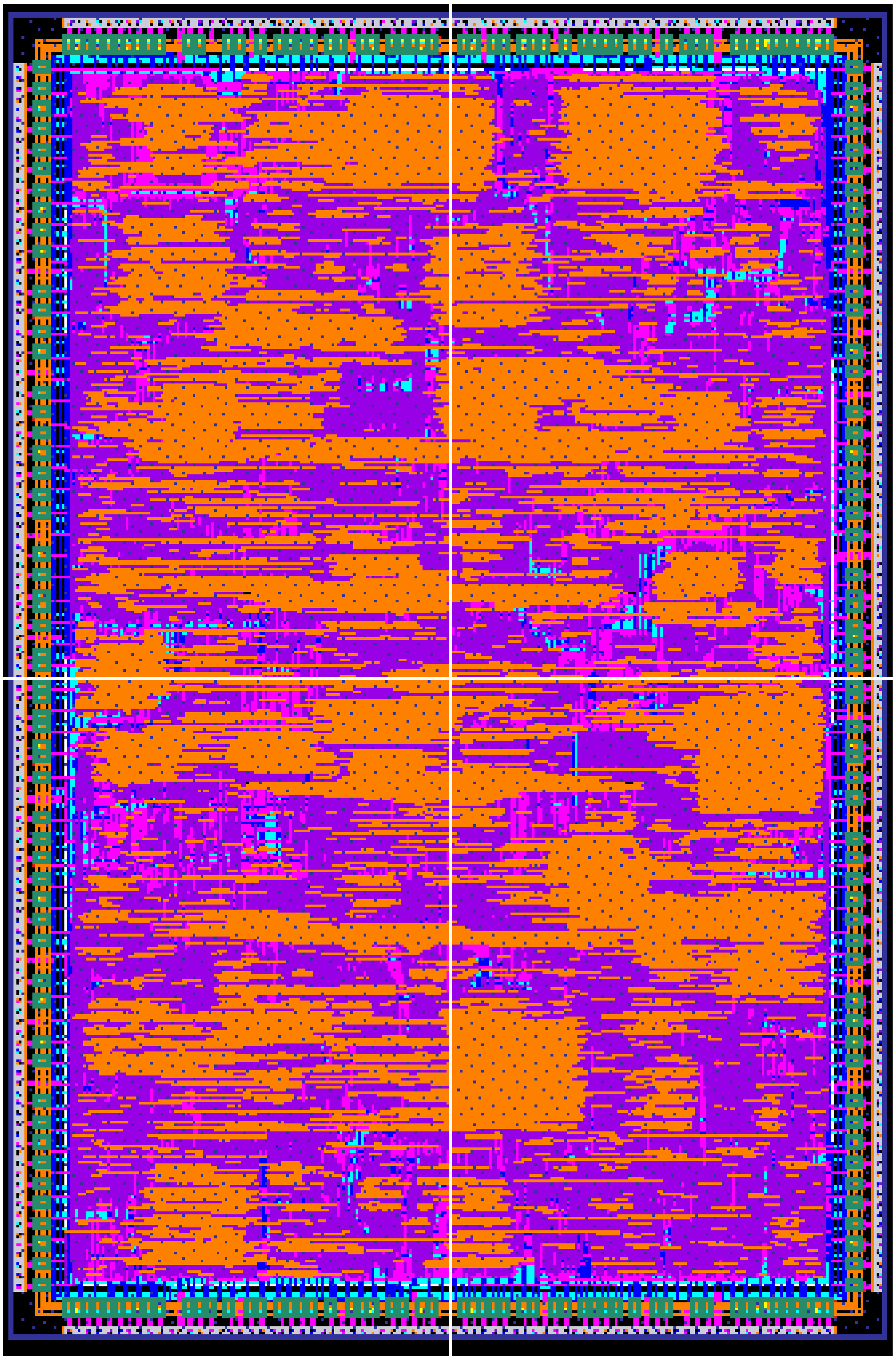


FIGURE 3.32: Layout of the ROHM-180nm 64-bit quad-core Rocket-chip (5.0×7.5-mm^2).

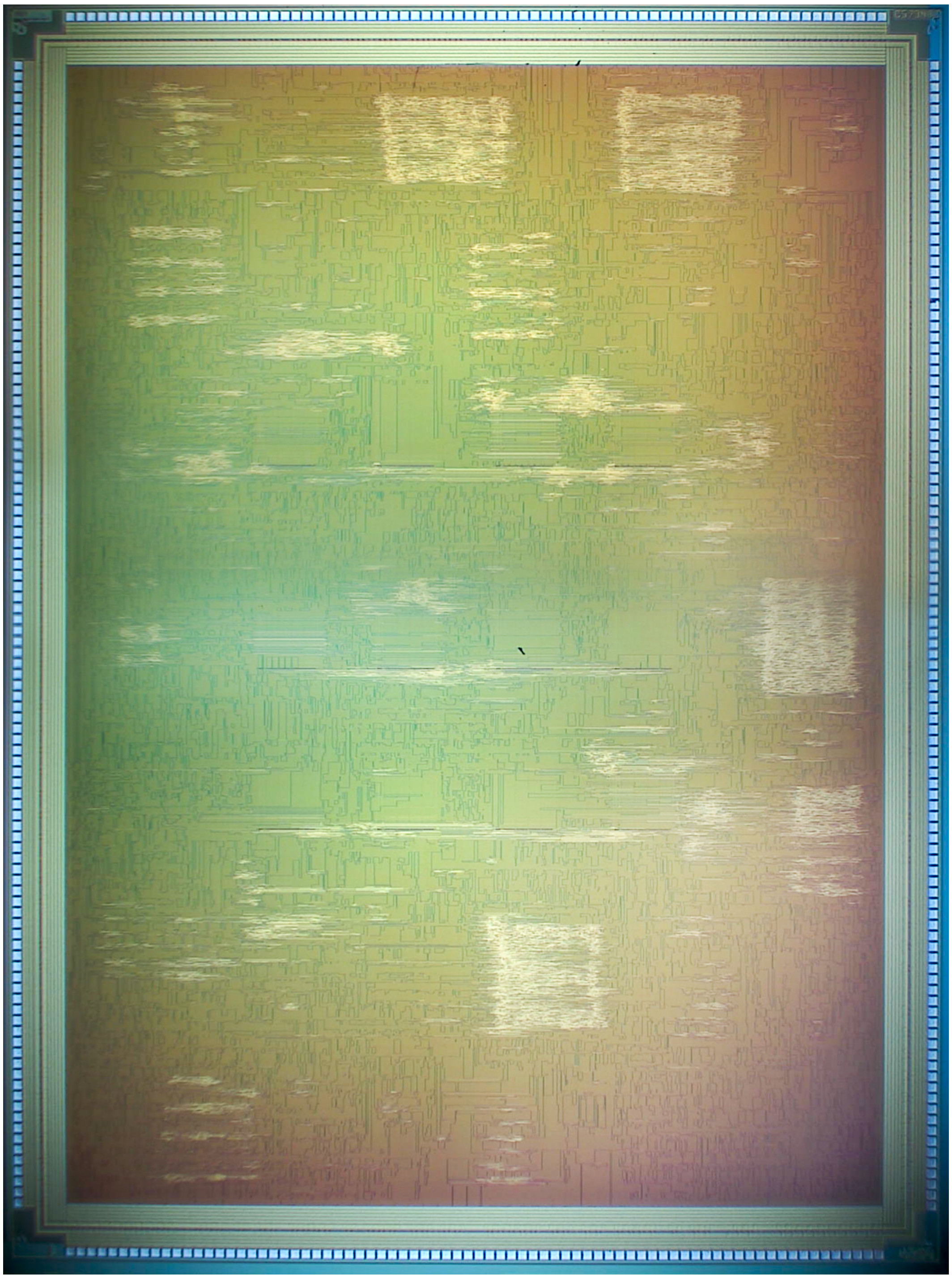


FIGURE 3.33: Barechip image of the ROHM-180nm 64-bit quad-core Rocket-chip (5.0×7.5-mm^2).

TABLE 3.10: Quad-core Rocket-chip computer system synthesis reports.

Process		ROHM-180nm (1.8-V @ 100-MHz)
Processor		Rocket-chip $\times 4$
Cache		Inst: 16-KB; Data: 16-KB
ISA		RV64IMAFDC
Area	**Die**	5.0×7.5-mm^2
	Core	$4{,}512\times 7{,}172$-μm^2 = 32.36-mm^2 $\approx$ 1,343,800-NAND2
	Cell	502,821
Power (mW)	**@Synthesis**	301.18
	@PnR	391.13
F_{Max} (MHz)	**@Synthesis**	91
	@PnR	92

3.4.3 Chip Measurement Result

One ROHM-180nm chip was made with the architecture given in Figure 3.31. The chip was fabricated with the 257-pin PGA packaging. A PCB test board was also developed to test the chips. Figure 3.34 shows the testing PCB with a built-in USB-to-UART interface, SD-card socket, JTAG header, LEDs, switches, and a programable clock circuit. For the power supply, the chip can draw the power from either the USB interface, external sources via power jumpers, or from the FPGA directly. The clock also has three options: an on-board clock circuit, external sources via the SMA connector, or the FPGA directly. Figure 3.35 shows the working PCB with a chip inside mounting on the TR4 FPGA board to use the FPGA's DIMM RAM.

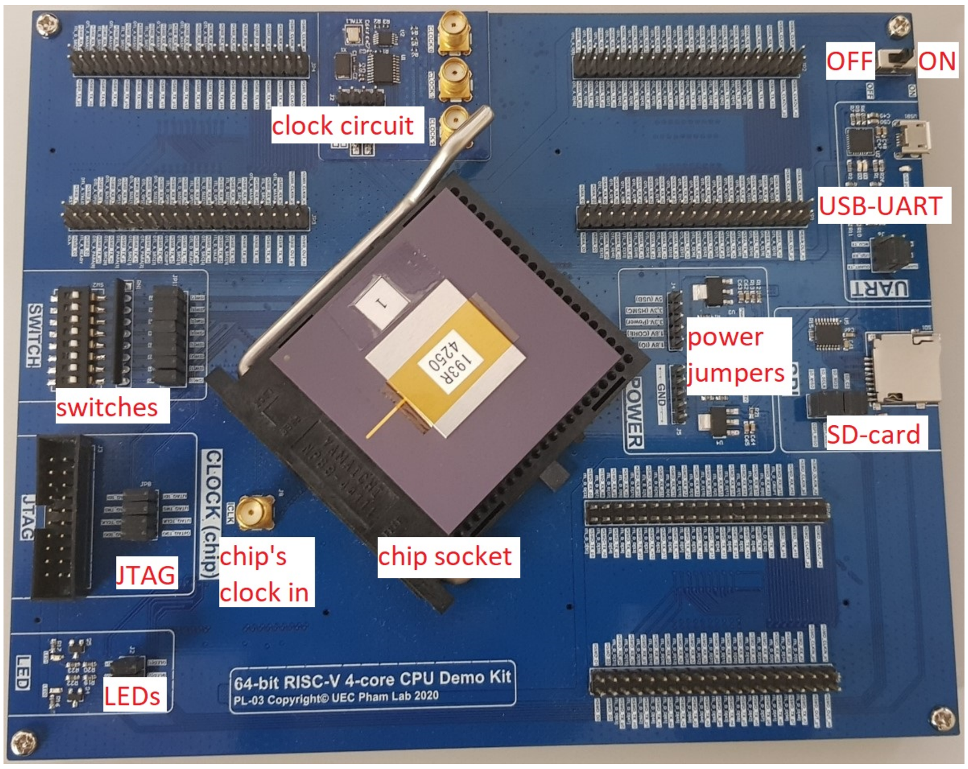


FIGURE 3.34: PCB for testing the 64-bit quad-core Rocket-computer ROHM-180nm chip.

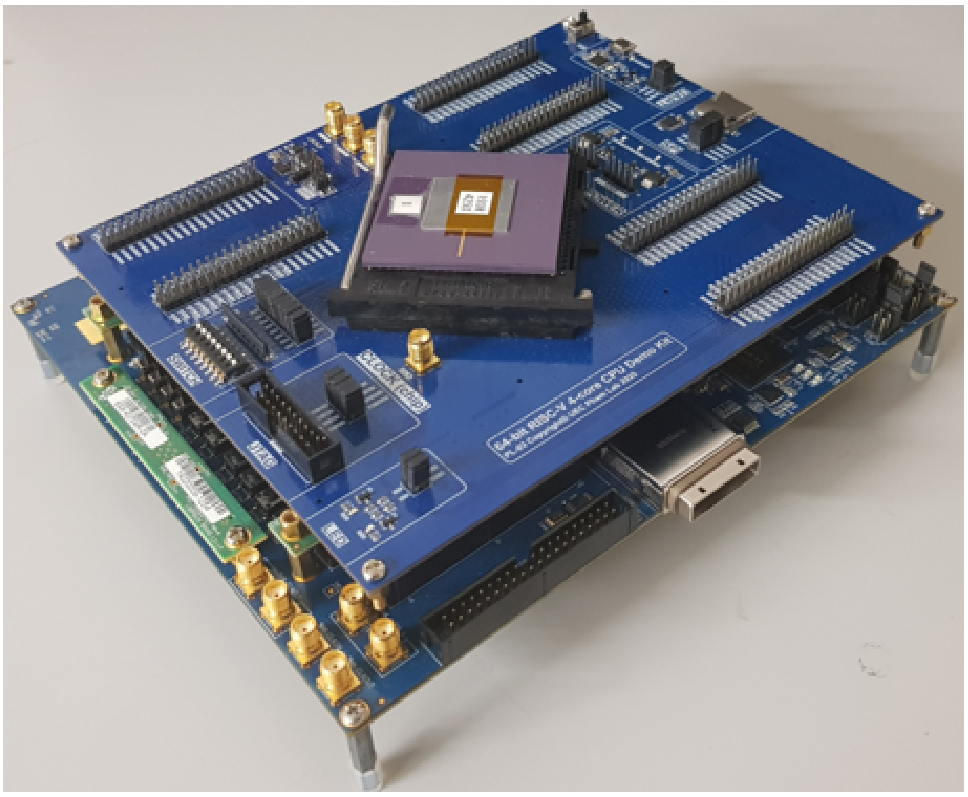


FIGURE 3.35: The quad-core Rocket-computer PCB mounts on the TR4 FPGA board.

In ROHM-180nm technology, the default V_{TH} is about 1.0-V and the recommended operating V_{DD} is 1.8-V. Therefore, the ROHM-180nm chip measurement was done with the V_{DD} range of 1.0-V to 2.0-V. Figure 3.36 shows the changes in F_{Max} with V_{DD}. We can see that the F_{Max} performances increased almost linear with V_{DD} increment, about 1.5-MHz per 0.1-V of V_{DD} increment. The lowest and highest F_{Max} were 23-MHz and 35-MHz achieved at 1.2-V and 2.0-V V_{DD}, respectively.

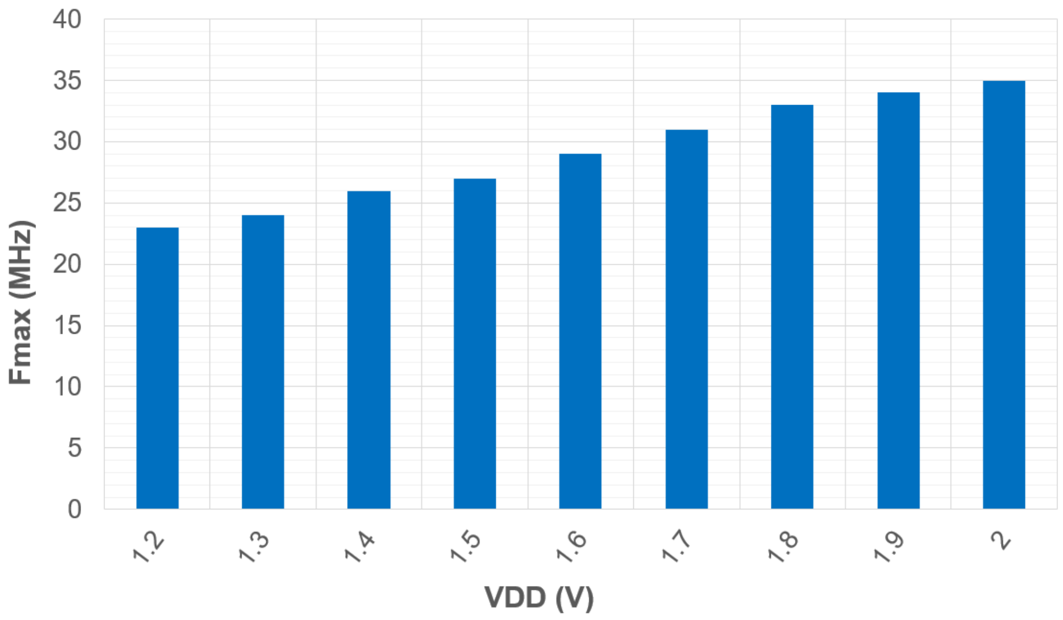


FIGURE 3.36: ROHM-180nm 64-bit quad-core Rocket-computer: F_{Max} vs. V_{DD}.

Figure 3.37 and Figure 3.38 show the P_{Active} and P_{Active}/F_{Max} of the chip, respectively. In the figures, the idle-state is when the chip was held at reset, and the active-state is when it was running the Zero Stage BootLoader (ZSBL) program. In general, their changes were almost linear, with roughly about 28.88-mW (0.69-mW/MHz) per 0.1-V V_{DD}. The lowest and highest P_{Active} were 74.18-mW (3.23-mW/MHz) and 305.18-mW (8.72-mW/MHz) at 1.2-V and 2.0-V of V_{DD}, respectively.

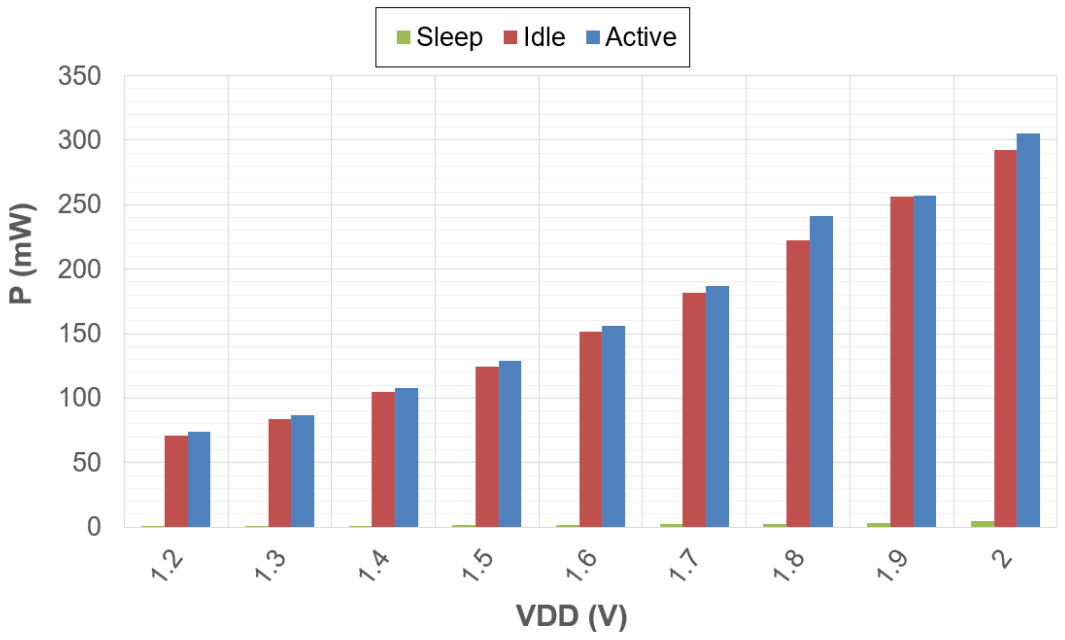


FIGURE 3.37: ROHM-180nm 64-bit quad-core Rocket-computer: P_{Active} vs. V_{DD}.

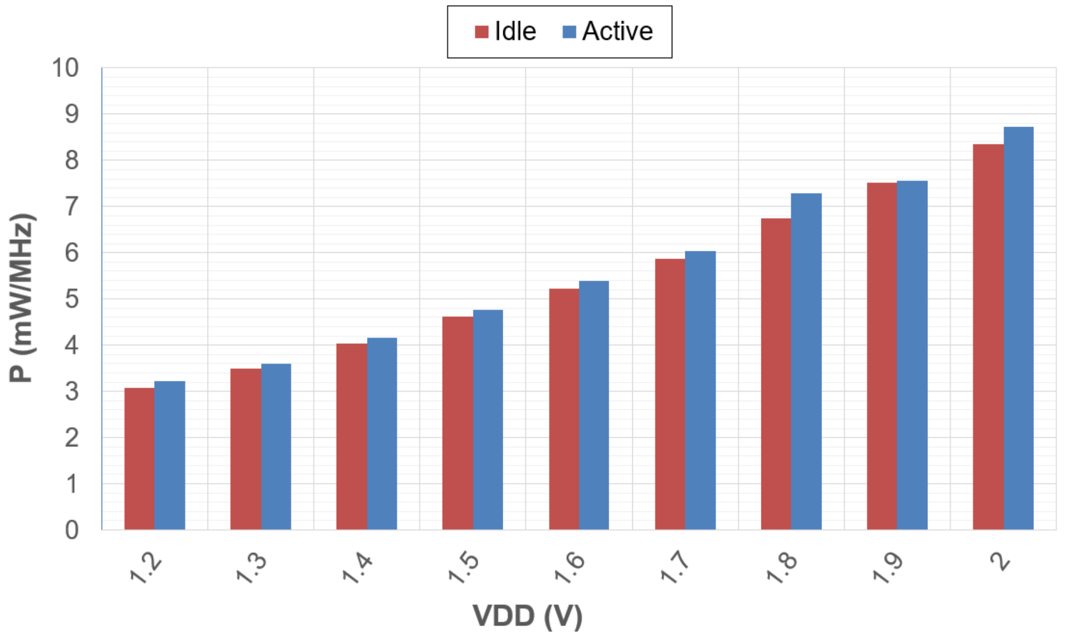


FIGURE 3.38: ROHM-180nm 64-bit quad-core Rocket-computer: P_{Active}/F_{Max} vs. V_{DD}.

Figure 3.39 shows the leakage power of the chip in the changes with V_{DD}. The sleep-state was measured when the clock was cut off. From the figure, there was a nice curve of increment. The changes could be seen roughly as linear, with about 0.47-mW per 0.1-V V_{DD}. The lowest and highest leakage power consumption were 0.55-mW and 4.29-mW at 1.2-V and 2.0-V V_{DD}, respectively. Finally, the key features of the quad-core Rocket-computer ROHM-180nm chip are summarized in Table 3.11.

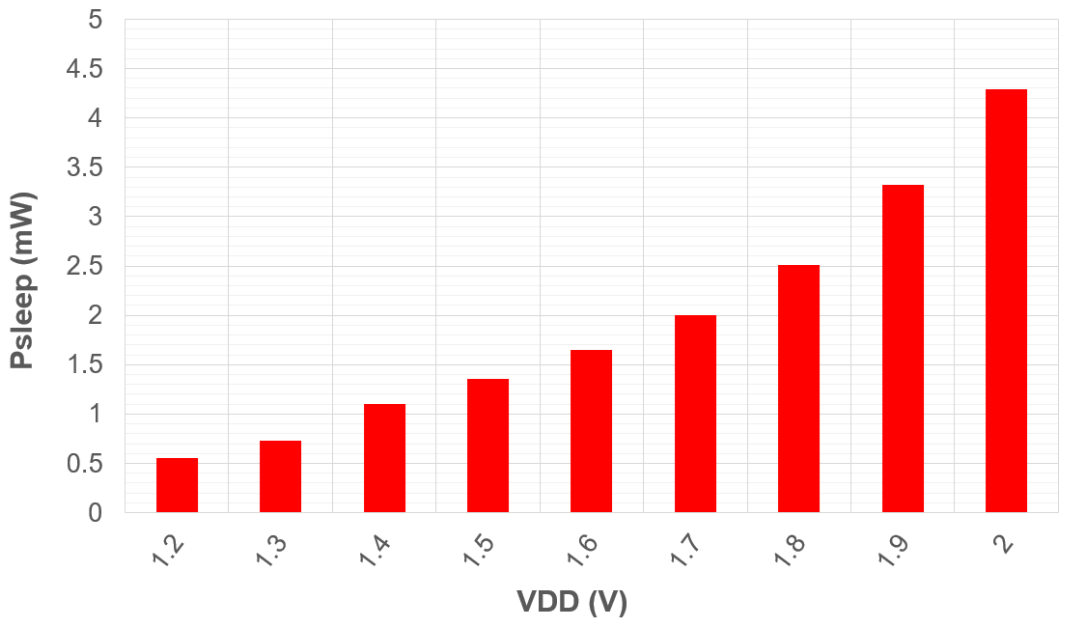


FIGURE 3.39: ROHM-180nm 64-bit quad-core Rocket-computer: P_{Leak} vs. V_{DD}.

TABLE 3.11: 64-bit Quad-core Rocket-computer ROHM-180nm chip's features summary.

Process		ROHM-180nm
Processor		Rocket-chip $\times 4$
Cache		Inst: 16-KB; Data: 16-KB
ISA		RV64IMAFDC
Area	**Die**	5.0×7.5-mm^2
	Core	$4{,}512\times7{,}172$-μm^2 = 32.36-mm^2 $\approx$ 1,343,800-NAND2
	Cell	502,821
$\mathbf{V_{DD}}$	**I/O**	1.8-V
	Core	1.2-V to 2.0-V
Peak performance		at 2.0-V V_{DD} F_{Max}=35-MHz P_{Active}=8.72-mW/MHz
Best power density		at 1.2-V V_{DD} F_{Max}=23-MHz P_{Active}=3.23-mW/MHz

3.5 Summary

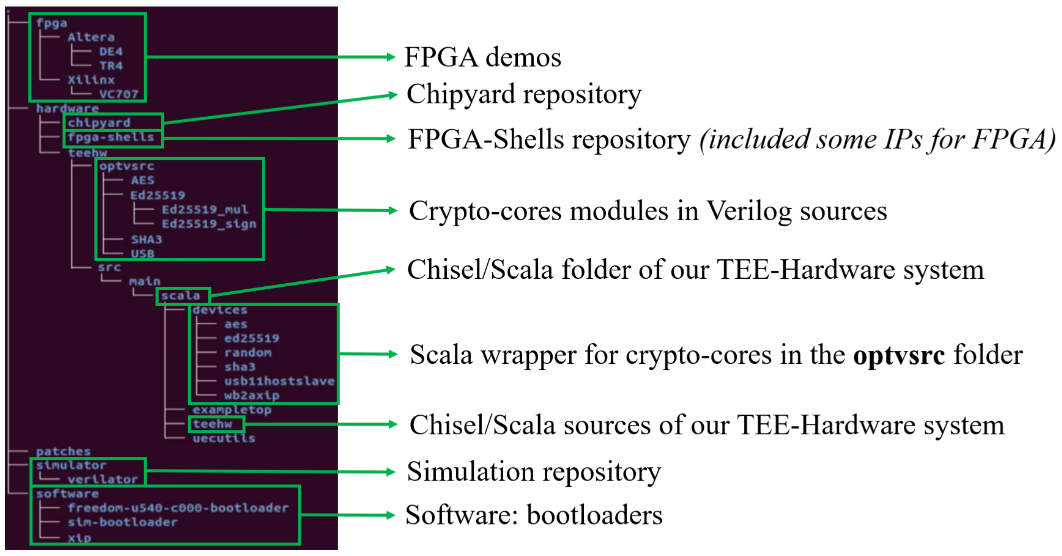
This chapter briefly explained the RISC-V architecture and the TEE terminology. This chapter introduced the two RISC-V-based computer systems, the 32-bit small MCU with VexRiscv core and the Linux bootable big platform with Rocket-chip processors. For the VexRiscv MCU, FPGA and VLSI implementations were investigated, and two chips were made in SOTB-65nm and ROHM-180nm technologies. For the Rocket-chip computer system, FPGA and VLSI implementations were also done, and one ROHM-180nm chip was made with quad-core of Rocket-chip processors. The results of the three chips were reported in this chapter. The goal of these implementations was to initiate the study of RISC-V, thus gaining experience in building a RISC-V system. Based on the knowledge gained from this chapter, the following two chapters, Chapter 5 and Chapter 6, will develop more advanced versions for the TEE purpose.

Chapter 4

Proposed TEE Hardware Computer System

4.1 TEE Hardware Framework

The primary goal of this section is to develop a RISC-V-based TEE HardWare (TEE-HW) framework that is easy to use and easy to develop over time. This framework is not only for creating a secure computer system but also for security developers in the long run. The TEE-HW framework was developed based on the Chipyard repository [126], a RISC-V generator with all sorts of chisel-related utilities and hardware assemblies. The proposed framework was published in the repository [59], and Figure 4.1 shows its folder structure. The core processor, the overall architecture with buses, and several peripherals were already provided in chisel by Chipyard library. For other cryptographic accelerators written in Verilog HDL, a proper chisel wrapper must be made for the final integration. When compiled, the chisel generator first generates an intermediate RTL representation named FIRRTL [47] and subsequently converts the FIRRTL into Verilog HDL sources. The generated Verilog file can be used in both FPGA and VLSI flows [48, 49].



```
.
├── fpga
│   ├── Altera
│   │   ├── DE4
│   │   └── TR4
│   └── Xilinx
│       └── VC707
├── hardware
│   ├── chipyard
│   ├── fpga-shells
│   └── teehw
│       ├── optvsrc
│       │   ├── AES
│       │   ├── Ed25519
│       │   │   ├── Ed25519_mul
│       │   │   └── Ed25519_sign
│       │   ├── SHA3
│       │   └── USB
│       └── src
│           └── main
│               └── scala
│                   ├── devices
│                   │   ├── aes
│                   │   ├── ed25519
│                   │   ├── random
│                   │   ├── sha3
│                   │   ├── usb11hostslave
│                   │   └── wb2axip
│                   ├── exampletop
│                   ├── teehw
│                   └── uecutils
├── patches
├── simulator
│   └── verilator
└── software
    ├── freedom-u540-c000-bootloader
    ├── sim-bootloader
    └── xip
```

FPGA demos
Chipyard repository
FPGA-Shells repository *(included some IPs for FPGA)*
Crypto-cores modules in Verilog sources
Chisel/Scala folder of our TEE-Hardware system
Scala wrapper for crypto-cores in the **optvsrc** folder
Chisel/Scala sources of our TEE-Hardware system
Simulation repository
Software: bootloaders

FIGURE 4.1: Folder structure of the TEE-HW framework repository.

Currently, the proposed TEE-HW framework supports four FPGA boards, including two Altera's FPGAs of DE4 and TR4 and two Xilinx's FPGAs of VC707 and VCU118. The makefile system was designed with many options that can be changed easily for a quick compilation to make this framework as flexible as possible. Table 4.1 gives the supported configurations, and Figure 4.2 shows a makefile example. Noted that these variables are just for an immediate change in the system, the potential development of this framework is beyond those variables presented in Table 4.1. From time to time, many other designs and architectures with many different options can be added or removed, with the current trends of development in the field.

TABLE 4.1: Supported variables in the TEE-HW framework's makefile.

Variable	Available option	Description
BOARD	- VC707 - VCU118 - DE4 - TR4	Select the FPGA board
ISACONF	- RV64GC - RV64IMAC - RV32GC - RV32IMAC	Select the ISA
MBUS	- MBus64 - MBus32	Select the bit-width for the memory bus
BOOTSRC	- BOOTROM - QSPI	Select the boot source
PCIE	- WPCIe - WoPCIe	- Include PCIe controller IP in the system - Remove PCIe controller IP from the system
DDRCLK	- WSepaDDRClk - WoSepaDDRClk	- Separate DDR-clock with System-clock - Not separate DDR-clock with System-clock
HYBRID	- Rocket - Boom - RocketBoom - BoomRocket	- Dual-core Rockets - Dual-core BOOMs - Dual-core Rocket-BOOM (Rocket core first) - Dual-core BOOM-Rocket (BOOM core first)

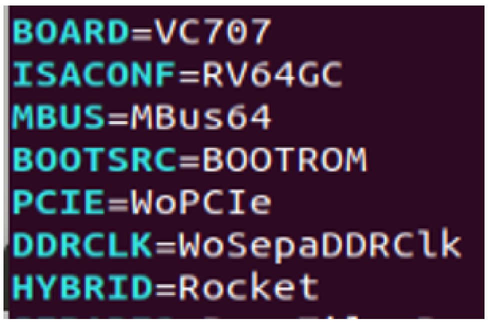

```
BOARD=VC707
ISACONF=RV64GC
MBUS=MBus64
BOOTSRC=BOOTROM
PCIE=WoPCIe
DDRCLK=WoSepaDDRClk
HYBRID=Rocket
```

FIGURE 4.2: Example for selecting parameters in the TEE-HW build, according to Table 4.1.

4.2 TEE Hardware with Cryptographic Accelerators

4.2.1 The Architecture

Based on the framework proposed in the previous Section 4.1, a completed TEE-HW architecture featuring cryptographic accelerators is proposed in this section. Figure 4.3 presents the architecture of the TEE-HW system. The core processor can be Rocket-chip [40] (open-source at [127]) or BOOM [86] (open-source at [128]), with the default ISA of RV64IMAFDC (also called RV64GC). The settings for the number of cores, type of cores, and type of ISA are reconfigurable. The implemented crypto-cores are SHA3, AES, and Ed25519. They were written in Verilog; hence, they are wrapped by a chisel-wrapper to be integrated into the system. The Ed25519 algorithm [116] is implemented with two crypto-cores, one is for the base-point multiplier, and one is for the signature generation. The buses highlighted in blue in Figure 4.3 are TileLink [129] buses, including System Bus (SBus), Peripheral Bus (PBus), and Memory Bus (MBus). The figure shows that the MBus also have a coherence cache named L2 cache, which is also reconfigurable. On the PBus, several peripherals are attached, and they are divided into two groups: the utility group and the crypto-core group. The utility group contains several essential modules such as boot ROM to store the Zero Stage BootLoader (ZSBL), UART for communication during boot time, SPI for MMC controller to read outside SD-card, and SPI for reading outside flash. The system also supports1-GB of memory via the utilization of a DDR controller. Because the DDR controllers are often the FPGA's IPs with AXI4 interfaces [130], a TileLink to AXI4 bridge is needed, as shown in Figure 4.3. The system with the accelerators and generator integration is included in a single repository [59].

The three cryptographic accelerators were chosen because of their reasonably recent specification publications, the low overhead in their execution, and they are part of the current Keystone implementation [7]. Currently, SHA3 is the latest secure hash standard, and its efficiency has been proven to be quantum-resistant due to the sponge function in Keccak-1600 [131]. On the other hand, the Ed25519, according to its authors [116], has a security level equivalent to that of a 3000-bit Rivest-Shamir-Adleman (RSA) [113] key-strength, while its implementation cost is much cheaper than an equivalent RSA implementation. To conclude, combining the SHA-3 and Ed25519 with the customizable open-source processor of RISC-V makes a significant improvement for the TEE-HW framework. The accelerators of SHA3, AES, and Ed25519 are used in the TEE boot procedure, utilizing the Keystone framework as the base environment [7]. The software calculations of the Keystone are implemented using hardware accelerators. Figure 4.4 shows the booting terminal of the architecture, and Figure 4.5 gives the Keystone's attestation test after boot.

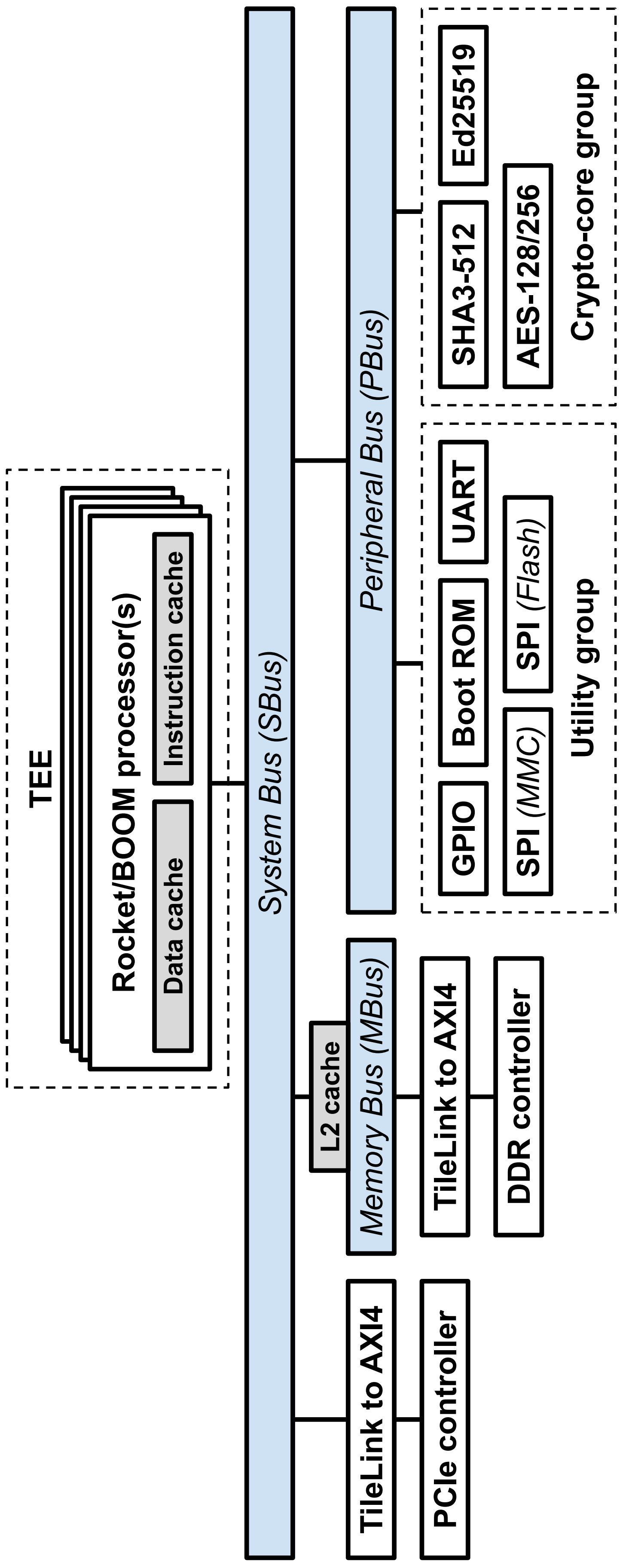
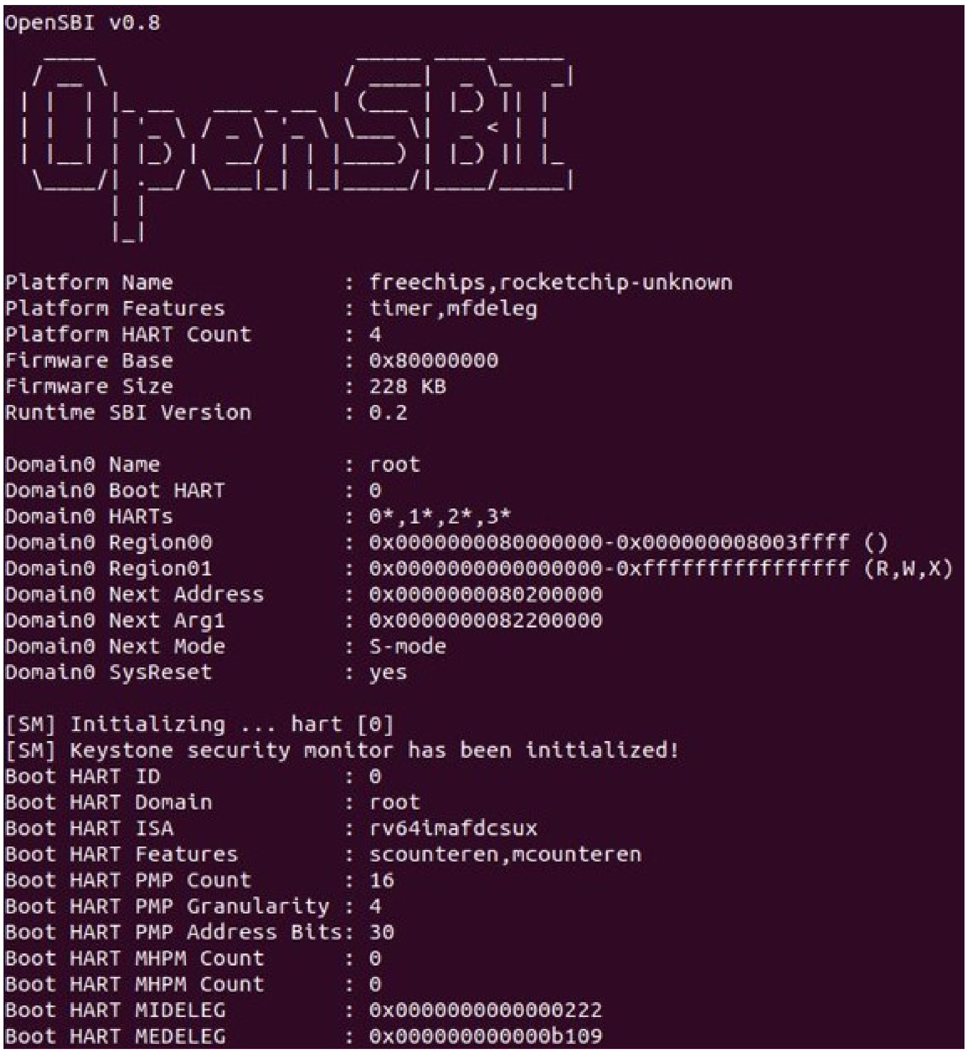


FIGURE 4.3: The proposed TEE hardware architecture with crypto-cores.



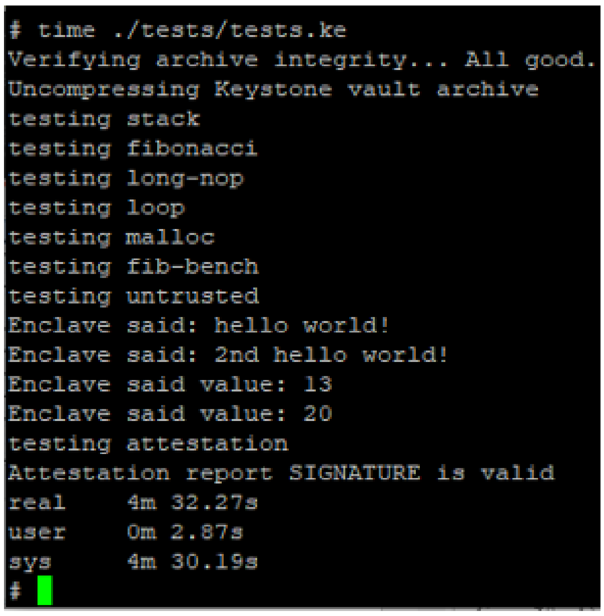
```
OpenSBI v0.8
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : freechips,rocketchip-unknown
Platform Features         : timer,mfdeleg
Platform HART Count       : 4
Firmware Base             : 0x80000000
Firmware Size             : 228 KB
Runtime SBI Version       : 0.2

Domain0 Name              : root
Domain0 Boot HART         : 0
Domain0 HARTs             : 0*,1*,2*,3*
Domain0 Region00          : 0x0000000080000000-0x000000008003ffff ()
Domain0 Region01          : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000080200000
Domain0 Next Arg1         : 0x0000000082200000
Domain0 Next Mode         : S-mode
Domain0 SysReset          : yes

[SM] Initializing ... hart [0]
[SM] Keystone security monitor has been initialized!
Boot HART ID              : 0
Boot HART Domain          : root
Boot HART ISA             : rv64imafdcsux
Boot HART Features        : scounteren,mcounteren
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 30
Boot HART MHPM Count      : 0
Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x0000000000000222
Boot HART MEDELEG         : 0x000000000000b109
```

FIGURE 4.4: Boot terminal of the TEE hardware architecture.



```
# time ./tests/tests.ke
Verifying archive integrity... All good.
Uncompressing Keystone vault archive
testing stack
testing fibonacci
testing long-nop
testing loop
testing malloc
testing fib-bench
testing untrusted
Enclave said: hello world!
Enclave said: 2nd hello world!
Enclave said value: 13
Enclave said value: 20
testing attestation
Attestation report SIGNATURE is valid
real    4m 32.27s
user    0m 2.87s
sys     4m 30.19s
# 
```

FIGURE 4.5: The keys attestation test after boot.

4.2.2 The Crypto-cores

The first implemented security accelerator for the TEE boot sequence is the SHA3 hash calculator. Figure 4.6 shows the peripheral architecture of the SHA3 crypto-core wrapped to communicate with the PBus. This accelerator contains a padding module and a Keccak-1600 round calculator [131]. The padding module retrieves 64-bit data from the register-router, then is pushed through the 576-bit buffer using a shifter. When the buffer is complete, the accelerator performs a round calculation. A constant counter keeps track of the number of rounds and the constant non-linearity of the iota phase of the Keccak round. The first round is calculated from the first 64-bit data push through the padding module. Every round state is stored in a 1600-bit status register. When the final data is pushed through the padding module, the round calculation performs the final rounds in the status registers, and then the first 512-bit word can be used for the hash output of the calculation. The usage of this module consists of pushing 64-bit data stream segments through the SHA3 crypto-core. The final stream that does not fit in a 64-bit sub-stream needs to be pushed with the help of the final size register, which is ignored for intermediate streams.

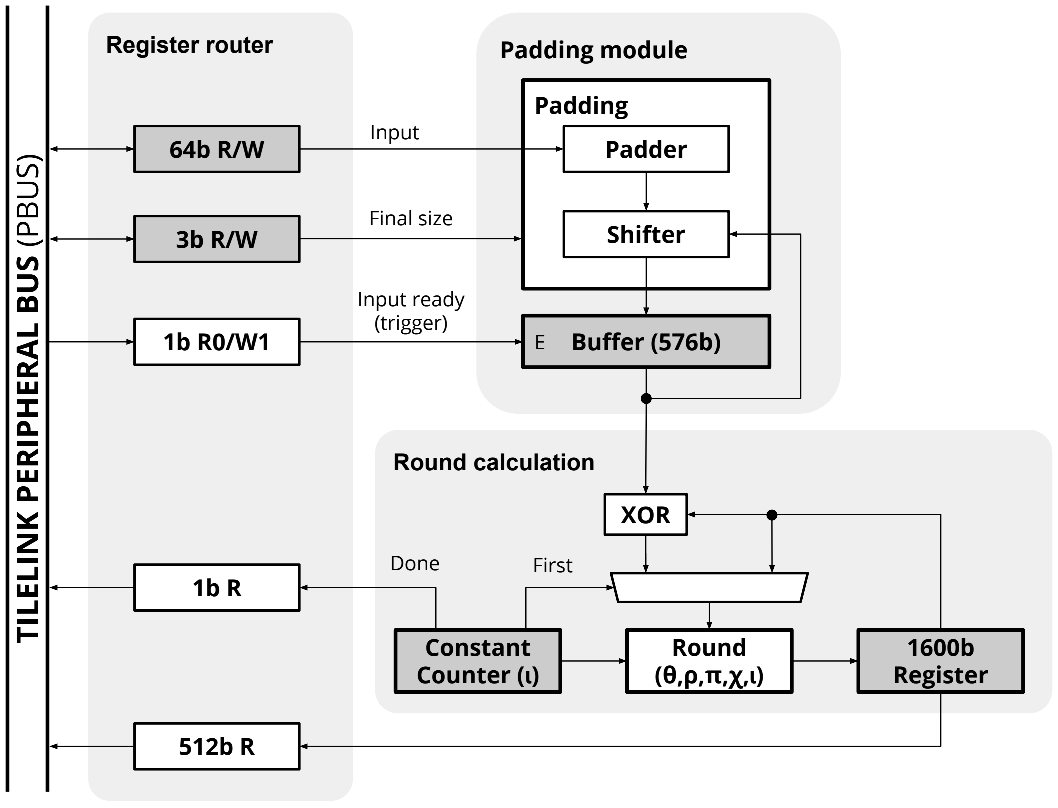


FIGURE 4.6: SHA3 crypto-core block diagram, wrapped to communicate with the PBus.

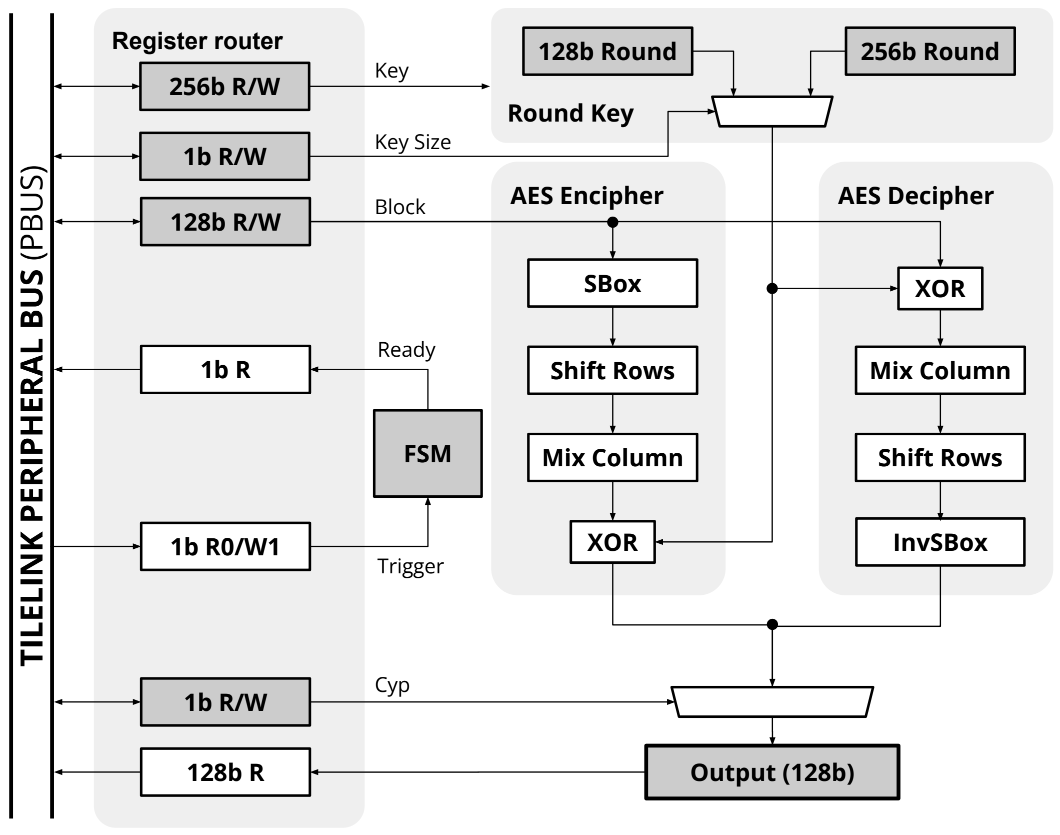


FIGURE 4.7: AES crypto-core block diagram, wrapped to communicate with the PBus.

The second implemented security accelerator belongs to the AES cipher [132]. Figure 4.7 shows the AES crypto-core block diagram wrapped by a TileLink bus to communicate over the PBus. The AES accelerator enciphers or deciphers over 128-bit or 256-bit blocks depending on the configuration. The bit-length can be changed on the fly. Each datapath performs the Substitution Box (SBox) or Inverted S-Box (InvSBox), shift rows, mix columns, and an additional round key. The round key is calculated externally for both 128-bit and 256-bit key modes. The AES calculation is performed by a state machine that enables the datapath and signals the written data into the output register.

The third accelerator is the Ed25519 base-point multiplier, and its block diagram is given in Figure 4.8. This crypto-core operates $P = sB$ in the Curve25519 elliptic curve [116]. This multiplier is useful for generating public and private keys, which will later be used for the sign and verification processes. The base point multiplier inputs the data through a memory-mapped RAM controller with write-only capabilities, where SHA3 hashed private key is stored. The Ed25519 multiplier extracts each one of the bits from memory for the base point multiplication. Each of the bits passes through a microcode FSM, which triggers different states according to the initialization, 0- and

1-bit calculation states from the extraction, rounding logic, and final calculations. This FSM provides the starting point to a microcode ROM for executing arbitrary instructions. These instructions are decoded and executed by a microprocessor containing the necessary modules for performing the 512-bit operations for the base point calculation. The microprocessor is composed of an adder, a subtract module, and a multiplier. All the calculation modules are provided with a simple calculator, useful for wrapping the value to the $2^{252} - 19$ prime number, which is necessary for this algorithm to perform the operations. The results for any operation can be stored in a 2KB memory bank, which acts as a register file, or the final output memory-mapped memory for the final output. The memory bank also contains useful constants mapped to specific addresses. The microcode contains instructions for performing vector-based operations according to the Curve25519 definition.

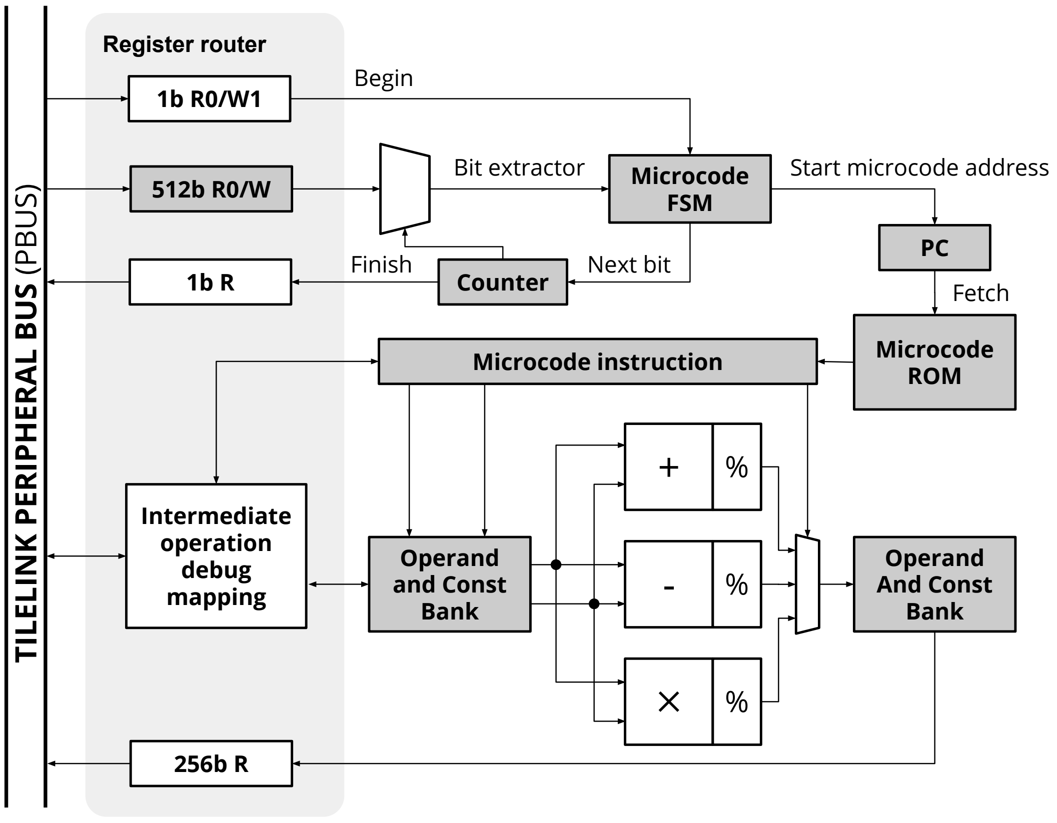


FIGURE 4.8: Ed25519-multiplier crypto-core block diagram, wrapped to communicate with the PBus.

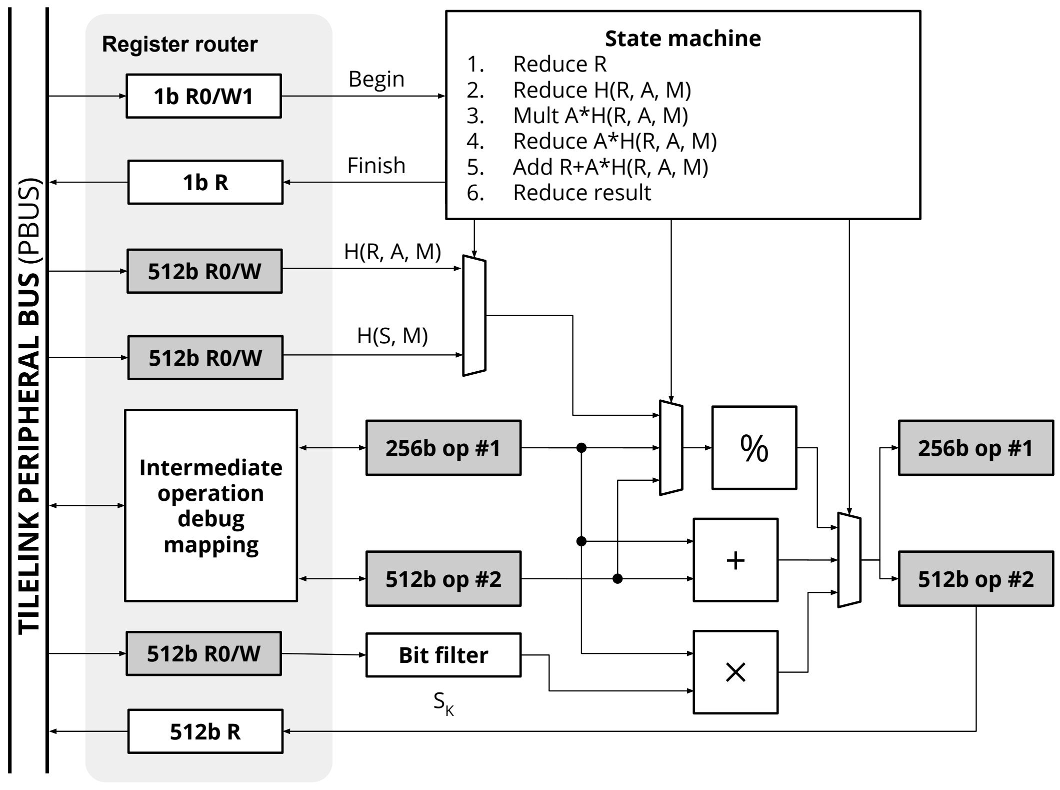


FIGURE 4.9: Ed25519-sign crypto-core block diagram, wrapped to communicate with the PBus.

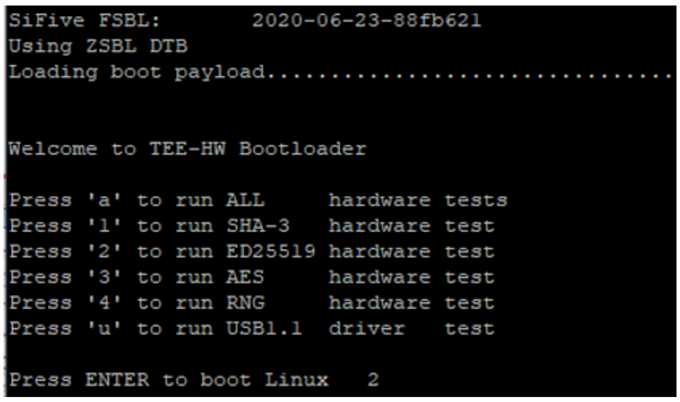
Finally, the last accelerator is the Ed25519 sign, which performs the result of the operation $S = r + H(R, A, M) \times s$, half of the signature for any message of M [116]. The other half of the signature can be calculated using the Ed25519 base-point multiplier previously described. It is also noted that the number in each operation of the Ed25519 sign is wrapped into the large prime of l. This signature is used in the Keystone for signing the bootloader program to prevent tampering. Figure 4.9 shows the datapath for the signature calculation. The main three inputs of the signature are the hashed message with the public key and the half signature $H(R, A, M)$, the hashed message with the private key $H(S, M)$, and the public key A. This information is taken for the calculation of the other half of the signature triggering a begin en-able from a memory-mapped register. The overall operation is directed by a simple FSM, which performs each of the necessary procedures of the signature equation. When the FSM is ready, the enable signal triggers further states for calculating each one of the steps of the equation. The datapath this composed of a 256-bit adder, a multiplier, and a module calculator for wrapping value through the large prime l. The logic contains a 256-bit register for holding the sum results and a 512-bit register for the multiplication results. The FSM first reduces the R from the $H(S, M)$ value and stores it inside the 256-bit register. The reduction of $H(R, A, M)$ and the subsequent multiplications and reductions with A are stored in the 512-bit register. The final sum and reduction are calculated from the registers and stored in the final 512-bit register output.

Most of the hashing and key generation calculations are done by pushing the data to specific data registers. The PBus reads the data from the registers and performs the desired output. The signature procedure follows a different way to accelerate the result. According to Figure 4.9, the signature receives calculated parts of the signature equation [116]. The algorithm 1 presents the procedure in software to calculate a full signature from the original data using the signature accelerator. This procedure uses the SHA3 and Ed25519-multiplier crypto-cores to obtain the two halves of the signature P and Q. The message and the secret key most-significant bits are first hashed using SHA3 and stored in r. To calculate R, the previous value is passed to the hardware Ed25519-multiplier. Again, the SHA3 is used with the message, the public key, and the result of R. The next steps involve pushing the previously calculated data into the registers. The hardware procedure performs the operations to calculate the P and Q halves of the message signature.

Algorithm 1 Ed25519-sign crypto-core in operation using the other SHA3 and Ed25519-multiplier crypto-cores.

```
1:  procedure HW $Ed25519_{sign}(message, P_k, S_k)$
2:      $r \leftarrow$ HW SHA3$(S_k[high], message)$
3:      $R \leftarrow$ HW $Ed25519_{mult}(r)$        ($Note$: $R = rB$)
4:      $RAM \leftarrow$ HW SHA3$(R, P_k, message)$
5:      (In HW) $H(S, M) \leftarrow r$
6:      (In HW) $H(R, A, M) \leftarrow RAM$
7:      (In HW) $S_K \leftarrow S_K$
8:      (In HW) $TriggerAndWait()$
9:      return $[P, Q]$
10: end procedure
```

To verify the functionalities of all crypto-cores, besides the Verilog-based simulation tests, a simple software at M-mode was developed and embedded in-side the FSBL to check the integration of hardware accelerators. Figure 4.10 shows the FSBL's welcoming screen with multiple options for testing various crypto-cores. Figure 4.11, Figure 4.12, and Figure 4.13 show the tests for the SHA3, AES, and Ed25519 crypto-cores, respectively. Each test will do the software program first and then execute the hardware alternative with the same test set. The execution times were also recorded, as shown in the figures.



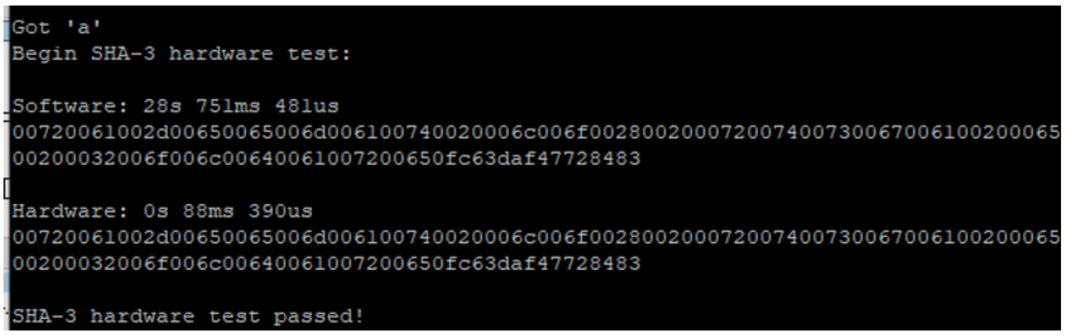
```
SiFive FSBL:       2020-06-23-88fb621
Using ZSBL DTB
Loading boot payload..............................


Welcome to TEE-HW Bootloader

Press 'a' to run ALL     hardware tests
Press '1' to run SHA-3   hardware test
Press '2' to run ED25519 hardware test
Press '3' to run AES     hardware test
Press '4' to run RNG     hardware test
Press 'u' to run USB1.1  driver   test

Press ENTER to boot Linux   2
```

FIGURE 4.10: Welcoming screen at the FSBL to select various tests for crypto-cores.



```
Got 'a'
Begin SHA-3 hardware test:

Software: 28s 751ms 481us
00720061002d00650065006d006100740020006c006f00280020007200740073006700610020006500200032006f006c0064006100720065
00200032006f006c0064006100720065
```

FIGURE 4.11: The SHA3 hardware test at FSBL.

4.2.3 Boot Procedure with Crypto-cores

The TEE boot process is based on the Keystone framework [7] (open-sources at [102]), as described in sub-section 3.2.2. Noted that the original Keystone boot flow does not include the RoT yet, it assumes that the device pair keys already generated, and the TEE is being built upon. The boot flow in this section follows the same suit, just replacing the hash and keys generation software with the equivalent hardware accelerators. The bootloader named Zero-State BootLoader (ZSBL) will be executed first and foremost, and it is stored in the boot ROM (refers to Figure 4.3). The ZSBL is also the one that will do the first authentication and then call the next bootloader named First Stage BootLoader (FSBL) stored in the SD-card. After the FSBL, a Linux bootloader such as Berkeley BootLoader (BBL) or OpenSBI will be loaded and boot into the Linux kernel with the initial filesystem. To demonstrate the TEE authentication process, the whole Linux bootloader payload will be used for verification in this case.

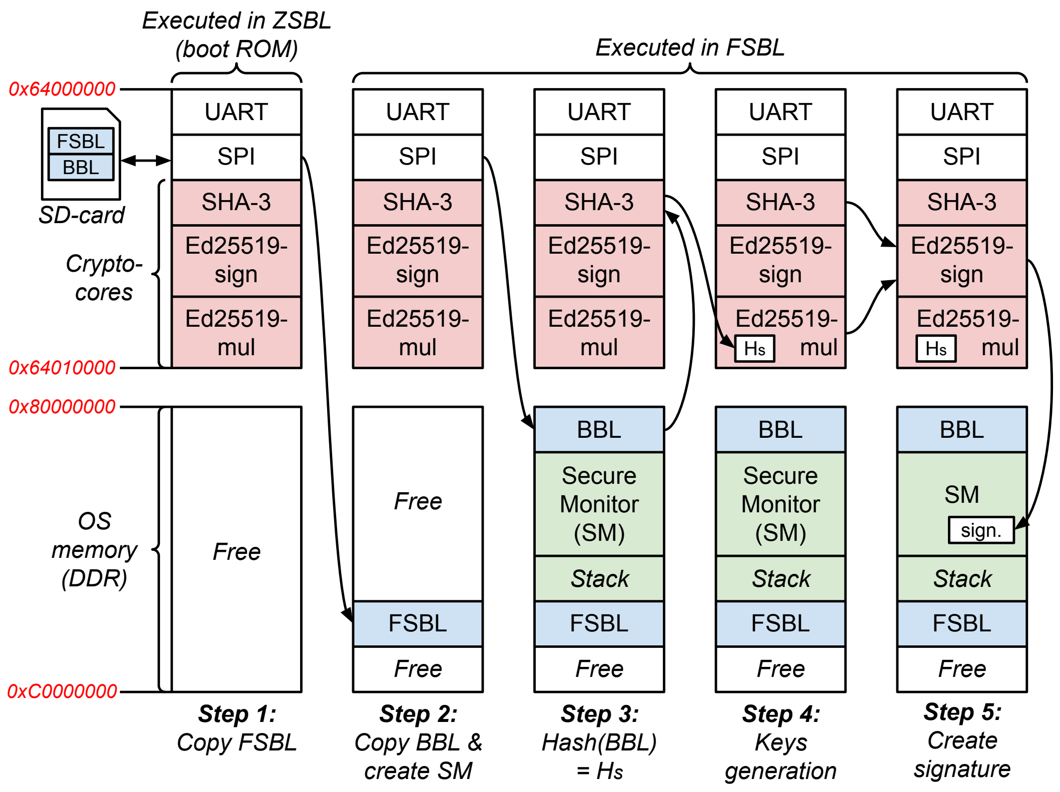


FIGURE 4.14: TEE boot process with SHA3 and Ed25519 crypto-cores.

Figure 4.14 describes the TEE boot process using the crypto-cores. First, the ZSBL in the boot ROM locates and copies the FSBL from the external media such as SD-card to the main DDR memory. Then, it jumps to the FSBL and executes there. After that, the FSBL locates and copies the BBL from the SD-card to the DDR. It also creates the Secure Monitor (SM) in the process. The SM then extracts the initial seed for the keys by hashing the BBL payload. The hash calculation is performed by the SHA3 crypto-core previously described. The hash result is then pushed to the Ed25519 base-point multiplier crypto-core for generating the private and public keys. With the help of the SHA3 accelerator and the newly created keys, the SM performs the signature of the BBL payload then stores the result in a secure memory address for further use. At this point, the Linux kernel can be booted by executing the BBL. Finally, after boot, the BBL authentication can be done by the keys attestation function provided by the SM via signature verification.

4.3 TEE Hardware with Isolated Root-of-Trust

4.3.1 The Main-System

Although various RoT implementations were proposed, some of them relied on obscurity for preserving the root keys [1–3, 9], and some others used a fixed circuit with a specific set of constraints to do the RoT [11, 12, 57, 66]. Therefore, they did not provide the flexibility for the keys generation scheme. As a result, the main goal in this section is to develop an RoT implementation in an isolated architecture for RISC-V-based TEE SoCs. The proposed design is given in Figure 4.15. This design is a continuous version of the one presented in the previous section 4.2. Compared to the previous work, the isolated sub-system and the TRNG implementation are the two new things being added to the architecture. Similar to the previous version in Section 4.2, the architecture in Figure 4.15 also has several properties that can be reconfigured easily depending on specific requirements, such as the number of cores, the type of cores, the ISA configuration, and the sizes of the L1 and L2 caches. Furthermore, the PCIe connection, the whole isolated sub-system, and each crypto-core can be included or excluded depending on needs. The default configuration is a dual-core system with the Rocket core first and the BOOM core second; each core has a 16-KB instruction cache and 16-KB data cache. By default, the ISA is RV64GC, the size of the L2 cache is 512-KB, the PCIe controller is excluded, the isolated architecture is included, and all of the peripherals shown in Figure 4.15 are included.

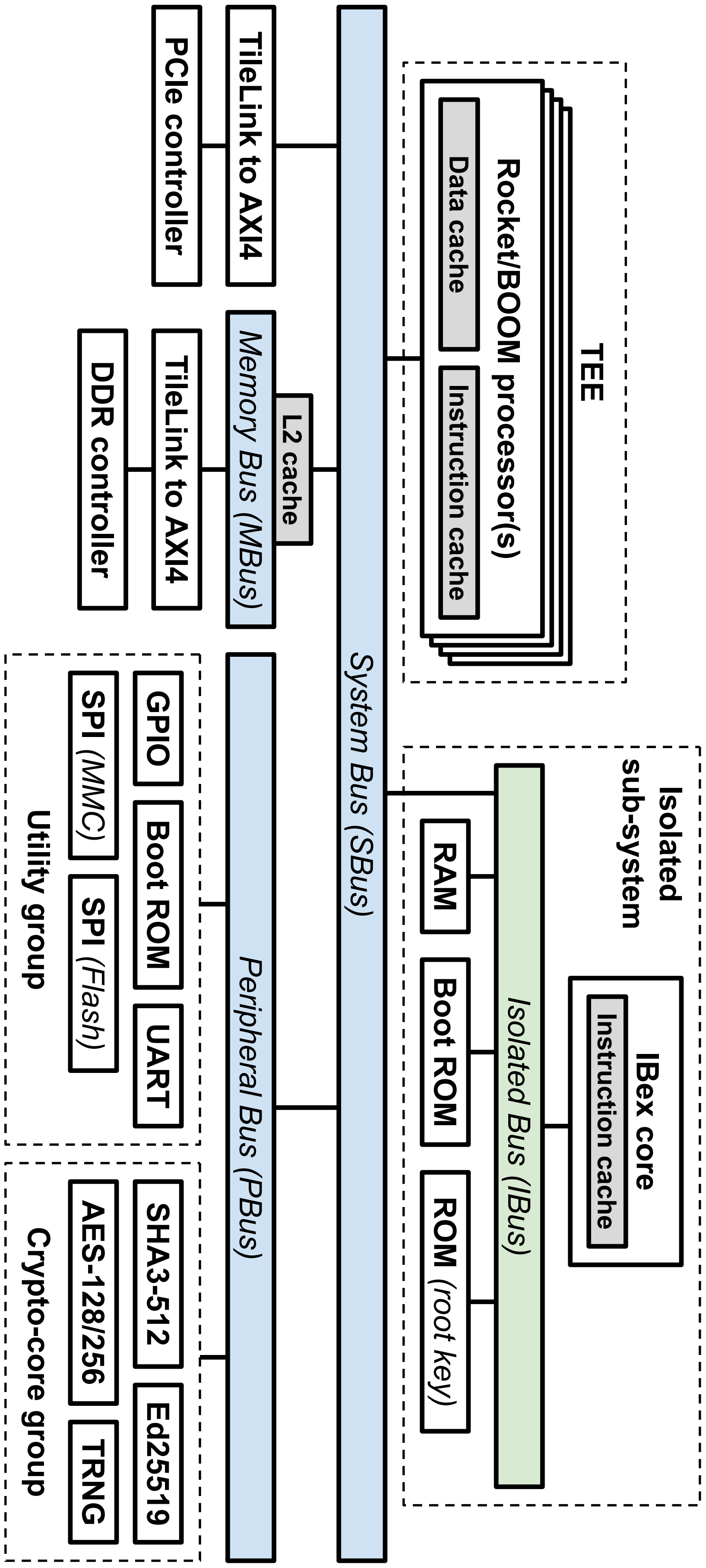


FIGURE 4.15: The proposed TEE-HW architecture with isolated sub-system.

The main design idea is the heterogeneous architecture of 64-bit Linux-capable TEE processors combined with a 32-bit isolated MCU. At reset, the isolated MCU will boot first, do the first authentication, and then generate keys using root keys salted with random numbers. After that, the TEE processors will be woken up to follow the conventional TEE boot flow [7]. The main bus of the isolated sub-system is called the Isolated Bus (IBus), and it connects to the conventional SBus via a master-only TileLink protocol [129]. Therefore, the MCU can access all the sub-modules in the SoC, but the TEE processors cannot access the IBus peripherals. As a result, the memory in the isolated sub-system is the best place for storing root keys. The SBus also includes the option of having a coherence cache manager, the L2 cache. As shown in Figure 4.15, the PBus contains several peripherals that can be categorized into two groups of utility group and crypto-core group. The utility group consists of a GPIO, a UART, a boot ROM, an SPI for using SD-card, and an SPI for using a flash device. All cryptographic accelerators needed for TEE are included in the crypto-core group, such as SHA3, AES, Ed25519, and TRNG. For the OS memory space, the TEE hardware system offers a 1-GB DDR controller. This controller is driven by an AXI4 [130] bus connected to the MBus via a TileLink-to-AXI4 bridge. The DDR IP could be Altera's or Xilinx's depends on which FPGA is being used. For the VLSI implementation, the MBus signals can be exported to the outside like GPIO digital buffers. Therefore, the fabricated chip can be mounted on an FPGA to use that FPGA's DDR IP as its primary OS memory.

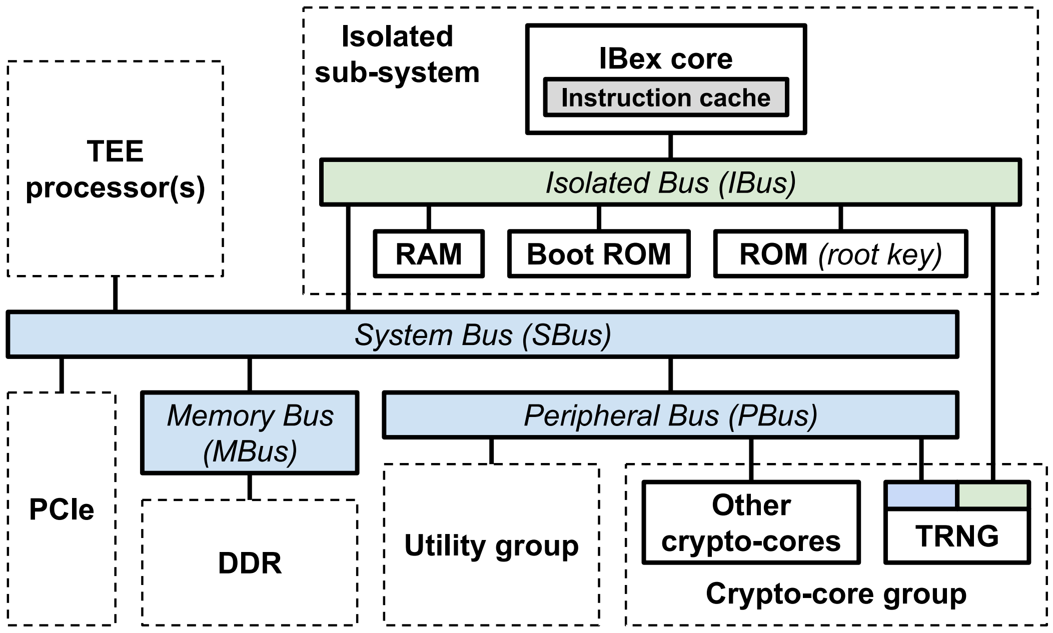


FIGURE 4.16: The isolated sub-system with direct connection to the TRNG module.

4.3.2 The Isolated Sub-system

Figure 4.16 shows the isolated 32-bit architecture side-by-side with the conventional 64-bit TEE processors. The isolated sub-system contains a RISC-V-based RV32IMC IBex core [133]. The IBex was chosen because it is a small 32-bit core with tamper awareness. The main bus in the isolated architecture is a TileLink bus named IBus, and its peripherals are a boot ROM, a 16-KB RAM, and a ROM for storing root keys. Additionally, this isolated sub-system contains its own Platform Level Interrupt Controller (PLIC) and Core Local INTerrupt (CLINT). The isolated CLINT handles internal core-level interrupts for scheduling. Through the PLIC, the outside TEE processors can issue commands to the isolated core for keys attestation. Then, the IBex core handles the PLIC's interrupts through programs stored in its boot ROM.

The TRNG used in this section is based on the previous work [134]. Because the NIST standard requires the keys generation and TRNG to be in the same environment [135], the peripheral that wraps the TRNG core was designed with two interfaces, one for the PBus and one for the IBus, as shown in Figure 4.16. The blue interface responds to the PBus, and the green interface responds to the IBus. As a result, the IBex core has a direct connection to the TRNG module, and it could use the TRNG core without the risk of exposing data to the public domain. Figure 4.17 shows the TRNG module with two bus connections. In the figure, the TRNG core highlighted in blue comes from the previous work [134]. The generated random number can be extracted via any channel. However, the TRNG's peripheral is configured to respond with a higher priority for the IBus. As shown in Figure 4.17, if the *isolated enable* is activated, the accumulation process will halt and prevent the shift-register from accumulating the result. When the TRNG finishes its IBus transaction, it will self-reset and then resume the PBus transaction if there was one. Because the commands that come from the two channels are not treated as equals, the TRNG's outputs are classified as non-Independence and Identically Distributed (non-IID) data in this case. The TRNG core was proven to pass the NIST non-IID restart test [134]. As a result, the two-channel TRNG's peripheral did not affect the quality of generated random numbers.

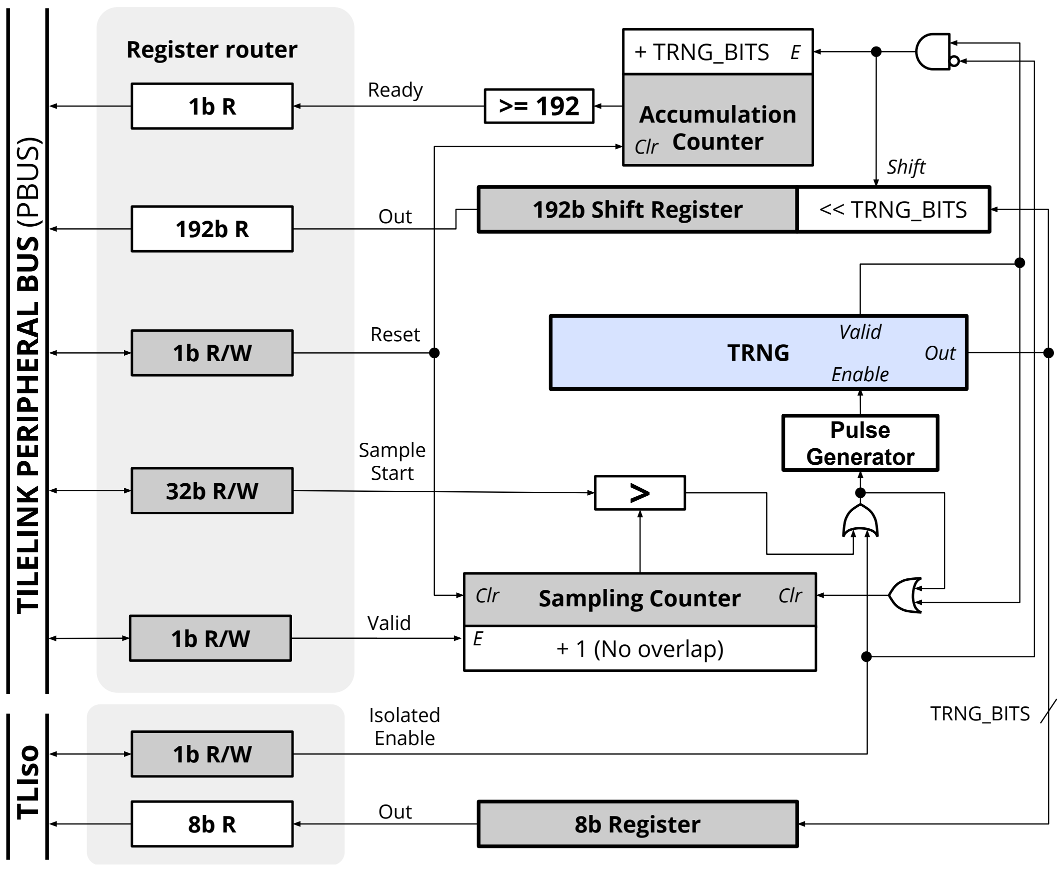


FIGURE 4.17: The TRNG's peripheral with two interfaces.

4.3.3 Secure Boot Process with Root-of-Trust

Figure 4.18 shows the keys management scheme with the secure boot process and the Keystone boot flow done by the isolated processor and TEE processors, respectively. The idea is that the trusted manufacturer plays the role of a root CA; thus, its public key of P_M is well-known, and its certificate $M_{Cert.}$ is self-signed. One manufacturer can have many key pairs, but each pair must be unique for different manufacturers. Because the S_M and P_M generation are done offline, they should be high-bit RSA keys with many years of validity.

The subsequent keys of S_R and P_R, called the root keys, are EC keys, and they are created by the manufacturer at the design time. The root certificate $R_{Cert.}$ is signed offline by the manufacturer's secret key S_M. The root's secret key S_R is not stored in the chip, but the root's public key P_R is stored in the isolated ROM for the ZSBL authentication. The secure BootLoader (sBL) is stored in the same place with the P_R, the isolated boot ROM. The content in sBL is signed by the S_M previously, as shown in Figure 4.18. At reset, the IBex core executes the sBL, and its very first task is to verify and load the ZSBL using the given P_R.

The next step is the generation of the device/chip's EC keys pair, S_D and P_D. As shown in Figure 4.18, they are also generated offline by the manufacturer. The device's secret key S_D is stored in the isolated ROM, and the device's public key P_D is stored in the public domain (on-chip ROM or off-chip flash).

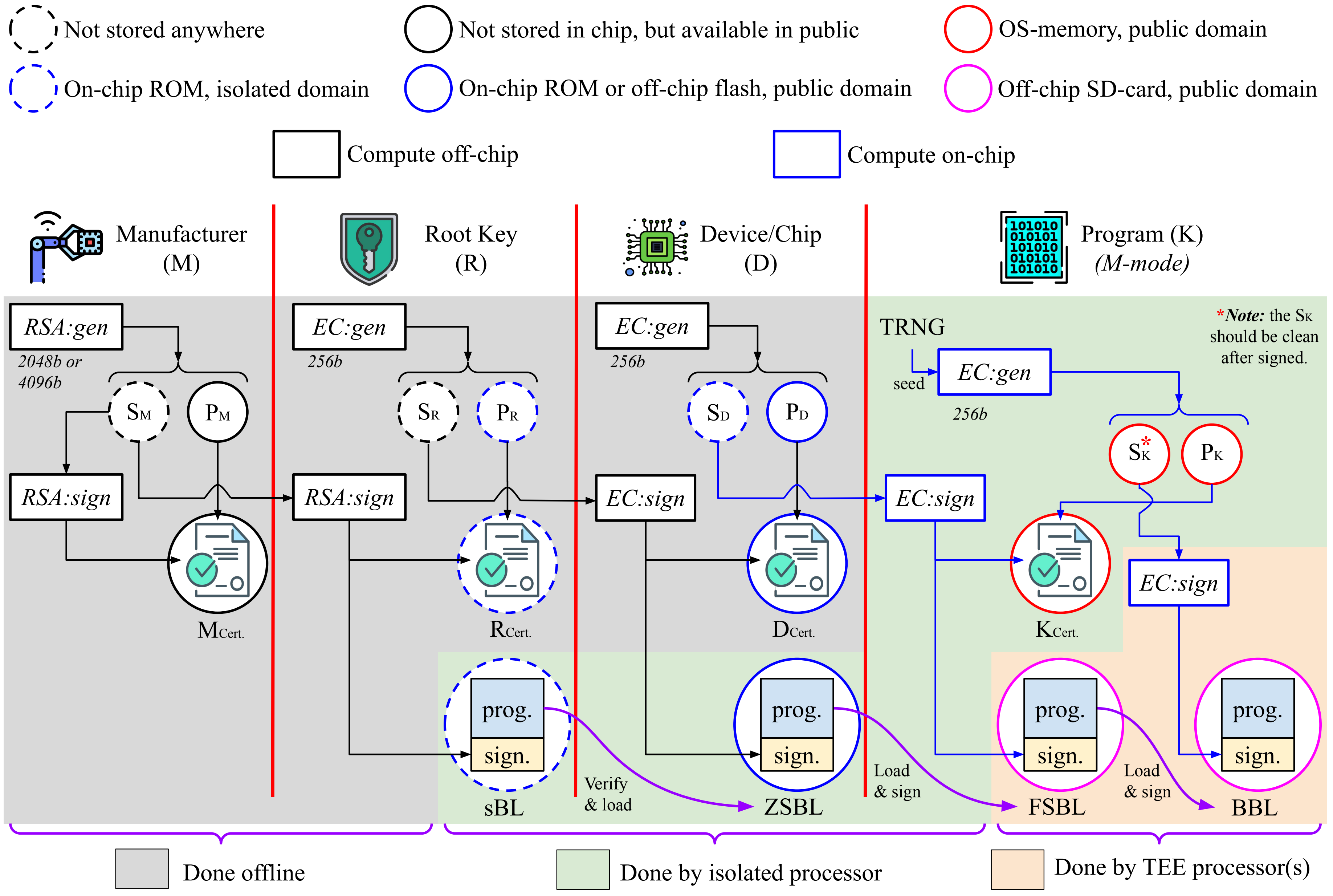


FIGURE 4.18: Proposed keys management scheme with the secure boot process. The recommended storages for sBl, ZSBL, FSBL, and BBL are the isolated ROM, off-chip flash, SD-card, and SD-card/hard-drive, respectively.

The ZSBL is stored in the same place as the P_D, and it has a signature pre-signed by the root's secret key S_R. Because the first act of the isolated processor is to verify and load the ZSBL, this scheme allows the manufacturer to update the ZSBL securely, even if the ZSBL is stored in public as in an off-chip flash.

After verified and loaded, ZSBL then uses the TRNG as seed for the EC-genkey to generate the subsequent keys pair of S_K and P_K, called Keystone keys. These keys are stored in a public RAM on the TEE's side. Then, the device's secret key S_D is used to sign on the Keystone's public key P_K, thus creating the Keystone certificate $K_{Cert.}$, as seen in Figure 4.18. The content of FSBL is then loaded from the SD-card outside to the TEE's main memory, hashed, and signed by the S_D. After this step, both Keystone keys pair and FSBL content are in the TEE's main memory, ready to be executed by the TEE processors. Finally, the isolated core wakes the TEE processors to follow the conventional TEE boot flow. From this point forward, the procedure of loading BBL and setting up SM using crypto-cores is the same as described in 4.2.3.

The connection from the IBus to the SBus, shown in Figure 4.15 and Figure 4.16, is master-only TileLink protocol. That means all the sub-modules below the SBus are accessible from IBus but not the other way round. Therefore, all the information processing inside the isolated sub-system are inaccessible for TEE processors due to the bus architecture. Hence, the ROMs and RAM inside the isolated sub-system are ideal for storing root/device keys and executing the sBL and ZSBL. When booting, the isolated sub-system will boot first to establish the RoT. Figure 4.19 describes the software execution in the isolated environment. This is the secure boot program modified from the boot flow with crypto-cores described in sub-section 4.2.3. This program will run once the system is in the reset state.

At reset, the TEE processors will be in a wait state, waiting for an interrupt. The isolated sub-system fetches the root/device keys from the ROM and then salts it with TRNG to create a seed for Ed25519, thus generating the Keystone keys pair. After making the S_K and P_K, the S_K will be stored in a write-only memory (refers to the Ed25519 crypto-core described in Section 4.2), which can be read only by crypto-cores for usage, but not by TEE processors or even the IBex core. The crypto-cores can use this write-only memory to calculate the signature for a stream of bits. In this case, the OS bootloader (S) is hashed and then signed internally by the previously-stored private key Curve25519 function.

Due to the isolation, the software described in Figure 4.19 cannot be tampered with by any external program on the TEE side. Moreover, the TEE processors execute programs only after the secure boot stage. Hence, the only potential threat from the TEE side is by exploiting the interrupt channel for attestation. However, the IBex core only responds to the external interrupts based on its program written in the isolated boot ROM. Thus, we can

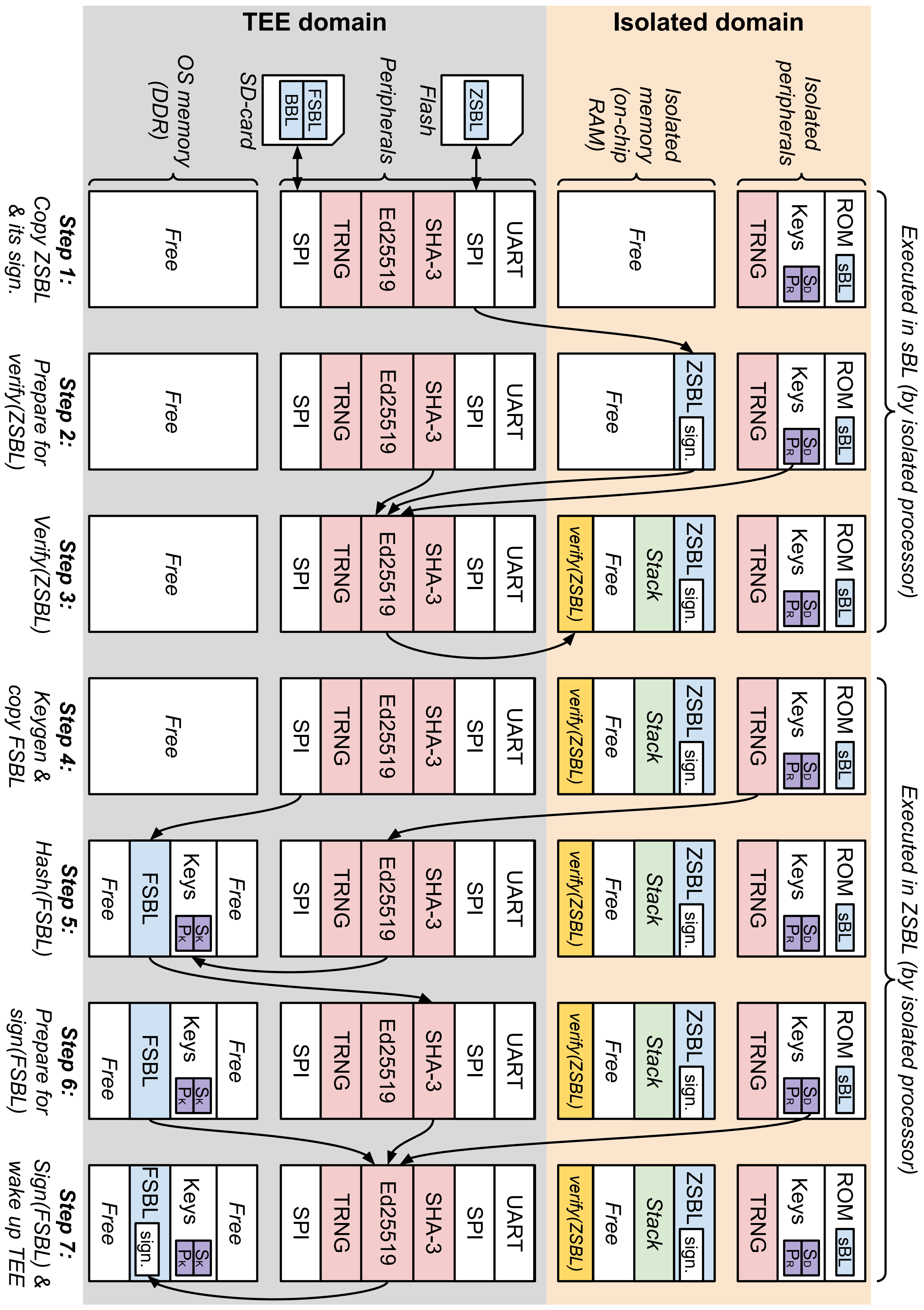

Isolated domain
TEE domain
Isolated peripherals
Isolated memory (on-chip RAM)
Peripherals
Flash
ZSBL
SD-card
FSBL
BBL
OS memory (DDR)
Executed in sBL (by isolated processor)
Executed in ZSBL (by isolated processor)
ROM
sBL
Keys
SD
PR
SK
PK
TRNG
Free
UART
SPI
SHA-3
Ed25519
ZSBL
sign.
Stack
verify(ZSBL)
FSBL
Step 1: Copy ZSBL & its sign.
Step 2: Prepare for verify(ZSBL)
Step 3: Verify(ZSBL)
Step 4: Keygen & copy FSBL
Step 5: Hash(FSBL)
Step 6: Prepare for sign(FSBL)
Step 7: Sign(FSBL) & wake up TEE

FIGURE 4.19: Boot sequence in the isolated environment.

change the IBex's program in the future to adapt with the attack vector if there is one.

4.4 Summary

In this chapter, a TEE-HW framework was proposed. This framework will be used as a tool not only for creating a secure computer system but also for future development. Many architecture aspects were kept optional and can be changed easily by selecting proper parameters in the Makefile. The available hardware accelerators are SHA-3, AES, Ed25519, and TRNG. The supported processors are Rocket-chip and BOOM. Based on the proposed framework, a completed TEE-HW system featuring several cryptographic accelerators was built. A TEE boot flow was developed based on the Keystone framework with some modifications for using hardware accelerators. Finally, a heterogeneous design featuring an isolated sub-system was also presented in this chapter. The chosen secret core is the RV32IMC IBex with the anti-tampering feature. The isolated architecture contains ROMs and RAM for storing root/device keys and executing the secure boot program. The complete secure boot flow with an elaborated keys management scheme is also presented in the chapter. With various crypto-cores available and the hidden MCU to store the RoT, the secure boot program is flexible and adapts to future threats.

This page intentionally left blank.

Chapter 5

Performance Analysis

5.1 FPGA and VLSI Implementations

5.1.1 Processors VLSI Reports on Different Configurations

The proposed TEE-HW system supports Rocket, BOOM, and IBex cores with various ISA settings of RV64GC, RV64IMAC, RV32GC, and RV32IMAC. To compare different cores with different settings, two tables were made. They are Table 5.1 for Rocket cores and Table 5.2 for BOOM and IBex cores. The L1 caches in Rocket and BOOM cores are set at the default setting with 16-KB for each instruction and data cache. The complete VLSI flow, from synthesis, PNR, to RC-extract, is done for each core with each available setting, and the results are given in the two tables. In Table 5.2, because BOOM version 3 (SonicBOOM [86]) currently doesn't support 32-bit FPU, the setting of RV32GC is not available for the BOOM cores. Additionally, the ISA of the IBex core is fixed at RV32IMC [133].

According to the results in Table 5.1, Rocket cores are doing well in VLSI implementations. Their achieved F_{Max} values are always sign-off according to the set constraint. On the other hand, the BOOM cores in Table 5.2 have trouble meeting with the timings. To be specific, the BOOM's F_{Max} values are roughly around 20-MHz to 30-MHz after synthesized and reach its top of around 85-MHz after PNR, even with the 100-MHz set constraint. Furthermore, in the case of BOOM-RV64GC with the 100-MHz set constraint, the PNR just fails to route. For the IBex core's performance, according to Table 5.2, it could reach up to around 90-MHz with the 100-MHz set constraint.

For the sizes, the Rocket cores are around 6-mm^2 to 8-mm^2 while the BOOM cores are around 21-mm^2 to 25-mm^2. The BOOM costs are closely $2\times$ to $2.5\times$ bigger than the Rocket cores with the same configuration. The IBex's size is about 1.2-mm^2 or less, too small in comparison with the Rocket or the BOOM at the default setting. For the power consumption, the Rocket cores consume around 400-mW to 500-mW with the 100-MHz clock, and around 200-mW to 250-mW with the 50-MHz clock. For the BOOM cores, the power consumptions reach around 1,200-mW and 600-mW for 100-MHz and 50-MHz clocks, respectively. Hence, the power consumptions of the BOOMs are about $2.5\times$ to $3\times$ more than that of the Rockets. Finally, for the IBex core, the values of 29.52-mW to 134.4-mW are just a fraction compared to the other two.

TABLE 5.1: Rocket core's VLSI reports on different configurations and constraints, using ROHM-180nm library.

Processor	Core	**Rocket**							
	ISA	**RV64GC**		**RV64IMAC**		**RV32GC**		**RV32IMAC**	
Cache	Inst.	16-KB		16-KB		16-KB		16-KB	
	Data	16-KB		16-KB		16-KB		16-KB	
@sdc constraint		100-MHz	50-MHz	100-MHz	50-MHz	100-MHz	50-MHz	100-MHz	50-MHz
@SYN	#Cell	107,472	102,814	66,649	63,996	94,829	91,547	54,563	53,055
	Area (mm^2)	5.74	5.68	4.80	7.78	5.37	5.31	4.44	4.42
	SRAM (mm^2)	3.11	3.11	3.11	3.11	3.04	3.04	3.04	3.04
	SRAM (%)	54.15%	54.73%	64.72%	65.04%	56.68%	57.29%	68.58%	68.84%
	Power (mW)	723.23	385.31	394.50	207.98	653.74	342.60	309.62	168.39
	F_{Max} (MHz)	93	48	99	48	93	48	94	59
@PNR	#Gate	600,909	595,424	501,890	499,757	561,096	554,917	464,883	462,875
	#Cell	111,242	107,054	69,504	66,879	98,098	94,636	57,921	56,029
	Size (mm^2)	7.90	7.82	6.60	6.57	7.40	7.32	6.11	6.09
	Density	74.68%	74.77%	74.71%	74.75%	74.53%	74.47%	74.88%	74.83%
	Power (mW)	551.4	280.9	414.6	208.7	503.5	255.6	363.5	184.2
	F_{Max} (MHz)	100	50	100	50	100	50	100	59
@RC extract	#MOSFET	3,198,750	3,216,446	2,763,356	2,760,678	3,008,196	2,986,976	2,558,976	2,564,534
	F_{Max} (MHz)	102	50	103	50	102	50	102	59

TABLE 5.2: BOOM and IBex cores' VLSI reports on different configurations and constraints, using ROHM-180nm library.

Processor	Core	BOOM (*v3*)						IBex	
	ISA	**RV64GC**		**RV64IMAC**		**RV32IMAC**		**RV32IMC**	
Cache	Inst.	16-KB		16-KB		16-KB		none	
	Data	16-KB		16-KB		16-KB		none	
@sdc constraint		100-MHz	50-MHz	100-MHz	50-MHz	100-MHz	50-MHz	100-MHz	50-MHz
@SYN	#Cell	362,892	359,569	289,997	283,308	252,674	250,429	35,348	22,184
	Area (mm^2)	18.77	18.69	16.86	16.80	15.68	15.65	0.87	0.54
	SRAM (mm^2)	9.03	9.03	9.03	9.03	8.78	8.78	0.00	0.00
	SRAM (%)	48.11%	48.31%	53.54%	53.75%	55.99%	56.08%	0.00%	0.00%
	Power (mW)	2,406.49	1,281.92	1,784.70	968.16	1,514.57	782.61	326.18	93.65
	F_{Max} (MHz)	37	24	35	20	31	25	56	44
@PNR	#Gate	Failed to PNR	1,950,993	1,786,832	1,755,903	1,657,757	1,631,193	104,223	53,765
	#Cell		371,560	311,974	296,423	272,489	259,391	39,112	22,529
	Size (mm^2)		25.48	23.02	22.93	21.42	21.39	1.20	0.76
	Density		74.69%	75.74%	74.73%	75.56%	74.46%	87.06%	72.09%
	Power (mW)		761.7	1,284	627.6	1,130	555.9	134.4	29.52
	F_{Max} (MHz)		50	82	50	84	50	88	51
@RC extract	#MOSFET	Failed to PNR	9,305,990	8,447,134	8,395,652	7,831,036	7,823,444	451,506	237,230
	F_{Max} (MHz)		54	84	54	87	53	89	53

5.1.2 TEE-HW with Crypto-cores

Table 5.3 shows the results of the FPGA implementation; the used FPGA board is the Altera DE4 with the FPGA chip of Stratix IV GX EP4SGX230. The resources crypto-cores are compared in percentages to a single-core Rocket tile. According to the table, the Rocket tile is the most extensive core of all; the others are about 10% to 25% compared to it. The instruction and data caches of the Rocket were configured to be 16-KB for each one of them; hence the 318,208-bit (about 38.8-KB) of memory was reported, as shown in Table 5.3. The Ed25519-multiplier also contains 65,536-bit (8-KB) of memory for the register banks in the temporal calculations. The Ed25519-multiplier also needs DSP blocks for extensive calculations units in the rounding machine. And for the Rocket tile, the DSP blocks are for implementing Floating-Point Unit (FPU)and integer multipliers/dividers.

Table 5.4 gives the synthesis results on the ROHM-180nm process for the TEE-HW architecture with cryptographic accelerators. The build reports of crypto-cores are compared in percentages with one core Rocket-chip. As shown in the table, one core Rocket-chip with the settings of 16-KB instruction cache and 16-KB data cache occupied a 7.9-mm^2 die area; it is the most significant core in the table. The second biggest core is the Ed25519-multiplier with 2.87-mm^2. The smallest core is the AES crypto-core with only 0.65-mm^2. All the cores achieved roughly 90-MHz to 100-MHz for the maximum operating frequencies. For the power consumption, besides the Rocket, Ed25519-sign is the next core that has the most consuming power with 80.89-mW, 14.67% compared to the Rocket processor.

TABLE 5.3: TEE-HW with cryptographic accelerators, an FPGA build report on Stratix IV GX (EP4SGX230).

	SHA3	**AES**	**Ed25519-multiplier**	**Ed25519-sign**	**Rocket-chip (1-core)**
Combi. ALUT	8,108	3,195	2,737	3,969	31,374
Combi. ALUT (%)	25.84	10.18	8.72	12.65	100
Register	2,790	2,854	4,778	4,617	19,483
Register (%)	14.32	14.65	24.52	23.70	100
Memory (bit)	0	0	65,536	0	318,208
DSP block	0	0	48	0	22

TABLE 5.4: TEE-HW with cryptographic accelerators, an VLSI synthesis report on ROHM-180nm process.

	SHA3	**AES**	**Ed25519-multiplier**	**Ed25519-sign**	**Rocket-chip (1-core)**
Area (mm^2)	1.32	0.65	2.87	1.81	7.90
Area (%)	16.71	8.23	36.33	22.91	100
NAND2-gate	102,500	50,560	222,432	140,442	600,909
F$_{Max}$ (MHz)	104	90	106	91	100
Power (mW)	42.75	37.57	53.06	80.89	551.4
Power (%)	7.75	6.81	9.62	14.67	100
(1.8-V @ 100-MHz)					

5.1.3 TEE-HW with Isolated Root-of-Trust

The proposed TEE-HW SoC in Figure 4.15 was implemented in both FPGA and VLSI with the default configuration. That means the TEE processors are the dual-core of Rocket and BOOM with the ISA of RV64GC; each core has 16-KB for each instruction cache and data cache. The size of the L2 cache is 512-KB. The isolated sub-system is included, and the IBex core has a 4-KB of the instruction cache. The PCIe connection is excluded. And all modules in the utility group and crypto-core group, as shown in Figure 4.15, are included.

The chosen FPGA is the Virtex-7 XC7VX485T Xilinx FPGA, and Table 5.5 gives the resources utilization by total and by different parts of the system. The entire design occupied 51.3% of the FPGA resources, and nearly half of it was because of the BOOM core with 22.86%. The 7.64% FPGA resources were for the Rocket core, almost one-third compared to the BOOM core. The whole isolated sub-system cost 2.02% FPGA resources, nearly a quarter compared to the Rocket-core. The crypto-cores needed a very few FPGA resources with 0.0451%, 0.51%, 1.1%, 1.86%, and 0.59% for the TRNG, Ed25519 base-point multiplication, Ed25519 sign, SHA3-512, and AES-128/256 modules, respectively.

TABLE 5.5: Performances of the proposed TEE SoC in Virtex-7 FPGA (XC7VX485T).

		BOOM	Rocket	IBex*	TRNG	
Slices	**Logic LUT**	66,525	24,817	7,465	198	
	Flip-Flop	44,520	12,312	3,253	21	
	Total LUT	111,045	37,129	9,793	219	
BRAM		62	63	12	0	
DSP block		36	15	4	0	
FPGA utilization (%)		22.86	7.64	2.02	0.0451	
Area overhead	**Rocket-based (%)**	299.08	100	26.38	0.59	
	Total-based (%)	44.57	14.9	3.93	0.0879	
		Ed25519		**SHA3**	**AES**	**Total**
		mul	**sign**			
Slices	**Logic LUT**	2,305	5,344	8,881	2,710	149,765
	Flip-Flop	3,767	4,630	2,825	2,860	99,411
	Total LUT	2,465	5,344	9,013	2,842	249,176
BRAM		4	0	0	0	283
DSP block		16	0	0	0	71
FPGA utilization (%)		0.51	1.1	1.86	0.59	51.3
Area overhead	**Rocket-based (%)**	6.64	14.39	24.28	7.65	671.11
	Total-based (%)	0.99	2.15	3.62	1.14	100

**Including the isolated sub-system.*

TABLE 5.6: Area synthesis results of the proposed TEE SoC in ROHM-180nm technology.

	Cell-count (NAND2)	**Cell-area**	
		μm^2	%
Total system	666,957	63,163,554	100.00
BOOM	362,038	18,745,262	29.68
core	142,090	3,615,820	5.72
dcache	20,722	4,171,936	6.61
icache	170,942	9,955,331	15.76
fpu	18,103	434,456	0.69
Rocket	94,663	6,673,530	10.57
core	23,290	593,378	0.94
dcache	10,584	2,773,067	4.39
icache	17,848	2,283,889	3.62
fpu	37,864	880,321	1.39
IBex*	25,508	2,641,567	4.18
core	22,113	1,234,939	1.96
L2 cache	41,072	30,663,467	48.55
Ed25519	58,559	2,336,819	3.70
sign	25,380	630,950	1.00
mul	26,464	1,488,695	2.36
SHA3	26,130	650,604	1.03
AES	16,325	412,932	0.65
BootROM	7,465	120,438	0.19
TRNG	268	3,984	0.0063

**Including the isolated sub-system.*

For the VLSI implementation, the TEE SoC was also synthesized by a typical bulk process; the ROHM-180nm process library was chosen, and Table 5.6 and Table 5.7 gives the results of the system together with its submodules. The synthesis results were reported by using the Cadence's Genus tool version 18.13. According to the table, the L2 cache cost the most die area with 48.55%, nearly half of the chip, while consuming just 5.84% total power. For the most power consumption, the BOOM tile, including the core, two caches, and its floating-point unit, achieved 54.35%, more than half of the total power, while occupied only 29.68% of the chip area. The Rocket tile reached a fair value of 10.57% and 15.66% for the total area and power consumption, respectively. The IBex tile was relatively small, with 4.18% and 2.14% for the total size and power consumption, respectively. Compared to the Rocket tile, the area and power overheads of the whole isolated subsystem were 39.58% and 13.64%, respectively.

TABLE 5.7: Power consumption synthesis results of the proposed TEE SoC in ROHM-180nm technology.

	Power			
	Leakage (nW)	**Dynamic (μW)**	**Total (μW)**	**%**
Total system	8,725.42	2,121,323	2,121,332	100.00
BOOM	4,890.36	1,152,855	1,152,860	54.35
core	1,668.91	532,203	532,205	25.09
dcache	327.11	60,191	60,191	2.84
icache	2,549.61	501,332	501,335	23.63
fpu	203.45	31,607	31,607	1.49
Rocket	1,138.11	332,103	332,104	15.66
core	288.90	67,958	67,959	3.20
dcache	138.99	30,684	30,684	1.45
icache	260.58	49,793	49,793	2.35
fpu	368.97	166,620	166,620	7.85
IBex*	265.06	45,291	45,291	2.14
core	203.98	32,507	32,508	1.53
L2 cache	648.99	123,889	123,889	5.84
Ed25519	783.50	240,425	240,425	11.33
sign	311.78	65,797	65,797	3.10
mul	341.72	154,598	154,598	7.29
SHA3	276.19	31,792	31,792	1.50
AES	205.29	37,515	37,515	1.77
BootROM	47.71	2,889	2,889	0.14
TRNG	1.76	133	133	0.0063

**Including the isolated sub-system.*

5.2 TEE Boot Performance

To test the performance of the TEE boot flow, the FPGA implementation was run with pure-software and hardware-accelerated FSBL programs. The SHA3, Ed25519-multiplier, and Ed25519-sign crypto-cores push the bootloader payload for hashing, keys generation, and signing. Several payload sizes were run in the same setting to measure the execution times. Figure 5.1 presents the results of software versus hardware; both were run with the system clock of 100-MHz. The chosen FPGA board for testing is the Altera DE4 FPGA. The payload includes the Linux bootloader, the Linux kernel, and the initial filesystem. The payload size was changed by expanding the initial filesystem with random data in the image file. For both software and hardware implementations, the time increments exponentially. For any stream size, the execution time on the hardware-accelerated version decreased about 2.5-decades compared to the pure-software approach.

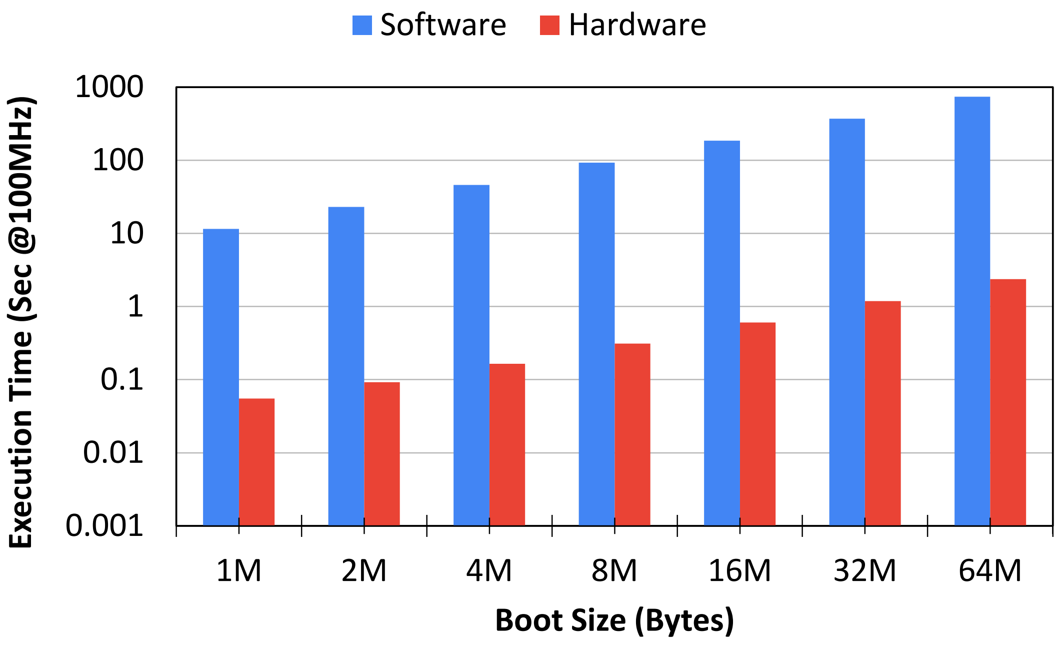


FIGURE 5.1: Comparison between software and hardware implementation of the TEE boot including SHA3 and Ed25519 accelerations.

The TEE boot flow presented in Figure 4.14 mainly used the SHA3 hashing, leaving the Ed25519 genkey and sign as a new calculation process. Therefore, Table 5.8 was made to offer a broader perspective on the individual processes of genkey and sign. According to the table, the all software approach presents a significant execution time in the case of the signing procedure. The main reason is that the payload needs to be hashed twice in the sign program. On the other hand, for the genkey, replacing software SHA3 with hardware SHA3 did not impact as much time as it did in the signing process. It is because, in the genkey process, the hash was performed only on the 256-bit stream. Comparing the second column with the third column in Table 5.8, we see nearly the same portion of decrements, about 90%, when the hardware equivalent replaced the software Ed25519.

TABLE 5.8: Execution times for genkey and sign processes.

2MB bootloader	**Software SHA3 Software Ed25519**	**Hardware SHA3 Software Ed25519**	**Hardware SHA3 Hardware Ed25519**
genkey (ms)	109.5	93.4	4.6
sign (ms)	231,019	82.6	4.7

5.3 System-on-Chip Implementation

5.3.1 Design

Finally, the proposed architecture in Figure 4.15 was made into two ROHM-180nm chips, one with 64-bit dual-core Rocket-BOOM and one with 32-bit dual-core Rocket-BOOM. To better fit in chips, several things were reduced compared to the architecture shown in Figure 4.15. The PCIe connection was excluded, the DDR controller was cut, and the MBus was exported as chip's I/Os. All the peripherals shown in the utility and crypto-core groups are included. The isolated sub-system is included. The 512-KB L2 cache was removed, and the 16-KB of L1 caches were reduced to 4-KB. Two chips have dual-core processors with Rocket core first and BOOM core second. The 64-bit TEE processors version was made into a 5.0×7.5-mm^2 chip, and the 32-bit TEE processors version was made into a 5.0×5.0-mm^2 chip. Figure 5.2 and Figure 5.3 show the floorplans of the two chips, and Figure 5.4 and Figure 5.5 give the layouts of them. Barechip images are shown in Figure 5.6 and Figure 5.7 with proper notes for each sub-module. The two chips were made using the Cadence design flow with Genus tool for synthesis and Innovus tool for PnR. Table 5.9 gives the results reported by the synthesis tools.

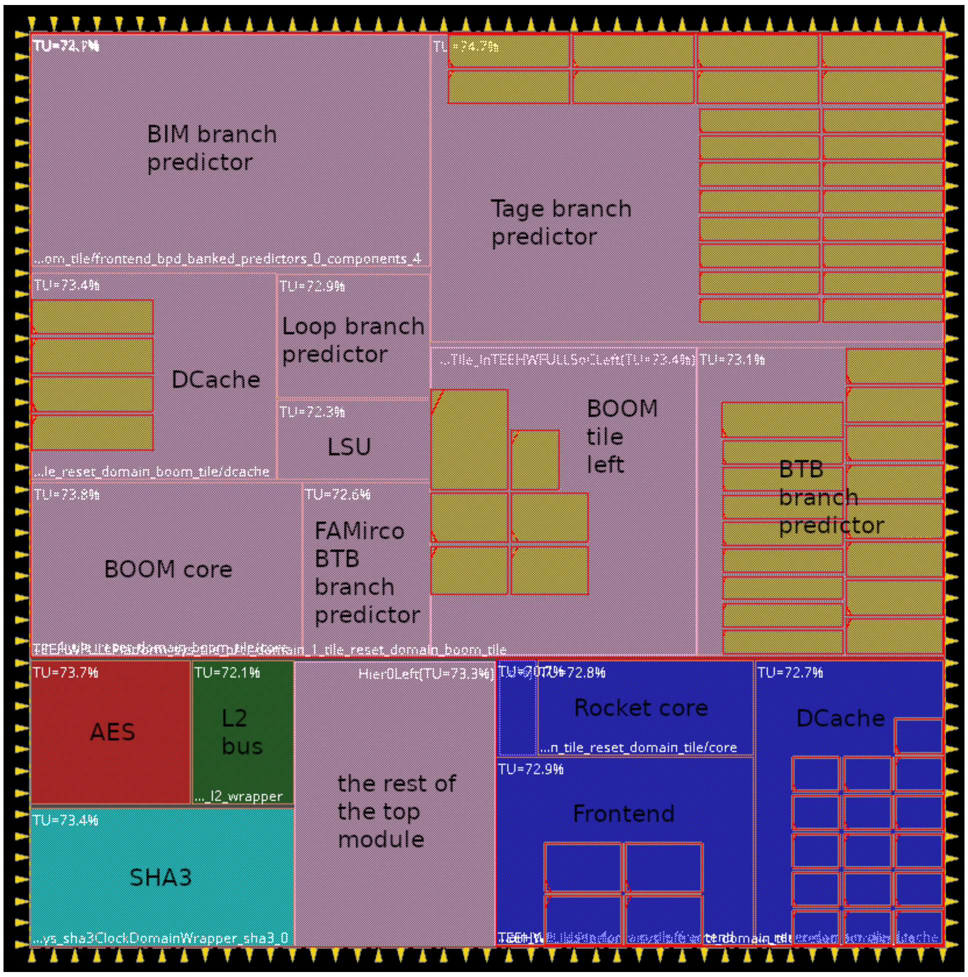


FIGURE 5.2: Floorplan of the ROHM-180nm 32-bit dual-core Rocket-BOOM plus the isolated sub-system (5.0×5.0-mm^2).

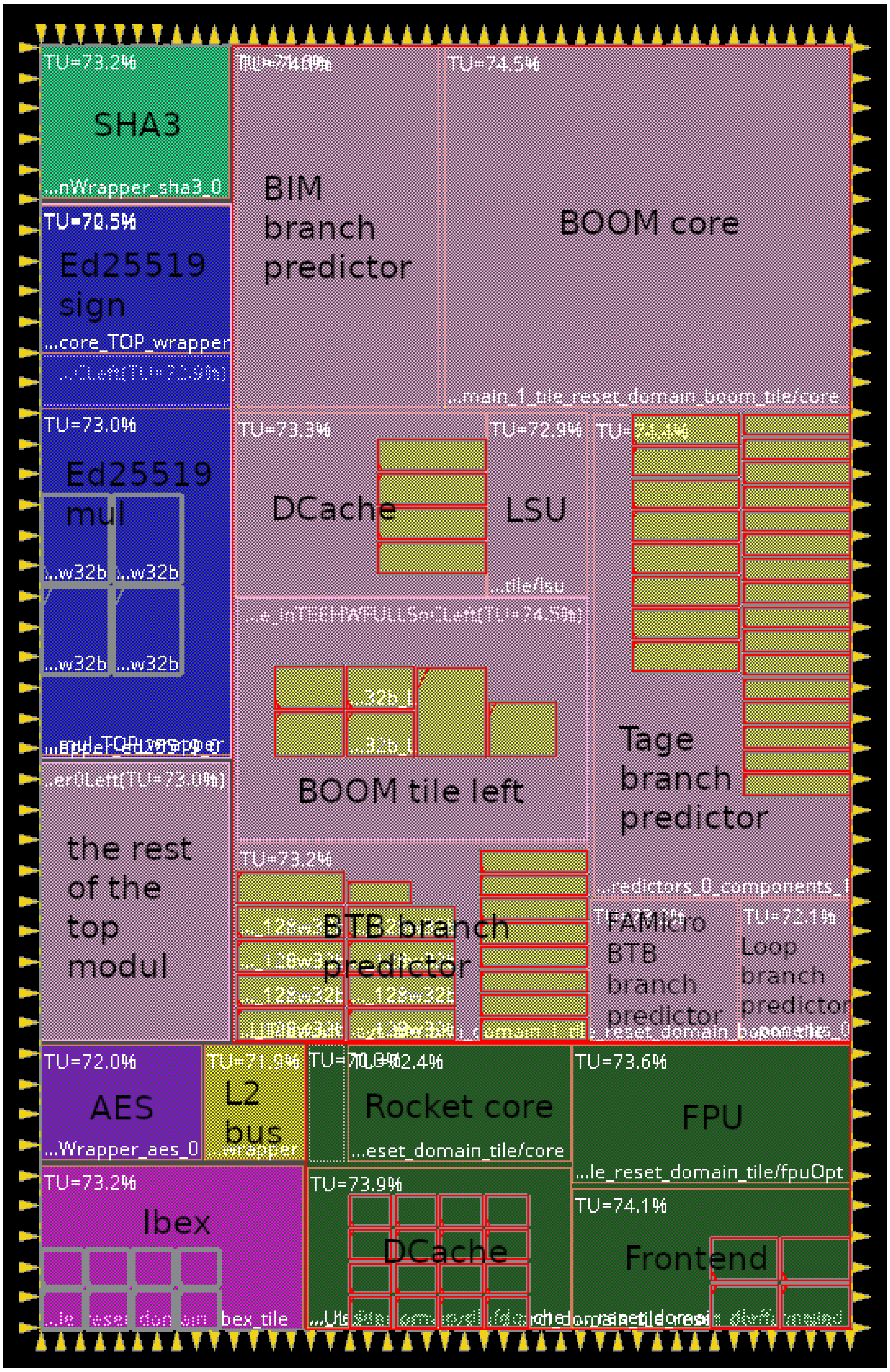


FIGURE 5.3: Floorplan of the ROHM-180nm 64-bit dual-core Rocket-BOOM plus the isolated sub-system (5.0×7.5-mm^2).

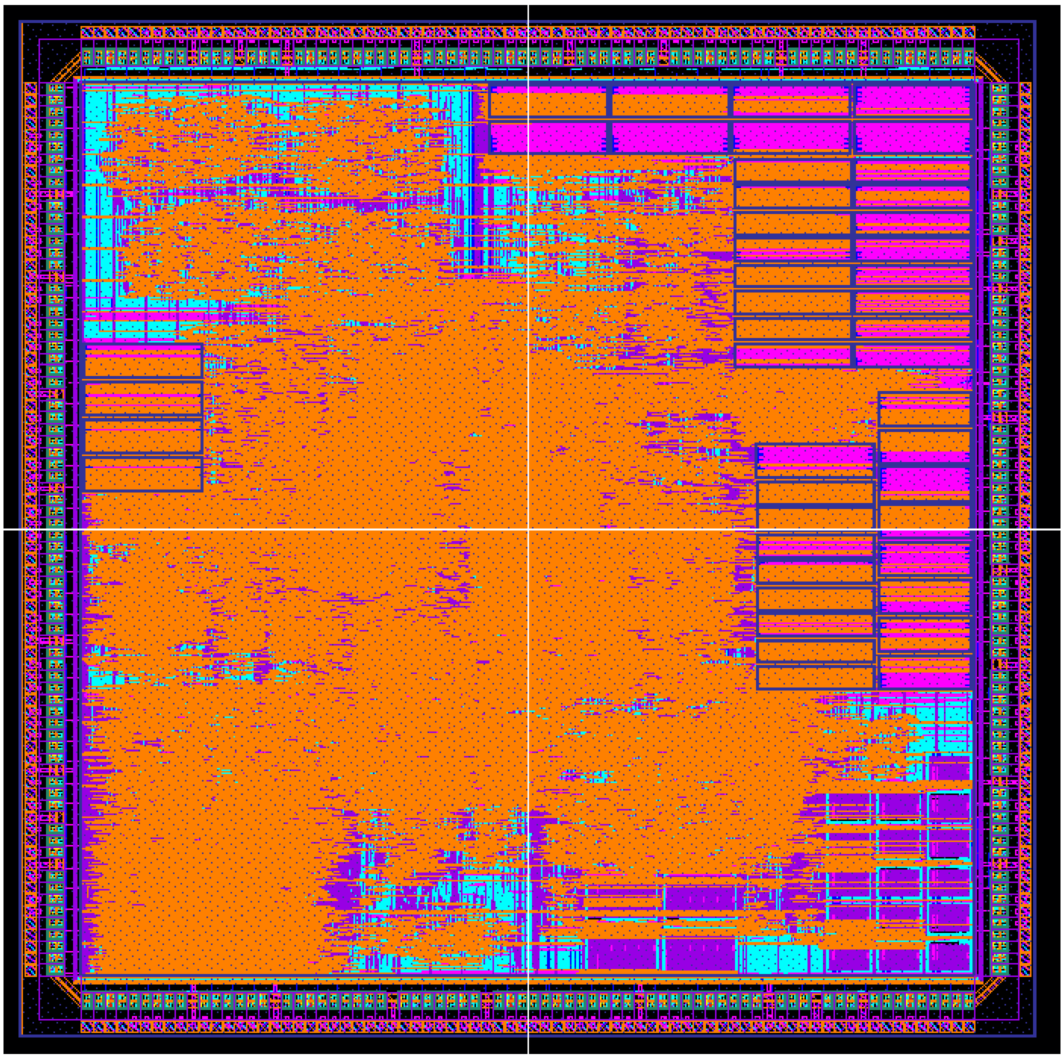


FIGURE 5.4: Layout of the ROHM-180nm 32-bit dual-core Rocket-BOOM plus the isolated sub-system (5.0×5.0-mm^2).

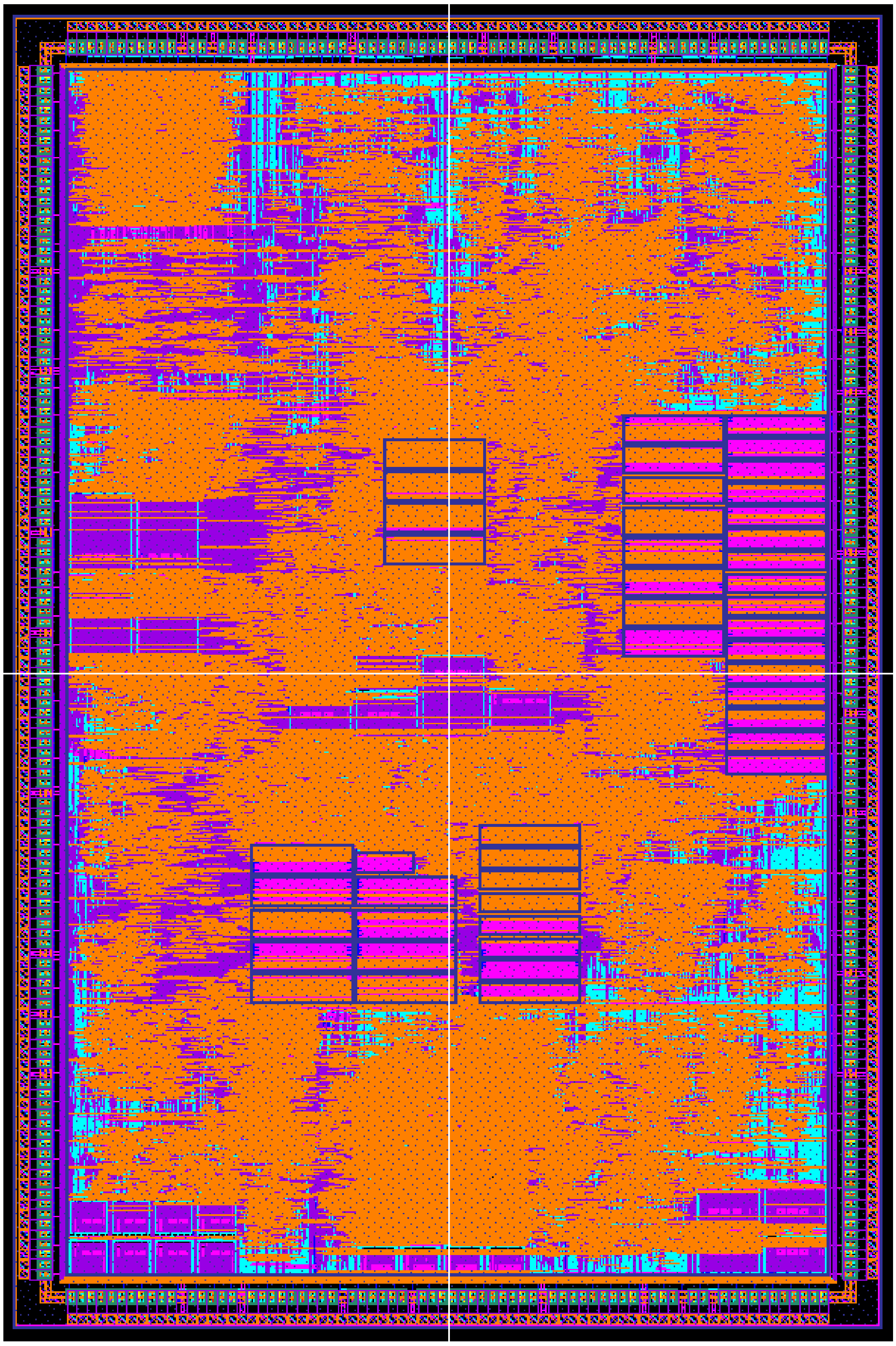


FIGURE 5.5: Layout of the ROHM-180nm 64-bit dual-core Rocket-BOOM plus the isolated sub-system (5.0×7.5-mm^2).

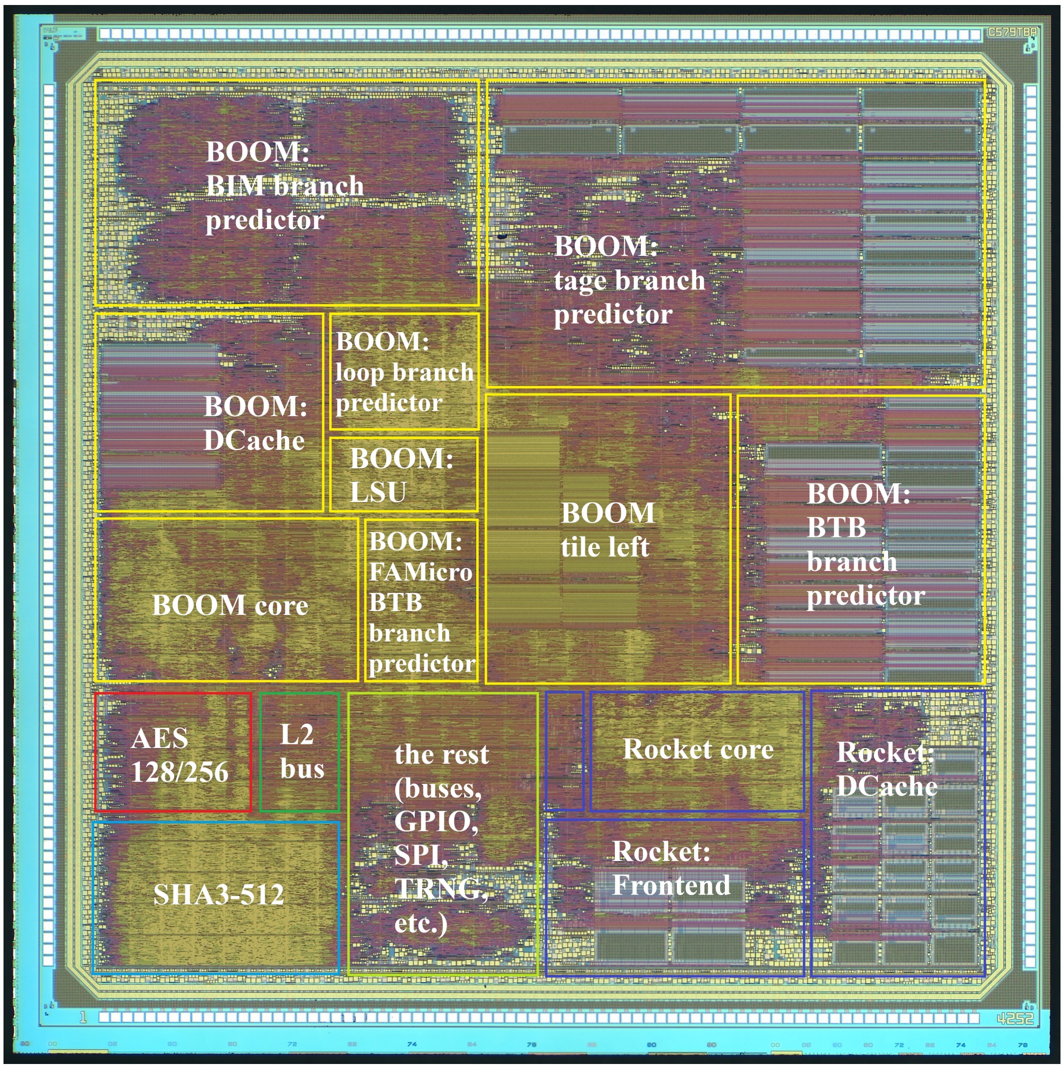

BOOM: BIM branch predictor
BOOM: tage branch predictor
BOOM: DCache
BOOM: loop branch predictor
BOOM: LSU
BOOM tile left
BOOM: BTB branch predictor
BOOM core
BOOM: FAMicro BTB branch predictor
AES 128/256
L2 bus
the rest (buses, GPIO, SPI, TRNG, etc.)
Rocket core
Rocket: DCache
SHA3-512
Rocket: Frontend

FIGURE 5.6: Barechip image of the ROHM-180nm 32-bit dual-core Rocket-BOOM plus the isolated sub-system (5.0×5.0-mm^2).

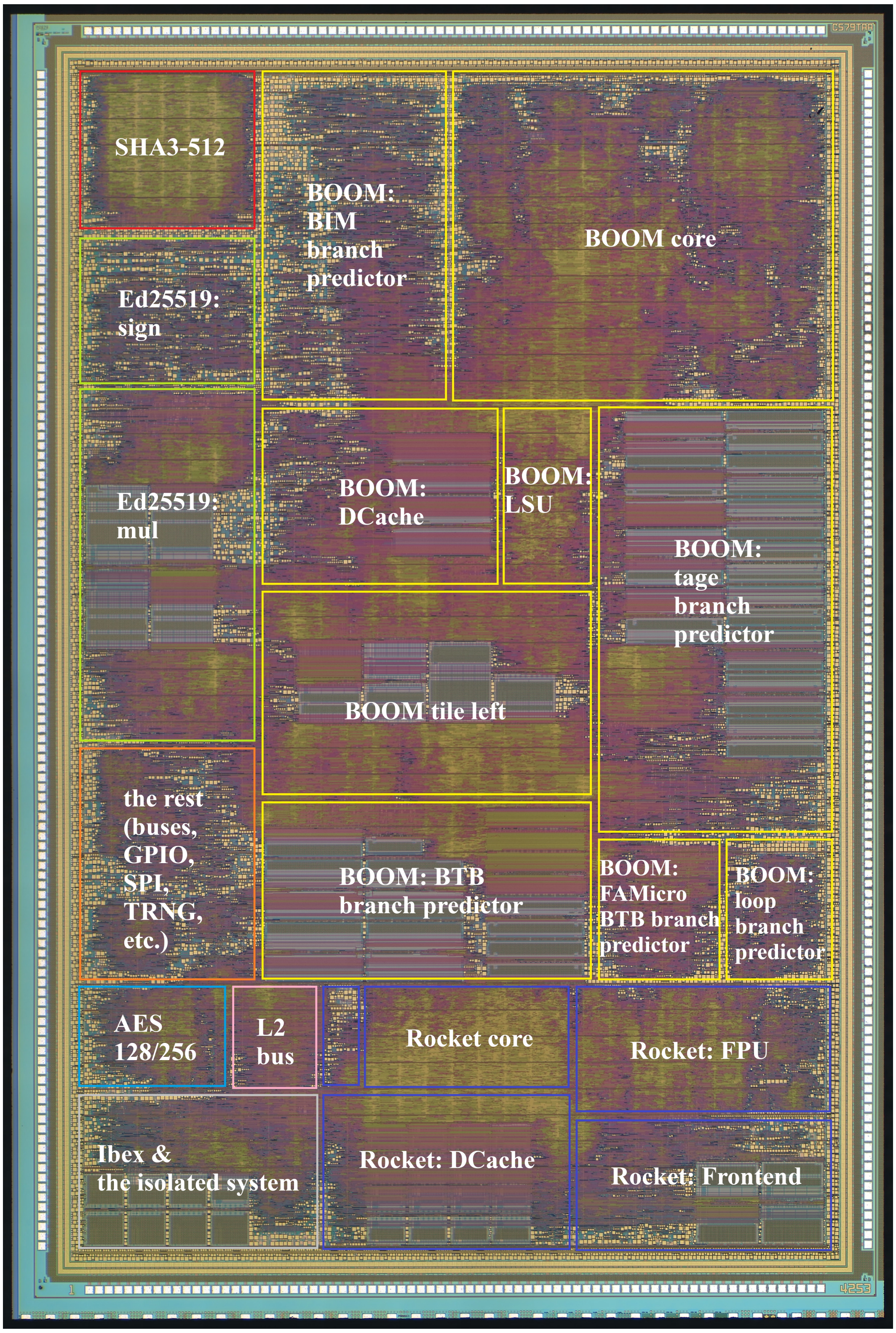

SHA3-512
BOOM: BIM branch predictor
BOOM core
Ed25519: sign
Ed25519: mul
BOOM: DCache
BOOM: LSU
BOOM: tage branch predictor
BOOM tile left
the rest (buses, GPIO, SPI, TRNG, etc.)
BOOM: BTB branch predictor
BOOM: FAMicro BTB branch predictor
BOOM: loop branch predictor
AES 128/256
L2 bus
Rocket core
Rocket: FPU
Ibex & the isolated system
Rocket: DCache
Rocket: Frontend

FIGURE 5.7: Barechip image of the ROHM-180nm 64-bit dual-core Rocket-BOOM plus the isolated sub-system (5.0×7.5-mm^2).

TABLE 5.9: VLSI synthesis reports of the two ROHM-180nm chips, the design of dual-core Rocket-BOOM TEE-HW system with isolated architecture.

Process		ROHM-180nm (1.8-V @ 100-MHz)	
Cores		Rocket-BOOM and IBex	Rocket-BOOM and IBex
ISA		RV32IMAC	RV64IMAFDC
Caches	**Instruction**	4-KB	4-KB
	Data	4-KB	4-KB
	L2	none	none
Area	**Die**	5.0×5.0-mm^2	5.0×7.5-mm^2
	Core	$4{,}559.52\times4{,}556.16$-$\mu$m^2 = 14.96-mm^2 $\approx$ 1,545,599-NAND2	$4{,}559.52\times7{,}217.28$-$\mu$m^2 = 23.4-mm^2 $\approx$ 2,418,005-NAND2
	Cell	360,339	631,597
	MOSFET	7,332,462	11,722,810
Density		72.64%	71.62%
Power (mW)	**@Synthesis**	1,098.74	2,084.05
	@PnR	846.5	1,378
F_{Max} (MHz)	**@Synthesis**	26	10
	@PnR	50	50

5.3.2 Measurement Results

Two ROHM-180nm chips were made for the demonstration, one with 32-bit and one with 64-bit TEE processors. Because they are different only in the ISA, their pinouts are identical. Hence, both chips were packaged in the same 257-pin PGA with the same pinouts to save resources. As a result, there is only one PCB needed for testing the chips. Figure 5.8 shows the testing PCB with a chip inside its socket. For better stability, the critical peripherals such as SD-card, UART, and flash were designed in PMOD headers and attached with small circuits outside. Similar to the previous PCBs, this PCB also can choose a power supply and clock source. The power and clock can be provided by either the FPGA or from external sources via jumpers. Figure 5.9 shows the working PCB mounting on the TR4 FPGA board to use the FPGA's DIMM RAM.

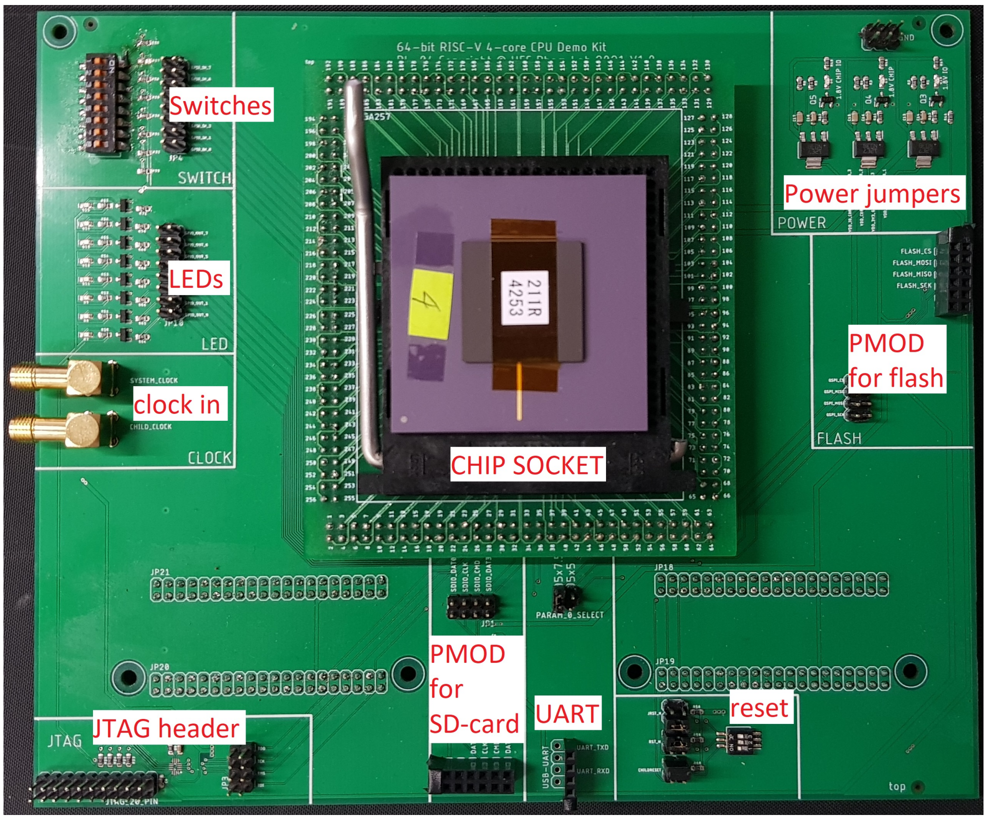


FIGURE 5.8: PCB for testing the two TEE-HW with isolated architecture ROHM-180nm chips.

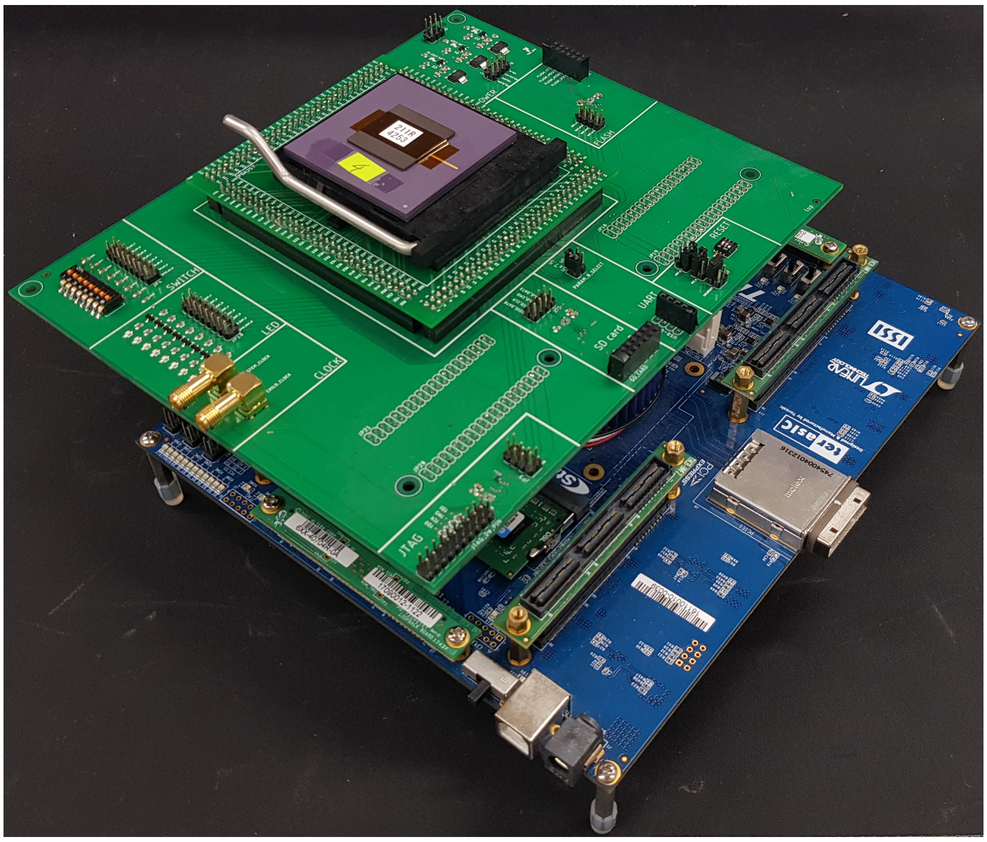


FIGURE 5.9: The TEE-HW with isolated architecture PCB mounts on the TR4 FPGA board.

In ROHM-180nm technology, the default V_{TH} is about 1.0-V and the recommended operating V_{DD} is 1.8-V. Therefore, the ROHM-180nm chip measurement was done with the V_{DD} range of 1.0-V to 2.0-V. Figure 5.10 shows the changes in F_{Max} with V_{DD} for the 32-bit 5.0×5.0-mm^2 version. We can see that the F_{Max} performances increased almost linear with V_{DD} increment, about 5.75-MHz per 0.1-V of V_{DD} increment. The lowest and highest F_{Max} were 36-MHz and 82-MHz achieved at 1.2-V and 2.0-V V_{DD}, respectively. Figure 5.11 and Figure 5.12 show the P_{Active} and P_{Active}/F_{Max} of the 5.0×5.0-mm^2 chip, respectively. In general, their changes were almost linear, with roughly about 58.9-mW (0.51-mW/MHz) per 0.1-V V_{DD}. The lowest and highest P_{Active} were 105.47-mW (2.93-mW/MHz) and 576.68-mW (7.03-mW/MHz) at 1.2-V and 2.0-V of V_{DD}, respectively.

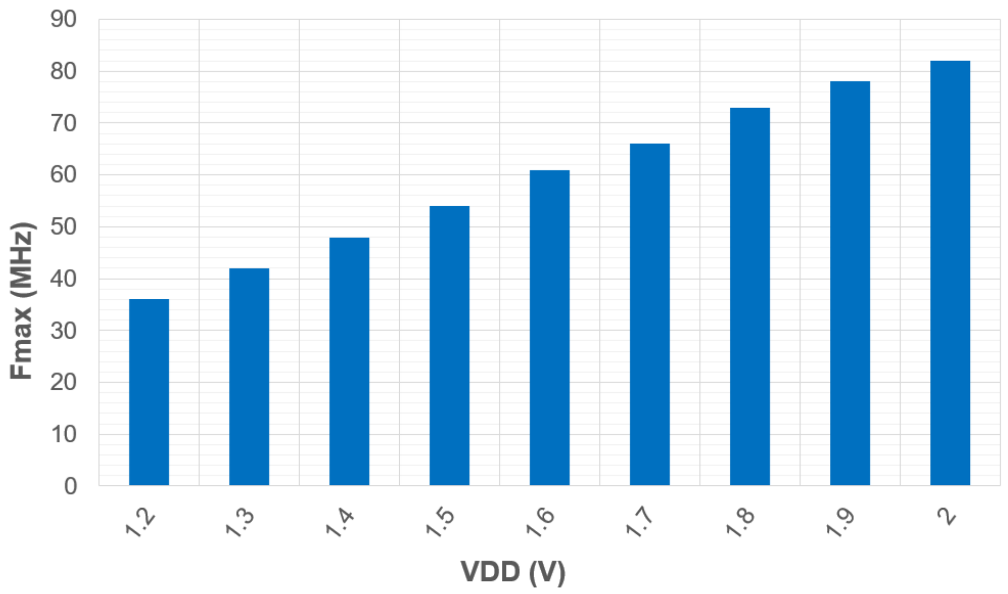


FIGURE 5.10: ROHM-180nm TEE-HW with isolated RoT, 32-bit version (5.0×5.0-mm^2): F_{Max} vs. supply voltages.

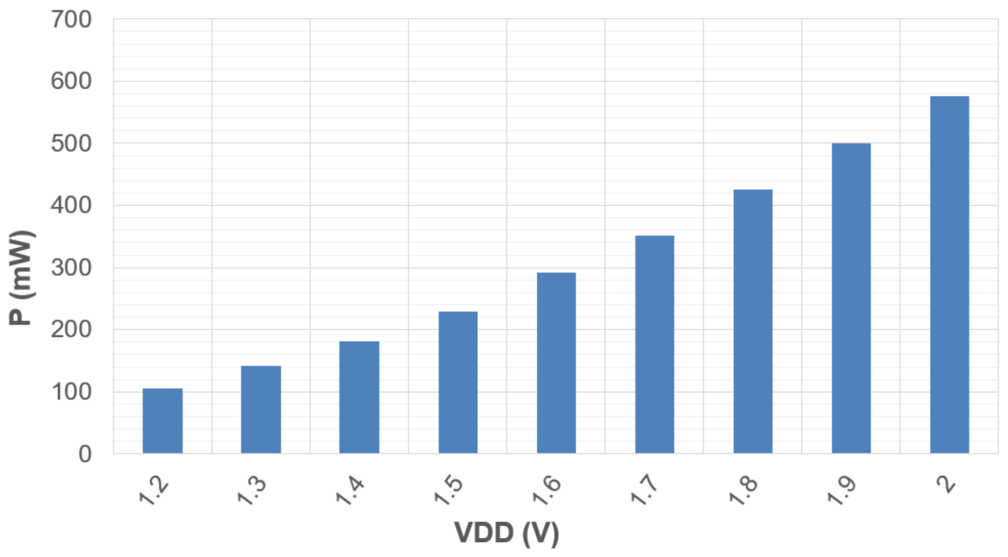


FIGURE 5.11: ROHM-180nm TEE-HW with isolated RoT, 32-bit version (5.0×5.0-mm^2): P_{Active} vs. supply voltages.

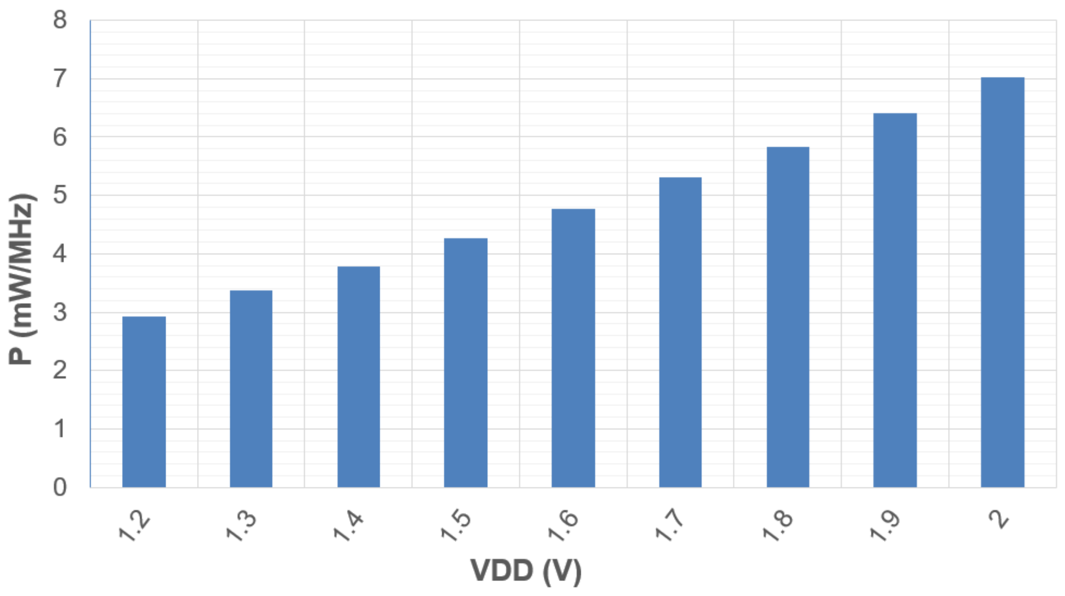


FIGURE 5.12: ROHM-180nm TEE-HW with isolated RoT, 32-bit version (5.0×5.0-mm^2): P_{Active}/F_{Max} vs. supply voltages.

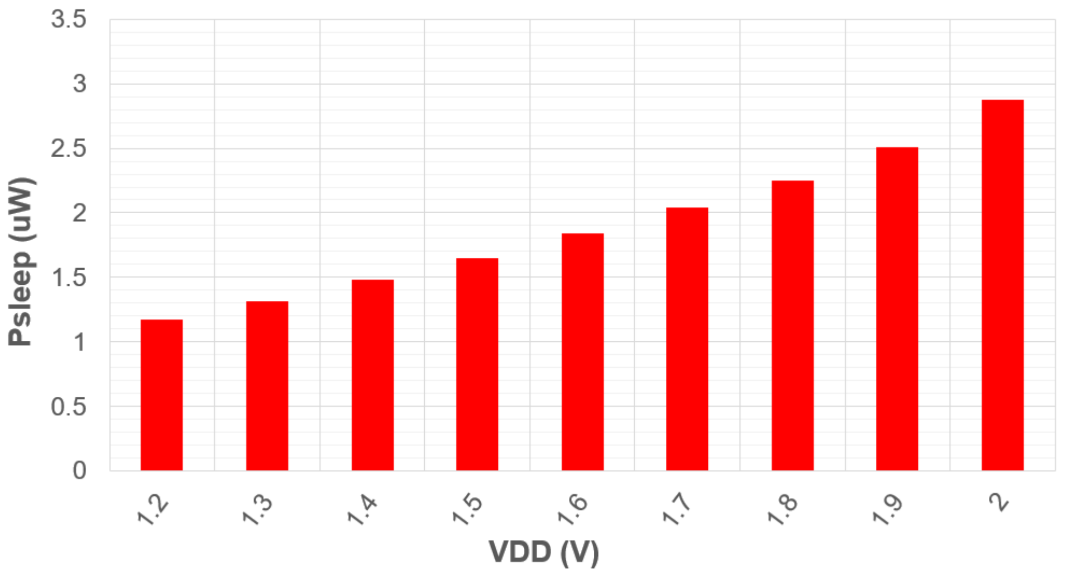


FIGURE 5.13: ROHM-180nm TEE-HW with isolated RoT, 32-bit version (5.0×5.0-mm^2): P_{Sleep} vs. supply voltages.

Figure 5.13 shows the leakage power of the 5.0×5.0-mm^2 chip in the changes with V_{DD}. The sleep-state was measured when the clock was cut off. From the figure, the changes is nearly linear, with about 0.21-μW per 0.1-V V_{DD}. The lowest and highest leakage power consumption were 1.17-μW and 2.88-μW at 1.2-V and 2.0-V V_{DD}, respectively.

For the 64-bit version of the chip, the same measurement was done with the V_{DD} range of 1.0-V to 2.0-V. Figure 5.14 shows the changes in F_{Max} with V_{DD} for the 5.0×7.5-mm^2 version. We can see that the F_{Max} performances increased almost linear with V_{DD} increment, about 3.25-MHz per 0.1-V of V_{DD} increment. The lowest and highest F_{Max} were 27-MHz and 53-MHz achieved at 1.2-V and 2.0-V V_{DD}, respectively. Figure 5.15 and Figure 5.16 show the P_{Active} and P_{Active}/F_{Max} of the 64-bit chip, respectively. In general, their changes were almost linear, with roughly about 61.06-mW (0.87-mW/MHz) per 0.1-V V_{DD}. The lowest and highest P_{Active} were 123.71-mW (4.58-mW/MHz) and 612.18-mW (11.55-mW/MHz) at 1.2-V and 2.0-V of V_{DD}, respectively.

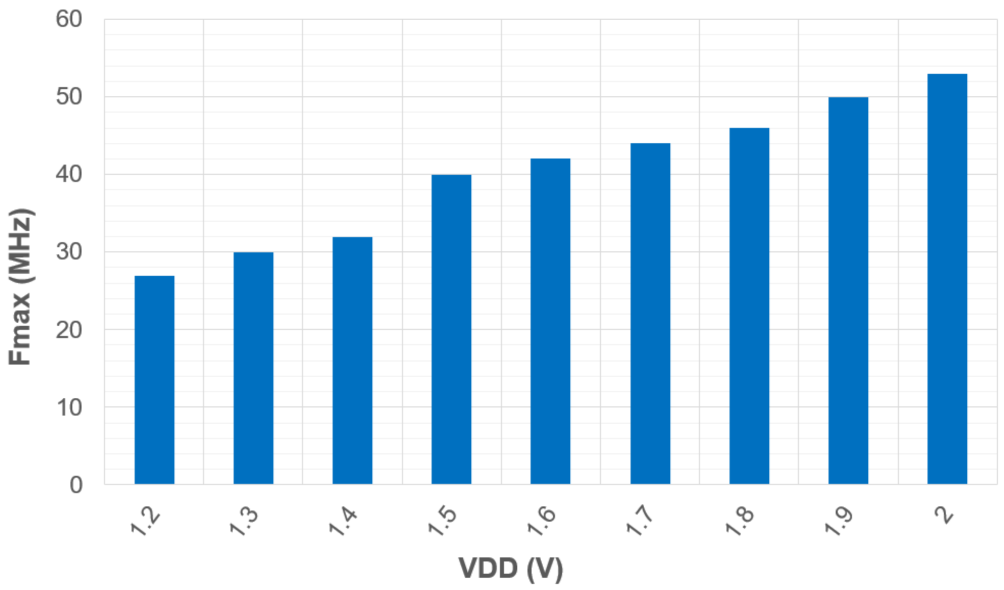


FIGURE 5.14: ROHM-180nm TEE-HW with isolated RoT, 64-bit version (5.0×7.5-mm^2): F_{Max} vs. supply voltages.

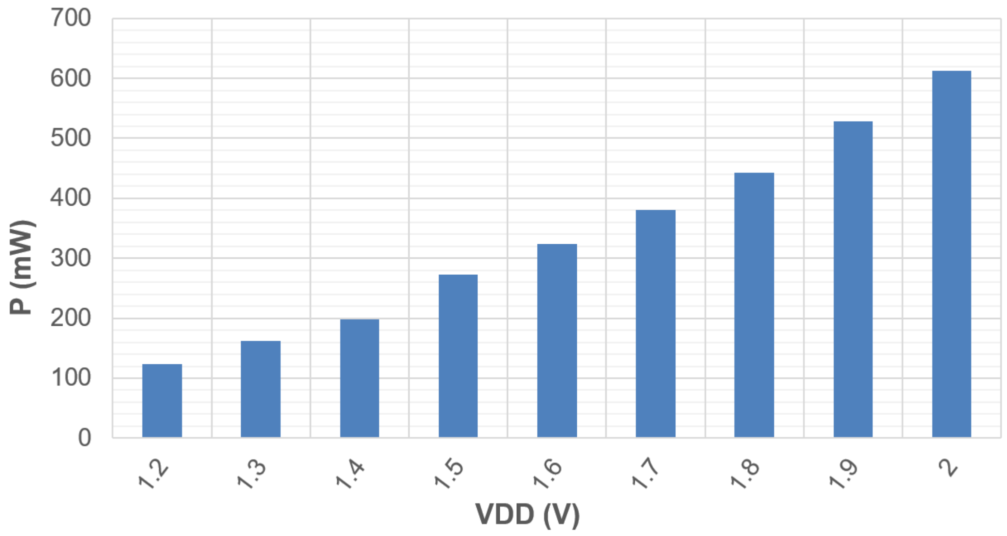


FIGURE 5.15: ROHM-180nm TEE-HW with isolated RoT, 64-bit version (5.0×7.5-mm^2): P_{Active} vs. supply voltages.

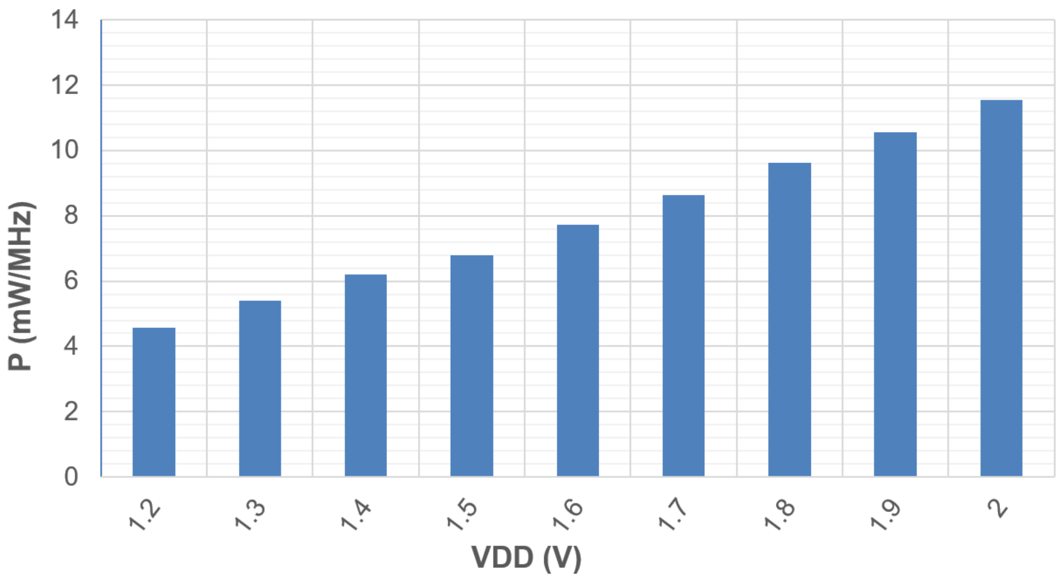


FIGURE 5.16: ROHM-180nm TEE-HW with isolated RoT, 64-bit version (5.0×7.5-mm^2): P_{Active}/F_{Max} vs. supply voltages.

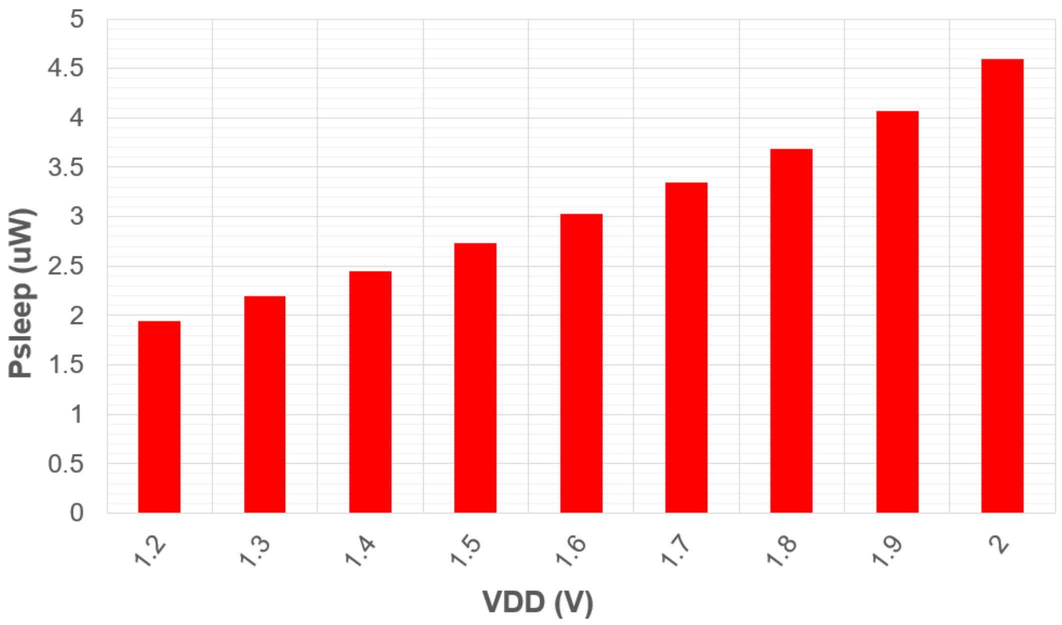


FIGURE 5.17: ROHM-180nm TEE-HW with isolated RoT, 64-bit version (5.0×7.5-mm^2): P_{Sleep} vs. supply voltages.

Figure 5.17 shows the leakage power of the 64-bit TEE-HW with isolated architecture chip. From the figure, the changes is nearly linear, with about 0.33-μW per 0.1-V V_{DD}. The lowest and highest leakage power consumption were 1.94-μW and 4.6-μW at 1.2-V and 2.0-V V_{DD}, respectively. For a better comparison between the two measurements, key features of the two chips are summarized in Table 5.10.

TABLE 5.10: TEE-HW with isolated architecture chips' features summary.

Process		ROHM-180nm	
Cores		Rocket-BOOM and IBex	Rocket-BOOM and IBex
ISA		RV32IMAC	RV64IMAFDC
Caches	**Instruction**	4-KB	4-KB
	Data	4-KB	4-KB
	L2	none	none
Area	**Die**	5.0×5.0-mm^2	5.0×7.5-mm^2
	Core	$4{,}559.52\times4{,}556.16$-$\mu$m^2 = 14.96-mm^2 $\approx$ 1,545,599-NAND2	$4{,}559.52\times7{,}217.28$-$\mu$m^2 = 23.4-mm^2 $\approx$ 2,418,005-NAND2
	Cell	360,339	631,597
	MOSFET	7,332,462	11,722,810
V_{DD}	**I/O**	1.8-V	1.8-V
	Core	1.2-V to 2.0-V	1.2-V to 2.0-V
Peak performance		at 2.0-V V_{DD} F_{Max}=82-MHz P_{Active}=7.03-mW/MHz	at 2.0-V V_{DD} F_{Max}=53-MHz P_{Active}=11.55-mW/MHz
Best power density		at 1.2-V V_{DD} F_{Max}=36-MHz P_{Active}=2.93-μW/MHz	at 1.2-V V_{DD} F_{Max}=27-MHz P_{Active}=4.58-μW/MHz

5.4 Comparison and Discussion

5.4.1 Comparison

For the comparison between cores, the Drystone test was conducted for both IBex core and Rocket core, and Table 5.11 gives the experimental results. Because the ISA of the IBex is RV32IMC, the test on the Rocket core was repeated by the same ISA. The Dhrystone test was run 500 times, and the average values were recorded. According to Table 5.11, the Rocket core achieved 1.573 to 1.713-DMIPS/MHz, which is a good result compared to an average processor [136]. The IBex core scored 0.434-DMIPS/MHz, which falls into the mid-range MCUs [137]. The DMIPS/MHz results of the Rocket core were about 3.62× to 3.95× compared to that of the IBex core. Although the IBex core is much slower than the Rocket, considering it still can use crypto-cores to accelerate its boot program, the boot speed when swapping a Rocket for an IBex is not much of a change.

TABLE 5.11: Dhrystone tests comparison between IBex and Rocket cores.

Core	ISA	Dhrystone/s	DMIPS/MHz	Changes
Rocket	RV64GC	150,511	1.713	3.95×
	RV32IMC	138,197	1.573	3.62×
IBex	RV32IMC	38,165	0.434	1.00×

TABLE 5.12: Area comparison with recent RISC-V-based TEE SoCs that support secure boot process.

Design		Registers Overhead (+%)	LUTs Overhead (+%)
This work (2021)	Baseline: Dual-Rocket	24,624	74,258
	+ IBex[1]	+3,253 (13.21%)	+9,793 (13.19%)
	+ crypto-cores[2]	+14,103 (52.27%)	+19,883 (26.78%)
	+ IBex[1] + crypto-cores[2]	+17,356 (70.48%)	+29,676 (39.96%)
ITUS [11, 12] (2019)	Baseline: Dual-Rocket	24,624	74,258
	+ CAU	+6,722 (27.30%)	+27,170 (36.59%)
	+ KMU	+3,344 (13.58%)	+29,529 (39.77%)
	+ CAU + KMU	+10,066 (40.88%)	+56,699 (76.35%)
HECTOR-V [9] (2021)	Baseline: Single-lowRISC	N/A	55,443
	with RI5CY		+8,205 (14.80%)
	with Remus		+11,581 (20.89%)
	with Frankenstein		+13,303 (23.99%)

[1]*Including the isolated sub-system.*
[2]*Including SHA-3, AES, Ed25519, and TRNG.*

Table 5.12 shows the FPGA results compared to recent similar implementations, and Table 5.13 gives the comparison regarding the security and flexibility features. For details about ITUS, WorldGuard, HECTOR-V, and CURE architectures, please refer to the section 2.3.

TABLE 5.13: Comparison with recent security-driven RISC-V-based SoCs, regarding the security and flexibility features; ●, ◐, and ○ rank the performance from best to worst, respectively.

	CURE [8]	HECTOR-V [9]	WorldGuard [10]	ITUS [11, 12]	This work
Open-source	○	○	◐	○	◐
Secure boot	◐	●	◐	●	🔴
Flexible boot process	●	●	●	○	🔴
TEE & secure boot iso.	○	○	○	●	🔴
Exclusive TEE processor	◐	●	◐	○	○
Exclusive secure storage	○	●	○	●	🔴
Secure I/O paths	●	●	◐	○	○
Crypto. accel.	○	○	◐	●	●
SCA resilience	●	●	◐	○	○
Hardware cost	●	◐	●	○	◐
High expressiveness	◐	●	◐	○	◐
Low porting efforts	○	○	◐	●	●

The comparison keys used in Table 5.13 are:

- **Open-source.** The source codes are available for modification.
- **Secure boot.** Has a clear secure boot process with RoT.
- **Flexible boot process.** The boot process is flexible and can be updated.
- **TEE and secure boot isolation.** The secure boot process is done by the TEE processor or by another party?
- **Exclusive TEE processor.** TEE is executed by an exclusive processor or sharing processors with the REE?
- **Exclusive secure storage.** The proposed system has a safe place to store the root key(s) and the first bootloader. The storage could be ROM or otherwise, as long as it is inaccessible for the application processors after boot.
- **Secure I/O paths.** The proposed architecture could bind a peripheral to an isolated environment and make a direct secure connection.
- **Cryptographic accelerators.** Hardware accelerators for cryptographic functions are provided in the system.
- **SCA resilience.** The ability to prevent some SCAs.
- **Hardware cost.** The amount of resources that the proposed architecture needs, compared in the overhead ratio. Low is around 5-10%, medium is around 20%-25%, and high is around 50%.
- **High expressiveness.** The proposed architecture allows multi/hyper-threading, flexible isolation enforcement, memory encryption, cache partitioning, and so on.
- **Low porting effort.** The development effort for adapting/porting the proposed design.

In ITUS [11,12], they try to solve the secure boot in TEE by a pure hardware approach. The new hardware modules are introduced, Code Authentication Unit (CAU) and Key Management Unit (KMU). The KMU contains a PUF and a TRNG for keys generation and distribution. The CAU handles program authentication by using an ECDSA and an SHA-3 to sign and verify. Because their solution is based solely on hardware, their approach would not be flexible. On the other hand, the IBex's program could do any cryptographic functions given in [11,12]. The crypto-cores in our system are not necessary for a secure boot; they just help accelerate the program. On the other hand, due to the complete hardware approach, ITUS has to realize all cryptographic functions needed in the boot process, thus increasing the resources significantly when the complexity of those functions goes up. For the comparison in Table 5.12, if the IBex sub-system is compared to CAU or KMU, our work will achieve the least expense. If the crypto-cores are also included in the comparison, our resources would be approximately equal to ITUS with CAU + KMU. Because ITUS has a similar goal with ours, which is to remove the RoT and boot process from the TEE domain, the SCAs in ITUS and ours are considered out of scope.

In WorldGuard [10], a security scheme with IDs is implemented across the entire system for solid isolation. The goal of WorldGuard is to strengthen the existing TEEs, not to provide a secure boot process like ours. For the secure boot, WorldGuard utilizes the conventional boot flow with root keys and the first bootloader pre-stored in ROM at the time manufactured. The bootloader is then executed first to verify and load the SM into the working RAM. Therefore, the RoT and the boot program are still in the TEE domain. Hence, the attack surface in WorldGuard still exists. Although WorldGuard opens its bootloader program, its hardware is not opened. Thus, we cannot compare WorldGuard's sizes with our implementation.

In HECTOR-V [9], the design comes from two novel ideas, the heterogeneous architecture for separating REE and TEE domains, and the security-hardened TEE processor for SCAs resilience. In HECTOR-V, the secure boot process is done using the TEE core. Therefore, HECTOR-V's boot program is flexible and can be updated like our approach. Although the secure storage element is introduced that can be inaccessible from the REE domain after boot, the RoT and the boot program are still visible from the TEE's eyes. In contrast, our implementation ultimately moves the RoT and the secure boot process out of the TEE's domain, thus eliminating the potential threats from the malicious TEE's enclaves. Although the HECTOR-V's set goal is different from ours, we could still compare in terms of the secure boot's hardware requirements, as given in Table 5.12. According to the table, the IBex sub-system is smaller than HECTOR-V with Remus or HECTOR-V with Frankenstein but bigger than HECTOR-V with RI5CY.

In CURE [8], a new TEE model is proposed together with new hardware primitives for fine-tuning the security strength in TEE. The main goal of the CURE is a TEE model that can support multi-type enclaves while maintaining strong isolation between them. To achieve that, new hardware modifications are added across the system at many architectural levels, such as registers in the core, access controller in the system bus, and way-based partitioning in the shared cache. Although CURE tailors the hardware for increasing the TEE's security level, it considers the secure boot with RoT out of scope. In CURE, the RoT is assumed at reset, and the secure boot process is provided beforehand. Therefore, besides security features, we cannot include the CURE resources in the comparison in Table 5.12.

5.4.2 Security Analysis

The goals of the proposed TEE-HW are: (i) accelerating the boot process by using crypto-cores and (ii) isolating the boot program with RoT from the TEE processors. The used TEE model in the implementation is Keystone [7]; thus, the proposed design inherits all of the Keystone's advantages and disadvantages. For more detail on the Keystone TEE, please refer to section 2.2.7.

To summarize Keystone, the threat model of Keystone considers a strong software adversary that can compromise all of the software stack and a strong physical adversary that can corrupt peripherals and memory communications. A malicious enclave is also considered in the Keystone model. At S-mode, the Eyrie runtime provides the OS-equivalent services and ensures the validity of address mappings, thus preventing mapping attacks [138]. Furthermore, thanks to the runtime, the enclaves don't have to rely on the OS for critical functions; hence, they can defend against controlled SCAs that exploit the sharing states across domains, like interrupt handlers and table paging. At M-mode, the PMP feature defines the memory access rights. Therefore, any direct attempt to read enclave's data will be denied by the PMP, preventing the software adversary [7]. In Keystone, the SM performs a clean context switch by flushing enclave states. Together with cache partitioning, the cache-based SCAs are prevented. Moreover, an enclave can be encrypted, and its page table can be self-managed; thus, any subtle attacks like controlled SCAs are impossible. Finally, Keystone also offers plugins to strengthen the TEE, such as memory encryption, enclave dynamic resizing, edge call service, and syscall services [7]. In Keystone and our implementation, the speculative attacks, timing SCAs, and SCAs that exploit hardware flaws in the off-chip components are considered out of scope. Finally, although possible, the enclave-to-peripheral binding is not recommended in the current implementation because introducing a peripheral driver into the runtime is not a two-way binding process, thus allowing DMA-capable peripherals attacks.

About the RoT and the secure boot process, not just Keystone but TEE models, in general, consider the secure boot process out of scope. In the end, TEE is just an isolated environment; it shouldn't be the RoT. As a general rule of thumb, a secure boot process with RoT is recommended to be run by hardware primitives or another entity inaccessible from the TEE's eyes. Common TEEs use extra hardware or third-party IPs to deliver the secure boot. In Keystone, the trusted domain operates based on the assumption that the hardware signed the SM during boot. Therefore, trusted hardware is needed for delivering that secure boot process with RoT. And that is the primary goal of the proposed TEE-HW in this dissertation. By introducing a secure boot mechanism with silicon level RoT, the CoT is completed, and the device's integrity is guaranteed.

As described in section 4.3.3, with all of the cryptographic keys are interlocked with each other, a direct attack to the keys chain is impossible. For example, The public root key P_R and the secret device key S_D are stored in the hidden ROM inside the isolated domain, and the secret root key S_R is not even in the chip. Only the public device key P_D is available in the public domain after boot and for the sole purpose of verification. Additionally, due to the isolation dictated by the bus architecture, even if a malicious enclave could hack the TEE side, it cannot retrieve any data in the hidden ROMs by any means. From the software perspective, the only attack surface left is by exploiting the interrupt channel for attestation. However, the IBex core only responds to external interrupts based on its program. Thus, the IBex's program can be updated in the future at any time to adapt to the new attack vector, if there is one. To conclude, the proposed secure boot scheme can still safeguard its secrets even if the TEE processors were compromised.

5.5 Summary

In this chapter, the performances of TEE-HW were presented. The completed systems were implemented in both FPGA and VLSI, and the results were reported. The boot performance using cryptographic accelerators was also given in this chapter. The heterogeneous design with isolated sub-system for secure boot process was made into two ROHM-180nm chips, and their measurement results were presented. The comparison with other recent works was analyzed, and security analysis was given.

This page intentionally left blank.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

6.1.1 Achievements

This dissertation proposed a Trusted Execution Environment HardWare (TEE-HW) framework that is easy to use, flexible for various needs, and easy to update in the future. Then based on the framework, a completed TEE-HW computer system was developed and tested. The proposed TEE-HW architecture contains several cryptographic accelerators for enhancing boot performances and increasing the security level. Finally, a heterogeneous architecture with an isolated sub-system was developed. The hidden Micro-Controller Unit (MCU) in the isolated architecture provides not only the secure Root-of-Trust (RoT) implementation but also the ability to adapt to the future changes of the boot sequence. To summary, the key achievements are described as follows:

1. **The TEE-HW framework:** by understanding that security is a race between attacker and defender, the goal of this dissertation is not only about the development of a fully protected computer system but also about a hardware framework that can be easily updated in the long run. In addition, all the source codes will be published in the open-source RISC-V community, thus opening up many opportunities for future developers to reuse and make further improvements.
2. **TEE-HW with cryptographic accelerators:** although the TEE hardware has been defined for memory isolation, the security algorithms often are executed using software implementations. Therefore, a custom TEE-HW was made exclusively for accelerating the TEE. Currently, the architecture contains several crypto-cores such as Advanced Encryption Standard (AES), Secure Hashing Algorithm 3 (SHA3), Ed25519, and True Random Number Generator (TRNG). The crypto-cores have been proven to be efficient not only for performance but also for security strength. The best part about the proposed architecture is that it is flexible and easily reconfigured for specific requirements just by changing variables in the Makefile system.

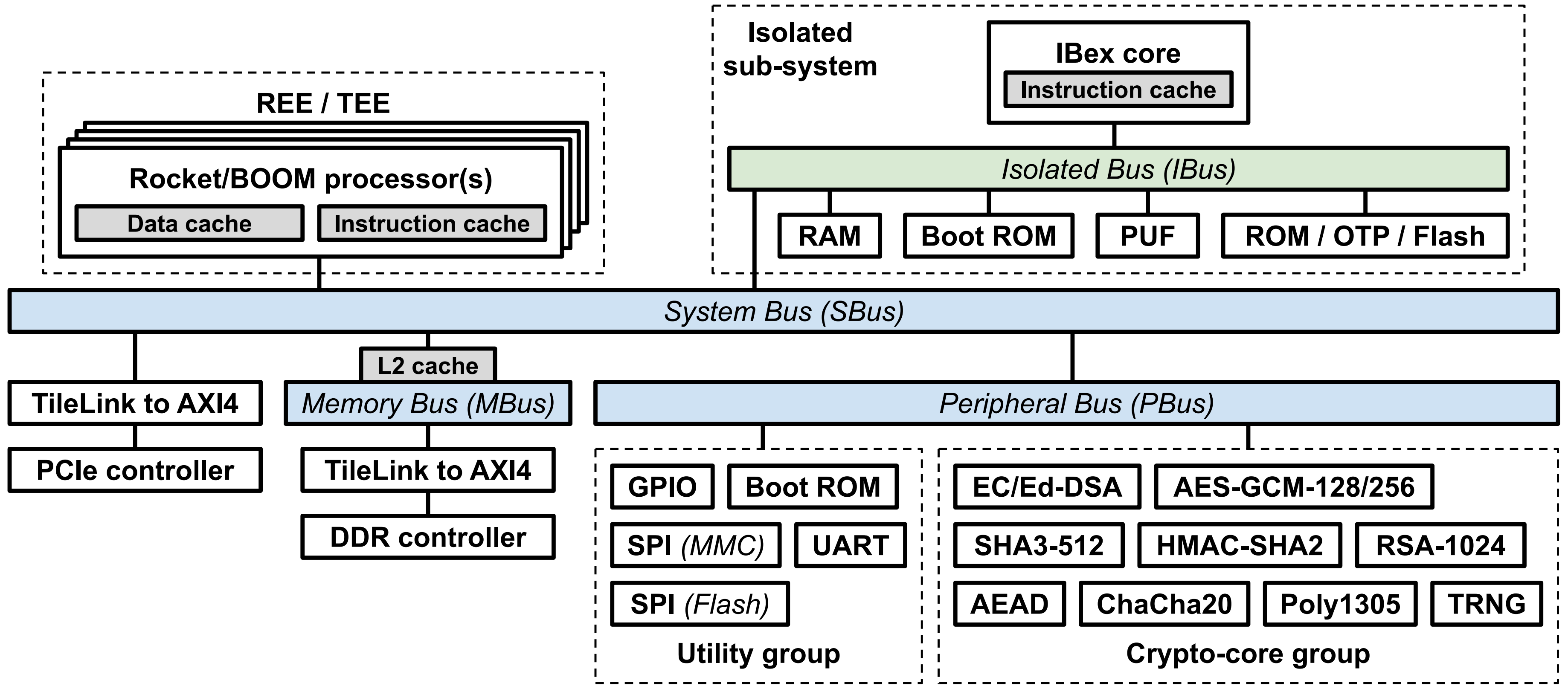


FIGURE 6.1: The next version of the TEE-HW architecture, FPGA version.

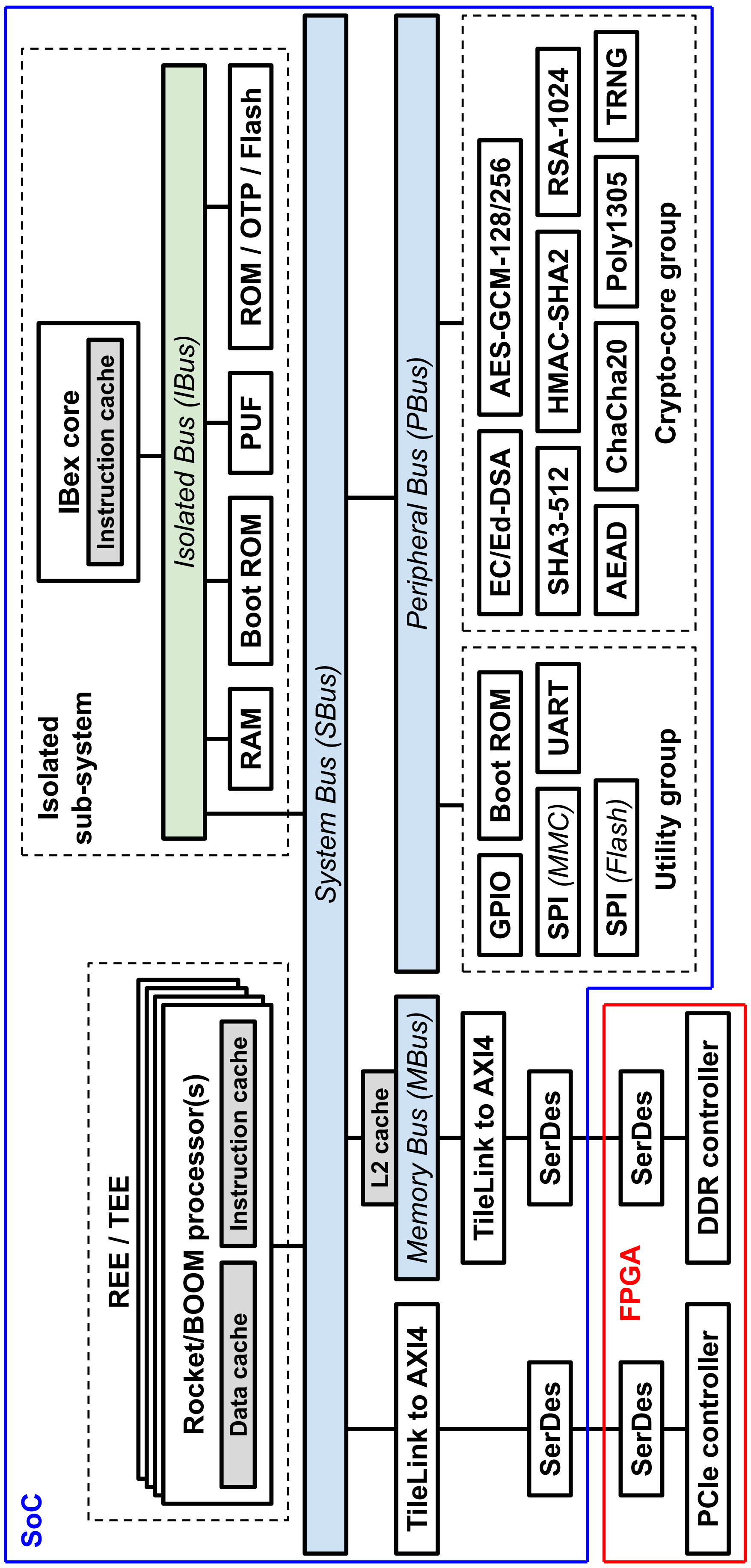


FIGURE 6.2: The next version of the TEE-HW architecture, chip version.

3. **TEE-HW with isolated RoT:** although various RoT implementations were proposed, some relied on obscurity to preserve the root keys, and others used a fixed circuit to do the RoT. As a result, they did not provide the flexibility for the keys generation scheme. Therefore, this dissertation's critical achievement is developing an RoT implementation in an isolated architecture for RISC-V-based TEE System-on-Chips (SoCs). The idea was based on a heterogeneous machine by combining an outside untrusted user-mode, typically Operating System (OS), with an inside isolated machine-mode (a hidden MCU). The untrusted side allows the user to interact with everyday activities, while the hidden MCU will take care of the sensitive data and cryptographic activities, such as secure boot, key generation, and sign and authenticate.

4. **Silicon-proof TEE-HW chips:** the proposed TEE-HW SoC was tested on Field-Programmaple Gate Array (FPGA) and then realized in Very Large Scale Integrated circuit (VLSI). Two ROHM-180nm chips were made, and the measurements were delivered.

6.1.2 Limitations

In this dissertation, there are several limitations have been recognized as follows:

1. During the chip measurement, significant time was wasted on fixing the testing Printed Circuit Board (PCB). This problem is related to the number of pinouts of the SoC. Because we have to mount the PCB onto an FPGA to use its Double Data Rate (DDR) memory, the number of pinouts in both parties is extensive. Furthermore, the overall performance of the system could be potentially limited by this problem.

2. The maximum operating frequencies (F_{Max}) of the chips are low. They are roughly about 50-MHz, although the design itself could go more than 100-MHz. This problem is because of the Inputs/Outpus (I/Os). In ROHM-180nm technology, the oscillating frequency that I/O allows is about 50-MHz to 100-MHz. And because an I/O pin will feed the internal system clock, thus the final F_{Max} is hard to get close the 100-MHz.

3. The next issue is related to the limited optional security features. Currently, only four crypto-cores are available, including SHA3, AES, Ed25519, and TRNG. The number of recent cryptographic functions is much more than that.

4. The final problem is about the root keys in the proposed isolated architecture. By using Read-Only Memory (ROM) to store the root keys, the flexibility of the booting sequence is limited. With ROM inside the chip, the root keys can not be changed after being fabricated. Thus, some other implementations such as One-Time Programmable (OTP) memory of flash will be more helpful.

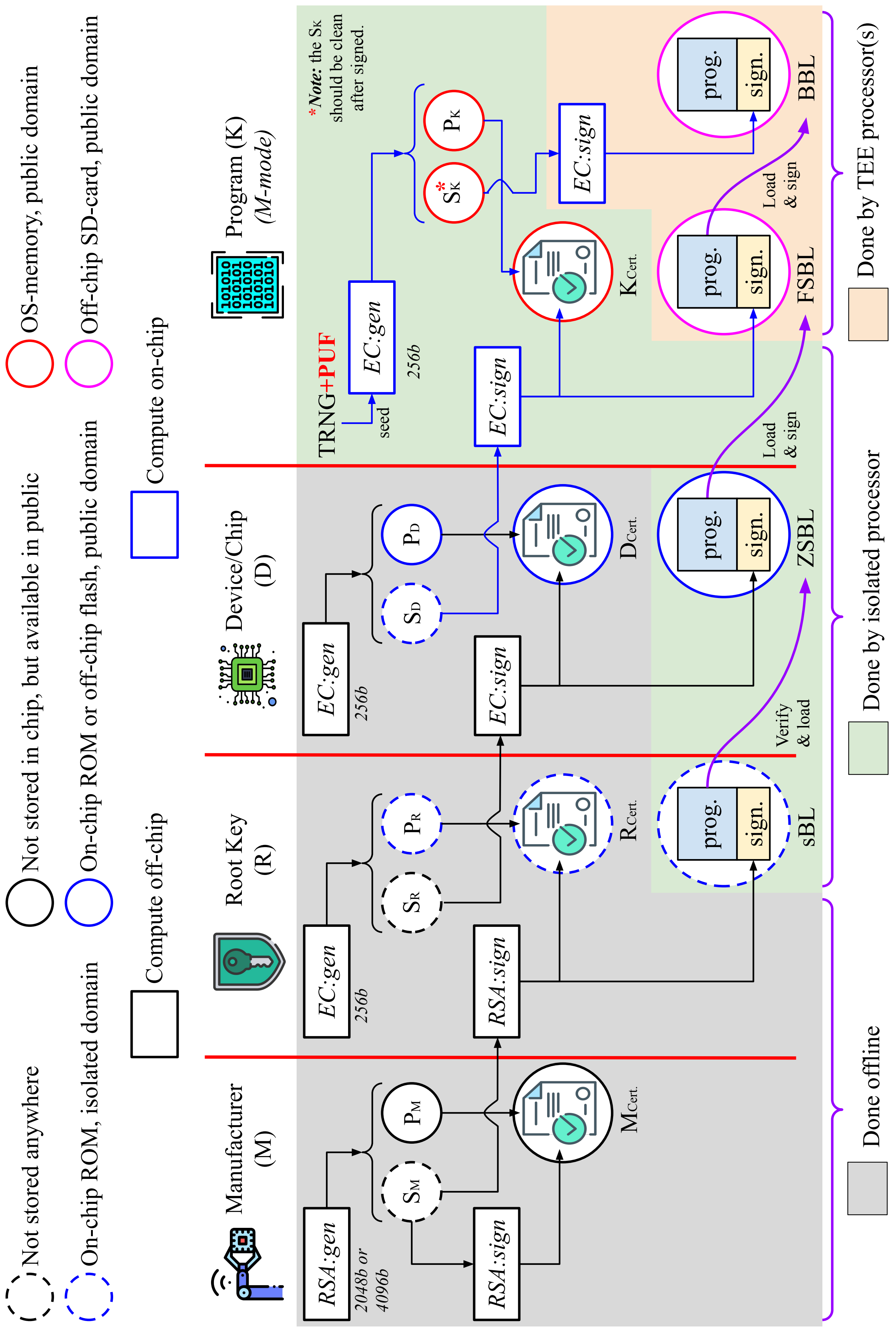


FIGURE 6.3: In the future, the PUF should be included together with TRNG to avoid TRNG fault attacks [14, 15], and the P_R and S_D should be stored in on-chip OTP/flash for more versatile usage.

6.2 Future Work

6.2.1 Improve I/O Performance

To overcome the issue with the number of pinouts, Serializer/Deserializer (SerDes) architecture [139] could be used. SerDes is a way to convert any bus protocol into a series of parallel-to-serial and serial-to-parallel packages. Thus, we can fine-tune our performances to the available resources. Furthermore, unlike TileTink, SerDes is the protocol that was designed to use for off-chip communication. Hence, the expected speed of the SerDes to be used as General-Purpose IO (GPIO) will be higher than directly exporting out the TileLink bus.

For the problem with the maximum operating frequencies, a Phase Locked Loop (PLL) or Frequency Locked Loop (FLL) is suggested to be implemented inside the chip. Then, the on-chip-communication system-clock and the off-chip-communication SerDes-clock can be separated. The SerDes-clock speed will be limited by I/Os at around 50-MHz as usual. But the internal system clock will be fed by the embedded PLL or FLL, which can be much higher than 100-MHz.

6.2.2 Develop Recent Cryptographic Hardware

To keep up with the security development trend, many crypto-cores must be added to the current TEE-HW framework. Figure 6.1 shows the future version of the architecture presented in this dissertation. The reason why those crypto-cores were chosen for the next version is because of the latest version of Transport Layer Security (TLS), version 1.3 [140]. The TLS v1.3 was chosen as a guide for developing new crypto-cores because it is a well-known and the most used cryptographic protocol, specially designed for secure communication over the internet. According to the TLS v1.3, the crypto-cores that need to be developed are the AES Galois Counter Mode (AES-GCM) [141], Hashed Message Authentication Code SHA2 (HMAC-SHA2) [142], Rivest-Shamir-Adleman (RSA) [113], ChaCha20 [143], Poly1305 [144], Authenticated Encryption with Associated Data (AEAD) [145], Elliptic Curve Digital Signature Algorithm (ECDSA) [114], and Edwards-curve Digital Signature Algorithm (EdDSA) [115].

6.2.3 Other Improvements

A Physical Unclonable Function (PUF) should be developed in the near future, as shown in Figure 6.1, for adding more flexibility to the current isolated architecture. With PUF and TRNG, the system will have more options for the secure boot program. For example, by using PUF together with TRNG, the TRNG fault attacks [14, 15] can be prevented, thus increasing the security level for the EC-genkey in the current boot flow, as shown in Figure 6.3.

Finally, the storing place for the root/device keys (i.e., P_R and S_D in Figure 6.3) should have the option for OTP memory or flash memory instead of ROM. With an OTP memory, the fabrication process does not fix those root/device keys. Hence, the keys can be programmed after taped-out, thus leading to different keys for each chip. Similarly, if we have on-chip flash memory, we can re-program the root/device keys for other purposes or other keys scheduling schemes. The on-chip OTP or flash memories are for more versatile usage, not for security reasons.

This page intentionally left blank.

Bibliography

[1] Intel Corp. Intel Software Guard Extensions (Intel SGX) Developer Guide. [Online]. Available: https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf

[2] ARM Ltd., "ARM Security Technology: Building a Secure System using TrustZone Technology," ARM Ltd., Tech. Rep. PRD29-GENC-009492C, Apr. 2009. [Online]. Available: https://documentation-service.arm.com/static/5f212796500e883ab8e74531?token=

[3] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation," in *ACM SIGSAC Conf. on Computer and Comm. Secu. (CCS)*, London, UK, Nov. 2019, p. 1087–1099.

[4] Hex Five Security, Inc. MultiZone Hex-Five Security. [Online]. Available: https://hex-five.com/

[5] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation," in *Secu. Symp. (USENIX)*, Aug. 2016, pp. 857—874.

[6] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V," in *Network and Distributed Syst. Secu. Symp. (NDSS)*, San Diego, CA, USA, Feb. 2019, pp. 1–15.

[7] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, "Keystone: An Open Framework for Architecting Trusted Execution Environments," in *European Conf. on Computer Syst. (EUROSYS)*, Heraklion, Greece, Apr. 2020, pp. 1–16.

[8] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "CURE: A Security Architecture with CUstomizable and Resilient Enclaves," in *USENIX Secu. Symp. (USENIX Security)*, Virtual Event, Aug. 2021, pp. 1073–1090.

[9] P. Nasahl, R. Schilling, M. Werner, and S. Mangard, "HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment," in *ACM Asia Conf. on Computer and Comm. Secu. (ASIA CCS)*, Virtual Event, Hong Kong, China, Jun. 2021, pp. 187—-199.

[10] SiFive, Inc. Securing The RISC-V Revolution. [Online]. Available: https://www.sifive.com/technology/shield-soc-security

[11] V. B. Y. Kumar, A. Chattopadhyay, J. H.-Yahya, and A. Mendelson, “ITUS: A Secure RISC-V System-on-Chip,” in *IEEE Int. System-on-Chip Conf. (SOCC)*, Singapore, Singapore, Sep. 2019, pp. 418–423.

[12] J. H.-Yahya, M. M. Wong, V. Pudi, S. Bhasin, and A. Chattopadhyay, “Lightweight Secure-Boot Architecture for RISC-V System-on-Chip,” in *Int. Symp. on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA, Mar. 2019, pp. 216–223.

[13] A. Waterman, K. Asanović, and John Hauser, “The RISC-V Instruction Set Manual Volume II: Privileged Architecture,” SiFive Inc. and EECS Dep., Univ. of California, Berkeley, Tech. Rep., Dec. 2021. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

[14] H. Martín, T. Korak, E. S. Millán, and M. Hutter, “Fault Attacks on STRNGs: Impact of Glitches, Temperature, and Underpowering on Randomness,” *IEEE Trans. on Info. Forensics and Secu.*, vol. 10, no. 2, pp. 266–277, Feb. 2015.

[15] S. Larimian, M. R. Mahmoodi, and D. B. Strukov, “Lightweight Integrated Design of PUF and TRNG Security Primitives Based on eFlash Memory in 55-nm CMOS,” *IEEE Trans. on Electron Devices*, vol. 67, no. 4, pp. 1586–1592, Mar. 2020.

[16] A. Waterman and K. Asanovi ́c, “The RISC-V Instruction Set Manual Volume I: Unprivileged ISA,” SiFive Inc. and EECS Dep., Univ. of California, Berkeley, Tech. Rep., Dec. 2019. [Online]. Available: https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf

[17] Unified Extensible Firmware Interface Forum. (May 2020) Unified Extensible Firmware Interface (UEFI) Specification, Version 2.8 Errata B. [Online]. Available: https://uefi.org/sites/default/files/resources/UEFI%20Spec%202.8B%20May%202020.pdf

[18] Jessie Frazelle, “Securing the Boot Process,” *Commun. ACM*, vol. 63, no. 3, pp. 38—42, Feb. 2020.

[19] V. Costan and S. Devadas, “Intel SGX Explained,” Cryptology ePrint Archive, Report 2016/086, Jan. 2016, https://eprint.iacr.org/2016/086.

[20] V. Costan, I. Lebedev, and S. Devadas, *Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture*. Now Foundations and Trends, Jul. 2017.

[21] ——, *Secure Processors Part II: Intel SGX Security Analysis and MIT Sanctum Architecture*. Now Foundations and Trends, Jul. 2017.

[22] A. Baumann, M. Peinado, and G. Hunt, “Shielding Applications from an Untrusted Cloud with Haven,” in *USENIX Symp. on Operating Syst. Design and Imple. (OSDI)*, Broomfield, CO, USA, Oct. 2014, pp. 267–283.

[23] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX," in *USENIX Annual Technical Conf. (ATC)*, Santa Clara, CA, USA, Jul. 2017, pp. 645–658.

[24] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure Linux Containers with Intel SGX," in *USENIX Symp. on Operating Syst. Design and Impl. (OSDI)*, Savannah, GA, USA, Nov. 2016, pp. 689–703.

[25] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–36, Nov. 2019.

[26] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using Verification to Disentangle Secure-Enclave Hardware from Software," in *Symp. on Operating Syst. Principles (SOSP)*, Shanghai, China, Oct. 2017, pp. 287–305.

[27] Linaro Ltd. Open Portable Trusted Execution Environment. [Online]. Available: https://www.op-tee.org/

[28] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "SANCTUARY: ARMing TrustZone with User-space Enclaves," in *Network and Distributed Syst. Secu. (NDSS) Symp.*, San Diego, CA, USA, Feb. 2019, pp. 1–15.

[29] David Kaplan. (Feb. 2017) Protecting VM Register State with SEV-ES. [Online]. Available: https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf

[30] (Jan. 2020) AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. [Online]. Available: https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[31] S. P. Dandamudi, *Guide to RISC Processors: for Programmers and Engineers*. New York, NY, USA: Springer, 2005.

[32] ARM Ltd. Arm Architecture Reference Manual Armv8, for A-profile architecture. [Online]. Available: https://developer.arm.com/documentation/ddi0487/latest/

[33] Wikipedia. Reduced Instruction Set Computer. [Online]. Available: https://en.wikipedia.org/wiki/Reduced_instruction_set_computer

[34] Rupert Goodwins. (Mar. 2020) Chips that pass in the night: How risky is RISC-V to Arm, Intel and the others? Very. [Online]. Available: https://www.theregister.co.uk/2020/03/09/risc_v_intel_amd_arm/

[35] Y. Wang and N. Tan, "An Application-Specific Microprocessor for Energy Metering Based on RISC-V," in *IEEE Int. Conf. on IC Design and Tech. (ICICDT)*, Suzhou, China, Jun. 2019, pp. 2381–3555.

[36] A. Munir, M. Magdy, S. Ahmed, S. Nasr, S. E.-Ashry, and A. Shalaby, "Fast Reliable Verification Methodology for RISC-V Without a Reference Model," in *Int. Workshop on Microprocessor and SOC Test and Verification (MTV)*, Austin, TX, USA, Dec. 2018, pp. 12–17.

[37] E. Gür, Z. E. Sataner, Y. H. Durkaya, and S. Bayar, "FPGA Implementation of 32-bit RISC-V Processor with Web-Based Assembler-Disassembler," in *Int. Symp. on Fundamentals of Electrical Engi. (ISFEE)*, Bucharest, Romania, Nov. 2018, pp. 1–4.

[38] D. K. Dennis, A. Priyam, S. S. Virk, S. Agrawal, T. Sharma, A. Mondal, and K. C. Ray, "Single Cycle RISC-V Micro Architecture Processor and Its FPGA Prototype," in *Int. Symp. on Embedded Computing and Syst. Design (ISED)*, Durgapur, India, Dec. 2017, pp. 1–5.

[39] K. Patsidis, D. Konstantinou, C. Nicopoulos, and G. Dimitrakopoulos, "A Low-cost Synthesizable RISC-V Dual-issue Processor Core Leveraging the Compressed Instruction Set Extension," *Microprocessors and Microsystems*, vol. 61, pp. 1–10, Sep. 2018.

[40] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket Chip Generator," EECS Dep., Univ. of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr. 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[41] SiFive, Inc. SiFive E34 Core Complex Manual. [Online]. Available: https://sifive.cdn.prismic.io/sifive/4aeb590e-86ea-45ee-b21c-1f0770a42798_e34_core_complex_manual_21G2.pdf

[42] ——. SiFive U54 Core Complex Manual. [Online]. Available: https://sifive.cdn.prismic.io/sifive/a07d1a9a-2cb8-4cf5-bb75-5351888ea2e1_u54_core_complex_manual_21G2.pdf

[43] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Trans. on Very Large Scale Integration (VLSI) Syst.*, vol. 25, no. 10, pp. 2700–2713, Feb. 2017.

[44] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Trans. on Very Large Scale Integration (VLSI) Syst.*, vol. 27, no. 11, pp. 2629–2640, Jul. 2019.

[45] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0," EECS

Dep., Univ. of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[46] RISC-V Foundation. [Online]. Available: https://riscv.org/risc-v-foundation/

[47] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing Hardware in a Scala Embedded Language," in *Design Automation Conf. (DAC)*, San Francisco, CA, USA, Jun. 2012, pp. 1212–1221.

[48] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach, "Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, Irvine, CA, USA, Nov. 2017, pp. 209–216.

[49] P. S. Li, A. M. Izraelevitz, and J. Bachrach, "Specification for the FIRRTL Language," EECS Dep., Univ. of California, Berkeley, Tech. Rep. UCB/EECS-2016-9, Feb. 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html

[50] RISC-V Foundation. RISC-V Exchange: Cores & SoCs. [Online]. Available: https://riscv.org/exchange/cores-socs/

[51] S. Sau, J. H.-Yahya, M. M. Wong, K. Y. Lam, and A. Chattopadhyay, "Survey of Secure Processors," in *Int. Conf. on Embedded Computer Syst.: Arch., Modeling, and Simulation (SAMOS)*, Pythagorion, Greece, Jul. 2017, pp. 253–260.

[52] A. Furtak, Y. Bulygin, O. Bazhaniuk, J. Loucaides, A. Matrosov, and M. Gorobets, "BIOS and Secure Boot Attacks Uncovered," in *Ekoparty Secu. Conf.*, Buenos Aires, Argentina, 2014, pp. 1–79. [Online]. Available: http://www.c7zero.info/stuff/BIOSandSecureBootAttacksUncovered_eko10.pdf

[53] Intel Corp. Intel Active Management Technology (AMT) Developers Guide. [Online]. Available: https://www.intel.com/content/www/us/en/develop/documentation/amt-developer-guide/top.html

[54] ARM Ltd. ARM Security IP: CryptoCell-700 Family. [Online]. Available: https://developer.arm.com/ip-products/security-ip/cryptocell-700-family

[55] Inside a Deeply Embedded Security Processor. [Online]. Available: https://i.blackhat.com/USA-20/Wednesday/us-20-Buhren-All-You-Ever-Wanted-To-Know-About-The-AMD-Platform-Security-Processor-And-Were-Afraid-To-Emulate.pdf

[56] Rambus, Inc. Security CryptoManager Provisioning. [Online]. Available: https://www.rambus.com/security/provisioning-and-key-management/cryptomanager-infrastructure/

[57] lowRISC CIC. OpenTitan. [Online]. Available: https://github.com/lowRISC/opentitan

[58] ISO/IEC, "Information Technology — Trusted Platform Module Library — Part 1: Architecture," ISO/IEC, Tech. Rep. 11889-1:2015, Aug. 2015. [Online]. Available: https://www.iso.org/standard/66510.html

[59] PHAM Laboratory. TEE Hardware Platform. [Online]. Available: https://github.com/uec-hanken/tee-hardware

[60] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing Root of Trust for ARM TrustZone Using On-Chip SRAM," in *Int. Workshop on Trustworthy Embedded Devices (TrustED)*, Nov. 2014, pp. 25—-36.

[61] C. Duran, D. L. Rueda, G. Castillo, A. Agudelo, C. Rojas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, J. Ardila, L. Rueda, H. Hernandez, J. Amaya, and E. Roa, "A 32-bit RISC-V AXI4-lite Bus-based Microcontroller with 10-bit SAR ADC," in *IEEE Latin American Symp. on Circ. & Syst. (LASCAS)*, Florianopolis, Brazil, Apr. 2016, pp. 315–318.

[62] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. L.-Hurd, and C. Rozas, "Intel Software Guard Extensions (Intel SGX) Support for Dynamic Memory Management Inside an Enclave," in *Hardware and Arch. Support for Secu. and Privacy (HASP)*, no. 10, Seoul, Korea, Jun. 2016, pp. 1–9.

[63] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "PULP: A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision," *Journal of Signal Processing Syst.*, vol. 84, pp. 339—-354, Sep. 2016.

[64] C. Duran, L. E. Rueda G., A. Amaya, R. Torres, J. Ardila, L. Rueda D., G. Castillo, A. Agudelo, C. Rojas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, H. Hernandez, and E. Roa, "A System-on-Chip Platform for the Internet of Things Featuring a 32-bit RISC-V Based Microcontroller," in *IEEE Latin American Symp. on Circ. & Syst. (LASCAS)*, Bariloche, Argentina, Feb. 2017, pp. 1–4.

[65] Yogesh Swami. (Jul. 2017) Intel SGX Remote Attestation is not sufficient. [Online]. Available: https://eprint.iacr.org/2017/736.pdf

[66] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling, "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices," *ACM Trans. on Priv. Secur.*, vol. 20, no. 3, pp. 1–33, Aug. 2017.

[67] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Teman, J. Constantin, A. Burg, I. M.-Panades, E. Beignè, F. Clermidy, P. Flatresse, and L. Benini, "Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster," *IEEE Micro*, vol. 37, no. 5, pp. 20–31, Oct. 2017.

[68] S. Contiu, R. Pires, S. Vaucher, M. Pasin, P. Felber, and L. Réveillère, "IBBE-SGX: Cryptographic Group Access Control Using Trusted Execution Environments," in *Annual IEEE/IFIP Int. Conf. on Dependable Syst. and Networks (DSN)*, Luxembourg City, Luxembourg, Jun. 2018, pp. 207–218.

[69] M. Staffa, G. Mazzeo, and L. Sgaglione, "Hardening ROS via Hardware-assisted Trusted Execution Environment," in *IEEE Int. Symp. on Robot and Human Interactive Comm. (RO-MAN)*, Nanjing, China, Aug. 2018, pp. 491–494.

[70] Y. Fan, S. Liu, G. Tan, X. Lin, G. Zhao, and J. Bai, "One Secure Access Scheme Based on Trusted Execution Environment," in *IEEE Int. Conf. On Trust, Secu. And Privacy in Computing And Comm. / IEEE Int. Conf. On Big Data Science And Engi. (TrustCom/BigDataSE)*, New York, NY, USA, Aug. 2018, pp. 16–21.

[71] Z. Ning, J. Liao, F. Zhang, and W. Shi, "Preliminary Study of Trusted Execution Environments on Heterogeneous Edge Platforms," in *IEEE/ACM Symp. on Edge Computing (SEC)*, Seattle, WA, USA, Oct. 2018, pp. 421–426.

[72] M. A. Mukhtar, M. K. Bhatti, and G. Gogniat, "Architectures for Security: A Comparative Analysis of Hardware Security Features in Intel SGX and ARM TrustZone," in *Int. Conf. on Communication, Computing, and Digi. Syst. (C-CODE)*, Islamabad, Pakistan, Mar. 2019, pp. 299–304.

[73] L. Guan, C. Cao, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "Building a Trustworthy Execution Environment to Defeat Exploits from both Cyber Space and Physical Space for ARM," *IEEE Trans. on Dependable and Secure Computing*, vol. 16, no. 3, pp. 438–453, May 2019.

[74] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer, and M. Linauer, "Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation," in *Mediterranean Conf. on Embedded Computing (MECO)*, Budva, Montenegro, Jun. 2019, pp. 1–6.

[75] M. Bailleu, D. Dragoti, P. Bhatotia, and C. Fetzer, "TEE-Perf: A Profiler for Trusted Execution Environments," in *Annual IEEE/IFIP Int. Conf. on Dependable Syst. and Networks (DSN)*, Portland, OR, USA, Jun. 2019, pp. 414–421.

[76] A. Gollamudi, S. Chong, and O. Arden, "Information Flow Control for Distributed Trusted Execution Environments," in *IEEE Computer Secu. Foundations Symp. (CSF)*, Hoboken, NJ, USA, Jun. 2019, pp. 304–30 414.

[77] H. Cui, H. Duan, Z. Qin, C. Wang, and Y. Zhou, "SPEED: Accelerating Enclave Applications via Secure Deduplication," in *IEEE Int. Conf. on Distributed Computing Syst. (ICDCS)*, Dallas, TX, USA, Jul. 2019, pp. 1072–1082.

[78] E. M. Benhani, L. Bossuet, and A. Aubert, "The Security of ARM TrustZone in a FPGA-Based SoC," *IEEE Trans. on Computers*, vol. 68, no. 8, pp. 1238–1248, Aug. 2019.

[79] D. J. Sebastian, U. Agrawal, A. Tamimi, and A. Hahn, "DER-TEE: Secure Distributed Energy Resource Operations Through Trusted Execution Environments," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6476–6486, Aug. 2019.

[80] K. Lin, W. Liu, K. Zhang, and B. Tu, "HyperMI: A Privilege-level VM Protection Approach Against Compromised Hypervisor," in *IEEE Int. Conf. On Trust, Secu. And Privacy in Computing and Comm. / IEEE Int. Conf. On Big Data Science And Engi. (TrustCom/BigDataSE)*, Rotorua, New Zealand, Aug. 2019, pp. 58–65.

[81] D. Ji, Q. Zhang, S. Zhao, Z. Shi, and Y. Guan, "MicroTEE: Designing TEE OS Based on the Microkernel Architecture," in *IEEE Int. Conf. On Trust, Secu. And Privacy in Computing and Comm. / IEEE Int. Conf. On Big Data Science And Engi. (TrustCom/BigDataSE)*, Rotorua, New Zealand, Aug. 2019, pp. 26–33.

[82] R. Ladjel, N. Anciaux, P. Pucheral, and G. Scerri, "Trustworthy Distributed Computations on Personal Data Using Trusted Execution Environments," in *IEEE Int. Conf. On Trust, Secu. And Privacy in Computing and Comm. / IEEE Int. Conf. On Big Data Science And Engi. (TrustCom/BigDataSE)*, Rotorua, New Zealand, Aug. 2019, pp. 381–388.

[83] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, Arvind, and S. Devadas, "MI6: Secure Enclaves in a Speculative Out-of-Order Processor," in *Annual IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, Columbus, OH, USA, Oct. 2019, pp. 42–56.

[84] J. V. Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes," in *ACM SIGSAC Conf. on Computer and Comm. Secu. (CCS)*, London, UK, Nov. 2019, pp. 1741—1758.

[85] Trong-Thuc Hoang, Ckristian Duran, Duc-Thinh Nguyen-Hoang, Duc-Hung Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Quick Boot of Trusted Execution Environment With Hardware Accelerators," *IEEE Access*, vol. 8, pp. 74 015–74 023, Apr. 2020.

[86] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine," *Workshop on Computer Arch. Research with RISC-V (CARRV)*, May 2020.

[87] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems," in *IEEE Symp. on Secu. and Privacy (SP)*, San Francisco, CA, USA, May 2020, pp. 1416–1432.

[88] Z. Yu, S. Shinde, T. E. Carlson, and P. Saxena, “Elasticlave: An Efficient Memory Model for Enclaves,” Jul. 2020.

[89] Y. Qin, J. Liu, S. Zhao, D. Feng, and W. Feng, “RIPTE: Runtime Integrity Protection Based on Trusted Execution for IoT Device,” *Secu. and Comm. Networks*, vol. 2020, pp. 1–14, Sep. 2020.

[90] S. Moritz, A. Dhar, I. Puddu, K. Kostiainen, and S. Capkun, “PIE: A Dynamic TCB for Remote Systems with a Platform Isolation Environment,” Oct. 2020.

[91] J. Jang and B. B. Kang, “Retrofitting the Partially Privileged Mode for TEE Communication Channel Protection,” *IEEE Trans. on Dependable and Secure Computing*, vol. 17, no. 5, pp. 1000–1014, Oct. 2020.

[92] SpinalHDL. A FPGA Friendly 32-bit RISC-V CPU Implementation. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[93] Trong-Thuc Hoang, Ckristian Duran, Khai-Duy Nguyen, Tuan-Kiet Dang, Quynh Nguyen Quang Nhu, Phuc Hong Than, Xuan-Tu Tran, Duc-Hung Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, “Low-power High-performance 32-bit RISC-V Microcontroller on 65-nm Silicon-On-Thin-BOX (SOTB),” *IEICE Elec. Express*, vol. 17, no. 20, p. 20200282, Oct. 2020.

[94] M. Boubakri, F. Chiatante, and B. Zouari, “Towards a Firmware TPM on RISC-V,” in *Design, Automation & Test in Europe Conf. Exhibition (DATE)*, Grenoble, France, Feb. 2021, pp. 647–650.

[95] Khai-Duy Nguyen, Dang Tuan Kiet, Trong-Thuc Hoang, Nguyen Quang Nhu Quynh, Xuan-Tu Tran, and Cong-Kha Pham, “A Trigonometric Hardware Acceleration in 32-bit RISC-V Microcontroller with Custom Instruction,” *IEICE Elec. Express (ELEX)*, vol. 18, no. 16, p. 20210266, Aug. 2021.

[96] K. Suzaki, K. Nakajima, T. Oi, and A. Tsukamoto, “TS-Perf: General Performance Measurement of Trusted Execution Environment and Rich Execution Environment on Intel SGX, Arm TrustZone, and RISC-V Keystone,” *IEEE Access*, vol. 9, pp. 133 520–133 530, Sep. 2021.

[97] K. Xia, Y. Luo, X. Xu, and S. Wei, “SGX-FPGA: Trusted Execution Environment for CPU-FPGA Heterogeneous Architecture,” in *ACM/IEEE Design Automation Conf. (DAC)*, San Francisco, CA, USA, Dec. 2021, pp. 301–306.

[98] D. Kaplan, J. Powell, and T. Woller. (Apr. 2016) AMD Memory Encryption. [Online]. Available: https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[99] Hex Five Security, Inc. [Online]. Available: https://github.com/hex-five

[100] The MIT Sanctum Processor System. [Online]. Available: https://github.com/ilebedev/sanctum

[101] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi. TIMBER-V. [Online]. Available: https://github.com/sam1013/timberv-riscv-tools/tree/timberv

[102] Keystone Enclave. Keystone: An Open-Source Secure Enclave Framework for RISC-V Processors. [Online]. Available: https://github.com/keystone-enclave/keystone

[103] IEEE, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul. 2019.

[104] RISC-V GNU Compiler Toolchain. [Online]. Available: https://github.com/riscv-collab/riscv-gnu-toolchain

[105] RISC-V Glibc. [Online]. Available: https://github.com/riscvarchive/riscv-glibc

[106] RISC-V Newlib. [Online]. Available: https://github.com/riscv-collab/riscv-newlib

[107] RISC-V OpenOCD. [Online]. Available: https://github.com/riscv/riscv-openocd

[108] Spike RISC-V ISA Simulator. [Online]. Available: https://github.com/riscv-software-src/riscv-isa-sim

[109] RISC-V QEMU. [Online]. Available: https://github.com/riscvarchive/riscv-qemu

[110] Verilator. [Online]. Available: https://github.com/verilator/verilator

[111] RISC-V Angel. [Online]. Available: https://github.com/riscv-software-src/riscv-angel

[112] Martin Schoeberl, *Digital Design with Chisel*. Kindle Direct Publishing, Aug. 2019. [Online]. Available: https://github.com/schoeberl/chisel-book

[113] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. of the ACM*, vol. 21, no. 2, p. 120–126, Feb. 1978.

[114] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *Int. Journal of Info. Secu.*, vol. 1, no. 1, p. 36–63, Aug. 2001.

[115] S. Josefsson and I. Liusvaara. (Jan. 2017) RFC8032: Edwards-Curve Digital Signature Algorithm (EdDSA). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8032

[116] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed High-security Signatures," *Journal of Crypto. Engi.*, vol. 2, no. 2, pp. 77–89, Sep. 2012.

[117] Trong-Thuc Hoang. VexRiscv 32-bit MCU. [Online]. Available: https://github.com/thuchoang90/VexRiscv

[118] SiFive, Inc. SiFive FE310-G000 Manual. [Online]. Available: https://sifive.cdn.prismic.io/sifive/4d063bf8-3ae6-4db6-9843-ee9076ebadf7_fe310-g000.pdf

[119] ——. SiFive FE310-G002 Manual. [Online]. Available: https://sifive.cdn.prismic.io/sifive%2F9ecbb623-7c7f-4acc-966f-9bb10ecdb62e_fe310-g002.pdf

[120] H. Okuhara, Y. Fujita, K. Usami, and H. Amano, "Power Optimization Methodology for Ultralow Power Microcontroller With Silicon on Thin BOX MOSFET," *IEEE Trans. on Very Large Scale Integration (VLSI) Syst.*, vol. 25, no. 4, pp. 1578–1582, Dec. 2016.

[121] T. Ishigaki, R. Tsuchiya, Y. Morita, N. Sugii, and S. Kimura, "Effects of Device Structure and Back Biasing on HCI and NBTI in Silicon-on-Thin-BOX (SOTB) CMOSFET," *IEEE Trans. on Electron Devices*, vol. 58, no. 4, pp. 1197–1204, Mar. 2011.

[122] Ba-Anh Dao, Trong-Thuc Hoang, Anh-Tien Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Exploiting the Back-Gate Biasing Technique as a Countermeasure Against Power Analysis Attacks," *IEEE Access*, vol. 9, pp. 24 768–24 786, Feb. 2021.

[123] A. Stillmaker and B. Baas, "Scaling Equations for the Accurate Prediction of CMOS Device Performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, Feb. 2017.

[124] SiFive. Freedom platforms. [Online]. Available: https://github.com/sifive/freedom

[125] Trong-Thuc Hoang. SIFIVE FREEDOM. [Online]. Available: https://thuchoang90.github.io/vc707

[126] Univ. of California at Berkeley, "Chipyard: An Agile RISC-V SoC Design Framework with In-Order Cores, Out-of-Order Cores, Accelerators, and More." [Online]. Available: https://github.com/ucb-bar/chipyard

[127] ChipsAlliance. Rocket Chip Generator. [Online]. Available: https://github.com/chipsalliance/rocket-chip

[128] Berkeley Architecture Research group. The Berkeley Out-of-Order RISC-V Processor. [Online]. Available: https://github.com/riscv-boom/riscv-boom

[129] SiFive, Inc. (Aug. 2019) SiFive TileLink Specication. [Online]. Available: https://www.sifive.com/documentation/tilelink/tilelink-spec/

[130] ARM, "AMBA AXI and ACE Protocol Specification," ARM, Tech. Rep. ARM IHI 0022H.c, Jan. 2021. [Online]. Available: https://developer.arm.com/architectures/system-architectures/amba/specifications

[131] Morris J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Aug. 2015. [Online]. Available: https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions

[132] NIST standard, "Advanced Encryption Standard (AES)," Nov. 2001. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

[133] lowRISC CIC, "IBex RISC-V Core." [Online]. Available: https://github.com/lowRISC/ibex

[134] Ronaldo Serrano, Ckristian Duran, Trong-Thuc Hoang, Marco Sarmiento, Khai-Duy Nguyen, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "A Fully Digital True Random Number Generator With Entropy Source Based in Frequency Collapse," *IEEE Access*, vol. 9, pp. 105 748–105 755, Jul. 2021.

[135] E. Barker, A. Roginsky, and R. Davis, "Recommendation for Cryptographic Key Generation," National Institute of Standards and Technology (NIST), Gaithersburg, MD, Tech. Rep., Jun. 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-133r2

[136] Roy Longbottom, "Dhrystone Benchmark Results On PCs," Oct. 2021. [Online]. Available: http://www.roylongbottom.org.uk/dhrystone%20results.htm

[137] Stratify Labs. (May 2019) Dhrystone Benchmarking on MCUs. [Online]. Available: https://blog.stratifylabs.co/device/2019-05-20-Dhrystone-Benchmarking-on-MCUs/

[138] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, "InkTag: Secure Applications on an Untrusted Operating System," *SIGPLAN Not.*, vol. 48, no. 4, p. 265–278, Mar. 2013.

[139] Wikipedia. SerDes. [Online]. Available: https://en.wikipedia.org/wiki/SerDes

[140] Internet Engineering Task Force (IETF). The Transport Layer Security (TLS) Protocol Version 1.3. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446

[141] D. A. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM). [Online]. Available: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

[142] H. Krawczyk, M. Bellare, and R. Canetti. (Feb. 1997) RFC2104: HMAC: Keyed-Hashing for Message Authentication. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2104

[143] Daniel J. Bernstein. (Jan. 2008) ChaCha, a variant of Salsa20. [Online]. Available: https://cr.yp.to/chacha/chacha-20080128.pdf

[144] ——, "The Poly1305-AES Message-Authentication Code," in *Int. Conf. on Fast Software Encryption*, ser. FSE'05, Feb. 2005, p. 32–49.

[145] Y. Nir and A. Langley. (Jun. 2018) RFC8439: ChaCha20 and Poly1305 for IETF Protocols. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8439

This page intentionally left blank.

Appendix A

Related Publications

A.1 Journal

[1] Trong-Thuc Hoang, Ckristian Duran, Duc-Thinh Nguyen-Hoang, Duc-Hung Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Quick Boot of Trusted Execution Environment with Hardware Accelerators," in *IEEE Access*, vol. 8, pp. 74015-74023, 2020.

[2] Trong-Thuc Hoang, Ckristian Duran, Khai-Duy Nguyen, Tuan-Kiet Dang, Quynh Nguyen Quang Nhu, Phuc Hong Than, Xuan-Tu Tran, Duc-Hung Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Low-power High-performance 32-bit RISC-V Microcontroller on 65-nm Silicon-On-Thin-BOX (SOTB)," in *IEICE Electronics Express (ELEX)*, vol. 17, no. 20, pp. 20200282, Oct. 2020.

[3] Ba-Anh Dao, Trong-Thuc Hoang, Anh-Tien Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Exploiting the Back-Gate Biasing Technique as a Countermeasure against Power Analysis Attacks," in *IEEE Access*, vol. 9, pp. 24768-24786, Feb. 2021.

[4] Ronaldo Serrano, Ckristian Duran, Trong-Thuc Hoang, Marco Sarmiento, Khai-Duy Nguyen, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "A Fully Digital True Random Number Generator with Entropy Source Based in Frequency Collapse," in *IEEE Access*, vol. 9, pp. 105748-105755, Jul. 2021.

[5] Khai-Duy Nguyen, Dang Tuan Kiet, Trong-Thuc Hoang, Nguyen Quang Nhu Quynh, Xuan-Tu Tran, and Cong-Kha Pham, "A Trigonometric Hardware Acceleration in 32-bit RISC-V Microcontroller with Custom Instruction," in *IEICE Electronics Express (ELEX)*, vol. 18, no. 16, pp. 20210266, Aug. 2021.

[6] Ba-Anh Dao, Trong-Thuc Hoang, Anh-Tien Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Correlation Power Analysis Attack Resisted Cryptographic RISC-V SoC With Random Dynamic Frequency Scaling Countermeasure," in *IEEE Access*, vol. 9, pp. 151993-152014, Nov. 2021.

[7] Anh-Tien Le, Trong-Thuc Hoang, Ba-Anh Dao, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "A Real-Time Cache Side-Channel Attack Detection System on RISC-V Out-of-Order Processor," in *IEEE Access*, vol. 9, pp. 164597–164612, Dec. 2021.

A.2 Conference

[1] Trong-Thuc Hoang, Ckristian Duran, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Cryptographic Accelerators for Trusted Execution Environment in RISC-V Processors," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, virtual conf., Sep. 2020, pp. 1-4.

[2] Ckristian Duran, Trong-Thuc Hoang, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "TEE Boot Procedure with Crypto-accelerators in RISC-V Processors," *Workshop on Computer Arch. Research with RISC-V*, virtual conf., May 2020, pp. 1-4.

[3] Khai-Duy Nguyen, Dang Tuan Kiet, Trong-Thuc Hoang, Nguyen Quang Nhu Quynh, and Cong-Kha Pham, "A CORDIC-based Trigonometric Hardware Accelerator with Custom Instruction in 32-bit RISC-V System-on-Chip," *IEEE Hot Chips Symp. (HCS)*, Palo Alto, CA, USA, Aug. 2021, pp. 1-13.

[4] Trong-Thuc Hoang, Ckristian Duran, Ronaldo Serrano, Marco Sarmiento, Khai-Duy Nguyen, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "System-on-Chip Implementation of Trusted Execution Environment with Heterogeneous Architecture," *IEEE Hot Chips Symp. (HCS)*, Palo Alto, CA, USA, Aug. 2021, pp. 1-16.

[5] Ronaldo Serrano, Ckristian Duran, Trong-Thuc Hoang, Marco Sarmiento, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "ChaCha20-Poly1305 Crypto Core Compatible with Transport Layer Security 1.3," in *Proc. of Int. SoC Design Conf. (ISOCC)*, Jeju Island, South Korea, Oct. 2021, pp. 17-18.

Appendix B

Full Publication List

B.1 Journal

[1] Hong-Thu Nguyen, Xuan-Thuan Nguyen, Trong-Thuc Hoang, Duc-Hung Le, and Cong-Kha Pham, "Low-resource Low-latency Hybrid Adaptive CORDIC with Floating-point Precision," in *IEICE Electronics Express (ELEX)*, vol. 12, no. 9, pp. 20150258, 2015.

[2] Trong-Thuc Hoang, Hong-Kiet Su, Hieu-Binh Nguyen, Duc-Hung Le, Huu-Thuan Huynh, Trong-Tu Bui, and Cong-Kha Pham, "Design of Co-processor for Real-time HMM-based Text-to-speech on Hardware System Applied to Vietnamese," in *IEICE Electronics Express (ELEX)*, vol .12, no. 14, pp. 20150448, 2015.

[3] Trong-Thuc Hoang, Ngoc-Hung Nguyen, and Trong-Tu Bui, "An FPGA-based Implementation of FastICA for Variable-length 4-channel Signal Separation," in *Journal of Science & Tech. on Info. and Comm.*, vol. 1, no. 1, pp. 81-87, 2016.

[4] Hong-Thu Nguyen, Xuan-Thuan Nguyen, Trong-Thuc Hoang, and Cong-Kha Pham, "A CORDIC-based QR decomposition for MIMO signal detector," in *IEICE Electronics Express (ELEX)*, vol .15, no. 6, pp. 20180174, 2018.

[5] Xuan-Thuan Nguyen, Trong-Thuc Hoang, Hong-Thu Nguyen, Katsumi Inoue, and Cong-Kha Pham, "An FPGA-Based Hardware Accelerator for Energy-Efficient Bitmap Index Creation," in *IEEE Access*, vol. 6, pp. 16046-16059, Mar. 2018.

[6] Trong-Thuc Hoang, Duc-Hung Le, and Cong-Kha Pham, "Minimum adder-delay architecture of 8/16/32-point DCT based on fixed-rotation adaptive CORDIC," in *IEICE Electronics Express (ELEX)*, vol. 15, no. 10, pp. 20180302, 2018.

[7] Katsumi Inoue, Trong-Thuc Hoang, and Cong-Kha Pham, "Frequent items counter based on binary decoders," in *IEICE Electronics Express (ELEX)*, vol. 15, no. 20, pp. 20180808, 2018.

[8] Xuan-Thuan Nguyen, Trong-Thuc Hoang, Hong-Thu Nguyen, Katsumi Inoue, and Cong-Kha Pham, "An Efficient I/O Architecture for RAM-based Content-Addressable Memory on FPGA," in *IEEE Trans. on Circ. and Syst. II: Express Briefs (TCAS-II)*, vol. 66, no. 3, pp. 472-476, 2018.

[9] Duc-Hung Le, Trong-Thuc Hoang, and Cong-Kha Pham, "A 1.05-V 62-MHz with 0.12-nW standby power SOTB-65 nm chip of 32-point DCT based on adaptive CORDIC," in *IEICE Electronics Express (ELEX)*, vol. 16, no. 10, pp. 20190116, 2019.

[10] Xuan-Thuan Nguyen, Trong-Thuc Hoang, Hong-Thu Nguyen, Katsumi Inoue, and Cong-Kha Pham, "A 1.2-V 162.9 pJ/cycle bitmap index creation core with 0.31-pW/bit standby power on 65-nm SOTB," in *Microprocessors and Microsystems*, vol. 69, pp. 112-117, Sep. 2019.

[11] Trong-Thuc Hoang, Xuan-Thuan Nguyen, Duc-Hung Le, and Cong-Kha Pham, "Low-Power Floating-Point Adaptive-CORDIC-Based FFT Twiddle Factor on 65-nm Silicon-on-Thin-BOX (SOTB) With Back-Gate Bias," in *IEEE Trans. on Circ. and Syst. II: Express Briefs (TCAS-II)*, vol. 66, no. 10, pp. 1723-1727, Jul. 2019.

[12] Xuan-Thuan Nguyen, Trong-Thuc Hoang, Hong-Thu Nguyen, Katsumi Inoue, and Cong-Kha Pham, "A 0.9-V 50-MHz 256-bit 1D-to-2D-based Single/Multi-match Priority Encoder with 0.67-nW Standby Power on 65-nm SOTB CMOS," in *Microprocessors and Microsystems*, vol. 73, pp. 102970, Mar. 2020.

[13] Trong-Thuc Hoang, Ckristian Duran, Duc-Thinh Nguyen-Hoang, Duc-Hung Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Quick Boot of Trusted Execution Environment with Hardware Accelerators," in *IEEE Access*, vol. 8, pp. 74015-74023, 2020.

[14] Trong-Thuc Hoang, Ckristian Duran, Khai-Duy Nguyen, Tuan-Kiet Dang, Quynh Nguyen Quang Nhu, Phuc Hong Than, Xuan-Tu Tran, Duc-Hung Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Low-power High-performance 32-bit RISC-V Microcontroller on 65-nm Silicon-On-Thin-BOX (SOTB)," in *IEICE Electronics Express (ELEX)*, vol. 17, no. 20, pp. 20200282, Oct. 2020.

[15] Ba-Anh Dao, Trong-Thuc Hoang, Anh-Tien Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Exploiting the Back-Gate Biasing Technique as a Countermeasure against Power Analysis Attacks," in *IEEE Access*, vol. 9, pp. 24768-24786, Feb. 2021.

[16] Ronaldo Serrano, Ckristian Duran, Trong-Thuc Hoang, Marco Sarmiento, Khai-Duy Nguyen, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "A Fully Digital True Random Number Generator with Entropy Source Based in Frequency Collapse," in *IEEE Access*, vol. 9, pp. 105748-105755, Jul. 2021.

[17] Dang Tuan Kiet, Binh Kieu-Do-Nguyen, Trong-Thuc Hoang, Khai-Duy Nguyen, Xuan-Tu Tran, and Cong-Kha Pham, "A Proposal for Enhancing

Training Speed in Deep Learning Models Based on Memory Activity Survey," in *IEICE Electronics Express (ELEX)*, vol. 18, no. 15, pp. 20210252, Aug. 2021.

[18] Khai-Duy Nguyen, Dang Tuan Kiet, Trong-Thuc Hoang, Nguyen Quang Nhu Quynh, Xuan-Tu Tran, and Cong-Kha Pham, "A Trigonometric Hardware Acceleration in 32-bit RISC-V Microcontroller with Custom Instruction," in *IEICE Electronics Express (ELEX)*, vol. 18, no. 16, pp. 20210266, Aug. 2021.

[19] Marco Sarmiento, Khai-Duy Nguyen, Ckristian Duran, Trong-Thuc Hoang, Ronaldo Serrano, Van-Phuc Hoang, Xuan-Tu Tran, Koichiro Ishibashi, and Cong-Kha Pham, "A Sub-μW Reversed-Body-Bias 8-bit Processor on 65-nm Silicon-On-Thin-Box (SOTB) for IoT Applications," in *IEEE Trans. on Circ. and Syst. II: Express Briefs (TCAS-II)*, vol. 68, no. 9, pp. 3182-3186, Jun. 2021.

[20] Ba-Anh Dao, Trong-Thuc Hoang, Anh-Tien Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Correlation Power Analysis Attack Resisted Cryptographic RISC-V SoC With Random Dynamic Frequency Scaling Countermeasure," in *IEEE Access*, vol. 9, pp. 151993-152014, Nov. 2021.

[21] Anh-Tien Le, Trong-Thuc Hoang, Ba-Anh Dao, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "A Real-Time Cache Side-Channel Attack Detection System on RISC-V Out-of-Order Processor," in *IEEE Access*, vol. 9, pp. 164597–164612, Dec. 2021.

[22] Ryuichi Tsutada, Trong-Thuc Hoang, and Cong-Kha Pham, "An Obstacle Avoidance Two-Wheeled Self-Balancing Robot," in Int. Journal of Mechanical Engi. and Robotics Research (IJMERR), vol. 11, no. 1, pp. 1-7, Jan. 2022.

B.2 Conference

[1] Trong-Thuc Hoang, Vi-Thuy Tran, Thanh Le, Minh-Triet Luu, Cao-Quyen Tran, Xuan-Thuan Nguyen, and Trong-Tu Bui, "A Case Study of Connect6 Game FPGA-based Implementation Using the Multi-turn Prediction Algorithm," *IEEE Int. Symp. on Signal Processing and Info. Tech. (ISSPIT)*, Hochiminh City, Vietnam, 2012.

[2] Trong-Thuc Hoang, Ngoc-Hung Nguyen, Xuan-Thuan Nguyen, and Trong-Tu Bui, "A Real-time Object-recognition System Based on PCNN Algorithm," *IEICE Int. Conf. on Integrated Circ. and Devices in Vietnam (ICDV)*, no. 2, Danang, Vietnam, 2012, pp. 155-160.

[3] Kim-Hung Nguyen, Trong-Thuc Hoang, and Trong-Tu Bui, "An FSM-based IP Protection Technique Using Added Watermarked States," *Int. Conf. on Advanced Tech. for Comm. (ATC)*, Hochiminh City, Vietnam, Oct. 2013, pp. 718-723.

[4] Thai-Bao Huynh, Trong-Thuc Hoang, and Trong-Tu Bui, "A Constraint-based Watermarking Technique Using Schmitt Trigger Insertion at Logic Synthesis Level," *Int. Conf. on Advanced Tech. for Comm. (ATC)*, Hochiminh City, Vietnam, Oct. 2013, pp. 115-120.

[5] Trong-Thuc Hoang, Quang-Trung Tran, and Trong-Tu Bui, "A Proposed Adaptive Image Segmentation Method Based on Local Excitatory Global Inhibitory Region Growing," *IEEE Int. Conf. on Comm. and Elec. (ICCE)*, Danang, Vietnam, Jul. 2014, pp. 458-463.

[6] Xuan-Vy Luu, Trong-Thuc Hoang, Trong-Tu Bui, and Anh-Vu Dinh-Duc, "A High-speed Unsigned 32-bit Multiplier Based on Booth-encoder and Wallace-tree Modifications," *Int. Conf. on Advanced Tech. for Comm. (ATC)*, Hanoi, Vietnam, Oct. 2014, pp. 739-744.

[7] Dinh-Thien Vu, Trong-Thuc Hoang, Ngoc-Hung Nguyen, and Trong-Tu Bui, "An FPGA-based Variable-length 4-Channel Separation FastICA Implementation," *IEICE Int. Conf. on Integrated Circ. and Devices in Vietnam (ICDV)*, Hanoi, Vietnam, 2014, pp. 739-744.

[8] Hong-Thu Nguyen, Xuan-Thuan Nguyen, Trong-Thuc Hoang, Duc-Hung Le, and Cong-Kha Pham, "A Low-resource Low-latency Hybrid Adaptive CORDIC in 180-nm CMOS Technology," *IEEE Region 10 Conf. (TENCON)*, Singapore, Singapore, Nov. 2015, pp. 1-4.

[9] Trong-Thuc Hoang, Duc-Hung Le, Hong-Thu Nguyen, Xuan-Thuan Nguyen, Cong-Kha Pham, "A Hybrid Adaptive CORDIC in 65nm SOTB CMOS Process," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, Montreal, Canada, May 2016, pp. 2158-2161.

[10] Xuan-Thuan Nguyen, Hong-Thu Nguyen, Trong-Thuc Hoang, Katsumi Inoue, Osamu Shimojo, Toshio Murayama, Kenji Tominaga, and Cong-Kha Pham, "An Efficient FPGA-based Batabase Processor for Fast Database Analytics," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, Montreal, Canada, May 2016, pp. 1758-1761.

[11] Trong-Thuc Hoang, Hong-Thu Nguyen, Xuan-Thuan Nguyen, Cong-Kha Pham, and Duc-Hung Le, "High-performance DCT Architecture Based on Angle Recoding CORDIC and Scale-Free Factor," *IEEE Int. Conf. on Comm. and Elec. (ICCE)*, Halong, Vietnam, Jul. 2016, pp. 199-204.

[12] Hong-Thu Nguyen, Xuan-Thuan Nguyen, Cong-Kha Pham, Trong-Thuc Hoang, Duc-Hung Le, "A Parallel Pipeline CORDIC Based on Adaptive Angle Selection," *Int. Conf. on Elec., Info., and Comm. (ICEIC)*, Danang, Vietnam, Jan. 2016, pp. 1-4.

[13] Trong-Thuc Hoang, Xuan-Thuan Nguyen, Hong-Thu Nguyen, Nhu-Quynh Truong, Duc-Hung Le, Katsumi Inoue, and Cong-Kha Pham, "FPGA-based Frequent Items Counting Using Matrix of Equality Comparators," *IEEE Int. Midwest Symp. on Circ. and Syst. (MWSCAS)*, Boston, MA, USA, Aug. 2017, pp. 285-288.

[14] Phuong-Thao Vo-Thi, Trong-Thuc Hoang, Cong-Kha Pham, and Duc-Hung Le, "A Floating-point FFT Twiddle Factor Implementation Based on Adaptive Angle Recoding CORDIC," *Int. Conf. on Recent Advances in Signal Processing, Telecomm. & Computing (SigTelCom)*, Danang, Vietnam, Jan. 2017, pp. 21-26.

[15] Trong-Thuc Hoang, Cong-Kha Pham, and Duc-Hung Le, "High-speed 8/16/32-point DCT Architecture Using Fixed-rotation Adaptive CORDIC," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1-5.

[16] Xuan-Thuan Nguyen, Trong-Thuc Hoang, Hong-Thu Nguyen, Katsumi Inoue, and Cong-Kha Pham, "A 219-μW 1D-to-2D-Based Priority Encoder on 65-nm SOTB CMOS," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1-4.

[17] Katsumi Inoue, Trong-Thuc Hoang, Xuan-Thuan Nguyen, Hong-Thu Nguyen, Cong-Kha Pham, "VLSI Design of Frequent Items Counting Using Binary Decoders Applied to 8-bit per Item Case-study," *Conf. on Ph.D. Research in Microelec. and Elec. (PRIME)*, Prague, Czech Republic, Jul. 2018, pp. 161–164.

[18] Trong-Thuc Hoang, Duc-Hung Le, and Cong-Kha Pham, "VLSI Design of Floating-Point Twiddle Factor Using Adaptive CORDIC on Various Iteration Limitations," *Int. Symp. on Embedded Multicore/Many-core SoCs (MCSoC)*, Hanoi, Vietnam, Sep. 2018, pp. 225-232.

[19] Takahiro Hosaka, Trong-Thuc Hoang, Van-Phuc Hoang, Duc-Hung Le, Katsumi Inoue, and Cong-Kha Pham, "Live Demonstration: Real-time Auto-exposure Histogram Equalization Video-system Using Frequent Items Counter," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, Sapporo, Japan, May 2019, pp. 1-1.

[20] Xuan-Thuan Nguyen, Trong-Thuc Hoang, Katsumi Inoue, Ngoc-Tu Bui, Van-Phuc Hoang, Cong-Kha Pham, "A 1.2-V 90-MHz Bitmap Index Creation Accelerator with 0.27-nW Standby Power on 65-nm Silicon-On-Thin-Box (SOTB) CMOS," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, Sapporo, Japan, May 2019, pp. 1-4.

[21] Ngoc-Tu Bui, Trong-Thuc Hoang, Duc-Hung Le, and Cong-Kha Pham, "A 0.75-V 32-MHz 181-μW SOTB-65nm Floating-point Twiddle Factor Using Adaptive CORDIC," *IEEE Int. Conf. on Industrial Tech. (ICIT)*, Melbourne, Australia, Feb. 2019, pp. 835-840.

[22] Ngoc-Tu Bui, Trong-Thuc Hoang, Akinori Yamamoto, Duc-Hung Le, and Cong-Kha Pham, "A 0.75-V 58-MHz 340-μW SOTB-65nm 32-point DCT Implementation Based on Fixed-rotation Adaptive CORDIC," *IEEE SOI-3D-Subthreshold Microelectronics Tech. Unified Conf. (S3S)*, San Jose, CA, USA, Oct. 2019, pp. 1-3.

[23] Akinori Yamamoto, Trong-Thuc Hoang, Cong-Kha Pham, "A Ring Oscillator Using Bootstrap Inverter," *IEEE SOI-3D-Subthreshold Microelectronics Tech. Unified Conf. (S3S)*, San Jose, CA, USA, Oct. 2019, pp. 1-2.

[24] Ckristian Duran, Trong-Thuc Hoang, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "TEE Boot Procedure with Crypto-accelerators in RISC-V Processors," *Workshop on Computer Arch. Research with RISC-V*, virtual conf., May 2020, pp. 1-4.

[25] Trong-Thuc Hoang, Ckristian Duran, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "Cryptographic Accelerators for Trusted Execution Environment in RISC-V Processors," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, virtual conf., Sep. 2020, pp. 1-4.

[26] Khai-Duy Nguyen, Dang Tuan Kiet, Trong-Thuc Hoang, Nguyen Quang Nhu Quynh, and Cong-Kha Pham, "A CORDIC-based Trigonometric Hardware Accelerator with Custom Instruction in 32-bit RISC-V System-on-Chip," *IEEE Hot Chips Symp. (HCS)*, Palo Alto, CA, USA, Aug. 2021, pp. 1-13.

[27] Trong-Thuc Hoang, Ckristian Duran, Ronaldo Serrano, Marco Sarmiento, Khai-Duy Nguyen, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "System-on-Chip Implementation of Trusted Execution Environment with Heterogeneous Architecture," *IEEE Hot Chips Symp. (HCS)*, Palo Alto, CA, USA, Aug. 2021, pp. 1-16.

[28] Binh Kieu-Do-Nguyen, Trong-Thuc Hoang, Cong-Kha Pham, and Cuong Pham-Quoc, "A Power-efficient Implementation of SHA-256 Hash Function for Embedded Applications," in *Proc. of Int. Conf. on Advanced Tech. for Comm. (ATC)*, Hochiminh city, Vietnam, Oct. 2021, pp. 39-44.

[29] Ronaldo Serrano, Marco Sarmiento, Ckristian Duran, Khai-Duy Nguyen, Trong-Thuc Hoang, Koichiro Ishibashi, and Cong-Kha Pham, "A Low-Power Low-Area SoC based in RISC-V Processor for IoT Applications," in *Proc. of Int. SoC Design Conf. (ISOCC)*, Jeju Island, South Korea, Oct. 2021, pp. 375-376.

[30] Ronaldo Serrano, Ckristian Duran, Trong-Thuc Hoang, Marco Sarmiento, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham, "ChaCha20-Poly1305 Crypto Core Compatible with Transport Layer Security 1.3," in *Proc. of Int. SoC Design Conf. (ISOCC)*, Jeju Island, South Korea, Oct. 2021, pp. 17-18.

Author Biography

Trong-Thuc Hoang received the B.Sc. degree in Electronics and Telecommunications from the University of Science, Vietnam National University, Hochiminh city (HCMUS), Vietnam, in 2012 and the M.S. degree in Microelectronics from the same university in 2017. From 2012 to 2017, he was a researcher and lecturer assistant in the Faculty of Electronics and Telecommunications, HCMUS. Since 2017, he has been a Ph.D. student in the Department of Computer and Network Engineering at the University of Electro-Communications (UEC), Tokyo, Japan. From 2019 to 2020, he was a researcher assistant at PHAM laboratory, UEC. Since 2019, he has also been a researcher assistant at the Cyber-Physical Security Research Center (CPSEC), National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan. His research interests include computer science, computer security, and VLSI design. He is a member of IEEE and IEICE.

THE END.