ENEL 592 - Final Project Report

Table of Contents

Introduction

The aim of my ENEL 592 final project is to insert a set of security bugs into an System-on-Chip (SoC) design, and create associated testbenchs and firmware that demonstrate their implications. This is the culmination of my two previous assignments, where I surveyed hardware security verification and open-source SoC designs. The bugs should be as "realistic" as possible; they should resemble bugs found in-the-wild and be impactful.

Next semester, I will build on this project and approach the problem from the other side of the coin -- bug detection and/or correction. The resulting SoC will also serve as a good testbench for this future work.

System-on-Chip Platform

The SoC I will be using for bug injection is the OpenTitan SoC, which I detailed in assignment 2. An excerpt of assignment 2 describing the OpenTitan SoC can be found in the appendix A.

In summary, OpenTitan is ...

Bug Selection

Insertion Method

Inserted Bugs

Bug 1:

Conclusion

Appendix A: OpenTitan

The OpenTitan SoC homepage can be found here, the documentation here, and the GitHub repository containing all source code here. OpenTitan is an open-source Root-of-Trust (RoT) SoC maintained by lowRISC and Google. It is the only open-source RoT currently available, making it an interesting case study for this assignment as it contains extensive security features and documentation. It implements various cryptographic

hardware, such as the Advanced Encryption Standard (AES), HMAC, KMAC, and security countermeasures like access control to ensure the Confidentiality, Integrity, and Availability (CIA) of its functions.

This design is the most promising out of the box, and fulfills all three criteria.

The OpenTitan SoC is detailed below.

1.3.6. Summary

The final list of designs inspected is as summarized below:

Table 2: Summary of Projects Inspected

According to the criteria outlined previously, I determined that OpenTitan was the most promising design for my purposes. In the following section, I discuss the SoC in detail.

1.4. Selected Design: OpenTitan

OpenTitan Earl Grey Features

- RV32IMCB RISC-V "Ibex" core:
  - 3-stage pipeline, single-cycle multiplier
  - Selected subset of the bit-manipulation extension
  - 4kB instruction cache with 2 ways
  - RISC-V compliant JTAG DM (debug module)
  - PLIC (platform level interrupt controller)
  - U/M (user/machine) execution modes
  - Enhanced Physical Memory Protection (ePMP)
  - Security features:
    - Low-latency memory scrambling on the icache
    - Dual-core lockstep configuration
    - Data independent timing
    - Dummy instruction insertion
    - Bus and register file integrity
    - Hardened PC
- Security peripherals:
  - AES-128/192/256 with ECB/CBC/CFB/OFB/CTR modes
  - HMAC / SHA2-256
  - KMAC / SHA3-224, 256, 384, 512, [c]SHAKE-128, 256
  - Programmable big number accelerator for RSA and ECC (OTBN)
  - NIST-compliant cryptographically secure random number generator (CSRNG)
  - Digital wrapper for analog entropy source with FIPS and CC-compliant health checks
  - Key manager with DICE support
  - Manufacturing life cycle manager
  - Alert handler for handling critical security events
  - OTP controller with access controls and memory scrambling
  - Flash controller with access controls and memory scrambling
  - ROM and SRAM controllers with low-latency memory scrambling
- Memory:
  - 2x512kB banks eFlash
  - 128kB main SRAM
  - 4KB Always ON (AON) retention SRAM
  - 32kB ROM
  - 2kB OTP
- IO peripherals:
  - 47x multiplexable IO pads with pad control
  - 32x GPIO (using multiplexable IO)
  - 4x UART (using multiplexable IO)
  - 3x I2C with host and device modes (using multiplexable IO)
  - SPI device (using fixed IO) with TPM, generic, flash and passthrough modes
  - 2x SPI host (using both fixed and multiplexable IO)
- Other peripherals:
  - Clock, reset and power management
  - Fixed-frequency timer
  - Always ON (AON) timer
  - Pulse-width modulator (PWM)
  - Pattern Generator
- Software:
  - Boot ROM code implementing secure boot and chip configuration
  - Bare metal applications and validation tests

Figure 1: OpenTitan Features

The OpenTitan project has well defined and documented threat models and countermeasures. They outline the secure assets, adversary, attack surfaces, and attack methods. The assets are mainly centered around the

cryptopgraphic keys, with other loosely defined statements such as "Integrity and authenticity of stored data".

The adversaries they consider are (i) a bad actor with physical access to the device during fabrication or deployment, (ii) a malicious device owner, (iii) malicious users with remote access.

1.4.1. Architecture

The OpenTitan SoC's architecture follows the standard Network-on-Chip (NoC) design paradigm, with various IP cores interconnected a high-speed communication protocol allowing them to communicate with one another. The processor is able to configure and use the pheripherals by writing and reading to memory-mapped IO registers.

The interconnect responsible for connecting all IP cores is a TileLink Uncached Lightweight (TL-UL) crossbar which is autogenerated using a custom crossbar generation tool. The top-level module dubbed *Earl Grey*, is also auto-generated using a top generation tool. Both tools are configured by using hjson files that are scattered throughout the project.

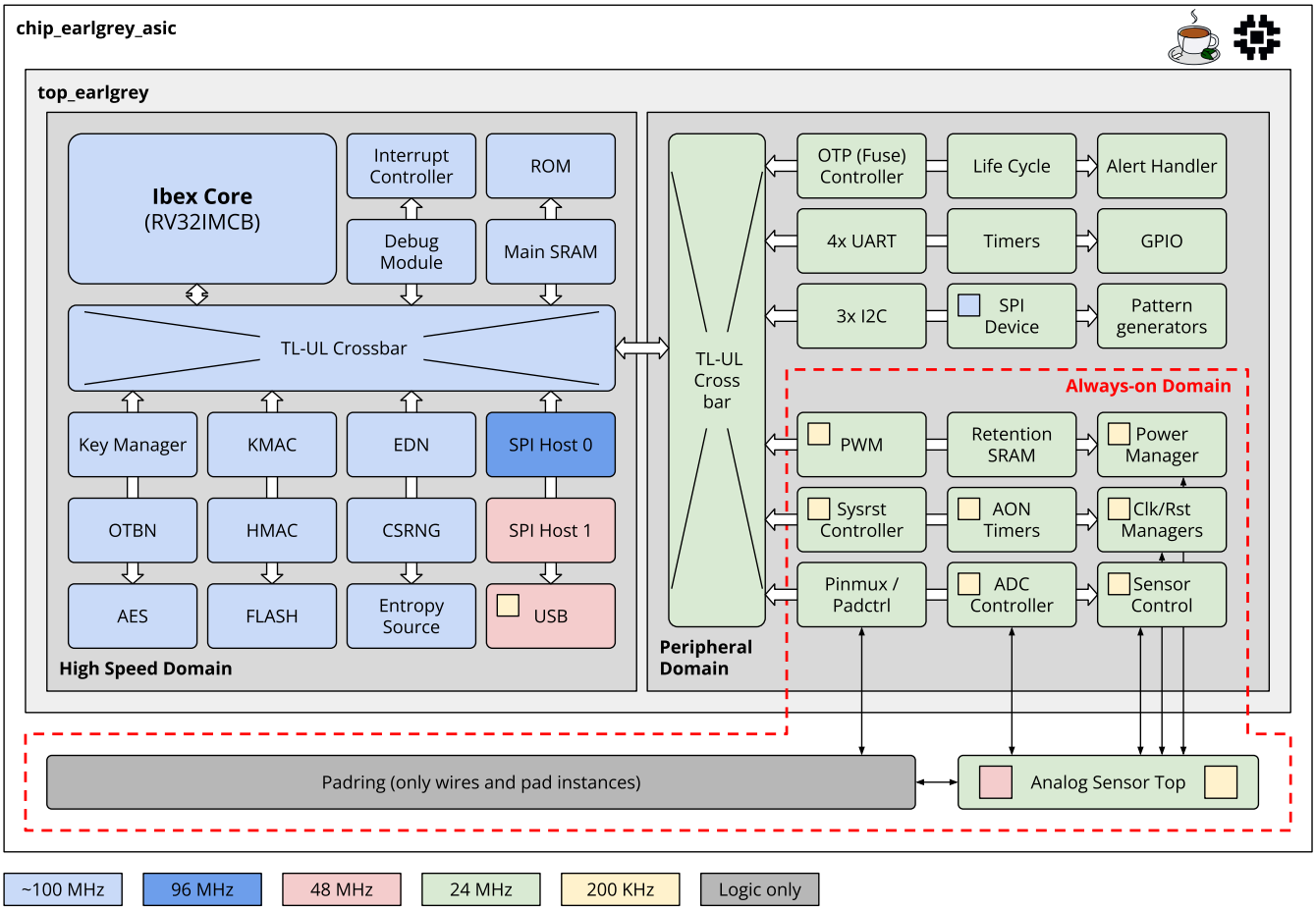

Figure 2: OpenTitan EarlGrey Top

The memories are integrated in the chip with configurable size and address. By default, the instruction ROM is 32 kB, the flash is 1024 kB, and SRAM is 128 kB. The processor core used is the RISC-V Ibex core which we discuss here. As seen in Figure 1, the SoC is seperated into high speed and peripheral domains, with many of its critical functions residing in the high speed domain.

It also provides bebug functionality by way of the RISC-V debug specification 0.13.2 and the JTAG TAP specification.

1.4.2. Security Features

As a RoT, the OpenTitan SoC implements various security features. Outside of its secure cryptopgraphic functions, it also provides a secure boot flow that integrates multiple memory integrity checks, various access control measures such as lock bits for pheripheral configuration registers and memory regions, an integrity scheme integrated into the TL-UL crossbar, and security alerts that are triggered under defined conditions that suggest suspicious behaviour.

There is curently no detailed documentation for the secure boot flow available, but at a high level, on boot-up the hard-coded instructions in the ROM memory are used for platform checking and memory integrity checking. At this stage, the integrity of the full contents of the non-hard-coded bootloader in the Flash memory is checked by an RSA-check algorithm.

Another fundamental piece of memory which is not directly mentioned in the secure boot process is the one time programmable (OTP) memory. An OTP controller is provided but the OTP IP (fuse memory) must be source externally. Together, they provide secure one-time-programming functionality that is used throughout the life cycle (LC) of a device. The OTP is broken up in partitions responsible for storing different attributes of the device. The specific attributes for each partition (and the partition themselves) are configurable and will likely vary widely for different applications. Critical data stored in the OTP include the root keys used to derive all other keys for cryptographic functions and memory scrambling.

The end-to-end cross integrity scheme consists of additional signals embedded into the interconnect that ensures the integrity of data as it travels through the SoC. There is no detailed documentation on its operation yet. From what is available -- the integrity scheme is an extension of the TL-UL specification and consists of 2 additional `SystemVerilog buses` that carry the "integrity" of the data, which is checked by the consumer. From inspecting the design, the integrity scheme utilizes uses Hsiao code (modified version of Hamming code + parity) as its error-detection code.

On the cryptographic side, the relevant IPs comprise of the Key Manager, KMAC, HMAC, AES, the Entropy source, EDN, and CSRNG. The key manager is responsible for generating the keys used for all cryptographic operations and identification. On reset, it rejects all software requests until it is initialized again. Initialization consists of first loading in random values from the entropy source then the root key from the OTP. This ensures that the hamming delta (the difference in hamming weights between the random number and the root key) are non-deterministic and the root key is thus not susceptible to power side-channel leakage **(This is my interpretation, I am probably wrong)**. The key manager interatively completes KMAC operations using the KMAC IP to progress to different states and generate different keys. The states transistions of the Key Manager are illustrated in Figure 3. The Key manager implements various security countermeasures such as sparse FSM encoding, and automatic locking of configuration registers during operation.

The Keccak Message Authentication Code (KMAC) IP core is a Keccak-based message authentication code generator to check the integrity of an incoming message and a signature signed with the same secret key. It implements the NIST FIPS 202 SHA-3 standard. The secret key length can vary up to 512 bits. The KMAC generates at most 1600 bits of the digest value at a time which can be read from the STATE memory region. It also implements masked storage and Domain-Oriented Masking (DOM) inside the Keccak function to protect against 1st-order SCA attacks. As mentioned earlier, the KMAC core is used extensively by the key manager. Its security countermeasures include sparse FSM encoding, counter redundancy, and lock bits to ensure configuration registers are not written during operation.

The Keyed-Hash Message Authentication Code (HMAC) IP Core implements the SHA256 hashing algorithm. It achieves similar functions to the KMAC core but is not hardened against power side-channels. It is meant as a faster alternative to the KMAC core. It does not contain any security countermeasures other than the bus integrity scheme present in all IP.

The final cryptographic core is the AES accelerator responsible for all encryption/decryption operations of the SoC. It implements NIST's Advanced Encryption Standard. It supports multiple standard block modes of operation (ECB, CBC, CFB, OFB, CTR) and 128/192/256-bit key sizes. The accelerator implements the same masking scheme as the KMAC core to protect itself against 1st order side-channel attacks. It also implements many other security countermeasures: lock bits, clearing of sensitive registers after operation, sparse FSM and control register encoding, and logic rail redundancy for FSMs.

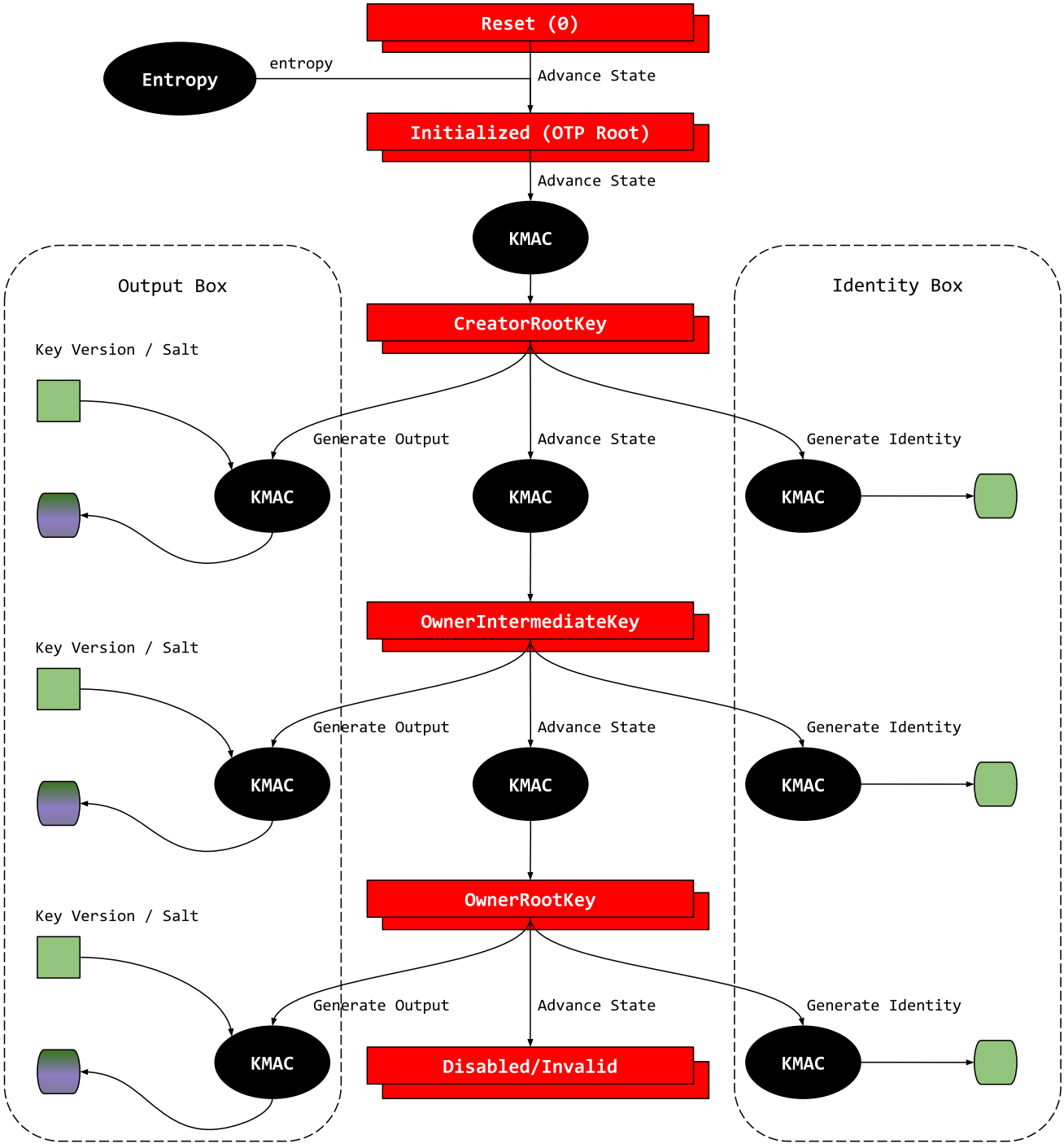

Figure 3: Key Manager State Transitions

Finally, the ROM, SRAM, and Flash controllers manage accesses to memory. They integrate multiple security features.

The ROM controller contains a startup checker which verify the integrity of its contents by utilizing the KMAC IP to hash all but the 8 top words of its data. The hash received from the KMAC operation is then compared to the 8 top words. The read addresses are passed through a substitution and permutation (S&P) block then passed to the ROM memory and a PRINCE cipher in parallel. The pre-scrambled data read from the ROM is also passed through an S&P block, and XOR from the results of the PRINCE cipher to obtain the final read data.

The data in the SRAM is also scrambled in similar fashion to the ROM, and additionally contains 7 integrity bits for each 32 bit word. It also provides a Linear Feedback Shift Register (LFSR) block to feature that can overwrite the entire memory with pseudorandom data via a software request.

The flash controller provides also optional memory scrambling and integrity bits. It also provides up to software-configurable 8 memory regions with configurable access policies.

1.4.3. Collateral

The OpenTitan SoC provides extensive collateral. Collateral in this context, refers to any additional information that describes the functionality of a design and its components. The collateral for this SoC consists of the documentation for all of its IP and contains its security features, interfaces, interactions with software, testplans, and block diagrams. Unique to this SoC are the hjson files that describe all of an IP's parameters, registers, security countermeasures, etc. This is extremely useful to obtain designer context behind the design. For example, from the AES hjson file, we can understand the function of parameter `SecMasking`, as shown in figure 4.

```
{ name:    "SecMasking",
  type:    "bit",
  default: "1'b1",
  desc:    '''
    Disable (0) or enable (1) first-order masking of the AES cipher core.
    Masking requires the use of a masked S-Box, see SecSBoxImpl parameter.
  '''
  local:   "false",
  expose:  "true"
},
```

Figure 4: AES SecMasking .hjson snippet

Another aspect of collateral is the test environment provided. OpenTitan currently provides automated Dynamic Verification (DV) for all IP which perform simulate the IP and perform automated checks using a Golden Reference model. They also an FPV test suite using SystemVerilog Assertions which mainly verify the compliance to the TL-UL protocol. The SoC was setup locally with relative ease, thanks to the detailed instructions and reliable scripts, and the UVM tests were successful run using Verilator.

*[SoC]: System-on-Chip