PULPino attack scenarios & Defense

How to hack the *PULPino* platform? And, defend it.

Zhiyang Ong

Department of Electrical and Computer Engineering
Dwight Look College of Engineering,
Texas A&M University
College Station, TX

March 20, 2018

Table of Contents

Acknowledgments

Dott. Francesco Stefanni, University of Verona

Teammates:

- Sheena Goel
- Saumil Pankajbhai Gogri
- Bhavani Bedre Shankar

Contest Advisor: Mike Quinn.

People on Hack DAC '18 *Slack* channel:

- Ghada Dessouky

Warnings!!!

"The code is rather buggy; use is at own risk."
– Donald Chai

"Beware of bugs in the above code.
I have only proved it correct, not tried it."
– Donald E. Knuth

Unlike my Fantastic and Fabulous teammates, I have yet to implement these attack scenarios.

Table of Contents

System Architecture

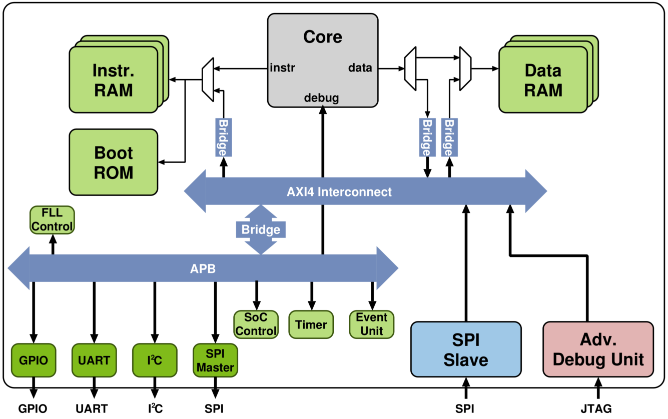

Figure: Pulpino system architecture

Table of Contents

Memory Map

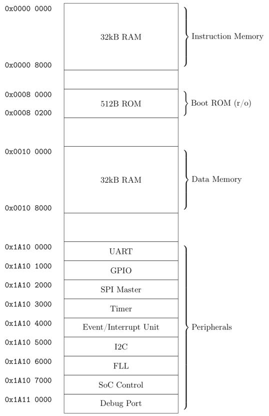

Figure: Pulpino memory map

Table of Contents

Topics in Hardware Security (1)

Supply-side attacks for supply-chain hardware security

- hardware backdoor attacks
    - hardware trojans, or malicious circuits
        - add, modify, or delete data
        - do something unintended
        - don't do something intended
    - malicious firmware (insertion & execution)
    - Test and verification features
        - Advanced Debug Unit (for JTAG)
        - Debug port for APB-based memory mapped debugging

Topics in Hardware Security (2)

More supply-side attacks for supply-chain hardware security

- Attackers
  - third-party RTL vendors
  - disgruntled employees of IC design team
  - EDA vendors can produce malicious/defective circuits
  - Everyone in downstream of supply-chain are liable
- Hardware piracy
- Counterfeit attack

Table of Contents

High-level attack scenarios (1)

- Assume that Boot ROM contains initial values
  - Access to Boot ROM
    - Read from data memory (RAM) instead.
    - Read “garbage” (maliciously inserted instructions) from instruction memory (RAM) instead.
    - Read data standard input port instead (software-driven process/operation).
    - Unauthenticated read access to boot ROM outside of boot up procedure.
  - Writing to Boot ROM should always be forbidden.
- If authentication is required for read access to the boot ROM,
  - Only the authorized read access (in normal mode or debug/testing mode) during boot up or rebooting is allowed.
  - Else, forbid read access outside of booting up or rebooting is allowed.

High-level attack scenarios (2)

Read/Write access data in data/instruction RAM via input/output ports, without authentication (1):

- Input/Output (I/O) ports and buses to consider
    - General-purpose input/output (GPIO)
    - UART
    - SPI Master / AXI Master, and SPI Slave; see Serial Peripheral Interface bus (SPI)
- Read "important" data that I should not access, without authentication; e.g., keys for encryption/decryption.

High-level attack scenarios (3)

Read/Write access data in data/instruction RAM via input/output ports, without authentication (2):

- Fill up the data and/or instruction RAM with garbage, and/or malicious data and instructions.
  - Invalid instructions fetched and decoded by the RISV-V processor(s) should result in raising an illegal instruction exception at address 0x00000084 (§2.1, Table 2.1, page 7 of the datasheet)
- Invalid memory access (i.e., areas not specified in the memory map) shall raise an invalid memory access exception at address 0x0000008C (§2.1, Table 2.1, page 7 of the datasheet).
- Load malicious firmware to the instruction RAM:
  - add, modify, or delete data
  - do something unintended
  - don't do something intended

Table of Contents

Abstract Attack Scenario for Hardware Trojans or Malicious Circuits (1)

- Attack objectives/goals, or
  - read (G1), add (G2), modify (G3), or delete (G4) data
  - do something unintended (G5)
  - don't do something intended (G6)
- Man-in-the-middle attack: G1, G2, G3, G4, G5, G6
- Backdoor attack: G1, G2, G3, G4, G5, G6
- Eavesdropping attack: G1
- Privilege escalation: G1, G2, G3, G4, G5, G6
- Side-channel attack: G1, G2, G3, G4, G5, G6

Table of Contents

Abstract Attack Scenario for Hardware Test and Verification Features (1)

- Trade-off between hardware security and testability/verifiability
- An attacker (*Mallory* [Wikipedia contributors 2016]) can exploit the observability and controllability of the system that VLSI verification and testing need.
- *Mallory* exploits the observability and controllability of the system, via its debug and test modes/features.

Abstract Attack Scenario for Hardware Test and Verification Features (2)

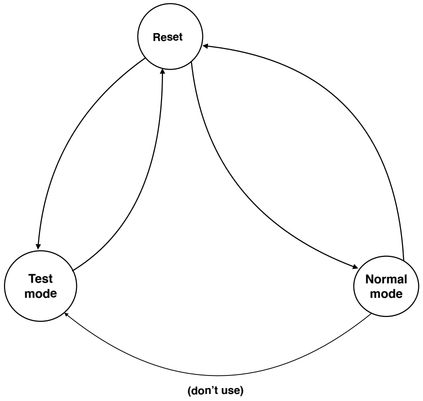

Figure: Unsafe Finite State Machine Model of a System-Under-Test

Abstract Attack Scenario for Hardware Test and Verification Features (3)

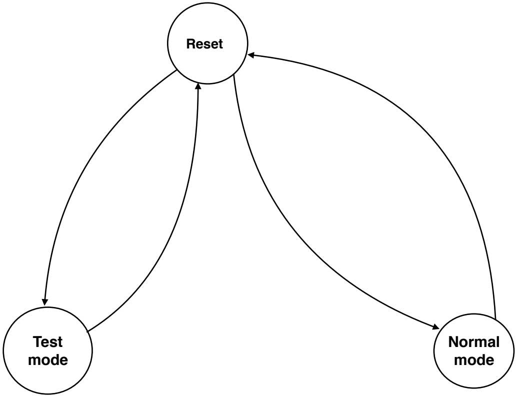

Figure: Safer Finite State Machine Model of a System-Under-Test

Attack Scenario for JTAG #1: Switch directly from normal mode to test mode (1)

- Set input signal tms (test mode select), so that it transits from the “idle” state/mode to the “test” state/mode, in the state machine of the test access port (TAP) controller (or JTAG state machine).
  - At “idle” state, $TMS = 0$ will cause the TAP controller to remain the the “idle” state.
  - From the “idle” state, $TMS = 1$ will cause the TAP controller to transit to the “run test” state, without going to the “test logic reset” state.
- According to Prof. Jeyavijayan (JV) Rajendran, an updated JTAG standard requires/forces the TAP controller to transit from “idle” state to the “run test” state, via the “reset” state.
  - See Table 10.1 about the IEEE 1149 standard family [Wang 2006]

Attack Scenario for JTAG #1: Switch directly from normal mode to test mode (2)

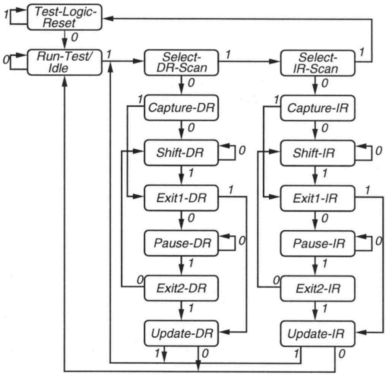

Figure: Finite state machine of the JTAG test access port (TAP) controller [Bushnell 2000, §16.2.1, Figure 16.11, pp. 558]

Attack Scenario for JTAG #1: Switch directly from normal mode to test mode (3)

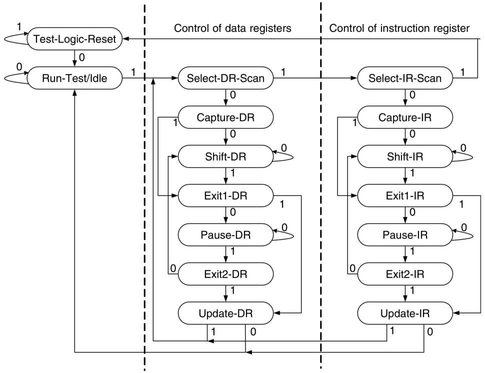

Figure: Finite state machine of the JTAG test access port (TAP) controller [Wang 2006, §10.2.5, Figure 10.7, pp. 567]

Attack Scenario for JTAG #2: Access Boot ROM

- Assume that Boot ROM contains initial values
- Without authentication, access the boot ROM.
    - If this is feasible, it may be a security bug if security-related functions are performed during booting/rebooting.
- With authentication, modify the boot ROM.
    - This attempt should fail.
    - Else, it is a security bug, since malicious instructions can be inserted into the boot ROM.

Attack Scenario for JTAG #2: Access Instruction and Data RAMs

- Without authentication, scan the data from the instruction RAM to the instruction register (IR), and subsequently scan the instructions out (to an output data acquisition device).
  - If this is feasible, it may be a security bug if security-related functions are performed during operation of the *Pulpino* platform.
- Without authentication, scan the data from the data RAM to the data registers (DR), and subsequently scan the data out (to an output data acquisition device).
  - If this is feasible, it may be a security bug if private keys (for encryption/decryption) are stored in the data RAM without encryption.
- With authentication, the aforementioned tasks should fail.

Attack Scenario for Debug Port

- In debug mode
  - Insert malicious firmware into instruction RAM
    - Unauthorized access to, addition to, modification of, or deletion of data
    - To do something unintended
    - To avoid doing something intended
- Gain unauthorized access to data
  - Add, modify, or delete data
  - Send data to output device
- When authorized, ensure that data cannot be modified or accessed inappropriately.

Table of Contents

Attack Scenario Regarding Privileges

- Hardware trojan can generate an interrupt to write value into a Control/Status Register (CSR), via I^2C (Inter-Integrated Circuit).
- This value may not be the true/actual value of the CSR and/or interrupt handler.
- [Khabarov 2015]

Attack Scenario Regarding Race Conditions

- Use the frequency lock loop (FLL) “to generate clock frequencies faster [or slower???] than the reference clock frequency” (Ghada Dessouky, Hack DAC ’18 *Slack* channel)
- Can cause race conditions and race hazards
- Can cause timing violations of registers (i.e., flip-flops and latches)
- Cause the system to do something unintended

Table of Contents

Inadequately Addressed Hardware Security Challenges

Hardware trojans, or malicious circuits, that:

- Do something unintended (not in the design specifications).
- Carry out unauthorized access, addition, modification, or deletion of data
- Don't do something intended.
- Check the connections (i.e., I/O ports) of buses (i.e., APB & AXI interconnect) for unspecified connections.

Table of Contents

How to Detect Hardware Trojans (1)

- Walk through code (i.e., code walkthrough)... Tedious.
- Use verification-based techniques to detect security bugs.
  - Tailor these techniques to specific VLSI circuit/system designs, or family of such designs.
  - *Bring the FSM to an invalid state???*
- Proof-carrying code techniques, which were initially developed for software security vulnerabilities
  - Use divide & conquer to partition hardware system into RTL/TLM code segments
  - The arbitrator checks each RTL/TLM component (or code segment) with its corresponding theoretical proof.

How to Detect Hardware Trojans (2)

Modify techniques for mitigating hardware trojans at the circuit- (i.e., cell-/gate- or logic level) or physical- (i.e., layout) levels

- Make use of software redundancy to use a hardware component (e.g., ALU/adder) in redundant ways and compare their results.
    - If the component doesn't behave as expected, a trojan exists; can't prove information leaks.
    - Composability problem: Hardware trojan may only be activated when the components are integrated to form the composed system.
    - Given code and proof, we can check if there exists vulnerabilities. But, we can't attest to anything outside the proof, such as the absence of data modification, system/hardware reset, and/or information leakage.

How to Detect Hardware Trojans (3)

- Compare different implementations of a component for the *PULPino* system (e.g., ALU/adder for the RISC-V ISA)
    - The same input is given to these different implementations.
    - The output of these different implementations is given to a majority gate (i.e., use majority voting).
    - Components that yield a different output from the majority of the components have hardware trojans.
    - That is, use techniques from fault tolerance for detecting hardware trojans.
    - While we cannot use TLM/RTL models from others, we can create our own TLM/RTL models (using *SystemC*, *Verilog*, and/or *Chisel*) for components of the *PULPino* system.

How to Detect Hardware Trojans (4)

- Use the aforementioned software redundancy technique to determine which hardware component went bad, and when and how (i.e.., sequence of input patterns) did it go bad.
- Additional comments on hardware security
    - In functional verification, we know what to check for.
    - In security verification, we don't know what to check for.
    - See Church thesis, Rice theorem, . . .

Table of Contents

References (1)

- [Wikipedia contributors 2016] Wikipedia contributors, "Alice and Bob," in *Wikipedia, The Free Encyclopedia: Cryptographic protocols*, Wikimedia Foundation, San Francisco, CA, February 28, 2018. Available online at: `https://en.wikipedia.org/wiki/Alice_and_Bob`; last accessed on October 26, 2016.
- [Khabarov 2015] Sergey Khabarov, answer to "What is meant by the FENCE instruction in the RISC-V instruction set?," Stack Exchange Inc., New York, NY, November 23, 2015. Available online from *Stack Exchange Inc.: Stack Overflow: Questions* at: `https://stackoverflow.com/questions/26374435/what-is-meant-by-the-fence-instruction-in-the-risc-v-i` March 16, 2016 was the last accessed date.

References (2)

- [Bushnell 2000] Michael L. Bushnell and Vishwani D. Agrawal, “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits,” in *Frontiers in Electronic Testing* series, Vol. 17, Springer Science+Business Media, Inc., New York, NY, 2000. DOI:10.1007/b117406.
- [Wang 2006] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen, “VLSI Test Principles And Architectures: Design for Testability,” in *The Morgan Kaufmann Series in Systems on Silicon*, Morgan Kaufmann, San Francisco, CA, 2006.

Table of Contents

Questions

- Is authentication required for read access to the boot ROM?