Hardware Trojans

in Incompletely Specified Digital Systems

Çetin Kaya Koç
University of California Santa Barbara
Acknowledgements

- This material is primarily the PhD work of Nicole Fern
- Furthermore, Dr. İsmail San was a collaborator of the project
- Research Institution: University of California Santa Barbara
- Funding Organization: NSF, Grant NSF/SRC STARSS (1526695)
- PIs: Tim Cheng and Çetin Kaya Koç

Koç is pronounced as “Coach”
Motivation

- Infrastructure depends on digital systems
- Hardware is the root of trust
- Hardware security topics:
  - IP Counterfeiting
  - Data Protection and Integrity
  - Cryptographic Hardware Design
  - Side-channel Information Leakage
  - Hardware Trojans
Hardware Trojans

- Malicious circuitry inserted in the hardware design
- It can be inserted by any party with access to the design
- Goals: leak information, induce faults, chip failure, gain root privileges, ..
Traditional verification question:

Does my design correctly perform the intended specified functionality?

A new verification question:

Does my design perform malicious functionality in addition to the intended functionality?
Trojan Classes – Class 1

- The logic functions of some design signals are altered, system specifications are violated.

- Example: Trojan flipping 1-bit during AES computation to perform fault attack.

- Requires triggering circuitry to stay hidden.

---

S. Bhasin et al. “Hardware Trojan Horses in Cryptographic IP Cores.” Workshop on Fault Diagnosis and Tolerance in Cryptography, 2013
Trojan Classes – Class 2

- The Trojan leaks information through side-channels
- It cannot be detected with functional testing, only side-channel characterization
- Example: Leakage of cryptographic key through power side-channel

Trojan Classes

- **Class 1:** The logic functions of some design signals are altered, system specifications are violated.

- **Class 2:** The Trojan leaks information through side-channels, but cannot be detected with functional testing, only side-channel characterization.

- **Class 3:** The logic functions of only those design signals which have *unspecified behavior* are altered to add malicious functionality without violating system specifications.
Class 3 Trojan Example: Simple FIFO

What is the value of read_data when read_enable is 0?
Existing Detection Methodologies

- **Class 1 Trojans:** Focus on identifying logic activated by triggering circuitry
  - Pre-silicon: Quantify “rareness” of logic in the RTL code or gate-level netlist using Boolean Functional Analysis [4,5] or simulation traces [6,7]
  - Post-silicon: Bias test vectors to favor low-controllability and low-observability nodes [8]

- **Class 2 Trojans:** Develop fingerprint for side-channel characteristics of Trojan-free design, then compare with suspect chips [9,10,11]
Verification Methodologies for Class 3 Trojans

- Don’t rely on rare triggering conditions to stay hidden (existing detection methods don’t apply)
- Only alter the logic functions of design signals which have unspecified behavior
- Never violate the design specification (hard to detect using traditional verification techniques)
Identification and Detection

- Identifying Dangerous Unspecified Functionality
  - RTL Don’t Care Classification
  - Mutation Testing Based Method
  - Unspecified On-Chip Bus Behavior

- Trojan Detection
  - Formulation as a SAT Problem
  - Gate-level Equivalence Checking
  - RTL Code Analysis + SMT Solver
RTL Don’t Cares

- Synthesis tool (and attacker) free to assign
- All key bits in Elliptic Curve Processor leaked by Trojan modifying don’t care bits
- Prevention methodology based on equivalence checking

```
module simple(clk, rst, control, data, key, out);
    input clk, rst;
    input [1:0] control;
    input [3:0] data, key;
    output reg [3:0] out;
    reg [3:0] tmp;
    always @(*)
        case(control)
            2'b00: tmp <= data;
            2'b01: tmp <= data ^ key;
            2'b10: tmp <= ~data;
            default: tmp <= 4'bxxxx;
        endcase
    always @(posedge clk)
        if (~reset) out <= 4'b0;
        else out <= tmp;
    endmodule
```

Computes $G = [k]P$, where $P$ is added to itself $k$ times

Elliptic curve discrete logarithm problem provides security

Don’t Cares in Control Unit

During each of the 38 states, control signals cwl[9:0] and cwh[22:0] are assigned.

Replacing don’t cares with 0s leads to an 8% area increase.
The ECP Trojan

- During State 15:
  - Address for Register Bank 2 is unknown
  - Write enable for Register Bank 2 is unknown
- Primary output sy corresponds to a register from Bank 2
- Trojan can replace sy with key value during State 15
  - Final encryption value still correct

---

Signals

- $tb_1.v.clo = 1$
- $tb_1.v.dono = 0$
- $tb_1.v.key[31:0] = DEADBEF$
- $tb_1.v.uut.k[31:0] = 80000000$
- $tb_1.v.nrst = 1$
- $tb_1.v.sx[232:0] = 14B64C9C3F920AE358A933002780E9CBF151F143ACBC9668A7CBE9F81CB$
- $tb_1.v.sy[232:0] = \ldots$
- $tb_1.v.uut.state[5:0] = 15$

Waves

- $DEADBEF$
- $CO000000$
- $1857BB9F1E + 19 + 10 + 0 + 018BF + 0 + 14 + 10 + 10 + 14B64C9C3F920AE358A933002780E9CBF151F143ACBC9668A7CBE9F81CB$
- $0A + 7 + 196F + 09 + 7 + 19 + 12 + 9FAC9 + 12 + 10 + 04 + 04D79 + 09 + 0FBE3 + 16914 + 1 + \ldots$
Automated Don’t Care Analysis

- Classify each don’t care as dangerous or safe
  - Replace dangerous don’t cares with static values
- Dangerous don’t cares cause a change in observable signals based on if they are 0 or 1
- **Equivalence Checking Formulation**: Compare 2 versions of the design each with different assignments of the don’t care bits
Use **combinational** equivalence checking with dead-code analysis for scalability
Equivalence Checking Formulation

- Make all don’t care bits primary inputs
- Create 2 copies of the circuit where $dc_i = 0$ and $dc_i = 1$
- Check if $C_0$ and $C_1$ are equivalent
Overhead

- Dangerous don’t cares must be assigned known values by designer
  - Increase in area/timing/power overhead
- Without reachability analysis, technique overestimates the number of dangerous don’t cares
- Trade-off between don’t care analysis effort and overhead due to prevention technique
Elliptic curve processor with 538 don’t care bits, 572 PIs, 467 POs, and 11,232 state elements

<table>
<thead>
<tr>
<th>Don’t Cares Replaced w/ Static Values</th>
<th>% Area Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dangerous after equivalence checking + dead-code analysis</td>
<td>0.04</td>
</tr>
<tr>
<td>Dangerous after comb. equivalence checking</td>
<td>1.80</td>
</tr>
<tr>
<td>All Don’t Care Bits (538 total)</td>
<td>8.00</td>
</tr>
</tbody>
</table>
Identifying Dangerous Unspecified Functionality

- Use **mutation** testing to uniformly sample possible design modifications (ie: very simple Trojan modifications)
  - Design independent and applicable to various abstraction levels
- Use information about how mutation affects **attacker-observable signals** to determine if modification is “dangerous”

---

Mutation Testing

Basic Idea: If the testbench cannot detect an artificial error, testbench likely incapable of detecting a real error [12]
Susceptibility to FIFO Trojan

Trojan leaks secret_data when read_enable = 0

Modify design, run tests
is it detected?

Modification is related to
functionality useful to attacker if:
1) Modification is undetected
2) Causes changes in attacker-observable signals

Listing 1: FIFO Read Behavior

1  //Memory Access Behavior
2  if (read_enable && !buffer_empty)
3    read_data <= mem[read_ptr];
4  //Pointer Update Behavior
5  if (read_enable && !buffer_empty)
6    read_ptr <= read_ptr + 1;
Fault Ranking Mechanism

- Might be too expensive to analyze all faults classified as dangerous
- Ideal to identify and fix functionality related to the “most dangerous” faults first
- Quantities easy to measure for each fault:
  - Number of attacker-observable bits differing
  - Total time attacker-observable signals differ
  - Number of distinct tests producing differences in attacker-observable signals
Hardware Trojans

NATO Workshop on Secure Implementation of PQC

UART Controller Case Study

- OpenCores IP [14], OVM testbench from EDA vendor
- Verification Infrastructure: 80 tests, 846 cover points
- Mutation Testing: 1183 total faults injected
  - 110 faults not detected
  - 32 caused differences in attacker-observable signals
  - 4 discounted cover points

Output signals going to main processor

```
wb_dat_o
wb_ack_o
int_o
```

Output signals for serial data transmission

```
baud_o
stx_pad_o
rts_pad_o
dtr_pad_o
```

38 attacker-observable bits

Cetin Kaya Koc

http://koclab.cs.ucsb.edu

September 27, 2016 26 / 41
UART Controller Case Study

- Analyzed 3 most dangerous faults
  - All affect bus between UART and main processor
  - All affect output enable signal for data bus
- Information can be leaked on data bus if a valid read transaction is NOT occurring

Wishbone bus: http://opencores.org/opencores,wishbone
3 undetected faults:

```vhdl
assign oe = ~wb_we_is || wb_stb_i || wb_cyc_i;

if (oe) then
  case (wb_sel_is) of
    4'b0001: wb_dat_o <= {24'b0, wb_dat8_o};
    4'b0010: wb_dat_o <= {16'b0, wb_dat8_o, 8'b0};
    ...;
endcase
```

- Write enable is de-asserted (read transaction)
- Slave is selected
- Valid bus cycle in progress
Wishbone Bus Trojan

Data can be leaked on `wb_dat_o` whenever all 4 conditions for a valid read transaction are not simultaneously met.

```
assign oe = ~wb_we_is & wb_stb_i & wb_cyc_i |
            wbstate==2'b01;

if (oe) begin
  if (wb_we_i | ~wb_stb_i | ~wb_cyc_i)
    wb_dat_o <= 32'hdeadbeef;
else
  case (wb_sel_is)
    4'b0001: wb_dat_o <= {24'b0, wb_dat8_o};
    4'b0010: wb_dat_o <= {16'b0, wb_dat8_o, 8'b0};
    ...
```

We will leak data when `wbstate==01` but not during a valid read transaction.

```
+-----------------+-----------------+-----------------+
| Time            | wb_dat_o[31:0]  | wb_dat8_o[7:0] |
+-----------------+-----------------+-----------------+
| 0x000000        | 0x00000000      | 0x00000000      |
+-----------------+-----------------+-----------------+
| 0x00000001      | 0x00000001      | 0x00000001      |
+-----------------+-----------------+-----------------+
| 0x00000002      | 0x00000002      | 0x00000002      |
+-----------------+-----------------+-----------------+
| 0x00000003      | 0x00000003      | 0x00000003      |
+-----------------+-----------------+-----------------+
| 0x00000004      | 0x00000004      | 0x00000004      |
+-----------------+-----------------+-----------------+
| 0x00000005      | 0x00000005      | 0x00000005      |
+-----------------+-----------------+-----------------+
| 0x00000006      | 0x00000006      | 0x00000006      |
+-----------------+-----------------+-----------------+
| 0x00000007      | 0x00000007      | 0x00000007      |
```
Data during invalid bus cycle, or output data during write cycles was not checked

New assertion added: value of wb_dat_o cannot change unless design has been reset or read request is being acknowledged

New check detects 3 faults and bus Trojan

The method is able to highlight unspecified functionality in on-chip bus protocols allowing attacker to leak information using the system bus
Trojan Channel Goal

- Covertly connect components spread across SoC
- Form new information channels by maximizing utilization of existing bus infrastructure
- Channel easily detected if new interface signals are created

---

Common bus protocols only partially specify signal behavior

AXI Channels: Read Data, Write Data, Read Address, Write Address, Write Response

AMBA AXI Channel Handshake Spec [15]

Can build covert Trojan communication channel using only existing bus signals when they are unspecified
Connectivity may not exist between the components containing Trojans.

Security mechanisms (for example, ARM TrustZone) monitor valid bus transactions to implement access control policies.
Trojan Channel Components

- **Factors Influencing Channel Properties: Topology & Protocol**

- **Data:** Existing bus signal(s) chosen to transmit Trojan data
  - Typically the address or data signals in the original protocol
- **Control:** Signal(s) chosen to mark Trojan transactions (distinguish from an idle bus)
- **Leakage Conditions Logic:** Monitors bus interface to determine when Trojan data can be sent/received
Hardware Trojans

AXI Example Channel

- **Sender**: Trojan snoops $k$ bits of S0’s read data signal
- **Receiver**: $k$ bits leaked to S1’s write data signal
- $\text{leak}_r = \text{WVALID’} \& \text{fifo_empty’}$
- **Receiver Control**: Write strobe signal is 1001
Example: AXI4-Lite System

- S0 and S1 are adder coprocessors that receive operands from an AXI4-Lite bus
- 2-way Trojan channel inserted to allow S0 to view S1’s operands and vice versa
- 50+ AXI4-Lite assertions [17] active during simulation, none are violated despite data flowing through Trojan channel
To guarantee that no Trojan channel exists:

- Fully specify the behavior of every bus signal
- Modify the bus implementation to comply with the fully refined specification
- Formally prove the bus implementation conforms to the specified behavior
Identification of dangerous unspecified functionality is far from complete
References


References

17. ARM: AMBA 4 AXI4, AXI4-Lite and AXI4-Stream Protocol Assertions BP063, Release Note (r0p1-00rel0).