Detecting Hardware Trojans in Unspecified Functionality Through Solving Satisfiability Problems

Nicole Fern[1,2] Ismail San[1,3] Kwang-Ting (Tim) Cheng[1,2]

[1]University of California Santa Barbara, USA

[2]Hong Kong University of Science and Technology, Hong Kong

[3]Anadolu University, Turkey

January 19, 2017

Outline

Hardware Trojans

Definition

Hardware Trojan: Malicious circuitry inserted in the hardware design during any stage in the design lifecycle

Who can insert Trojans?

- Rouge RTL designer, disgruntled employee
- 3rd party IP Provider
- Synthesis, layout, other EDA tools
- Fabrication facility
- Chip packaging and product integration facility

Security Risks of Unspecified Functionality

What can Trojans modify?

- **Critical design functionality** (ex. cause chip failure, induce faults, gain root privileges, remove memory protections, etc.) [12]
- **Non-digital circuit characteristics** (ex. amplify side-channel leakage, cause advanced circuit aging, etc.) [12]

Focus of this work

- Trojans modifying only **unspecified functionality**
- Trojan affects signals in digital domain, but does not cause violation of specified behavior

Example 1: RTL Don't Cares

- Don't cares minimize circuit area/timing/power overhead during synthesis
- Attacker can assign don't cares *any value* without violating the design specification

```
module simple (...);
  input clk, reset;
  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
```

Nicole Fern, Shrikant Kulkarni, and Kwang-Ting Cheng. "Hardware Trojans Hidden in RTL Don't Cares - Automated Insertion and Prevention Methodologies". In: *ITC*. 2015.

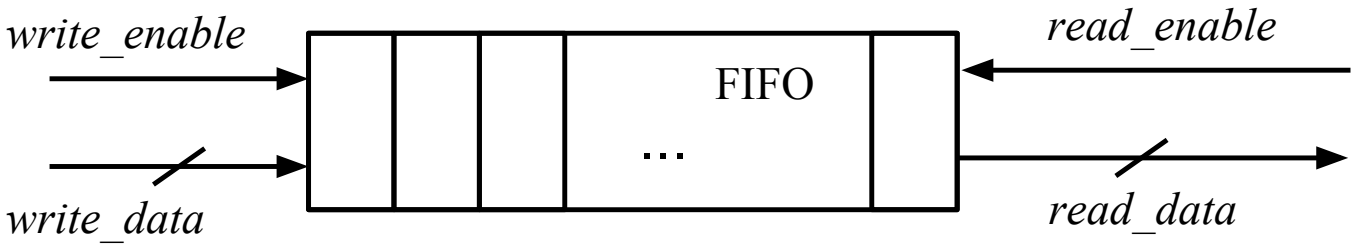
Example 1: RTL Don't Cares

- Don't cares minimize circuit area/timing/power overhead during synthesis
- Attacker can assign don't cares *any value* without violating the design specification

```
module simple (...);
  input clk, reset;
  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;
      2'b11: tmp <= key;
    endcase
  always @ (posedge clk)
    if (~reset) out <= 4'b0;
    else out <= tmp;
endmodule
```

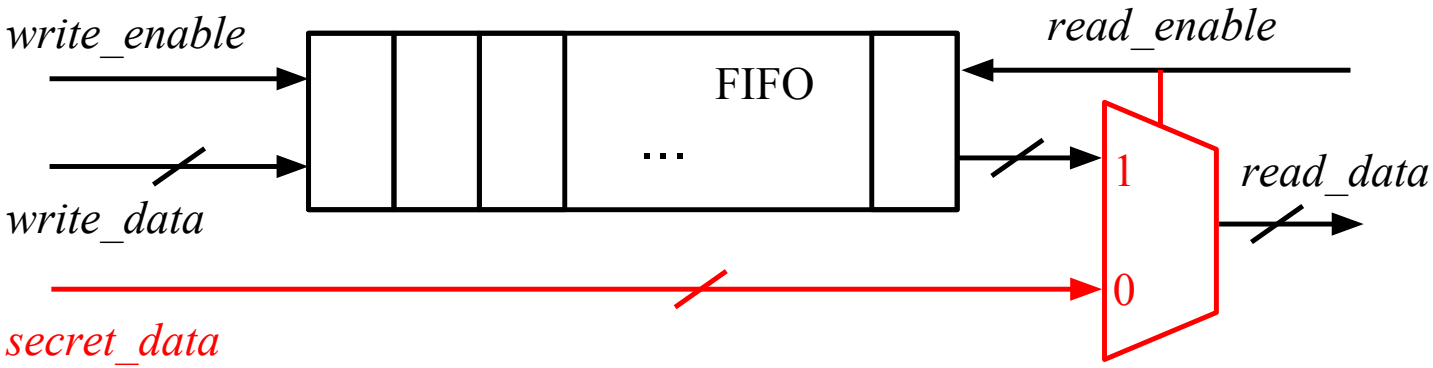
Nicole Fern, Shrikant Kulkarni, and Kwang-Ting Cheng. "Hardware Trojans Hidden in RTL Don't Cares - Automated Insertion and Prevention Methodologies". In: *ITC*. 2015.

Example 2: FIFO



- What is the correct value of *read_data* when *read_enable* is 0?

Example 2: FIFO



- What is the correct value of *read_data* when *read_enable* is 0?
- Does modification violate the specification? How can we detect it?

Why Is There Unspecified Functionality?

Answer: Design Complexity

- Fully specifying design behavior often impossible
- Only a subset of logic is involved in a particular task any given cycle
- Complete specification (if even possible) incurs significant implementation *and* verification overhead

Why Is There Unspecified Functionality?

Answer: Design Complexity

- Fully specifying design behavior often impossible
- Only a subset of logic is involved in a particular task any given cycle
- Complete specification (if even possible) incurs significant implementation *and* verification overhead

Additional Examples:

- Signals in floating point unit during a branch instruction
- Bus data lines during idle cycles
- Unused register fields and unmapped addresses
- Internet networking protocols

Verification and Trojan Detection Blind Spot

Verification Ignores Unspecified Functionality for Efficiency

- It is estimated that over 70% of hardware development resources are consumed by the verification task
- Verification focuses on increasing confidence in the correctness of *specified functionality*

Functional Trojan Detection Emphasizes Triggering Conditions

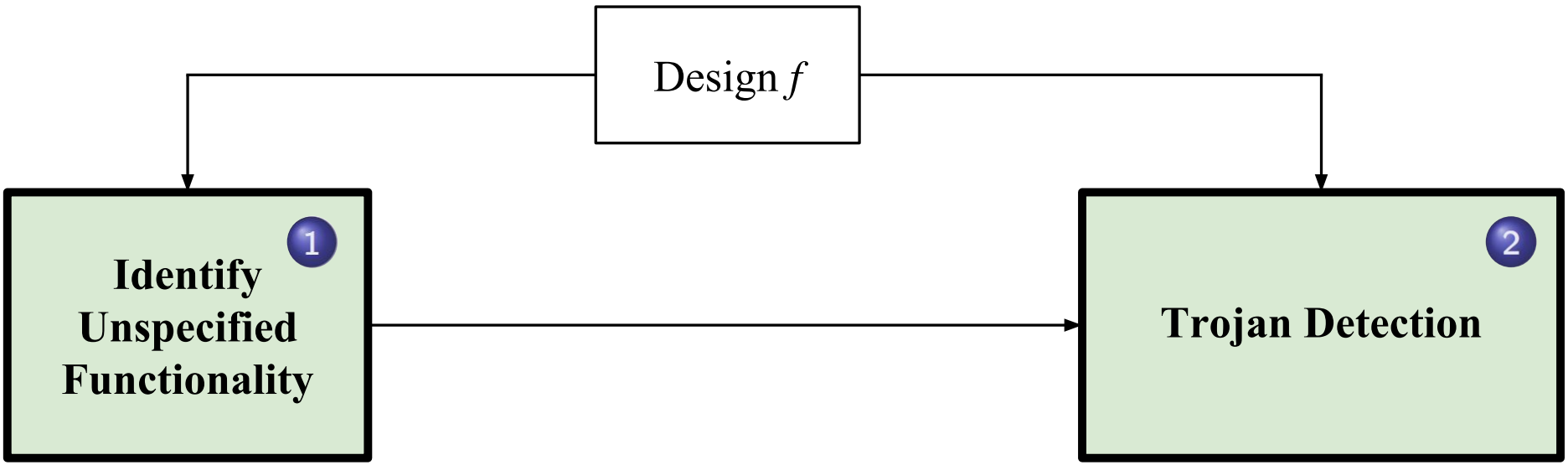
- Trojans only modifying unspecified functionality do not need triggering logic because no specifications are violated during activation
- Avoids detection by methods which identify triggering logic [11, 14, 8]

1. Introduction
   - Hardware Trojans
   - Unspecified Functionality

2. Securing Hardware Against Trojans in Unspecified Functionality
   - Overview
   - Formulating Trojan Detection as a Satisfiability Problem
   - Adder Coprocessor and UART Examples

3. Conclusions and Future Challenges

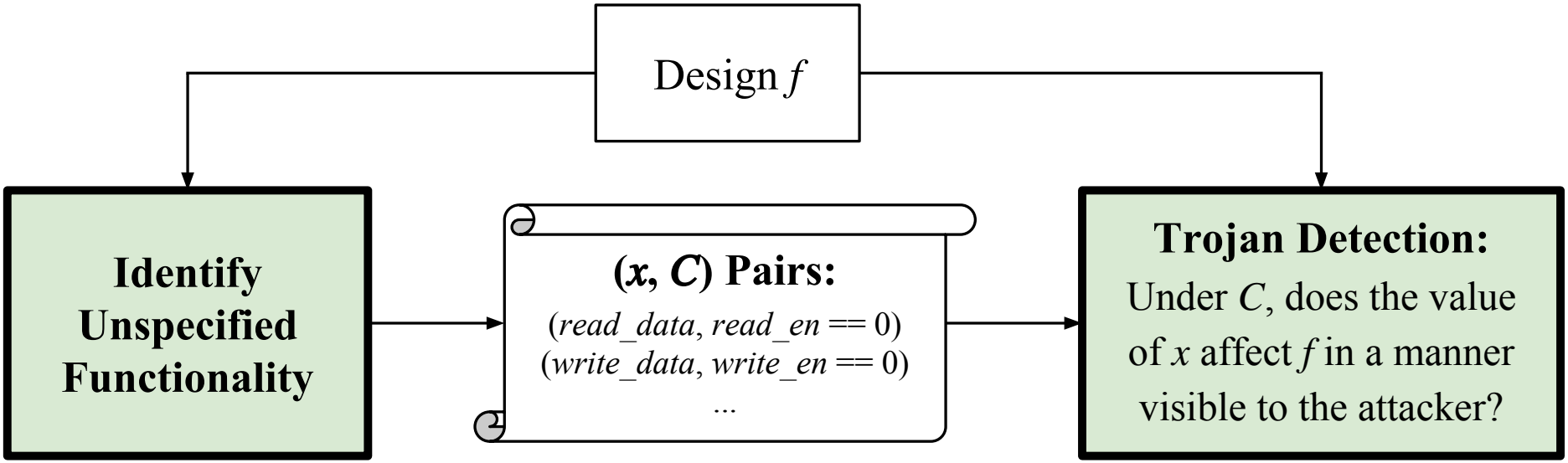
Two Important Steps for Trojan Detection



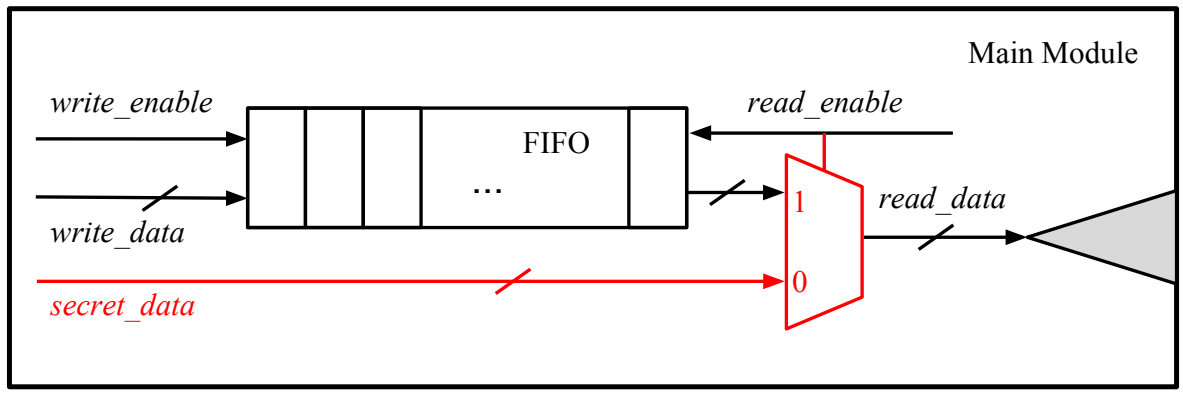
1. Can be done manually or using semi-automated method[1]
2. Guarantee absence of Trojans **without** specifying unspecified behavior

[1] Nicole Fern and Kwang-Ting Cheng. "Detecting Hardware Trojans in Unspecified Functionality Using Mutation Testing". In: *ICCAD*. 2015.

Detection Overview



- x: is a signal in f
- $\mathcal{C}$: is a condition under which x is unspecified



Formulating Trojan Detection as a Satisfiability Problem

- **Goal:** Identify if two different values of x during $\mathcal{C}$ can cause output or state elements in the design to differ [1]
- If Equation 1 is **satisfiable** x is likely involved in Trojan circuitry

$$\mathcal{C} \wedge (f_{x \to x_0} \oplus f_{x \to x_1}) \tag{1}$$

FIFO Example

- $x = read\_data$, $\mathcal{C} = \neg read\_enable$
- If $\neg read\_enable \wedge (f_{read\_data \to x_0} \oplus f_{read\_data \to x_1})$ satisfiable, FIFO data propagates to outputs when FIFO is not being read from!

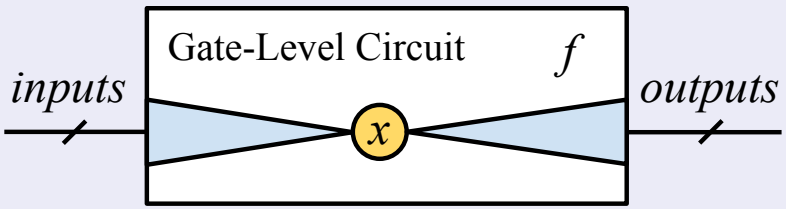
[1] f is a formula built from the design (can be boolean or contain operators such as $+$, $<$, etc.)

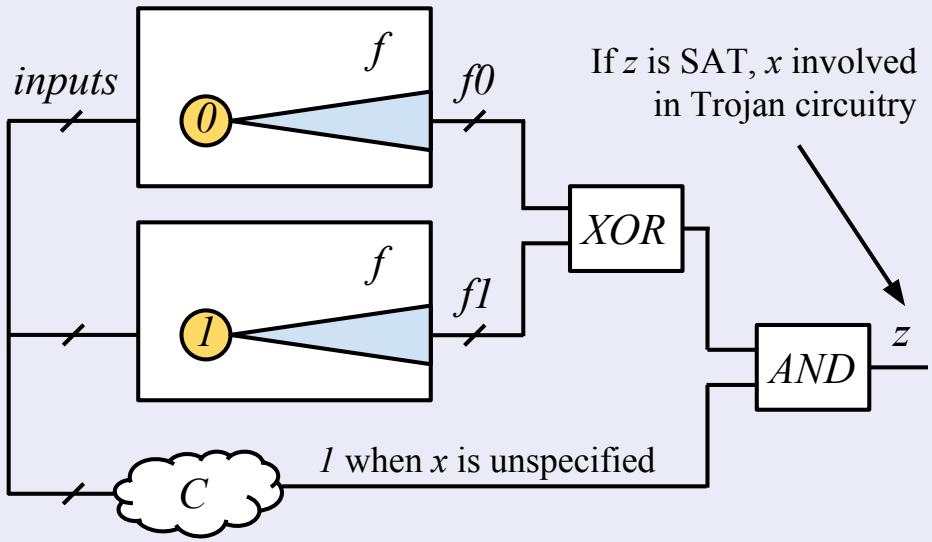
Determining Satisfiability when f is Boolean

- f obtained from gate-level netlist (produced from RTL design using synthesis tools)
- Use boolean SAT solver or logic equivalence checking tools (ex. Cadence Conformal LEC [2], Synopsys Formality [9], ABC [1],..)

Trojan Detection Using Equivalence Checking

Form miter for every $(x, \mathcal{C})$ pair:





Determining Satisfiability Using SMT Solvers

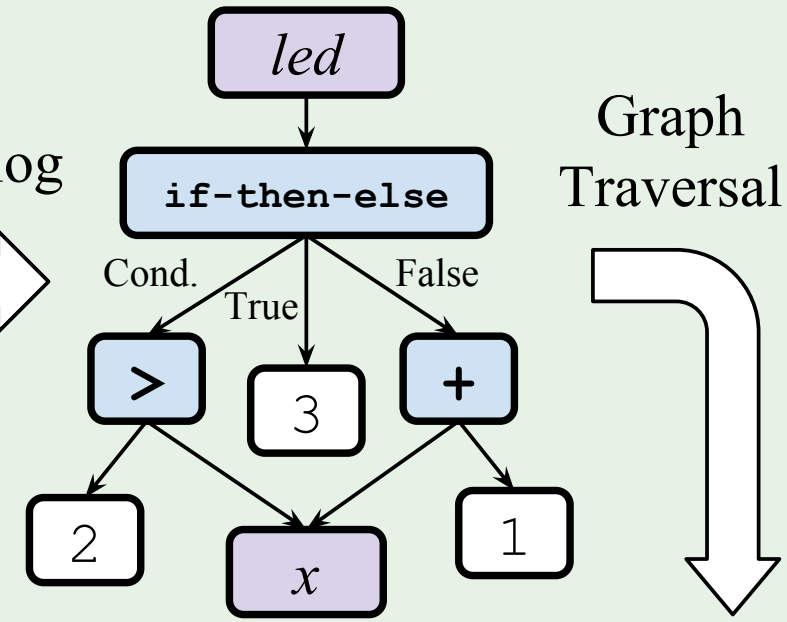
1. Build SMT formula for each attacker-observable signal o by constructing the signal data-flow graph using PyVerilog [10]

Example

Verilog Code

```
output [1:0] led;
reg [1:0] x;
always @(posedge clk)
    if (x > 2)
        led <= 2'd3;
    else
        led <= x + 1;
```

PyVerilog



Graph Traversal

SMT Formula: $\texttt{led} = \texttt{ite}(\texttt{gt}(x, 2), 3, \texttt{plus}(x, 1))$

Determining Satisfiability Using SMT Solvers

2. For each $(x, \mathcal{C})$ pair and o use PySMT [7] to determine satisfiability of $\mathcal{C} \wedge (o_{x \to x_0} \oplus o_{x \to x_1})$

Example

- Determine satisfiability of $\mathcal{C} \wedge (\texttt{led}_{x \to x_0} \oplus \texttt{led}_{x \to x_1})$
- Use PySMT formula built from traversing the data-flow graph:

$$\texttt{led} = \texttt{ite}(\texttt{gt}(x, 2), 3, \texttt{plus}(x, 1))$$

$$\texttt{SAT}(\texttt{and}(\mathcal{C}, \texttt{xor}(\underbrace{\texttt{ite}(\texttt{gt}(x_0, 2), 3, \texttt{plus}(x_0, 1))}_{\texttt{led, where } x \to x_0}, \underbrace{\texttt{ite}(\texttt{gt}(x_1, 2), 3, \texttt{plus}(x_1, 1))}_{\texttt{led, where } x \to x_1})))$$

Trojans in Unspecified On-Chip Bus Functionality

- Common bus protocols (ex. AMBA AXI, APB, Wishbone) only **partially specify** signal behavior

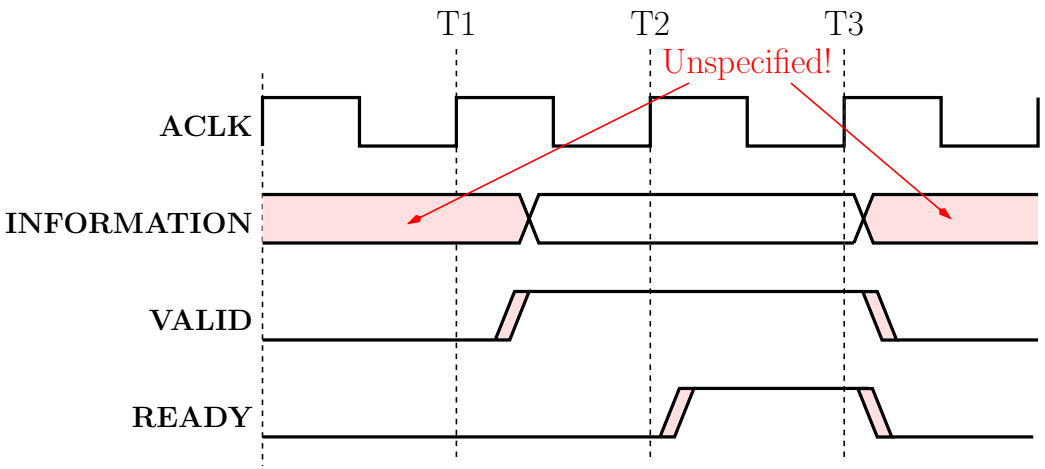


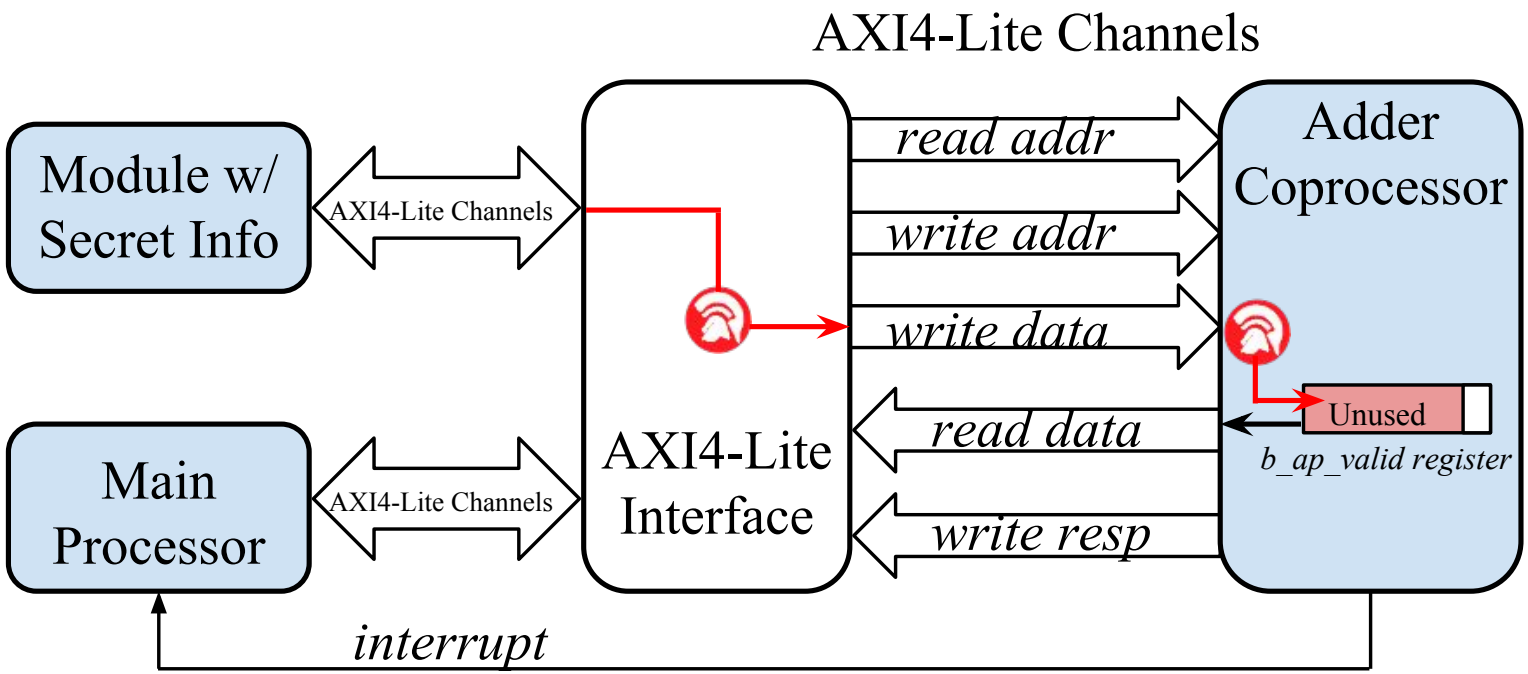
Figure: AXI Bus Protocol VALID/READY Handshake: Bus data can be anything (**including Trojan communications**) when VALID is LOW!

Nicole Fern et al. "Hiding Hardware Trojan Communication Channels in Partially Specified SoC Bus Functionality". In: *TCAD*. 2016.

Nicole Fern et al. "Hardware Trojans in Incompletely Specified On-chip Bus Systems". In: *DATE*. 2016.

Adder Coprocessor Trojan

- AXI4-Lite bus interface allows R/W to 8-bit registers
- **Trojan Operation:** 4-bits data leaked via on-chip bus to coprocessor's write data channel during idle bus cycles, then data stored in unused register field (read out later by attacker)



Detecting Adder Coprocessor Trojan

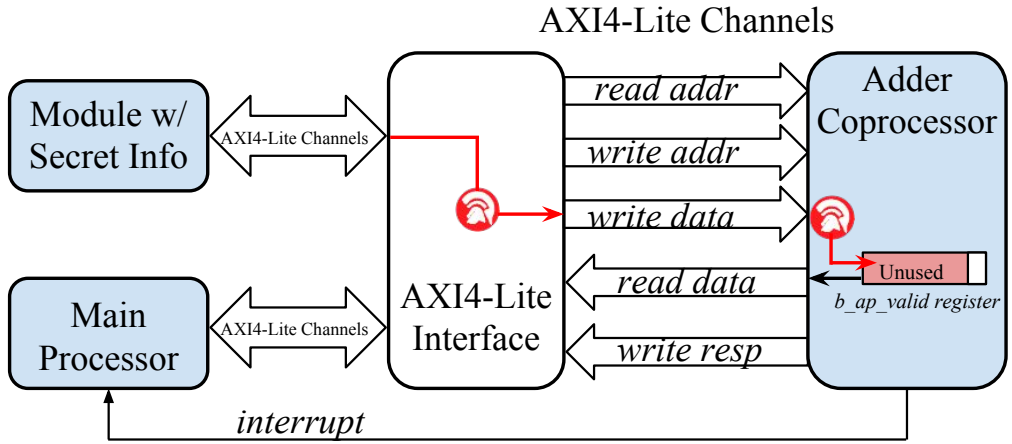
Objective

Determine if bus signals can influence adder coprocessor output under conditions where the bus is idle or control signals are unspecified.

(x, $\mathcal{C}$) Pairs: Input bus channel signals when channel *VALID* signal is *LOW*

x	$\mathcal{C}$
AWADDR	¬AWVALID
WDATA	¬WVALID
WSTRB	¬WVALID
ARADDR	¬ARVALID

Outputs: *AWREADY*, *WREADY*, *BRESP*, *BVALID*, *ARREADY*, *RDATA*, *RRESP*, *RVALID*, *interrupt*



Detecting Adder Coprocessor Trojan

1. Build data flow graph for all design outputs
2. For every $(x, \mathcal{C})$ pair and output o:
   - Determine satisfiability of $\mathcal{C} \wedge (o_{x \to x_0} \oplus o_{x \to x_1})$
   - If SAT, flag x as involved in Trojan circuitry and examine further

x	$\mathcal{C}$	**Outputs SAT**	
		Trojan-free	Trojan-infected
AWADDR	¬AWVALID	*None*	*None*
WDATA	¬WVALID	*None*	RDATA
WSTRB	¬WVALID	*None*	RDATA
ARADDR	¬ARVALID	*None*	*None*

- Technique highlights the bus signals involved in the Trojan circuitry
- No false positives when analyzing Trojan-free design

UART Example

- Wishbone [13] bus interface to registers
- **Trojan Operation:** Allows another slave to write to UART registers, when in original design only bus master can control UART

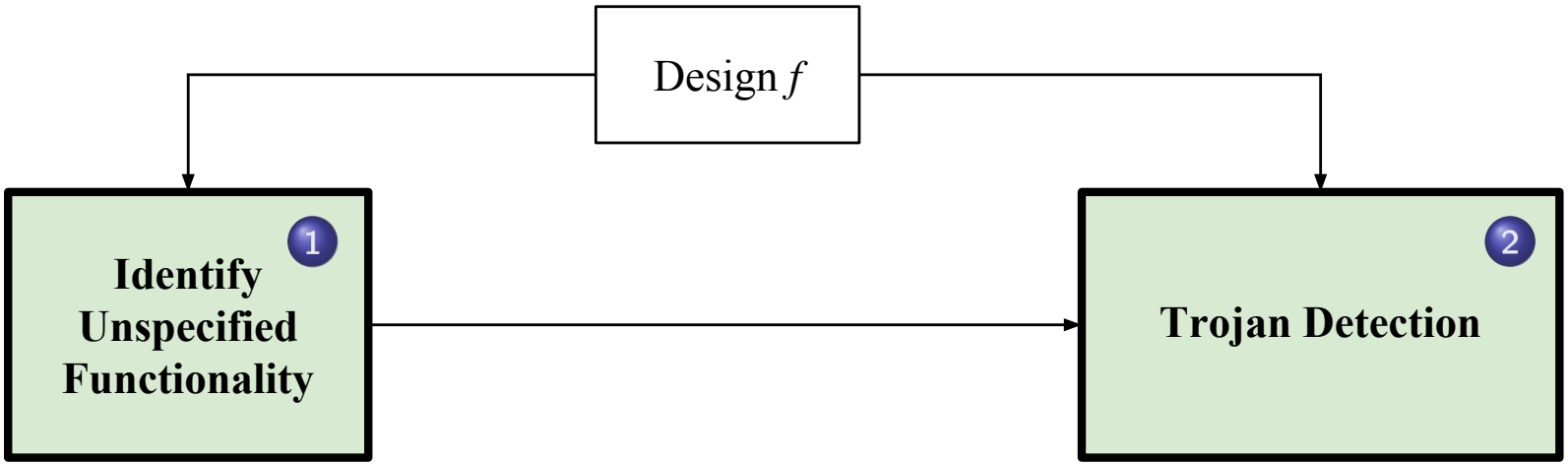
x	$\mathcal{C}$	**Outputs SAT**	
		Trojan-free	Trojan-infected
wb_adr_i	¬wb_stb_i ∨ ¬wb_cyc_i	*None*	int_o, baud_o, dtr_pad_o, stx_pad_o, rts_pad_o
wb_dat_i	¬wb_stb_i ∨ ¬wb_we_i ∨ ¬wb_cyc_i	*None*	int_o, baud_o, dtr_pad_o, stx_pad_o, rts_pad_o
wb_sel_i	¬wb_stb_i ∨ ¬wb_we_i ∨ ¬wb_cyc_i	*None*	int_o, baud_o, dtr_pad_o, wb_ack_o, stx_pad_o, rts_pad_o

1. Introduction
   - Hardware Trojans
   - Unspecified Functionality

2. Securing Hardware Against Trojans in Unspecified Functionality
   - Overview
   - Formulating Trojan Detection as a Satisfiability Problem
   - Adder Coprocessor and UART Examples

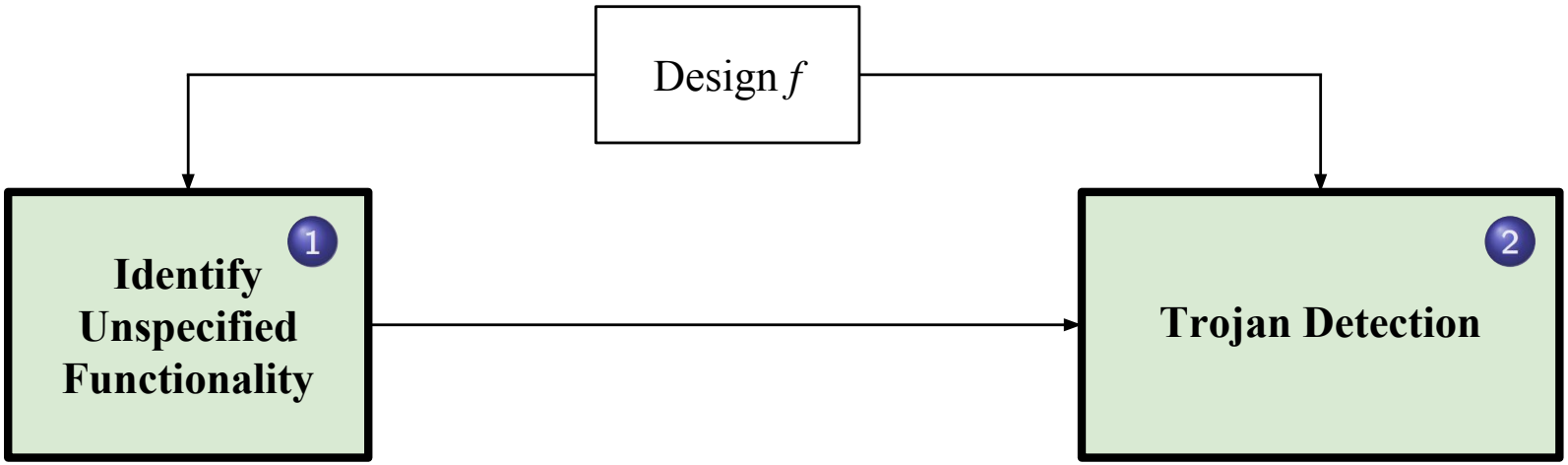
3. Conclusions and Future Challenges

Conclusions and Future Challenges



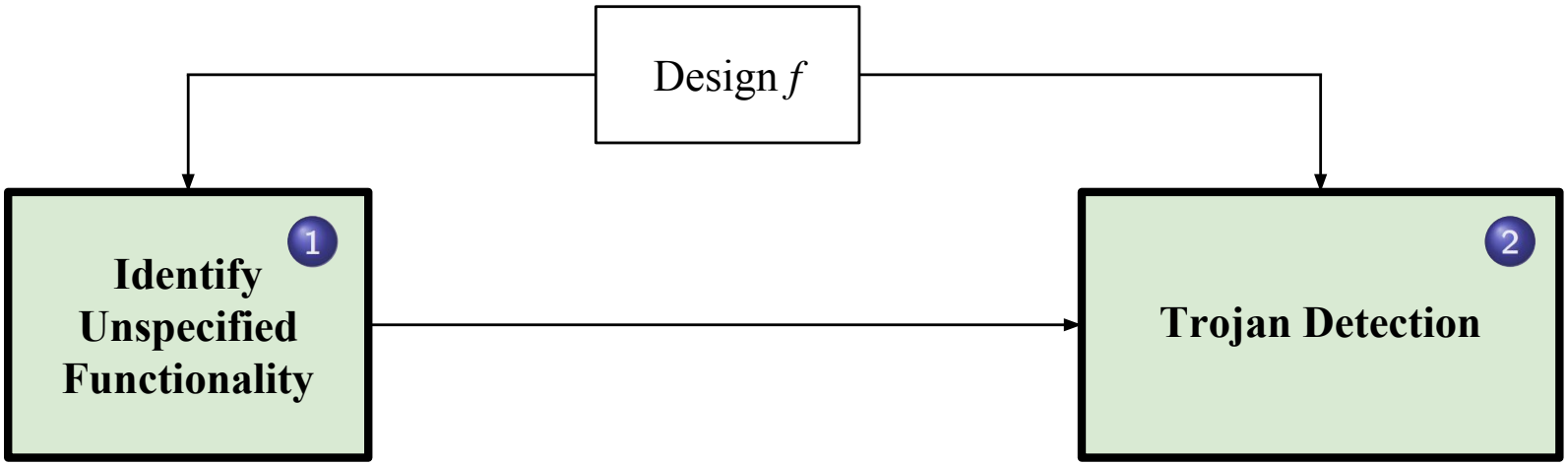
- An attacker can modify unspecified functionality to leak information without detection by existing verification techniques

Conclusions and Future Challenges



- An attacker can modify unspecified functionality to leak information without detection by existing verification techniques
- Detection methodology highlights Trojans in unspecified functionality *without* overhead of defining and implementing "benign" behavior

Conclusions and Future Challenges



- An attacker can modify unspecified functionality to leak information without detection by existing verification techniques
- Detection methodology highlights Trojans in unspecified functionality *without* overhead of defining and implementing "benign" behavior
- **Future Work:** Identifying $(x, \mathcal{C})$ pairs is still far from complete (always new threat models to discover)

Questions?

Email: nicole@ece.ucsb.edu/eenicole@ust.hk

Bibliography I

[1] *ABC*. URL: http://www.eecs.berkeley.edu/~alanmi/abc/.

[2] *Cadence Conformal Equivalence Checker*. URL: http://www.cadence.com/products/ld/equivalence_checker.

[3] Nicole Fern and Kwang-Ting Cheng. "Detecting Hardware Trojans in Unspecified Functionality Using Mutation Testing". In: *ICCAD*. 2015.

[4] Nicole Fern, Shrikant Kulkarni, and Kwang-Ting Cheng. "Hardware Trojans Hidden in RTL Don't Cares - Automated Insertion and Prevention Methodologies". In: *ITC*. 2015.

[5] Nicole Fern et al. "Hardware Trojans in Incompletely Specified On-chip Bus Systems". In: *DATE*. 2016.

[6] Nicole Fern et al. "Hiding Hardware Trojan Communication Channels in Partially Specified SoC Bus Functionality". In: *TCAD*. 2016.

[7] Marco Gario and Andrea Micheli. "PySMT: a Solver-agnostic Library for Fast Prototyping of SMT-Based Algorithms". In: 2015.

[8] Matthew Hicks et al. "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically". In: *Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP'10*. IEEE Computer Society, 2010, pp. 159–172.

[9] *Synopsys Formality*. URL: http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/Formality.aspx.

Bibliography II

[10] Shinya Takamaeda-Yamazaki. "Pyverilog: A Python-Based Hardware Design Processing Toolkit for Verilog HDL". In: *Applied Reconfigurable Computing*. 2015, pp. 451–460. DOI: 10.1007/978-3-319-16214-0_42.

[11] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. "FANCI: Identification of Stealthy Malicious Logic Using Boolean Functional Analysis". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS'13*. Berlin, Germany: ACM, 2013, pp. 697–708.

[12] Edgar Weippl et al. *Hardware Malware*. Morgan & Claypool Publishers, 2013.

[13] *Wishbone Bus*. URL: http://opencores.org/opencores,wishbone.

[14] Jie Zhang et al. "VeriTrust: Verification for Hardware Trust". In: *Proceedings of the 50th Annual Design Automation Conference, DAC'13*. Austin, Texas: ACM, 2013, 61:1–61:8.

Backup Slides

Scalability Issues

- SMT-based: design size limited by the robustness of Verilog parser
- Equivalence Checking: scalable and robust commercial tools exist

Design	**LOC**		**# 2NAND**		**Time (sec.)**	
	Orig.	Trj.	Orig.	Trj.	Orig.	Trj.
Adder	614	616	839	877	0.61	0.69
UART	2269	2273	5829	5836	8.59	8.63

Table: Design Size and Total Analysis Time For All $(x, \mathcal{C})$ Pairs

Modeling Sequential Behavior

- Both methods detected Trojan in Adder Coprocessor, however *combinational* equivalence checking failed to analyze UART design
- UART design latches the bus signals
- Pseudo-primary outputs trivially non-equivalent, but if only primary outputs analyzed, Trojan goes undetected
- Bounded sequential equivalence checking possible solution