Secure Information Flow Analysis for Hardware Design: Using the Right Abstraction for the Job

Xun Li
xun@cs.ucsb.edu

Mohit Tiwari Ben Hardekopf Timothy Sherwood Frederic T Chong

Department of Computer Science
University of California, Santa Barbara
Demand for Secure Hardware

- Cryptographic Hardware
- Hardware Controllers
- High-assurance Embedded Systems
  - Banks, aircrafts, etc.
Approaches to Security

- Verify by simulation
  - Long design cycle
  - Can not cover all execution paths
  - Sometimes we just need static guarantees!

- Program Analysis on HDLs
  - Conventional Analysis is imprecise
Our Goal

- A framework for designing hardware in a common paradigm and precisely verifying information flow properties at design time
Design Hardware as State Machines

Diagram:

- States: $S_0$, $S_1$, $S_2$
- Transitions between states
Design Hardware as State Machines

- case (cur_state)
  - S0: begin
    - …
    - cur_state = S1
    - end
  - S1: begin
    - …
    - cur_state = S2
    - end
  - S2: …
    - …
Conventional Analysis

Meet of all information flows into every concrete state

Meet of all information flows out from every concrete state

$S_0$

$S_1$

$S_2$

$\cdots$

$S_n$

$cur\_state$

Security Label
Example: Resource Controller
always @ * // The state machine will execute repeatedly
begin
  case (cur_state)
    0: // Master State (trusted)
      if(timer) begin
        next_state <= 1; // Jump to slave state
        next_timer <= timer; // Trusted Timer
      end
    1: // Slave 1 (untrusted):
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      else begin
        next_timer <= timer - 1; // Do something with untrusted data
        if (data) begin // Untrusted Data
          next_state <= 2
        end
      end
    2: // Slave 2 (untrusted)
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      next_timer <= timer - 1;
  endcase
end
always @ * // The state machine will execute repeatedly
begin
  case (cur_state)

  0: // Master State (trusted)
    if(timer) begin
      next_state <= 1; // Jump to slave state
      next_timer <= timer; // Trusted Timer
    end

  1: // Slave 1 (untrusted):
    if (timer == 0) begin
      next_state <= 0; // Jump back to master state
    end
    else begin
      next_timer <= timer - 1; // Do something with untrusted data
      if (data) begin // Untrusted Data
        next_state <= 2
      end
    end

  2: // Slave 2 (untrusted)
    if (timer == 0) begin
      next_state <= 0; // Jump back to master state
    end
    else begin
      next_timer <= timer - 1;
    end

  endcase
end
always @ * //The state machine will execute repeatedly
begin
  case (cur_state)
    0: //Master State (trusted)
      if(timer) begin
        next_state <= 1; //Jump to slave state
        next_timer <= timer; //Trusted Timer
      end
    1: //Slave 1 (untrusted):
      if (timer == 0) begin
        next_state <= 0; //Jump back to master state
      end
      else begin
        next_timer <= timer - 1;//Do something with untrusted data
        if (data) begin //Untrusted Data
          next_state <= 2
        end
      end
    2: //Slave 2 (untrusted)
      if (timer == 0) begin
        next_state <= 0; //Jump back to master state
      end
      next_timer <= timer - 1;
  endcase
end
always @ * // The state machine will execute repeatedly
begin
  case (cur_state)
    0: // Master State (trusted)
      if(timer) begin
        next_state <= 1; // Jump to slave state
        next_timer <= timer; // Trusted Timer
      end
    1: // Slave 1 (untrusted):
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      else begin
        next_timer <= timer - 1; // Do something with untrusted data
        if (data) begin // Untrusted Data
          next_state <= 2
        end
      end
    2: // Slave 2 (untrusted)
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      next_timer <= timer - 1;
  endcase
end
Example: Resource Controller

```
always @ * // The state machine will execute repeatedly
begin
  case (cur_state)
    0: // Master State (trusted)
      if(timer) begin
        next_state <= 1; // Jump to slave state
        next_timer <= timer; // Trusted Timer
      end
    1: // Slave 1 (untrusted):
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      else begin
        next_timer <= timer - 1; // Do something with untrusted data
      end
    2: // Slave 2 (untrusted)
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      else begin
        next_timer <= timer - 1;
      end
  endcase
end
```
Example: Resource Controller

```verilog
always @ (*) // The state machine will execute repeatedly
begin
    case (cur_state)
        0: // Master State (trusted)
            if (timer) begin
                next_state <= 1; // Jump to slave state
                next_timer <= timer; // Trusted Timer
            end
        1: // Slave 1 (untrusted):
            if (timer == 0) begin
                next_state <= 0; // Jump back to master state
            end
            else begin
                next_timer <= timer - 1; // Do something with untrusted data
                if (data) begin // Untrusted Data
                    next_state <= 2
                end
            end
        2: // Slave 2 (untrusted)
            if (timer == 0) begin
                next_state <= 0; // Jump back to master state
            end
            next_timer <= timer - 1;
    endcase
end
```
Example: Resource Controller

```verbatim
always @ * // The state machine will execute repeatedly
begin
    case (cur_state)
        0: // Master State (trusted)
            if(timer) begin
                next_state <= 1; // Jump to slave state
                next_timer <= timer; // Trusted Timer
            end
        1: // Slave 1 (untrusted):
            if (timer == 0) begin
                next_state <= 0; // Jump back to master state
            end
            else begin
                next_timer <= timer - 1; // Do something with untrusted data
            end
        2: // Slave 2 (untrusted)
            if (timer == 0) begin
                next_state <= 0; // Jump back to master state
            end
            next_timer <= timer - 1;
    endcase
end
```
always @ * // The state machine will execute repeatedly
begin
  case (cur_state)
    0: // Master State (trusted)
      if (timer) begin
        next_state <= 1; // Jump to slave state
        next_timer <= timer; // Trusted Timer
      end
    1: // Slave 1 (untrusted):
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      else begin
        next_timer <= timer - 1; // Do something with untrusted data
        if (data) begin // Untrusted Data
          next_state <= 2
        end
      end
    2: // Slave 2 (untrusted)
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      next_timer <= timer - 1;
  endcase
end
always @ * // The state machine will execute repeatedly
begin
    case (cur_state)
        0: // Master State (trusted)
            if(timer) begin
                next_state <= 1; // Jump to slave state
                next_timer <= timer; // Trusted Timer
            end
        1: // Slave 1 (untrusted):
            if (timer == 0) begin
                next_state <= 0; // Jump back to master state
            end
            else begin
                next_timer <= timer - 1; // Do something with untrusted data
                if (data) begin // Untrusted Data
                    next_state <= 2
                end
            end
        2: // Slave 2 (untrusted)
            if (timer == 0) begin
                next_state <= 0; // Jump back to master state
            end
            next_timer <= timer - 1;
    endcase
end
Conventional Analysis

Meet of all information flows into every concrete state

Meet of all information flows out from every concrete state

$S_0$

$S_1$

$S_2$

$\ldots$

$S_n$

$\textit{cur\_state}$

Security Label
State Machine Aware Analysis

Meet of all information flows into every concrete state

Meet of all information flows out from every concrete state

$S_0$

$S_1$

$S_2$

$S_n$

cur_state

Security Label
State Machine Aware Analysis

Meet of all information flows into every concrete state

Meet of all information flows out from every concrete state

cur_state
State Machine Aware Analysis

Security Label $S_0$

Security Label $S_1$

Security Label $S_2$

Security Label $S_n$
State Machine Aware Analysis

Information flows through every individual state

- Security Label $S_0$
- Security Label $S_1$
- Security Label $S_2$
- Security Label $S_n$
Does it work?

Timer expires Slave S2

Slave S1

Set timer

Master So

Slave will taint Master!!
Shared Behavior Among Slaves

always @ * // The state machine will execute repeatedly
begin
  case (cur_state)
    0: // Master State (trusted)
      if(timer) begin
        next_state <= 1; // Jump to slave state
        next_timer <= timer; // Trusted Timer
      end
    1: // Slave 1 (untrusted):
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      else begin
        next_timer <= timer - 1;
        // Do something with untrusted data
        if (data) begin // Untrusted Data
          next_state <= 2
        end
      end
    2: // Slave 2 (untrusted)
      if (timer == 0) begin
        next_state <= 0; // Jump back to master state
      end
      next_timer <= timer - 1;
  endcase
end
Instead of…
Hierarchical States

- Master So
- Slave S1
- Untrusted cloud
- Slave S2
- GROUP

- Set timer
- Timer expires
State Machine Aware Analysis

Timer expires

Set timer

Untrusted cloud

Master So

Slave S1

Slave S2
State Machine Aware Analysis

Untrusted cloud

Timer expires

Set timer

Master S0

Slave SI

Slave S2
State Machine Aware Analysis

Timer expires

Set timer

Untrusted cloud

Master So

Slave S1

Slave S2
State Machine Aware Analysis

Master

Slave S1

Slave S2

Timer expires

Set timer

Untrusted cloud

Yeah!
Existing Tool Chain

Conservative Information Flow Analysis

VHDL/Verilog

Synthesis Tool

Physical Hardware
Proposed Tool Chain

State Machine Description

Conservative Information Flow Analysis

VHDL/Verilog

Synthesis Tool

Physical Hardware
Proposed Tool Chain

State Machine Description

Precise Information Flow Analysis

VHDL/Verilog

Synthesis Tool

Physical Hardware
Proposed Tool Chain

- State Machine Description
  - Auto-generate
  - VHDL/Verilog
  - Synthesis Tool
  - Physical Hardware

Precise Information Flow Analysis
Proposed Tool Chain

- **State Machine Description**
- **Precise Information Flow Analysis**
- **Auto-generate**
- **VHDL/Verilog**
- **Synthesis Tool**
- **Physical Hardware**

- **Our Contribution**
Future Work

- A new state machine based hardware design language
- State machine based software protocol design
Conclusion

- By performing program analysis on state machine aware HDLs
  - Good programmability
  - Early feedback at design time
  - More precise information flow analysis

THANKS!