Automatized High-Level Evaluation of Security Properties for RTL Hardware Designs

Andrea Höller¹, Armin Krieg², Christopher Preschern¹, Christian Steger¹, Christian Kreiner¹, Holger Bock² and Josef Haid²

¹ Institute for Technical Informatics, Graz University of Technology, Austria
² Infineon Technologies Austria AG, Design Center Graz, Austria

8th Workshop on Embedded Systems Security (WESS 2013)
Montreal, September 29, 2013
Outline

- Introduction
- System Analysis and FSM Extraction Methodology
- Model-Based Fault Injection
- Experimental Results
- Conclusions and Future Work
Automated High-Level Evaluation of Security Properties for RTL Hardware Designs

Outline

- Introduction
- System Analysis and FSM Extraction Methodology
- Model-Based Fault Injection
- Experimental Results
- Conclusions and Future Work
The Formal Verification Gap
[Beckert2010]

Common Criteria
- Standard for computer security
- EALs: measure of quality
- Formal methods to verify security properties

Implementation of security properties neglected
→ Translation of the RTL implementation for model checkers
Model Checking

- Model in own language
- Automated extracting of FSMs from RTL
Formal Verification and Common Criteria

Related Work

Assurance Class Development

Model Checking the Policy Model

[Beuster2011]

Proposed Extension

[Beckert2010, Beuster2011]

Andrea Höller (andrea.hoeller@tugraz.at)
Automized High-Level Evaluation of Security Properties for RTL Hardware Designs
Outline

- Introduction / Related Work
- System Analysis and FSM Extraction Methodology
- Model-Based Fault Injection
- Experimental Results
- Conclusions and Future Work
Global Model-Checking Based Flow

- Java-based framework
- NuSMV model checker

Andrea Höller (andrea.hoeller@tugraz.at)
Automized High-Level Evaluation of Security Properties for RTL Hardware Designs
Global Model-Checking Based Flow

- Synthesizeable
- Control dominated
Global Model-Checking Based Flow

- Synthesizeable
- Control dominated
VHDL Structural Analysis

1. VHDL structure parsing
   - Extract system-internal structures
   - Dependency lists

2. Translation into CFG Description [Lohse1994]
   - Nodes for branches
   - Condition
   - List of transitions
   - One parent
3. FSM Identification

- Realization

- Find state register
  1. Find all registers
     - Synchronous register: reacts on active clock edge
  2. Decide, which of them are state register
     - State signal depends on itself
     - Dependency list \(\rightarrow\) search for circular dependency
Global Model-Checking Based Flow

System Analysis and FSM Extraction Methodology

Andrea Höller (andrea.hoeller@tugraz.at)
Automized High-Level Evaluation of Security Properties for RTL Hardware Designs
FSM Extraction

- Next-state logic
  - Find next-state signal
  - Collect transitions and conditions

- Initial Values

- Input Signals
  - Signals in the dependency list of the state

- Output Signals
  - Signals that depend on the state
  - Collect transitions
Translation to the NuSMV language

(1) Variable declarations

<table>
<thead>
<tr>
<th>VHDL type</th>
<th>NuSMV type</th>
</tr>
</thead>
<tbody>
<tr>
<td>std_logic</td>
<td>boolean</td>
</tr>
<tr>
<td>std_logic_vector(&lt;range&gt;)</td>
<td>word[&lt;size&gt;]</td>
</tr>
<tr>
<td>type &lt;identifier&gt; is range &lt;range&gt;</td>
<td>&lt;min&gt;..&lt;max&gt;</td>
</tr>
<tr>
<td>type &lt;identifier&gt; is array</td>
<td>array &lt;min&gt;..&lt;max&gt;</td>
</tr>
<tr>
<td>&lt;index_constrained&gt;</td>
<td>of &lt;element subtype&gt;</td>
</tr>
<tr>
<td>type typename is (enum1, enum2, ...)</td>
<td>{enum1, enum2,...}</td>
</tr>
</tbody>
</table>
Translation to NuSMV

(2) Translation of FSMs into NuSMV

(3) FSM interconnection
Global Model-Checking Based Flow

1. Hardware Description
2. VHDL Structural Analysis
3. FSM Extraction
4. Model Checking
5. Result Analysis
6. Functional Requirements Attack Descriptions
Outline

- Introduction / Related Work
- System Analysis and FSM Extraction Methodology
- Model-Based Fault Injection
- Experimental Results
- Conclusions and Future Work
Model-Based Fault Injection

- **Common Criteria requirement**

  **FRU FLT.1.1:** The TOE Security Functionality (TSF) shall ensure the operation of [assignment: list of TOE capabilities] when the following failures occur: [assignment: list of type of failures].

- **Advantages**
  + Completeness
  + No manipulation of original design

```plaintext
MODULE main
VAR
  var : boolean;
  ---- random occurrence of failures
  random_failure : boolean;
  stuckat_failure : boolean;
  inverted_failure : boolean;
ASSIGN
  next(var): case
    random_failure : {TRUE, FALSE};
    stuckat_failure : var;
    inverted_failure : !var
  esac;
```
Outline

- Introduction
- System Analysis and FSM Extraction Methodology
- Model-Based Fault Injection
- Experimental Results
- Conclusions and Future Work
Experimental Results

Model-based Fault Injection

VHDL modulo counter

VHDL code

```
process (cnt)
begin
  if (cnt = 8) then
    next_cnt <= 0;
  else
    next_cnt <= cnt + 1;
  end if;
end process;
```

generated NuSMV model

```
MODULE FSM_CNTVAR
  cnt : word[4];
  set : boolean; <-
  fi : boolean; <-
  errorpos : 0..3; <-
ASSIGN
  init(cnt) := 0h_0;
  next(cnt) := case
    set : cnt xor (0b4_0001 << errorpos); <-
    (cnt = 0h_8) : 0h_0;
    !((cnt = 0h_8)) : cnt + 0h_1;
    TRUE: cnt;
  esac;
  next(set) := set ? FALSE : fi; <-
```
Model-based Fault Injection

LTL Specification

\[ G((cnt \geq 8) \rightarrow X(cnt < 8)) \]

→ Robust design support

Result

Trace Description: LTL Counterexample

→ State: 1.1
  fsm_cnt.cnt = 0ud4_0
  fsm_cnt.set = FALSE
  fsm_cnt.fi = FALSE
  fsm_cnt.errorpos = 0

→ State: 1.2
  fsm_cnt.cnt = 0ud4_1
  fsm_cnt.fi = TRUE

→ State: 1.3
  fsm_cnt.cnt = 0ud4_2
  fsm_cnt.set = TRUE
  fsm_cnt.fi = FALSE
  fsm_cnt.errorpos = 3

→ State: 1.4
  fsm_cnt.cnt = 0ud4_10
  fsm_cnt.set = FALSE
  fsm_cnt.fi = TRUE
  fsm_cnt.errorpos = 0

→ State: 1.5
  fsm_cnt.cnt = 0ud4_11
Automated High-Level Evaluation of Security Properties for RTL Hardware Designs

Outlines

- Introduction
- System Analysis and FSM Extraction Methodology
- Model-Based Fault Injection
- Experimental Results
- Conclusions and Future Work
Conclusion

- Step towards filling the Common Criteria verification gap
- Automatic generation of high-level representation of RTL implementation
- Evaluation of fault-attack robustness

Future Work

- Model-based fault injection for the safety domain
Thank you very much for your attention!

Any questions?
Sources


FSM Extraction

- Next-state logic
  - Find next-state signal
  - Collect transitions and conditions

One-segment code styling

Multi-segment code styling
Verification of Security Policies

- Verification of security requirements
  - Common Criteria property
    \[ FIA\ SOS.2.2: \textit{The TOE Security Function (TSF) shall be able to enforce the use of TSF generated secrets for [assignment: list of TSF functions]} \]

- LTL specification
  \[ G(\neg(tx = UART\_IDLE) \land \neg pwd\_given) \]

\rightarrow Verified property
Verification of Security Policies

UART control logic with password feature

```vhdl
tx_proc: process(clk)
begin
  if clk'event and clk = '1' then
    tx <= UART_IDLE;
  case tx_fsm is
    when IDLE =>
      if tx_init = '1' then
        if pwd_given = '1' then -- check pwd
          tx fsm <= DATA;
          -- initializations etc...
        when DATA =>
          -- parallel-to-serial conv. etc...
          tx <= tx_data(0);
          tx fsm <= STOP;
        when STOP =>
          -- Send stop bit etc...
          tx fsm = IDLE
      end if;
    end case;
  end if;
end process;
```

```vhdl
MODULE TX_FSM(pwd_given)
VAR
tx_fsm : {idle, data, parity, stop};
tx : boolean;
...
IVAR
tx_data : unsigned word[8];
ASSIGN
-- several initialisations...
next(tx_fsm):= case
  tx_fsm=idle & tx_init & pwd_given : data;
  tx_fsm=idle & !tx_init : idle;
  tx_fsm=stop : idle;
  ... esac;
next(tx) := case
  tx_fsm=data : tx_data[0];
  TRUE : UART_IDLE;
  ...
```

Experiment Results

Automated High-Level Evaluation of Security Properties for RTL Hardware Designs

Andrea Höller (andrea.hoeller@tugraz.at)
Initial Values

Reset Patterns

--synchronous reset
\[
\text{process}(\text{clk, reset}) \\
\quad \text{if rising\_edge(} \text{clk} \text{)} \text{ then} \\
\quad \quad \text{if (reset) then} \\
\quad \quad \quad \ldots \text{initial value} \\
\quad \quad \text{assign} \ldots \\
\quad \quad \text{else} \\
\quad \quad \quad \ldots \text{logic} \\
\quad \text{end if} \\
\text{end process;}
\]

--asynchronous reset
\[
\text{process}(\text{clk, reset}) \\
\quad \text{if (reset) then} \\
\quad \quad \ldots \text{initial value} \\
\quad \quad \text{assign} \ldots \\
\quad \quad \text{else if} \\
\quad \quad \text{rising\_edge(} \text{clk} \text{)} \\
\quad \quad \quad \ldots \text{logic} \\
\quad \text{end if} \\
\text{end process;}
\]