Specification-Driven Automated Conformance Checking for Virtual Prototype and Post-Silicon Designs

Haifeng Gu\textsuperscript{1}, Mingsong Chen\textsuperscript{1}, Tongquan Wei\textsuperscript{1}, Li Lei\textsuperscript{2}, and Fei Xie\textsuperscript{3}

\textsuperscript{1}Shanghai Key Lab of Trustworthy Computing, East China Normal University, China
\textsuperscript{2}Intel Labs, United States
\textsuperscript{3}Department of Computer Science, Portland State University, United States
Outline

- **Motivation**

- **Formal Device Model Generation**
  - Syntax Extensions for SystemRDL
  - Automated Generation of Formal Device Models

- **Specification-Driven Conformance Checking**
  - Device Trace Collection
  - Conformance Checking with FDMs

- Performance Evaluation Results

- Conclusion
Virtual Prototypes Are Increasingly Used

- Virtualization prototyping can
  - Maximize device utilization, e.g., could computing
  - Improve production efficiency, e.g., ESL design
Challenges to be Addressed

- Various causes
  - Software bugs hidden on virtual prototypes, silicon hardware bugs
  - Observability is limited in silicon troubleshooting
Things Are Even Worse …

- Correctness of virtual/silicon devices cannot be guaranteed
  - Due to lack of golden reference models!
Specification-Driven Conformance Checking

- **Formal device model generation using SystemRDL**
  - **Pros:** Consistent register definitions across abstraction levels
  - **Cons:** Lack of register behavior modeling mechanisms

---

Formal Device Model

- Extended SystemRDL Specification
  - HW/SW Interface
  - HW Model

---

Conformance Checking

Virtual Prototype

[Extended SystemRDL Specification](#)

Silicon Prototype

Syntax Extensions for SystemRDL

- Define **macro** and **function** components for registers’ logics
  - Macro facilitates the programming of function components
  - Function supports **behavior modeling** of interface registers

```
component_def ::= new_comp_body | component_type
                component_name {{[component_def:] [property;][. . . ;]}} *

component_type ::= field | reg | regfile | addrmap | signal | enum | macro | function

property ::= property_name = value;

new_comp_body ::= comp_body | macro_body | function_body

macro_body ::= {macro_name = value;}*

function_body ::= fun_type fun_name ( argument_list ) { statements }
```
Our Framework

- Device Driver
- OS Kernel
- Kernel API Interception
- Hardware Device
- Trace Recorder
- Trace File
- Conformance Checker
- Extended SystemRDL Specification
- Formal Device Model Generator
- Formal Device Model
- Inconsistency Reports
Automated Generation of FDMs

Transformation Rules

- A set of 10 rules is proposed for the transformation to FDMs
Our Framework

- **Device Driver**
- **OS Kernel**
- **Kernel API Interception**
- **Hardware Device**
- **Trace Recorder**
- **Trace File**
- **Conformance Checker**
- **Extended SystemRDL Specification**
- **Formal Device Model Generator**
- **Formal Device Model**
- **Inconsistency Reports**
FDM State & Virtual/Silicon Device State

- **Formal Device Model state**
  - \( F = <F_i, F_n> \)
  - \( F_i \): interface register state
  - \( F_n \): internal register state

- **Virtual/Silicon device state**
  - \( S_i \): interface register state

- **Definition of Conformance**
  - An FDM state \( F \) and a virtual/silicon device state \( S_i \) conform to each other if \( S_i \models F \).
A device trace is a sequence of \( <S_i, A_i> \) pairs

- \( S_i \): current assignments to all the registers
- \( A_i \): forthcoming driver request
Our Framework

- Device Driver
- OS Kernel
- Kernel API Interception
- Hardware Device
- Trace Recorder
- Trace File
- Conformance Checker

- Extended SystemRDL Specification
- Formal Device Model Generator
- Formal Device Model
- Inconsistency Reports
Conformance Checking Procedure

i=0

Fetch \( <S_i, A_i> \) from Trace File

\( F_i \leftarrow S_i \)

Symbolic Execution FDM with \( F_i, A_i \)

Generate a Set \( G \) of FDM States

Check Conformance between \( G \) & \( S_{i+1} \)

Last Event?

Y

Fetch \( <S_i, A_i> \) from Trace File

\( F_i \leftarrow S_i \)

Generate a Set \( G \) of FDM States

Check Conformance between \( G \) & \( S_{i+1} \)

N

Buffer the Inconsistency

Y

Inconsistency Reports

N

i=i+1
Outline

- Motivation
- Formal Device Model Generation
  - Syntax Extensions for SystemRDL
  - Automated Generation of Formal Device Models
- Specification-Driven Conformance Checking
  - Device Trace Collection
  - Conformance Checking with FDMs
- Performance Evaluation Results
- Conclusion
All the experiments were obtained on an Ubuntu desktop with 3.2GHz AMD CPU and 16GB RAM
Experimental Settings

Virtual Devices:

- QEMU 0.15.1
- Contains e1000 and eeapro100 virtual devices

Silicon Devices:

- Intel E1000 Gigabit NIC (e1000)
- Intel EEPPro100 Megabit NIC (eeapro100)

Table 1: Experimental Settings for Network Adapters

<table>
<thead>
<tr>
<th>Devices</th>
<th>Spec. (LoC)</th>
<th>FDM (LoC)</th>
<th>VP (LoC)</th>
<th>Select. Captured Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel e1000 Gigabit NIC</td>
<td>546</td>
<td>1805</td>
<td>2099</td>
<td>1224</td>
</tr>
<tr>
<td>Intel eeapro100 Megabit NIC</td>
<td>587</td>
<td>903</td>
<td>2178</td>
<td>74</td>
</tr>
</tbody>
</table>
Experimental Results

- Validated designs using 4 types of network commands
  - e.g., `ifconfig`, `ping`, `scp`, `ifup`, `hping3`, ...
- Detected 12 bugs from virtual/silicon devices

<table>
<thead>
<tr>
<th>Indices</th>
<th>Bug Types</th>
<th>Num.</th>
<th>Bug Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>Update the bits of reserved SD register</td>
<td>3</td>
<td>SD</td>
</tr>
<tr>
<td>E2</td>
<td>Generate unnecessary interrupts</td>
<td>1</td>
<td>VP</td>
</tr>
<tr>
<td>E3</td>
<td>Fail to update register when necessary</td>
<td>2</td>
<td>VP</td>
</tr>
<tr>
<td>E4</td>
<td>Write incorrect values to registers</td>
<td>3</td>
<td>VP</td>
</tr>
<tr>
<td>E5</td>
<td>Update the bits of reserved VP register</td>
<td>1</td>
<td>VP</td>
</tr>
<tr>
<td>E6</td>
<td>Driver issues a write to reserved registers</td>
<td>2</td>
<td>Driver</td>
</tr>
</tbody>
</table>

* VP and SD stand for Virtual Prototype and Silicon Device, respectively

Driver issues invalid requests!
Experimental Results

- Better **false negative** ratio due to more bugs detected
  - E5: Update the bits of reserved VP register
  - E6: Driver issues a write to reserved registers

Table 3: Comparison of Different Methods

<table>
<thead>
<tr>
<th>Bug Source</th>
<th>Bug Type</th>
<th>FDM-VP</th>
<th>FDM-SD</th>
<th>VP-SD [7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicon Devices</td>
<td>E1</td>
<td>-</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>E2</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>E3</td>
<td>2</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>E4</td>
<td>3</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>Virtual Devices</td>
<td>E5</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Driver</td>
<td>E6</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Experimental Results

- Better conformance checking time: **Up to 67X improvement**
- Better resource utilization: **Up to 2X less memory used**

![Fig.1 Time Usage for e1000 NIC](chart1.png)

**Fig.1 Time Usage for e1000 NIC**

![Fig.2 Memory Usage for e1000 NIC](chart2.png)

**Fig.2 Memory Usage for e1000 NIC**
Conclusion

**Contributions:**
- SystemRDL extension for register access behavior modeling
- Transformation rules from extended SystemRDL to FDMs
- Symbolic execution-based conformance checking framework

**Experimental results on industrial network adapters**
- New bugs found in virtual/silicon devices
- Better performance than state-of-the-art methods

**Future work**
- Directed test generation for virtual/silicon prototypes
- Runtime validation
Thank you!