Caisson: A Hardware Description Language for Secure Information Flow

Xun Li
xun@cs.ucsb.edu

Mohit Tiwari    Jason K. Oberg    Vineeth Kashyap
Frederic T. Chong    Timothy Sherwood    Ben Hardekopf

UCSB    UCSD

Xun Li    PLDI 2011
Why security is important?

• FUN Story #1
  – Boeing 787: Passenger network is connected to the plane's control systems
Why security is important?

• FUN Story #2
  – Hospitals: The same computers are used to control patients’ dosage/life-maintainance as well as for doctors/nurses to check emails
Why security is important?

• Bank systems, military systems, etc,
How hard is being secure?

- INTEGRITY Real-Time Operating System
  - Common Criteria EAL6 (1-7)
  - $10K per line of code
  - Over 10 years to complete
Where to enforce security?

• Applications (GIFT-06, LIFT-06, etc.)
• Programming Languages (Jif-99, etc.)
• Operating Systems (HiStar-06, Flume-07, etc.)
• Architecture (DIFT-04, Raksha-07, etc.)
• Micro-architectures (PL/RPCache-07, etc.)
• Hardware Designs
Existing Approach

• Tracking Information from Gates Up
  – GLIFT-09, ExecutionLease-09
  – Dynamic track information at the gate level
  – Large tracking overhead
  – Cannot provide static guarantees
Our Goal

• A Hardware Description Language that
  – Familiar to Hardware Designers
  – Reason about security policies at design time
  – Promote thinking in security while designing
Outline

• Motivation

• Security Policy

• Caisson

• Case Study: A Secure Processor Design

• Future Work
What is our security policy?

- **Integrity**
  - Passenger Network (Untrusted)
  - Flight Control Network (trusted)

- **Secrecy**
  - Confidential Data (Secret)
  - Open Network (Unclassified)
What is our security policy?

- Passenger Network: Untrusted
- Confidential Data: Secret
- Flight Control Network: trusted
- Open Network: Unclassified
- Non-interference

Xun Li  PLDI 2011
Outline

• Motivation
• Security Policy
• Caisson
• Case Study: A Secure Processor Design
• Future Work
Caisson

• What is it?
  – An security extension to a subset of Verilog

• What are the cool features in it?
  – Type System Extension to Verilog
  – State Machine Aware Programming
    • Nested States
    • Parameterized States
Caisson

• What is it?
  – An security extension to a subset of Verilog

• What are the cool features in it?
  – Type System Extension to Verilog
  – State Machine Aware Programming
    • Nested States
    • Parameterized States
Type System Extension to Verilog

• Security Type for Signals

```
reg secret;  reg secret:H;  reg public;  reg public:L;
```

• Typing Rules to Enforce Non-interference

```
x := y;

x := 1;
else
x := 2;
```

- $x := y$:
  - $x:H$, $y:L$: ✔
  - $x:L$, $y:H$: ❌

- if (y)
  - $x := 1$:
    - $x:H$, $y:L$: ✔
    - $x:L$, $y:H$: ❌

Volpano’96;
Jif’99, FlowCaml’03;
Tolstrup’05

Xun Li  PLDI 2011
Caisson

• What is it?
  – An security extension to a subset of Verilog

• What are the cool features in it?
  – Type System Extension to Verilog
  – State Machine Aware Programming
    • Nested States
    • Parameterized States
State Machine Aware Programming

• Hardware Designers love state machines
  
  • case (cur_state)
    
    $S_0$: begin
    ...  
    cur_state = $S_1$
    end
    $S_1$: begin
    ...  
    cur_state = $S_2$
    end
    $S_2$: ...
    ...

Xun Li  PLDI 2011
State Machine Aware Programming

- Hardware Designers love state machines
  - case (cur_state)
    - \$S_0\$: begin
      - ...
      - \$cur\_state = S_1\$
      - end
    - \$S_1\$: begin
      - ...
      - \$cur\_state = S_2\$
      - end
    - \$S_2\$: ...
      - ...
      - ...
Conventional Implementation

Meet of all information flows into every concrete state

Meet of all information flows out from every concrete state

cur_state

Security Label
State Machine Aware

Meet of all information flows into every concrete state

cur_state

Meet of all information flows out from every concrete state

S0

S1

S2

... Sn
State Machine Aware

Meet of all information flows into every concrete state

cur_state

Meet of all information flows out from every concrete state
State Machine Aware

Security Label $S_0$

Security Label $S_1$

Security Label $S_2$

\[ \ldots \]

Security Label $S_n$
Information flows through every individual state
New Syntax for State Machines

- state S1:L = {
  public := public + 1;
  goto S2;
}
state S2:H = {
  secret := secret + 1;
  goto S1;
}

THE PROBLEM:
State transitions carry information!
*High* states will still taint *low* states.

THE QUESTION:
What kind of state machine would have some states being *high* and some states being *low* while still being secure?
Secure Hardware Design Pattern

• How to design secure hardware that deals with different levels of security?
  – Physical Separation/Replication
  – Controlled Sharing
    • Time Division Multiple Access (TDMA)
We want to express the fact that Slave states are not able to affect the Master EVEN THOUGH there are transitions from Slave to Master
Caisson

• What is it?
  – An security extension to a subset of Verilog

• What are the cool features in it?
  – Type System Extension to Verilog
  – State Machine Aware Programming
    • Nested States
    • Parameterized States
A Typical Secure Controller
Nested States
Nested States

prog lease = timer:L, data1: H, data2: L, mode:L
in
let state master:L = {
  if mode = 0
    then mode := 1 goto S1
    else mode := 0 goto S3
}
state S1:H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S2 else goto S1
}
state S2:H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S1 else goto S2
}
state S3:L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S4 else goto S3
}
state S4:L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S3 else goto S4
}
in fall
Nested States

prog lease = timer:L, data1: H, data2: L, mode:L
in
let state master:L = {
  if mode = 0
    then mode := 1 goto S1
    else mode := 0 goto S3
}
state S1:H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S2 else goto S1
}
state S2:H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S1 else goto S2
}
state S3:L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S4 else goto S3
}
state S4:L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S3 else goto S4
}
in fall
Nested States

prog lease = timer: L, data1: H, data2: L, mode: L
in
let state master: L = {
  if mode = 0
    then mode := 1 goto S1
    else mode := 0 goto S3
}
state S1: H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S2 else goto S1
}
state S2: H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S2 else goto S1
}
state S3: L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S4 else goto S3
}
state S4: L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S3 else goto S4
}
in
state group1: L = {
  let state S1: H = {
    if data1 = 0 then goto S2 else goto S1
  }
  state S2: H = {
    if data1 = 0 then goto S1 else goto S2
  }
}
in
state S1: H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S1 else goto S2
}
in
state S2: H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S1 else goto S2
}
in
state S3: L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S4 else goto S3
}
in
state S4: L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S3 else goto S4
}
prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L = {
    if mode = 0
        then mode := 1 goto S1
        else mode := 0 goto S3
}
state S1:H = {
    timer := timer - 1;
    if timer = 0 then goto master else skip;
    if data1 = 0 then goto S2 else goto S1
}
state S2:H = {
    timer := timer - 1;
    if timer = 0 then goto master else skip;
    if data1 = 0 then goto S1 else goto S2
}
state S3:L = {
    timer := timer - 1;
    if timer = 0 then goto master else skip;
    if data2 = 0 then goto S4 else goto S3
}
state S4:L = {
    timer := timer - 1;
    if timer = 0 then goto master else skip;
    if data2 = 0 then goto S3 else goto S4
}
in fall

state group1:L = {
    let state S1:H = {
        if data1 = 0 then goto S2 else goto S1
    }
    state S2:H = {
        if data1 = 0 then goto S1 else goto S2
    }
    timer := timer - 1;
    if timer = 0 then goto master else skip;
    if data1 = 0 then goto S1 else goto S2
}
in
state S4:L = {
    timer := timer - 1;
    if timer = 0 then goto master else skip;
    if data2 = 0 then goto S3 else goto S4
}
in fall
Nested States

prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L = {
if mode = 0
then mode := 1 goto S1
else mode := 0 goto S3
}
state S1:H = {
timer := timer - 1;
if timer = 0 then goto master else skip;
if data1 = 0 then goto S2 else goto S1
}
state S2:H = {
timer := timer - 1;
if timer = 0 then goto master else goto S2
}
state S3:L = {
timer := timer - 1;
if timer = 0 then goto S1 else goto S2
}
state S4:L = {
timer := timer - 1;
if timer = 0 then goto master else goto S4
}
in fall

prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L = {
if mode = 0
then mode := 1; goto group1
else mode := 0; goto group2
}
state group1:L = {
let state S1:H = {
if data1 = 0 then goto S2 else goto S1
}
state S2:H = {
if data1 = 0 then goto S1 else goto S2
}
state S3:L = {
timer := timer - 1;
if timer = 0 then goto master else fall
}
state group2:L = {
let state S3:L = {
if data2 = 0 then goto S4 else goto S3
}
state S4:L = {
if data2 = 0 then goto S3 else goto S4
}
in fall

Xun Li PLDI 2011 34
Nested States

prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L = {
  if mode = 0
    then mode := 1 goto S1
    else mode := 0 goto S3
}
state S1:H = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data1 = 0 then goto S2 else goto S1
}
state S2:H = {
  timer := timer - 1;
  if timer = 0 then goto master else goto S2
}
state S3:L = {
  timer := timer - 1;
  if timer = 0 then goto master else skip;
  if data2 = 0 then goto S4 else goto S3
}
state S4:L = {
  timer := timer - 1;
  if timer = 0 then goto master else goto S4
}
in fall

prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L = {
  if mode = 0
    then mode := 1; goto group1
    else mode := 0; goto group2
}
state group1:L = {
  let state S1:H = {
    if data1 = 0 then goto S2 else goto S1
  }
  state S2:H = {
    if data1 = 0 then goto S1 else goto S2
  }
  timer := timer - 1;
  if timer = 0 then goto master else fall;
  if data1 = 0 then goto S1 else goto S2
}
state group2:L = {
  let state S3:L = {
    if data2 = 0 then goto S4 else goto S3
  }
  state S4:L = {
    if data2 = 0 then goto S3 else goto S4
  }
  timer := timer - 1;
  if timer = 0 then goto master else fall;
  if data2 = 0 then goto S3 else goto S4
}
in fall

Xun Li  PLDI 2011
Caisson

• What is it?
  – An security extension to a subset of Verilog

• What are the cool features in it?
  – Type System Extension to Verilog
  – State Machine Aware Programming
    • Nested States
    • Parameterized States
Parameterized States

\[
\text{prog lease} = \text{timer:}L, \text{data1:}H, \text{data2:}L, \text{mode:}L
\]
in
let state master: L = {
    if mode = 0
        then mode := 1; goto group1
        else mode := 0; goto group2
}
state group1: L = {
    let state S1: H = {
        if data1 = 0 then goto S2 else goto S1
    }
    state S2: H = {
        if data1 = 0 then goto S1 else goto S2
    }
    in
    timer := timer - 1;
    if timer = 0 then goto master else fall
}
state group2: L = {
    let state S3: L = {
        if data2 = 0 then goto S4 else goto S3
    }
    state S4: L = {
        if data2 = 0 then goto S3 else goto S4
    }
    in
    timer := timer - 1;
    if timer = 0 then goto master else fall
}
in fall
Parameterized States

• We want to have the same hardware logic to operate at different security levels

• Solution: Type Polymorphism
Parameterized States

prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L = {
  if mode = 0
    then mode := 1; goto group1
  else mode := 0; goto group2
}
state group1:L = {
  let state S1:H = {
    if data1 = 0 then goto S2 else goto S1
  }
  state S2:H = {
    if data1 = 0 then goto S1 else goto S2
  }
  in
  timer := timer - 1;
  if timer = 0 then goto master else fall
}
state group2:L = {
  let state S3:L = {
    if data2 = 0 then goto S4 else goto S3
  }
  state S4:L = {
    if data2 = 0 then goto S3 else goto S4
  }
  in
  timer := timer - 1;
  if timer = 0 then goto master else fall
}
in fall

prog lease = timer:L, data1: H, data2: L , mode:L
in
let state master:L() = {
  if mode = 0
    then mode := 1; goto group1(data1)
  else mode := 0; goto group2(data2)
}
state group1:L (data: A) [L < A] = {
  let state S1:A() = {
    if data = 0 then goto S2 else goto S1
    if data1 = 0 then goto S1 else goto S2
  }
  in
  timer := timer - 1;
  if timer = 0 then goto master else fall
}
in fall
Outline

• Motivation
• Type System Based Non-interference
• Caisson
• **Case Study: A Secure Processor Design**
• Future Work
First-ever Provably Secure Processor

• Processor Features
  – RISC ISA
  – Pipeline
  – Cache/Memory

• Security Policy
  – Two level security lattice: High and Low

• Goal
  – High never flows to Low
  – Allow the same processor logic to process data at different security levels
Evaluation

• Toolset
  – Caisson-Compiler: Caisson -> Verilog
  – Synthesize: Stratix II FPGA
  – Area, timing and power: Synopsis Design Compiler and the SAED 90nm technology library

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>GLIFT</th>
<th>Caisson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis Time (min)</td>
<td>1:50</td>
<td>153:56</td>
<td>4:04</td>
</tr>
<tr>
<td>Area (µm²)</td>
<td>38462.86</td>
<td>128506.89</td>
<td>52088.78</td>
</tr>
<tr>
<td>CPU delay (ns)</td>
<td>2.96</td>
<td>7.79</td>
<td>4.32</td>
</tr>
<tr>
<td>Power (µW)</td>
<td>614.148</td>
<td>1730</td>
<td>666.912</td>
</tr>
</tbody>
</table>

Only 10 lines of extra code for security!

Xun Li  PLDI 2011
Outline

• Motivation
• Type System Based Non-interference
• Caisson
• Case Study: A Secure Processor Design
• Future Work
Future Work

• Drawback of Caisson
  – Require resource to be statically partitioned among security levels

• Provably Secure architecture:

  ![Diagram]

  - APP1
  - APP2
  - Micro-Kernel
  - Processor

• Verify Hardware Designs for other properties
Conclusion

• Information Flow Security needs to be enforced from **ground up**

• **Caisson**, as an **SM-based** security extension to Verilog, allows us to design **provably information flow secure** hardware, and more.

• Compiler from Caisson to Verilog available at
  – [https://github.com/vineethk/Caisson](https://github.com/vineethk/Caisson)