Formal Method on Hardware Security

Oct 2022

Table of Contents

Table of Contents

CPU Models

- RTL Implementation

CPU Models

- RTL Implementation
- ISA Specification

CPU Models

- RTL Implementation
- ISA Specification: A set of implementations

Table of Contents

Refinement Property

- RTL refines ISA

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU
- 1-entry FIFO refines 2-entry FIFO

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU
- 1-entry FIFO refines 2-entry FIFO
  - compare the enqueue(), dequeue() trace of the FIFO

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU
- 1-entry FIFO refines 2-entry FIFO
  - compare the enqueue(), dequeue() trace of the FIFO
  - 1-entry: enq(), deq(), enq(), deq() ...
  - 2-entry: enq(), deq() ... or enq(), enq(), deq(), ... or ...

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU
- 1-entry FIFO refines 2-entry FIFO
  - compare the enqueue(), dequeue() trace of the FIFO
  - 1-entry: enq(), deq(), enq(), deq() ...
  - 2-entry: enq(), deq() ... or enq(), enq(), deq(), ... or ...
  - Not correct in Verilog, is a good property in Bluespec

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU
- 1-entry FIFO refines 2-entry FIFO regarding to enq(), deq() event sequences.

Refinement Property

- RTL refines ISA
- 1-cycle ALU refines symbolic-cycle ALU regarding to RTL implementations.
- 1-entry FIFO refines 2-entry FIFO regarding to enq(), deq() event sequences.

Refinement Property

- RTL refines ISA regarding to commit cycle of each instruction. (How to define the commit cycle of a OoO CPU?)
- 1-cycle ALU refines symbolic-cycle ALU regarding to RTL implementations.
- 1-entry FIFO refines 2-entry FIFO regarding to enq(), deq() event sequences.

Refinement Property

- RTL refines ISA regarding to commit cycle of each instruction. (How to define the commit cycle of a OoO CPU?)
- 1-cycle ALU refines symbolic-cycle ALU regarding to RTL implementations.
- 1-entry FIFO refines 2-entry FIFO regarding to enq(), deq() event sequences.

Refinement Property

- RTL refines ISA regarding to commit cycle of each instruction. (How to define the commit cycle of a OoO CPU?)
- 1-cycle ALU refines symbolic-cycle ALU regarding to RTL implementations.
- 1-entry FIFO refines 2-entry FIFO regarding to enq(), deq() event sequences. Bluespec cares about event, Verilog cares about signals

Refinement Property

- RTL refines ISA regarding to commit cycle of each instruction. (How to define the commit cycle of a OoO CPU?)
- 1-cycle ALU refines symbolic-cycle ALU regarding to RTL implementations.
- 1-entry FIFO refines 2-entry FIFO regarding to enq(), deq() event sequences. Bluespec cares about event, Verilog cares about signals
- The real implementation refines our model in the paper regarding to ... :)

Table of Contents

More CPU Models

- RTL Implementation
- ISA Specification: A set of implementations

More CPU Models

- RTL Implementation
- RTL, but replace ALU with a symbolic-cycle ALU
- RTL, but replace all caches with symbolic-cyle MEM
- ISA Specification: A set of implementations

Table of Contents

More Properties on CPU

- Example: Memory Safety: any memory read/write instruction should within the boundary

More Properties on CPU

- Example: Memory Safety: any memory read/write instruction should within the boundary
- Safety property and Liveness property

More Properties on CPU

- Example: Memory Safety: any memory read/write instruction should within the boundary
- Safety property and Liveness property
  - Safety property: something bad will not happen
  - Liveness property: something good will eventually happen (no live lock)

More Properties on CPU

- Example: Memory Safety: any memory read/write instruction should within the boundary
- Safety property and Liveness property
  - Safety property: something bad will not happen
  - Liveness property: something good will eventually happen (no live lock)
- Single-trace property and Hyperproperty

Table of Contents

Single-trace Property and Hyperproperty

- Memory Safety: any memory read/write instruction should within the boundary

Single-trace Property and Hyperproperty

- Memory Safety: any memory read/write instruction should within the boundary
- Non-interference property: Say we have a combinational function $y_1, y_2 = f(x_1, x_2)$

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_1', x_2, \\
\text{let} \quad & y_1, y_2 = f(x_1, x_2), \\
& y_1', y_2' = f(x_1', x_2), \\
\text{we have,} \quad & y_1 = y_1'
\end{aligned}
$$

Single-trace Property and Hyperproperty

- Memory Safety: any memory read/write instruction should within the boundary
- Non-interference property: Say we have a combinational function $y_1, y_2 = f(x_1, x_2)$

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_1', x_2, \\
\text{let} \quad & y_1, y_2 = f(x_1, x_2), \\
& y_1', y_2' = f(x_1', x_2), \\
\text{we have,} \quad & y_1 = y_1'
\end{aligned}
$$

- Hyperproperty is a property on a multiple traces

Table of Contents

Example: Non-interference Property

- A property on a pair of execution traces: for a function $y_1, y_2 = f(x_1, x_2)$:

$$
\begin{array}{rl}
 & x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall & x_1, x_1', x_2, \\
\text{let} & y_1, y_2 = f(x_1, x_2), \\
 & y_1', y_2' = f(x_1', x_2), \\
\text{we have,} & y_1 = y_1'
\end{array}
$$

Example: Non-interference Property

- A property on a pair of execution traces: for a function $y_1, y_2 = f(x_1, x_2)$:

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_1', x_2, \\
\text{let} \quad & y_1, y_2 = f(x_1, x_2), \\
& y_1', y_2' = f(x_1', x_2), \\
\text{we have,} \quad & y_1 = y_1'
\end{aligned}
$$

- Verify it with single trace: taint analysis: for a function $y_1, y_2 = f(x_1, x_2)$, we design an augment $yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2)$, such that:

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_2, \\
\text{let} \quad & yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2), \\
\text{we have,} \quad & yt_1 = 0
\end{aligned}
$$

Table of Contents

An Estimation Scheme: Taint Analysis

- Taint analysis: for a function $y_1, y_2 = f(x_1, x_2)$, we design an augment $yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2)$, such that:

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_2, \\
\text{let} \quad & yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2), \\
\text{we have,} \quad & yt_1 = 0
\end{aligned}
$$

An Estimation Scheme: Taint Analysis

- Taint analysis: for a function $y_1, y_2 = f(x_1, x_2)$, we design an augment $yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2)$, such that:

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_2, \\
\text{let} \quad & yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2), \\
\text{we have,} \quad & yt_1 = 0
\end{aligned}
$$

- Finally, let me show a tool to automatically augment verilog with Taint functions.

An Estimation Scheme: Taint Analysis

- Taint analysis: for a function $y_1, y_2 = f(x_1, x_2)$, we design an augment $yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2)$, such that:

$$
\begin{aligned}
& x_1 \text{ not interfere } y_1 \quad \text{iff} \\
\forall \quad & x_1, x_2, \\
\text{let} \quad & yt_1, yt_2 = ft(x_1, xt_1, x_2, xt_2), \\
\text{we have,} \quad & yt_1 = 0
\end{aligned}
$$

- Finally, let me show a tool to automatically augment verilog with Taint functions. More tools (JasperGold and $\sim$100 more are coming `http://mtlcad.mit.edu/setup.html`).

Taint Analysis: Gate-Level Information Flow Tracking (GLIFT)

- Augment at each gate
- More on `https://ieeexplore.ieee.org/document/5948366`

Taint Analysis: Gate-Level Information Flow Tracking (GLIFT)

- Augment at each gate
- When compose gates together or compose cycles together, will be conservative.
- More on `https://ieeexplore.ieee.org/document/5948366`