Stanford University

Symbolic Quick Error Detection with CoSA

PIs: Clark Barrett[1], Subhasish Mitra[1,2]
Students: Eshan Singh[2], Karthik Ganesan[2], Makai Mann[2]

1: Department of Computer Science
2: Department of Electrical Engineering

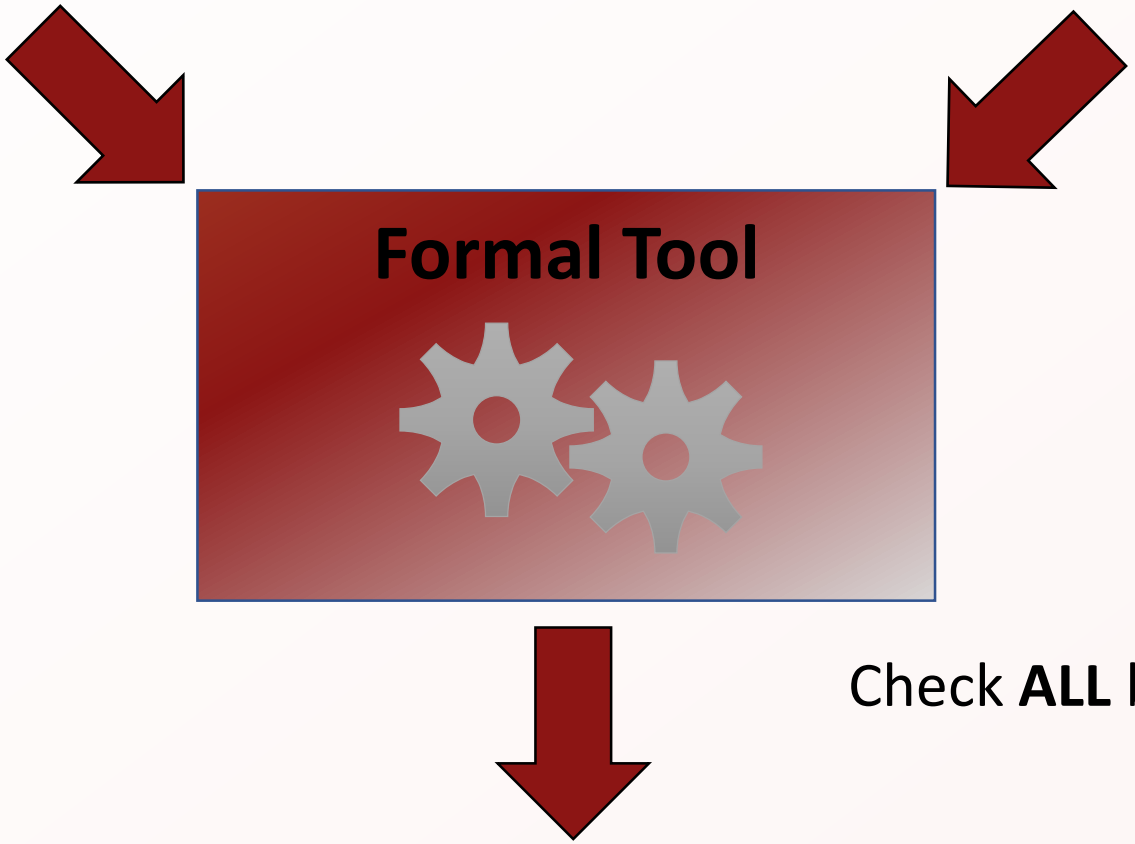
Overview

- Model Checking
  - CoSA open-source model checker
- Symbolic QED
  - Processor verification built on top of model checking
- Demo
  - Find a bug in open source RiscV core

Model Checking

Property

RTL Design

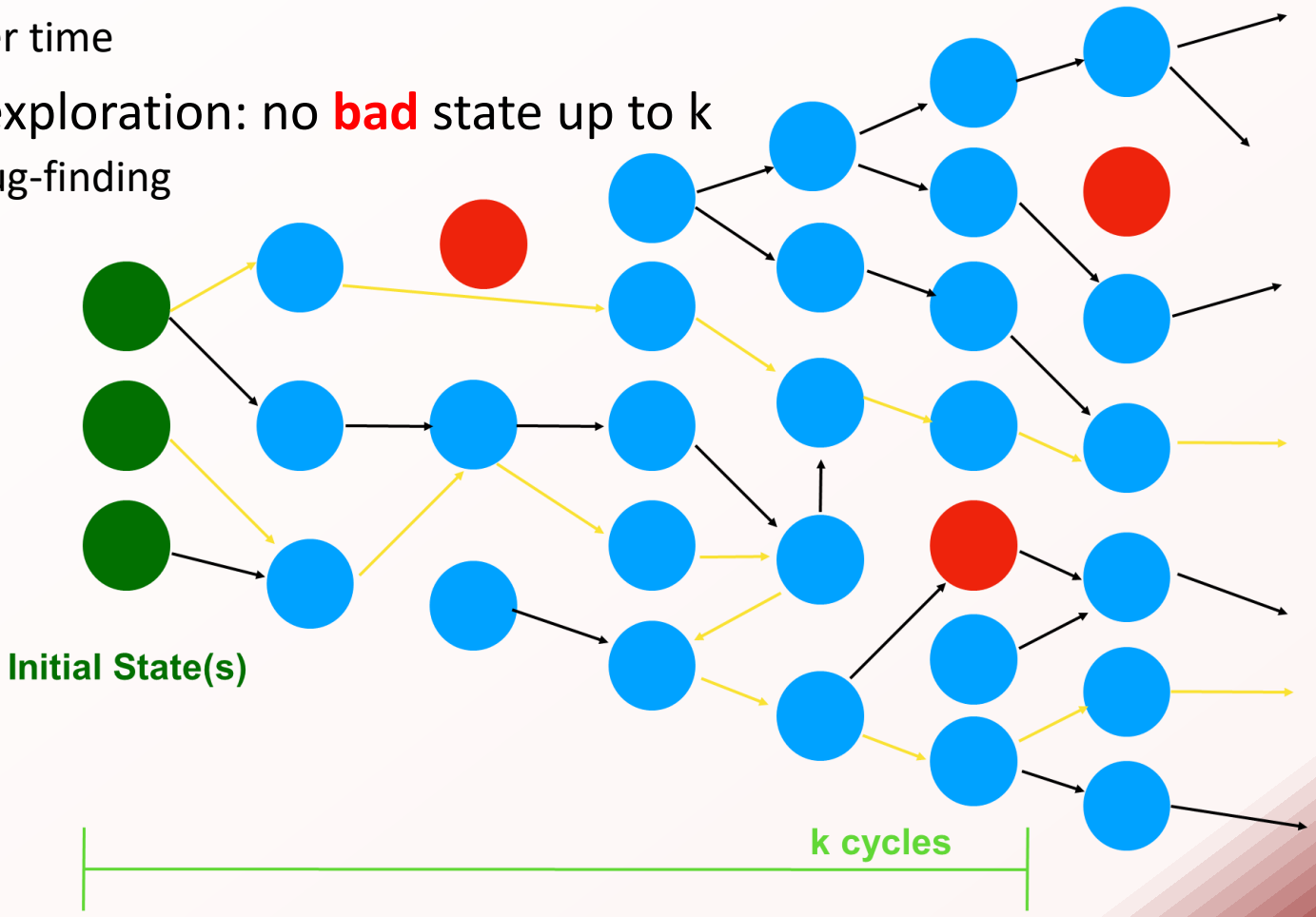


Check **ALL** behaviors

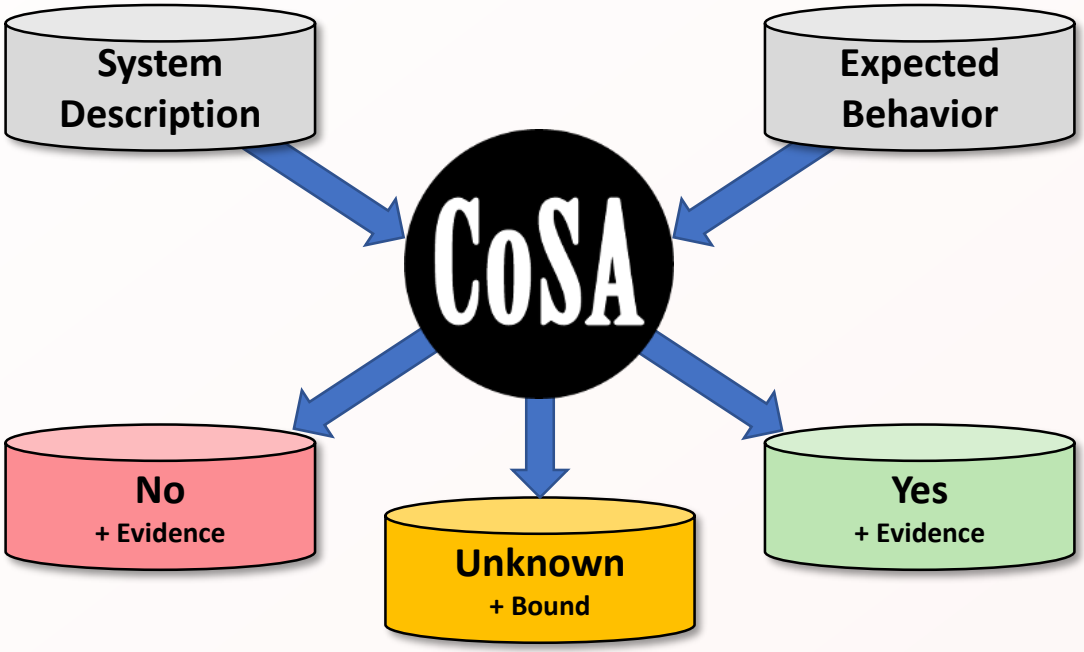
Counterexample

Bounded Model Checking

- Start with a circuit description
  - "Unroll" it over time
- Breadth-first exploration: no **bad** state up to k
  - Focused on bug-finding
- Simulation



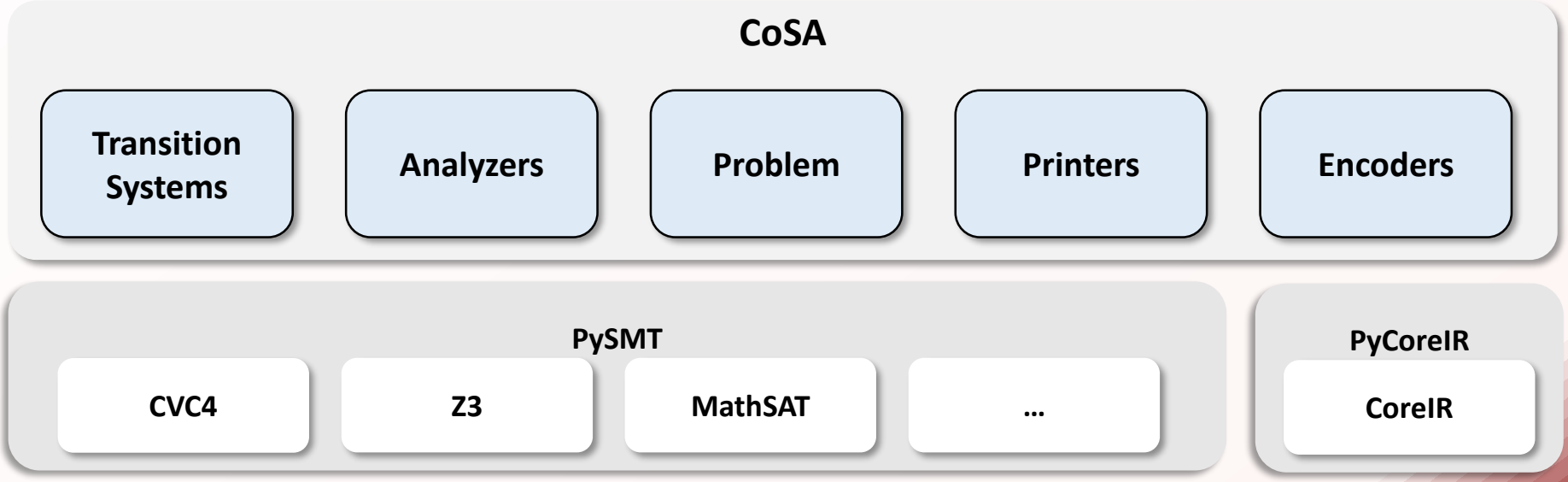
CoSA Model Checker



- Given a **System Description** (M) and an **Expected Behavior** (φ), check if $M \models \varphi$
- **Bounded** techniques respond with **No (bug)** or **Unknown (bounded proof)**
- **Unbounded** techniques can also prove the property: answer **Yes**
- **Open-source** SMT-based model checker

CoSA Architecture

- Inputs include:
  - Verilog
  - CoreIR
  - BTOR
- Verilog/SystemVerilog support using Yosys as a frontend encoder
- PySMT for interfacing with state-of-the-art SMT solvers



QED Tests

- Idea from post-silicon validation
- Take an existing test program and transform it into a QED test
- Divide register file of processor in half and associate registers
  - R1 : R17, R2: R18, …
- Run a sequence of instructions
- Run the same sequence on the duplicate registers

QED Trace

```
• • •
R1  ← R1  + 5
R17 ← R17 + 5
R1  == R17
R2  ← R2  – R1
R18 ← R18 – R17
R2  == R18
R3  ← R1  * R2
R19 ← R17 * R18
R3  == R19
• • •
```

Quick!

Symbolic QED

- Use formal tool to search over *all possible* QED tests
- Start with QED tests with 1 instruction, then 2, 3, …
- Use model checker with "universal" property:
  - QED-consistent state: each original register/memory location has the same value as its duplicate register/memory location
  - Starting from a QED-consistent state, every QED-test ends in a QED-consistent state

Symbolic QED

"Universal" Property +
QED-consistent initial state

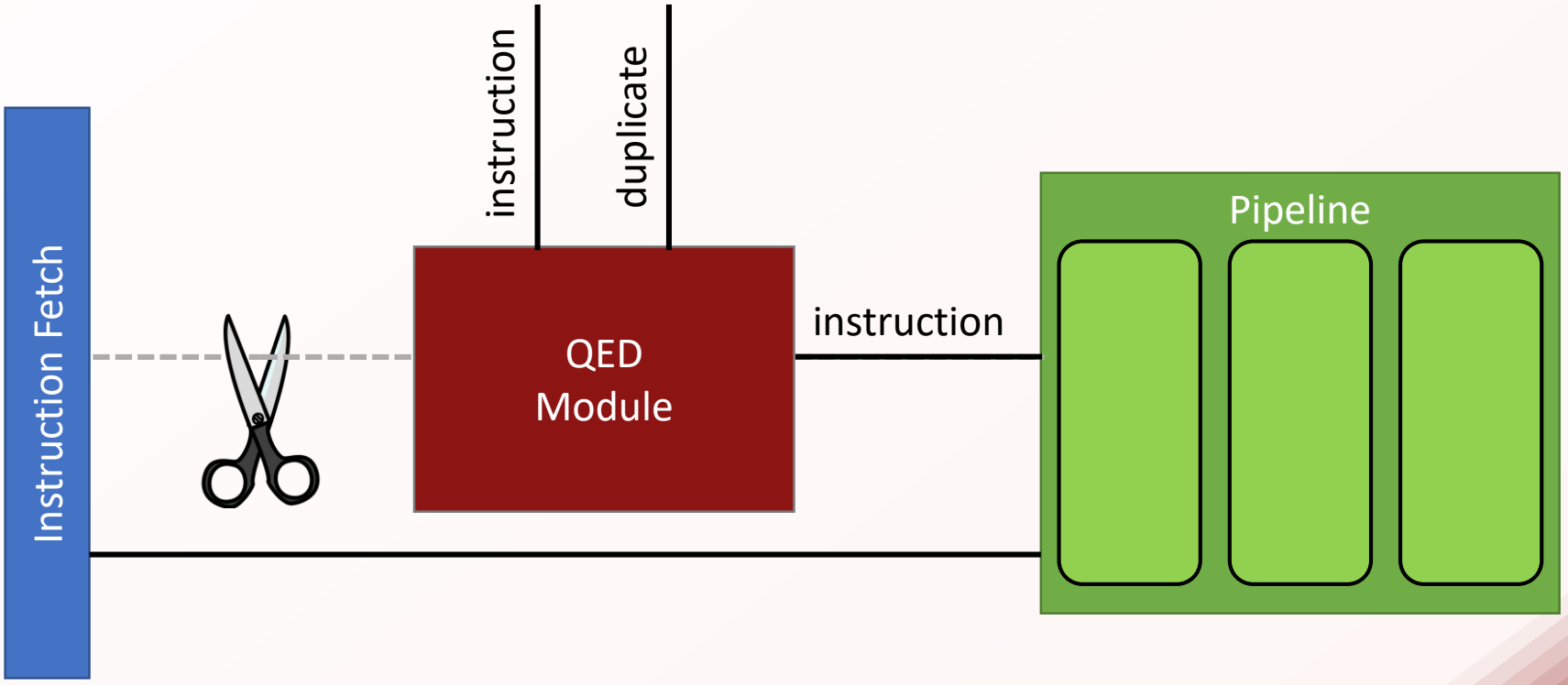
RTL Design + Special QED Module
(no hardware overhead)

Check **ALL** QED tests up to a
certain number of instructions

Counterexample

QED Module

- Analysis enabled by QED module
  - Small module used only in pre-silicon
  - Transforms instructions to a QED test
  - Used to constrain valid QED sequences in formal tool



Verifying RIDECORE

- Interactive tutorial at: https://github.com/makaimann/ride-core-demo/tree/demo
- Recreates how a bug in the open-source RISC-V processor, RIDECORE, was found using SQED
- Start demo with: `./start.sh`
- Step forward through commits with: `./next.sh`

Step 1: Minor RTL Modifications

- topsim.v: Add `(* keep *)` attributes to wires for Yosys front-end
  - Yosys does “cone of influence” reduction on primary outputs
  - Attribute tells it not to remove signal and to keep the name
- pipeline.v: Turn the ride-core into a 1-wide instead of 2-wide fetch
  - Disable second instruction by hardcoding **inv2_if** to 1
  - Not necessary, *optimization only*
- pipeline_if.v: Disable branch prediction
  - Not necessary, *optimization only*
  - Manually cut and assign **predict_cond** to zero

Step 2: Add the QED module

- pipeline.v: Instantiate the QED module
- pipeline.v: Add logic for tracking the number of committed instructions
  - Processor dependent
- pipeline.v, topsim.v: disconnect instruction from instruction fetch and make it a primary input
  - **Any signal you make assumptions on should be a primary input to avoid accidentally over-constraining**
- topsim.v: Constrain the instruction to be from the instruction set, with inst_constraint.sv

Step 2: Add the QED module

```
// EDIT- add in the QED module.

wire                    qed_vld_out;
(* keep *)
wire                    qed_exec_dup;
wire [31:0]         qed_ifu_instruction;

// instruction1 and qed_exec_dup are cutpoints
qed qed0 ( // Inputs
   .clk(clk),
         .rst(reset),
         .ena(1'b1),
         .ifu_qed_instruction(inst1),
         .exec_dup(qed_exec_dup),
         .stall_IF(stall_IF),
   // outputs
         .qed_ifu_instruction(qed_ifu_instruction),
         .vld_out(qed_vld_out));

// EDIT
```

```
// EDIT: Use the inst_constraint module to constrain instruction to be
//       a valid instruction from the ISA
(* keep *)
wire [6:0] opcode;
(* keep *)
wire [4:0] rd;
(* keep *)
wire [4:0] rs1;
(* keep *)
wire [4:0] rs2;
assign opcode = instruction[6:0];
assign rd = instruction[11:7];
assign rs1 = instruction[19:15];
assign rs2 = instruction[24:20];
inst_constraint inst_constraint0(.clk(clk),
                                 .instruction(instruction));
// EDIT END
```

```
// EDIT: Insert the qed ready logic -- tracks number of committed instructions
(* keep *)
wire qed_ready;
(* keep *)
reg [15:0] num_orig_insts;
(* keep *)
reg [15:0] num_dup_insts;
wire [1:0] num_orig_commits;
wire [1:0] num_dup_commits;

// Instruction with destination register as 5'b0 is a NOP so ignore those
assign num_orig_commits = ((arfwe1 == 1)&&(dstarf1 < 16)&&(dstarf1 != 5'b0)
                 &&(arfwe2 == 1)&&(dstarf2 < 16)&&(dstarf2 != 5'b0)) ? 2'b10 :
                ((((arfwe1 == 1)&&(dstarf1 < 16)&&(dstarf1 != 5'b0)
                  &&((arfwe2 != 1)||(dstarf2 >= 16)||(dstarf2 == 5'b0)))
                  ||((arfwe2 == 1)&&(dstarf2 < 16)&&(dstarf2 != 5'b0)
                &&((arfwe1 != 1)||(dstarf1 >= 16)||(dstarf1 == 5'b0)))) ? 2'b01 : 2'b00) ;


// When destination register is 5'b0, it remains the same for both original and duplicate
assign num_dup_commits = ((arfwe1 == 1)&&(dstarf1 >= 16)
                 &&(arfwe2 == 1)&&(dstarf2 >= 16)) ? 2'b10 :
                ((((arfwe1 == 1)&&(dstarf1 >= 16)
                  &&((arfwe2 != 1)||(dstarf2 < 16)))
                  ||((arfwe2 == 1)&&(dstarf2 >= 16)
                &&((arfwe1 != 1)||(dstarf1 < 16)))) ? 2'b01 : 2'b00) ;

always @(posedge clk)
  begin
 if (reset) begin
    num_orig_insts <= 16'b0;
    num_dup_insts <= 16'b0;
 end else begin
    num_orig_insts <= num_orig_insts + {14'b0,num_orig_commits};
    num_dup_insts <= num_dup_insts + {14'b0,num_dup_commits};
 end
  end

assign qed_ready = (num_orig_insts == num_dup_insts);

// EDIT END
```

```
module pipeline
 (
  input wire [`INSN_LEN-1:0] inst1,
  input wire           clk,
  input wire           reset,
  output reg [`ADDR_LEN-1:0]   pc,
  input wire [4*`INSN_LEN-1:0] idata,
  output wire [`DATA_LEN-1:0]  dmem_wdata,
  output wire          dmem_we,
  output wire [`ADDR_LEN-1:0]  dmem_addr,
  input wire [`DATA_LEN-1:0]   dmem_data
  );
  wire  stall_IF;
  wire  kill_IF;
  wire  stall_ID;
  wire  kill_ID;
  wire  stall_DP;
  wire  kill_DP;
//   reg [`ADDR_LEN-1:0] pc;
```

```
// EDIT: manually cut inst1, want to drive this from the
//       we don't need to include the instruction fetch
wire [`INSN_LEN-1:0] cut_inst1;

pipeline_if pipe_if(
            .clk(clk),
            .reset(reset),
            .pc(pc),
            .predict_cond(prcond),
            .npc(npc),
            .inst1(cut_inst1),
            .inst2(inst2),
            .invalid2(invalid2_pipe),
            .btbpht_we(combranch),
            .btbpht_pc(pc_combranch),
            .btb_jmpdst(jmpaddr_combranch),
            .pht_wcond(brcond_combranch),
            .mpft_valid(mpft_valid),
            .pht_bhr(bhr_combranch), //when PHT write
            .prmiss(prmiss),
            .prsuccess(prsuccess),
            .prtag(buf_spectag_branch),
            .bhr(bhr),
            .spectagnow(tagreg),
            .idata(idata)
            );
```

Step 3: Add CoSA configuration files

- Top-level CoSA *problem* file

Dumps a vcd file if a counterexample is found

If not already initialized, start at 0

Sets up a verification problem, can have multiple

```
[General]
model_file:
ridecore.vlist[top],reset_procedure.ets
clock_behaviors: DetClock(clk, 1)
vcd: True

[DEFAULT]
bmc_length_min: 24
precondition: reset_done
default_initial_value: 0

[QED]
description: “Check for Symbolic QED
consistency”
formula: property.txt
verification: safety
```

Step 3: Add CoSA configuration files

- QED property file
- On posedge clk (clk = 1)
- And qed_ready = 1
  - Same number of original and duplicate instructions have been committed
- Then the register file should be QED consistent

```
((clk = 1_1) & (pipe.qed_ready = 1_1)) ->
('pipe.aregfile.regfile.mem[1]' = 'pipe.aregfile.regfile.mem[17]') &
('pipe.aregfile.regfile.mem[2]' = 'pipe.aregfile.regfile.mem[18]') &
('pipe.aregfile.regfile.mem[3]' = 'pipe.aregfile.regfile.mem[19]') &
('pipe.aregfile.regfile.mem[4]' = 'pipe.aregfile.regfile.mem[20]') &
('pipe.aregfile.regfile.mem[5]' = 'pipe.aregfile.regfile.mem[21]') &
('pipe.aregfile.regfile.mem[6]' = 'pipe.aregfile.regfile.mem[22]') &
('pipe.aregfile.regfile.mem[7]' = 'pipe.aregfile.regfile.mem[23]') &
('pipe.aregfile.regfile.mem[8]' = 'pipe.aregfile.regfile.mem[24]') &
('pipe.aregfile.regfile.mem[9]' = 'pipe.aregfile.regfile.mem[25]') &
('pipe.aregfile.regfile.mem[10]' = 'pipe.aregfile.regfile.mem[26]') &
('pipe.aregfile.regfile.mem[11]' = 'pipe.aregfile.regfile.mem[27]') &
('pipe.aregfile.regfile.mem[12]' = 'pipe.aregfile.regfile.mem[28]') &
('pipe.aregfile.regfile.mem[13]' = 'pipe.aregfile.regfile.mem[29]') &
('pipe.aregfile.regfile.mem[14]' = 'pipe.aregfile.regfile.mem[30]') &
('pipe.aregfile.regfile.mem[15]' = 'pipe.aregfile.regfile.mem[31]'))
```

Note: Should actually all be on one line, but expanded for readability

Step 3.1: Run CoSA

```
CoSA --problems ./cosa/problem.txt [-v <1..4>]
```

Step 4: Fix the bug

ridecore / ridecore

Watch 28 | Star 132 | Fork 26

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Insights

signed/lohi signal bug fixed in rs_mul.v

master

Browse files

FUJIV committed on Jul 12, 2017

1 parent 112a9bf commit 200c6a663e01cb2231004bb2543e7ce8b1c92cca

Showing 1 changed file with 3 additions and 3 deletions.

Unified | Split

6 src/fpga/rs_mul.v

View file

```
@@ -346,9 +346,9 @@ module rs_mul
346          .wrrftag((we1 && (waddr1 == 1)) ? wrrftag_1 : wrrftag_2),
347          .wdstval((we1 && (waddr1 == 1)) ? wdstval_1 : wdstval_2),
348          .wspectag((we1 && (waddr1 == 1)) ? wspectag_1 : wspectag_2),
349 -        .wsrc1_signed((we1 && (waddr1 == 0)) ? wsrc1_signed_1 : wsrc1_signed_2),
350 -        .wsrc2_signed((we1 && (waddr1 == 0)) ? wsrc2_signed_1 : wsrc2_signed_2),
351 -        .wsel_lohi((we1 && (waddr1 == 0)) ? wsel_lohi_1 : wsel_lohi_2),
352          .we((we1 && (waddr1 == 1)) || (we2 && (waddr2 == 1))),
353          .ex_src1(ex_src1_1),
354          .ex_src2(ex_src2_1),
```

```
346          .wrrftag((we1 && (waddr1 == 1)) ? wrrftag_1 : wrrftag_2),
347          .wdstval((we1 && (waddr1 == 1)) ? wdstval_1 : wdstval_2),
348          .wspectag((we1 && (waddr1 == 1)) ? wspectag_1 : wspectag_2),
349 +        .wsrc1_signed((we1 && (waddr1 == 1)) ? wsrc1_signed_1 : wsrc1_signed_2),
350 +        .wsrc2_signed((we1 && (waddr1 == 1)) ? wsrc2_signed_1 : wsrc2_signed_2),
351 +        .wsel_lohi((we1 && (waddr1 == 1)) ? wsel_lohi_1 : wsel_lohi_2),
352          .we((we1 && (waddr1 == 1)) || (we2 && (waddr2 == 1))),
353          .ex_src1(ex_src1_1),
354          .ex_src2(ex_src2_1),
```

Thank you!