

Front-End Symposium Intro

Steve Hoover
Redwood EDA

FRONT-END AGENDA

Steve:

- Open-Source H/W Trends
- TL-Verilog Primer

Ákos:

- Formal Verification of WARP-V, a TL-Verilog RISC-V Core Generator

Ahmed:

- Top-Down Transaction-Level Design with TL-Verilog

OPEN SOURCE H/W - What's keeping us back?

1. Access to tools
2. Access to h/w
3. Complexity/Scale

...AND THE WALLS ARE BREAKING DOWN!!!

1. Access to tools
 - Complete open-source FPGA design flows now exist
2. Access to h/w
 - Cloud FPGAs are the answer!
3. Complexity/Scale
 - TL-Verilog

IMPACT

- Greater impact than open-source s/w
- Start-up culture will thrive!
- Just in time for the “golden age of computing”

4th Barrier - Patents



MAKERCHIP

1. Access to tools
 - in browser
2. Access to h/w
 - Cloud FPGAs
3. Complex./Scale
 - TL-Verilog

The screenshot displays the Makerchip web interface with three main panels:

- EDITOR:** Shows Verilog code for a pipelined Pythagorean theorem calculation:

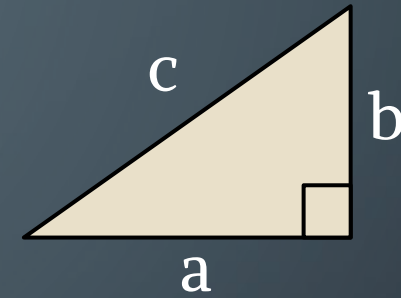
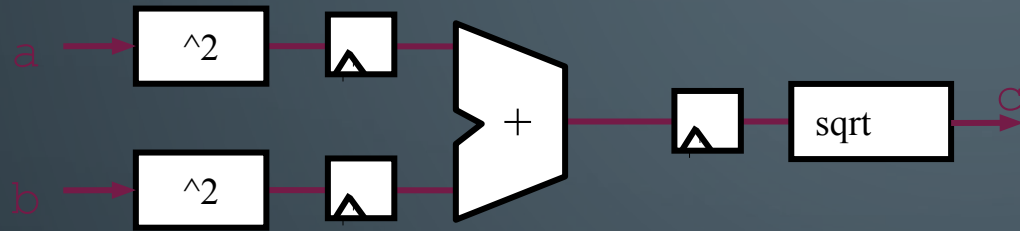
```
@1
Saa_sq[7:0] = $aa[3:0] ** 2;
Sbb_sq[7:0] = $bb[3:0] ** 2;
@2
Scc_sq[8:0] = $aa_sq + $bb_sq;
@3
Scc[4:0] = sqrt($cc_sq);
```
- DIAGRAM:** A block diagram showing three pipeline stages: @1 (calculating a^2 and b^2), @2 (adding them to get c^2), and @3 (calculating the square root of c^2 to get c).
- WAVEFORM:** A timing diagram showing the clock (clk) and the outputs of the pipeline stages: a^2 , b^2 , c^2 , and c .

TUTORIAL-VALID panel shows a diagram of the pipelined logic and a caption: "Figure 1: Pipelined Pythagorean Theorem Logic".

This pipeline is 3 cycles deep. It has a throughput of one transaction per cycle, where a transaction performs one Pythagorean Theorem calculation per cycle.

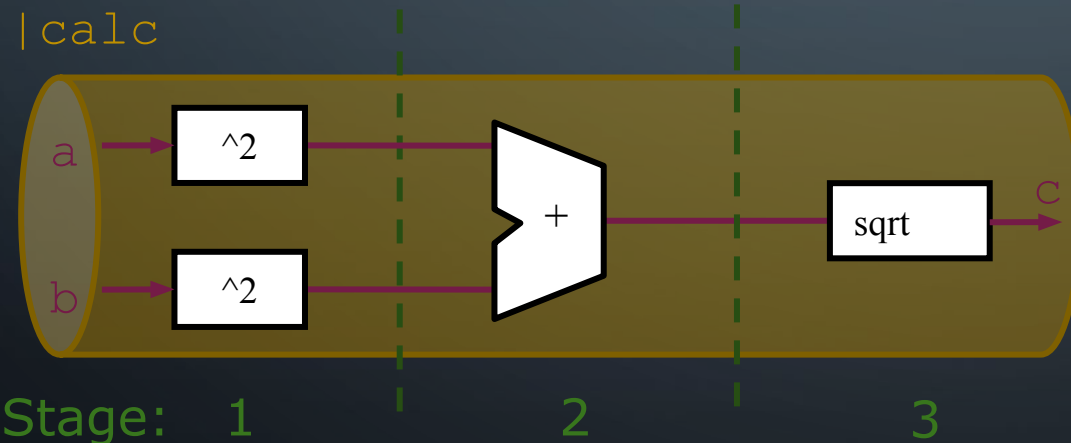
TIMING ABSTRACTION

RTL:



$$c = \text{sqrt}(a^2 + b^2)$$

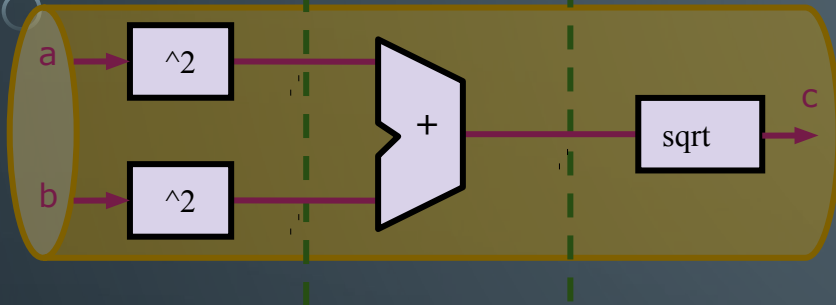
Timing-abstract:



→ Flip-flops and staged signals are implied from context.

TL-VERILOG VS. SYSTEMVERILOG

|calc



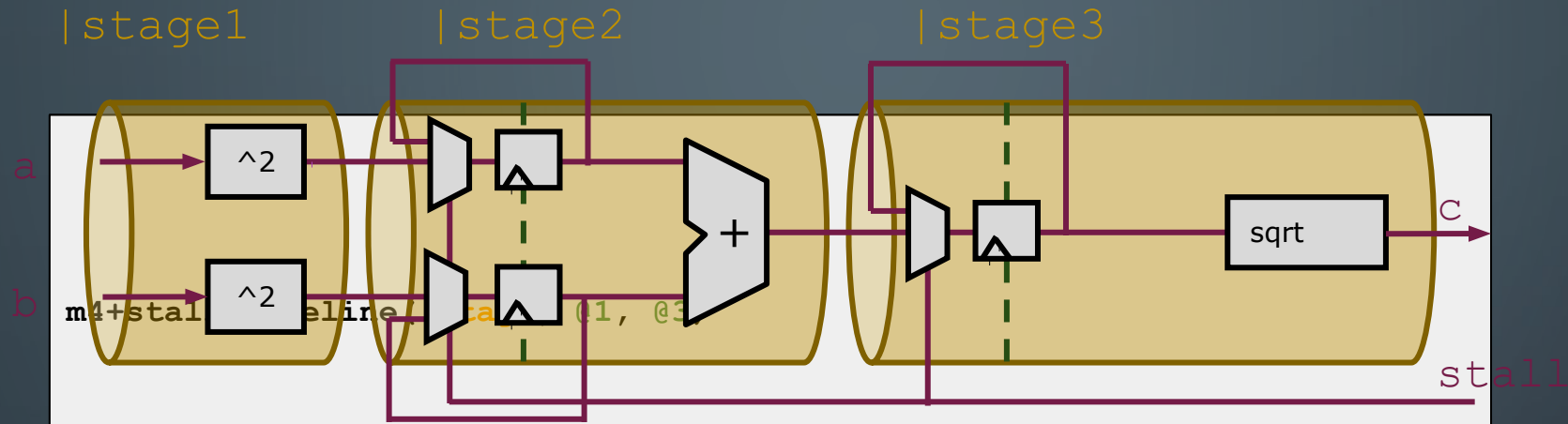
System
Verilog

```
// Calc Pipeline
logic [31:0] a_C1;
logic [31:0] b_C1;
logic [31:0] a_sq_C1,
           a_sq_C2;
logic [31:0] b_sq_C1,
           b_sq_C2;
logic [31:0] c_sq_C2,
           c_sq_C3;
logic [31:0] c_C3;
always_ff @(posedge clk) a_sq_C2 <= a_sq_C1;
always_ff @(posedge clk) b_sq_C2 <= b_sq_C1;
always_ff @(posedge clk) c_sq_C3 <= c_sq_C2;
// Stage 1
assign a_sq_C1 = a_C1 * a_C1;
assign b_sq_C1 = b_C1 * b_C1;
// Stage 2
assign c_sq_C2 = a_sq_C2 + b_sq_C2;
// Stage 3
assign c_C3 = sqrt(c_sq_C3);
```

TL-Verilog

```
|calc
@1
$aa_sq[31:0] = $aa * $aa;
$bb_sq[31:0] = $bb * $bb;
@2
$cc_sq[31:0] = $aa_sq + $bb_sq;
@3
$cc[31:0] = sqrt($cc_sq);
```

ADDING BACKPRESSURE



```

|stage1
@1
    $aa_sq[31:0] = $aa * $aa;
    $bb_sq[31:0] = $bb * $bb;
|stage2

@1
    $cc_sq[31:0] = $aa_sq + $bb_sq;
|stage3

@1
    $cc[31:0] = sqrt($cc_sq);
    
```