

Formally Verifying Many RISC-V Implementations with One Page of Code

(seriously, we did!)

Steve Hoover

Redwood EDA

Agenda

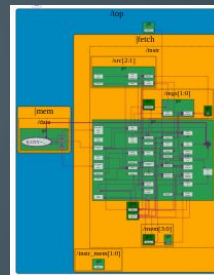
- WARP-V
 - Design Methodology
 - Comparison
- Formal Verification
 - Methodology
 - Analysis
- Wrap-Up

WARP-V

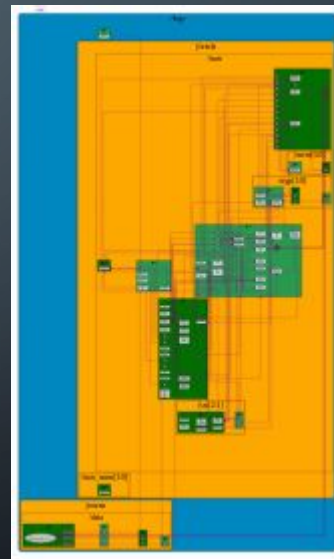
Goal: To showcase the flexibility of “transaction-level design”

IP needs to be flexible, but:

- Small conceptual variation requires extensive RTL parameterization of:
 - **staging (flip flops)**
 - stitching through hierarchy
 - clock gating/enabling
 - etc.
- SystemC+HLS is not ideal for CPUs, with tight cycle-level interactions



Low-Power
1-Stage
FPGA



Mid-Range
7-Stage
ASIC

Approach

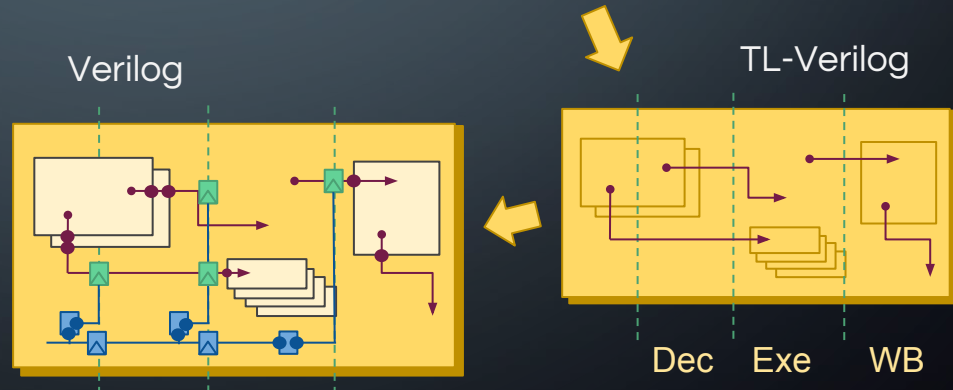
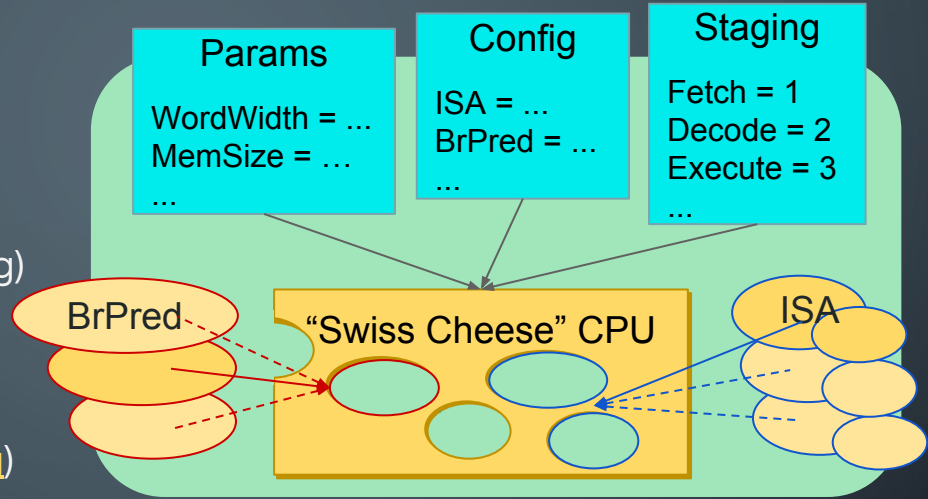
Macro Preprocessor ([M4](#)) provides:

- parameterization (incl. staging)
- component selection
- code generation

Transaction-Level Verilog ([TL-X.org](#)) implies from context:

- **staging (flip flops)**
- stitching through hierarchy
- clock gating/enabling

⇒ No need to parameterize details
(or code them at all)



Developed Online (makerchip.com)

makerchip PROJECT TUTORIALS HELP saved 4 minutes ago

EDITOR NAV-TLV LOG DIAGRAM

```
@1
Saa_sq[7:0] = $aa[3:0] ** 2;
Sbb_sq[7:0] = $bb[3:0] ** 2;
@2
$cc_sq[8:0] = $aa_sq + $bb_sq;
@3
$cc[4:0] = sqrt($cc_sq);
Last updated 10 minutes ago
```

TUTORIAL-VALID

|calc

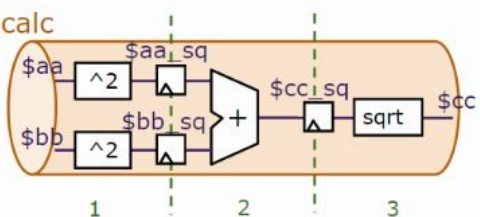
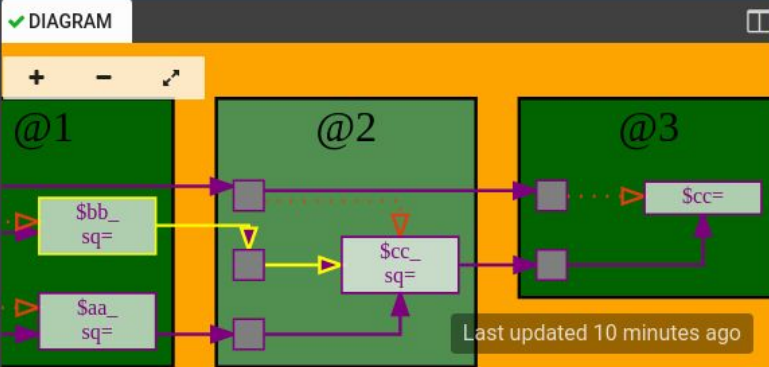


Figure 1: Pipelined Pythagorean Theorem Logic

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.

DIAGRAM



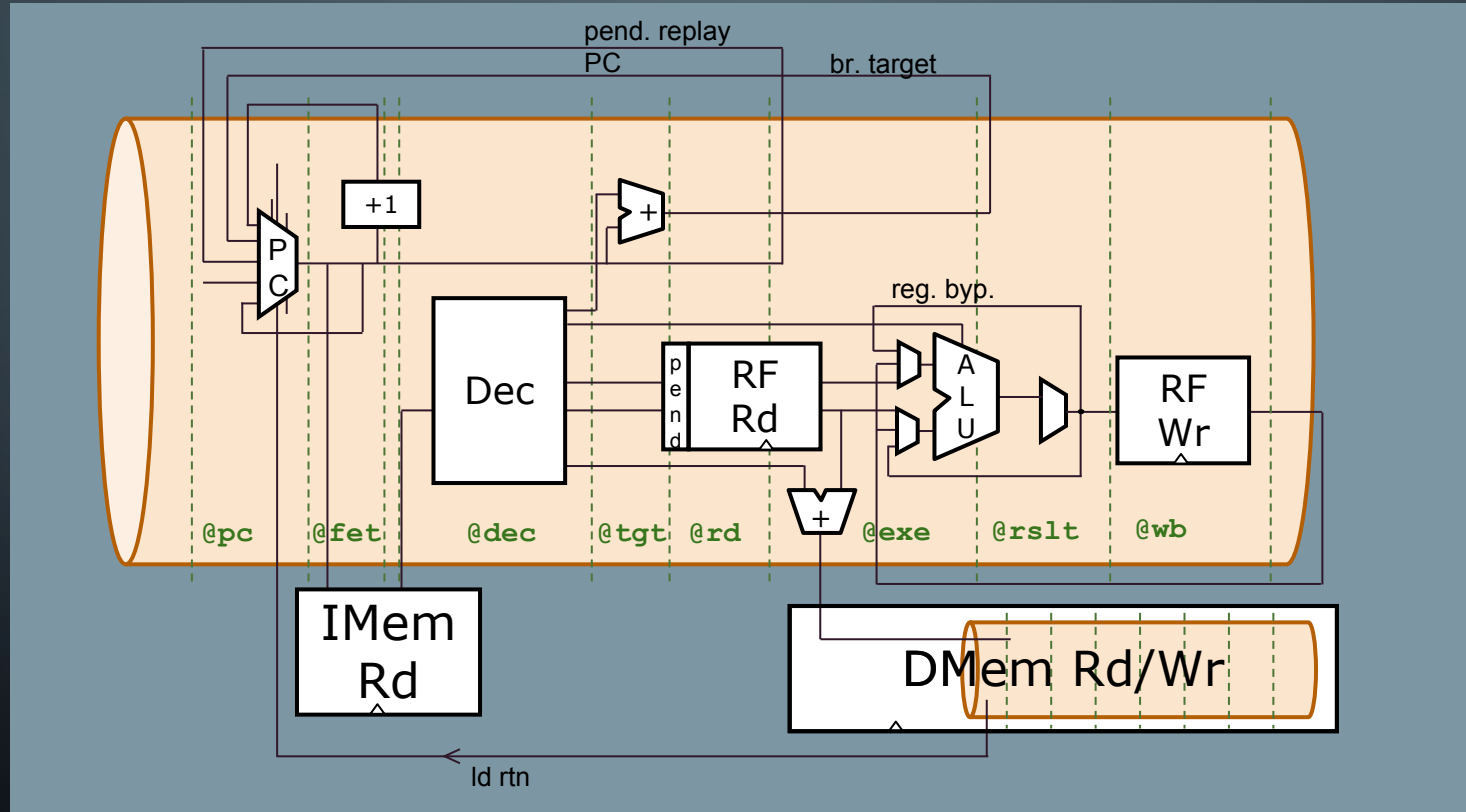
WAVEFORM

ZOOM IN ZOOM OUT ZOOM FULL << >>

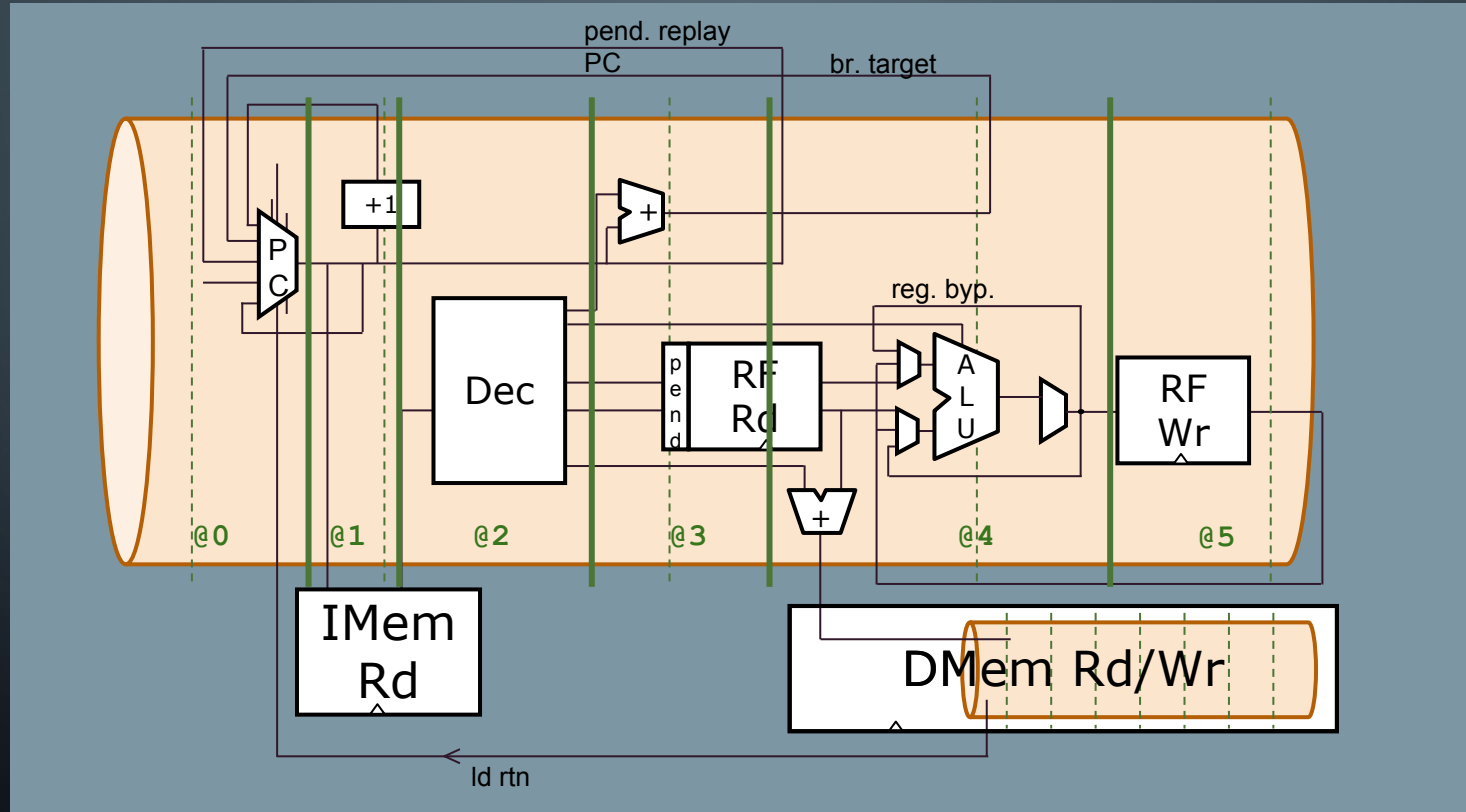
Signal	Time	Value
clk	0	1
TLV	0	TLV
calc	0	calc
@0\$aa	0	0 b 3 0 0 c 3 0 5 a 1 7 e 0
@0\$bb	0	2 5 0 0 d a 0 8 2 3 6 c 0
@1\$aa_sq	00	01 00 00 00 00 04 04
@1\$bb_sq	00	04 00 a9 a9 01 01
@2\$cc_sq	000	005 005
@3\$cc	00	02 02 0a 0a

Last updated 10 minutes ago

WARP-V



WARP-V



Comparison

Logic comparison of main CPU pipeline (as apples-to-apples as possible)

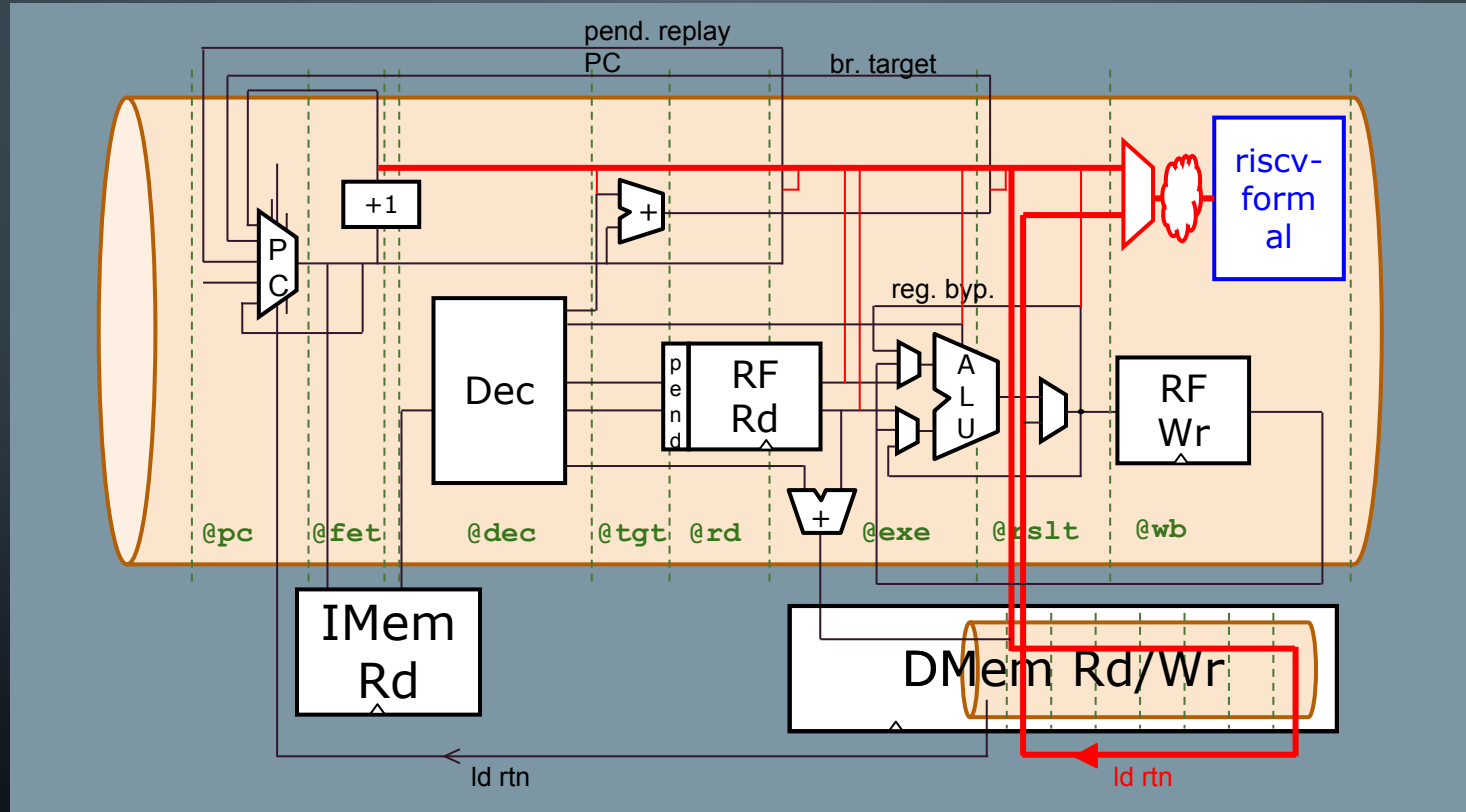
	picorv32	Rocket	WARP-V
Language	Verilog	Chisel	TL-Verilog
Construct. Language	Verilog preproc.	Scala	M4 (plus Perl)
Core pipeline	~5 stages (unpipelined)	5 stages	1-7 stages
Lines of code	~983	~944	~811

- All three express RTL detail
- WARP-V in TL-Verilog is slightly smaller and much more flexible

Verification Methodology

- First demonstration of TL-Verilog for verification modeling.
- WARP-V brought to life in 1.5 wks. using an 11-instruction test program w/ assembler and test program also in M4+TL-Verilog. (Jan 2018)
- Remaining verification done by Ákos Hadnagy in Google Summer of Code 2018
- Uses open-source formal verification tools: RISC-V-Formal by Clifford Wolf

Verification Modeling

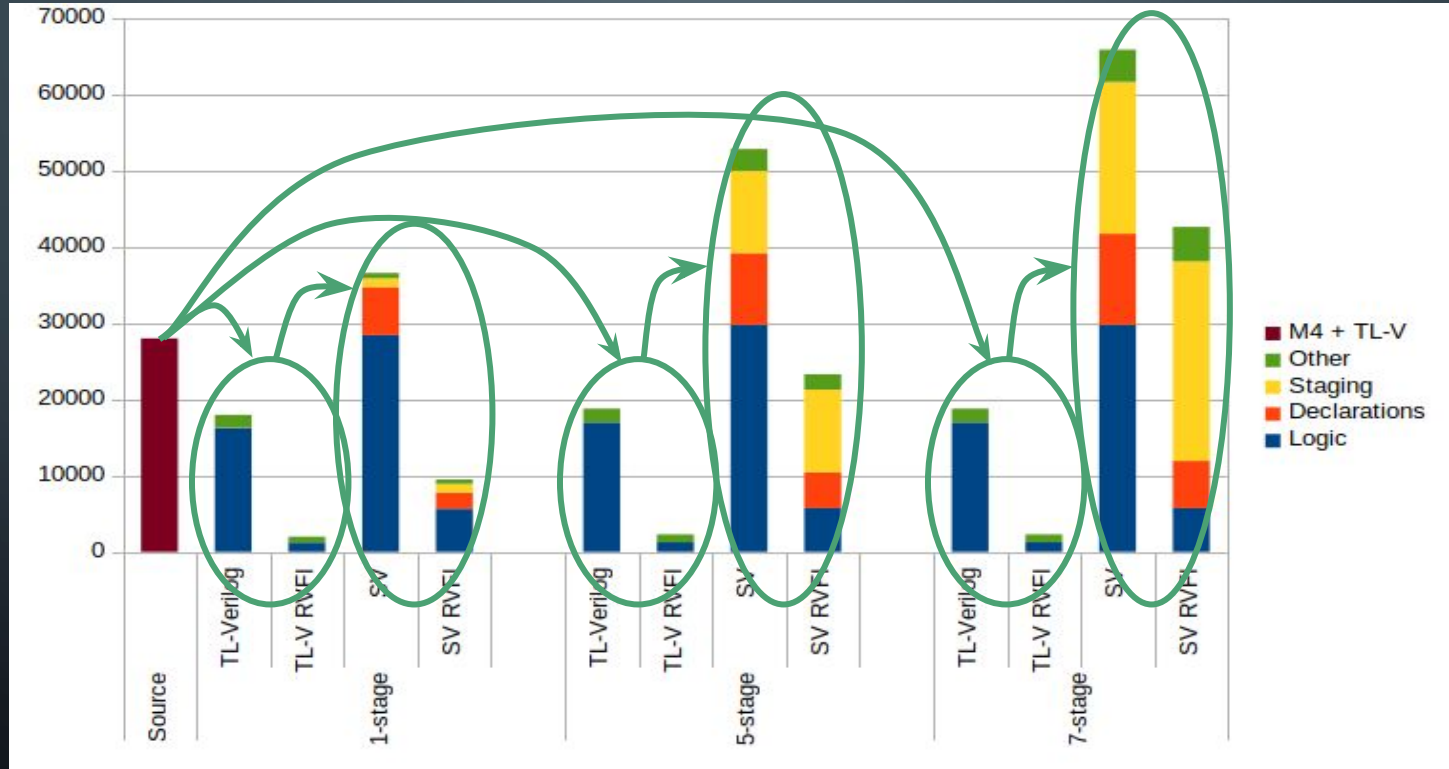


As Promised, 1 Page

```
m4_ifelse_block(M4_FORMAL, [''], ['
$pc[M4_PC_RANGE] = $pc[M4_PC_RANGE]; // A version of PC we can pull through $ANYs.
// This scope is a copy of /instr or /instr/original_ld if $returning_ld.
/original
  $ANY = /instr$returning_ld ? /instr/original_ld$ANY : /instr$ANY;
  /src[2:1]
  $ANY = /instr$returning_ld ? /instr/original_ld/src$ANY : /instr/src$ANY;

// RVFI interface for formal verification.
$trap = $aborting_trap ||
  $non_aborting_trap;
$rvfi_trap = ! $reset && >>M4_eval(-M4_MAX_REDIRECT_BUBBLES + 1)$next_rvfi_good_path_mask[M4_MAX_REDIRECT_BUBBLES] &&
  $trap && ! $replay && ! $returning_ld; // Good-path trap, not aborted for other reasons.
// Order for the instruction/trap for RVFI check. (For ld, this is associated with the ld itself, not the returning_ld.)
$rvfi_order[63:0] = $reset
  ? 64'b0 :
  ($commit || $rvfi_trap) ? >>1$rvfi_order + 64'b1 :
  $RETAIN;
$rvfi_valid = ! <<m4_eval(M4_REG_WR_STAGE - (M4_NEXT_PC_STAGE - 1))$reset && // Avoid asserting before $reset propagates
  (($commit && ! $ld) || $rvfi_trap || $returning_ld);
*rvfi_valid = $rvfi_valid;
*rvfi_insn = /original$raw;
*rvfi_halt = $rvfi_trap;
*rvfi_trap = $rvfi_trap;
*rvfi_order = /original$rvfi_order;
*rvfi_intr = 1'b0;
*rvfi_rs1_addr = /original/src[1]$is_reg ? /original$raw_rs1 : 5'b0;
*rvfi_rs2_addr = /original/src[2]$is_reg ? /original$raw_rs2 : 5'b0;
*rvfi_rs1_rdata = /original/src[1]$is_reg ? /original/src[1]$reg_value : M4_WORD_CNT'b0;
*rvfi_rs2_rdata = /original/src[2]$is_reg ? /original/src[2]$reg_value : M4_WORD_CNT'b0;
*rvfi_rd_addr = (/original$dest_reg_valid && ! $abort) ? /original$raw_rd : 5'b0;
*rvfi_rd_wdata = *rvfi_rd_addr ? $slt : 32'b0;
*rvfi_pc_rdata = (/original$pc[31:2], 2'b00);
*rvfi_pc_wdata = {$reset ? M4_PC_CNT'b0 :
  $returning_ld ? /original_ld$pc + 1'b1 :
  $trap ? $trap_target :
  $jump ? $jump_target :
  $mispred_branch ? ($staken ? $branch_target[M4_PC_RANGE] : $pc + M4_PC_CNT'b1) :
  m4_ifelse(M4_BRANCH_PRED, ['fallthrough'], [''], ['$pred_taken_branch ? $branch_target[M4_PC_RANGE] :'])
  $indirect_jump ? $indirect_jump_target :
  $pc[31:2] + 1'b1, 2'b00);
*rvfi_mem_addr = (/original$ld || $valid_st) ? /original$addr[M4_ADDR_MAX:2], 2'b0 : 0;
*rvfi_mem_rmask = /original$ld ? /original_ld$ld_mask : 0;
*rvfi_mem_wmask = $valid_st ? $st_mask : 0;
*rvfi_mem_rdata = /original$ld ? /original_ld$ld_value : 0;
*rvfi_mem_wdata = $valid_st ? $st_value : 0;
```

Code Size Comparison



Wrap-Up

- Methodology implications
 - **Single, small codebase provides logic and verification of flexible IP**
 - We focused on verifying the 5-stage version, others just worked (mostly)
- Futures
 - Many-core
 - Opportunities again in Google Summer of Code 2019 (ask me)

Other Things I Love to Discuss Lately

- Why open-source hardware is poised to explode
- Cloud FPGAs and hardware-accelerated web applications
- Start-up life
- Internship/co-op opportunities