

Overcoming RTL

The Most-Adaptable Open-Source RISC-V Core

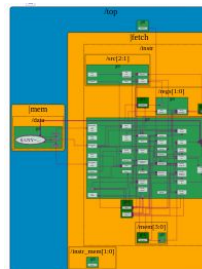
Steve Hoover
Redwood EDA
DAC 2018

Motivation

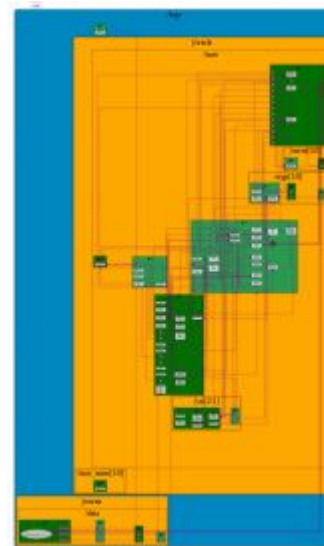
We wanted to build the world's **most flexible** CPU core IP, but...

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

I present the tools and methodology we used to solve these problems.



Low-Power
1-Stage
FPGA






High-Perf
7-Stage
ASIC

Approach

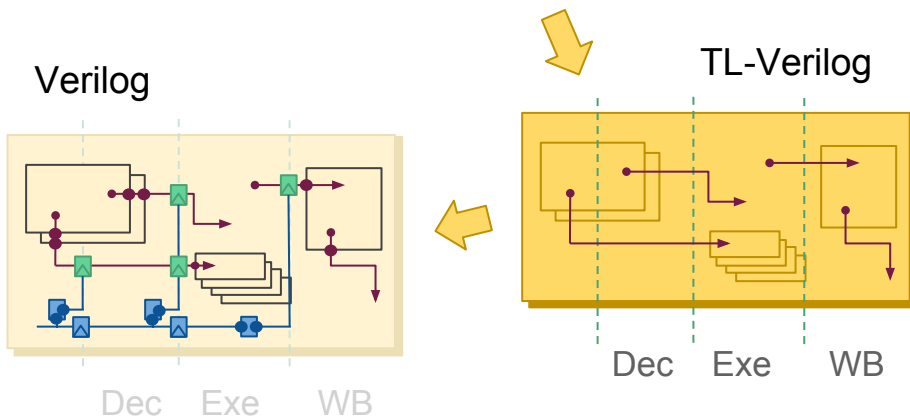
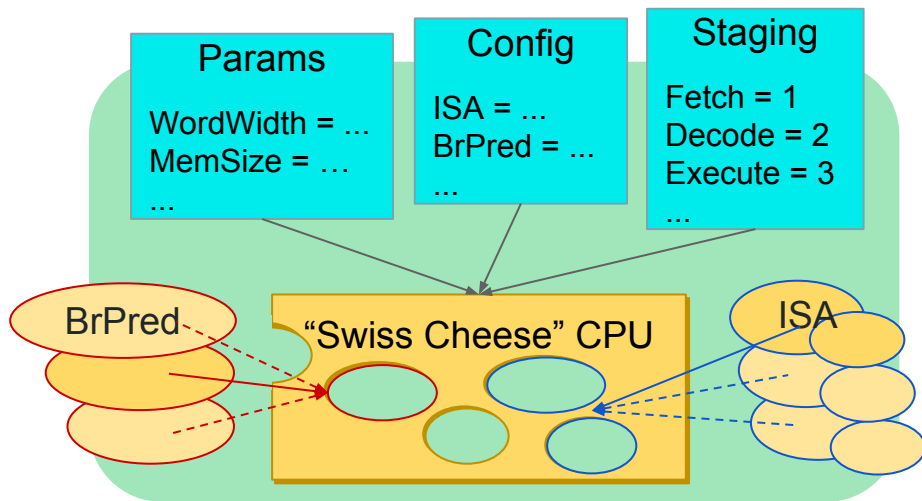
Macro Preprocessor ([M4](#)) provides:

- 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).

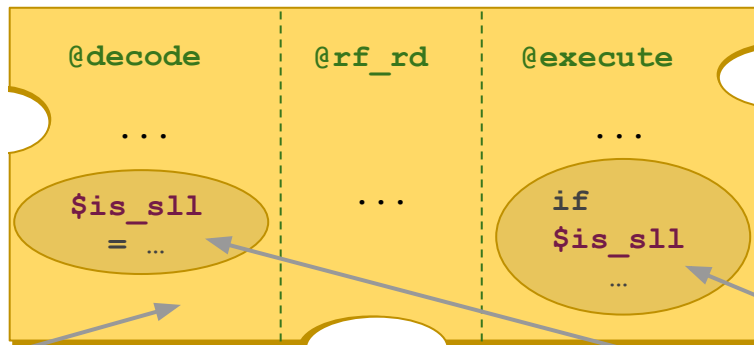


For Example

RISC-V has a `sll` instruction.

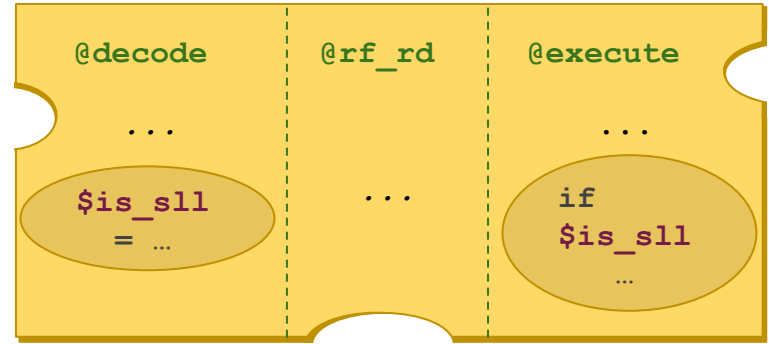
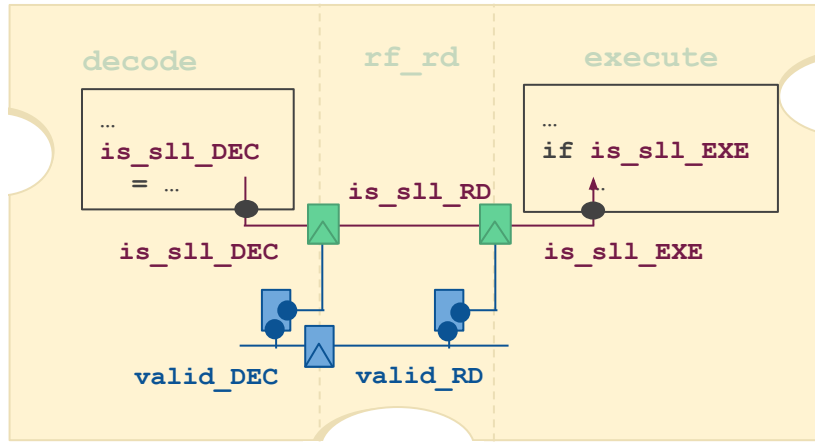
| `cpu` (pipeline)




```
\TLV cheese()  
|cpu  
@decode  
...  
@rf_rd  
...  
@execute  
...
```



```
\TLV riscv()  
|cpu  
@decode  
  $is_sll = ...  
@execute  
  if $is_sll  
...
```

Inferring Staging/Interfaces/Gating

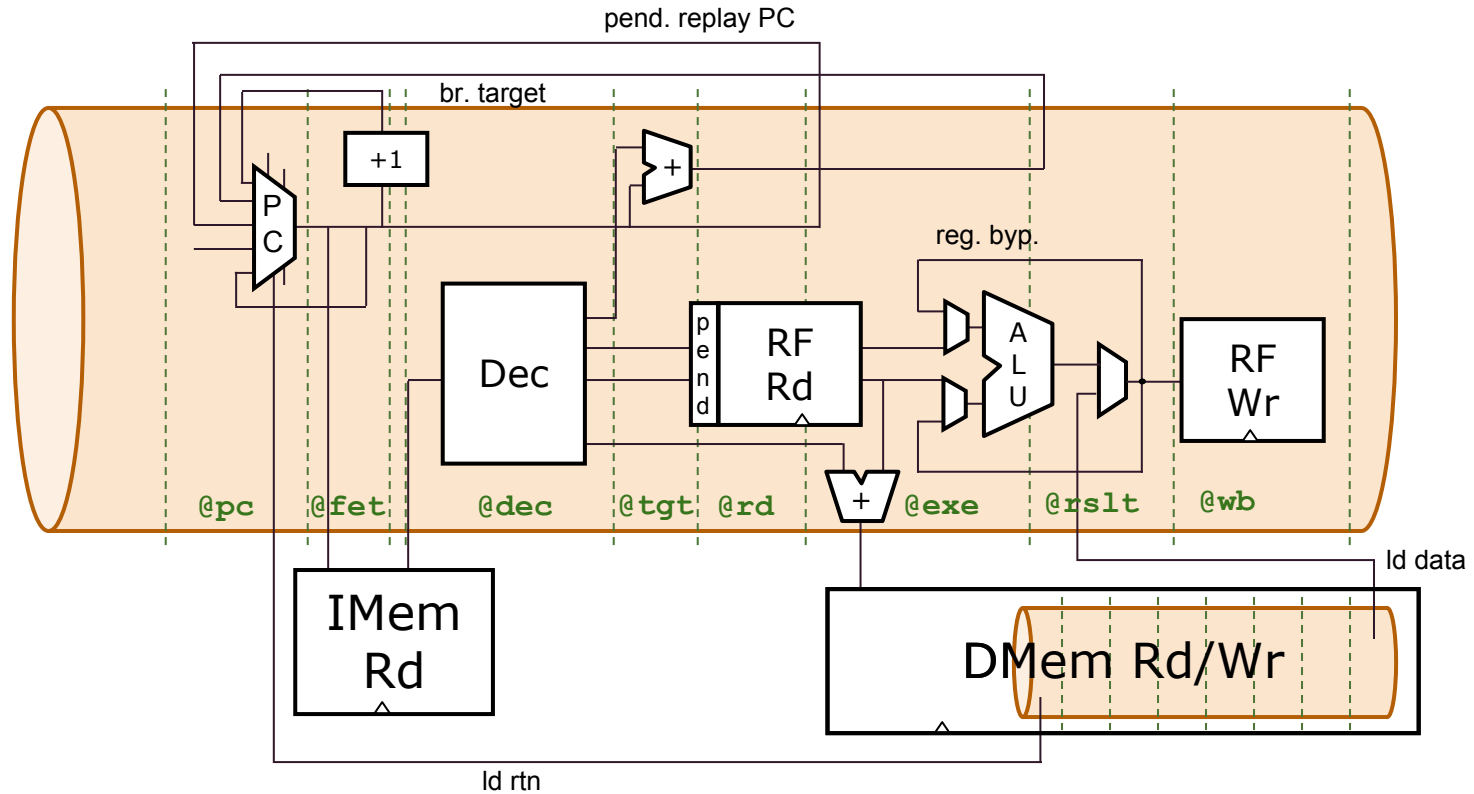


-  Staging is implied from pipeline context.
-  TL-Verilog signal references reach through hierarchy.
-  Fine-grained clock gating is implied from knowledge of validity. (Gate if no instruction.)

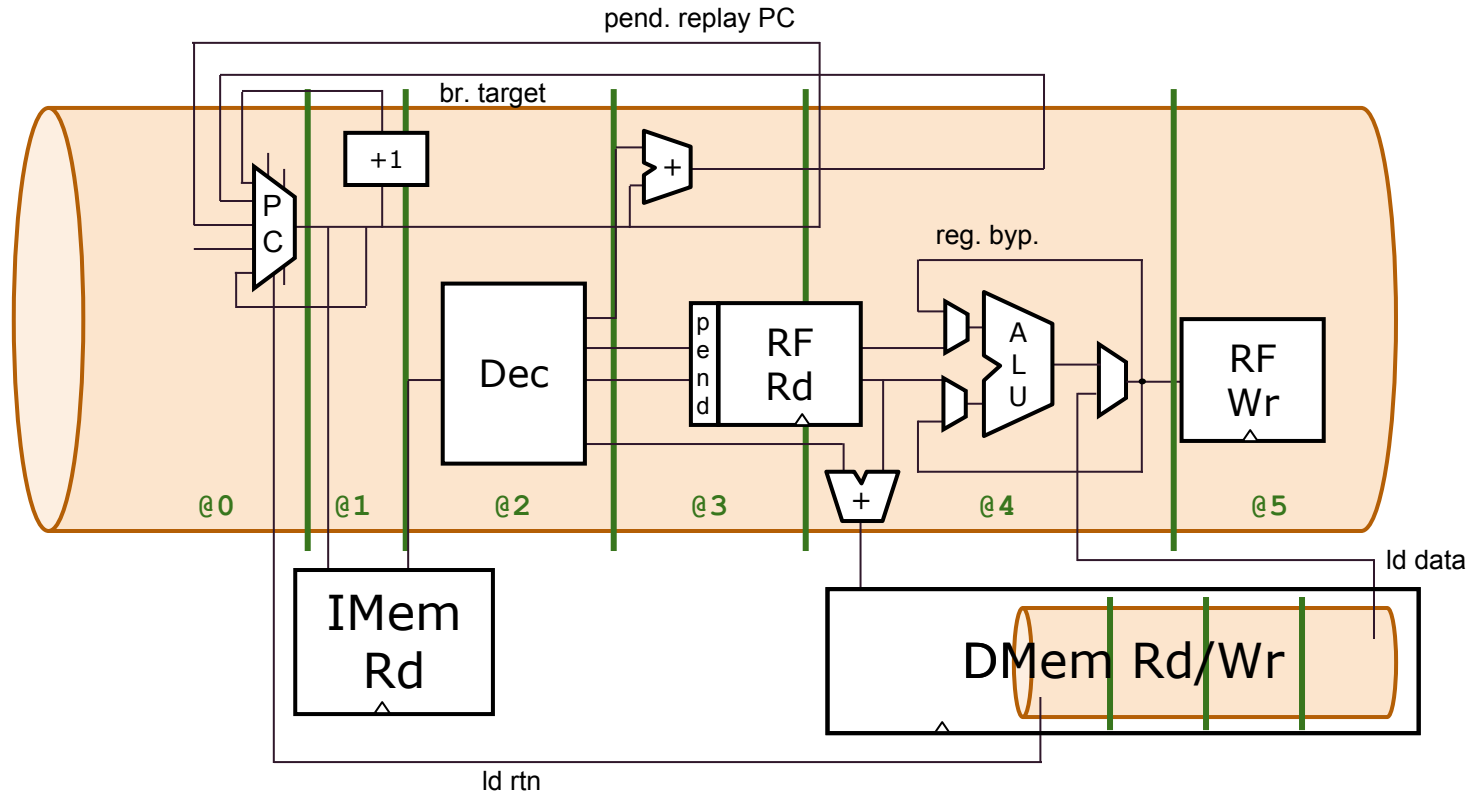
⇒ No staging/interfaces/gating to parameterize (or code)

⇒ No impact to the “cheese”

WARP-V



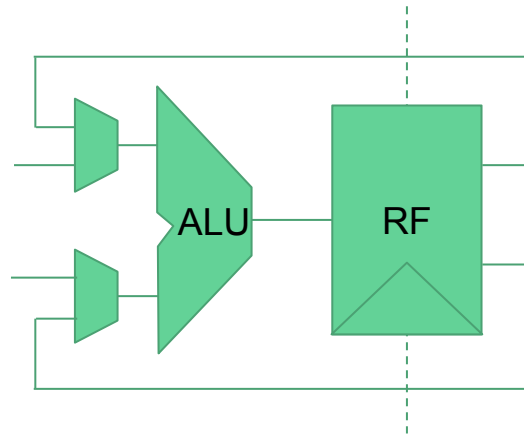
6-Stage WARP-V



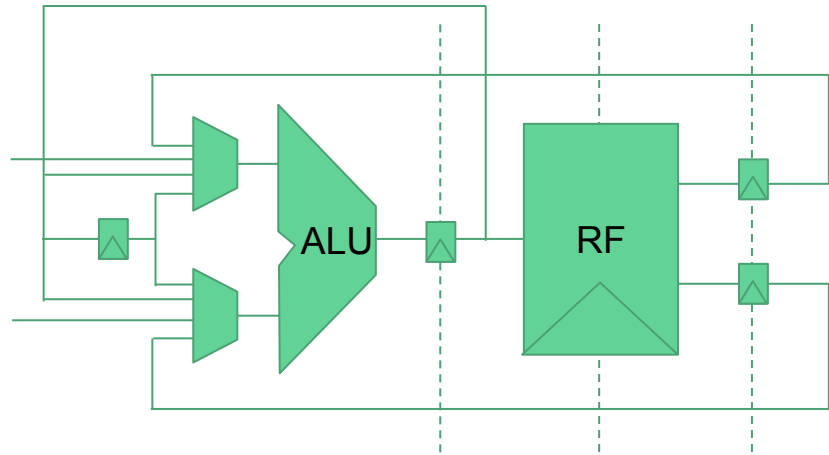
Staging-Dependent Logic

Feedback/feed-forward logic depends on staging. (This is where HLS is awkward.)
E.g. mispredicts, replays, register bypass, etc.

Register bypass:



1-Cycle



3-Cycle: ALU, wr, rd

Register Bypass MUX Code

0, 1, 2, or 3 bypass cycles (based on M4_REG_BYPASS_STAGES)

```
$reg_value[M4_WORD_RANGE] =  
    // Bypass stages:  
    m4_ifexpr(M4_REG_BYPASS_STAGES >= 1, ['(/instr>>1$dest_reg_valid && (/instr>>1$dest_reg == $reg)) ? /instr>>1$rslt :'])  
    m4_ifexpr(M4_REG_BYPASS_STAGES >= 2, ['(/instr>>2$dest_reg_valid && (/instr>>2$dest_reg == $reg)) ? /instr>>2$rslt :'])  
    m4_ifexpr(M4_REG_BYPASS_STAGES >= 3, ['(/instr>>3$dest_reg_valid && (/instr>>3$dest_reg == $reg)) ? /instr>>3$rslt :'])  
    /instr/regs[$reg]>>M4_REG_BYPASS_STAGES$value;
```

Results

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.
- Flexible pipeline overhead: **3.1%** (vs. infeasible)
- ISA swapping overhead: **4.4%** (vs. impractical)

More Info

- makerchip.com
 - WARP-V is under "Examples"
 - TL-Verilog tutorials and docs



- Ask me:
 - Email: steve.hoover@redwoodeda.com
 - Poster Session: @ 5pm on Level 2 Exhibit Floor.

The poster is titled "Overcoming RTL" and subtitled "The Most-Adaptable Open-Source RISC-V Core". It features a background of a colorful, abstract pattern. The main content is organized into four quadrants: 1. "RTL Construction" shows a flow from "Params" (WordWidth, MemSize), "Config" (ISA, BrPred), and "Staging" (Fetch, Decode, Execute) to a "Swiss Cheese CPU" diagram. 2. "Flexibility" shows two hardware images: a blue FPGA labeled "Low-Power, Low Freq., 1-yc FPGA" and a green ASIC labeled "High Freq., 7-yc ASIC". 3. "WARP-V uArch" shows a block diagram of a processor core with components like "Cache", "CPU", and "Open RISC-V". 4. "Built in the Cloud" shows a screenshot of a web-based interface for configuring and simulating the core. At the bottom right, it says "Steve Hoover, Redwood EDA DAC 2018".

Poster