



Instrumenting and Debugging FireSim-Simulated Designs

<https://fires.im>

 @firesimproject

MICRO 2024 Tutorial

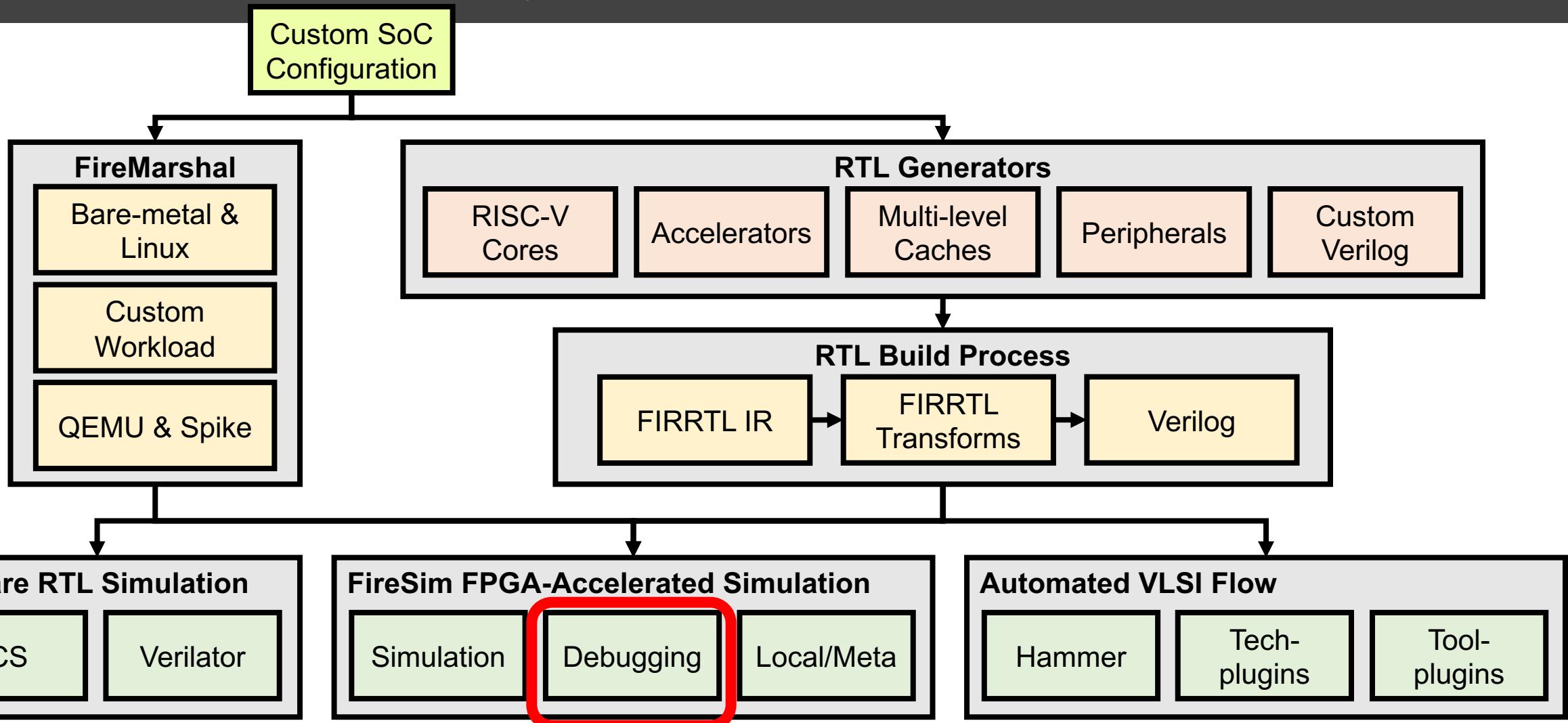
Speaker: Abe Gonzalez



Berkeley Architecture Research



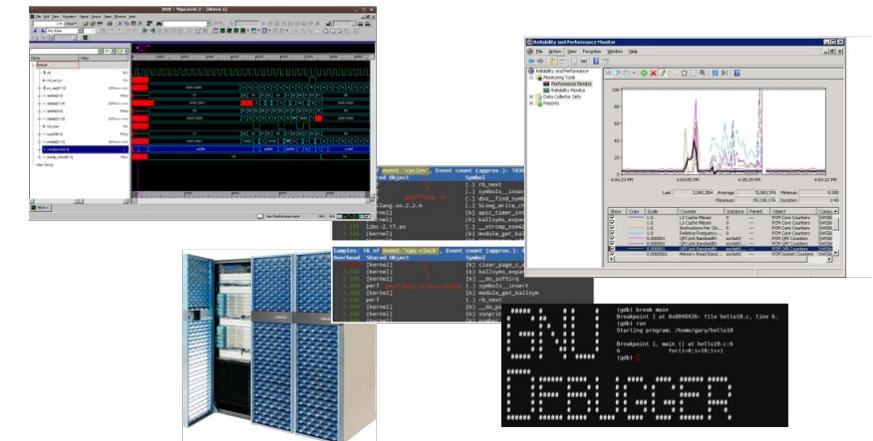
Tutorial Roadmap





Agenda

- FPGA-Accelerated Deep-Simulation Debugging
 - Debugging Using Integrated Logic Analyzers
 - Trace-based Debugging
 - Synthesizable Assertions/Prints
 - Synthesizable Counters
 - Hands-on example
- Debugging Co-Simulation
 - FireSim Debugging Using Software Simulation

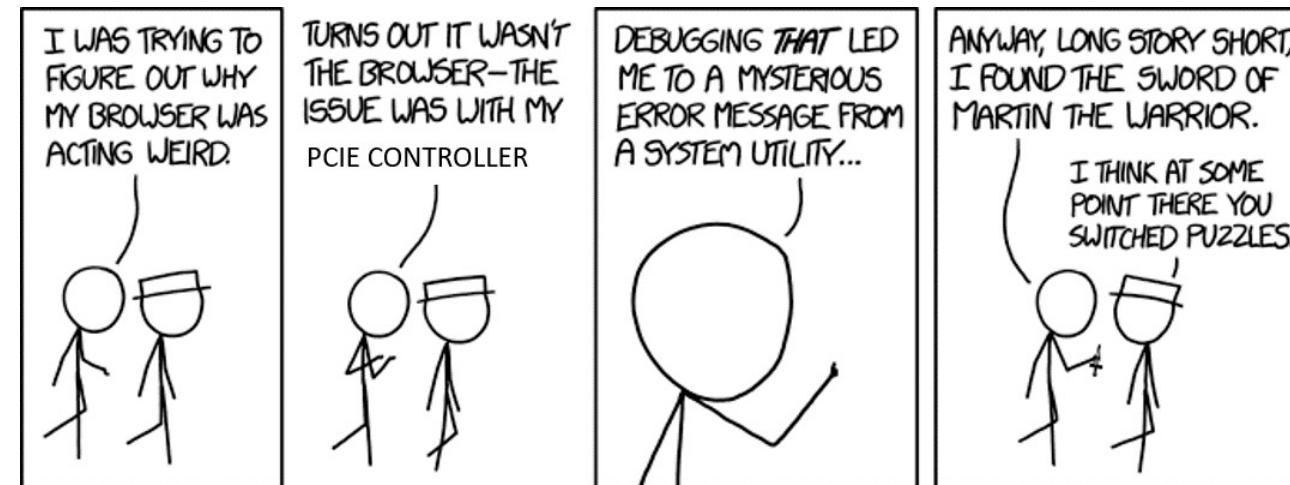




When SW RTL Simulation is Not Enough...

“Everything looks OK in SW simulation, but there is still a bug somewhere”

“My bug only appears after hours of running Linux on my simulated HW”





FPGA-Based Debugging Features

- High simulation speed in FPGA-based simulation enables advanced debugging and profiling tools.
- Reach “deep” in simulation time, and obtain large levels of coverage and data
- Examples:
 - ILAs
 - TracerV
 - AutoCounter
 - Synthesizable assertions, prints



SW
Simulation



FPGA-based
Simulation

Simulated
Time





Debugging Using Integrated Logic Analyzers

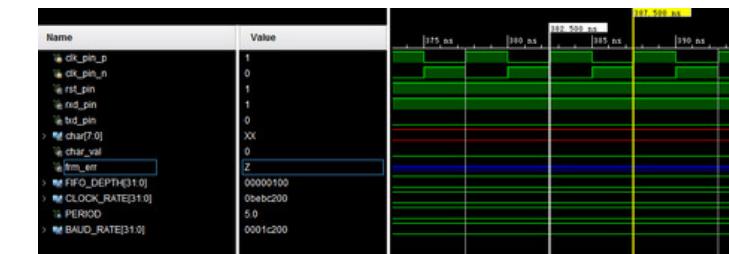
From: aws-fpga cl_hello_world example

Integrated Logic Analyzers (ILAs)

- Common debugging feature provided by FPGA vendors
- Continuous recording of a sampling window
 - Up to 1024 cycles by default.
 - Stores recorded samples in BRAM.
- Realtime trigger-based sampled output of probed signals
 - Multiple probes ports can be combined to a single trigger
 - Trigger can be in any location within the sampling window
- On the AWS F1-Instances, ILA interfaced through a debug-bridge and server

```
// Integrated Logic Analyzers (ILA)
ila_0 CL_ILA_0 (
    .clk      (clk_main_a0),
    .probe0  (sh_ocl_awvalid_q),
    .probe1  (sh_ocl_awaddr_q ),
    .probe2  (ocl_sh_awready_q),
    .probe3  (sh_ocl_arvalid_q),
    .probe4  (sh_ocl_araddr_q ),
    .probe5  (ocl_sh_arready_q)
);

// Debug Bridge
cl_debug_bridge CL_DEBUG_BRIDGE (
    .clk(clk_main_a0),
    .S_BSCAN_drck(drck),
    .S_BSCAN_shift(shift),
    .S_BSCAN_tdi(tdi),
    .S_BSCAN_update(update),
    .S_BSCAN_sel(sel),
    .S_BSCAN_tdo(tdo),
    .S_BSCAN_tms(tms),
    .S_BSCAN_tck(tck),
    .S_BSCAN_runtest(runtest),
    .S_BSCAN_reset(reset),
    .S_BSCAN_capture(capture),
    .S_BSCAN_bscanid_en(bscanid_en)
);
```





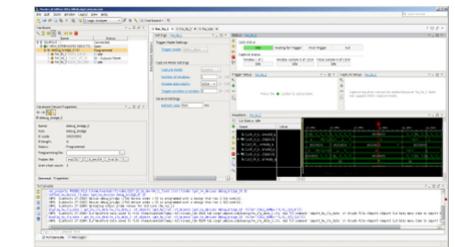
Debugging Using Integrated Logic Analyzers

AutoILA – Automation of ILA integration with FireSim

- Annotate requested signals and bundles in the Chisel source code
- Automatic configuration and generation of the ILA IP in the FPGA toolchain
- Automatic expansion and wiring of annotated signals to the top level of a design using a FIRRTL transform.
- Remote waveform and trigger setup from the manager instance

```
import midas.targetutils.FpgaDebug

class SomeModuleIO(implicit p: Parameters) extends SomeIO()(p){
    val out1 = Output(Bool())
    val in1 = Input(Bool())
    FpgaDebug(out1, in1)
}
```





BOOM Example

- Debugging an out-of-order processor is hard
 - Throughout this talk, we'll have examples of FPGA debugging used in BOOM.
- Example from `boom/src/main/scala/l1su/dcache.scala`
- Debugging a non-blocking data cache hanging after Linux boots

```
class BoomNonBlockingDCacheModule(outer: BoomNonBlockingDCache) extends LazyModuleImp(outer)
    with HasL1HellaCacheParameters
{
    implicit val edge = outer.node.edges.out(0)
    val (tl_out, _) = outer.node.out(0)
    val io = IO(new BoomDCacheBundle)

    FpgaDebug(tl_out)
    FpgaDebug(io.req)
    FpgaDebug(io.resp)
    FpgaDebug(io.s1_kill)
    FpgaDebug(io.nack)
    ...
}
```





Debugging using Integrated Logic Analyzers

Pros:

- No emulated parts – what you see is what's running on the FPGA
- FPGA simulation speed - $O(\text{MHz})$ compared to $O(\text{KHz})$ in software simulation
- Real-time trigger-based

Cons:

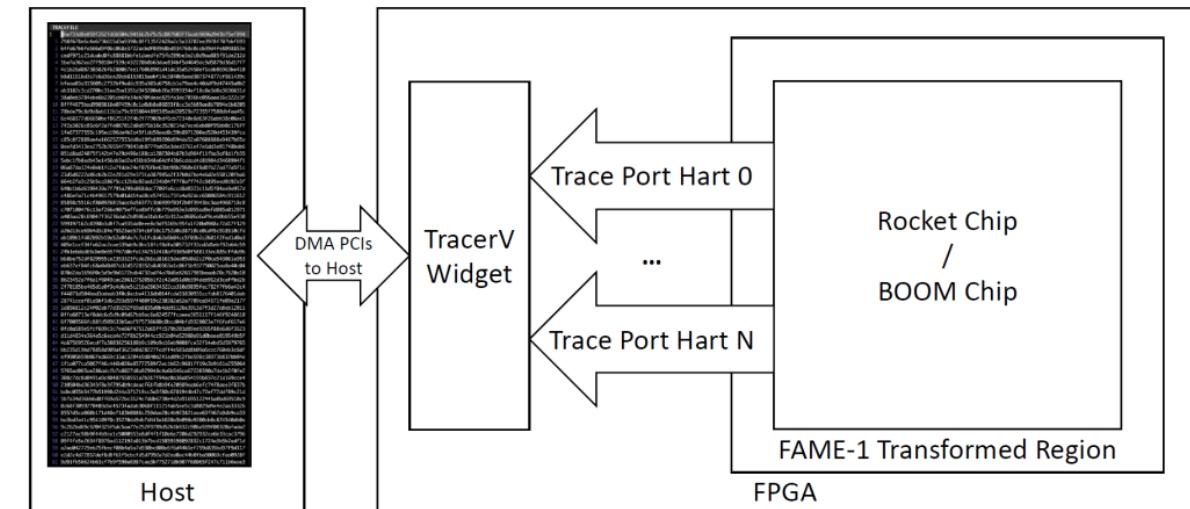
- Requires a full build to modify visible signals/triggers (takes several hours)
- Limited sampling window size
- Consumes FPGA resources





TracerV

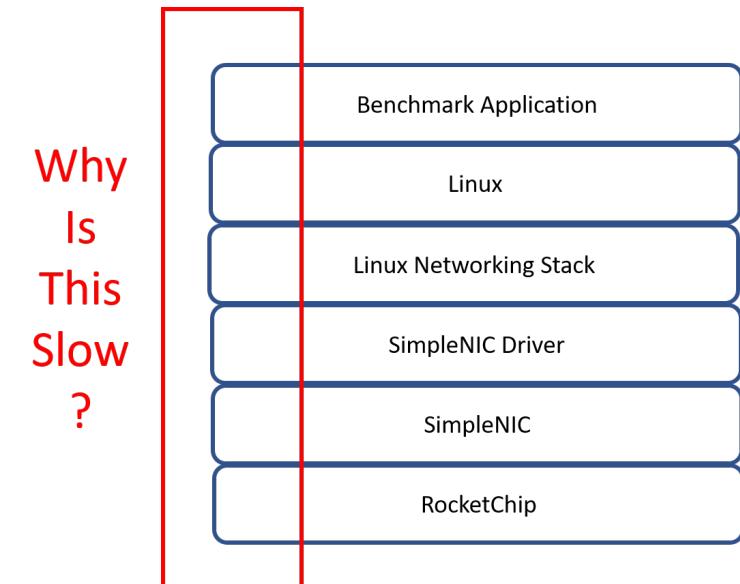
- **Out-of-band** full instruction execution trace
- Bridge connected to target trace ports
- By default, large amount of info wired out of Rocket/BOOM, per-hart, per-cycle:
 - Instruction Address
 - Instruction
 - Privilege Level
 - Exception/Interrupt Status, Cause
- TracerV can rapidly generate several TB of data.





TracerV

- Out-of-Band: profiling does not perturb execution
- Useful for kernel and hypervisor level cycle-sensitive profiling
- Examples:
 - Co-Optimization of NIC and Network Driver
 - Keystone Secure Enclave Project
 - High-performance hardware-specific code (supercomputing?)
- Requires large-scale analytics for insightful profiling and optimization.





Trigger Mechanisms

- Full trace files can be very large (100s GB – TB)
- We are usually interested only in a specific region of execution
- TracerV can be enabled based on in-band and out-of-band triggers
 - Program counter
 - Unique instruction
 - Cycle count
- Can use the same trigger for some other simulation outputs
 - AutoCounter perf. counters

config_runtime.yaml

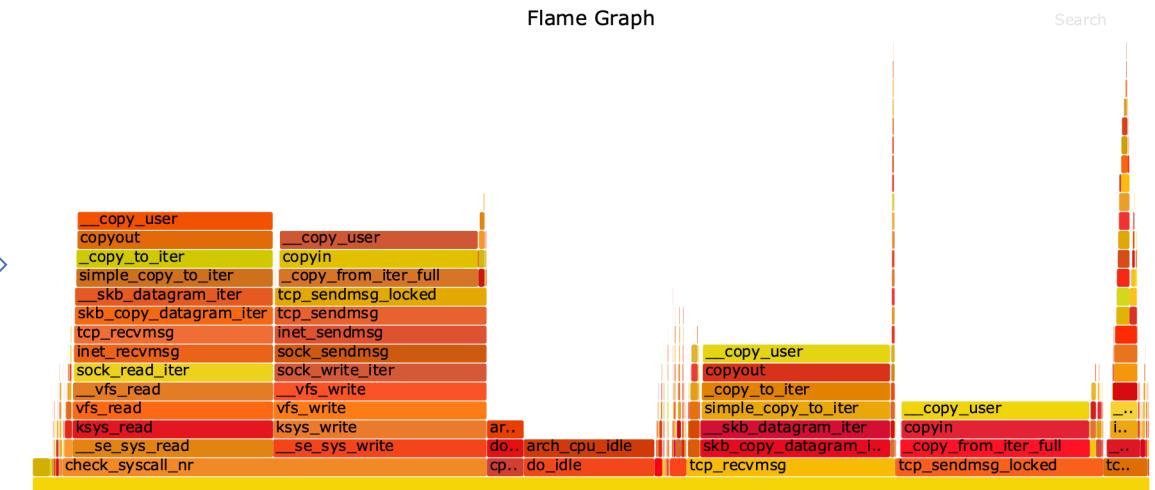
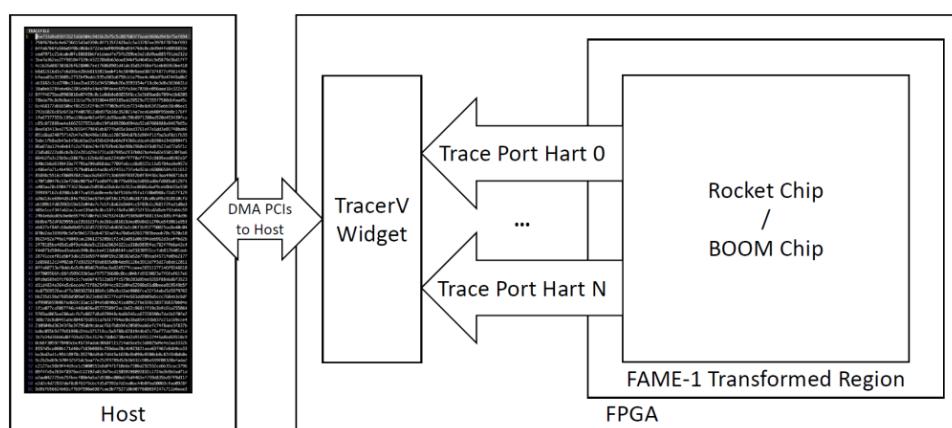
```
tracing:  
  enable: no  
  #0 = no trigger  
  #1 = cycle count trigger  
  #2 = program counter trigger  
  #3 = instruction trigger  
  selector: 1  
  startcycle: 0  
  endcycle: -1
```





Integration with Flame Graphs

- Flame Graph – Open-source profiling visualization tool
- Direct integration with TracerV traces
 - Automated stack unwinding (kernel space)
 - Automated Flame-graph generation





TracerV

Pros:

- Out-of-Band (no impact on workload execution)
- SW-centric method
- Large amounts of data

Cons:

- Slower simulation performance (40 MHz)
- No HW visibility
- Large amounts of data

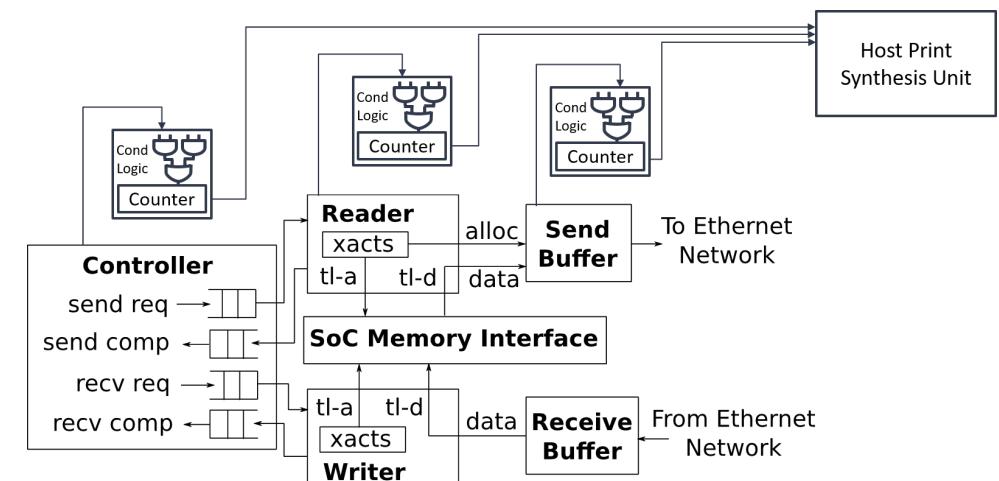




AutoCounter

- Automated out-of-band counter insertion
- Based on ad-hoc annotations and existing cover-points
 - No invasive RTL change
- Runtime-configure read rate

```
253 io.send.req.ready := state === s_idle
254 io.alloc.valid := helper.fire(io.alloc.ready, canSend)
255 io.alloc.bits.id := xactId
256 io.alloc.bits.count := (1.U << (reqSize - byteAddrBits.U))
257 tl.a.valid := helper.fire(tl.a.ready, canSend)
258 tl.a.bits := edge.Get(
  fromSource = xactId,
  toAddress = sendaddr,
  lgSize = reqSize)._2
262
263 cover((state === s_read) && xactBusy.andR && tl.a.ready, "NIC_SEND_XACT_ALL_BUSY", "nic send blocked by lack of transactions")
264 cover((state === s_read) && !io.alloc.ready && tl.a.ready, "NIC_SEND_BUF_FULL", "nic send blocked by full buffer")
265 cover(tl.a.valid && !tl.a.ready , "NIC_SEND_MEM_BUSY", "nic send blocked by memory bandwidth")
```





AutoCounter Example

- Example ad-hoc performance counters in the L2 cache

```
class SinkA(params: InclusiveCacheParameters) extends Module
{
    val io = new Bundle {
        val req = Decoupled(new FullRequest(params))
        val a = Decoupled(new TLBundleA(params.inner.bundle)).flip
        val pb_pop = Decoupled(new PutBufferPop(params)).flip
        val pb_beat = new PutBufferAEntry(params)
    }
    PerfCounter(io.a.fire(), "l2_requests", "Number of requests to the first bank of the L2");
}
```

- Simple configuration (`config_runtime.yaml`)
 - Readrate - Trade-off visibility/detail and performance
 - TracerV trigger - Collect results from singular point of interest

```
autocounter:
    read_rate: 1000000
```





AutoCounter Output CSV Schema

Version	Version Number				
Clock Domain Name	Domain Name	Multiplier	X	Divisor	Y
Labels	local_clock	Label0	Label1
Description	local clock cycle	Desc0	Desc1
Event Width	1	Width0	Width1
Accumulator Width	64	64	64
Type	Increment	Type0	Type1
N	Cycle @ time N	Value0 @ time N	Value1 @ time N
...
kN	Cycle @ time kN	Value0 @ time kN	Value1 @ time kN





AutoCounter Output CSV Schema

Version	Version Number				
Clock Domain Name	Domain Name	Multiplier	X	Divisor	Y
Labels	local_clock	Label0	Label1
Description	local clock cycle	Desc0	Desc1
Event Width	1	Width0	More counters		
Accumulator Width	64	64			
Type	Increment	Type0			
N	Cycle @ time N	Value0 @			
...
kN	Cycle @ time kN	Value0 @ time kN	Value1 @ time kN





AutoCounter Output CSV Schema

Version	Version Number					
Clock Domain Name	Domain Name	Multiplier	X	Divisor	Y	
Labels	local_clock	Label0	Label1	
Description	local clock cycle	Desc0	Desc1	
Event Width	1	Width0	Width1	
Accumulator Width	64	64	64	...	More samples	
Type	Increment	Type0	Type1	...		
N	Cycle @ time N	Value0 @ time N	Value1 @ time N	
...	
kN	Cycle @ time kN	Value0 @ time kN	Value1 @ time kN	





Automated Performance Counters

Pros:

- Macro view of execution behavior
- Trigger integration
- Pre-configured cover points, no RTL interference
- SW-controlled granularity (tradeoff simulation for read rate)

Cons:

- New counters require new FPGA images
- Simulation performance degradation depending on read rate and number of counters





Synthesizable Assertions

- Assertions – rapid error checking embedded in HW source code.
 - Commonly used in SW Simulation
 - Halts the simulation upon a triggered assertion. Represented as a “stop” statement in FIRRTL
 - By default, emitted as non-synthesizable SV functions (`$fatal`)

BROOM
An open-source out-of-order processor with resilient low-voltage operation in 28nm CMOS.

Christopher Celio, Pi-Feng Chiu, Krste Asanović, David Patterson, and Borivoje Nikolic
Hot Chips 2018

RISC-V ASPIRE UC Berkeley

Verification

- Directed tests and a randomized torture generator.
- Verilator/VCS/FPGA simulation at RTL.
- VCS for post-gl/par simulation.
- Speculative OOO pipelines are difficult to get good coverage on.
 - Need tests that build up a lot of speculative state.
 - Need tests that cover OoO- and platform-level use-cases.
- Assertions are king.

21

From: BROOM: An open-source Out-of-Order processor with resilient low-voltage operation in 28nm CMOS, Christopher Celio, Pi-Feng Chiu, Krste Asanovic, David Patterson and Borivoje Nikolic. HotChip 30, 2018

```
class Count extends Module {
    val io = IO(new Bundle {
        val en = Input(Bool())
        val done = Output(Bool())
        val cntr = Output(UInt(4.W))
    })
    // count until 10 when `io.en` is high
    val (cntr, done) = Counter(io.en, 10)
    io.cntr := cntr
    io.done := done

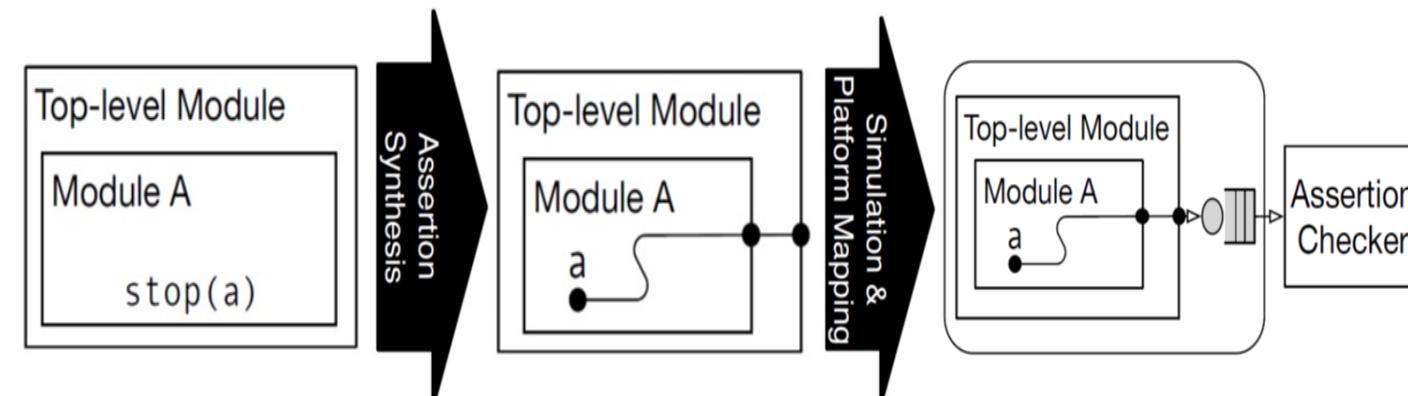
    // assertion for software simulation
    // `cntr` should be less than 10
    assert(cntr < 10.U)
}
```

From: Trillion-Cycle Bug Finding Using FPGA-Accelerated Simulation Donggyu Kim, Christopher Celio, Sagar Karandikar, David Biancolin, Jonathan Bachrach, Krste Asanović. ADEPT Winter Retreat 2018



Synthesizable Assertions

- Synthesizable Assertions on FPGA
 - Transform FIRRTL stop statements into synthesizable logic
 - Insert combinational logic and signals for the stop condition arguments
 - Insert encodings for each assertion (for matching error statements in SW)
 - Wire the assertion logic output to the Top-Level
 - Generate timing tokens for cycle-exact assertions
 - Assertion checker records the cycle and halts simulation when assertion is triggered





BOOM Example

- Example from `boom/src/main/scala/exu/rob.scala`
- Assert is the ROB is behaving un-expectedly
 - Overwriting a valid entry

```
assert (rob_val(rob_tail) === false.B, "[rob] overwriting a valid entry.")  
assert ((io.enq_uops(w).rob_idx >> log2Ceil(coreWidth)) === rob_tail)  
assert (!(io.wb_resps(i).valid && MatchBank(GetBankIdx(rob_idx)) &&  
!rob_val(GetRowIdx(rob_idx))), "[rob] writeback (" + i + ") occurred to an  
invalid ROB entry.")
```





BOOM Example

- How it looks in the UART output (while Linux is booting):

```
[    0.008000] VFS: Mounted root (ext2 filesystem) on device 253:0.  
[    0.008000] devtmpfs: mounted  
[    0.008000] Freeing unused kernel memory: 148K  
[    0.008000] This architecture does not have kernel memory protection.  
mount: mounting sysfs on /sys failed: No such device  
Starting syslogd: OK  
Starting klogd: OK  
Starting mdev...  
mdev: /sys/dev: No such file or directory  
[id: 1840, module: Rob, path: FireBoom.boom_tile_1.core.rob]  
Assertion failed: [rob] writeback (0) occurred to an invalid ROB entry.  
    at rob.scala:504 assert (!(io.wb_resps(i).valid && MatchBank(GetBankIdx(rob_idx)) ) &&  
    at cycle: 1112250469  
  
*** FAILED *** (code = 1841) after 1112250485 cycles  
time elapsed: 307.8 s, simulation speed = 3.61 MHz  
FPGA-Cycles-to-Model-Cycles Ratio (FMR): 2.77  
Beats available: 2165  
Runs 1112250485 cycles  
[FAIL] FireBoom Test  
SEED: 1569631756  
at cycle 4294967295
```

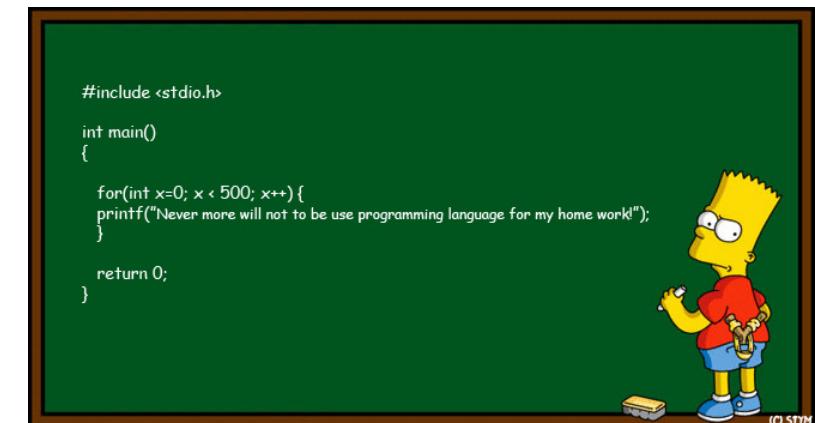
It would take ~62 hours to hit
this assertion is SW RTL
simulation (at 5 KHz sim rate),
vs. just a few minutes in FireSim





Synthesizable printf

- Research feature presented in DESSERT [1] (together with assertions)
- Enable “software-style” debugging using `printf` statements
- Convert Chisel `printf` statements to synthesizable blocks
 - Appropriate parsing in simulation bridge
 - Including signal values
- Impact on simulation performance depends on the frequency of `printf`s.
- Output includes the exact cycle of the `printf` event
 - Helps measure cycles counts between events



<https://www.deviantart.com/stym0r/art/Bart-Simpson-Programmer-134362686>





BOOM Example

- Example from boom/src/main/scala/lsu/lsu.scala
- Print a trace of all loads and stores, for verifying memory consistency.

```
if (MEMTRACE_PRINTF) {  
    when (commit_store || commit_load) {  
        val uop      = Mux(commit_store, stq(idx).bits.uop, ldq(idx).bits.uop)  
        val addr     = Mux(commit_store, stq(idx).bits.addr.bits, ldq(idx).bits.addr.bits)  
        val stdata   = Mux(commit_store, stq(idx).bits.data.bits, 0.U)  
        val wbdata   = Mux(commit_store, stq(idx).bits.debug_wb_data, ldq(idx).bits.debug_wb_data)  
        printf(midas.targetutils.SynthesizePrintf("MT %x %x %x %x %x %x %x\n",  
            io.core.tsc_reg, uop.uopc, uop.mem_cmd, uop.mem_size, addr, stdata, wbdata))  
    }  
}
```





Synthesizable printf/Assertions

Pros:

- FPGA simulation speed
- Real-time trigger-based
- Consumes small amount of FPGA resources (compared to ILA)
- Key signals have pre-written assertions in re-usable components/libraries

Cons:

- Low visibility: No waveform/state
- Assertions are best added while writing source RTL rather than during “investigative” debugging
- Large numbers of `printf`s can slow down simulation





Spike Co-Simulation

- Spike – Golden reference RISC-V functional simulator
- Can be used to debug BOOM in FireSim through functional co-simulation and comparison
- Find functional bugs billions of cycles into simulations
 - Find divergence against functional golden model
 - Dump waveforms for affected signals

```
[error] Spike PC ffffffe001055d84, DUT PC ffffffe001055d84
[error] Spike INSN 14102973, DUT INSN 14102973
[error] Spike WDATA 000220d6, DUT WDATA 000220d4
[error] Spike MSTATUS a000000a0, DUT MSTATUS 00000000
[error] DUT pending exception -1 pending interrupt -1
[ERROR] Copsike: Errorred during simulation tick with 8191

*** FAILED *** (code = 8191) after 2,356,509,311 cycles
time elapsed: 2740.8 s, simulation speed = 859.79 KHz
FPGA-Cycles-to-Model-Cycles Ratio (FMR): 8.14
Runs 2356509311 cycles
FAIL] FireSim Test
```

2 billion cycle divergence where receiving an interrupt during mis-speculation affects architectural state (EPC)





Spike Co-Simulation

Pros:

- FPGA simulation speed
- Verify against golden model
- Out-of-Band (no impact on workload execution)

Cons:

- Slower simulation performance (40 MHz)
- No uarch verification





Hands on with AutoCounter





Hands-on Synthesizable AutoCounter Example

- We would like to observe some statistics about when Gemmini stalls
- [**\\$ACYDIR/generators/gemmini/src/main/scala/gemmini/DMA.scala**](#)
 - Line 324 and 637

```
PerfCounter(tl.a.ready && translate_q.io.deq.valid && io.tlb.resp.miss, "rdma_tlb_wait_cycles",
            "cycles during which the read dma is stalling as it waits for a TLB response")
PerfCounter(tl.a.valid && !tl.a.ready, "rdma_tl_wait_cycles",
            "cycles during which the read dma is stalling as it waits for the TileLink port to be available")

PerfCounter(tl.a.ready && translate_q.io.deq.valid && io.tlb.resp.miss, "wdma_tlb_wait_cycles",
            "cycles during which the write dma is stalling as it waits for a TLB response")
PerfCounter(tl.a.valid && !tl.a.ready, "wdma_tl_wait_cycles",
            "cycles during which the write dma is stalling ... for the TileLink port to be available")
```





Hands-on Synthesizable AutoCounter Example

- For reference, the build recipe for this FPGA image (in \$FDIR/deploy/config_build_recipes.yaml) is:

```
firesim_gemmini_rocket_singlecore_no_nic:  
    DESIGN: FireSim  
    TARGET_CONFIG: FireSimLeanGemminiRocketConfig  
    PLATFORM_CONFIG: WithAutoCounter_BaseF1Config  
    deploy_triplet: null  
    platform_config_args:  
        fpga_frequency: 10  
        build_strategy: TIMING  
    post_build_hook: null  
    metasim_customruntimeconfig: null  
    bit_builder_recipe: bit-builder-recipes/f1.yaml
```

This is already set for you!
Pre-enabling the counters in
the bitstream build!





Hands-on Synthesizable AutoCounter Example

Update our workload to copy the output printf file:

- vim \$FDIR/deploy/workloads/resnet50-baremetal.json
- Add the AUTOCOUNTERFILE*.csv to our simulation output

```
{  
    "benchmark_name": "resnet50-baremetal",  
    "common_simulation_outputs": [  
        "uartlog", "AUTOCOUNTERFILE*.csv"  
    ],  
    "common_bootbinary": "...",  
    "common_rootfs": "..."  
}
```

Make sure to avoid adding an extra comma!





Hands-on Synthesizable AutoCounter Example

- Setup the `config_runtime.yaml`

```
$ vim $FDIR/deploy/config_runtime.yaml
```

- Select the AGFI that was synthesized with counters
- Select the baremetal ResNet50 workload
- Tell sample rate of counters to enable them
- Boot the simulation by running the following sequence of commands:

```
$ firesim infrasetup
```

- This should take about 3 minutes

```
$ firesim runworkload
```

- This should take about <1 minute

```
run_farm:  
  recipe_arg_overrides:  
    run_farm_hosts_to_use:  
      - f1.2xlarge: 1  
  
target_config:  
  topology: no_net_config  
  no_net_num_nodes: 1  
  link_latency: 6405  
  switching_latency: 10  
  net_bandwidth: 200  
  profile_interval: -1  
  
  default_hw_config:  
    firesim_gemmini_rocket_singlecore_n  
    o_nic  
  
workload:  
  workload_name: resnet50-  
  baremetal.json  
  
autocounter:  
  read_rate: 10000000
```



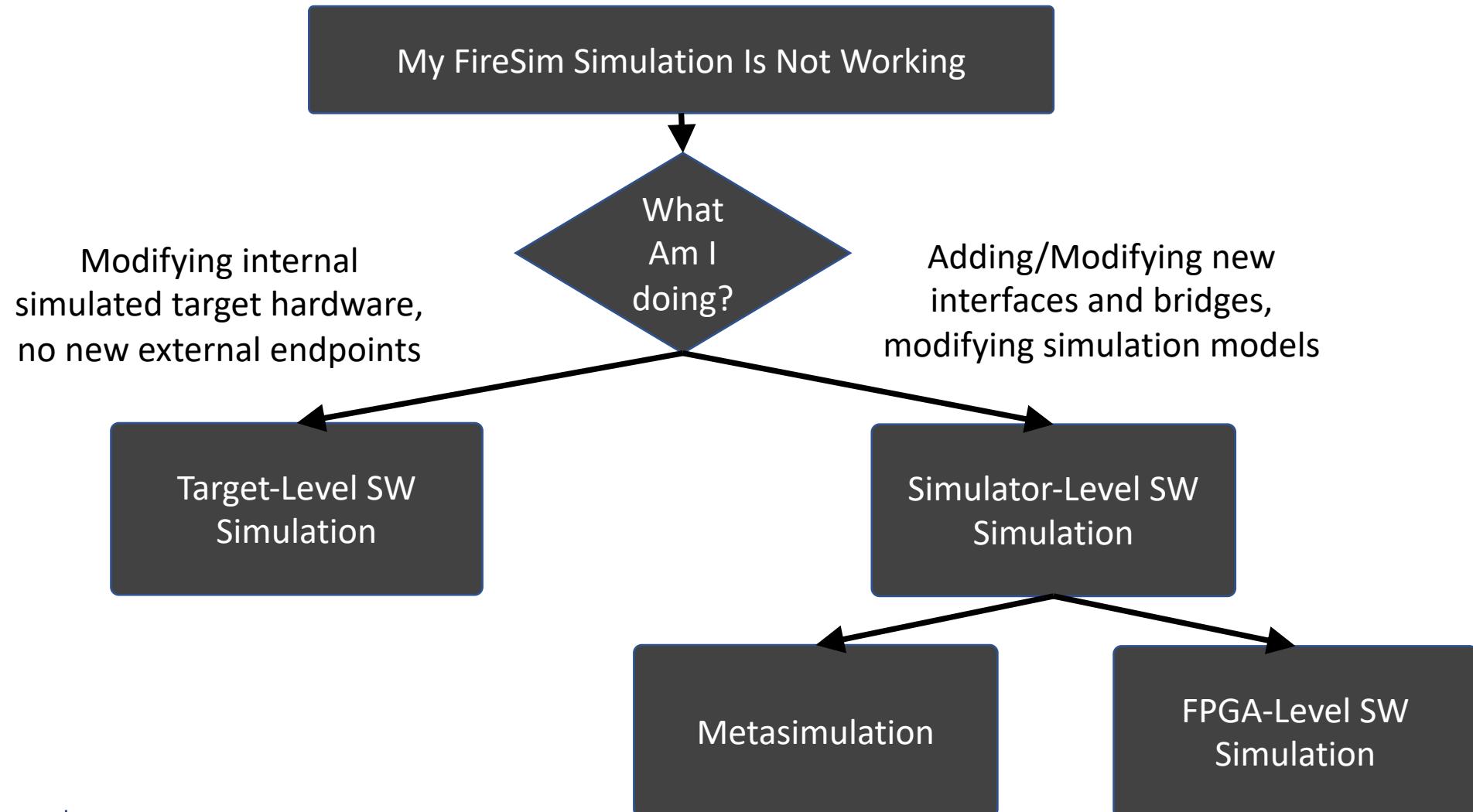


While this is running...





Debugging Using Software RTL Simulation





Debugging Using Software RTL Simulation

Target-Level Simulation

- Software Simulation
- Target Design Untransformed
- No Host-FPGA interfaces

Metasimulation

- Software Simulation
- Target Design Transformed by Golden Gate
- Host-FPGA interfaces/shell emulated using abstract models

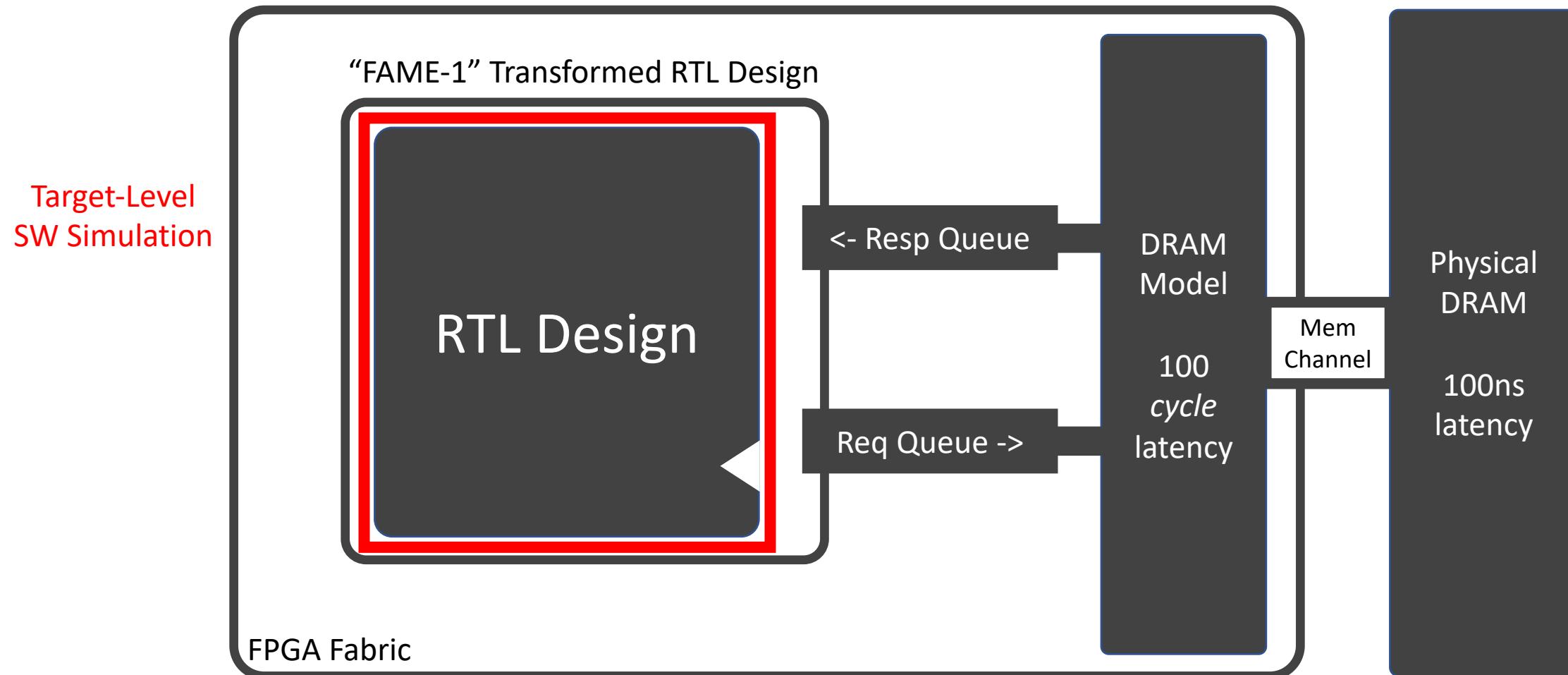
FPGA-Level Simulation

- Software Simulation
- Target Design Transformed by Golden Gate
- Host-FPGA interfaces/shell simulated by the FPGA tools





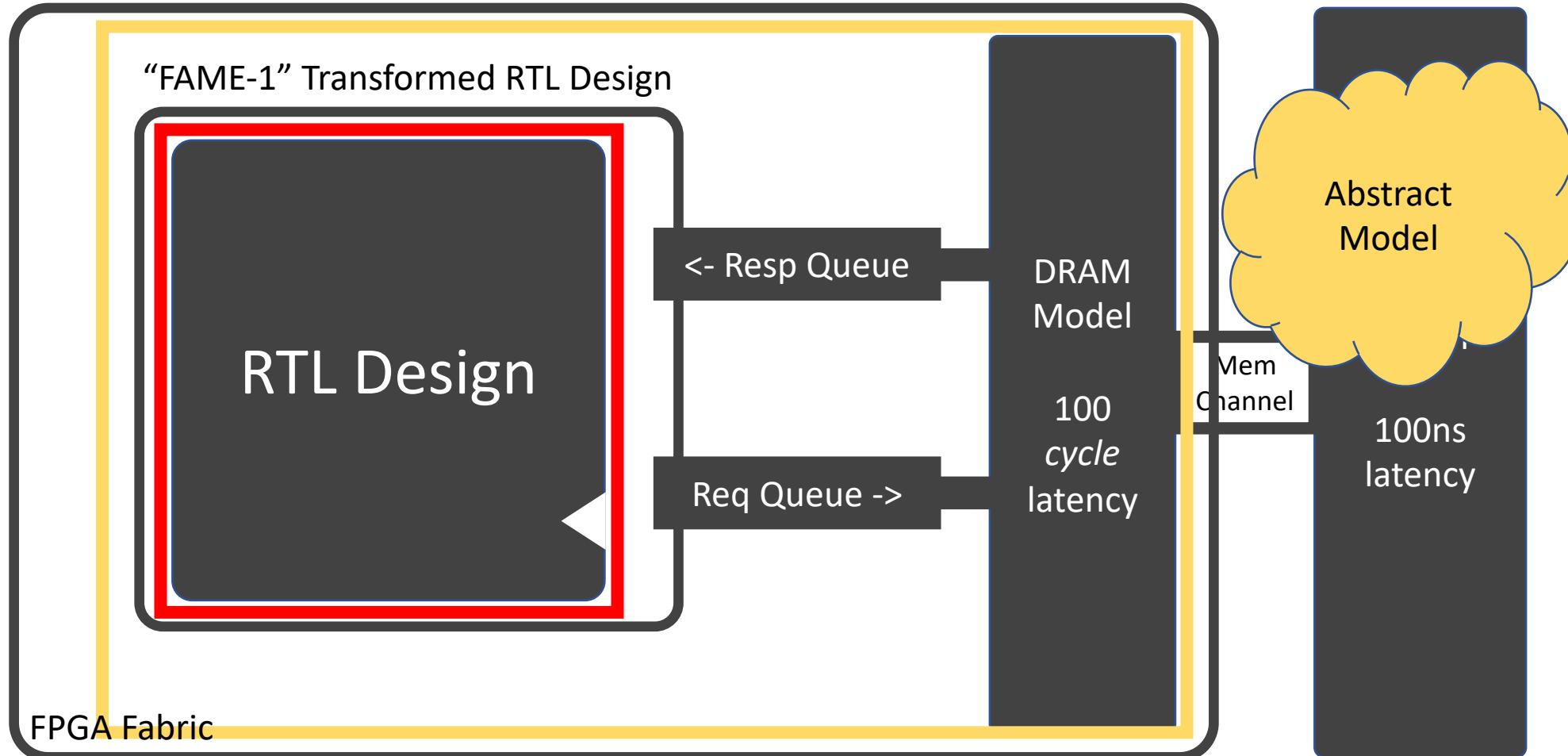
Debugging Using Software RTL Simulation





Debugging Using Software RTL Simulation

Target-Level
SW Simulation
Metasimulation



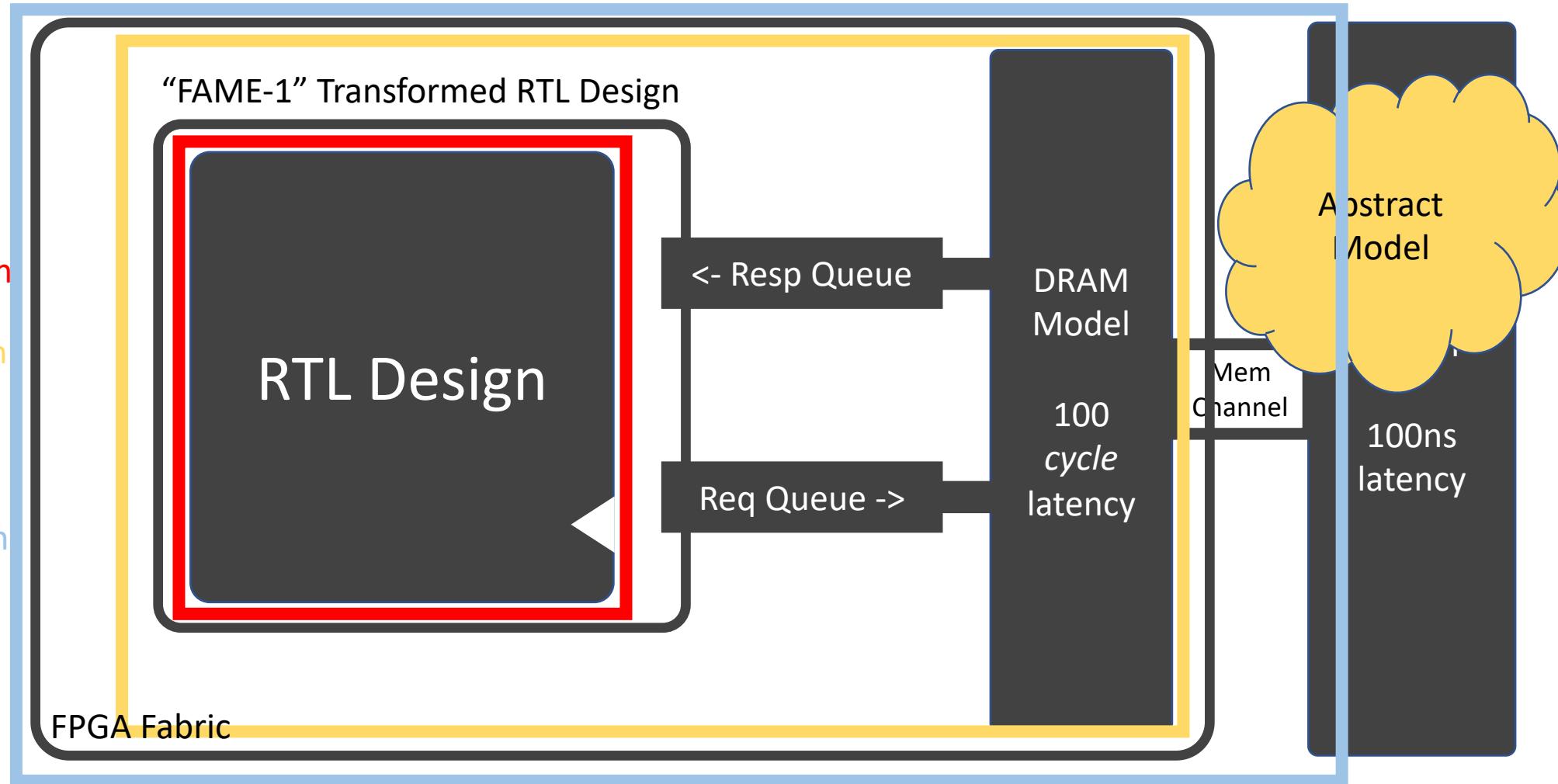


Debugging Using Software RTL Simulation

Target-Level
SW Simulation

Metasimulation

FPGA-Level
SW Simulation





Debugging Using Software RTL Simulation

Level	Waves	VCS	Verilator	XSIM
Target	Off	~5 kHz	~5 kHz	N/A
Target	On	~1 kHz	~5 kHz	N/A
Meta	Off	~4 kHz	~2 kHz	N/A
Meta	On	~3 kHz	~1 kHz	N/A
FPGA	On	~2 Hz	N/A	~0.5 Hz





Back to our hands-on example





Viewing the Simulation

Look for the run instance's IP address in the status:

```
FireSim Simulation Status @ 2022-06-18 00:17:10.188191
-----
This workload's output is located in:
/home/centos/chipyard/sims/firesim/deploy/results-workload/2022-06-18--00-16-00-
mobilenet-baremetal/
This run's log is located in:
/home/centos/chipyard/sims/firesim/deploy/logs/2022-06-18--00-16-00-runworkload-
NEZCRUKBA2M44B9M.log
This status will update every 10s.
-----
Instances
-----
Hostname/IP: 192.168.3.52 | Terminated: False
-----
Simulated Switches
-----
Simulated Nodes/Jobs
-----
Hostname/IP: 192.168.3.52 | Job: resnet50-baremetal0 | Sim running: True
-----
Summary
-----
1/1 instances are still running.
1/1 simulations are still running.
```





Viewing the Simulation

- On the *manager* instance, ssh into the run farm instance:

```
$ ssh 192.168.3.52
```

```
| ____| _ \ \V\ / | / \ \ | \ \ \V\ / / / | / \ \ \V\ / | \ \ \ | | | | | | |
| _ || _ / ( _ | / _ \ | _ | ) | _ | \ v / | / _ \ | \V | _ | _ |
| _ | _ | \ _ / _ \ \ _ | _ | / | _ | \ _ / | / _ \ _ | _ | _ |
AMI Version: 1.11.4
Xilinx Version: 2021.1
Readme: /home/centos/src/README.md
AMI Release Notes: /home/centos/src/RELEASE_NOTES.md
GUI/Cluster setup: https://github.com/aws/aws-fpga/blob/master/developer\_resources
```

- Then look at the stream of prints (or if complete, look at the output results)

```
$ tail -f sim slot 0/AUTOCOUNTER*
```





Hands-on Synthesizable AutoCounter Example

Output file in

\$FDIR/deploy/results-workload/<timestamp>-resnet50-baremetal/resnet50-baremetal0/AUTOCOUNTER0.csv

```
version,1
Clock Domain Name, ...
label, local_cycle, ...rdma_tlb_wait_cycles, ...wdma_tl_wait_cycles, ...wdma_tlb_wait_cycles, ...rdma_tl_wait_cycles
"description", ...
type, Accumulate, Accumulate, Accumulate, Accumulate
event width, 1, 1, 1, 1
accumulator width, 64, 64, 64, 64
...
50000000, 50000000, 37, 245382, 2175, 287878
...
170000000,170000000,5416,5073118,56093,10989706
...
2320000000,2320000000,24953,15317004,364054,25548250
...
```

... let's view this as a table





Hands-on Synthesizable AutoCounter Example

Output file in

`$FDIR/deploy/results-workload/<timestamp>-resnet50-baremetal/resnet50-baremetal0/synthesized-prints.out`

label	local_cycle	...rdma_tlb_wait_cycles	...wdma_tl_wait_cycles	...wdma_tlb_wait_cycles	...rdma_tl_wait_cycles
50000000	50000000	37	245382	2175	287878
170000000	170000000	5416	5073118	56093	10989706
	Sample at cycle 170M	24953		Showing waiting for memory response (TileLink) is much higher than waiting for a TLB response	48250





Hands-on Synthesizable AutoCounter Example

Output file in

\$FDIR/deploy/results-workload/<timestamp>-resnet50-baremetal/resnet50-baremetal0/synthesized-prints.out

label	local_cycle	...rdma_tlb_w ait_cycles	...wdma_tl_wa it_cycles	...wdma_tlb_w ait_cycles	...rdma_tl_wa it_cycles
50000000	50000000	37	245382	2175	287878
170000000	170000000				10989706
2320000000	2320000000		<ul style="list-style-type: none">• Exit tail by doing Ctrl-C• Then exit out of the simulation instance with Ctrl-d to return to the manager instance		25548250

If tail'ing the result...

- Exit tail by doing Ctrl-C
- Then exit out of the simulation instance with Ctrl-d to return to the manager instance





Hands-on Synthesizable AutoCounter Example

Don't forget to terminate your runfarms (otherwise, we are going to pay for a lot of FPGA time)

```
$ firesim terminaterunfarm
```

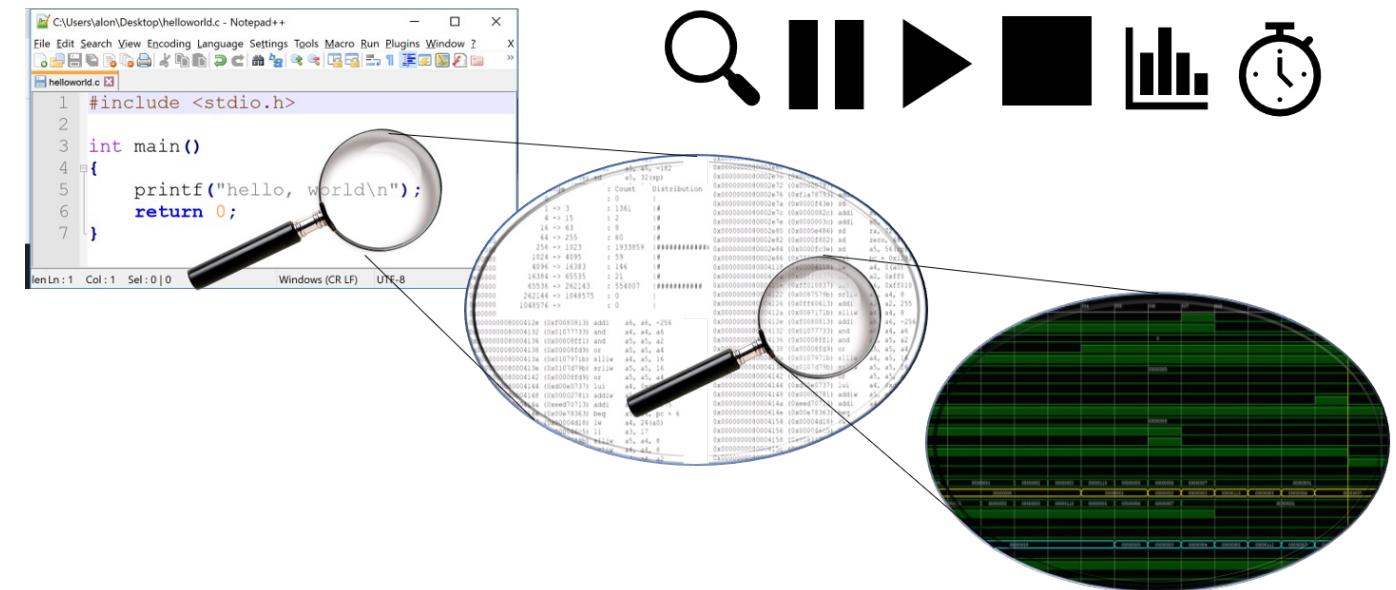
Type **yes** at the prompt to confirm





The FireSim Vision: Speed and Visibility

- High-performance simulation
- Full application workloads
- Tunable visibility & resolution
- Unique data-based insights





Summary

- Debugging Using Integrated Logic Analyzers ([docs](#))
- Advanced Debugging and Profiling Features
 - TracerV ([docs](#))
 - AutoCounter ([docs](#))
 - Assertion and Print Synthesis ([docs](#))
- Debugging Using Software Simulation ([docs](#))
 - Target-Level
 - Metasimulation
 - FPGA-Level
- FireSim Debugging and Profiling Future Vision

Check out <https://docs.firesim.org/> for more usage details

