



# FireSim

## Debugging, Testing, and Profiling

FireSim Intensive  
Chisel Community Conference 2018  
Speaker: Alon Amid

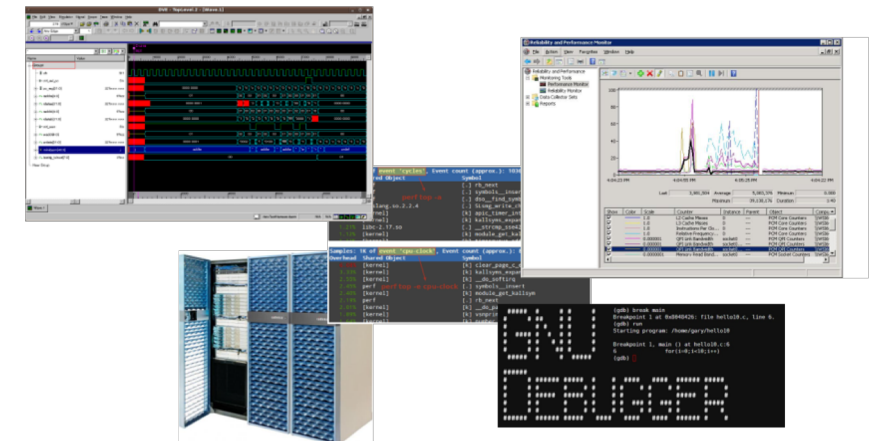


**Berkeley Architecture Research**



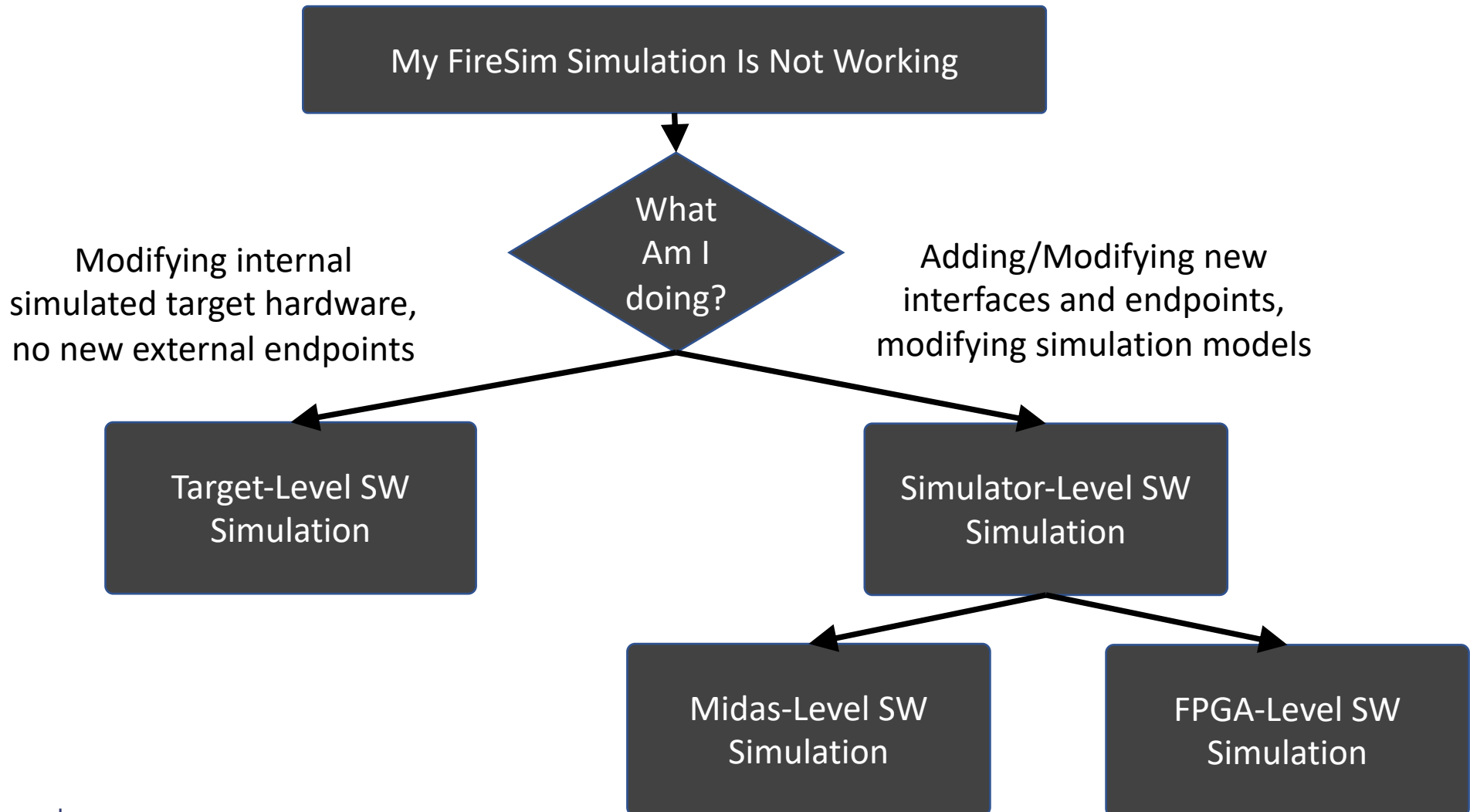
# Agenda

- FireSim Debugging Using Software Simulation
- FireSim Debugging Using Integrated Logic Analyzers
- Advanced FPGA-Based Debugging and Profiling Features
- The FireSim Vision for Debugging and Profiling





# Debugging Using Software Simulation





# Debugging Using Software Simulation

## Target-Level Simulation

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

## MIDAS-Level Simulation

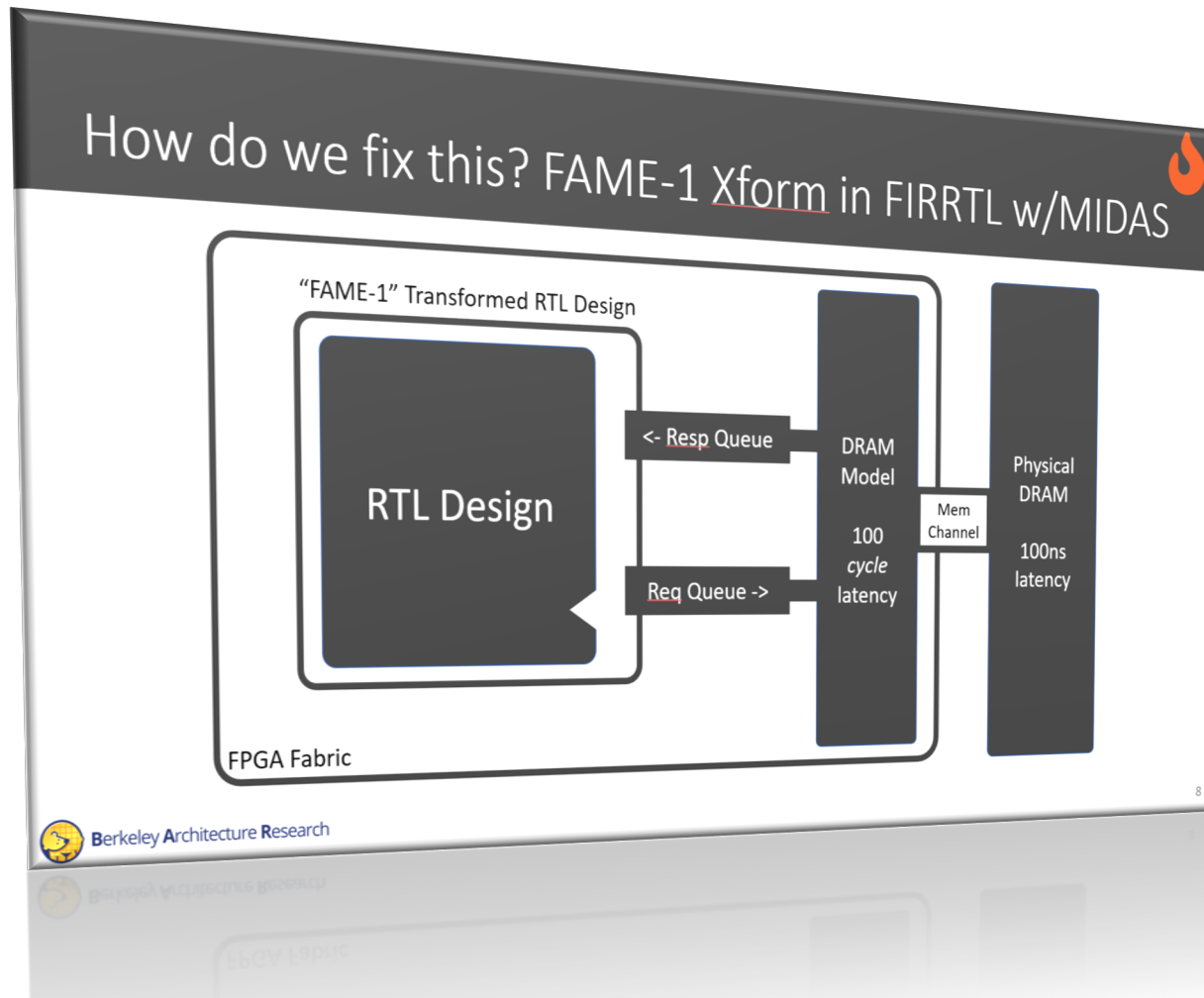
- Software Simulation
- Target Design Transformed by MIDAS
- Host-FPGA interfaces/shell emulated using abstract models

## FPGA-Level Simulation

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



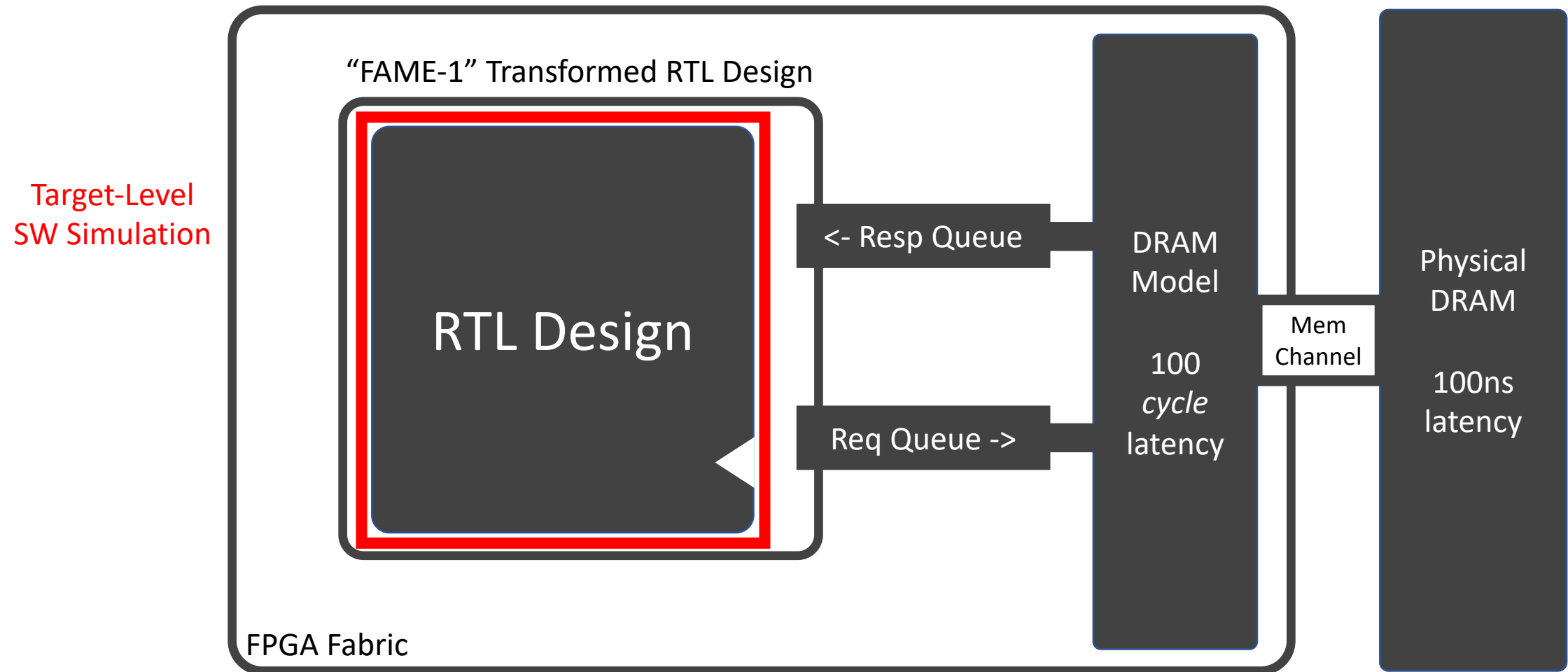
# Debugging Using Software Simulation



Remember  
this slide from  
Sagar's  
presentation?

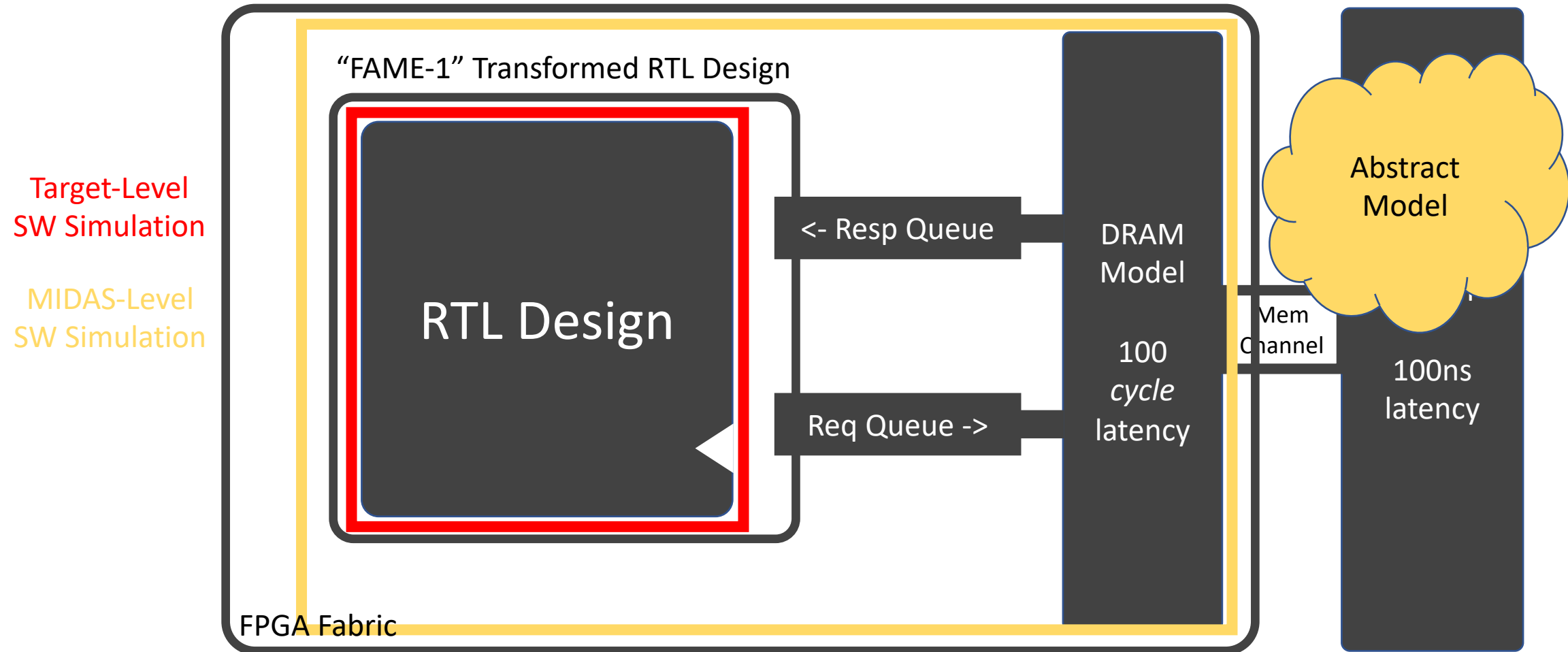


# Debugging Using Software Simulation



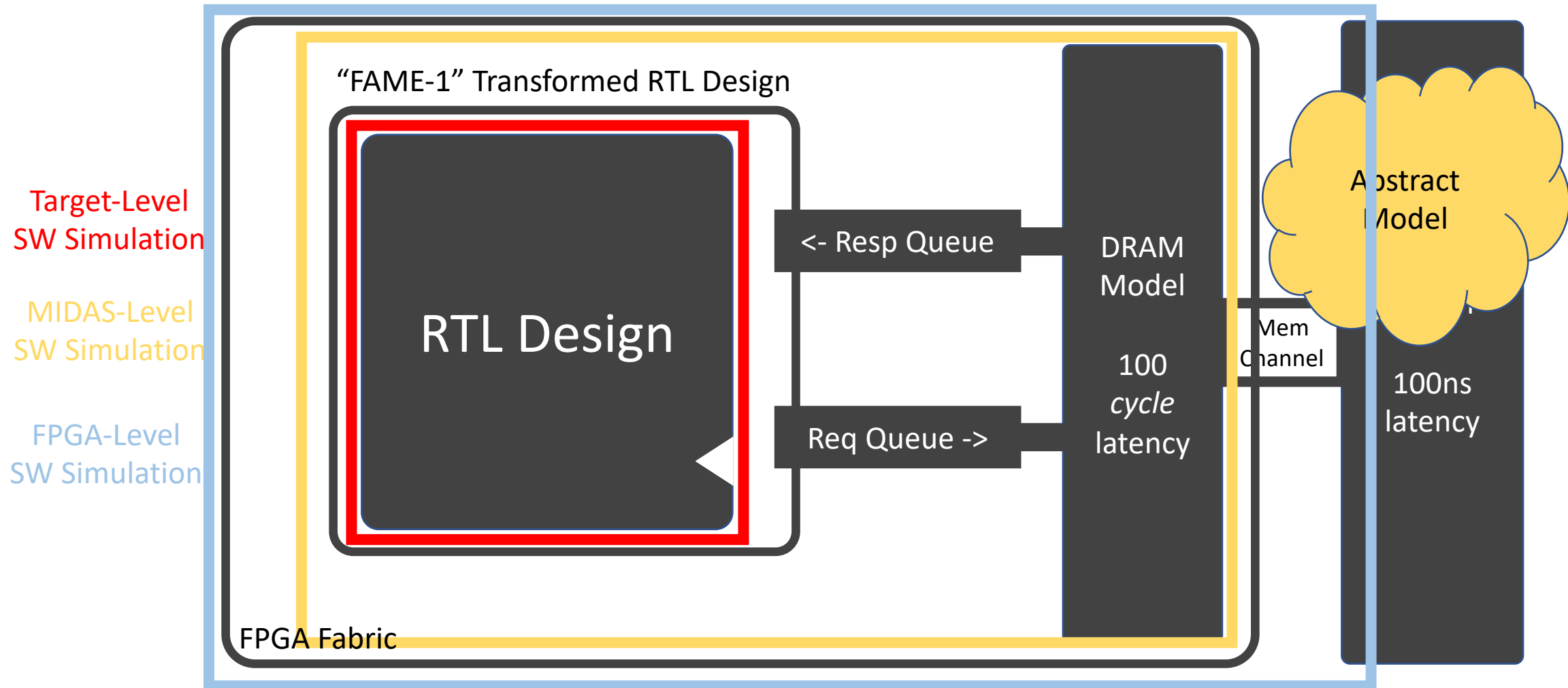


# Debugging Using Software Simulation





# Debugging Using Software Simulation





# Debugging Using Software Simulation

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



# Debugging Using Software Simulation

- **Target-Level Simulation**

In firesim/target-design/firechip/vsim

```
$ make DESIGN=<YourDesign> CONFIG=<YourConfig> debug
```

```
$ ./simv-<YourDesign>-<YourConfig>-debug +max-cycles=50000 +vcdplusfile=<WaveformFileName>.vpd  
../tests/<InputTest>.riscv
```

- **MIDAS-level Simulation**

In firesim/sim

```
$ make <verilator|vcs>-debug
```

```
$ make EMUL=<verilator|vcs> DESIGN=FireSimNoNIC run-asm-test-debug
```

- **FPGA-Level Simulation**

In firesim/sim

```
$ make xsim
```

```
$ make xsim-dut <VCS=1> &
```

```
$ make run-xsim SIM_BINARY=<PATH/TO/DRIVER/BINARY/FOR/TARGET/TO/RUN>
```

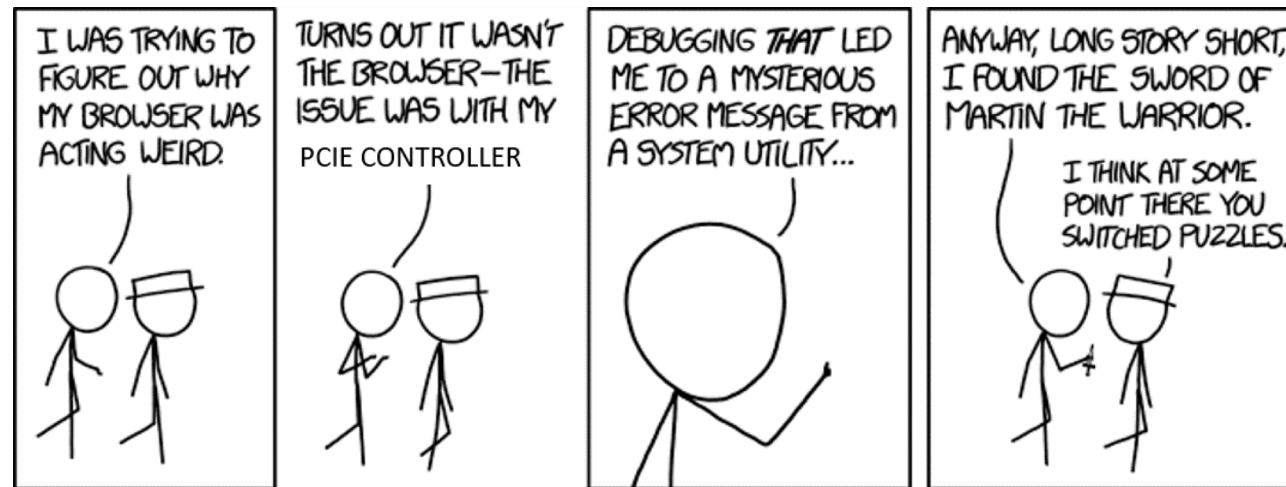
**\*\*FPGA-level simulation currently does not support DMA\_PCIS (which is used for the NIC interface)**



# When SW Simulation Debugging 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”





# Debugging Using Integrated Logic Analyzers

## Integrated Logic Analyzers (ILAs)

- Common debugging feature provided by FPGA vendors
- Continuous recording of a sampling window of up to 1024 cycles.
  - 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_avalid_q),
    .probe1 (sh_ocl_avalid_q),
    .probe2 (ocl_sh_avalid_q),
    .probe3 (sh_ocl_arvalid_q),
    .probe4 (sh_ocl_araddr_q),
    .probe5 (ocl_sh_arready_q)
);

ila_0 Cl_ILA_1 (
    .clk    (clk_main_a0),
    .probe0 (ocl_sh_bvalid_q),
    .probe1 (sh_cl_glcount0_q),
    .probe2 (sh_ocl_bready_q),
    .probe3 (ocl_sh_rvalid_q),
    .probe4 ({32'b0, ocl_sh_rdata_q[31:0]}),
    .probe5 (sh_ocl_rready_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)
);
```

From: aws-fpga cl\_hello\_world example





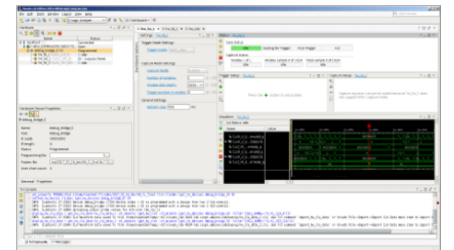
# Debugging Using Integrated Logic Analyzers

## 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.passes.FpgaDebugAnnotation

class SomeModuleIO(implicit p: Parameters) extends SomeIO()(p){
  val out1 = Output(Bool())
  val in1 = Input(Bool())
  chisel3.experimental.annotate(FpgaDebugAnnotation(out1))
}
```





# 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



# Advanced FPGA-Based Debugging Features

- FPGA-based simulation enables high simulation speed, which enables advanced debugging and profiling tools.
- Reach “deep” in simulation time, and obtain large levels of coverage and data
- Examples:
  - TracerV
  - Synthesizable Assertions



SW  
Simulation



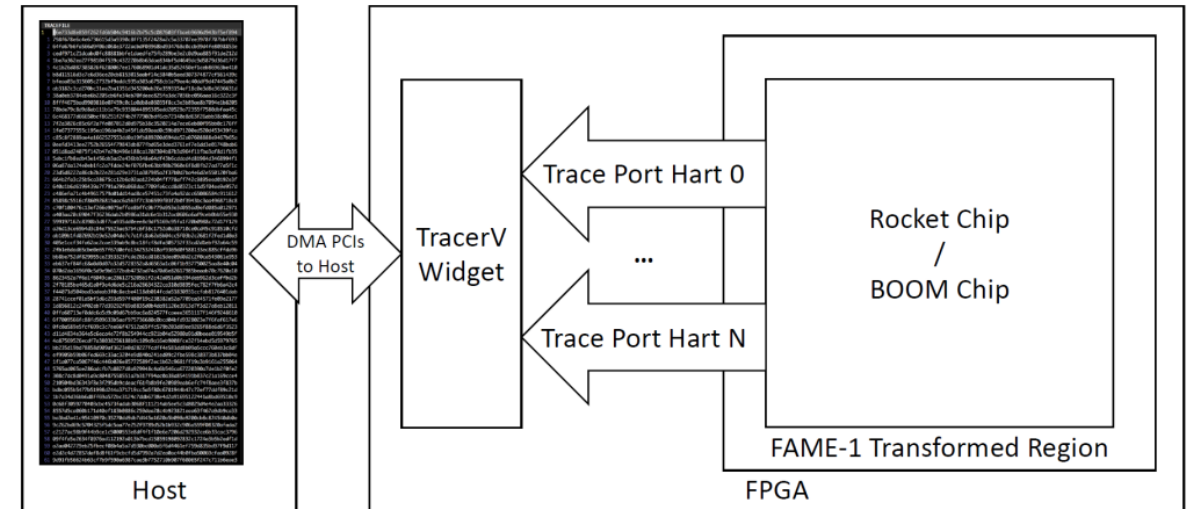
FPGA-based  
Simulation

Simulated  
Time



# TracerV

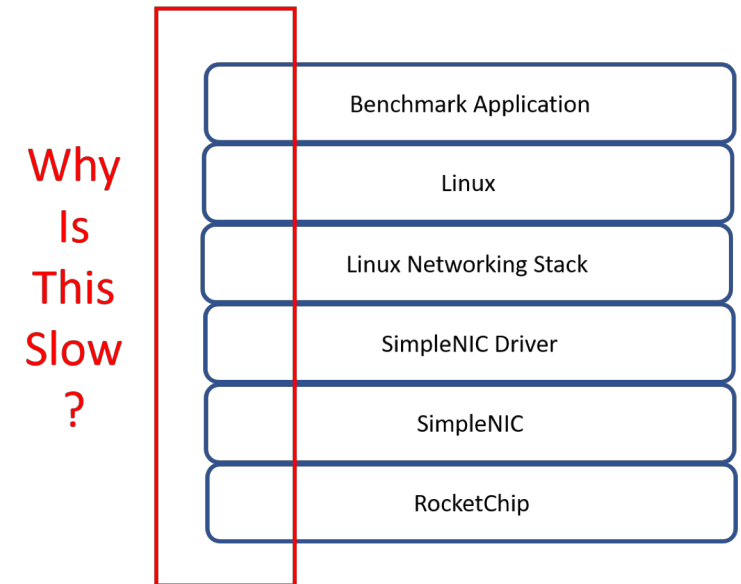
- **Out-of-band** full instruction execution trace
- MIDAS widget connected to target trace ports
- By default, large amount of info wired out of Rocket/BOOM, per-hart, per-cycle:
  - Instruction Address
  - Instruction
  - Privilege Level
  - Exception/Interrupt Status, Cause
- TracerV can rapidly generate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





# TracerV

- Out-of-Band software-level debugging and profiling: 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.





# 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



# 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 defaults, emitted as non-synthesizable SV functions (\$fatal)



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)  
}
```

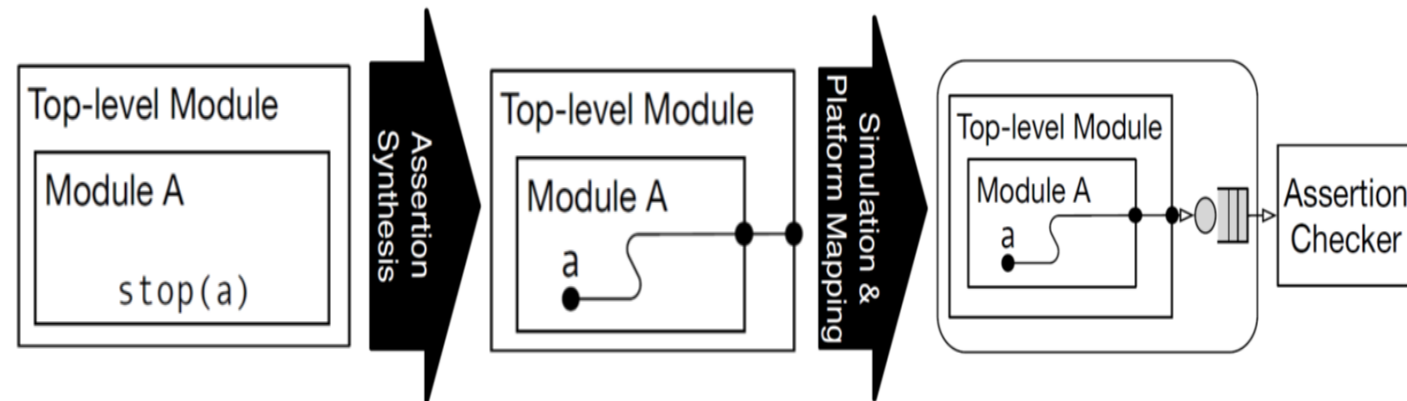
From: Trillion-Cycle Bug Finding Using FPGA-Accelerated Simulation Donggyu Kim, Christopher Celio, Sagar Karandikar, David Biancolin, Jonathan Bachrach, Krste Asanović. ADEPT Winter Retreat 2018





# 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





# Synthesizable Assertions

- Previously a research feature presented in DESSERT [1]
- Helped Identify BOOM bugs trillions of cycles into execution
- Integrated into FireSim in the latest release.

## DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of cycles

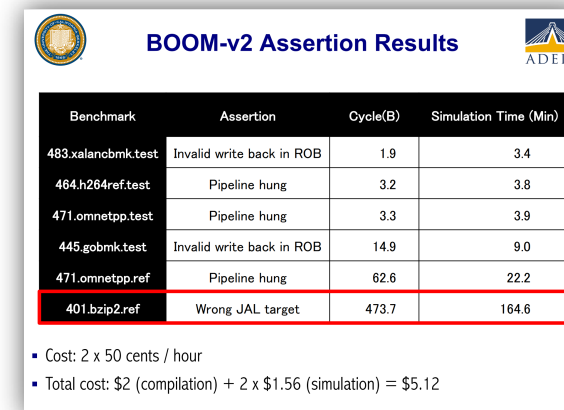
Donggyu Kim<sup>1</sup>, Christopher Celio<sup>2</sup>, Sagar Karandikar<sup>1</sup>, David Biancolin<sup>1</sup>, Jonathan Bachrach<sup>1</sup>, Krste Asanović<sup>1</sup>

<sup>1</sup>Department of Electrical Engineering and Computer Sciences, University of California, Berkeley

{dggkim, sagark, biancolin, jrb, krste}@eecs.berkeley.edu

<sup>2</sup>Esperanto Technologies

christopher.celio@esperantotech.com



Benchmark	Assertion	Cycle(B)	Simulation Time (Min)
483.xalancbmk.test	Invalid write back in ROB	1.9	3.4
464.h264ref.test	Pipeline hung	3.2	3.8
471.omnetpp.test	Pipeline hung	3.3	3.9
445.gobmk.test	Invalid write back in ROB	14.9	9.0
471.omnetpp.ref	Pipeline hung	62.6	22.2
401.bzip2.ref	Wrong JAL target	473.7	164.6

Cost: 2 x 50 cents / hour  
Total cost: \$2 (compilation) + 2 x \$1.56 (simulation) = \$5.12

[1] Kim, D., Celio, C., Karandikar, S., Biancolin, D., Bachrach, J. and Asanovic, K., DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of cycles. *The International Conference on Field-Programmable Logic and Applications (FPL)*, 2018





# Synthesizable 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:

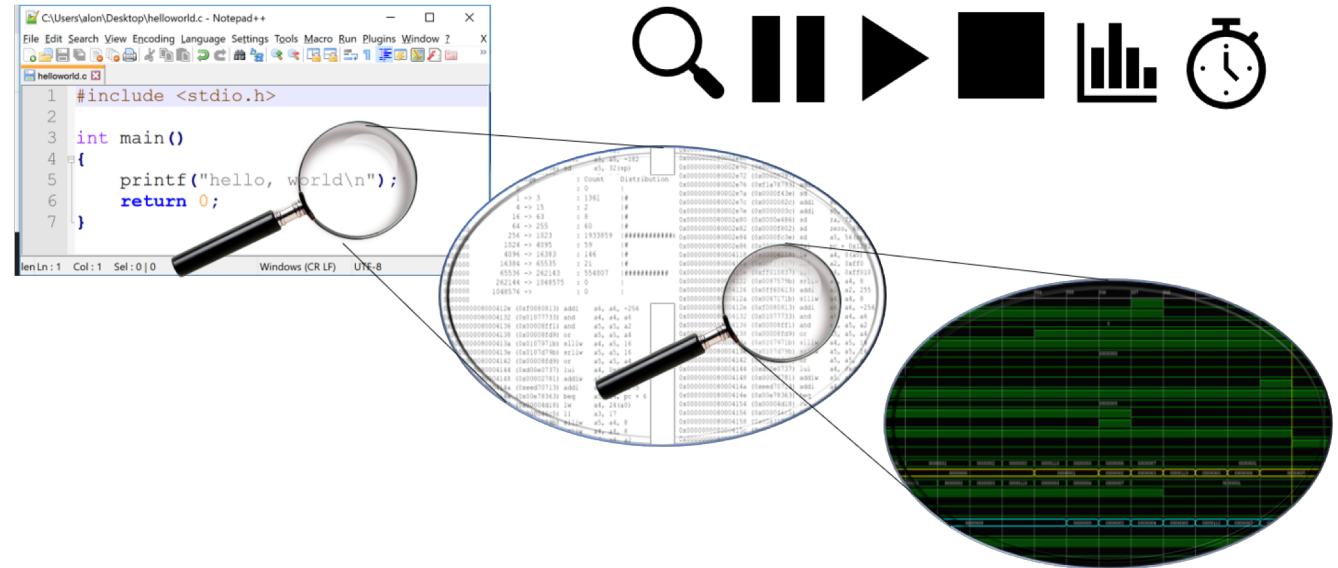
- Low visibility: No waveform/state
- Assertions are best added while writing source RTL rather than during “investigative” debugging





# The FireSim Vision: Speed and Visibility

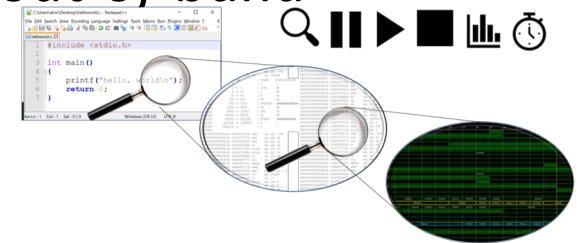
- High-performance simulation
- Full application workloads
- Tunable visibility & resolution
- Unique data-based insights





# Speed and Visibility – How do we get there?

- Integration of additional DESSERT features
  - Hardware state snapshot extraction from FPGA to software simulation using arbitrary software triggers
  - Easy-to-use information transfer between execution traces and software hooks
- Data-Processing pipeline for insights from large-scale traces
  - O(GB)-O(TB) out-of-band logs require big-data analysis methods for insights (Golden model comparison is one potential method)
  - Potential unique insights: can *collect globally-cycle-accurate out-of-band instruction traces from a networked datacenter simulation.*





# Summary

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

Check out <https://docs.fires.im/>  
for more usage details