FireSim

# Instrumenting and Debugging FireSim-Simulated Designs

**https://fires.im**

**@firesimproject**

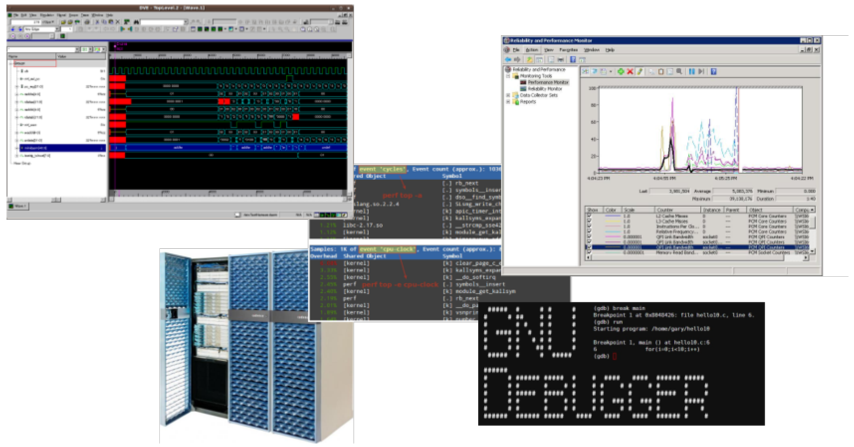
**MICRO 2021 Tutorial**

Abraham Gonzalez

Berkeley Architecture Research

# Agenda

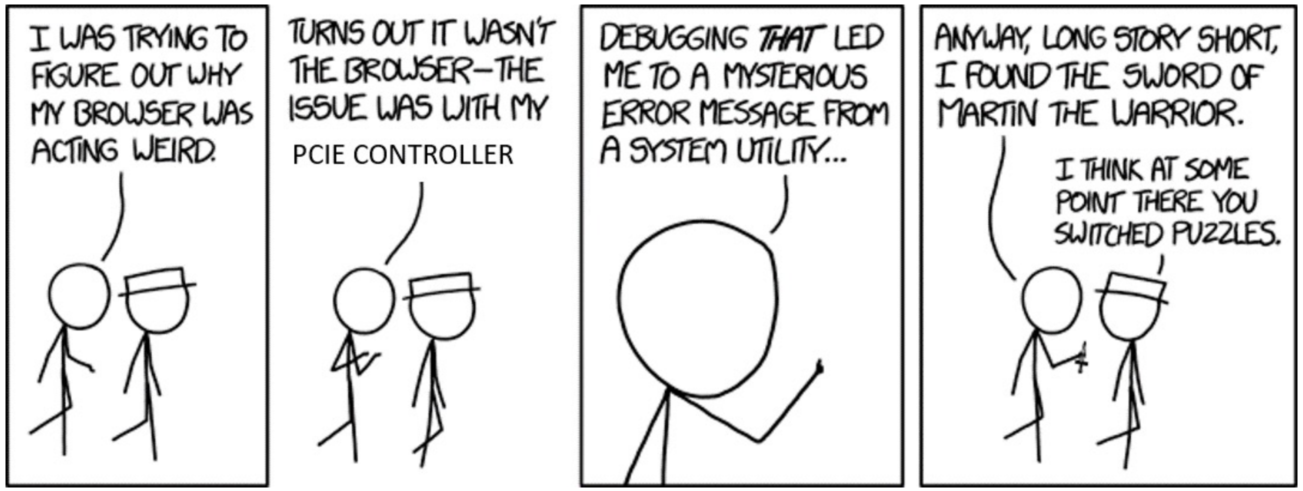
- FPGA-Accelerated Deep-Simulation Debugging
  - Debugging Using Integrated Logic Analyzers
  - Trace-based Debugging
  - Out-of-band Performance Counters
  - Synthesizable Assertions/Prints
  - Dromajo and FireSim
- Debugging Co-Simulation
  - FireSim Debugging Using Software Simulation



# When SW RTL Simulation is Not Enough...

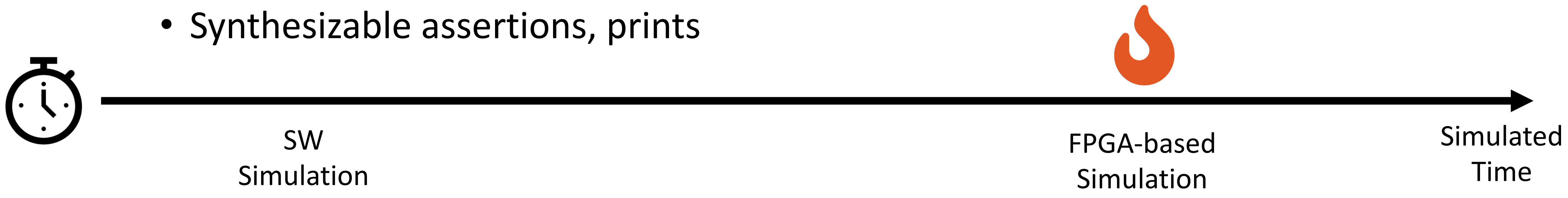
"Everything looks OK in SW simulation, but there is still a bug somewhere"

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


I WAS TRYING TO FIGURE OUT WHY MY BROWSER WAS ACTING WEIRD.
TURNS OUT IT WASN'T THE BROWSER—THE ISSUE WAS WITH MY PCIE CONTROLLER
DEBUGGING THAT LED ME TO A MYSTERIOUS ERROR MESSAGE FROM A SYSTEM UTILITY...
ANYWAY, LONG STORY SHORT, I FOUND THE SWORD OF MARTIN THE WARRIOR.
I THINK AT SOME POINT THERE YOU SWITCHED PUZZLES.

# FPGA-Based Debugging Features

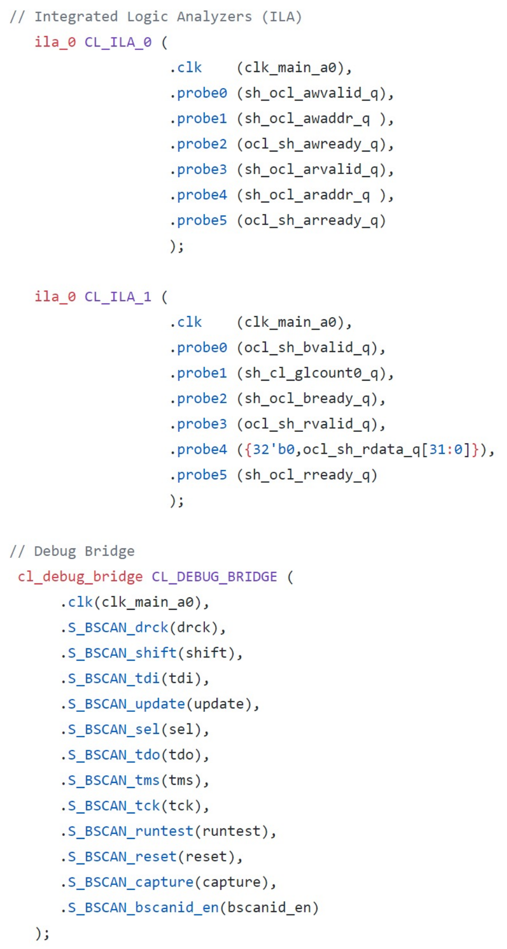
- High simulation speed in FPGA-based simulation enables advanced debugging and profiling tools.
- Reach "deep" in simulation time, and obtain large levels of coverage and data
- Examples:
  - ILAs
  - TracerV
  - Synthesizable assertions, prints



# Debugging Using Integrated Logic Analyzers

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



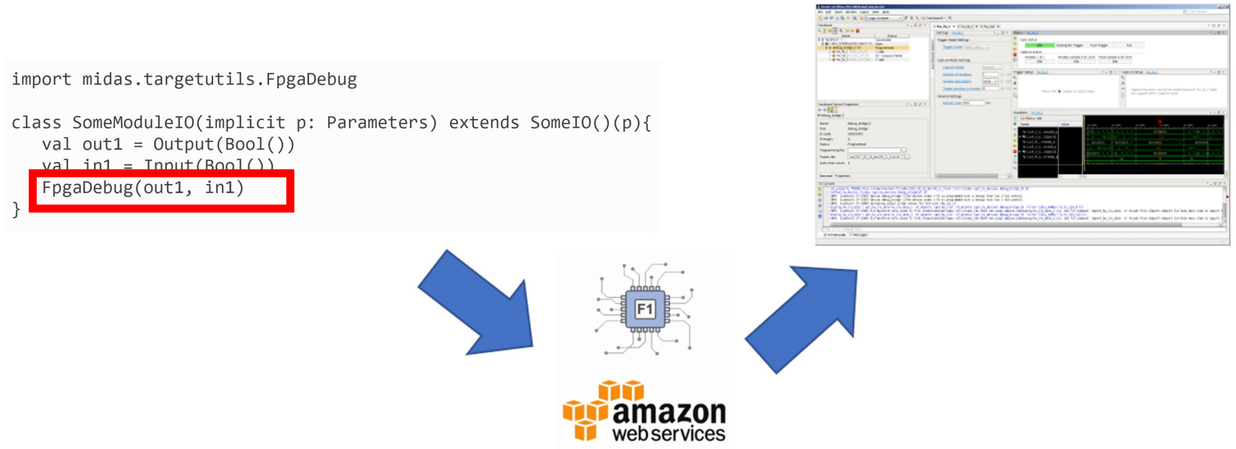
```
// Integrated Logic Analyzers (ILA)
   ila_0 CL_ILA_0 (
                   .clk    (clk_main_a0),
                   .probe0 (sh_ocl_awvalid_q),
                   .probe1 (sh_ocl_awaddr_q ),
                   .probe2 (ocl_sh_awready_q),
                   .probe3 (sh_ocl_arvalid_q),
          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

   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.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 OoO processor is hard
  - Throughout this talk, we'll have examples of FPGA debugging used in BOOM.
- Example from `boom/src/main/scala/lsu/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:

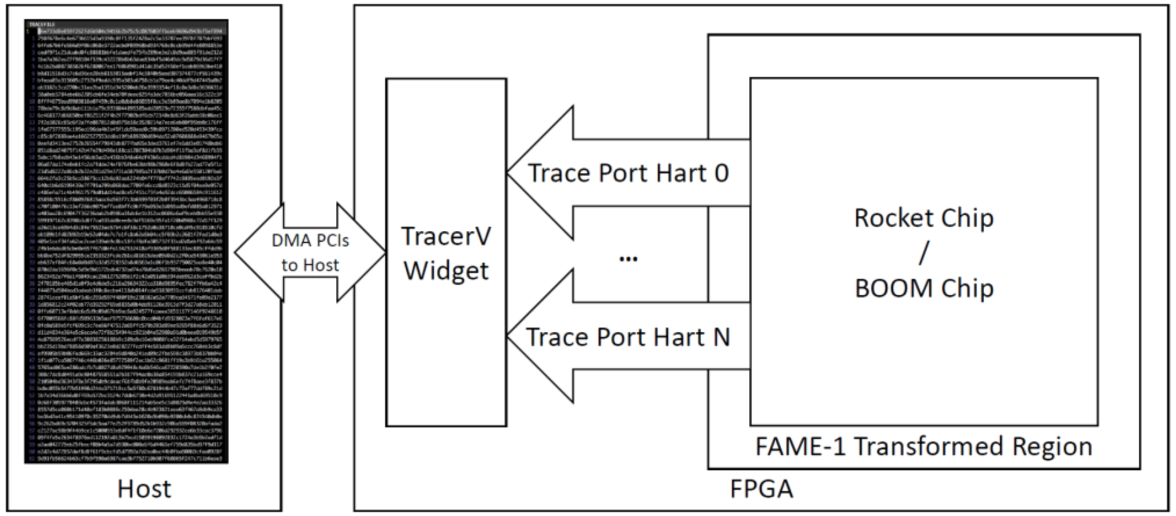
- No emulated parts – what you see is what's running on the FPGA
- FPGA simulation speed - O(MHz) compared to O(KHz) in software simulation
- Real-time trigger-based

Cons:

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

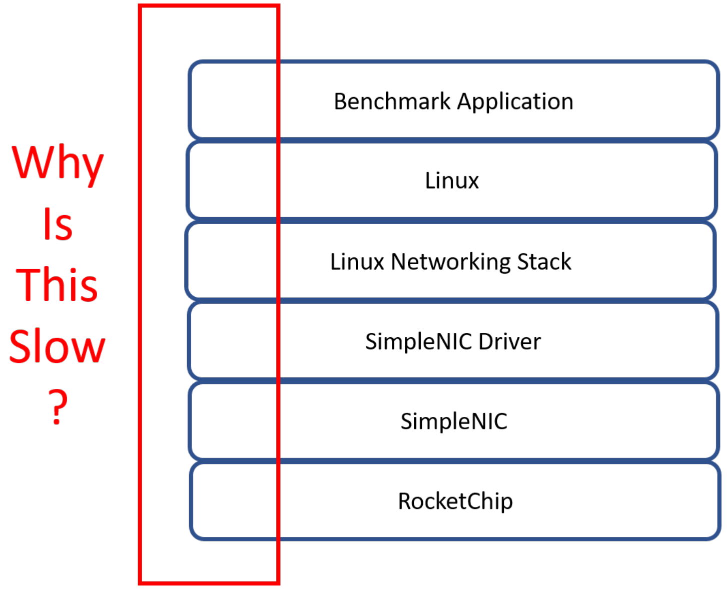
# TraceRV

- **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

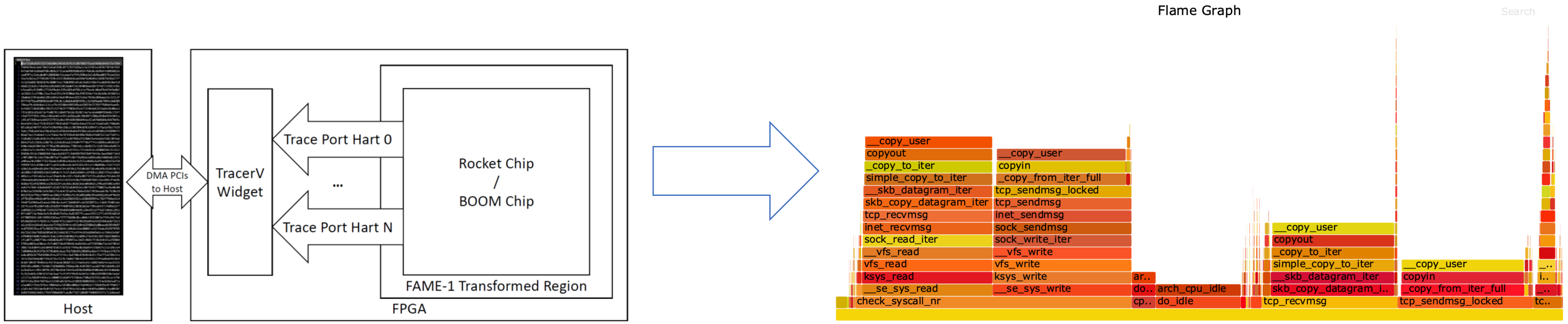
- Full trace files can be very large (100s GB – TB)
- We are usually interested only in a specific region of execution
- TraceRV can be enabled based on in-band and out-of-band triggers
  - Program counter
  - Unique instruction
  - Cycle count
- Can use the same trigger for some other simulation outputs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

config_runtime.ini

```
[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:**

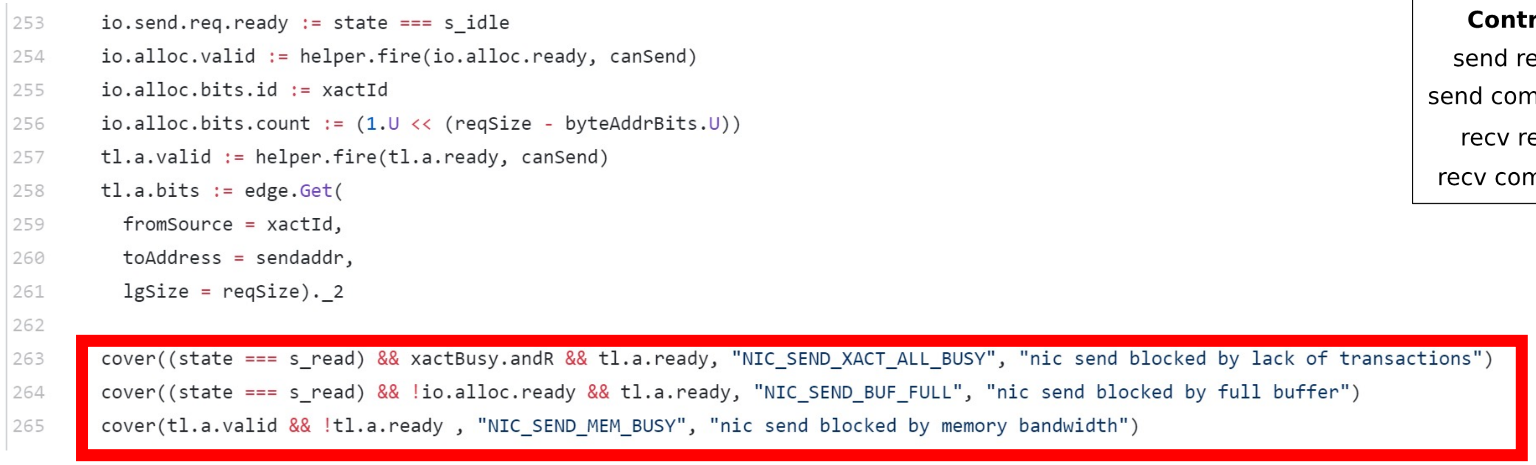
- Out-of-Band (no impact on workload execution)
- SW-centric method
- Large amounts of data

**Cons:**

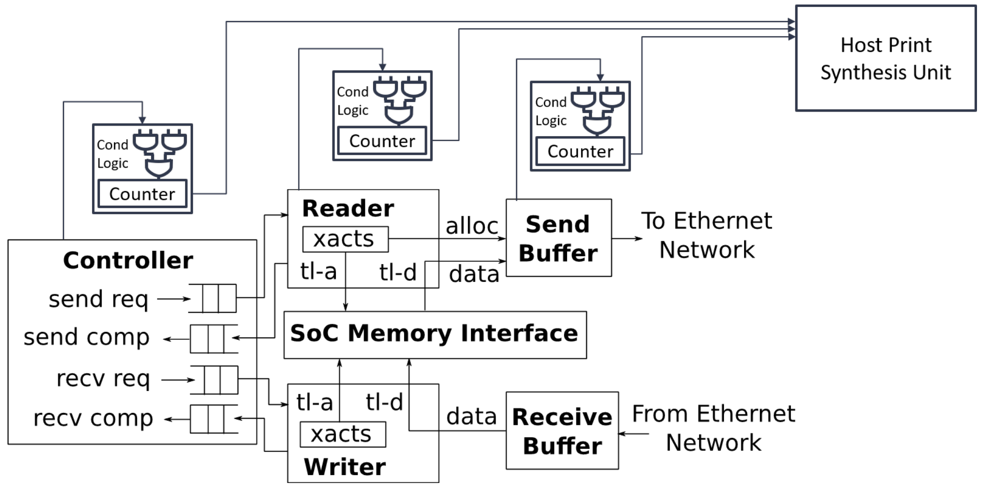
- Slower simulation performance (40 MHz)
- No HW visibility
- Large amounts of data

# AutoCounter

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



```
io.send.req.ready := state === s_idle
io.alloc.valid := helper.fire(io.alloc.ready, canSend)
io.alloc.bits.id := xactId
io.alloc.bits.count := (1.U << (reqSize - byteAddrBits.U))
tl.a.valid := helper.fire(tl.a.ready, canSend)
tl.a.bits := edge.Get(
  fromSource = xactId,
  toAddress = sendaddr,
  lgSize = reqSize)._2

cover((state === s_read) && xactBusy.andR && tl.a.ready, "NIC_SEND_XACT_ALL_BUSY", "nic send blocked by lack of transactions")
cover((state === s_read) && !io.alloc.ready && tl.a.ready, "NIC_SEND_BUF_FULL", "nic send blocked by full buffer")
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");
```

- Simle runtime read-rate configuration (`config_runtime.ini`)
  - Trade-off visibility/detail and performance

```
[autocounter]
readrate=1000000
```

## AutoCounter Example

- Example AutoCounter output file:

```
Cycle 2457999999
==============================
PerfCounter l2_misses_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sourceA: 16872407
PerfCounter l2_requests_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sinkA: 45143832

Cycle 2458999999
==============================
PerfCounter l2_misses_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sourceA: 16873445
PerfCounter l2_requests_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sinkA: 45182776

Cycle 2459999999
==============================
PerfCounter l2_misses_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sourceA: 16873752
PerfCounter l2_requests_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sinkA: 45183706

Cycle 2460999999
==============================
PerfCounter l2_misses_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sourceA: 16874798
PerfCounter l2_requests_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sinkA: 45222694

Cycle 2461999999
==============================
PerfCounter l2_misses_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sourceA: 16874798
PerfCounter l2_requests_FireSim_TestHarness_subsystem_l2_wrapper_l2_mods_0_sinkA: 45222694
```

# 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 pro
resilient low-voltage operation in
Christopher Celio, Pi-Feng Chi
Krste Asanović, David Patterson, and Bori
Hot Chips 2018

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.
  - S- and platform-level use-cases.
- Assertions are king.

21

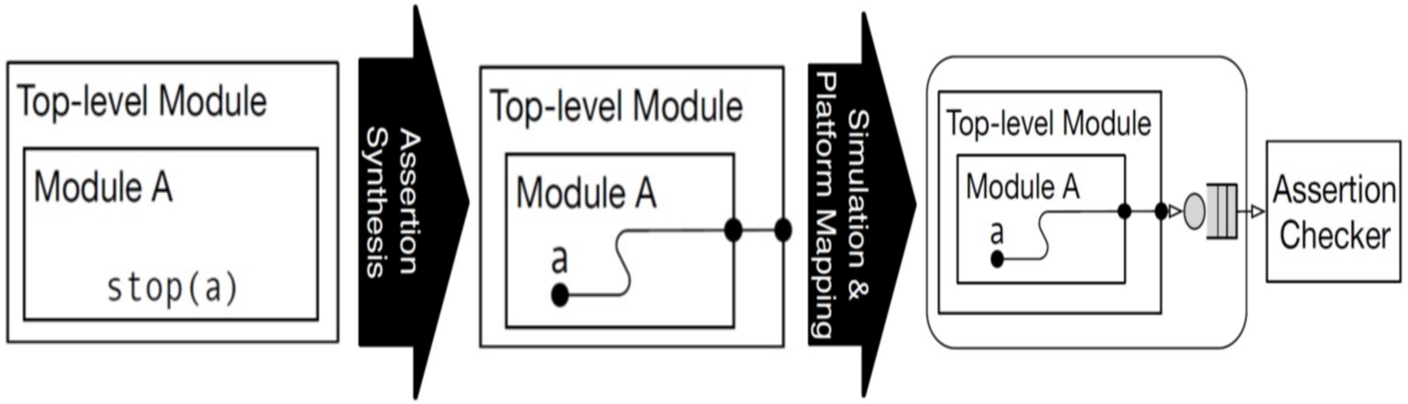
```
class Count extends Module {
  val io = IO(new Bundle {
    val en = Input(Bool())
    val done = Output(Bool())
    val cntr = Output(UInt(4.W))
  })
  // count until 10 when `io.en' is high
  val (cntr, done) = Counter(io.en, 10)
  io.cntr := cntr
  io.done := done

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

From: 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

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

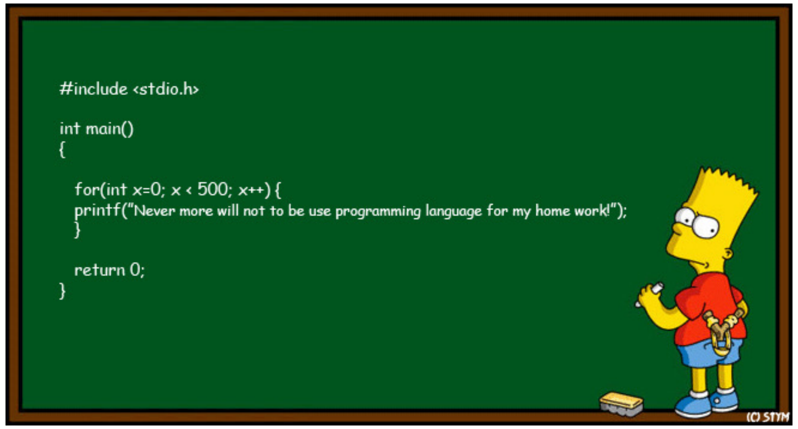
- How it looks in the UART output (while Linux is booting):

```
[    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 Printfs

- 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 printfs.
- Output includes the exact cycle of the printf event
  - Helps measure cycles counts between events



```
#include <stdio.h>

int main()
{

   for(int x=0; x < 500; x++) {
   printf("Never more will not to be use programming language for my home work!");
   }

   return 0;
}
```

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

[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

# 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 Printfs/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

# Dromajo Co-Simulation

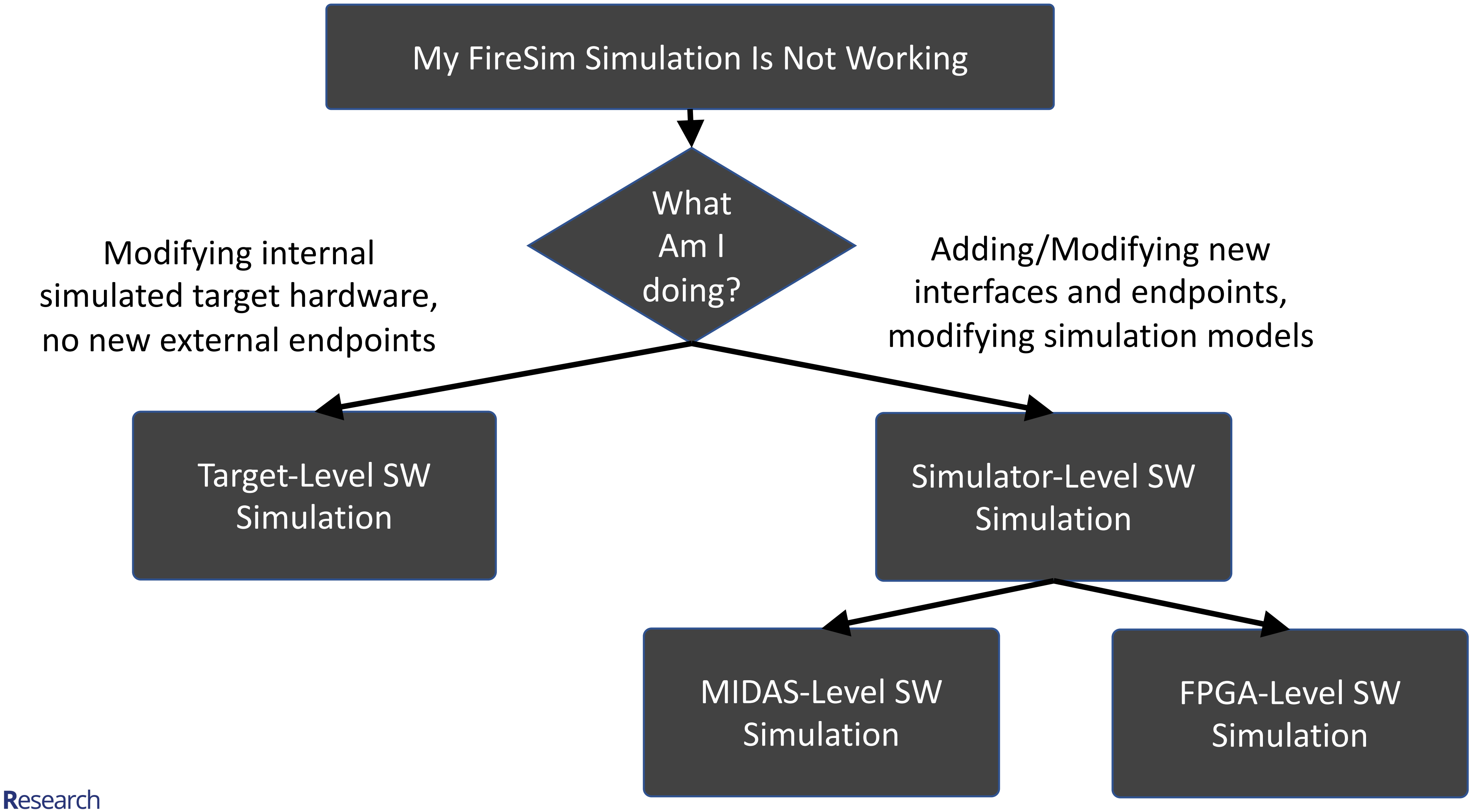
- Dromajo – RV64GC emulator designed for RTL co-simulation
- Can be used to debug BOOM in FireSim through functional co-simulation and comparison
  - Or any other design with a functional implementation in Dromajo
- Find functional bugs billions of cycles into simulations
  - Find divergence against functional golden model
  - Dump waveforms for affected signals

```
[error] EMU PC ffffffe001055d84, DUT PC ffffffe001055d84
[error] EMU INSN 14102973, DUT INSN 14102973
[error] EMU WDATA 000220d6, DUT WDATA 000220d4
[error] EMU MSTATUS a000000a0, DUT MSTATUS 00000000
[error] DUT pending exception -1 pending interrupt -1
[ERROR] Dromajo: Errored 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)

## Debugging Co-Simulation

# Debugging Using Software RTL Simulation


My FireSim Simulation Is Not Working
What Am I doing?
Modifying internal simulated target hardware, no new external endpoints
Adding/Modifying new interfaces and endpoints, modifying simulation models
Target-Level SW Simulation
Simulator-Level SW Simulation
MIDAS-Level SW Simulation
FPGA-Level SW Simulation

# Debugging Using Software RTL Simulation

## Target-Level Simulation

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

## MIDAS-Level Simulation

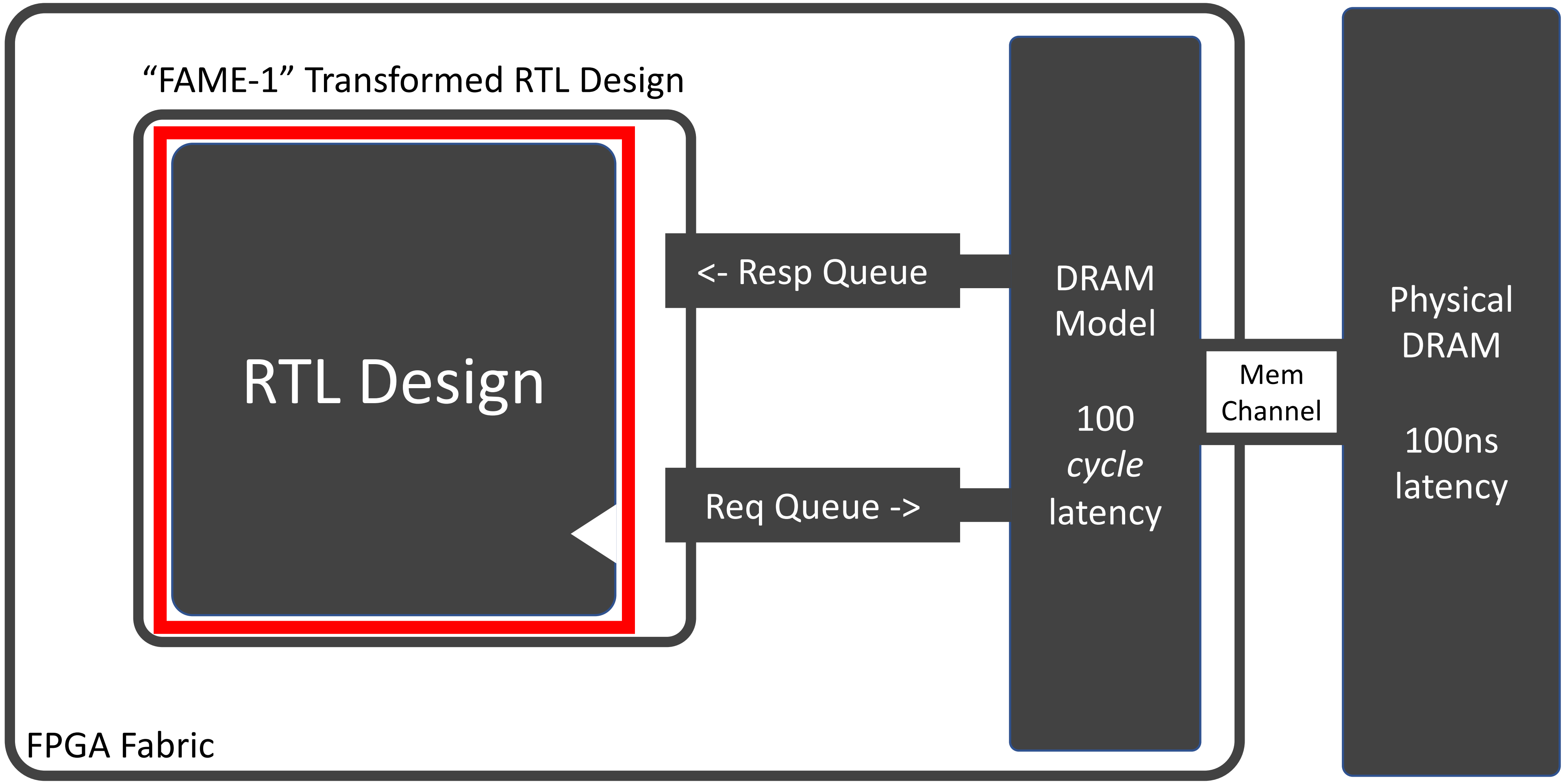
- Software Simulation
- Target Design Transformed by Golden Gate
- Host-FPGA interfaces/shell emulated using abstract models

## 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

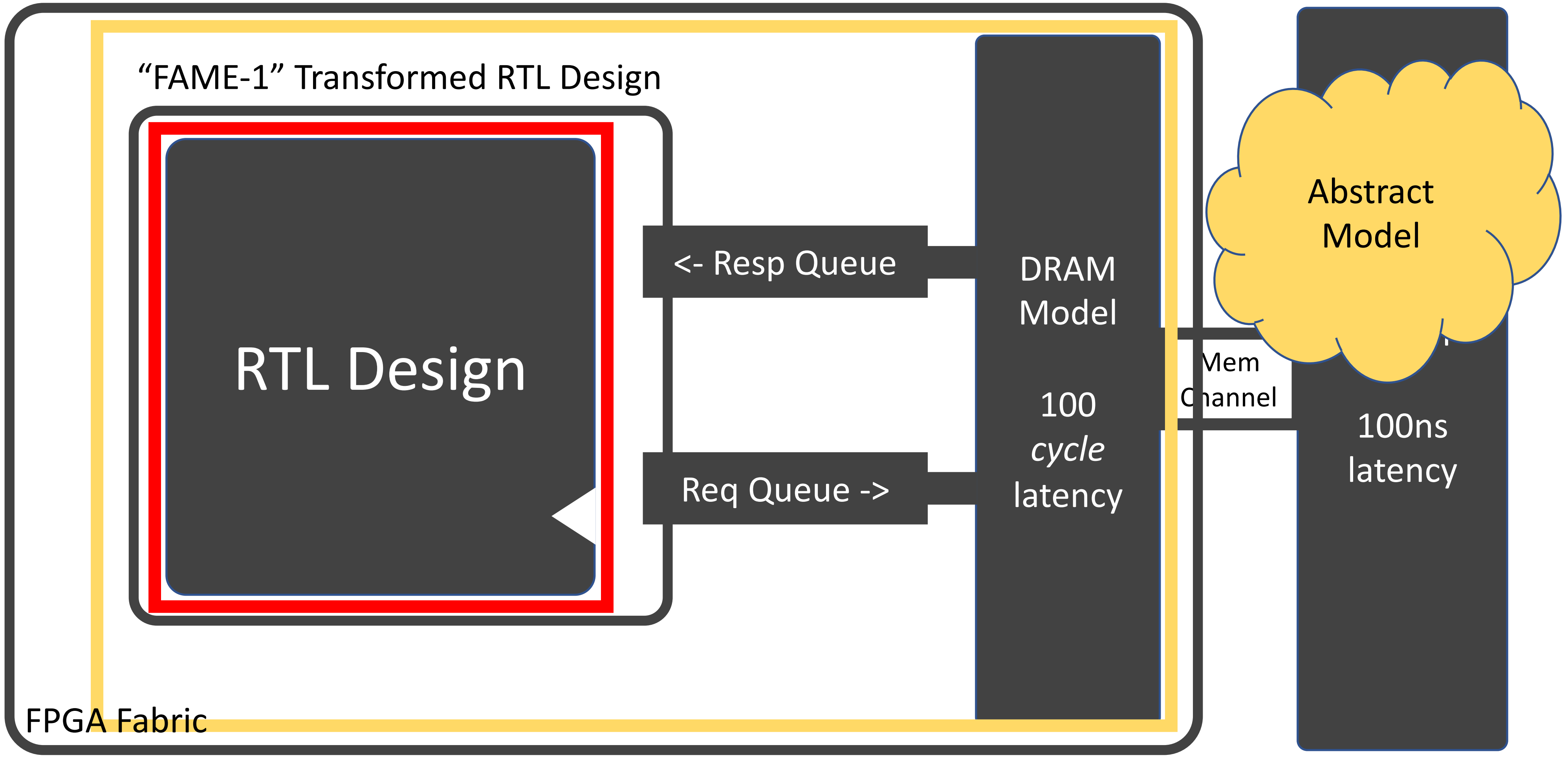
Target-Level
SW Simulation



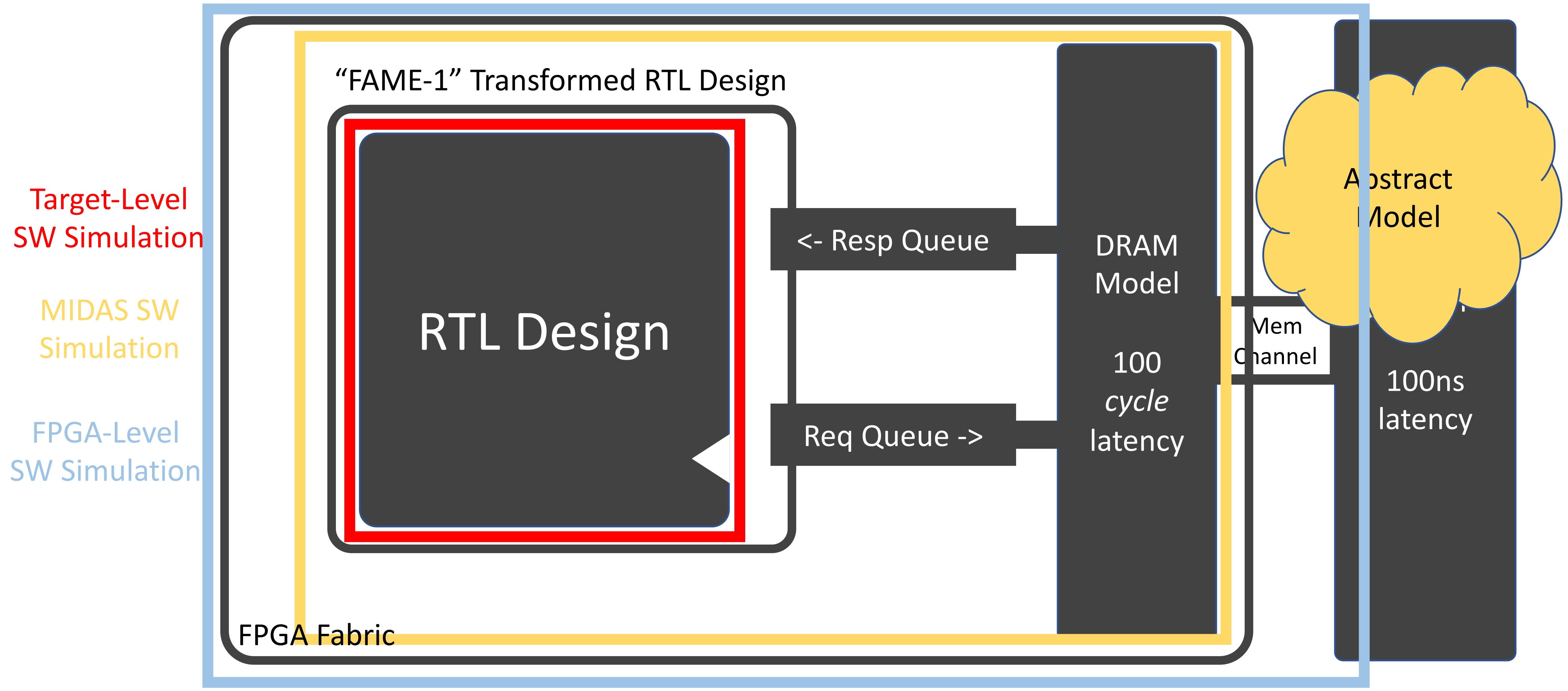
# Debugging Using Software RTL Simulation

Target-Level SW Simulation

MIDAS SW Simulation



# Debugging Using Software RTL Simulation

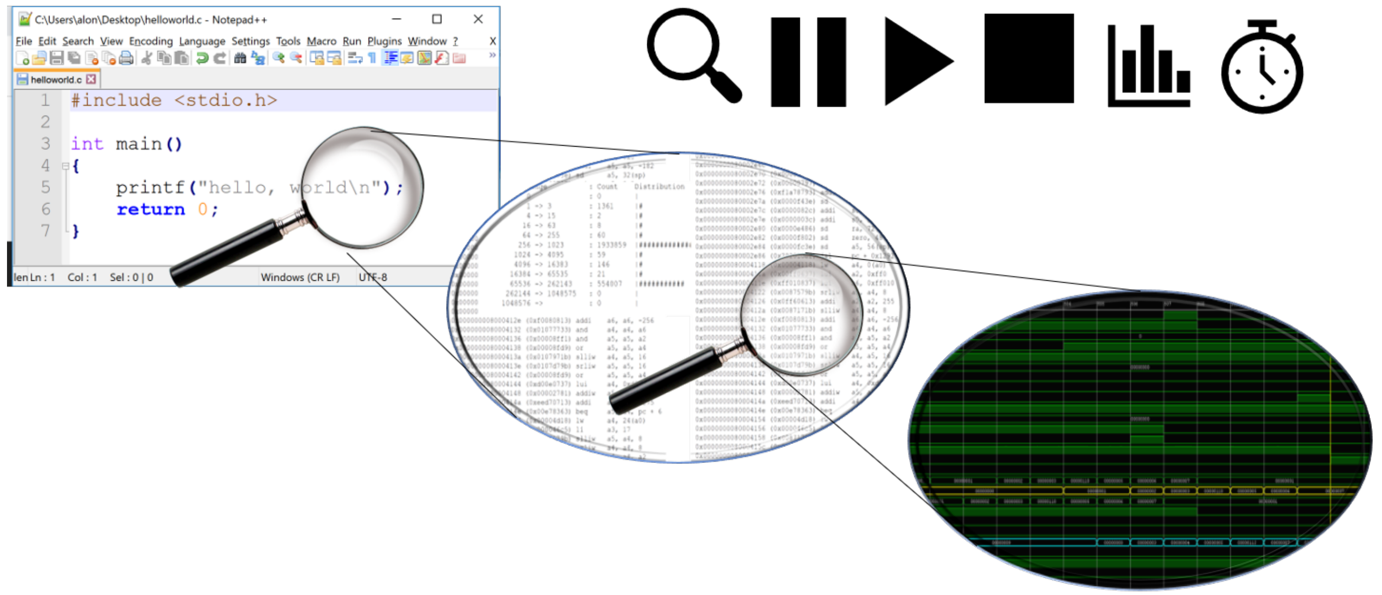


# Debugging Using Software RTL Simulation

| Level | Waves | VCS | Verilator | XSIM |
|---|---|---|---|---|
| Target | Off | ~5 kHz | ~5 kHz | N/A |
| Target | On | ~1 kHz | ~5 kHz | N/A |
| 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 |

# The FireSim Vision: Speed and Visibility

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


C:\Users\alon\Desktop\helloworld.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
helloworld.c
Windows (CR LF)
UTF-8

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

## Interactive Example

# Hands-on Synthesizable Printf Example

- We would like to observe when the SHA3 algorithm completes a round, and some details about the round. This is represented by the following code segment (https://github.com/ucb-bar/sha3/blob/master/src/main/scala/dpath.scala#L103)

```
when(io.absorb){
    state := state
    when(io.aindex < UInt(round_size_words)){
      state((io.aindex%UInt(5))*UInt(5)+(io.aindex/UInt(5))) :=
        state((io.aindex%UInt(5))*UInt(5)+(io.aindex/UInt(5))) ^ io.message_in
    }
  }
```

Berkeley Architecture Research

# Hands-on Synthesizable Printf Example

- We would like to observe when the SHA3 algorithm completes a round, and some details about the round. This is represented by the following code segment (https://github.com/ucb-bar/sha3/blob/master/src/main/scala/dpath.scala#L103)

```
when(io.absorb){
    state := state
    if(p(Sha3PrintfEnable)){
      printf(midas.targetutils.SynthesizePrintf("SHA3 finished an iteration with index %d and
message %x\n", io.aindex, io.message_in))
    }
    when(io.aindex < UInt(round_size_words)){
      state((io.aindex%UInt(5))*UInt(5)+(io.aindex/UInt(5))) :=
        state((io.aindex%UInt(5))*UInt(5)+(io.aindex/UInt(5))) ^ io.message_in
    }
  }
```

## Hands-on Synthesizable Printf Example

- We use the following build recipe for this FPGA image (in `deploy/config_build_recipes.ini`) is:

```
[firesim-singlecore-sha3-no-nic-l2-llc4mb-ddr3-print]
DESIGN=FireSim
TARGET_CONFIG=
DDR3FRFCFSLLC4MB_WithDefaultFireSimBridges_WithFireSimTestChipConfigTweaks_chipyard.Sha3RocketConfig
PLATFORM_CONFIG=WithPrintfSynthesis_F120MHz_BaseF1Config
instancetype= z1d.2xlarge
deploytriplet=None
```

# Hands-on Synthesizable Printf Example

Update our workload to copy the output printf file:

- `vim workloads/sha3-bare-rocc.json`
- Add the `synthesized-prints.out` to our simulation output

```
{
  "benchmark_name": "sha3-bare-rocc",
  "common_simulation_outputs": [
      "uartlog", "synthesized-prints.out"
  ],
  "common_bootbinary": "../../../../../generators/sha3/software/tests/bare/sha3-rocc.riscv",
  "common_rootfs": "../../../../../software/firemarshal/boards/default/installers/firesim/dummy.rootfs"}
```

## Hands-on Synthesizable Printf Example

- Setup the firesim/deploy/config_runtime.ini file:
  - Select the AGFI that was synthesized with the printf
  - Select the bare-metal SHA3 test workload
- Boot the simulation by running the following sequence of commands:

  -
    ```
    $ firesim infrasetup
    ```
    - This should take about 10 minutes

  -
    ```
    $ firesim runworkload
    ```
    - This should take about 2 minutes

```
f1_16xlarges=0
m4_16xlarges=0
f1_4xlarges=0
f1_2xlarges=1

runinstancemarket=ondemand
spotinterruptionbehavior=terminate
spotmaxprice=ondemand

[targetconfig]
topology=no_net_config
no_net_num_nodes=1
linklatency=6405
switchinglatency=10
netbandwidth=200
profileinterval=-1

defaulthwconfig=firesim-singlecore-
sha3-no-nic-l2-llc4mb-ddr3-print

[workload]
workloadname=sha3-bare-rocc.json
```

# Hands-on Synthesizable Printf Example

Output file in `deploy/results-workload/<timestamp>-sha3-bare-rocc/sha3-bare-rocc0/synthesized-prints.out`

```
CYCLE:     36086158 SHA3 finished an iteration with index  0 and message 0000000000000000
CYCLE:     36086159 SHA3 finished an iteration with index  1 and message 0000000000000000
CYCLE:     36086160 SHA3 finished an iteration with index  2 and message 0000000000000000
CYCLE:     36086161 SHA3 finished an iteration with index  3 and message 0000000000000000
CYCLE:     36086162 SHA3 finished an iteration with index  4 and message 0000000000000000
CYCLE:     36086163 SHA3 finished an iteration with index  5 and message 0000000000000000
CYCLE:     36086164 SHA3 finished an iteration with index  6 and message 0000000000000000
CYCLE:     36086165 SHA3 finished an iteration with index  7 and message 0000000000000000
CYCLE:     36086166 SHA3 finished an iteration with index  8 and message 0000000000000000
CYCLE:     36086167 SHA3 finished an iteration with index  9 and message 0000000000000000
CYCLE:     36086168 SHA3 finished an iteration with index 10 and message 0000000000000000
CYCLE:     36086169 SHA3 finished an iteration with index 11 and message 0000000000000000
CYCLE:     36086170 SHA3 finished an iteration with index 12 and message 0000000000000000
CYCLE:     36086171 SHA3 finished an iteration with index 13 and message 0000000000000000
CYCLE:     36086172 SHA3 finished an iteration with index 14 and message 0000000000000000
CYCLE:     36086173 SHA3 finished an iteration with index 15 and message 0000000000000000
CYCLE:     36086174 SHA3 finished an iteration with index 16 and message 0000000000000000
CYCLE:     36086175 SHA3 finished an iteration with index 17 and message 0000000000000000
CYCLE:     36086203 SHA3 finished an iteration with index  0 and message 0000000000000000
CYCLE:     36086204 SHA3 finished an iteration with index  1 and message 0006000000000000
CYCLE:     36086205 SHA3 finished an iteration with index  2 and message 0000000000000000
CYCLE:     36086206 SHA3 finished an iteration with index  3 and message 0000000000000000
CYCLE:     36086207 SHA3 finished an iteration with index  4 and message 0000000000000000
…
```

# Hands-on Synthesizable Printf 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

# 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 and Print Synthesis (docs)
  - AutoCounter (docs)
- FireSim Debugging and Profiling Future Vision

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