2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Finding Hidden Bugs In Deep Cycles

Advanced Debug Methodologies for Software-first System Validation

Youcef Qassid & Andy Jolley

SYNOPSYS®

accellera SYSTEMS INITIATIVE

Tutorial Overview & Agenda

- With the complexity of today's software the length of workloads to validate hardware and software has increased to 100s of billions of cycles.
- As teams adopt a software-first validation strategy, modern emulation and prototyping platforms are needed to enable the highest performance as well highly efficient debug technology.

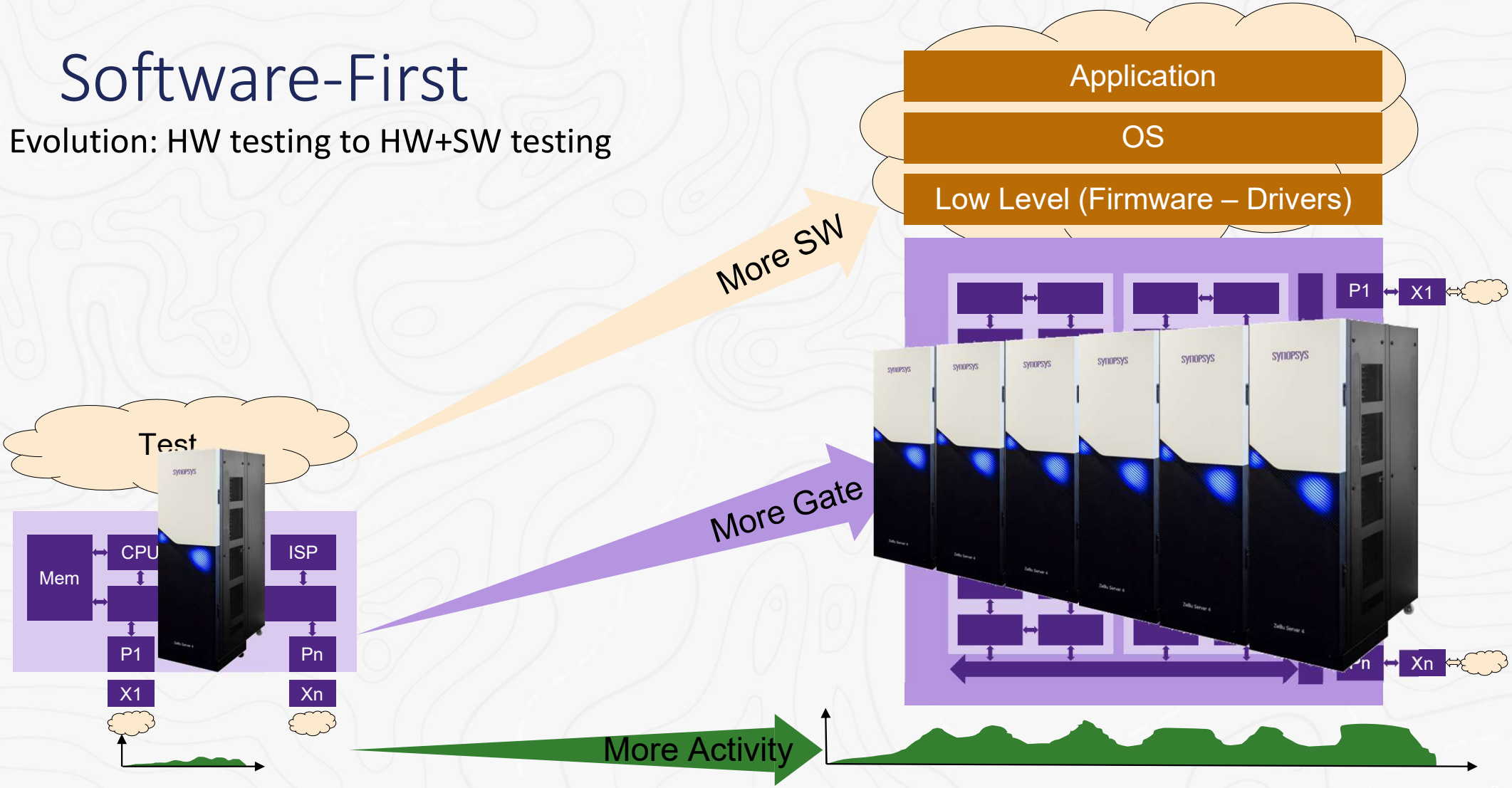
- In this 2-part tutorial, we will use a multi-processor design case study to illustrate how the latest Emulation and FPGA Prototyping Systems are both ideal platforms to achieve S/W first system validation

  - Part 1 : Debug using ZeBu® Server emulation system
  - Part 2 : Debug using HAPS®-100 FPGA prototyping

Why Software-First

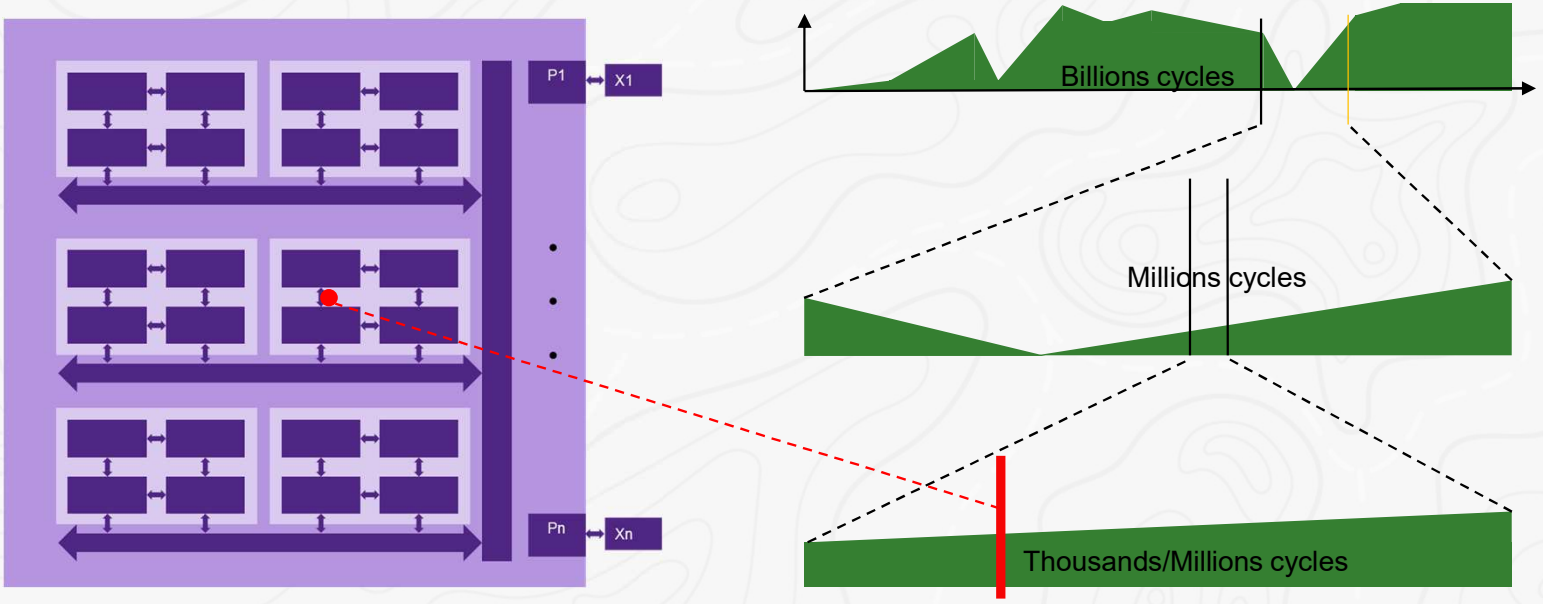
Software-First

Evolution: HW testing to HW+SW testing



Debug - High Level View

Identify Debug Window



- Monitor and Checker
- Assertions
- Key Signals Waveforms
- Full Visibility Waveforms

General Signals Monitoring – Unlimited Number of Cycles

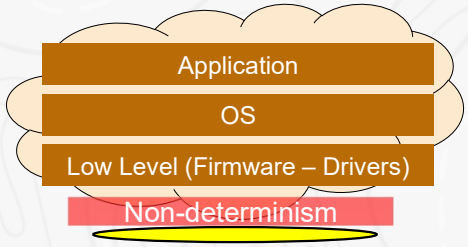
Checkers/Monitor – 1st Level RTL Debug

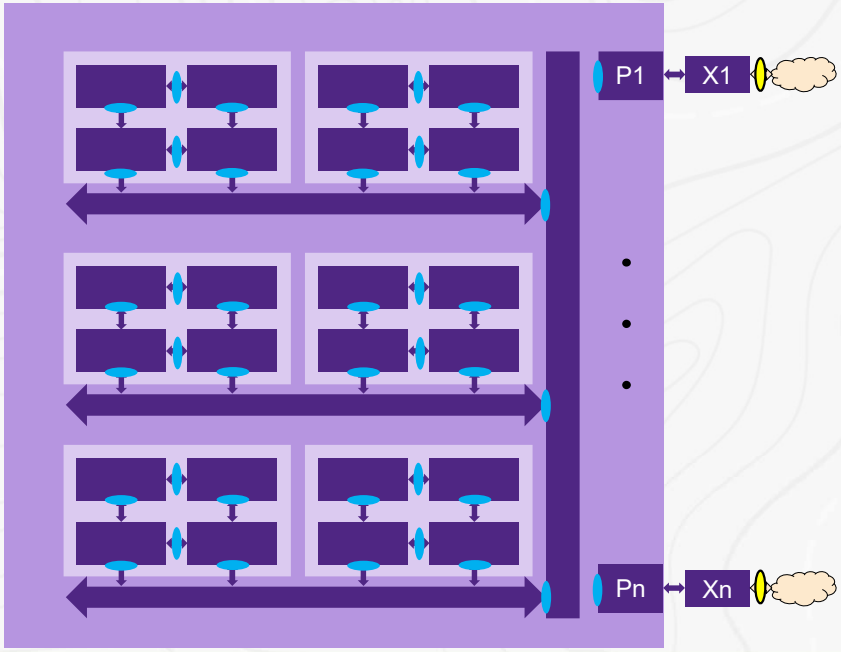
RTL Debug through Waveform

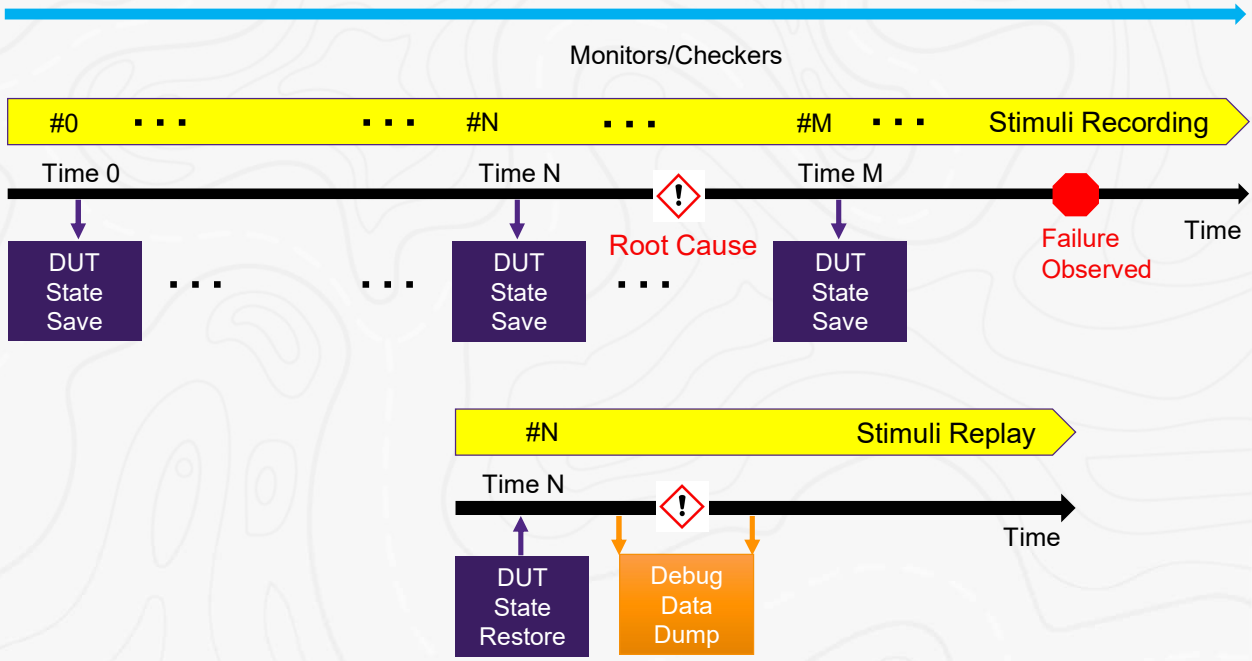
ZeBu Technologies

Monitors/Checkers and Stimuli

Running Emulation for days/weeks & Isolating Window of Debug

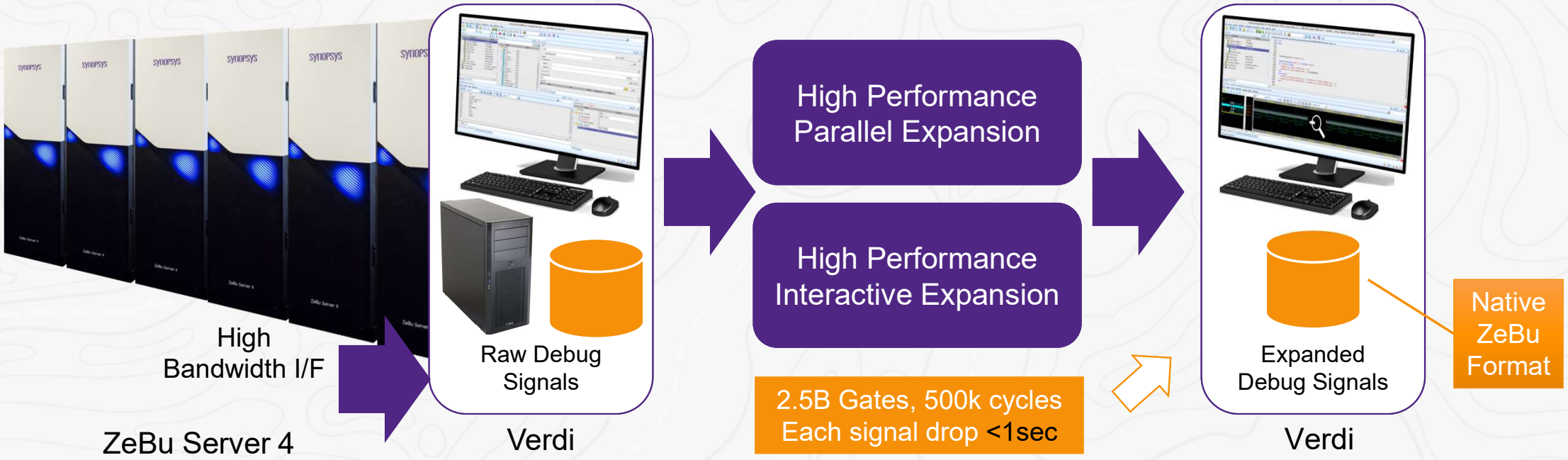






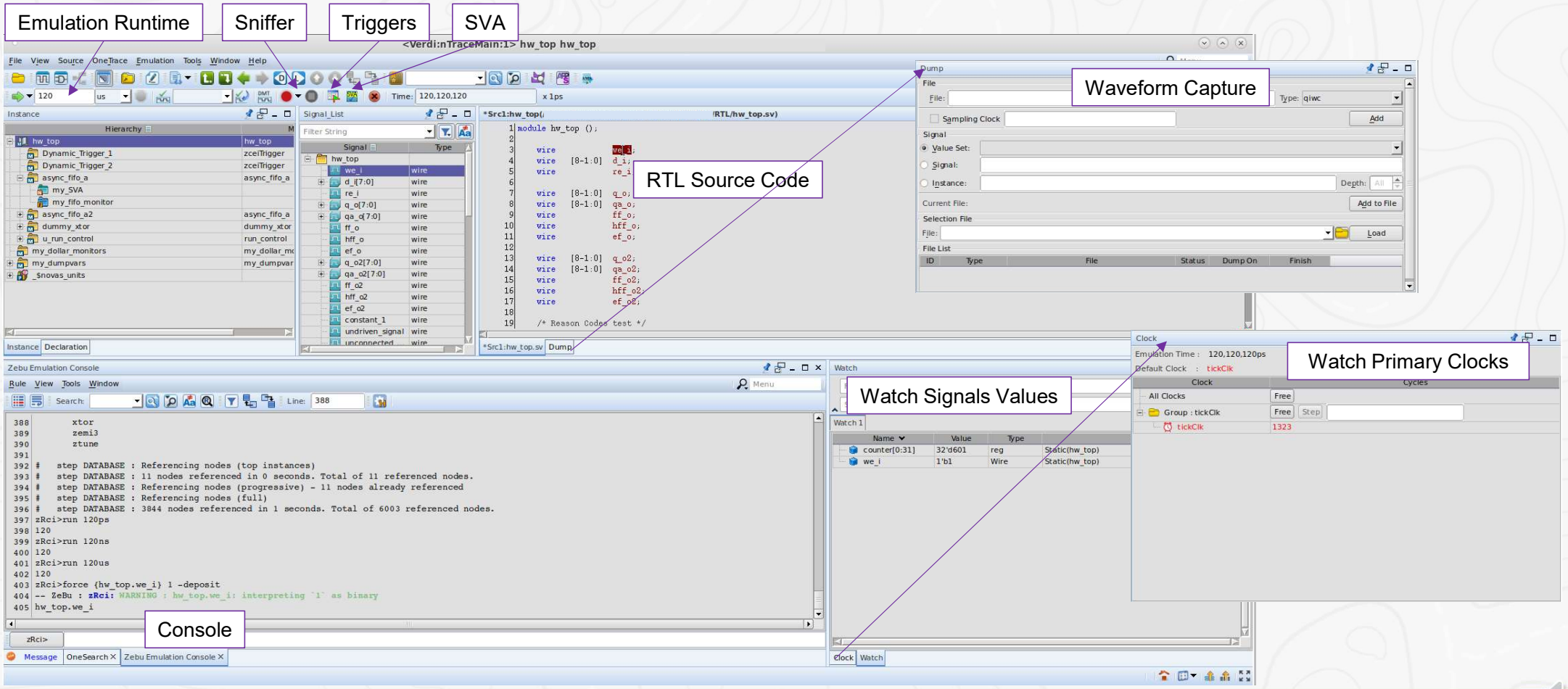
Faster Waveform Expansion and Debug with ZeBu and Verdi

Next generation tools for waveform-level debug



Scalable solution for complex billion gate SoC waveform-level debug with Verdi

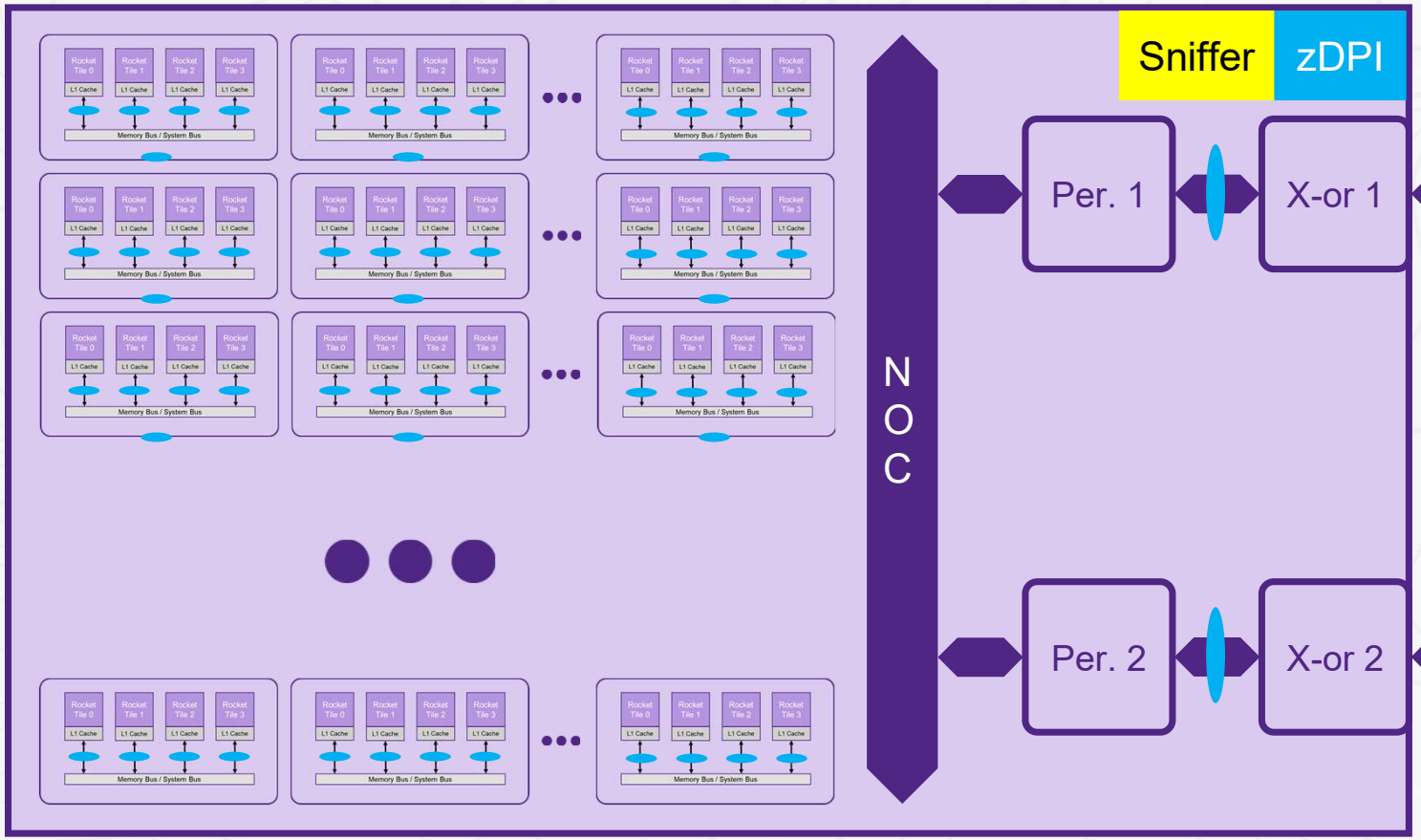
Emulation Runtime & DUT Debug with Verdi



ZeBu Case Study

Multi RISC-V Full SOC

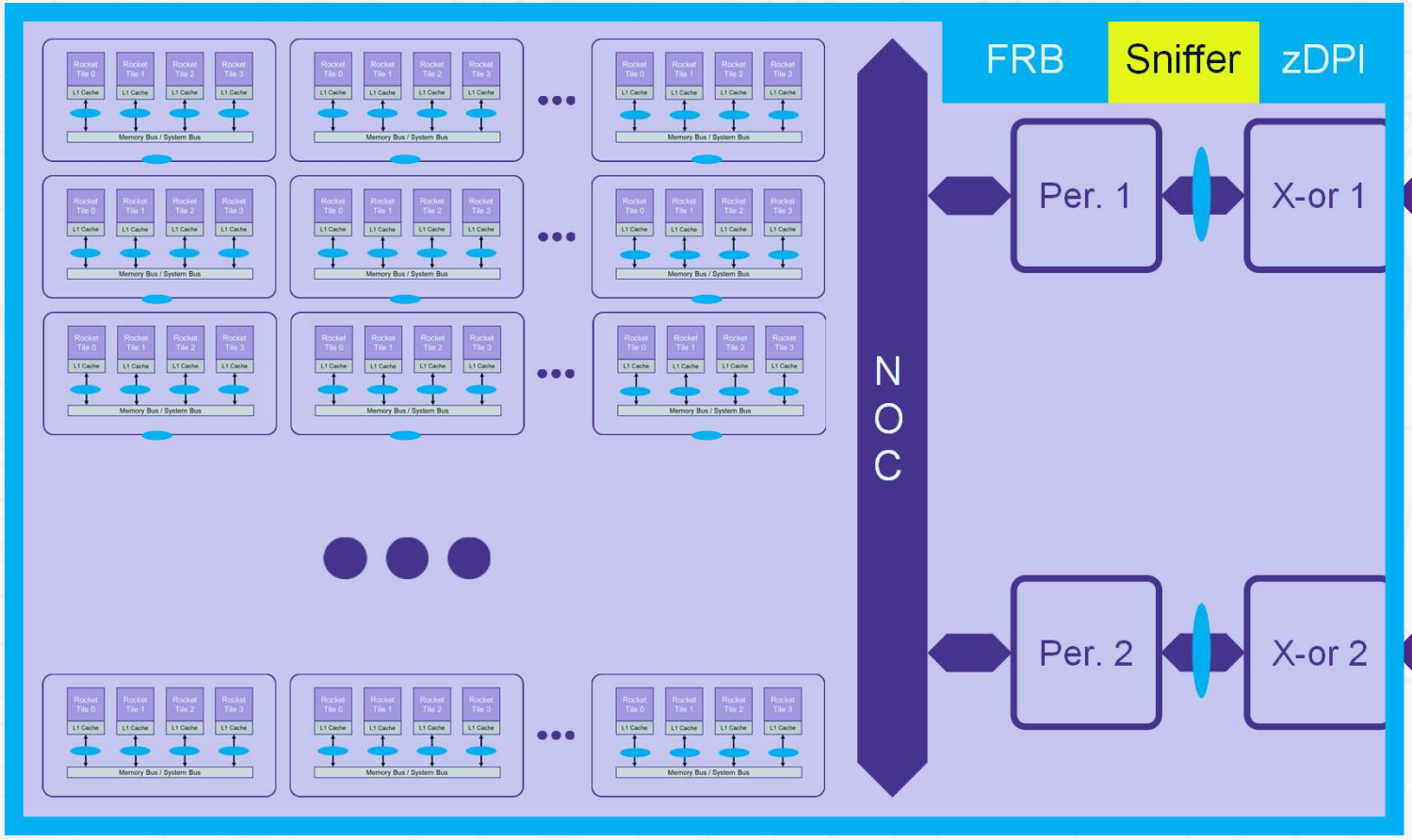
Emulation run setup



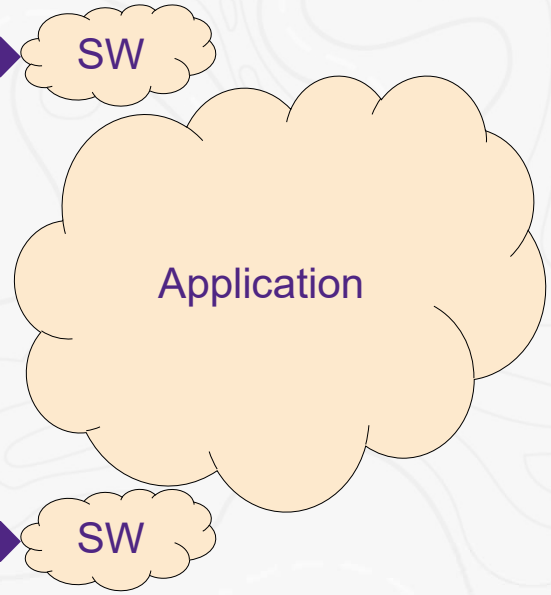
- zDPI
  - High Speed Monitor
- Sniffer
  - Saving SW Messages

Multi Risc-V Full SOC

Full Visibility

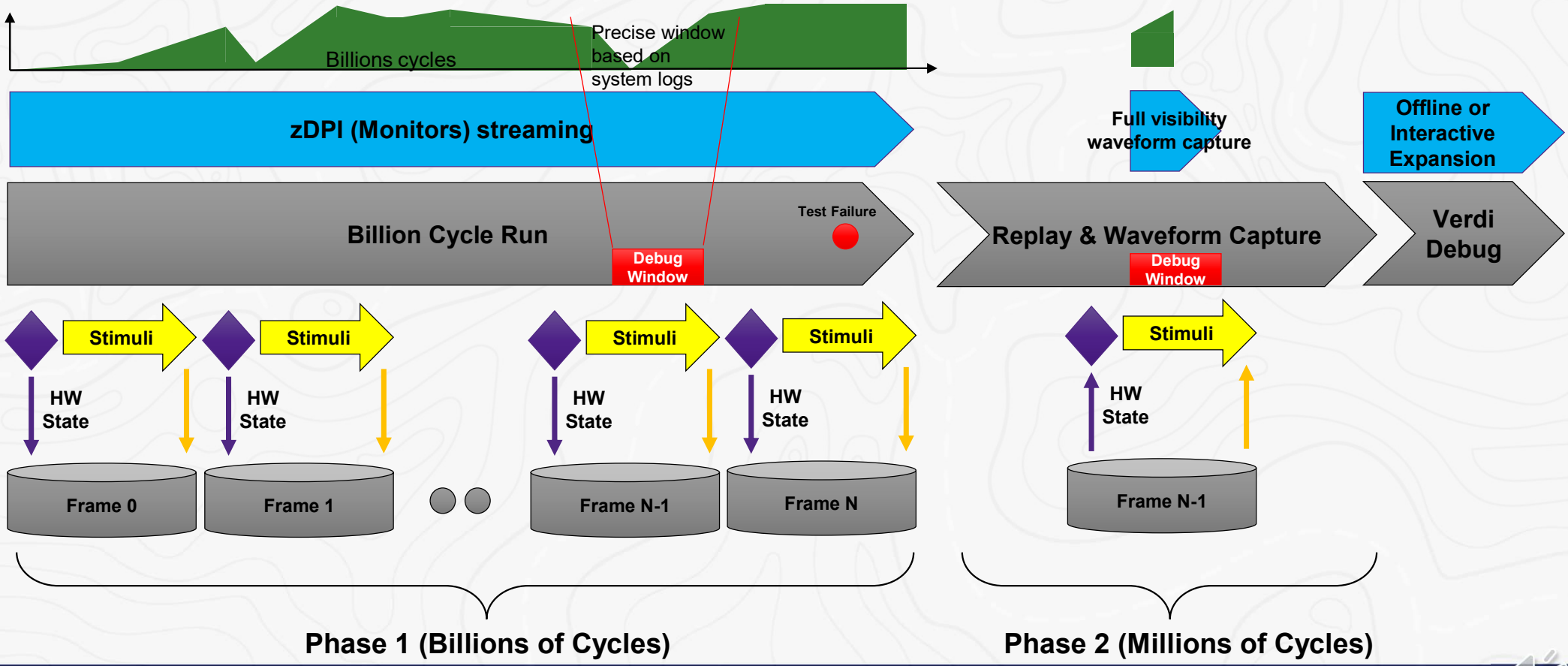


- Fast Readback



System-level Debug Using Monitors

Using highly efficient trackers/monitors to identify window for waveform level debug

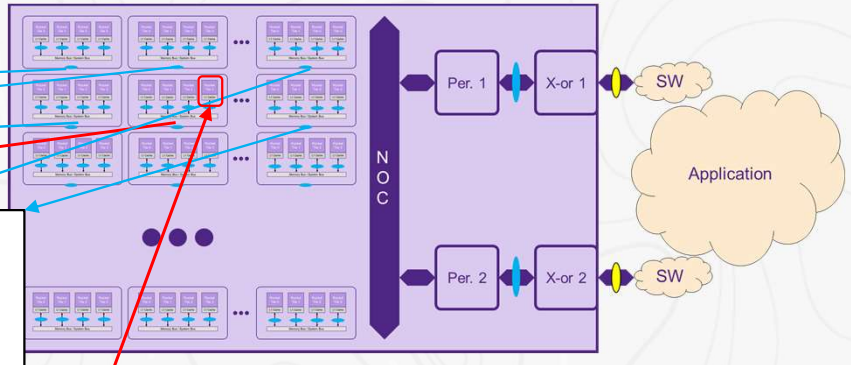


Checking Monitors Logs

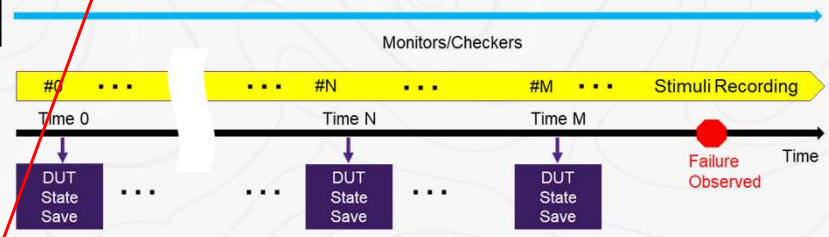
Identifying Failure Window

```
[      2325304 ][     0.000000] Zone ranges:
[      2346727 ][     0.000000]   DMA32    [mem 0x0000000080200000-0x00000000ffffffff]
[      2380422 ][     0.000000]   Normal   [mem 0x0000000090000000-0x000008ffffffffff]
[      2409473 ][     0.000000] Movable zone start for each node
[      2431044 ][     0.000000] Early memory node ranges
[      2452482 ][     0.000000]   node   0: [mem 0x0000000080200000-0x00000000ffffffff]
[      2485469 ][     0.000000] Initmem setup node 0 [mem 0x0000000080200000-0x00000000ffffffff]
```

```
[ 979174634 ]Welcome to Buildroot
[ 979213892 ]buildroot login:
```



Linux Boot not progressing after cycle 2,485,469 in RISC-V instance #3

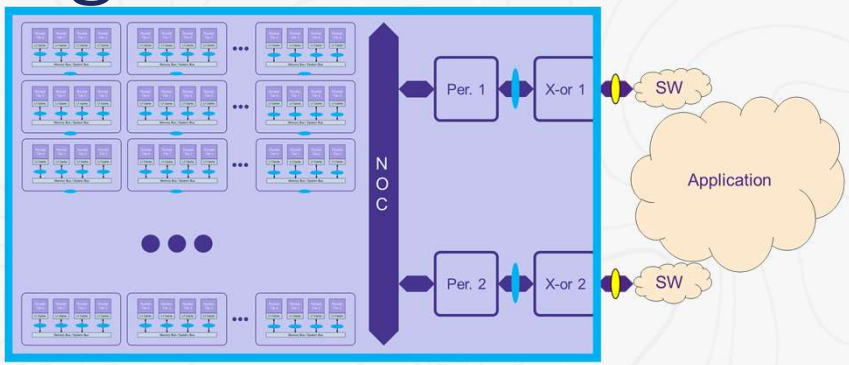



```
...
[      2519027 ] pc=e000c9d2d4 op=00e286b3 asm=unknown
                 X5 = ffffffe00fa80000  X14 = 380000
                 X13 <= ffffffe00fe00000
[      2519028 ] pc=e000c9d2d8 op=0f877713 asm=ANDI x14, x14, 000000f8
                 X14 = 380000  X24 = ffffffe00fa80003
                 X14 <= 0000000000000000
[      2519029 ] pc=e000c9d2dc op=0000cf11 asm=C.BNEZ x6, 018
                 X14 = 0  X0 = 0
```

RISC-V Core #3 hangs since cycle 2_519_029

ZeBu debug enables failure window detection in single pass during regression

Waveform Capture & RTL Debug



Monitors/Checkers

#N-1 #N #N+1 Stimuli Recording

Time N-1 Time N Debug Window

Time

DUT State Save

DUT State Save

DUT State Save

Failure Observed

Stimuli Replay

Time N

Full Visibility Capture

Time

DUT State Save

ZTDB

High Performance Parallel Expansion

High Performance Interactive Expansion

Expanded Debug Signals

Verdi

Debugging Waveforms in Verdi

Dumped: Captured in ZTDB

Computed: Simulated

Mixed: Bits having different Reason Codes

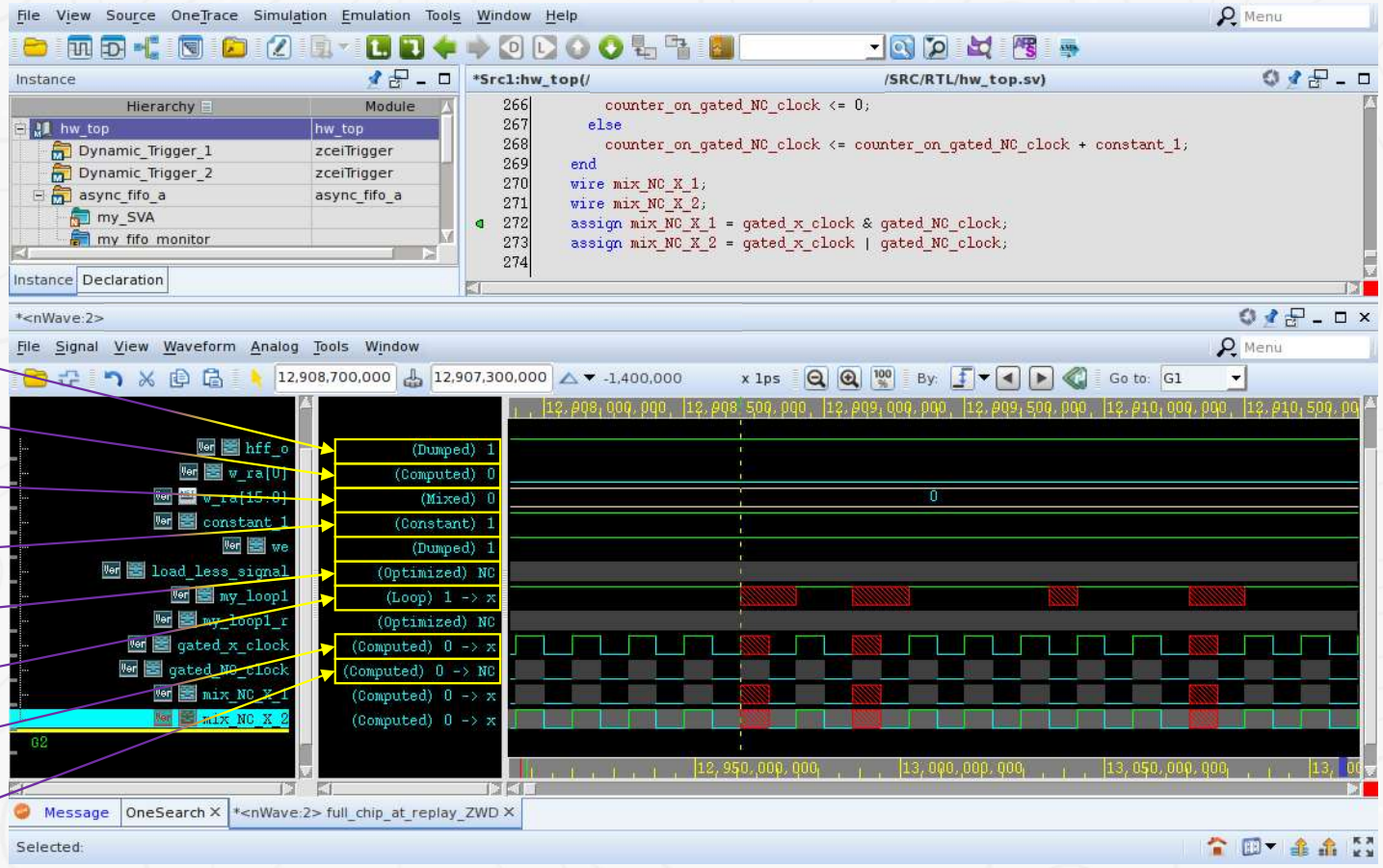
Constant: Optimized at 1 at compile

Optimized: Useless Reg. removed at compile

Loop: Simulated – X when oscillating, 0/1 when stable

Computed: Simulated – X coming from oscillating loop

Computed: Simulated – Gate is useless but still simulated thanks to Graphs (model) defined before optimizations



ZeBu Emulation Scripts

my_dumpvars.v
```
module my_dumpvars();
    initial begin: Full_Chip_VS
        (* qiwc *) $dumpvars (0, my_top_level)
    end
endmodule
```

my_vcs_script.csh
```
vlogan SRC/RTL/my_dumpvars.v
…
vcs my_top_level my_dumpvars
```

my_UTF_script.tcl
```
vcs_exec_command my_vcs_script.csh
architecture_file -filename .../zse_configuration.tcl
debug -all true
debug -offline_debug_params {INCL_XTORS=true}
debug -waveform_reconstruction_params {SIMZILLA=V2}
```

```
zCui -u my_UTF_script.tcl
```

main_run.tcl
```
start_zebu main_emulation_run_db
sniffer -auto_create  2000s
ccall   -dump_offline dpi.ztdb
run 10000000000ns
sniffer -stop
ccall -disable
finish
```

full_visibility.tcl
```
config db_path main_emulation_run_db
sniffer -restore -at 2000000ns
set fid [dump -file full_chip.ztdb -qiwc]
dump -enable -fid $fid
replay 1000000ps
dump -close -fid $fid
finish
```

```
zRci main_run.tcl --zebu-work zebu.work
zdpiReport -f list_of_dpi_function  -i dpi.ztdb/ -l ./my_fifo_monitor.so -z zebu.work |& tee run.log
zRci full_visibility.tcl --zebu-work zebu.work
zSimzilla --ztdb full_chip.ztdb --zwd full_chip_zwd
verdi -emulation --zebu-work zebu.work -ssf full_chip_zwd
```

HAPS Technologies

The Landscape for High Performance FPGA-Based Prototyping

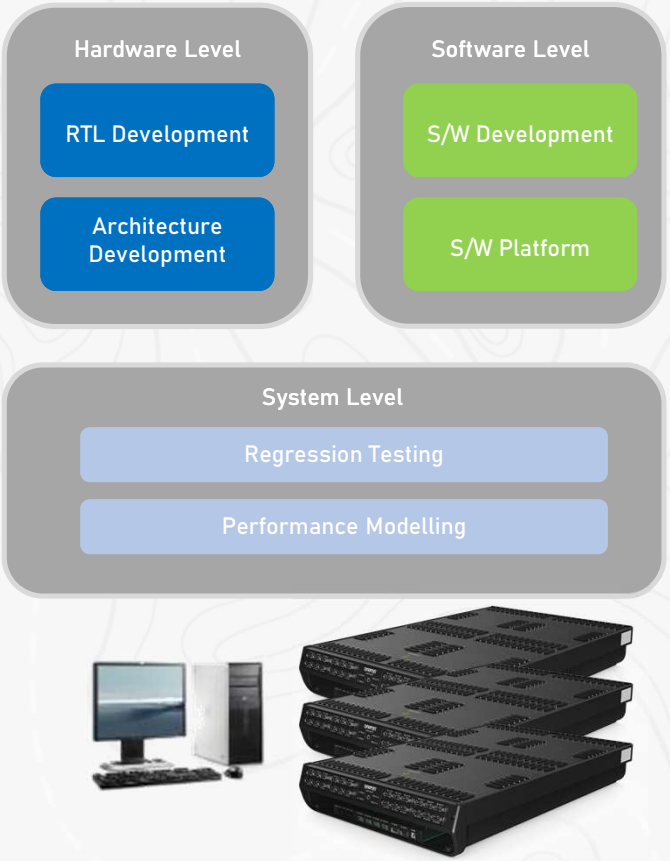
The Widening Scope

Integral Part of IP and SoC verification

- From early RTL debugging to SoC performance modelling …. and everything in between

Primary requirements for FPGA Prototyping

- High Performance
  - Peta cycles for verification coverage
- High Capacity, Scalable and Flexible
  - Growing IP and SoC sizes
  - Expanded verification tasks
- Visibility Capabilities
  - High capacity and at-speed debug
  - Highest visibility
  - Flexibility to locate the hardest to find bugs

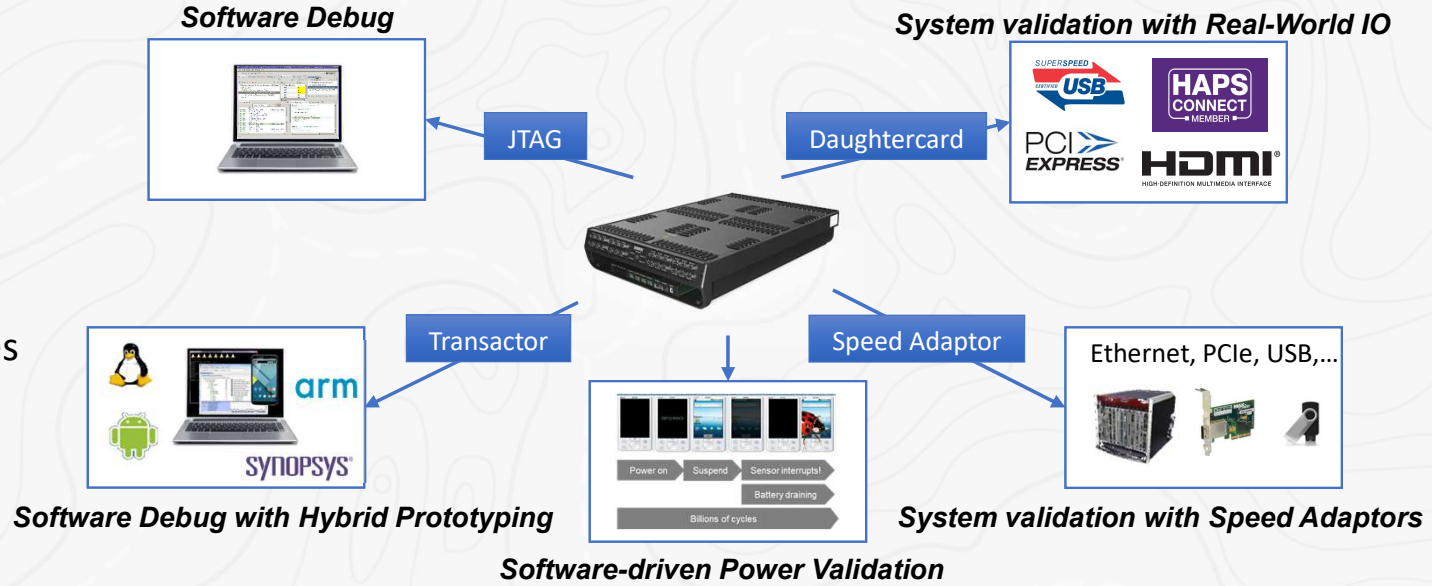


The Landscape for High Performance FPGA-Based Prototyping

The Widening Scope

Prototyping systems offer multiple verification opportunities

- Real-world interfaces for interface validation
  - IP level of SoC level
- Links to software debuggers
  - Software development
- High-speed transactor-based interfaces
  - Hybrid Prototyping
- Speed adaptors for system validation



Highest Capacity and Prototyping Performance with HAPS-100

Industries Highest Performance and Most Scalable Prototyping System

- Scalable to 1B+ gates at 10MHz

Built-In Debug for High Performance / Capacity Visibility

- Dedicated Debug Circuitry and Memory
  - Minimal Intrusion to the DUT

High Performance Host Interfaces

- High-Capacity Data transfer using USB3 and QSFP Interfaces

Designed for Desktop or Data Center Installations

- Single Form Factor suits all scalable requirements

HAPS-100

Debug Capabilities for Deep-Cycle Visibility

Get to the debug window of interest and provide database for Verdi debug

Multiple visibility options

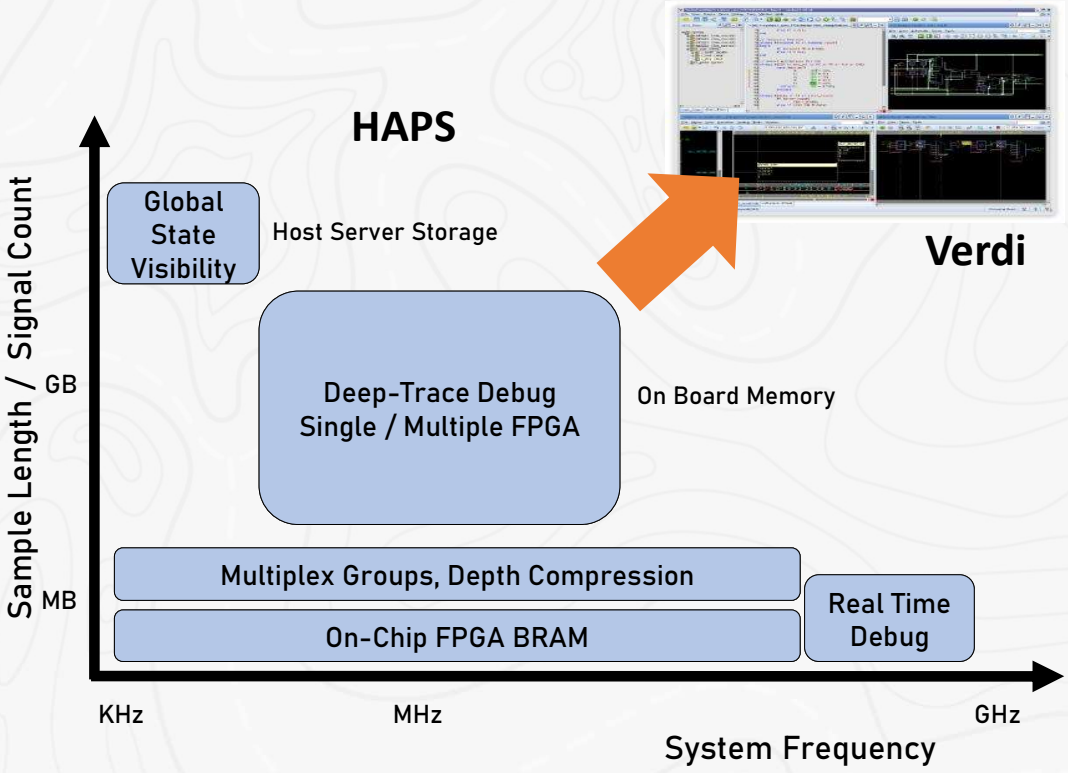
- Performance, capacity and intrusion into the DUT
- Real-time, at-speed debugging, full state capture

Increased visibility capacity and flexibility

- Greater event detection and capture resolution
- Eliminate re-spins to capture the necessary scope

Different technologies combine to great effect

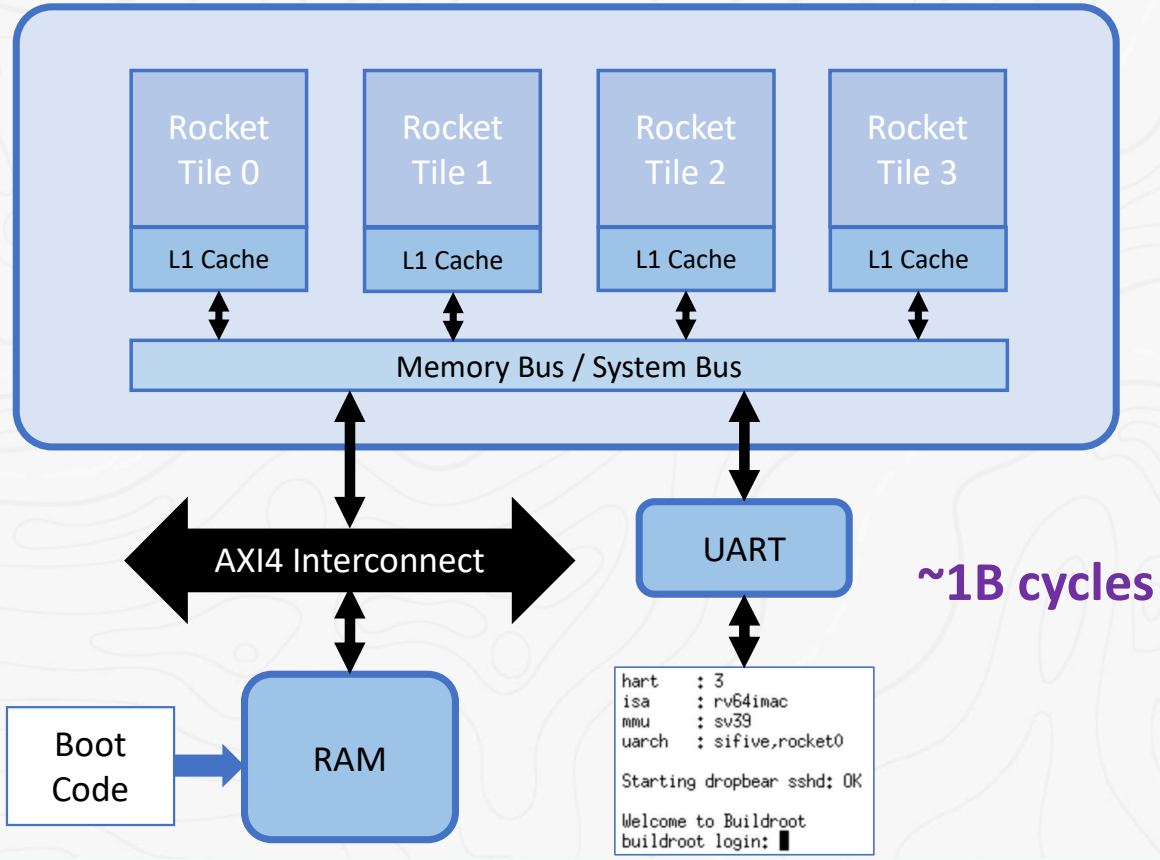
- First event detection at full speed
- Higher capacity visibility at reduced speeds
- Complete state capture at controlled speeds



HAPS Case Study

Case Study : RISC-V Based Processor SoC

Architecture Overview



~1B cycles

DUT Architecture

- RISC-V Architecture with 4-Core Processor

Software content

- Linux Boot ~ 1 Billon Cycles

Prototyping Requirements

- Single RISC-V In Single Xilinx VU19PFPGA
  - RAM to be replaced with external DDR4
  - Maximum system performance for 1B cycles
- Stand Alone DUT
  - Self stimulating from Boot Code
  - Monitor UART : Real World or Transactor Based
  - Enable debug to detect and capture activity

Prototyping Preparation : Stand-Alone DUT @ 100MHz

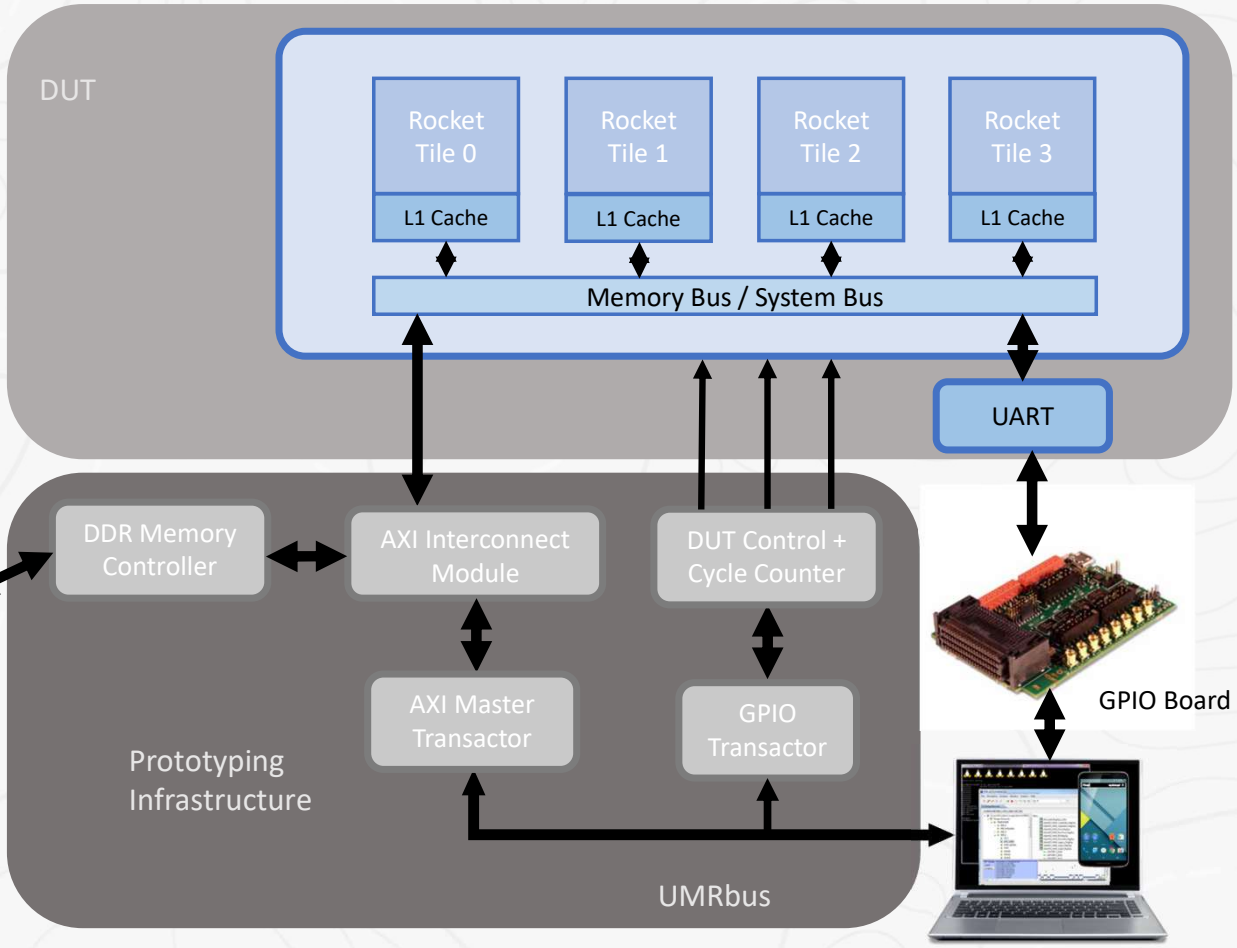
FPGA Prototyping Infrastructure

- Addition of modules for standalone prototyping
  - AXI Interconnect and DDR Memory Controller
    - To replace DUT RAM
  - AXI Transactors for Boot Code Upload
    - “Backdoor” Load Approach
  - GPIO Transactor for DUT Control
    - Clocks, Reset and Monitor
    - Cycle counter to track when error occurs

- Addition of HAPS Daughter Boards
  - DDR for Boot Code Store
  - GPIO for UART to Host PC

DUT Clocked @ 100MHz

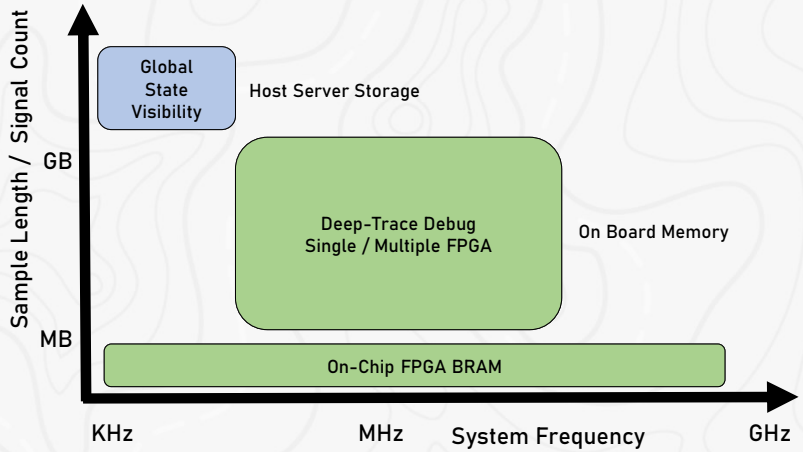
HAPS DDR Daughter Board



Prototyping Preparation : Debug Enablement

Enable multiple Debug Capabilities

- BRAM and DTD based Debug to monitor AXI Transactions
  - On-Chip BRAM for AXI Traffic between DUT and DDR3 Memory
  - Off-Chip DTD for all AXI Traffic between XTORs, DUT and DDR3 Memory
    - Debug capacity available to maximize visibility



RTL $dumpvars used to select debug signals

```
$dumpvars(1,    rocket_system_HAPS100.M1_AWREADY[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWVALID[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWID[1*AXI_M_ID_WIDTH-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWADDR[M_AXI_AW-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWLEN[7:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWSIZE[2:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWBURST[1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWLOCK[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWCACHE[3:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWPROT[2:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWQOS[3:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWREADY[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_AWVALID[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_WDATA[M_AXI_DW-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_WSTRB[M_AXI_DW/8-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_WLAST[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_BREADY[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_BVALID[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_BID[1*AXI_M_ID_WIDTH-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_BRESP[1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARREADY[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARVALID[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARID[1*AXI_M_ID_WIDTH-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARADDR[M_AXI_AW-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARLEN[7:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARSIZE[2:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARBURST[1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARLOCK[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARCACHE[3:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARPROT[2:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_ARQOS[3:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_RREADY[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_RVALID[0]);
$dumpvars(1,    rocket_system_HAPS100.M1_RID[1*AXI_M_ID_WIDTH-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_RDATA[M_AXI_DW-1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_RRESP[1:0]);
$dumpvars(1,    rocket_system_HAPS100.M1_RLAST[0]);
```

Define BRAM and DTD configuration

```
device jtagport umrbus
iice new {idxA} -type regular
iice controller -iice {idxA} none
iice controller -iice {idxA} statemachine
iice controller -iice {idxA} -triggerstates 4
iice controller -iice {idxA} -triggerconditions 8
iice controller -iice {idxA} -counterwidth 8
iice sampler -iice {idxA} haps100_DTD_builtin
iice sampler -iice {idxA} -depth 5000000
iice sampler -iice {idxA} -pipe 3
iice sampler -iice {idxA} -always_armed 1

device jtagport umrbus
iice new {iice_bram} -type regular
iice controller -iice {iice_bram} statemachine
iice controller -iice {iice_bram} -triggerstates 4
iice controller -iice {iice_bram} -triggerconditions 8
iice controller -iice {iice_bram} -counterwidth 8
iice sampler -iice {iice_bram} -depth 512
iice sampler -iice {iice_bram} -pipe 3
iice sampler -iice {iice_bram} -always_armed 1

iice clock -iice {idxA} -edge positive {rocket_system_HAPS100.M3_AXI_ACLK}
iice clock -iice {iice_bram} -edge positive {rocket_system_HAPS100.M3_AXI_ACLK}
```

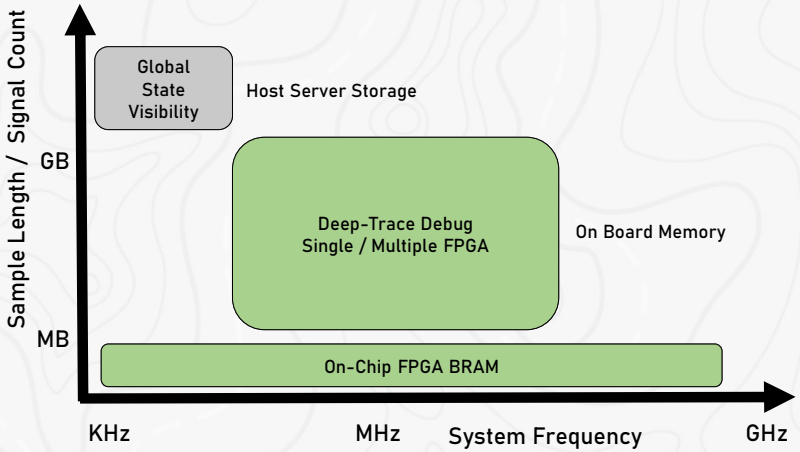
Check Resource Usage and Sample Rates from Implementation Logs

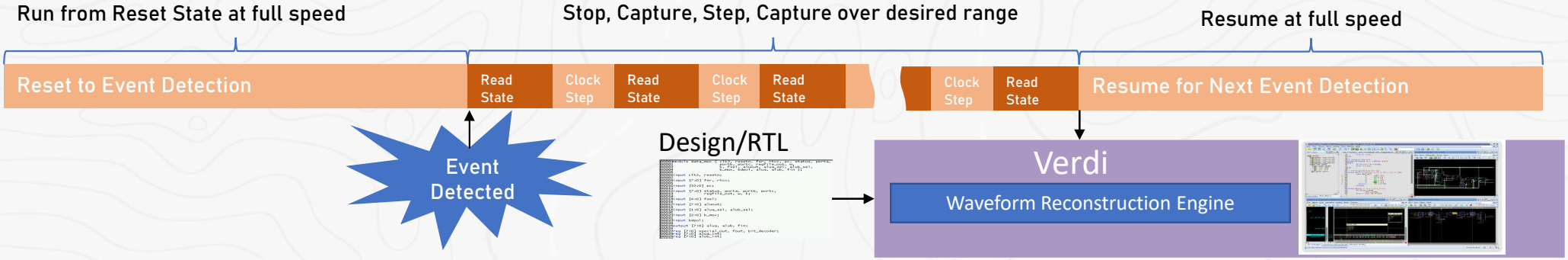
```
Current instrumentation information:    IICE=idxA
       FPGA=FB1.uA
       Total instrumentation in bits: Sample Only 0, Trigger Only 0, Sample and trigger 282
       Instrumentation in bits:
       Sample Only 0, Sample and trigger 282. All sampled 282 of 8192 signals allowed for DDR4 memory type
       Maximum sample clock frequency: 160.0000 MHz
   IICE=iice_bram
       FPGA=FB1.uA
       Total instrumentation in bits: Sample Only 0, Trigger Only 0, Sample and trigger 844
       Instrumentation in bits:
       Sample Only 0, Sample and trigger 844
       Maximum sample clock frequency is same as maximum user clock frequency (as reported by P&R)
```

Prototyping Preparation : Debug Enablement

Enable multiple Debug Capabilities

- Global State Visibility (GSV) – Synchronous Mode
  - Platform is run at speed until an event is detected
    - Event can come from multiple sources to initiate the capture
  - DUT is stopped, registered state is read, stepped and repeated over scope
    - Can capture 100k cycles in 1 hour
    - Database written for Verdi debug analysis
  - Requires Insertion of Clock Control Module
    - Automated via simple scripting in ProtoCompiler Flow





Prototyping Debug Flow : Initial Error Detect Run

Check for Error Condition on Initial Run

- Upload the Boot Code to the external DDR device via the Transactors – “Back Door” approach
  - Rocket System DUT is held in a reset state
  - Boot Code Uploaded and re-read to allow for off-line check

```
INFO: Will try to upload memory file ./memory_files/mem64b_linux.hex
INFO   : Initializing FLEXlm ...
INFO   : Checking out HAPS-100-AMBA-XTOR-MASTER license ...
INFO   : Done
INFO: Opened memory file ./memory_files/mem64b_linux.hex for reading
INFO: Found Start Address at: 2147483648
INFO: Found 2154814 memory lines
INFO: Starting memory upload
INFO: Starting memory download
INFO: Dumping read memory to read_mem.txt
INFO: script has terminated. Please compare the uploaded memory from ./memory_files/mem64b_linux.hex with the memory downloaded to read_mem.txt
Tcl_XACTORS_TaCleanup
  0
INFO   : Checking in HAPS-100-AMBA-XTOR-MASTER license ...
```

- Release DUT Reset and check for output from the UART

```
[18:59:44] [ 000002123628 ][    0.000000] bootconsole [early0] enabled^M
[18:59:45] [ 000002148804 ][    0.000000] Initial ramdisk at: 0x       (ptrval) (9929728 bytes)^M
[18:59:56] [ 000002325314 ][    0.000000] Zone ranges:^M
[18:59:56] [ 000002346737 ][    0.000000]   DMA32    [mem 0x0000000080200000-0x00000000ffffffff]^M
[18:59:56] [ 000002380432 ][    0.000000]   Normal   [mem 0x0000000090000000-0x00000fffffffffff]^M
[18:59:57] [ 000002409483 ][    0.000000] Movable zone start for each node^M
[18:59:57] [ 000002431054 ][    0.000000] Early memory node ranges^M
[18:59:58] [ 000002452492 ][    0.000000]   node   0: [mem 0x0000000080200000-0x00000000ffffffff]^M
[18:59:58] [ 000002485479 ][    0.000000] Initmem setup node 0 [mem 0x0000000080200000-0x00000000ffffffff]^M
```

  - See error state occur after cycle 2,485,479 from reset
    - 25mS to reach error condition @ 100MHz

UART locked at Cycle 2,485479

Prototyping Debug Flow : Debug Database Capture Run

Configure and Initiate the On-Chip Debug and GSV Capture

- Debug run with on-Chip BRAM Debug trigger at Cycle Count minus 50,000 Cycles

```
set current_script_dir [file dirname [file normalize [info script]]]
project -load $current_script_dir/work/Rocket_System_HAPS100/runtime_test_egsv_enable/system/debug/debug.prj

iice current {iice_bram}

watch enable -condition 0 -language verilog  {rocket_system_HAPS100.dut_reset} {1'b1} {1'b0}
watch enable -condition 1 -language verilog  {rocket_system_HAPS100.Cycle_Count} {32'h249F00}

statemachine clear 0
statemachine addtrans -from 0 -to 1 -cond {c0}
Transition 0 -> 1 when 'c0' added
statemachine clear 1
statemachine addtrans -from 1 -to 1 -cond {c1} -trigger
Transition 1 -> 1 when 'c1' added

run -iice {iice_bram}

write fsdb {$current_script_dir/work/Rocket_System_HAPS100/runtime_test_egsv_enable/system/debug/iice_debug.fsdb} -iice iice_bram
```

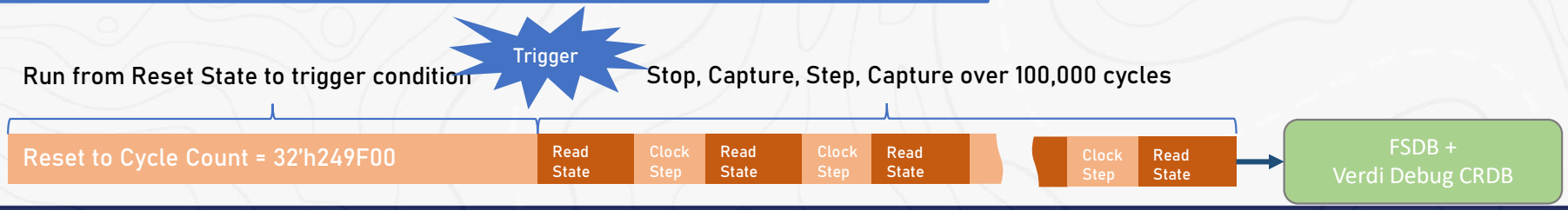
Condition 0 : DUT Reset de-assert
Condition 1 : Cycle Count = 2,400,000

State Machine Based Triggering
Condition 0 followed by Condition 1

- Global State Visibility armed to capture 100,000 cycles on trigger condition

```
set current_script_dir [file dirname [file normalize [info script]]]
project -load $current_script_dir/work/Rocket_System_HAPS100/runtime_test_egsv_enable/system/readback/debug.prj

readback set -clear
readback set -iice {Readback} -fpga {mb_1.uA} -llfile $current_script_dir/work/Rocket_System_HAPS100/runtime_test_egsv_enable/system/readback/FB1_uA.ll"
readback set -apply

set readback_cnt 1000000

readback stream_waveform -iice Readback -type FSDB readback_stream_$readback_cnt.fsdb

run -iice Readback -readback $readback_cnt -stepping 1 -wait_for_trigger extn -release_clock 1 -wait
```

Capture 100,000 Cycles upon Trigger detect



Off-line Verdi Debug Environment

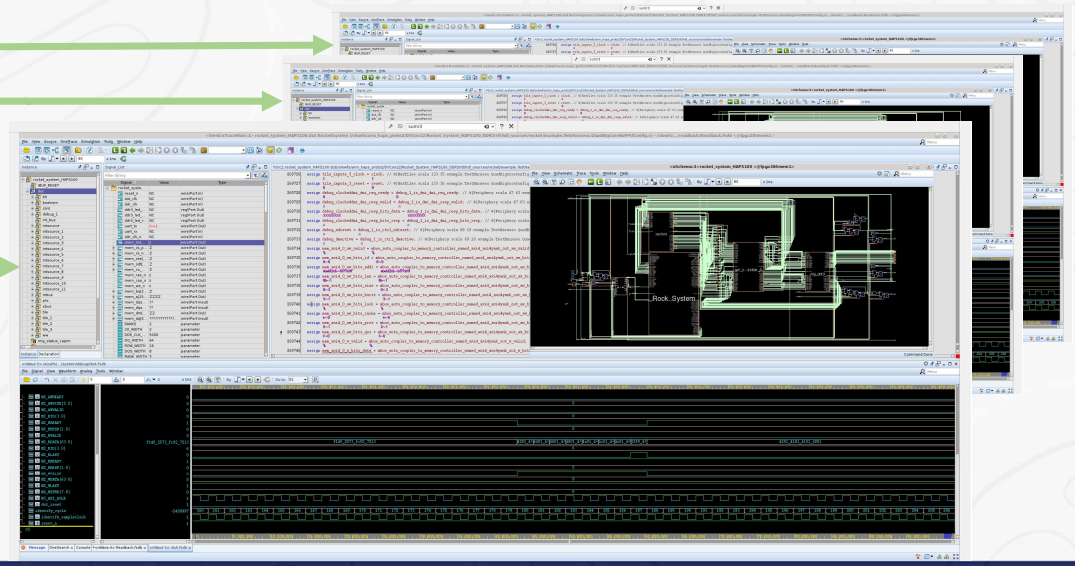
Complete Debug Environment for Multiple Offline Debug Session

- Verdi CRDB created using generated scripts

```
% rocket_system_HAPS100_run_verdi -crdbgen -rtldbdir $HAPS_PRJ_DIR/work/Rocket_System_HAPS100/pcs_uc/simv.daidir/ -dtop Rocket_System_HAPS100
```

- Complete debug environment exported as fully contained package
  - Can be run on any remote network with tool access
  - Allows multiple users in all world-wide locations to debug at RTL level with waveform expansion

FSDB +
Verdi Debug CRDB



Case Study : RISC-V Based Processor Chip

Debugging the Rocket Chip Implementation

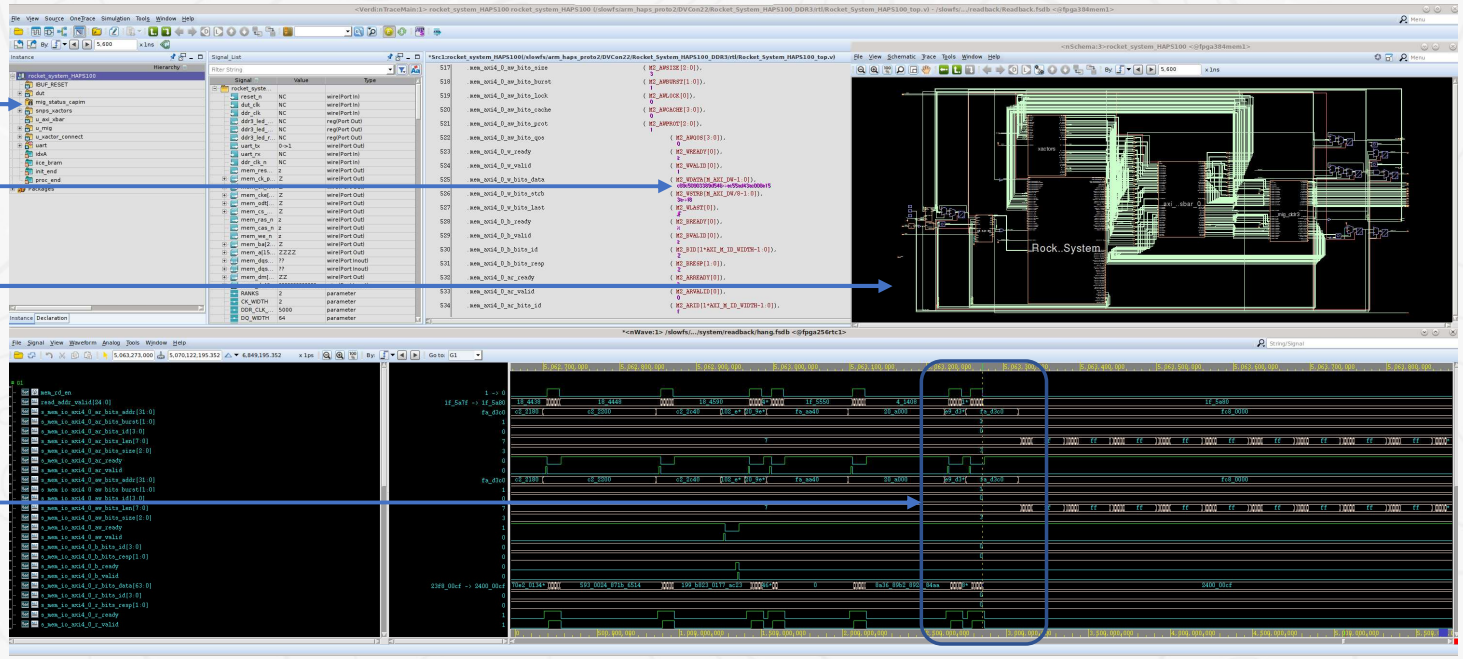
- GSV capture provides full visibility around the error point i.e. Cycle 2,485,479
  - Design knowledge is generally required to locate and track the source of the bug
  - Deep Cycle Visibility : Real world waveforms with expansion and correlation to RTL throughout DUT hierarchy over 100,000 cycles

DUT / Prototyping Infrastructure hierarchy

Waveform / RTL correlation

Schematic Browser

Lock Point Capture



Summary

Finding Hidden Bugs In Deep Cycles

ZeBu Server 4

- Large designs
- DPI/Monitors
- Stimuli Replay
- Fast Readback

HAPS-100

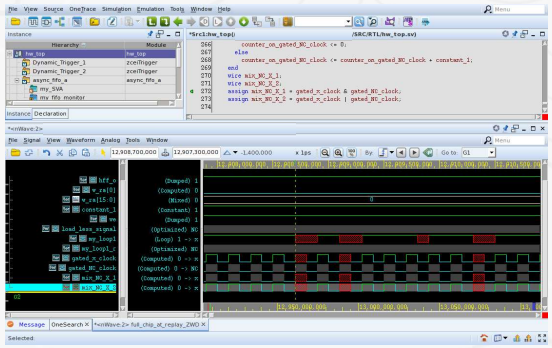
- Highest Performance
- Deep Tracing
- Global State Visibility

Verdi

- Industry standard debug
- ZeBu/HAPS outputs natively integrated

ZeBu Server 4

HAPS-100



Verdi