Finding Hidden Bugs in Deep Cycles
Advanced Debug Methodologies for Software-first System Validation

Yousef Qassid & Andy Jolley
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
System Validation using ZeBu Server emulation system

Why Software-First
Software-First

Evolution: HW testing to HW+SW testing

More SW

More Gate

More Activity
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

Monitors/Checkers

Stimuli Recording

Root Cause

Failure Observed

Stimuli Replay

Debug Data Dump

#0  ⋮  ⋮  ⋮  #N  ⋮  ⋮  #M  ⋮  Time

DUT State Save

DUT State Save

DUT State Save

Time 0

Time N

Time M

#N

Time N


Non-determinism

Low Level (Firmware – Drivers)

OS

Application
Faster Waveform Expansion and Debug with ZeBu and Verdi

Next generation tools for waveform-level debug

ZeBu Server 4 → Raw Debug Signals → Verdi

High Performance Parallel Expansion

High Performance Interactive Expansion

2.5B Gates, 500k cycles
Each signal drop <1sec

Expanded Debug Signals

Verdi

Native ZeBu Format

Scalable solution for complex billion gate SoC waveform-level debug with Verdi
Emulation Runtime & DUT Debug with Verdi

- Emulation Runtime
- Sniffer
- Triggers
- SVA
- Console
- RTL Source Code
- Watch Primary Clocks
- Waveform Capture
- Watch Signals Values
ZeBu Case Study
Multi RISC-V Full SOC

Emulation run setup

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

**NOC**

Per. 1 \(\rightarrow\) X-or 1

Per. 2 \(\rightarrow\) X-or 2

Application

SW
Multi Risc-V Full SOC

- Full Visibility

- Fast Readback

Application

FRB  Sniffer  zDPI

Per. 1  X-or 1

Per. 2  X-or 2

NOC

SW

SW
System-level Debug Using Monitors

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

Phase 1 (Billions of Cycles)
- Stimuli
- HW State
- Frame 0
- Frame 1

Phase 2 (Millions of Cycles)
- Stimuli
- HW State
- Frame N-1
- Frame N

Precise window based on system logs

Verdi Debug

Offline or Interactive Expansion

Full visibility waveform capture

Replay & Waveform Capture

Debug Window

Test Failure

Billion Cycle Run

zDPI (Monitors) streaming

Billions cycles

Stimuli

Frame 0

Frame 1

Frame N-1

Frame N
Identifying Failure Window

Checking Monitors Logs

ZeBu debug enables failure window detection in single pass during regression.
Waveform Capture & RTL Debug

- Monitors/Checkers
  - Monitors/Checkers
    - Time #N-1
    - DUT State Save
    - Time N
    - DUT State Save
    - Time #N+1
    - DUT State Save

- Stimuli Recording
  - Failure Observed
  - Debug Window
  - Time

- Full Visibility Capture
  - Time N
  - DUT State Save

- Stimuli Replay
  - ZTDB

- High Performance Parallel Expansion
  - Expanded Debug Signals

- High Performance Interactive Expansion
  - Application

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 -st 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
    zdpimReport -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
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

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

DUT Architecture
• RISC-V Architecture with 4-Core Processor

Software content
• Linux Boot ~ 1 Billion 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
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
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

Define BRAM and DTD configuration

Check Resource Usage and Sample Rates from Implementation Logs
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

Run from Reset State at full speed

Stop, Capture, Step, Capture over desired range

Resume for Next Event Detection

Event Detected

Design/RTL

Verdi

Waveform Reconstruction Engine
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

• Release DUT Reset and check for output from the UART
  – See error state occur after cycle 2,485,479 from reset
  – 25mS to reach error condition @ 100MHz
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

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

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

Capture 100,000 Cycles upon Trigger detect

- Run from Reset State to trigger condition
- Stop, Capture, Step, Capture over 100,000 cycles

Reset to Cycle Count = 32'h249F00

Read State Clock Step Read State Clock Step Read State Clock Step Read State

FSDB + Verdi Debug CRDB
Off-line Verdi Debug Environment

Complete Debug Environment for Multiple Offline Debug Session

- Verdi CRDB created using generated scripts
  
  ```bash
  % 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
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