Improving Verification Productivity of Embedded C Tests Using VCS Save/Restore

Neel Sonara — Broadcom
Amir Nilipour, Ajay Thiriveedhi — Synopsys

March 23, 2015
Silicon Valley

Agenda

Motivation and Challenges

Basics of Save Restore

Implementation on an Embedded C Environment

Applications of Save Restore

Summary

Embedded C for SoC Verification

- SoC architectures increasingly use more embedded processors
- C based processor driven tests drive SoC verification
- Embedded C code validates full chip scenarios
  - Test Interaction with different subsystems
  - Interrupt handling
  - Power sequencing
  - Gate Level Simulations (GLS)
  - Power Aware (PA) simulation with UPF's power intent

Embedded C testing Characteristics

- Most Embedded C tests share a common sequence
  - Common power up sequence
  - Common reset sequence
  - Initialization of stack pointer, cache, MMU, clock gating, DDR, etc.

Challenges with Embedded C testing

- Wasted time due to common initialization sequence
  - Milliseconds of simulation time (20-60 mins of actual wall clock time)
  - Longer runtime for Gate Level & Power Aware simulations
- Increased test development turn around time
  - Iterative test modification to
    - Add more checks for robustness
    - Add programming for missed features i.e. clock gating
    - Debug failures and unexpected behaviors
- Higher regression turn around time
  - All tests execute a common flavor of an initialization sequence
  - Degrades turn around time without testing new functionality

Agenda

Motivation and Challenges

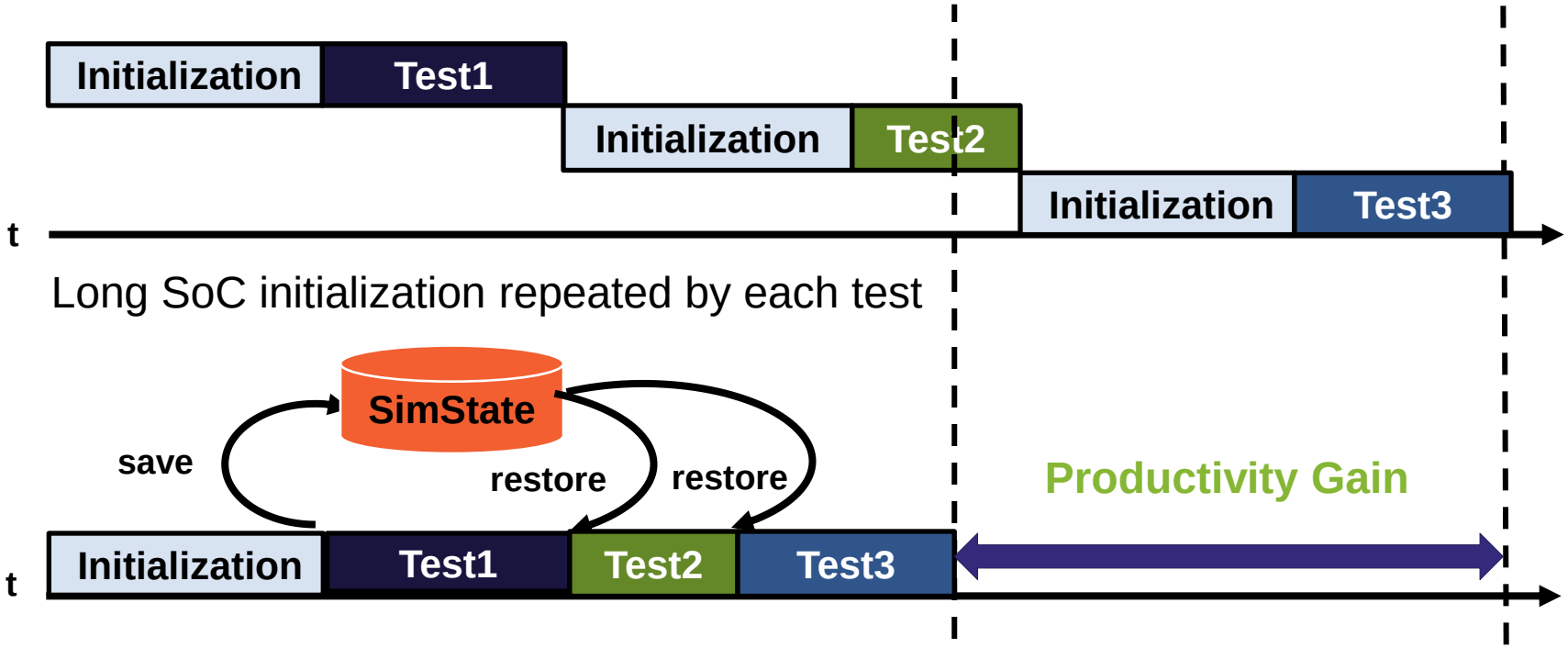
Basics of Save Restore

Implementation on an Embedded C Environment

Applications of Save Restore

Summary

Eliminate Redundant Cycles



- 'save' command captures simulation state at the end of initialization
- Subsequent tests begin with 'restore' of previously saved state
- Productivity gain depends on the amount of time spent in initialization relative to the average test time

20-60% runtime improvement observed

Enable at Run Time or in Testbench

- To save snapshot for test development or regression runs
  - Wait for end of SoC initialization on one test
  - Complete simulator's pending events
  - Create and save simulation snapshot in the disk
  - Use Tcl command for run time control: save snap
  - Or Use Verilog system task for testbench control: $save(snap)
- To restore to saved snapshot
  - Use Tcl command : restore snap
  - Use system task : $restore(snap)

Agenda

Motivation and Challenges

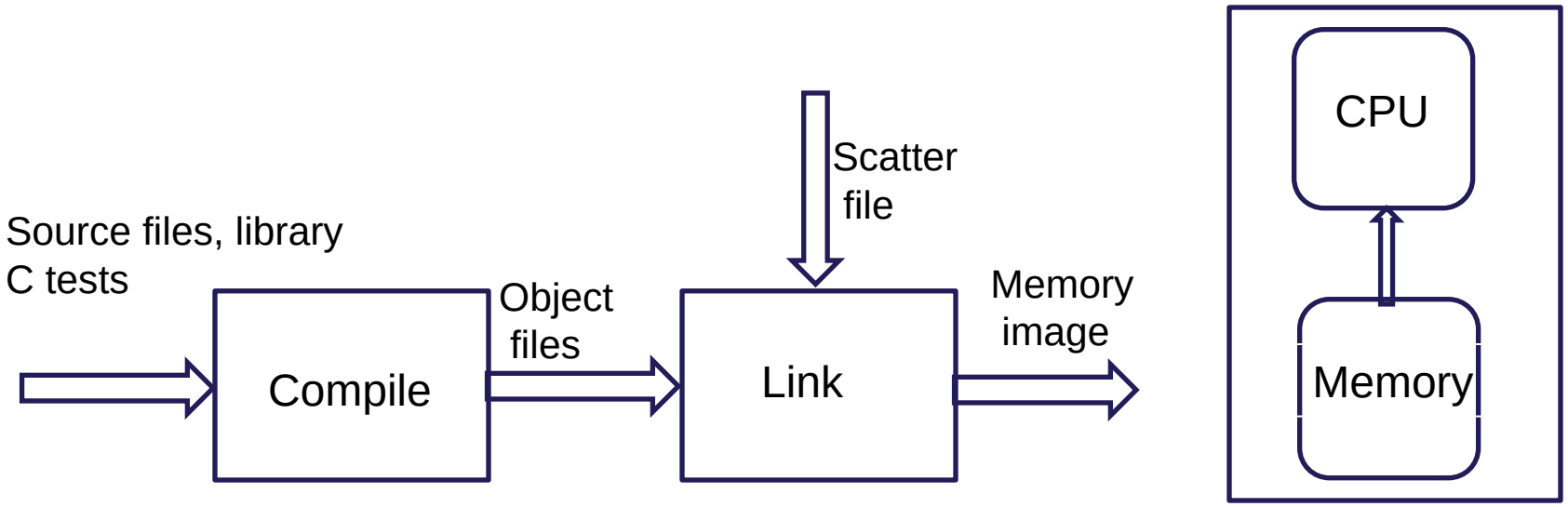
Basics of Save Restore

Implementation on an Embedded C Environment

Applications of Save Restore

Summary

Test Flow and Usage



- Compile all the C files t create object files
- Link object files using a scatter file to create memory images
- Memory images are pre-loaded into ROM and RAM using $readmemh
- Processor fetches and executes the code after reset is released.

Scatter File Details

```
SROM_LOAD 0x00000000 0x18000
{
   SROM 0x00000000
   {
      vectors_rom.o (ROMVect, +First)
      .ANY          (InRoot$$Sections)
      init.o        (+RO-CODE, +RO-DATA)
      *             (SEC_ROM_AREA)
   }
}

SCRATCH 0x10000000 0x8000
{
SCRATCH 0x10000000 ZEROPAD
UNINIT 0x5000
   {
      .ANY          (+RW, +ZI)
      *    (SCRATCH_AREA)
   }
}
```

```
SCRATCH_BASE 0x34044400 0x3C00
{
   ScratchVectors 0x34044400
ZEROPAD 0x40
   {
      vectors_ns.o  (NSVect, +First)
   }

   STACKS0 0x34045000
   {
      stack.o (+ZI)
   }
}
```

Flexibility to keep different C code in different memory sections

Implementation Considerations

- Typical flow without modifying embedded C is:
  - run the test until common initialization, create save point
  - modify embedded test or select another test
  - Re-compile, preload memories and restart it from saved point.
- main_test() can be located anywhere in memory during compilation.
- Can not preload complete memory image for new test
  - Variables can get corrupted
  - program flow can get corrupted
- Need a precise save point
  - After initialization is done and main_test() function is called
  - CPU puts address of main_test() on the fetch bus

Customization for Embedded C

- Dedicated region in scatter file for a main test code

```
SCRATCH_AREA 0x10000000
0x8000
{
  SCRATCH1 0x10000000 0x5000
   {
       .ANY          (+RW, +ZI)
       *    (SCRATCH_AREA)
   }

   MAIN_SECTION 0x10005000
0x3000 {

     *  (TEST_AREA)
   }
}
```

```
void __attribute__((section("TEST_AREA")))
main_test() {

      /*    main test code is here    */

}
```

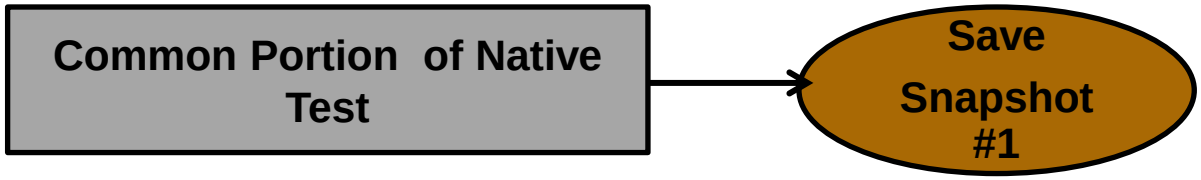
- Reload only memory image for TEST_AREA (new test) before Restore.

Implementing Save

- Need a precise save point
- After initialization is done and main_test() function is called
- Create Tcl breakpoint when processor fetch address is the start address of main_test() routine

```
stop -once
-condition {/top/…/ARADDR[39:0] == 40'h80000  } -continue
-command {run 0; save my_snap}
```

- Save simulation snap when breakpoint is hit
- main_test() can be changed as it is not fetched by processor yet

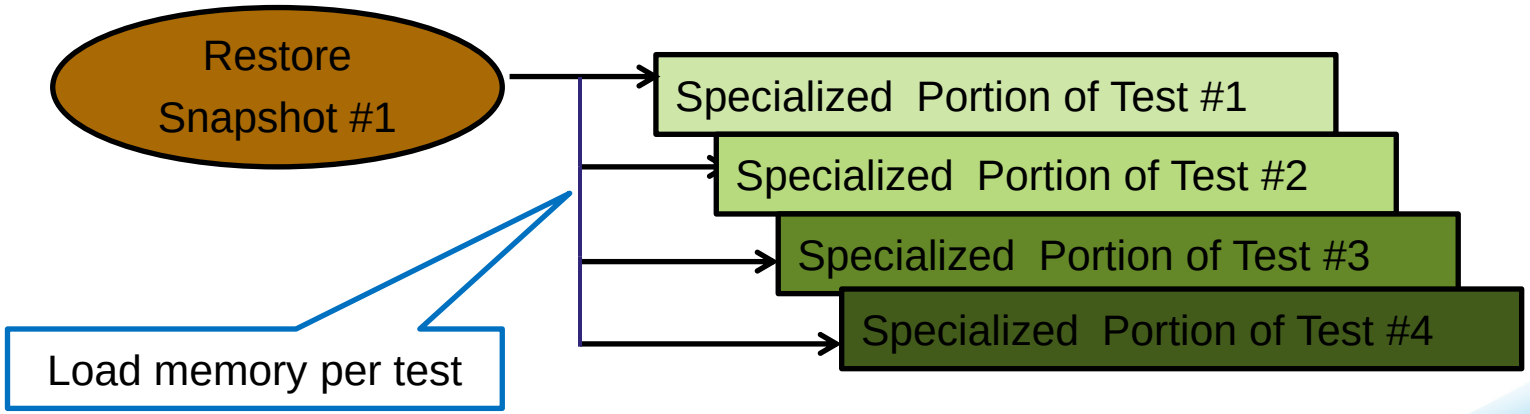


Implementing Restore

- Restore saved snap.
  - ***restore my_snap***

- Load memory region for TEST_AREA image for new test
  - ***memory -read "top/…/mem/array" -file "TEST_AREA" -radix hex -start 32'h0000***

- Run test



Agenda

Motivation and Challenges

Basics of Save Restore

Implementation on an Embedded C Environment

Applications of Save Restore

Summary

Flexibility in Waveform Dumping

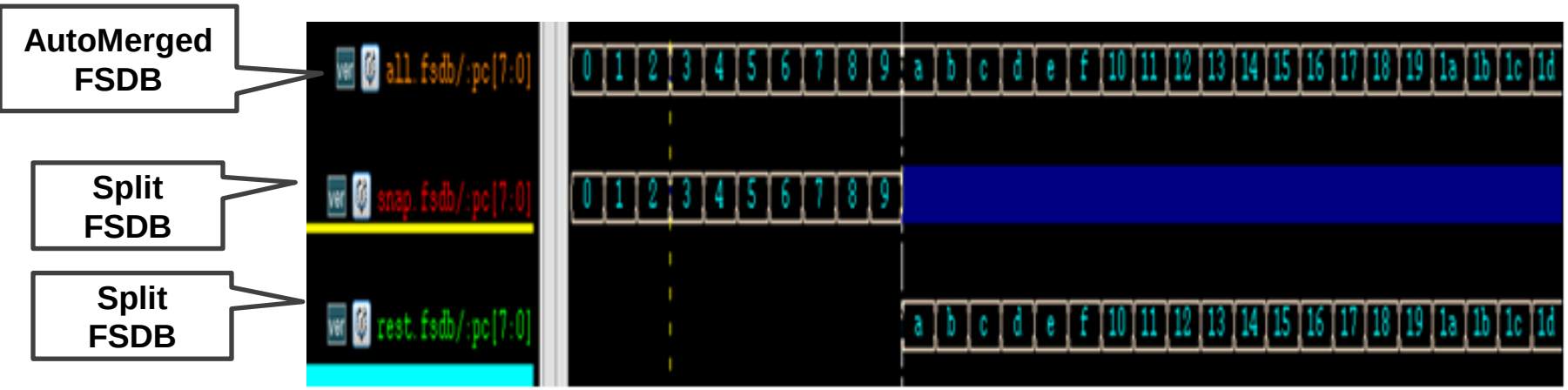
- User may need merged or split dump files for Save and Restore simulations

- Auto-Merge dumping for both simulations into one file
  - Turn FSDB on prior to Save point
    - ***fsdbDumpvars {"+fsdbfile+all.fsdb"}***

- Split dumping into two files, one for each simulation
  - Turn FSDB on prior to Save point
    - ***fsdbDumpvars {"+fsdbfile+snap.fsdb"}***
  - Redirect dump into a different file anytime after Restore
    - ***restore snap; fsdbDumpoff***
    - ***fsdbDumpvars {"+fsdbfile+rest.fsdb"}***

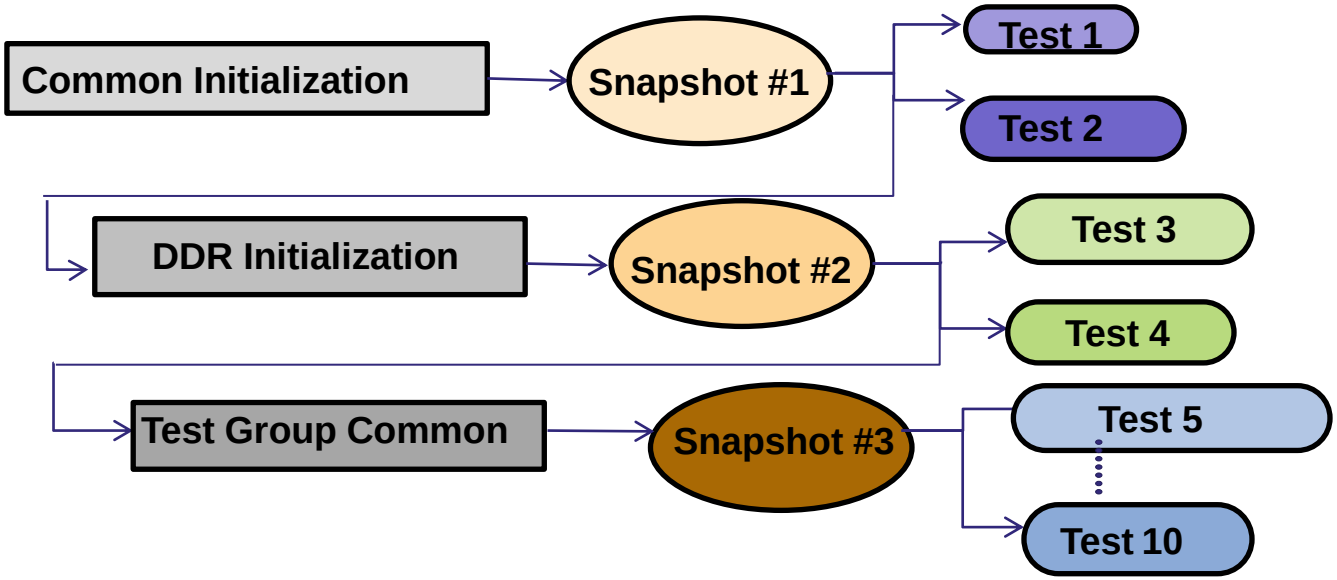
Flexibility in Waveform Dumping

- Enable dumping only during Restore Simulation
  - When no need to debug initialization sequence
    - ***restore snap***
    - ***fsdbDumpvars {"+fsdbfile+rest.fsdb"}***



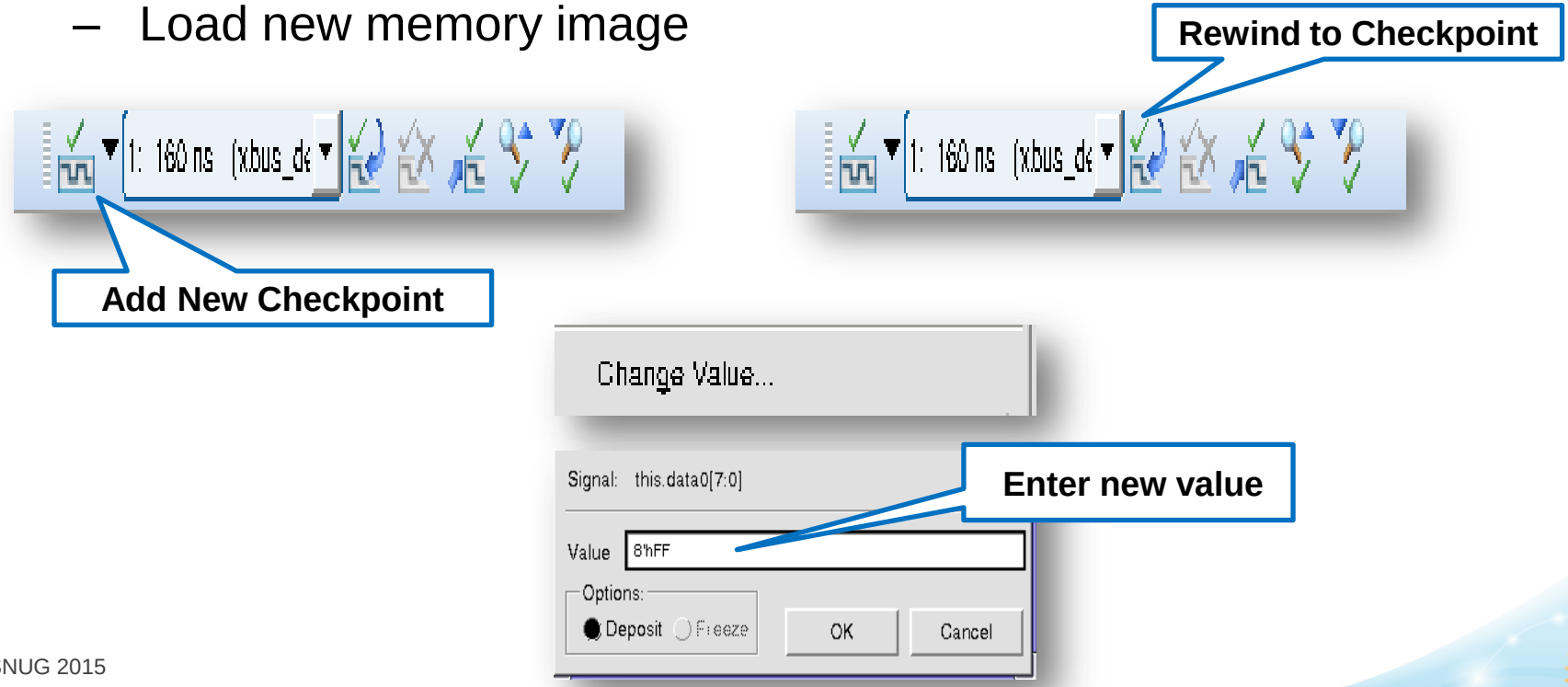
Multiple Save Points

- Different tests may start from various saved points
- Identify and create appropriate snapshots for reuse
  - End of common initialization
  - End of DDR initialization
  - Test Group Common tasks



Verdi Interactive Check/Rewind

- Creates dynamic simulation Checkpoints in memory
- Can Rewind to any Checkpoint during interactive simulation
- Allows What-If-Analysis without recompile
  - Force values on signals or variables
  - Load new memory image



Combine with Verdi Check/Rewind

- Combine with Static Save/Restore
  - Restore to a well tested point in simulation
    - ***restore snap***
  - Run to a suitable point to start interactive debug
  - Save dynamic checkpoint and continue debug
  - Rewind to any saved checkpoint and try another debug iteration

- If same dynamic checkpoint is needed across simulations, save as static snapshot

Agenda

Motivation and Challenges

Basics of Save Restore

Implementation on an Embedded C Environment

Applications of Save Restore

Summary

Results

- Improved test development with quicker iterations
  - Time saving of 30 to 90 minutes per debug iteration
- Achieved 30% time savings during regression testing
  - Runs completed more frequently and quickly
- Fast and flexible debug with FSDB
  - Auto-merge
  - Split-dump
  - Dump after or during restore
- Combined with Verdi Interactive Check/Rewind

Best Practices

- Embedded C test needs to be loaded in separate memory section with use of scatter file.

- The save point should be just before the test code is fetched by processor.

- Tcl Procedures are not saved automatically
  - Tcl procedures sourced before save will not be restored
  - Need to explicitly source all the Tcl procedures at restore

- The OS and patch versions of the **restore** host must match those of the **save** host.

BROADCOM.

snug 25
Synopsys Users Group

Thank You