# Reality and Challenges in Using Save/Restore in SoC Verification Environment

Satish Naidu
Freescale Semiconductor, Inc.

September 23, 2014
SNUG Austin

# Agenda

Introduction

Factors Influencing usage of Save/Restore

Building Verification Environment with Save/Restore

Best use cases with Save/Restore

Results

Challenges faced with Save/Restore

Conclusions and Recommendations

# Introduction

# Introduction

- SoC designs are getting bigger and more complex
- Need to put more efforts in verification
- Some of the implications on simulation productivity
  - Increasing regression completion time
  - Debug turn-around time
  - Resource utilization

# Introduction

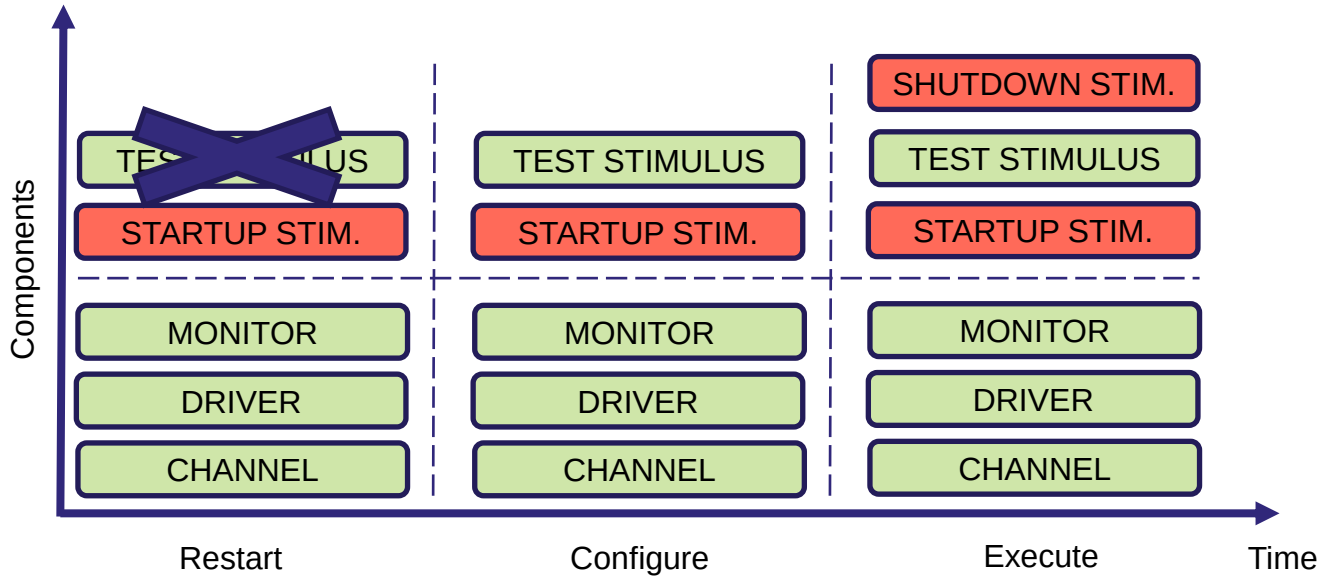
- One approach is to use Save/Restore
  - Save snapshots of common simulation cycles
  - Restore the simulation using saved snapshots

- How do we use this approach to address the complexities?
  - Divide and conquer approach
  - Identify common simulation patterns which can be shared across different simulations, like
    - Reset/initialization phase
    - Common design configuration phase
  - Architect the testbench to fully support Save/Restore

# Factors Influencing usage of Save/Restore

# Factors influencing Save/Restore

- Testbench Methodology
  - Verification methodology plays a major role
    - Phasing and sequencing shouldn't be coupled together
    - Stimuli/Sequences should be independent of phasing




An example of legacy methodology

# Factors influencing Save/Restore

- Testbench Methodology
  - Testbench infrastructure
    - Single test flow with multiple sequences
    - Use plusargs to run tests with different stimulus

      **Example:** `+UVM_TESTNAME or +TESTNAME`

    - Guard the plusargs based on an event trigger

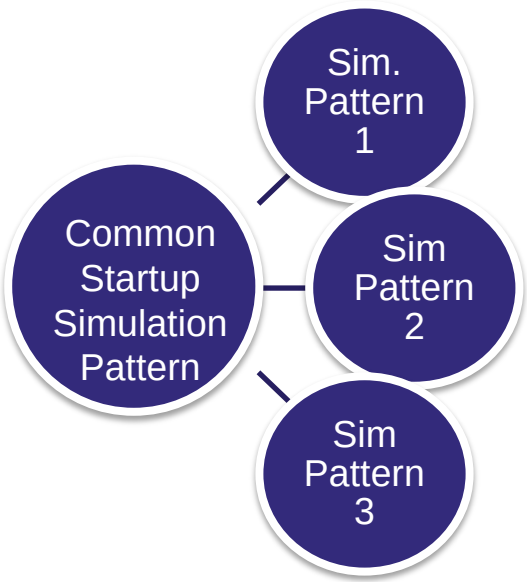
      **Example:**
      ```
      always@(restart)
      begin
        $value$plusargs("parm=%d",parm_val);
      end
      ```

    - Use always block for preloading instructions

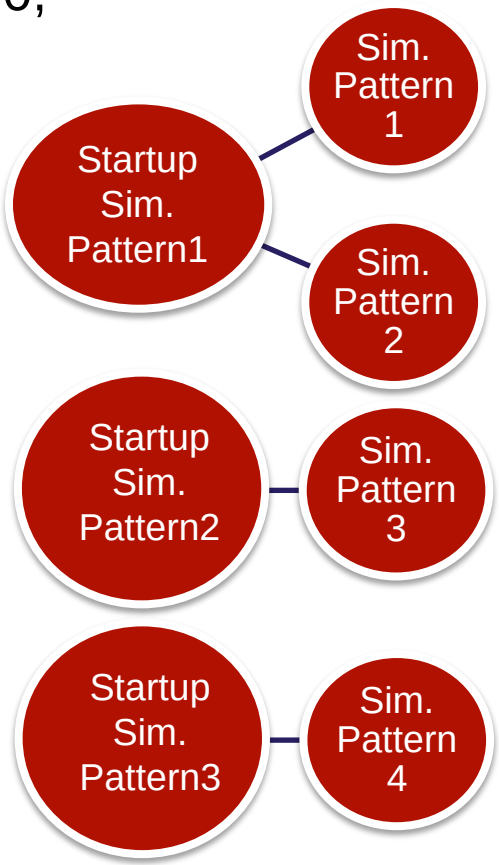
      **Example:**
      ```
      always@(restart)
      begin
        readmemh(instr_file, mem);
      end
      ```

# Factors influencing Save/Restore

- Design Configuration Requirements
  - More configuration of pin-muxing at time 0, implies more simulation patterns to save



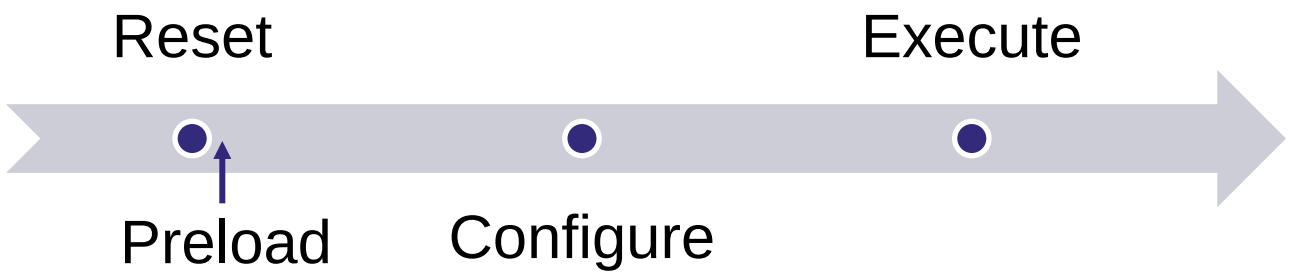

Less pin-muxing




More pin-muxing

# Factors influencing Save/Restore

- Design Configuration Requirements
  - Should we implement it at IP level or at SoC level?
    - IP level designs need to exercise more functional paths
    - SoC verification targets more application specific scenarios
    - More reset/configuration patterns limits utilization of Save/Restore at the SoC level
  - Loading instructions/data early during the simulation limits utilization of Save/Restore



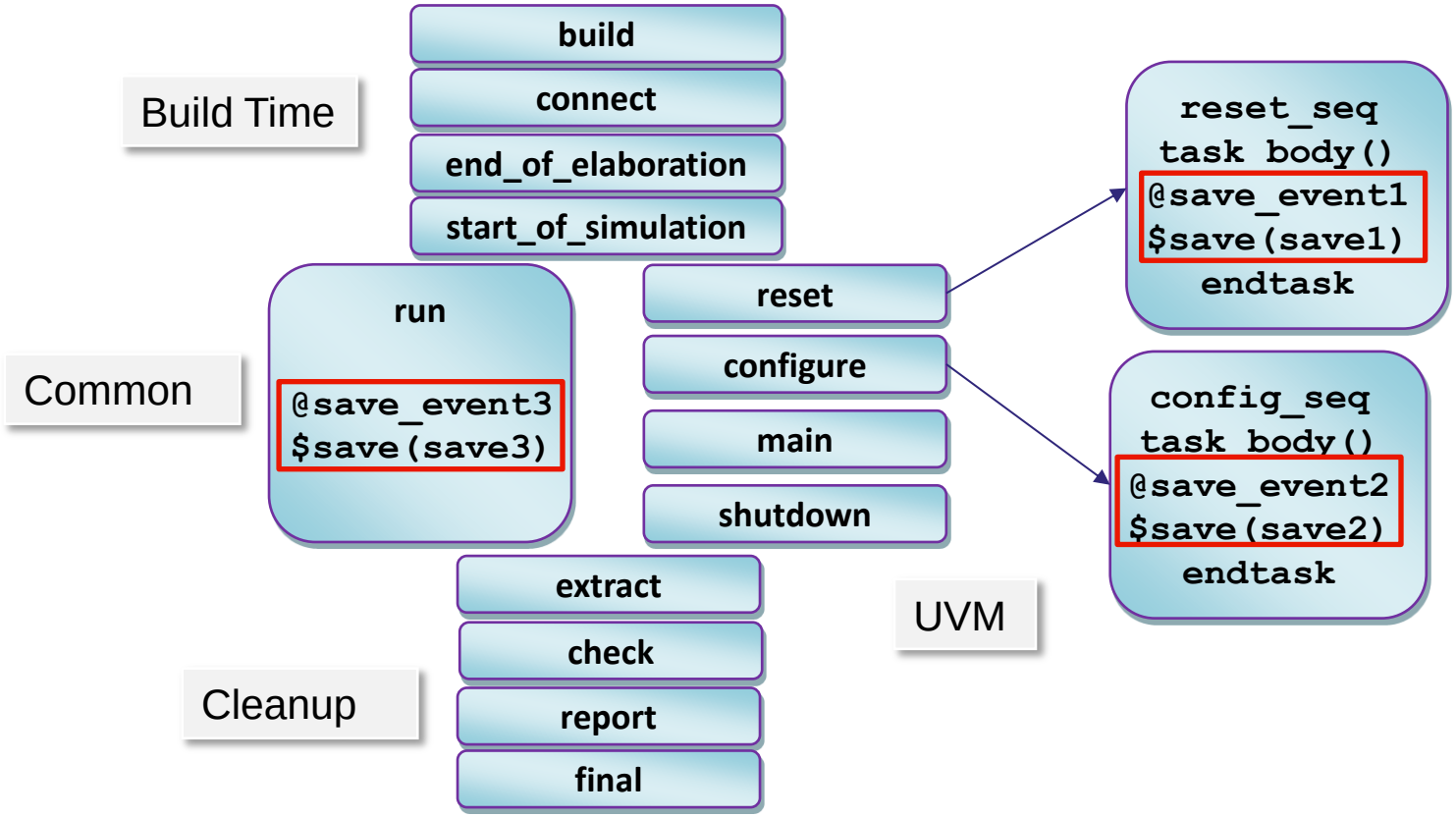
# Factors influencing Save/Restore

- Stimuli
  - Random stimuli/sequences
    - Fully random reset stimuli can't utilize this feature!
    - Standalone reset-independent sequences are the best candidates
  - C based tests/stimuli
    - Compile independent
    - Pre-loading done only when needed!
    - Faster turnaround time for debugging
    - One of the best ones to fully utilize this feature
  - TCL based tests/stimuli
    - Compile independent
    - Faster turnaround time for debugging

# Building Verification Environment with Save/Restore

# Building Env with Save/Restore

- Architecting the Testbench
  - Verification methodology plays a major role
    - UVM Phasing has a better solution for Save/Restore approach



# Building Env with Save/Restore

- Architecting the Testbench
  - Single test and multiple sequences
    - Use single `uvm_test` object
    - Use command line arguments to capture sequences which can be run in different phases
  - Where to implement the `$save` command
    - Call `$save` at the beginning of each phase under the `uvm_test` component
    - Can also be embedded inside sequence's body task

# Building Env with Save/Restore

- Architecting the Testbench
  - Avoid “shortcuts” if you can
    - If needed make sure it is triggered at the right point in simulation
    - Avoid initial blocks for shortcuts

      **Example:** `forces to shorten counters`
    - Shortcuts can hide bugs

      **Example:** `fuse read bypassing in a POR sequence`
  - Memory initialization and preloading
    - Initialize the memories only when needed
    - Pre-load embedded C code just before the CPU starts fetching
    - Can be part of standalone sequences

# Building Env with Save/Restore

- Compilation and simulation flow
  - Synopsys® VCS supports two flows
    - UCLI based save and restore command
    - IEEE standard system tasks flow using `$save` and `$restart`
  - We chose to use `$save` system task to create snapshots

```
simv +save -l sim_save_sim.log
```

  - For restoring from a snapshot we used `-r <snapshot_name>`

```
simv +soc_main_vseq=my_vseq -l my_vseq.log -r soc_config
```

# Best Use Cases of Save/Restore

# Best Use cases of Save/Restore

- Best simulation turn-around
  - C based tests for embedded processors
    - Create simulation snapshot just when the CPU is ready to execute its first instruction
    - Use this snapshot to run multiple C based tests avoiding redundant CPU reset/configure sequences
    - Updating the C code doesn’t need model recompilation!

# Best Use cases of Save/Restore

- Faster debugging approach
  - Save/Restore + VCS UCLI feature improves debugging time by a huge factor
    - Long running tests are very hard to debug/reproduce
    - Create multiple snapshots during simulation
    - Proceed further on subsequent debugging cycles using the snapshot
    - Use UCLI forces wherever necessary to make quick forward progress to overcome minor testbench/logic issues

# Results

# Results - RTL

- Improved simulation turnaround times
  - 30% savings in simulation time for regressions!

| RTL | Elapsed Time | Vir. Memory | % |
|---|---|---|---|
| Normal simulation | 79 min. | 12 GB | |
| Sim. with restore | 62 min. | 12 GB | 21.5% |

Time Saved

| RTL Regression | Total Time | % |
|---|---|---|
| Normal Regression | 30 hr. | |
| Regression with restore | 21 hr. | 30% |

Time Saved

  - Initial overhead in generating the default snapshots!

| RTL | Elapsed Time | Vir. Memory | % |
|---|---|---|---|
| Sim. with +save | 84 min. | 12 GB | - 6% |

One time performance overhead
(84-79) / 79 = 6%

# Results - GLS

- Huge improvements in Gate Level Simulation!
  - 86% savings in simulation time!

| GLS | Elapsed Time | Vir. Memory | % |
|---|---|---|---|
| Normal simulation | 76 min. | 104 GB | |
| Sim. with restore | 10 min. | 104 GB | 86% |

Time Saved

  - Initial overhead in generating the default snapshots!

| GLS | Elapsed Time | Vir. Memory | % |
|---|---|---|---|
| Sim. with +save | 155 min. | 104 GB | 100 |

One time performance overhead
(155 -76) / 76 = ~100%

# Challenges faced with Save/Restore

# Challenges with Save/Restore

- Challenges faced during debugging
  - Lacks mixed UCLI flow simulation support
    - A save simulation in a non-UCLI mode, wouldn't support restore in UCLI mode and vice-versa
  - Dumping waveforms with Save/Restore enabled
    - Use of UCLI to dump waves enforces to use UCLI for all simulations
    - Dumping waveform for restore simulation only is tricky!
    - Testbench need to be architected carefully to support this feature
  - Reliability and stability is a concern with the VCS Tool!
    - Simulation failed on newer design model which passed on earlier
  - Snapshot sizes are a little big for Gate Level Simulations!
    - ~20GB per snapshot
    - Depends on the design!

# Conclusions and Recommendations

# Conclusions and Recommendations

- Significant improvement in simulation turnaround times and debugging efforts
  - ~30% saving in RTL and ~86% saving in GLS
- Proved to be a good investment for returns in gate level environment
- Recommendations for implementing Save/Restore
  - Understand the design prior to implementing any testbench support
  - Look for common simulation patterns in your design
- Recommendations for VCS tool support
  - Provide option to use an existing snapshot on a recompiled model!
  - More tool debugging support when it crashes!
  - VCS Save/Restore compliance switch for VIPs
  - Systematic, conducive ecosystem to help productize this flow in reasonable time frame by cutting down on iterations

# Q & A


Thank You