2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Test driving Portable Stimulus at AMD

Prabhat Gupta, AMD
Matan Vax, Cadence

AMD cadence®

accellera SYSTEMS INITIATIVE

Agenda

Why Portable Test and Stimulus Standard (PSS)

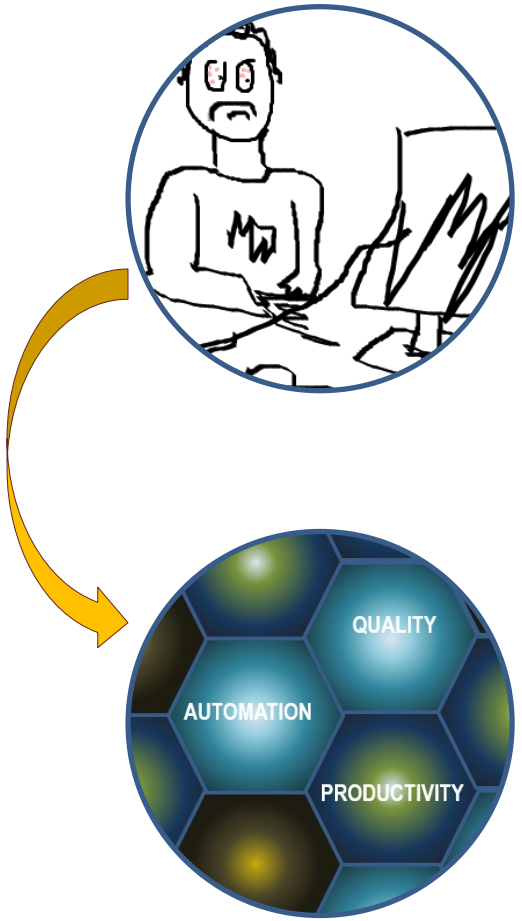
Walk through an example scenario

PSS guideline

Observations and conclusion

Why Portable Test and Stimulus Standard (PSS)

- Ease of test creation
  - New test scenarios find bugs
  - Declarative syntax with procedural support, C++ description
- Formal system description
  - Constraints, input/outputs, resource and randomization
    - UVM but for system-level and better
  - Automation for test generation
- Close approximation of real world scenarios
  - Deterministic runs
  - Ease of issue reproduction on emulator and simulation
- We used PSS tool Perspec™ from Cadence Design Systems for this exercise



Existing stimulus

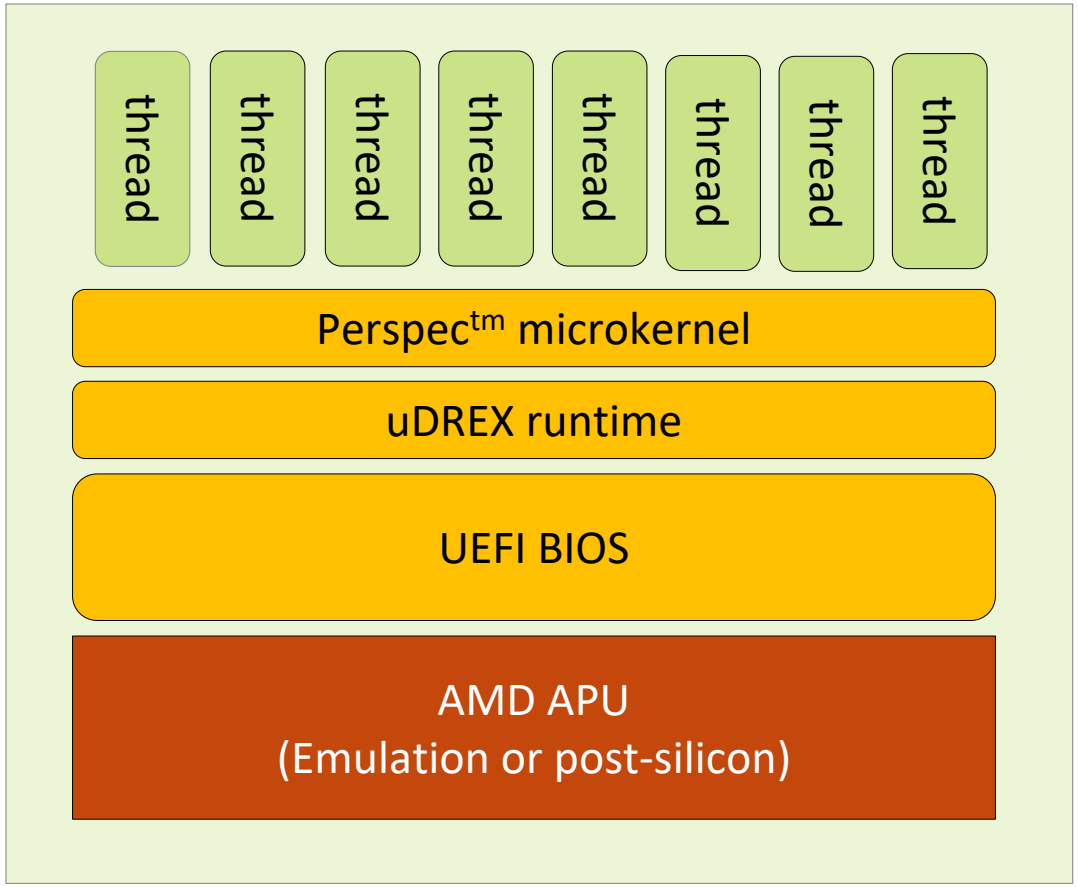
Post-silicon	• Generally OS-based tests • Longer debug time • Failures are difficult to bring to emulation or simulation for debug
Pre-silicon SoC	• Simple directed feature tests • Difficult to manually create complex scenarios • Long run time for complex scenarios
Pre-silicon IP	• Excellent UVM-based constrained random testbench • IP initialization sequences not easily portable to FW or post-silicon tests • IP level tests lack system context

Stimulus with PSS

Post-silicon	• Smaller deterministic bare-metal tests • Compose complex scenarios • Easily bring debug to Emulation • Generate large set of tests for regression
Pre-silicon SoC	• Describe test intent with PSS • Automation helps with complex scenario composition • Reuse tests post-silicon
Pre-silicon IP	• Reuse SoC scenarios • Export initialization sequences to firmware and post-silicon • Export IP specific scenarios to SoC

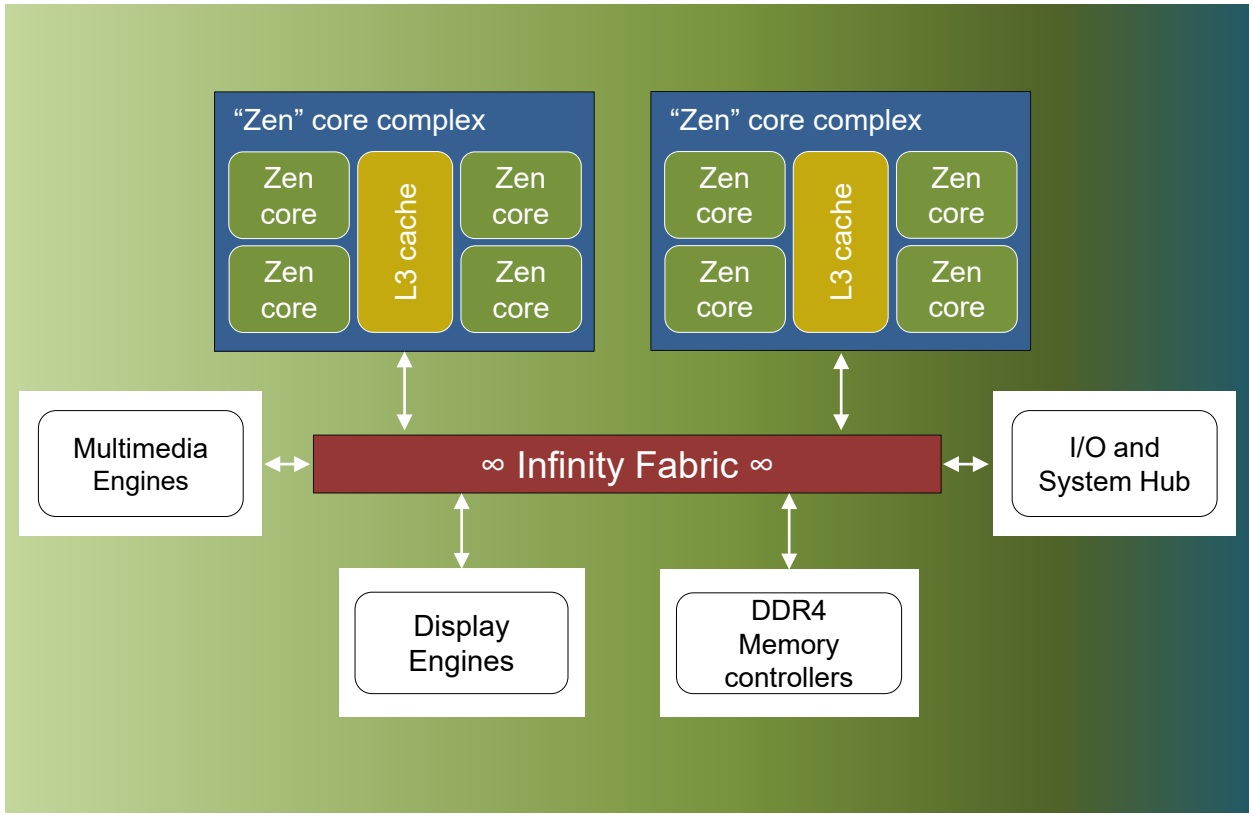
Test environment

- PSS generated tests run on UEFI BIOS
- Test are loaded and started by uDREX
  - uDREX is a thin layer to abstract BIOS and provide services to PSS tests
- Emulator
  - Test and uDREX are backdoor loaded into DRAM
- Post silicon
  - Test are copied in a known location on disk
  - uDREX finds and starts test on all enabled threads

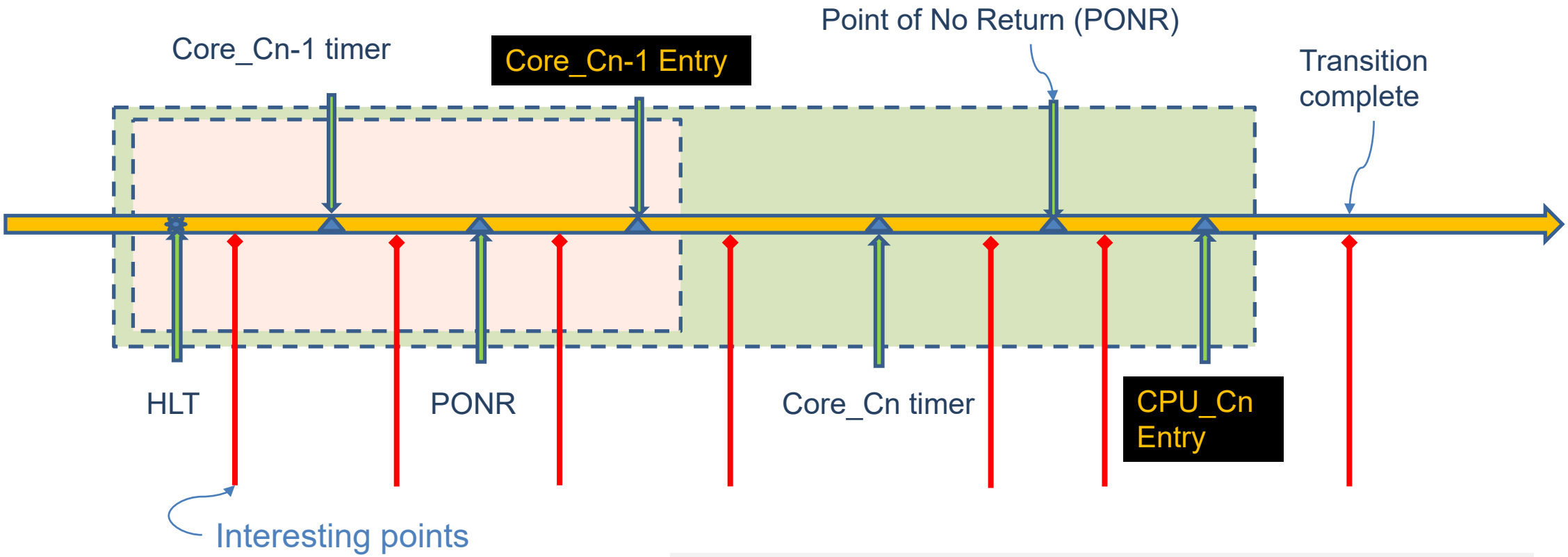


Example test scenarios

- Processor C-states scenarios
  - Transition all cores in all complexes to a C-state at same time
  - Transition one specified core to a C-state
  - Sequentially transition cores to a C-state on all complexes
  - Transition cores in and out of a C-state at specified duty cycle



Core C-State transition



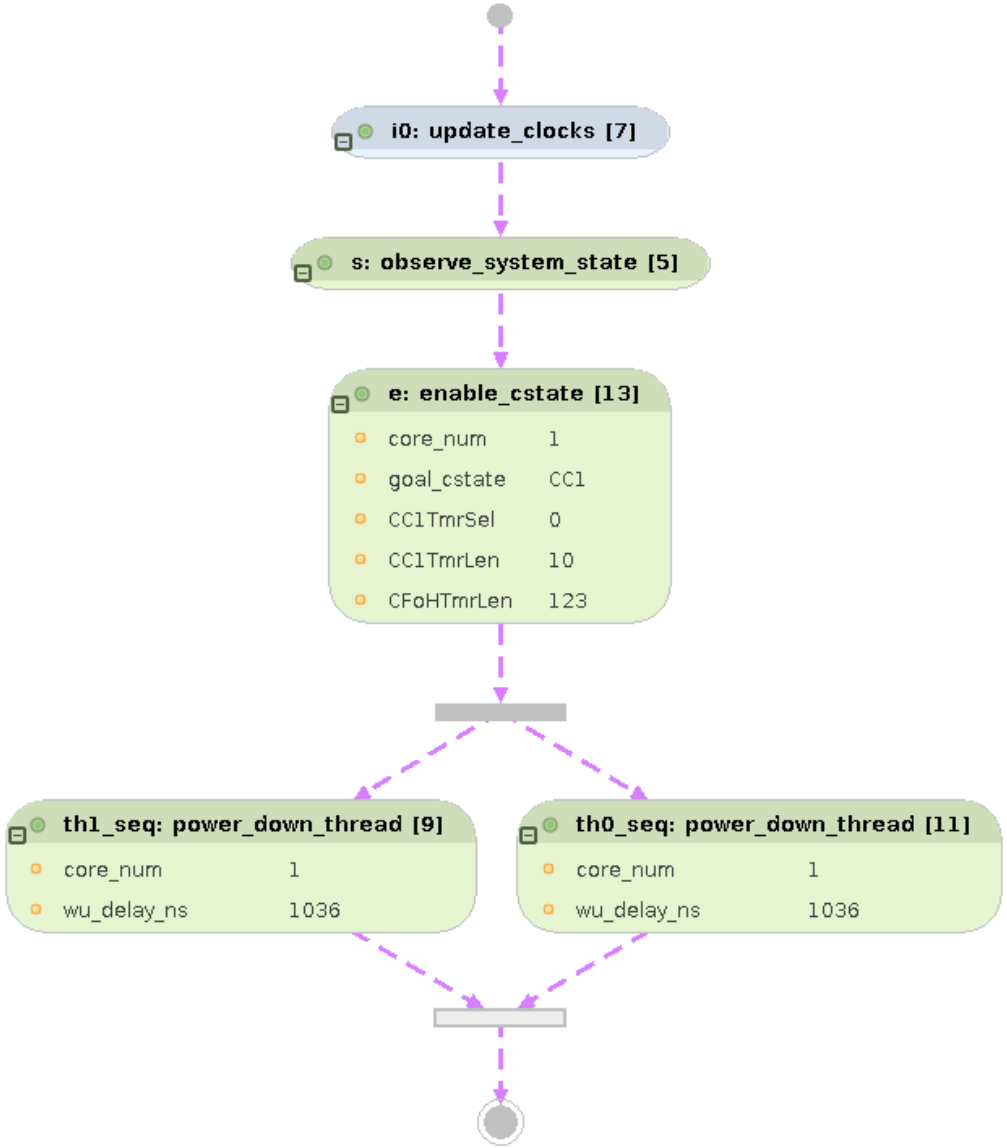
This timeline shows all interesting points an interrupt can arrive during C-state transitions. All these points matter for system-level power scenarios

Generic power down core activity

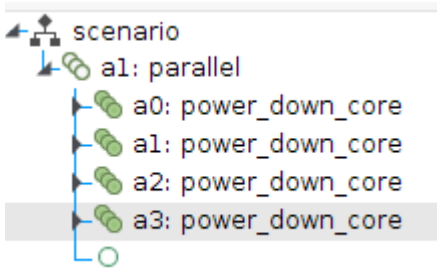
```
action power_down_core {

  rand int in [0..NUM_CORES-1] core_num;
  rand bool wakeup_core;
  rand bit[32] usr_delay_ms;
  rand cstate_e in [CoreC1, Core_C2] goal_cstate;

  activity {
    sequence {
      do observe_system_state;
      do enable_cstate with {
            core_num    == this.core_num;
            goal_cstate == this.goal_cstate; };
      parallel {
        do power_down_thread with {
          core_num   == this.core_num;
          thread_num == 0; };
        do power_down_thread with {
          core_num   == this.core_num;
          thread_num == 1; };
  }}};
```

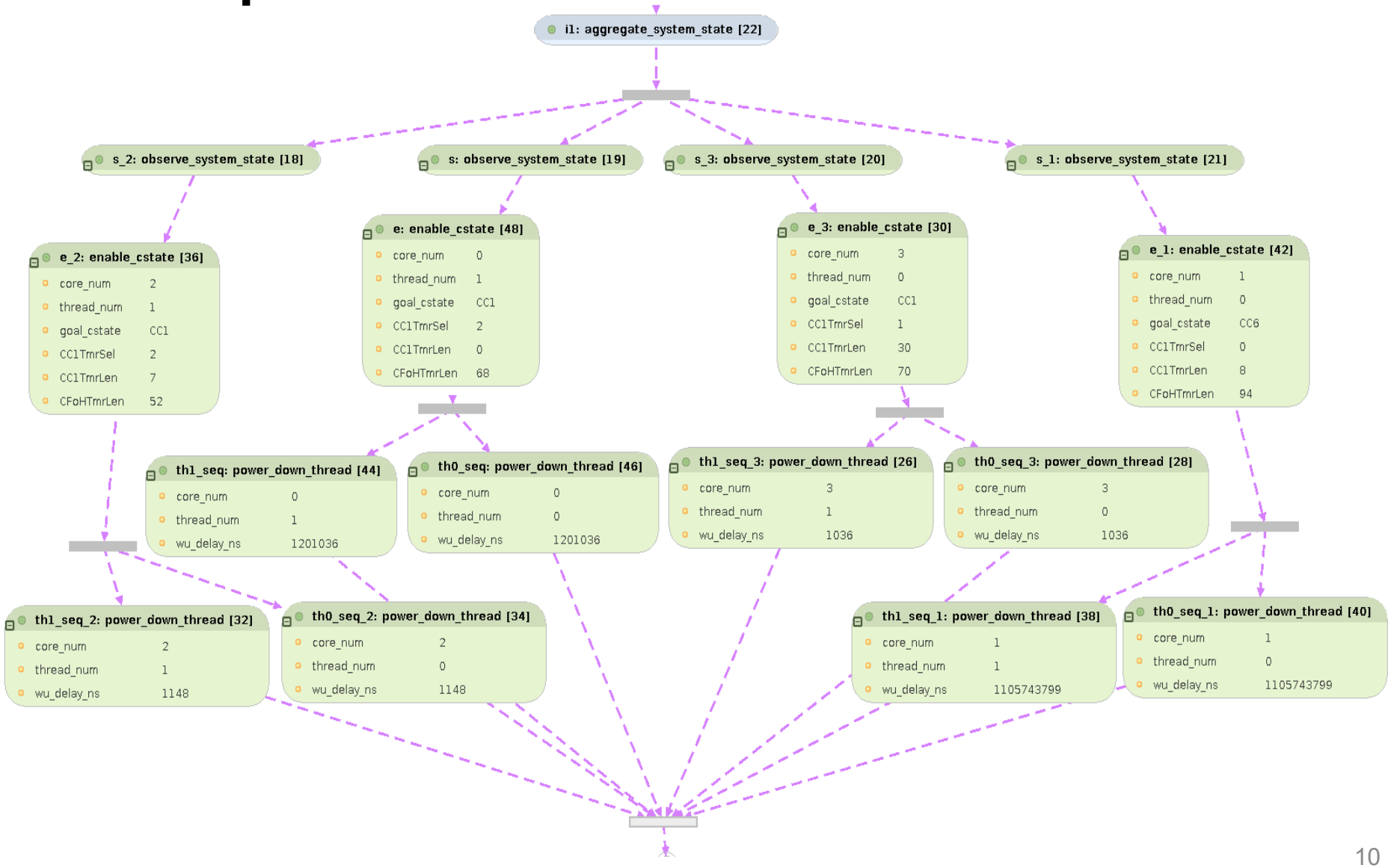


Compose a scenario

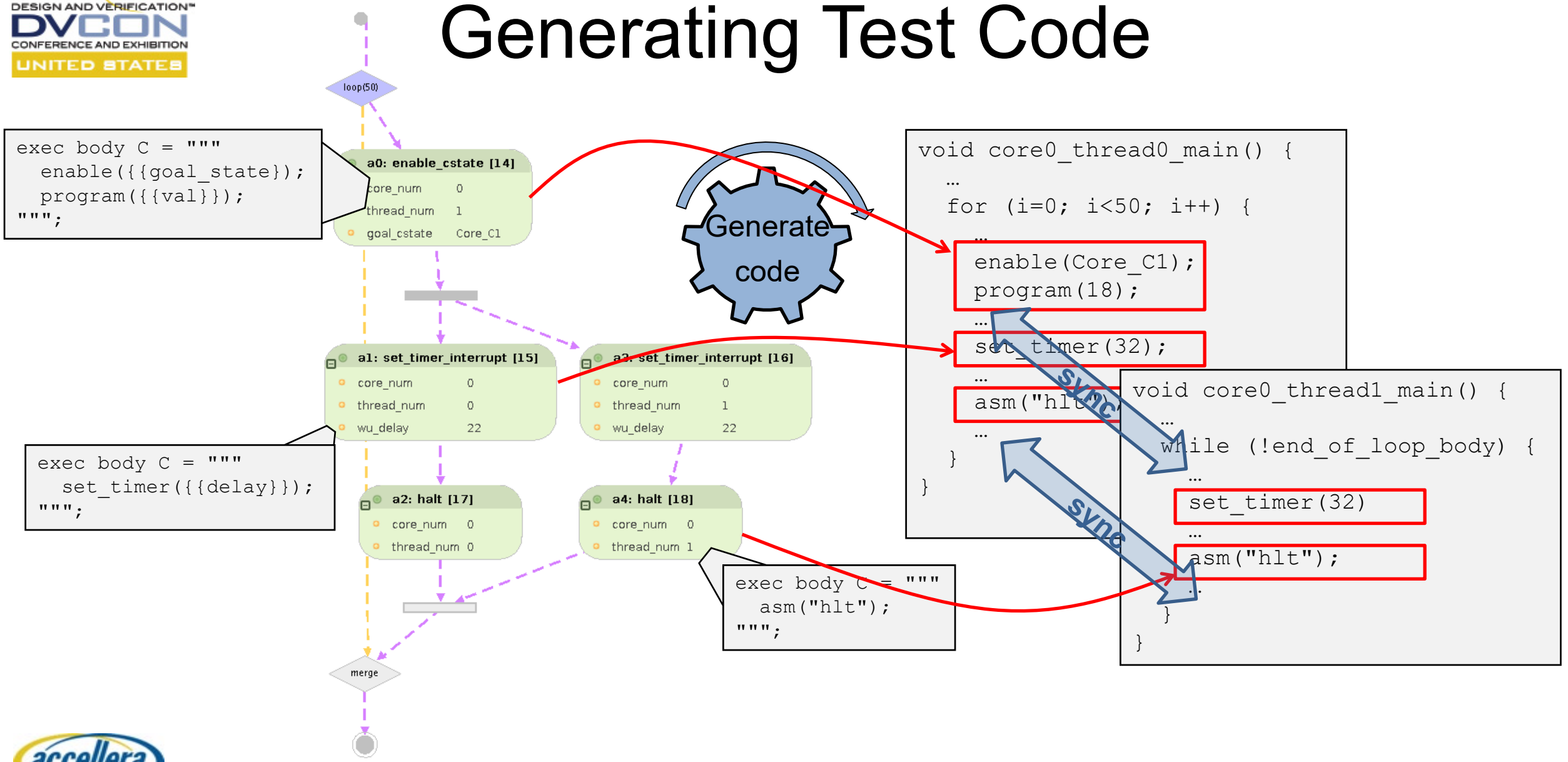


Four cores transition to a C-state then wake-up at same time.

Timer count was constrained by PSS tool based on current clocks

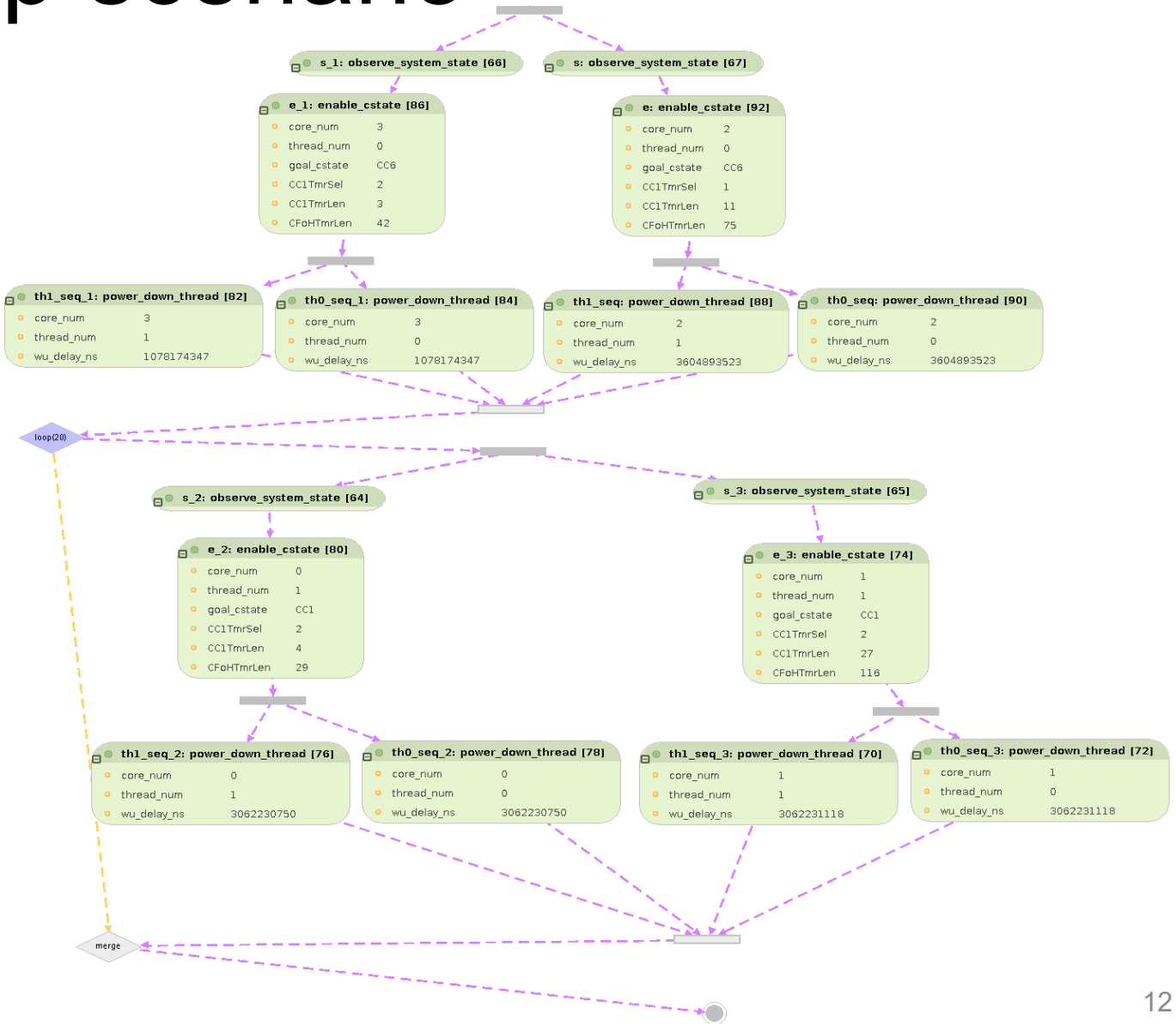


Generating Test Code

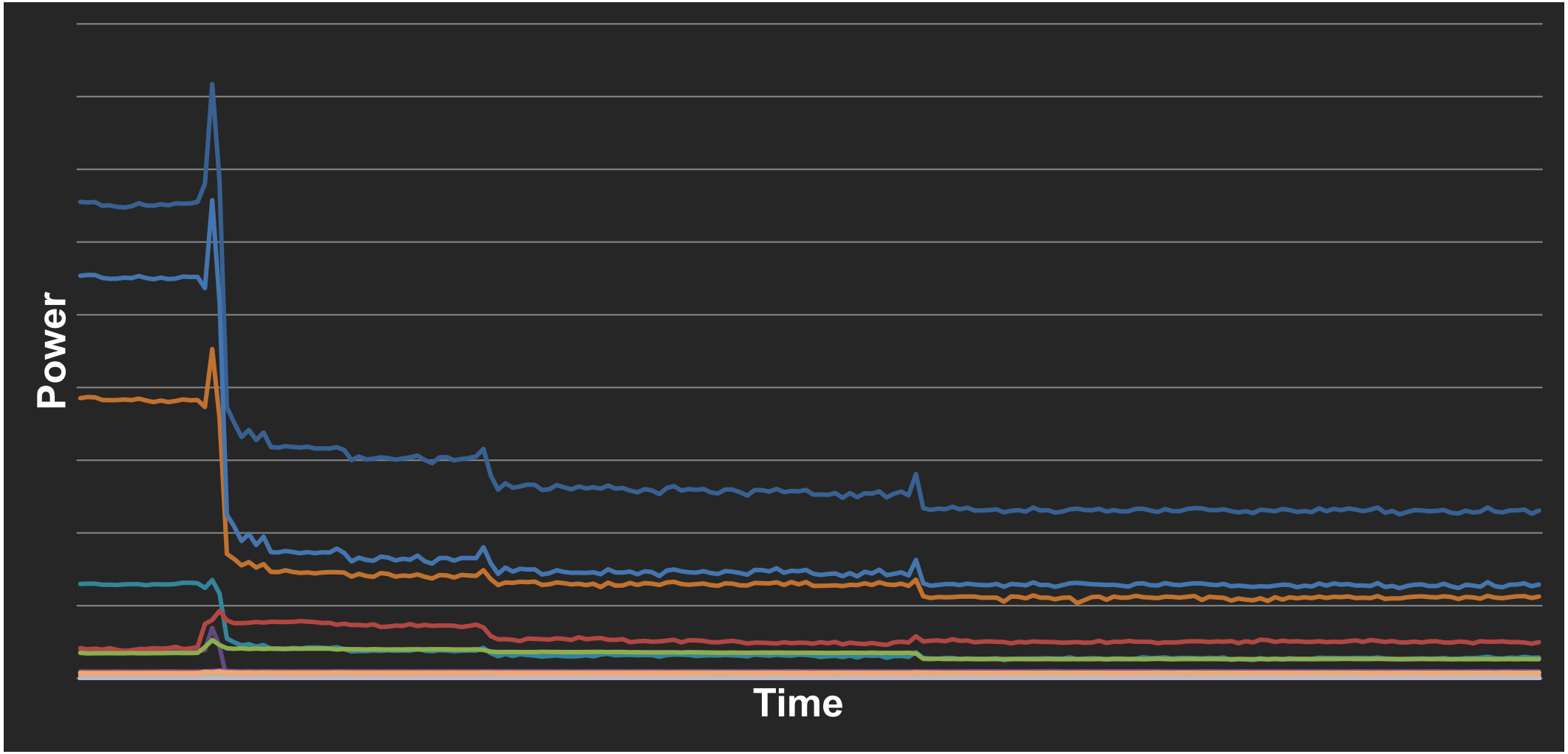


Sweep scenario

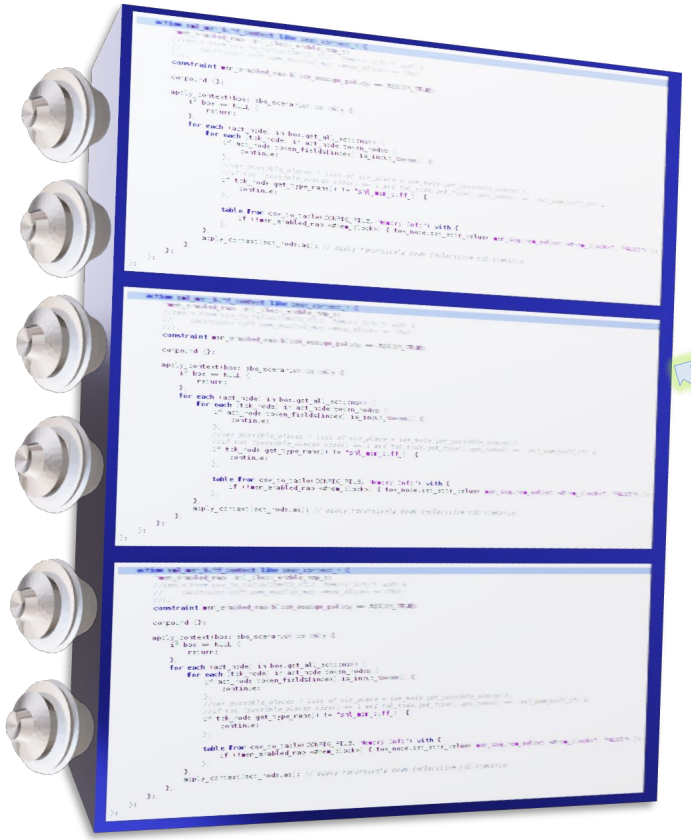
```
action sweep_scenario {
  rand int in [0..NUM_CORES-1] core_num;
  rand int in [1..10000] range;
  rand int in [1..1000] step;
  rand int in [1..1000] base_delay;
  activity {
    parallel {
      replicate (i: NUM_CORES) {
        if (i != this.core_num) {
          do power_down_core with {
            core_num == i;
          };
        }
      }
    }
    repeat (phase: range / step) {
      repeat (2) {
        do power_down_core with {
          core_num        == this.core_num;
          goal_cstate     == CPU_C1;
          transition_case == SWEEP;
          wu_delay == base_delay +
                      phase * step;
        };
```



Power-state transition measurement



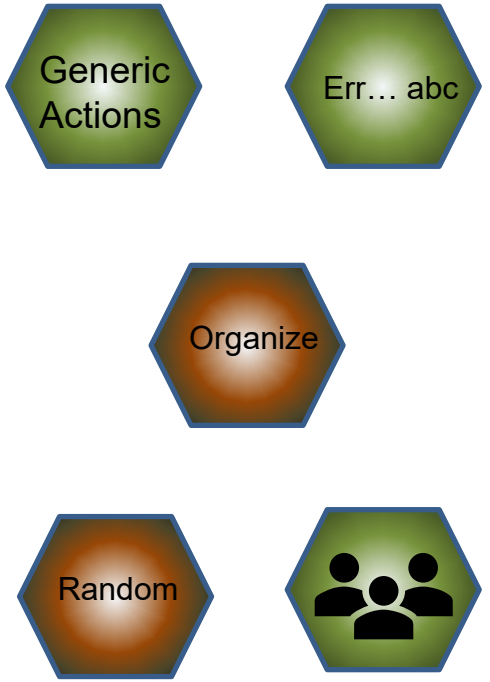
PSS Modeling Guidelines



- Create actions to bring system to known state
  - Encapsulate known state into state object
- Activities that depend on a system feature need to enable the feature using *inputs* or by explicitly using an action
- All generic actions in library should come with control knobs that have sensible defaults
- Project configurations should come from static configuration files
- Separate generic PSS code from project specific PSS

PSS - Real world observations

- Generic action/activity can be complex to create
  - May need procedural description
- Input/output for activities not available yet
- PSS tests prefer complete control of system
  - Baremetal environment
  - Statically allocated and scheduled tests
- Randomization should be carefully thought through
  - Test generation time has potential to increase exponentially
- Constraint solver errors can be terse
- Multi-discipline team collaboration needed to create baremetal library



PSS - Conclusions

- Excellent capability for composing test scenarios
  - Lab bring-up engineers or architects can easily craft scenarios and compose them quickly
    - Ease of creating scenarios with PSS declarative syntax leads to many new scenarios
- Partial scenario description of activity
  - A test creator doesn't need to know all prerequisites
- Runtime and test generation time coverage reports
  - Very helpful in closing coverage gaps
- Constraints in PSS
  - Provide a concise and precise way to specify system and tests
- Augmented power management firmware tuning tests in lab with new scenarios from PSS
  - Quickly generated new scenarios in lab
  - Improved power management algorithms
- We can use PSS beyond power management verification and tuning

Disclaimer and Attribution

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Attribution