

Verification of Multiple Power Supply Designs Using Power Aware Simulation

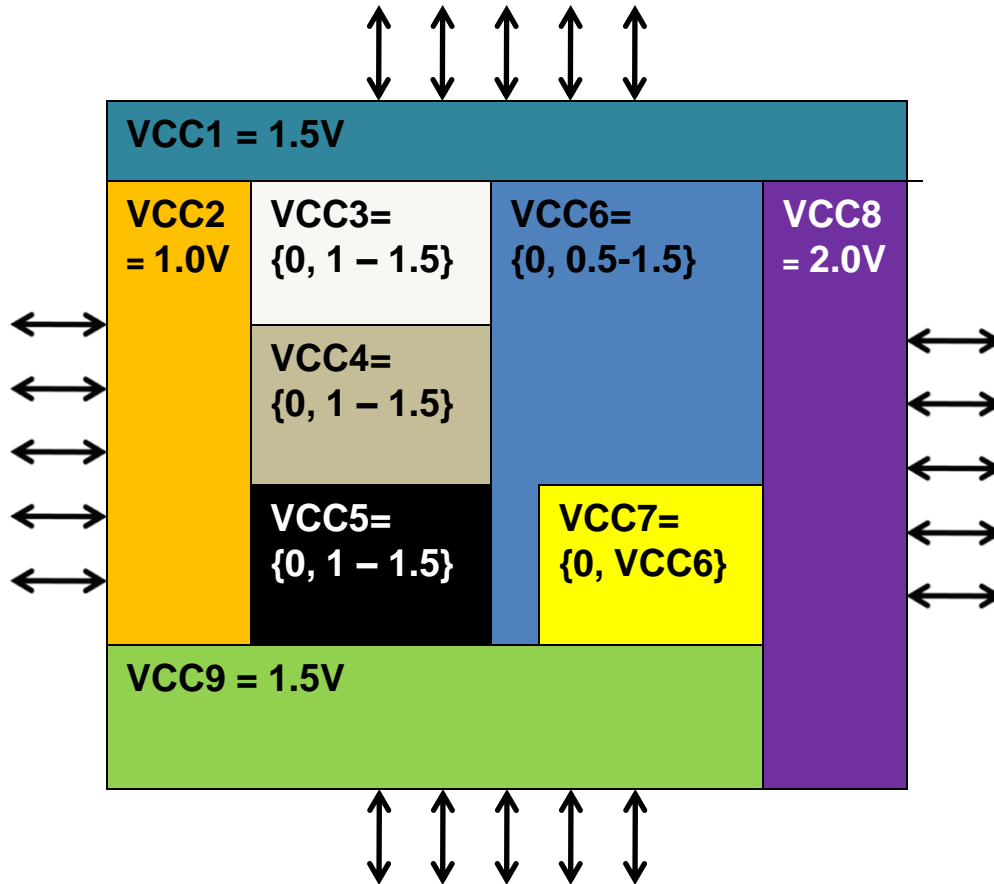
Osama Neiroukh, Aviv Barkai
First.Last@intel.com
Intel Corporation
Haifa, Israel

Outline

- Introduction
 - Multi Power Plane (MPP) Motivation, Verification, and Simulation
- Challenges
 - Partitioning, RTL, Assertions, and Test-Bench
- Defects
 - Examples of bugs found
- Summary
 - EDA tools retrospective, conclusions

Note: Additional details for some slides are in available Notes Pane

Multi Power Plane Motivation



- **Low power design** through ability to reduce supply voltage or completely shut it off for various blocks when inactive
- **IP integration** on same chip including multiple I/O ports

- **Notes:**

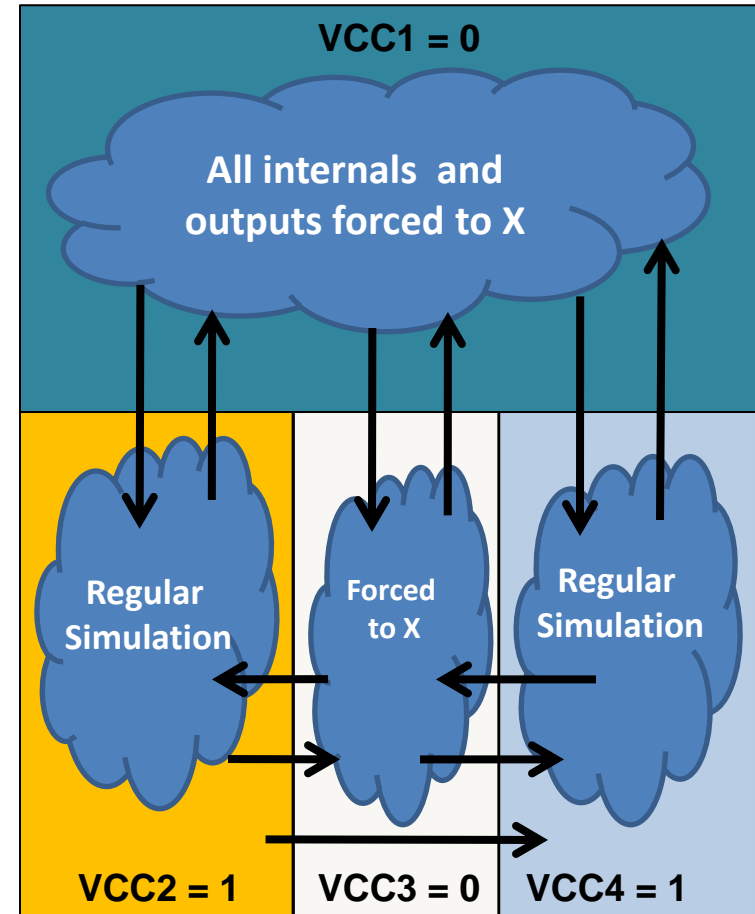
- Number and details of power supplies and chip layout illustrative only, not matching actual design
- Design uses a common ground supply

Multi Power Plane Verification

- MPP designs need multiple types of checks covering logic and circuit aspects to verify correct operation. This work focuses on dynamic logic verification
- Goals for logic verification for MPP designs should include:
 - Power domains can be shut off and turned back on without losing important data or events
 - Logic interactions between power domains operate correctly under all legal on/off scenarios and transitions
 - Un-driven signals emanating from gated domains do not reach live domains

Power Aware Simulation Concept

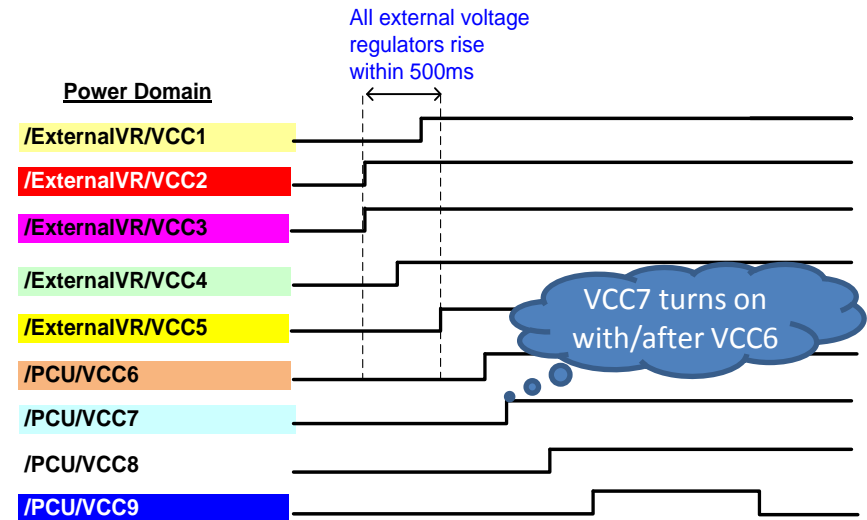
- Power aware simulation mimics circuit behavior more closely by modeling impact of power supply state on logic blocks
- When a block's power is *turned off*
 - Internals and outputs are continuously corrupted to "X"
 - Transitions on inputs are ignored
- When a block's power is *turned on*
 - Starts simulation as if at time-0, then continues regular simulation
- Power Aware Simulation is particularly important for regions of operations where power supplies are being turned on/off
 - Reset sequences starting from all domains turned off until steady state(s)
 - Power-down states where one or more supply domains are switched off and turned back



Power Aware Simulation In Practice

- Setting up power aware simulation entails the following steps:

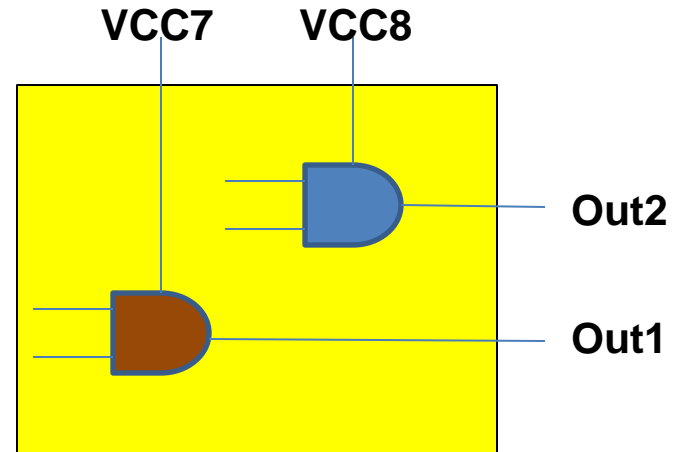
- Determine legal sequences in which power supplies are turned on/off (see example waveform on right)
- Create a set of control signals, one signal per supply, that mirror the turn on/off sequences of the chip's supplies:
 - **externally-controlled supplies** can be modeled with test-bench driven signals
 - **internally-controlled supplies** can often be mapped to hardware-based signals which indicate their state
- Provide simulation engine with a mapping of control signals to logical hierarchies for all supplies (e.g. see table on the right)
- Simulation engine will use mapped control signals as “On/Off” controls for each hierarchy during simulation



Supply Domain	Control Signal	Logic Hierarchies
VCC1	/TestBench/VCC1PwrGood	/top/a; /top/b;
VCC6	/PCU/VCC6	/top/port1
VCC7	/PCU/VCC7	/top/core0

Design Partitioning

- A fundamental enabler for tool-based power-aware simulation is having a single voltage supply per leaf logical hierarchy
 - A multi supply module with logic inside cannot be automatically simulated
- This requires a design partitioning strategy that targets a single power supply assignment per logical block
- Violating this strategy creates difficult tradeoffs
 - Turn such hierarchies always on or with earliest voltage at cost of reduced verification coverage
 - Explicitly code dependency on supplies, which has high RTL cost (must model VCC* signals in RTL and carefully account for all outputs and state elements, as shown on the right)



```
// E.g. two AND gates
// within a multi-Vcc
// hierarchy
//
assign Out1 = VCC7 ?
    (In1 & In2) : 'X;

assign Out2 = VCC8 ?
    (In3 & In4) : 'X;
```

RTL I

- In ordinary simulation, some Verilog initialization constructs execute only once at initialization time (usually “time 0”). These are typically used for instrumentation code
- In an MPP simulation, these must be recoded to execute with each domain turn-on event
- Our approach was to recode them into always blocks that re-trigger on VCC events using a special context-aware macro

```
module bar (...  
  
    logic start[7:0];  
    initial  
        start = 2;  
  
    logic count = 0;
```



```
module bar (...  
  
    logic start[7:0];  
    logic count;  
    `ALWAYS_INITIAL  
        start = 2;  
        count = 0;  
    `END
```

RTL II

- MPP simulation changes semantics of constant 0/1 values, this affects constants behavior as they can get corrupted (or be zero'ed out, depending on variable type)
- This has subtle effects on the manner in which constants are used, such as instrumentation logic, constants passed into module instantiations, or other usages

```
module bar (...  
  
    assign enable = 1;
```

- ▶ In module **bar**, constant **enable** becomes X when module bar is turned off

```
module foo (...  
  
    bar bar0 (  
        .port0(sig0),  
        .port1(sig1),  
        .instance(0)
```

- ▶ In module **foo**, the port **instance** gets an X when module **foo** is turned off, irrespective of instance **bar0**'s power state

Misc RTL

- We encountered other instances of RTL code being affected unexpectedly by power aware simulation. Many were difficult to debug and recode to reproduce intended behavior
- **Example 1:** A clock generator can often be coded thus

```
assign clk = #D ~clk;
```

 - As per Verilog spec, if **D** is undefined (or “X”, as in an MPP simulation and **D** is a logic variable), the statement above becomes an infinite loop
 - Our methodology for these was to create a different code stream for MPP simulation where we use a constant instead of a variable for the delay **D**
- **Example 2:** A multiply-driven wire (or tristate) with one terminal driven from a powered down module will be “X” everywhere with no ability to override it from other RTL or test-bench

Assertions

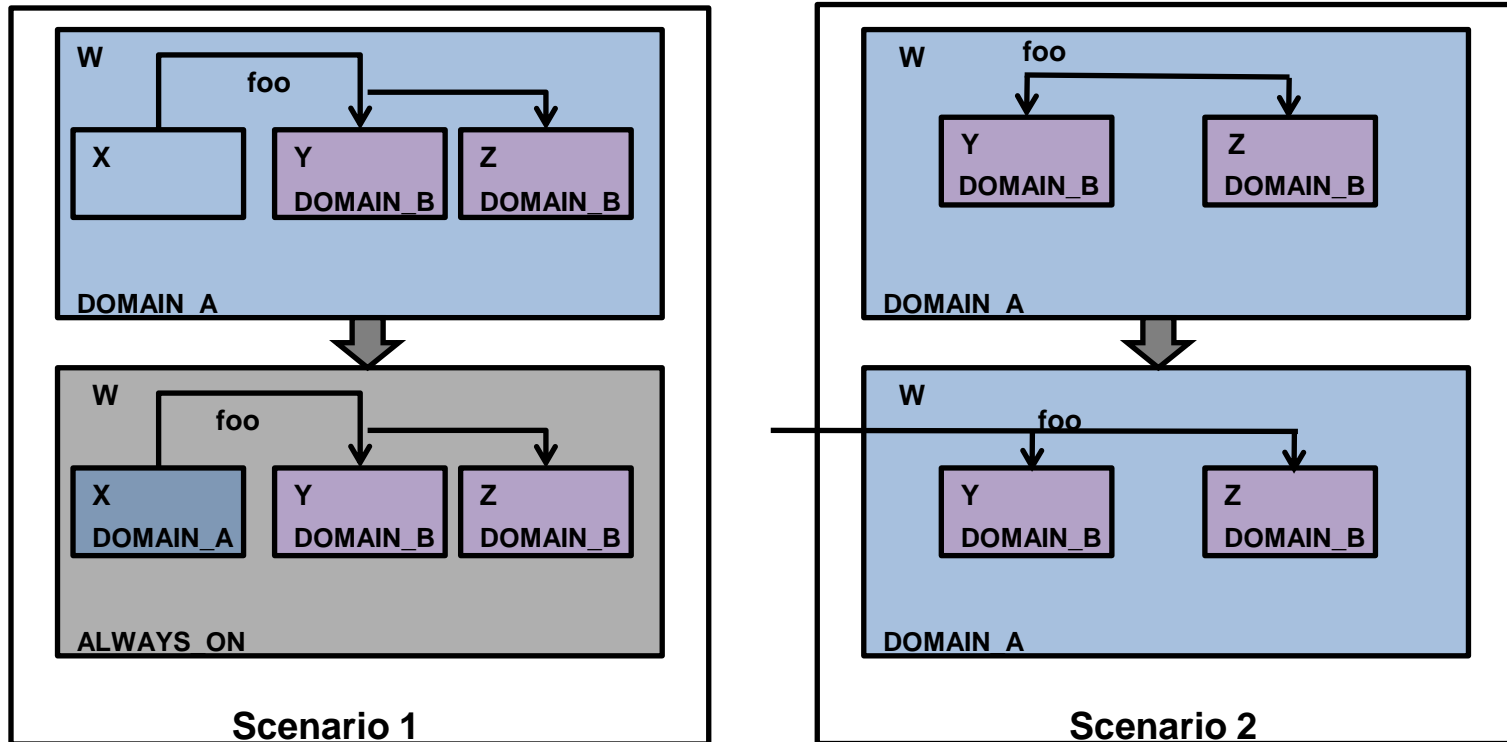
- Assertions are commonly used to check for legal behaviors during logic simulation
- However, managing assertions in a power aware simulation becomes more complex due to the following challenges
 - Assertions are not automatically switched off with surrounding logic by the simulator. Thus they must be written to tolerate undefined values when signals come from powered-down logic
 - Assertions that check low-power behavior should not be accidentally switched off if the test-bench switches off assertions globally or for entire regions in power down states
 - Similarly, instrumentation logic used to compute values necessary for assertions may need to be kept in always-on hierarchies to make sure assertions continue to operate correctly under all conditions

Test-Bench I

- While seeming obvious in retrospect, many operations found within the test-bench need special attention in presence of power-aware simulation
- In principle, all signal read/writes need to become power-aware. This has following implications:
 - Signals driven by test-bench can only be assigned when corresponding hierarchies are turned on (else stimulus will be silently lost as simulator will not queue them up)
 - TB readers (checkers, trackers, monitors,...etc) must all handle possibility of reading values from powered off regions with undefined values

Test-Bench II

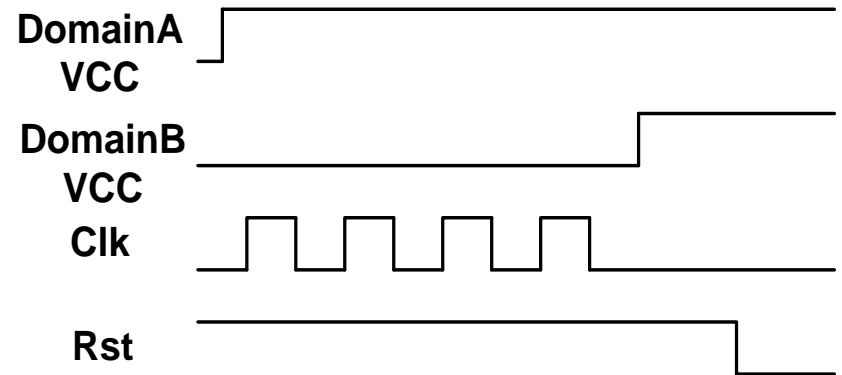
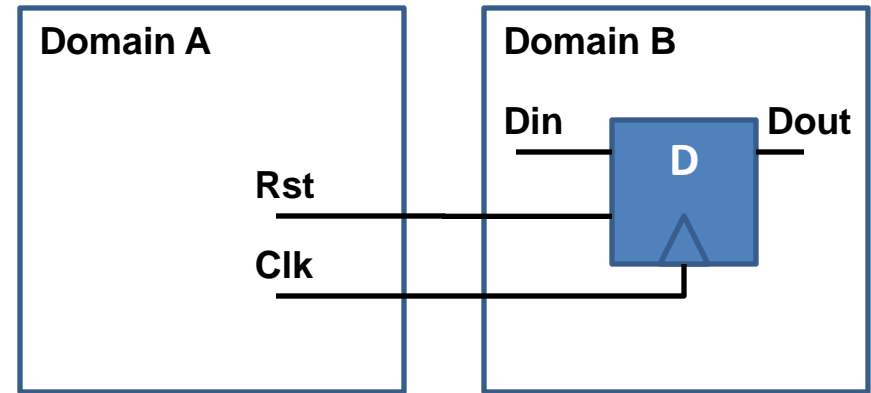
- Test-bench (TB) cannot force a signal internal to a powered-down hierarchy to a known value. To workaround this, either signal or power assignment for hierarchies involved must be changed as shown below



We spent significantly more time debugging simulator/RTL gotchas, assertion and test-bench problems than design problems!

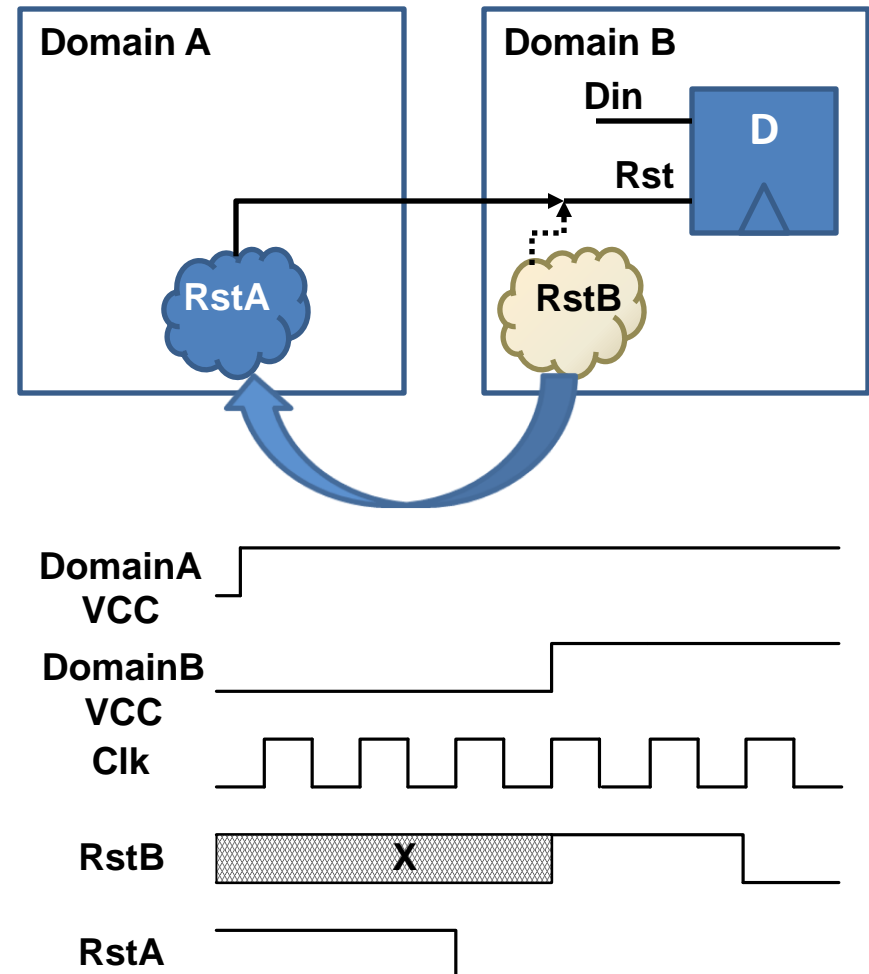
Bug I: Clock Gated Too Early

- **Power Domain A** is turned on ahead of **Power Domain B** and needs to propagate reset on to **Domain B**
- Clock **Clk** gets shut off before **Domain B** is turned on while reset remains asserted
- Receiver flop **D** is turned on after **Clk** was shut off so it remains stuck at **X**
 - Note that **Rst** is synchronous
- This bug cannot be found without power supply modeling. In regular simulation, the reset would be observed by **D** and the flop would reset



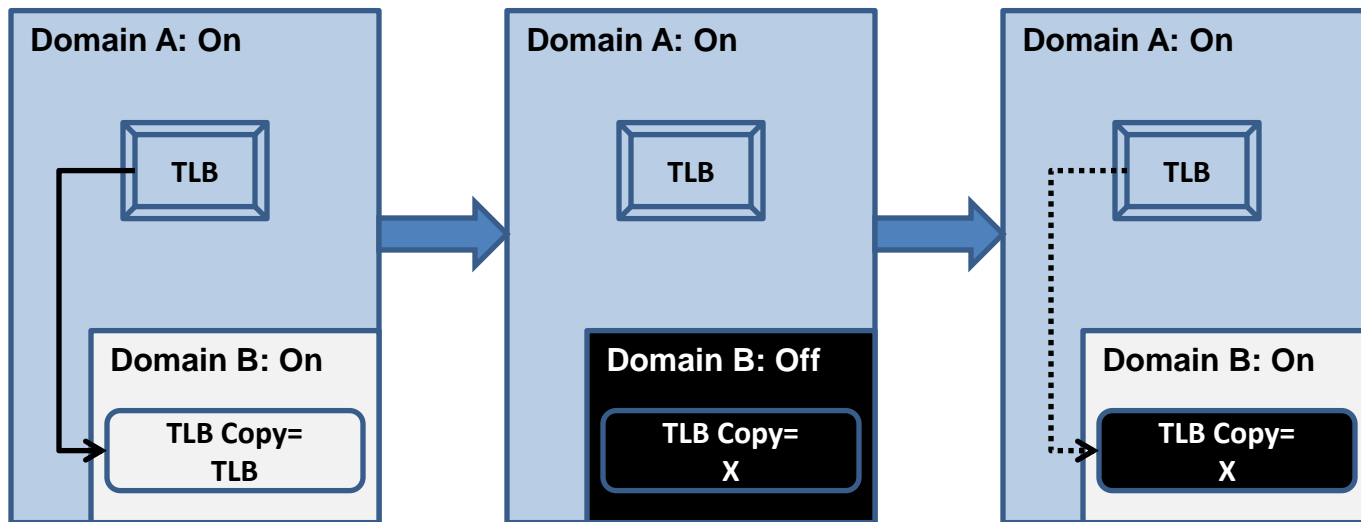
Bug II: Use of Wrong Reset

- Timing/area constraints caused a move of reset logic cone from **domain B (RstB)** to **domain A (RstA)**
- Domain A is turned on before domain B
- The reset logic cone itself was changed in the process causing its output **RstA** to de-assert earlier
- Since **RstA** now de-asserts before **Domain B** is powered on, register **D** is never reset
- The bug was caught at a hierarchy containing both **Domain A** and **Domain B**, and led to discovery that in the test-bench for **Domain B** alone, reset was incorrectly modeled as the bug should have been found there



Bug III: Using a corrupted TLB copy

- TLB lives in “Always On” power **domain A**
- A copy of the TLB is also stored in a different power **domain B**. Transactions access this copy for translation rather than the original
- In this scenario, **domain B** was turned off to save power then turned back on, however TLB copy was not re-synced correctly
- First transactions accessed garbage data in TLB copy in **domain B**



EDA Capabilities Perspective

- The ability of the logic simulation engine to perform power-aware simulation is clearly the key enabler for this effort but it is not sufficient by itself
- Our experience suggests additional work by EDA vendors can facilitate broader adoption of power-aware simulation
- Complementary capabilities might include
 - **RTL Simulation Engines:** Need comprehensive documentation of RTL constructs that behave differently in power-aware simulation to avoid reverse-engineering and debug
 - **Lint Tools:** Highlight problematic RTL constructs for power-aware simulation
 - **RTL Browsing, Waveform, and Debug tools:** Integrate power-dimension and awareness into all RTL development and debug capabilities. Verification engineers are not typically familiar with power architecture of design and need cues to map the power domains and their behavior to logic simulation for effective design and verification of such designs

Conclusions

- Use of Multiple Power Planes has become commonplace in recent electronic designs
- Performing power-aware simulation is critical in presence of multiple power supplies. We found showstopper defects that could not be found any other way, and which would have been exceptionally difficult to discover in post-silicon
- Planning for power-aware simulation requires careful design partitioning to obey single supply per leaf hierarchy as much as possible with backup plans for exceptions
- Providing clear guidelines for RTL and test-bench developers from the start is a must and can significantly reduce overhead associated with power-aware simulation debug and defect hunting later on