Formal Validation of Low-Power Designs

Cadence Design Systems, Inc.
Chris Komar
Tom Anderson
Jerry Church

Session 2.6

1 Abstract

Low-power chip designs are essential for today's portable devices as well as products that must conform to "green" initiatives. The most common techniques for controlling power, including power shut-off, clock gating, and multiple voltage domains, complicate the chip verification process. This paper discusses the intersection of these low-power design techniques and the use of formal analysis for verification. Topics covered include proper handling of assertions crossing power domain boundaries, use of formal analysis to verify power control modules (PCMs), and the role of Common Power Format (CPF) specification for formal analysis. This paper leverages the experience of several Cadence customers in using Incisive Formal Verifier (IFV) for PCM verification as well as assertion technology developed within Cadence.

2 Introduction to Low-Power Design

Given the surging importance of power in today’s shrinking technologies, low power verification is taking on a vital role. Design teams can no longer afford to just worry about dynamic power; they have to pay close attention to leakage power as well. This requires designing in power reduction methodologies at the system architectural stage. This paper surveys some of the power reduction techniques used today and discusses the use of formal analysis to verify proper application of these techniques.

2.1 Basic Power Reduction Techniques

There are a number of techniques used to lower the power consumption of chips fabricated in advanced geometries. Most of the complexity of using these techniques comes from the need to automatically create the necessary low power structures. Any time a design tool has to create logic, based on implied functionality, there is added complexity for verification of this logic. An alternative to the automatic creation of low power structures is the explicit addition of these structures into RTL. But this causes a different set of issues, including lack of flexibility and impact on logic designers.

Clock gating and *operand isolation* are perhaps the most common methods for reducing dynamic power, and have been in use for some time now. Clock gating is efficient for registers that do not change values often, while operand isolation is used when combinational results are not used in the current context. Both these methods require addition of enable signals in the RTL, but current synthesis tools recognize these enables and automatically create the necessary structures. The verification impact is low for these two methods, and the dynamic power savings can be significant.

Synthesis optimizations are used to reduce dynamic power, but may or may not be effective, based on the synthesis tool used. Optimizations such as gate sizing and merging, pin swapping, buffer optimization, and instance reduction often have been used.

Multi voltage threshold (MVT) synthesis can be used to reduce leakage current by replacing low threshold (fast) cells with high threshold (slow) cells on non-critical paths. *Substrate Biasing* is a technique used to reduce sub-threshold leakage current by decreasing the performance of functional blocks when speed is not critical. The impact on verification for all of the techniques is very low.

2.2 Advanced Power Reduction Techniques

Using *multiple supply voltages (MSV)* allows for reduced dynamic power on non-critical functionality. This can provide substantial power savings, but adds a high degree of difficulty to the design process. A large amount of "what-if" analysis needs to be done to make sure lower voltage blocks will still meet timing and area requirements. There is a need for insertion of a level shifter on every signal crossing voltage domains, a process that must be verified. The MSV technique is considered a more advanced method since it also has a high impact on architectural analysis, synthesis, place and route, and test.

Dynamic voltage and frequency scaling (DVFS) can save substantial dynamic and leakage power, but does come with added implementation and verification complexity. The latency to switch from one voltage/frequency setting to another must be weighed against the cost. *Adaptive voltage scaling (AVS)* is a method to reduce the voltage supply by monitoring process quality and temperature and provide feedback to a controller that can reduce voltage if it will have no impact on performance. Both DVFS and AVS require level shifter assertion and have some impact on verification.

On balance, *power shut off (PSO)* is the most effective way to save both dynamic and leakage power. The technique is simple in concept: power down portions of the chip not currently needed. The implications of PSO reach up the very earliest stages of chip development, since the system architect needs to determine what portions of the design can actually be shut off. Other questions that must be addressed include:

- How many power switches will be necessary for shut-off?
- Is there a need to maintain some state in the shut-off blocks?
- If so, is there enough time to scan in and then scan out the register contents to and from a memory, or are state retention power gates needed?
- Will outputs of the shut-off blocks need to be isolated?
- Will the chip need to be tested with all power domains on at the same time, and will this cause problems on the tester?

Although the scope of verifying proper PSO implementation is quite broad, the next part of this paper focuses on verifying the operation of the power control module (PCM). This portion of the chip supplies the power-up and power-down signals to the different power domains and also controls save and restore of any state that must be maintained within powered-down blocks. The final portion of this paper discusses the ramifications of PSO in terms of power domain interaction that places additional requirements on the user as well as formal analysis and simulation tools.

3 Power Control Module Verification

3.1 PCM Overview

A PCM is often used to control the power modes of a chip. In some applications there may be a single controller while other applications may have several PCMs to manage the power modes. A PCM is typically a slave to an embedded processor that ultimately controls when to initiate the power cycling. In this way, the PCM receives some indication to initiate a power sequence and then carries out the detail sequencing of the power cycle. This can be shown in Figure 1 below.

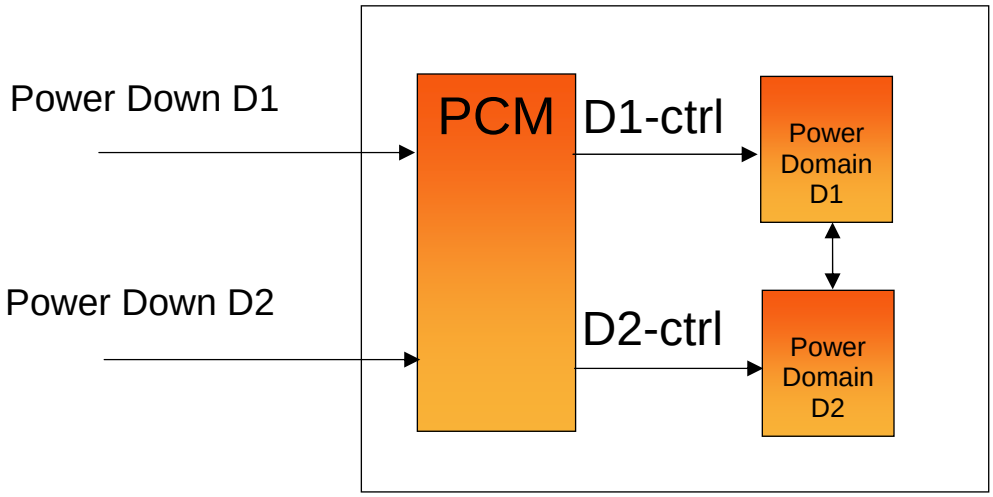

Figure 1 : PCM Example

In this example, the PCM receives instructions to initiate a power cycle as represented by the signals in the left hand side. Upon receiving the instruction, the PCM will control the detailed power control signals as represented by *D1-ctrl* and *D2-ctrl*.

3.2 Implementation

The PCM is often implemented as a finite state machine (FSM). Depending on the number of power domains and power modes (combinations of power domains that will be powered up or down), this FSM can become quite complex. In addition, this is a very critical piece of functionality for the chip. To address the complexity and importance of this block being correct, formal analysis should used as a key part of its verification.

Many formal analysis tools today have the ability to automatic extract and verify general behaviors about an FSM. In particular, a formal analysis tool should be able to validate the following:

- State Reachability: From an initial state, can every state in the state machine be reached?
- Arc Reachability: From an initial state, can every defined transition be exercised?
- Deadlock: Is there any way to enter a state and not be able to exit that state?

Any of these conditions could easily lead to silicon failure. Leveraging formal analysis to automatically verify these conditions provides significant confidence in the FSM.

3.3 Power Cycle Sequence

As described in the previous section, the FSM implementation of the PCM is a good target for formal analysis. Another good target is the actual control signals that the PCM generates to control the power domains. Before discussing this further consider an example waveform of the PCM control signals as shown in Figure 2.

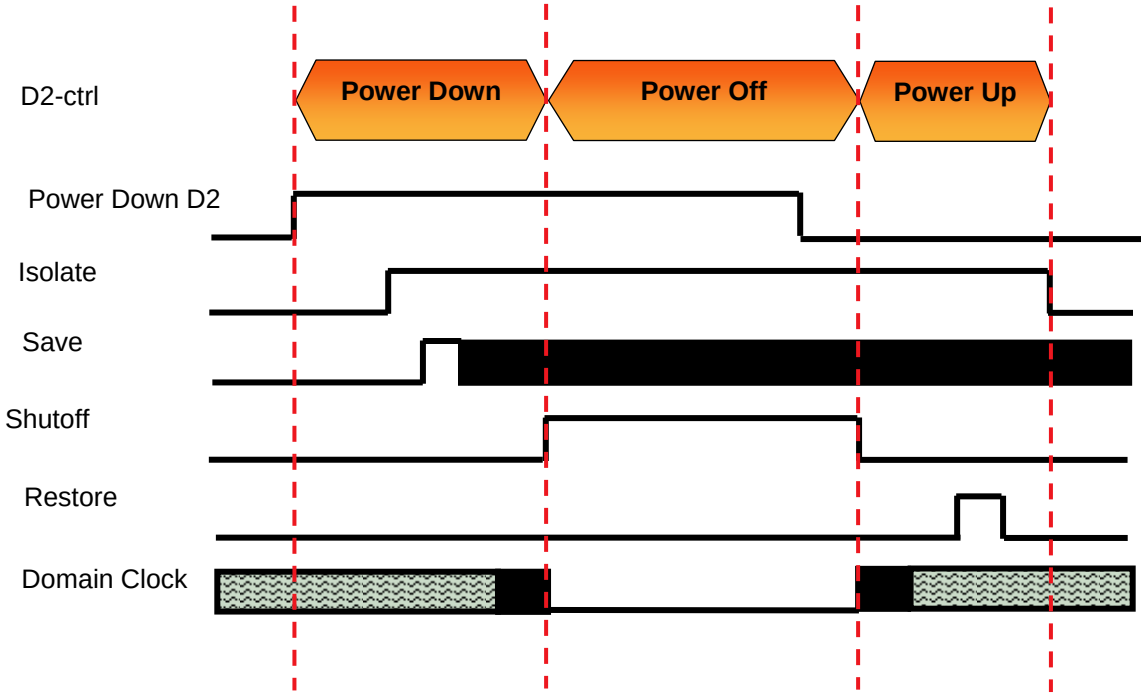

Figure 2 : Power Cycle Sequence Example

At the highest level, there are three phases of a power cycle sequence: Power Down, Power Off, and Power Up. As stated earlier, Power Down is initiated outside the PCM and starts the entire process. The details within these three phases are described below.

Step 1: *Isolate*
Outputs of the domain being shut off must be isolated and fixed to some value

Step 2: *Save*

Often, state retention is required for critical functionality to enable the design to power back up in a known good state

Step 3: *Shutoff*
Once isolation and save has occurred, the clocks to the domain should be shut off and the power removed

Step 4: *Power Up*
The embedded processor will signal the PCM to begin powering up the domain

Step 5: *Restore*
Restore any states that were previously saved during power down

Step 6: *Deassert Isolate*
Deassert the output isolation to regain normal functionality

3.4 Power Cycle Sequence Assertions

With the understanding of the power cycle sequence in the previous section, it is possible to discuss how to apply formal analysis. In addition to automatic checks such as the FSM checks, formal analysis can also target user-defined assertions. The relationships shown between signals described in
Figure 2 can easily be captured as a set of assertions for formal analysis to target. In fact, this line of thinking can be further generalized to say that *any* functional behavior that can be described as a waveform (i.e., a relationship of control signals) is an ideal application for assertions and formal analysis. In other words, this approach is not PCM specific.

The following list of behaviors can be extracted from the power cycle sequence shown in Figure 2:

- Isolation should only start during power down and should remain asserted
- Isolation must come during power down
- Save should only come during power down
- Save must come during power down
- During power down, isolation must occur before save
- No clock can run during power off
- Isolation must remain during power off
- Power off must be a minimum number of cycles
- When power up occurs, isolation must be enabled
- Restore should only happen during power up
- Restore must occur during power up
- Isolation should only be deasserted during power up
- There is a minimal latency between restore and removal of isolation

For the purposes of this paper, the actual assertion syntax implementation of these behaviors is not relevant and, therefore, not provided.

3.5 Packaging the Assertions

In addition to generating a set of assertions to be checked, it also important to think about how to package them for reuse. It is quite common in practice for each PCM within a design or across designs to have a slightly different set of requirements. Because of this, the recommended approach is to capture these assertions in a parameterized Verilog module known as a verification component, or vcomp. In this way, one power vcomp can be used to validate many different PCMs.

As an example, the following parameters would be useful in this context:

- ENABLE_SAVE_RESTORE_EDGE – not all designs/power domains require state retention. If not, assertion checking about this functionality is not required.
- MIN_SHUTOFF_CYCLES – the minimum number of cycles for which a domain should be powered off before it can be powered up again.
- MIN_SAVE_TO_SHUTOFF_CYCLES – the minimum number of cycles required between saving the state of the flops to shutoff. Depending on how the save is done (for example, writing state to a memory), the minimum number of cycles must be defined
- MIN_RESTORE_TO_ISO_CYCLES – similar to the previous parameter but specifically dealing with the restoration of state.

The vcomp can be connected manually for each PCM, a simple process since the signals shown in Figure 2 are clearly known. However, it is also possible to automatically connect the vcomp to the correct signals if a description of the PSO scheme is available. The standard Common Power Format (CPF) file captures the overall power intent for a chip design, including the definition of the power domains and the control signals. It is possible to read a chip's power intent from a CPF file and automatically generate the signals to connect a vcomp to each PCM.

4 Power Domain Interaction

While PSO is an excellent way to control the power consumption in a chip, it does add complexity to the verification process. In particular, there are additional assertion-related functional requirements placed on a block as it cycles through its power sequence. The logic designer must consider whether the assertions within a block should be disabled during the power cycle. The key question is whether the behaviors described by the assertions are expected to be violated during power transitions or the behaviors should be maintained (and therefore checked) even during the power cycling. Specific concerns with this might be:

- Isolation forces 1s and 0s on signals to other domains, which can cause assertions in other domains to fire
- State retention is typically not done for all registers, so if a domain clock is still running, normally unreachable states might be reached causing assertion failures
- Once clocks re-start on power-up, the design may be in an unexpected state that can cause assertion failures
- Upon disabling isolation, the designs from two different domains may be in abnormal states

The next sections of the paper discuss user and tool requirements necessary to address these concerns.

4.1 Designer Considerations

From a logic designer perspective, there are two main considerations. First, if a functional assertion is to hold true for all time, the designer must ensure it is written to work properly when its driving logic is isolated. For example, using the design in Figure 1, consider a two-bit signal **CS** going from Power Domain 1 to Power Domain 2 that is supposed to be one-hot. The following assertion might be written to verify this behavior:

```
output_cs_onehot : assert property (
        @(CS) $onehot(CS));
```

If the isolation logic in Power Domain 1 forces the CS signals to all 0s during power down, this assertion will be violated. In order to handle this situation, the designer might disable the assertion during isolation as follows:

```
output_cs_onehot : assert property (
@(CS) disable iff (iso) $onehot(CS));
```

Another logic designer consideration is ensuring that the assertion returns to a known good state after the domain in which it resides is powered back up. During power down, the assertion may be disabled or not checked because the clock is not running. However, upon power-up, it may be important to get both the design and assertions in a known good state. This can be accomplished by a change as simple as adding a reset in the **`disable iff`** portion of the assertion.

The goal with both these approaches is to separate false assertion failures from real assertion failures. If the logic designer does not take these considerations into account when specifying assertions, he or she is likely to spend far more time analyzing apparent assertion failures in formal analysis and simulation that turn out to be spurious.

4.2 EDA Tool Considerations

While the approaches in the previous section discuss manual user intervention to account for PSO, there are also tool requirements to enable this PSO verification as well as to minimize the amount of manual effort by the designers writing assertions. The first requirement of a formal analysis tool or simulator is to model the PSO structure, in particular the isolation cells at the RTL level. The power domain information required for this modeling can be extracted from a CPF file. An example of this flow is shown in Figure 3. The AND gates represent the isolation cells that are used to drive the output values of Domain 2 to a specific value during power down.

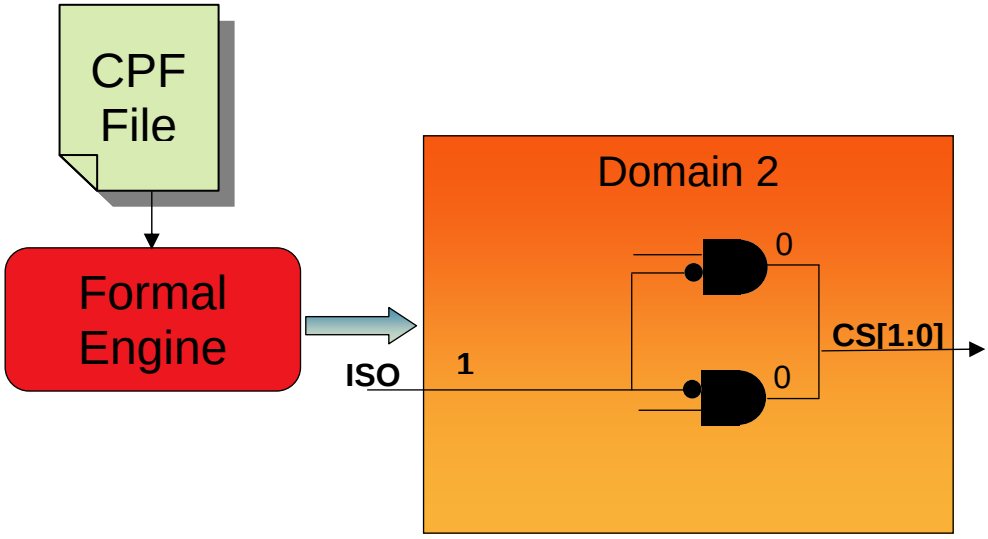

Figure 3 : PSO Flow

In addition to modeling the PSO structure, both formal and simulation tools should provide a built-in mechanism to intelligently and automatically disable assertions to prevent false failures. Control over this mechanism could be provided by tool-specific options or by extensions to the CPF format. Automatic assertion suppression, under user control, can significantly aid the designer in managing assertions in a PSO design.

5 Conclusion

Low-power design is an essential aspect of today's chip designs. Many of the schemes used to help reduce power have minimal impact on verification, but the highly effective PSO approach demands thorough verification, including use of formal analysis. By adopting the assertion-based approaches outlined in this paper, logic designers using PSO can ensure that power-down and power-up scenarios happen properly. This provides a level of design confidence simply not achievable with other verification methods.