Optimizing Area and Power Using Formal Methods

Alan Carlin
Freescale Semiconductor
6501 W Wm Cannon Dr
Austin, TX 78736
+1 (512) 895-6113
alan.carlin@freescale.com

Chris Komar
Cadence Design Systems, Inc
1620 W Fountainhead Pkwy
Tempe, AZ 85262
+1 (480) 704-2203
ckomar@cadence.com

Anuj Singhania
Freescale Semiconductor
6501 W Wm Cannon Dr
Austin, TX 78736
+1 (512) 895-3944
anujsinghania@freescale.com

ABSTRACT
Power consumption is a key differentiator for semiconductor products targeting the embedded market. The combination of system-level requirements and device-level characteristics presents a particular challenge for verifying the implementation of low power design features. Our focus was on the identification of state-retained power gated (SRPG) flip-flops whose clocks are not controllable during entry and exit from state-retention low power modes. Flip-flops which fail the controllability criteria must be replaced with clock-state independent (CSI) flops in the final netlist. The CSI flops have no requirements for clock controllability, however, consume greater power, area, and routing resources. Previously, traditional simulation based approaches had been used to identify CSI flops on earlier designs, but clearly missed a significant portion of the necessary flops. For the Freescale Semiconductor Kinetis™ K40 design, the team adopted a flow based on static formal property checking to identify a provable minimum set of CSI flops.

The methodology included analysis at the module-level to establish a baseline for each individual IP, followed by a single SoC-level analysis to arrive at the actual list of CSI instances for replacement in the final netlist. Module-level analysis served to highlight preventable clock controllability issues within the IP, while SoC-level analysis served to identify connectivity and architectural clock gating issues that only appear within the larger system. Several practical aspects of formal analysis needed to be addressed in order to make the SoC-level flow feasible, particularly with respect to managing run times and memory consumption.

Finally, Formal Equivalence Verification was successfully used to manage the implementation process by correlating the CSI flops identified on the RTL model of the design to instances in the gate netlist. Formal analysis of the design identified considerably more CSI flops compared to simulation based approaches, greatly improving the confidence and reliability of state-retained low power modes. This flow ultimately enabled a SoC to be realized with an optimal number of CSI flops, contributing to maximum functionality with the minimal logic footprint and power consumption.

Keywords
Assertions, SVA, ABV, Formal Verification, Low-Power, SRPG

1. INTRODUCTION
As SoCs become more complex, new challenges are constantly emerging in developing a functionally correct chip that meets performance, area and power requirements. In this paper the authors focus on one specific challenge that has emerged from the low power dimension, and also ties into the area dimension as well.

One aspect of today's low power SoCs is that functionality not in use is shutdown to reduce power. However, there is often a requirement that these functions, when needed again, can be re-energized and continue their operations right where they left off. To meet this requirement, portions of the logic must retain their state while powered down. A specific type of flip-flop known as state-retained power gated (SRPG) flop is used for this purpose. An SRPG flop requires an extra retention signal to indicate when the power is being shut down; and that the value should be saved and be available when powered back up.

There are a couple of different types of SRPG flops that can be used within a SoC, and these are described further below. The challenge discussed here is finding the optimal combination of these different SRPG flops to yield a design that utilizes minimal power and area, without sacrificing any functionality.

2. SRPG BACKGROUND
SRPG flops are used extensively to reduce the leakage current during low-power modes. The architecture of these SRPG flops is to operate the slave latch on a powered-up supply and the master latch on a switchable supply, with the slave latch being completely detached from master latch through pass-gates on assertion of state-retention enable pin. For the slave latch to maintain the state in low-power mode, the latch needs to be put in "non-transparent" mode, which puts a requirement on the state of the clock to the flop. For posedge triggered flops, the clock needs to be held low during the assertion/deassertion of the retention enable. For negedge triggered flops, the clock needs to be held high during assertion/deassertion of the retention enable.

In order to hold the system clocks to their inactive state, the SoC contains a low-power mode controller. This controller follows a sequence of requests that instruct the system clocks be disabled prior to the assertion of retention enable. Ideally, sufficient time elapses following the request to allow system-level clock gating logic to hold the system clocks at their inactive state. Retention enable can then safely assert without the risk of corrupting non-CSI flops. This sequencing of events is illustrated by Figure 1 below.

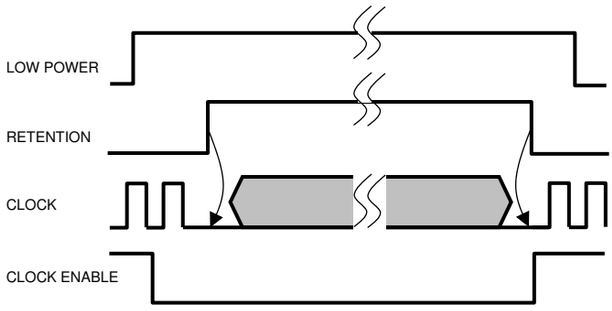

Figure 1. Low power sequencing

While this requirement may sound simple enough to fulfill, it is not always possible to ensure the correct state of the clock. For example, in the case of a clock tree derived from divider logic, the possibility exists that the clock will incorrectly stop at either a high or low state. For this and other situations described later, there is a special SRPG flop, referred to as a CSI (clock state independent) flop to tolerate these cases in the design. CSI flops are larger in size and consume more power compared to regular SRPG flops, but will be able to maintain the state whether the clock is held high or low during the assertion/deassertion of retention enable.

Note that the state-retention scheme needs to guarantee the state of system is exactly the same before and after exit from the low-power mode. Corruption of data in even one flop can be potentially hazardous to the system. Of course, a low-risk approach could be to only use CSI SPRG flops. However, the impact on die-size and routing resources would be prohibitive. Therefore, a technique to find the minimal set of CSI flops is needed to achieve an optimal result in terms of power and area.

3. OVERVIEW OF FORMAL

As with any verification problem there are two requirements for a solution. The first requirement is to provide a means to check that a design under test (DUT) matches its specification. In the simplest form, engineers visualize waveforms to verify the correctness DUT. In more advanced forms of checking, there are automated means such as assertions to ensure the DUT truly reflects the designer's intent. Assertions are an executable specification and represent the golden reference to measure if the design behavior is correct.

The second requirement for a verification solution is a means to exercise the DUT in a meaningful and sufficient way. While most people are comfortable with simulation, there are certain verification problems that even the most advanced, constraint-based random testbench methodologies would have difficulty achieving sufficient coverage of the DUT's behavior. This is where formal verification can be very powerful. Unlike simulation, formal will, by default, exercise all possible DUT states to see if the behavior described by an assertion is maintained by the DUT. While this exhaustive approach can have difficulty scaling to solve large problems, setting logical boundaries on a problem space can enable formal technology to deliver mathematically certain results making a formal approach extremely productive.

4. ASSERTION FLOW

As discussed in the previous section, formal methods were used as the engine to exercise the design in a meaningful and sufficient way. Therefore the challenge for this problem was to create the set of assertions for the formal engines to target. This was done in a two step approach.

4.1 Assertion Generation

The first step required identifying all the unique clock trees within the DUT and generating the associated assertions. A unique clock tree is defined as the clock network that is derived from a common source driver. Given this shared input, all flops driven by this clock tree can be checked by one assertion.

Since there are different requirements for the clock state depending on the polarity of the clock, the next step is to distinguish whether the clock trees were posedge or negedge triggered. As noted in Section 2 above, in the case of a posedge clock tree the clock must be low when the retention signal is asserted. Conversely, for a negedge clock tree the clock must be high when the retention signal is asserted.

The formal tool used for this flow, Cadence's Incisive Formal Verifier (IFV), can identify unique clock networks and their respective polarity. Specifically, via the tool's TCL interface the reporting of the unique clock trees drove the programmatic generation of the assertions. (For the purposes of the given example, the clock tree name corresponding to a unique clock tree identified by the tool will simply be labeled "*unique_clk_tree*".) The process of identifying the unique clocks trees and generating the assertions is shown in Figure 2 below.

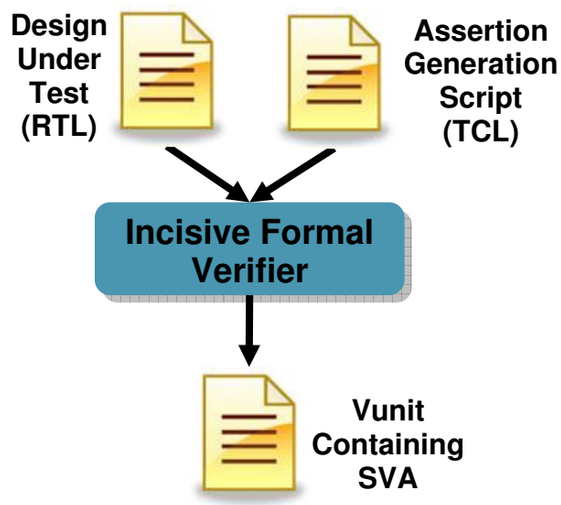

Figure 2. Generating assertions

Notice that the flow above does not discuss any mechanism to input low power intent (CPF or UPF). Consequently, the user must supply the name of the *retention* signal as well as its associated clock (*retention_clk*) to the TCL script. With this information from the user, and the clock trees reported by the tool, the TCL script will generate one SystemVerilog assertion (SVA) for each unique clock tree of the form:

Posedge clock network:

a_SRPG_<*unique_clk_tree*>_p : assert property
(@(posedge <*retention_clk*> or negedge <*retention_clk*>)
$rose(*retention*) |-> !*unique_clk_tree* && !$past(*unique_clk_tree*));

Negedge clock network:
a_SRPG_<*unique_clk_tree*>_n : assert property
(@(posedge <*retention_clk*> or negedge <*retention_clk*>)
$rose(*retention*) |-> *unique_clk_tree* && $past(*unique_clk_tree*));

The output of the process above is a Property Specification Language (PSL) "vunit" that contains the assertions for all the identified clock trees. (Note the "posedge" assertion example is checking the exact behavior described in Figure 1 for the CLOCK *unique_clk_tree*.) A PSL vunit was used to simplify dealing with the out of module references to the unique clock networks. While it might seem more intuitive to use a SystemVerilog bind file as a container and binding mechanism for the SVA, bind files can be cumbersome to automate as an explicit port map needs to be defined and mapped. For convenience, hierarchical paths were used to reference the signals within the design, and the entire vunit was easily bound to the top-most level of design hierarchy.

4.2 Assertion Evaluation

The next step of the flow is to incorporate the vunit containing the generated assertions and to rerun the tool as shown in Figure 3 below.

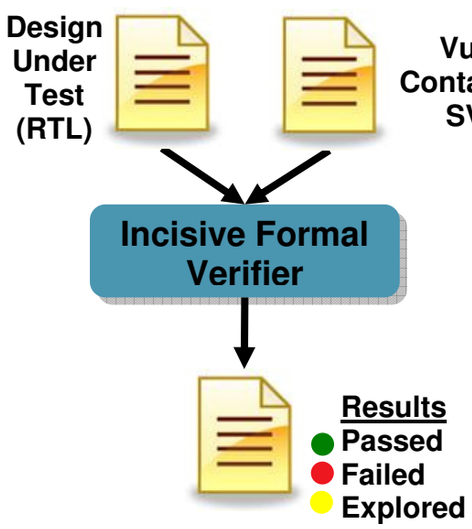

Figure 3. Evaluating the assertions

At this point the user receives the results of each assertion. For assertions that Pass, the user is guaranteed that the required SRPG behavior is met and no further action is needed. In design terms, this means that a CSI flop is not needed for the flops driven by the respective clock tree checked in the assertion. Conversely, for assertions that Fail the user must debug the properties in order to determine the reason for the failure and/or to identify any corrective action. For example, a failure could simply indicate that a CSI flop is needed. However, it could also indicate a bug in the logic intended to control the clocks during the activation/deactivation of the retention signal.

A third category of results is known as "Explored". The definition of an Explored is that after some effort by the tool, the assertion could not be exhaustively proven for all states of the design. The good news is that up to the point where the tool halted its analysis, the assertion did not Fail. This inconclusive result will be discussed further in the next section.

5. PROVING PROPERTIES

For convenience, the properties were grouped by module instance at the SoC-level. A single analysis job would cover an entire hierarchy, enabling the complete analysis to be performed incrementally. This instance-level grouping also enabled prioritization of the effort so that higher risk modules could be analyzed and debugged first, with smaller or lower risk instances being evaluated later.

There are two other benefits to partitioning the properties by module instance: each group of properties can be analyzed independently, allowing parallel jobs to be run on the dispatch system to increase overall throughput. Similarly, the debug effort can be distributed amongst multiple engineers so people can work in parallel, and the output can be vectored to the individuals most familiar with a given module or clock generation architecture.

Running property groups as separate jobs also allows meaningful statistics to be gathered on the run-times of those property groups. This information can be used to further partition the groups in order to level the run-times; and thus minimize the time-to-results when jobs can be executed in parallel. Run-time information can also suggest issues in the design itself—disproportionately high run-times often have an identifiable cause. Table 1 illustrates how significantly the nature of the design can affect the run-times, even after steps have been taken to optimize the analysis.

Table 1. Total property run-times by module

Module	Run-Time	Pass	Fail	Explore
Serial IO	4700 sec	404	40	0
Serial IO	3300 sec	275	0	0
Serial IO	2200 sec	8	10	4
Processor	1800 sec	593	5	0
Parallel IO	1600 sec	25	54	0

Intelligently partitioning the properties, and prioritizing the debug work, also tends to reduce the overall effort required. There is often a common failure mode amongst several properties, or several modules, and addressing similar modules or similar failures can frequently result in addressing an entire category of similar failures with a single corrective action.

6. METHODOLOGY

Recalling that CSI flops require more area and power than their equivalent standard SRPG flops, the objective is to minimize the number of these flops required in the final netlist. The official analysis for cell substitution must be performed on the final integrated SoC design and fed into synthesis to ensure that the logic and hierarchies match for post-synthesis substitution and equivalency checking. The final design also incorporates all of the clock generation, gating, and distribution structures that are critical to the analysis.

Instead of waiting for a completely integrated SoC design, module-level analysis can provide a valuable preview of the expected results at the SoC-level. In fact, module-level analysis can typically be performed months before the host SoC is available, providing time to identify problems and take corrective actions. The module-level analysis environment typically assumes generic low-power modes,

along with simplified clock generation and gating logic. Despite these simplifications, such experiments are valuable as they often reveal how well the endpoints within the module will perform with the external gating logic under nearly-ideal conditions.

Any significant degree of CSI flop requirements identified at the module-level should be corrected within the design, and the analysis repeated. It is highly unlikely that SoC-level analysis will yield results better than those seen at the module-level, barring some form of unanticipated clock network issue. Thus, this process is the best opportunity to easily make changes to the logic to improve the results. Also, since the module-level environment is a generic and simplified model of its eventual integration, the IP owners can assume primary responsibility for performing the analysis and improving the results. Enabling the IP owners to run the analysis in a standalone fashion makes it easier for them to deliver the IP at the level of quality expected for their SoC deliverables.

Once the bulk of the IP within a SoC has seen some level of module-level analysis, the clock manipulation logic within individual modules becomes less of a concern. Instead, the low-power mode gating of the system clocks, and their correct distribution to the modules, becomes the next highest risk area. Weaknesses in the handshaking between the low-power mode controller and the clock generation logic, as well as defects in the system-level clock gating logic itself, must be found. Clock distribution problems typically involve the passage of an uncontrolled clock directly to an IP input, instead of being passed through the system clock generation logic where they can be gated. Polarity mismatches in the inactive state of the clock signals during low-power modes are also common.

Controllability issues identified at the SoC-level tend to be more difficult to correct because there are more parties involved. For example, the low-power mode controller may not always be sequencing the system in the anticipated fashion, or the clock gating logic may require more time to properly gate the slowest clocks to an inactive state. Module clock inputs that require the same frequency and phase may differ in internal polarity—in which case they have different inactive states, and require separate gating logic and routing. Often it is not always be clear which module is at fault, since individually every component behaves according to its requirements, but once integrated holes in said requirements become apparent. Performing the SoC-level analysis in close cooperation with the owner of the system clock generation module and/or clock tree architects is valuable for their ability to resolve issues identified in the system, and also because their detailed knowledge can help to trace an assertion failure to a root cause.

7. COMMON PROBLEMS CAUGHT

The majority of the system clock generation is centralized within a dedicated module for the entire SoC. This module also includes most of the low-power sequencing of the system clocks. As a result, the typical problems with clock controllability when entering low-power modes fell into two categories: clocks being divided or inverted locally within a module, and clocks being sourced directly from an uncontrollable source.

Manipulating clocks locally within a module is not a problem if steps are taken to make endpoint flip-flops be consistent and predictable when entering low-power modes. Solutions include clock dividers designed in such a way that the output clock stops at the inactive state prior to mode entry (although this also requires some form of handshaking with the low-power mode entry controller). If the dividers can not reach and hold an inactive state within a short number of cycles, the divider output needs to be bypassed with the system clock input under the control of the mode controller. Frequently, clock dividers without any form of bypass or shutdown control are encountered in the design and can be addressed by using a CSI flop as shown in Figure 4 below.

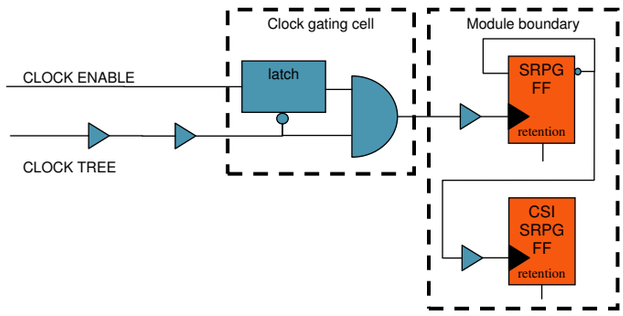

Figure 4. Locally divided clock

Although it is not a recommended practice, it may be possible to locally invert clocks within a module without issue. Posedge and negedge flops can even be sourced from the same system clock input. However, from the perspective of the system clock generator, all of the flip-flop endpoints must share the same inactive edge of the input clock. This requires that all posedge flops use the non-inverted clock, and all negedge flops use a locally inverted clock (or vice-versa). Either way, it is possible to idle the system clock at a known state that places all flip-flop clock endpoints to their inactive state. If this is not possible, local clock inversion may require the use of CSI flops as show below in Figure 5.

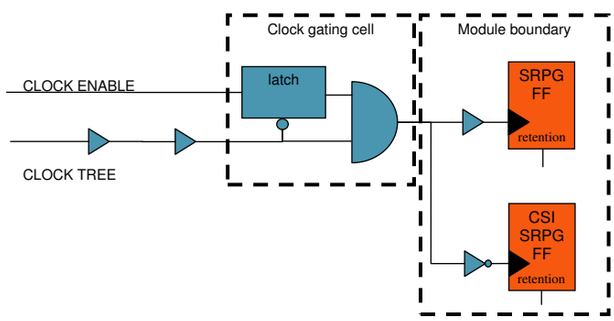

Figure 5. Locally inverted clock

The most obvious source of uncontrolled clocks when entering low-power modes are clock signals that are not sourced from the central clock generation module. These include primary input pins, internal PLLs, and various modules residing in a different power domain that contain no controlling/gating logic. Since these sources cannot, or should not, be shut off upon entry to a low-power mode, remedial steps are the only course of action to guarantee a reliable mode entry. The uncontrolled clock signals may be routed through the clock generation module so that they may be gated prior to fanning out to switchable modules. Or the fanout of the uncontrolled clocks can simply be limited to an acceptable level of endpoints—such endpoints will undoubtedly require CSI flops. Both approaches may even be combined, especially in the case of internal PLLs, where some portion of the clock tree requires minimal latency or custom layout.

8. MANAGING RUN TIMES

Capacity and wall-clock run-times must always be a consideration with static formal tools. For this application, the actual property being proven is quite trivial—the clock must be inactive for a couple of cycles prior to low-power mode entry. The real effort in the proof must therefore lie in the two remaining areas of investigation: sensitizing the properties by entering the low-power mode, and the logic involved in controlling the clock signals. The logic for these portions of the design can be quite extensive, occasionally causing the assertion results to end up as "Explored" described in section 4. While giving the tool more time to run might have allowed the assertions to converge, this approach was not feasible. Consequently, we systematically employed the following run-time management techniques..

The configuration and entry into low-power modes was taken as a given for the clock endpoint controllability analysis. Simulation regressions were testament to the correct sequencing of the power, mode, clock, and wakeup logic within the design. As a result, the majority of that logic could be excluded from formal analysis of the clock endpoints. These modules were either "black-boxed" to completely removed them from consideration, or individual signals were selectively cut and treated as primary inputs. Relevant signals could then be constrained appropriately to prime the design for entry into a specific low-power mode. The only control left to the tool was the decision of when to initiate the low-power mode entry sequence; the entry sequence would then complete in a short and predictable number of cycles following the request.

Also note that the clock controllability logic itself can influence the tool's analysis effort and run-times. While considerable care must be taken applying optimizations here (since this is the very logic that defines the clock behavior on entry into low-power modes), such optimizations can – and did -- have a significant impact on run-times.

The primary means of optimizing the clock control logic analysis was the use of cutpoints. Cutpoints create a virtual primary input to the design at a specific hierarchical location, eliminating the need to analyze the cone of influence up to that point. The primary consequence of cutpoint insertion is that the analysis of downstream logic is considerably easier and faster. The secondary consequence of inserting a cutpoint is that the analysis becomes more pessimistic; the design can now be analyzed assuming the worst-possible case behavior of a cutpointed signal, often exceeding the possible conditions that the actual design can generate. Cutpoints must therefore be employed judiciously, since analyzing the entire design under worst-possible case conditions would yield 100% CSI flop substitution—rendering the effort invested into the formal analysis entirely pointless since complete substitution requires no analysis whatsoever.

The insertion of cutpoints for our analysis was performed in three distinct phases, each targeting a different degree of performance optimization. First, the source for all of the system clocks were cut prior to their entry point to the clock gates within the central clock generation module. As the purpose of those clock gates is to stop the output clock to its inactive state prior to low-power mode entry, the actual clock signal that they are gating should be irrelevant. In our system, the clock gates, themselves, are the last stage in a series of clock sources, clock selection logic, and programmable clock dividers. As a result, a cutpoint immediately before the gating cells eliminates a surprising amount of complex logic related to producing the actual clock signal as shown in Figure 6 below.

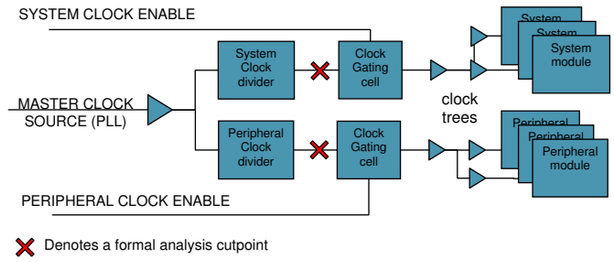

Figure 6. Bypassing system clock generation

The second phase of cutpoint insertion was targeted at general system controllability. Cutpoints were inserted, with accompanying constraints, to keep the system out of reset, out of test modes, and prevent premature wakeup from low-power modes—none of these situations are interesting for the analysis. Also, all of the primary system and peripheral busses were cut at the initiator. Many modules have some degree of sensitivity to their system or configuration busses, and cutpoints here allowed the tool to directly control bus transactions to the targets; no analysis of the initiators was required to wring out a useful sequence of transactions. Figure 7 illustrates some of these cutpoints, in order to highlight the degree of upstream logic being optimized out of the analysis.

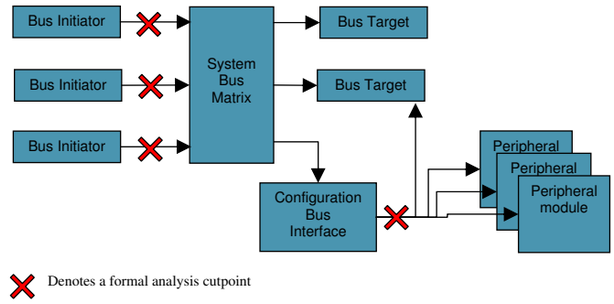

Figure 7. Bus Connectivity Optimization

The third and final phase of cutpoint insertion was targeted at specific clock proofs that exhibited excessively high run-times, or failed to converge to a definitive Pass/Fail. This required individual investigation of each offending assertion, tracing the clock tree, and trying to identify locations where the tool may be spending too much time. Fortunately, some common patterns emerged: there was always a gate, multiplexor, or divider in the clock path; there was always a state machine generating control signals for the logic in the clock path; and the original designer for the module was usually not available for consultation. The simplest resolution was to add a cut-point to the inputs for the logic in the clock path. This removed the state machines from the cone of influence, which improved the run-time(s) significantly. However, the proofs became more pessimistic, and the risk of false-failures became a definite possibility.

Despite these risks, the run-time improvements resulting from the intelligent use of black-boxes and cutpoint insertion were undeniable. Proofs that ran out of memory or failed to converge after hours of analysis often resolved in *seconds* after cutpoints were been applied. Constraints must be applied with the greatest care possible to ensure that you do not invalidate the results of your analysis. Cutpoints, however, increase pessimism and generally reduce run-times, making them relatively safer to employ. It is nearly impossible to precisely quantify, but our best guess is that no

more than 2% of our final CSI flop count is the result of cutpoint-induced pessimism. In contrast, had cutpoints not been used, the resulting non-convergent proofs would have increased our CSI flop count by 10% or more. In short, adoption of this technique itself resulted in significant power and area savings onboard the SoC.

9. RTL-TO-GATE EQUIVALENCE

In addition to identifying the unique clock trees, the formal analysis flow also generates a list of flip-flop instances that need to be mapped to CSI flops. Given this list, there are still three more hurdles to overcome: ensuring the instance path is understood by the implementation tools, forcing the implementation tools to use CSI flops on those defined instances, and checking that only and all the defined list of instances are mapped to the CSI flops.

Although the assertion analysis flow was independent of low-power intent specification, our implementation methodology is based on Common Power Format (CPF) to define, implement and verify the power intent of the design. CPF provides a method to define the state retention elements in the library and the design-level state retention rules. State-retention cells can have a user-defined text attribute which allows flexibility to refer to a group of cells with the same attribute. For example, we chose to define the non-CSI flops in the library with the attribute “srpg_noncsi” and the CSI flops with the attribute “srpg_csi”. Retention rules can be generic covering all sequential instances in a design or can be targeted for a specific list of sequential instances. We defined retention rules for all sequential instances excluding the required CSI list in a generic rule with cells having the attribute “srpg_noncsi”, and used a targeted retention rule for CSI list of instances with cells having attribute “srpg_csi”. With these rules, implementation tools were able to map the registers in RTL to functionally equivalent CSI or non-CSI flops.

There are some differences in name mapping between RTL elaboration by verification and implementation tools. For example, the verification tools inserted a logical hierarchy for every “generate” style construct; whereas the implementation tools did not insert these additional hierarchies but named the register in a unique way to incorporate the label from generate statements. Implementation tools may also use a different hierarchy separators or array delimiters. To resolve these issues, we post-processed the CSI register list generated by the formal analysis to generate a list in accordance with the naming conventions used by the implementation tools.

As we used a post-processed instance list for driving the implementation and an unprocessed version for RTL simulation, we required a way to check the consistency of the retention rule mapping between RTL verification and the implementation. We used Conformal from Cadence to perform this analysis and to ensure that the implementation has CSI flops on all required instances. The unprocessed list of instances was used for Conformal and a list of name mapping rules were defined in tool framework to match the post-processing done on the instance list. Provided with the design netlist (from implementation tools), the CPF (for retention rules) and the name mapping rules, the tool is capable of checking each instance in design and reporting incorrect substitution of CSI with non-CSI flops.

10. CONCLUSION

Compared to earlier simulation based techniques, formal techniques at the SoC-level were able to detect approximately five times as many flops with CSI requirements. No flops identified with simulation were missed by formal, which was an important initial validation of the flow. Formal analysis results were also completely reproducible between revisions of the design and testbench, something that was rarely achieved with simulation regressions. This complete and exhaustive analysis allowed us to replace a specific list of identified cells, instead of simply replacing entire levels of suspect hierarchy. In the end, the results of formal analysis allows us to replace only the minimum necessary set of flop instances, and have far higher confidence in the finished product.

We have taped-out multiple products successfully with this flow, and the silicon validation results have been entirely positive. More importantly, the thoroughness of formal analysis has allowed us to confidently reduce the CSI flop counts over time. Designs have been adjusted specifically to improve the clock controllability in low-power modes because of the significant area savings that has been demonstrated. This has been a valuable contribution to our low-power implementation methodology, and one that will continue to enhance our low-power product families.

Moving forward, we hope to improve our low-power implementation tools to reduce the perplexing variety in net and hierarchy naming conventions. Mapping the hierarchical instances between the various views of the design is a critical step in applying the information derived from formal analysis, and unexpected name changes create undesirable opportunities for errors. Reading the CPF file into the formal analysis steps would also help to ensure that the low-power mode signaling and sequencing precisely matches the eventual implementation.

11. ACKNOWLEDGMENTS

Thanks to Jose Barandiaran of Cadence Design Systems, Inc. for his expertise in TCL to make this flow a reality. Also thanks to Joseph Hupcey III of Cadence Design Systems, Inc. and Hugo Cavalcanti of Freescale Semiconductor for their feedback in reviewing the paper.

12. REFERENCES

Padhye, et al., State retention within a data processing system, U.S. Patent 7,183,825, February 27, 2007.