MEASURING ACTIVE POWER USING PT PX
A USER PERSPECTIVE

Duane E. Galbi (duane.e.galbi@intel.com)

Karthik Kannan (karthik.k.kannan@intel.com)

Intel Massachusetts, Inc.

Hudson, MA

ABSTRACT

The authors will review the use of PrimeTime PX (PT PX) to measure active power. With PT PX, one will always get a power value. However, one cannot always rely on this initial power measurement. We will review the conditions under which the generated power numbers are inaccurate and provide strategies to avoid these pitfalls. We will describe a forced gate based flow using VCS and PT PX which enables one to accurately measure power and effectively debug the power measurement results, and describe how this flow can be extended to accurately measure glitch power.

TABLE OF CONTENTS

INTRODUCTION – ACTIVE POWER VS. LEAKAGE POWER MEASUREMENT METHODOLOGY

Static leakage estimation is a fairly well solved problem with easy to use solutions. The full Synopsys tool suite supports static state probability propagation [1] for calculating gate-level leakage. Simple summations of the total device width or the total device width attached to power and ground supplies can be used to calculate reasonable leakage approximations. Full state dependent leakage is typically an output of most active gate-level power measurement flows [1]. Leakage power is not nearly as pattern dependent as active power. Hence, any method which gives a consistent estimation of whether an inverter is driving high or low can do a reasonable job at estimating leakage. Fine-grained state-dependent power gating would complicate leakage power determination, but the authors have yet to see much of this.

However, the full calculation of active gate-level power is not nearly as mature. The accurate calculation of active power requires determining an accurate activity at each gate-level node, and this is much more pattern dependent than leakage calculations. The three most common methods we have seen to determine the activity at each node are: static propagation of activity factor, full gate-level simulations, and forced-gate gate-level simulations. PrimeTime PX (PT PX) can be used to support all three of these methods.

PT PX (as well as the rest of the Synopsys tool set) supports static propagation of the activity factor and it is a fairly straightforward extension of the static state probability propagation used in leakage estimation. One creates a mapping between RTL nodes and gate-level nodes and forces the probability on the mapped gate-level nodes based on the results of an RTL simulation. However, due to the pattern dependence of active power, one needs to map the state of enough nodes such that the probability propagation from these nodes validly covers all other nodes. Mapping and forcing just the sequential nodes, EBB outputs, and top level inputs are typically not sufficient. Adding other mapping points, such as interfaces between modules, can increase the accuracy of static propagation, but the authors have yet to find a method to determine a set of sufficient additional mapping points.

Full gate-level simulation (GLS) is an obvious choice to get the activity at each gate-level node. However, full GLS runs are notoriously difficult to get running correctly, and this approach was not considered viable for the large number of tests we hope to run.

A forced-gate GLS supplies the same power estimation accuracy as a full GLS without the need to get the full GLS working. In this approach, a RTL-to-gate-level mapping of external interface signals, state nodes, and embedded black box (EBB) outputs is created, and the cycle-by-cycle values of these signals are collected during an RTL simulation. This collection of nodes is sufficient to define the state of the non-EBB part of the gate-level logic at any point in time. A GLS is then run where the states of these mapped signals are forced to the value collected during the RTL simulation, and the simulator calculates the non-forced nodes based on the value of the forced nodes. In reality, one does not need to have a complete mapping of all the gate-level states to an equivalent RTL node, because the GLS simulation can calculate the value of state-nodes base on the rest of the mapped values. The result of the GLS is the activity of all the gate-level nodes. This activity can then be used by a power analysis tool to accurately calculate active power.

We chose to implement a power analysis strategy based upon the forced-gate GLS methodology using VCS B-2008.12 and PT PX C-2009.06-SP1. We will review why the existing forced-gate methodologies were not sufficient, what could be done to improve them, and the issues we had with both VCS and PT PX while implementing this methodology.

SEARCH FOR A FORCED-GATE ACTIVE POWER MEASUREMENT FLOW

The choice of a forced-gate flow for active power measurement seemed obvious; however, finding an appropriate flow was not as easy. The flow needs to address the three main steps of the forced-gate methodology: creating a state mapping between RTL simulation VCD and gate-level netlist, running a GLS forcing the mapped nodes, and then using the GLS results to calculate active power. The PT PX cycle-accurate peak power (CAPP) flow [2] seemed to provide the ultimate forced-gate power flow, and promised to elegantly solve the three main steps of the forced-gate process.

In order to create the state mapping between the RTL simulation VCD and the gate-level netlist, the Design Compiler/Design Compiler Topographical (DC/DCT) synthesis tool has the ability to keep track of all the name remapping that happens to the RTL during synthesis, and writes out a SAIF map file which records these changes. The SAIF map file output from DC/DCT can then be read into PT PX to allow PT PX to correlate RTL VCD names to their gate-level equivalent names. This problem gets complicated by the fact that the simulation and synthesis models for sequential elements are different in our database. The RTL simulation cannot be run with the synthesis models because the synthesis models use Intel-specific width-based array instantiation enhancements which are recognized by DC/DCT, but not by VCS.

With the SAIF map file, PT PX can then read in a RTL VCD simulation file, map the appropriate nodes to the appropriate gate-level states, and logically propagate the RTL mapped nodes to the unmapped nodes. This feature is called “cycle-accurate peak power” or CAPP flow. PT PX logically propagates the nodes by, in essence, running an internal GLS starting from the forced nodes. PT PX then internally captures this simulation data, eliminating any potential data volume problems inherent in using a large VCD from a GLS, and calculates the appropriate instantaneous power and average power.

Unfortunately, the promise of the PT PX CAPP flow was clouded with mapping issues. Debugging mapping issues was very slow, and there were no good ways to determine which RTL nodes were mapped to which gate-level nodes. STARs were filed against PT PX and Power Compiler for mapping issues [3, 4, 5, 6, 7], and we were not confident in using the CAPP flow as we expanded to rest of our design. Although it seemed as though the logic propagation simulation and subsequent power calculation worked as expected, we stopped work on this PT PX flow because we could not validate that the mapping was working appropriately.

Synopsys has since then updated the CAPP flow to address concerns related to RTL-to-gate-name mapping, especially in the D-2009.12 version, where the name mapping file is given preference over automatic name mapping from the SAIF or VCD file. Synopsys has also enhanced the reporting of the name mapping process to specify which design object in the RTL switching activity file (VCD or SAIF) is mapped to which design object in the gate-level design [8]. We plan to reevaluate the CAPP flow in the future. Eventually we expect the CAPP flow to be working appropriately, but for the near term we still expect the added ability to debug a non-integrated flow to outweigh the advantages of the integrated CAPP flow.

In the end, we decided to create our own forced-gate simulation flow using VCS. The steps are:

1a. Run formal comparison, or equivalence checking, to generate a RTL-to-gate mapping file consisting of primary inputs, sequential cell outputs, and EBB outputs

1b. Translate the RTL-to-gate mapping into RTL-simulation-VCD-names-to-gate-names mapping

2a. Take the mapping file and RTL VCD and generate the appropriate inputs to control the VCS GLS (through a custom Verilog Procedural Interface, or VPI, routine) [20,21]

2b. Run VCS and generate gate-level VCD file

2c. Post-process VCD file (bit-blast VCD file), and run `vcd2saif` to generate a SAIF file from the VCD

3. Run PT PX with the gate-level SAIF file to determine active power

In order to ease the generation of the RTL VCD name to gate name mapping (Step 1b), the library simulation models were updated to have a consistent naming convention for the appropriate net or pin within the standard cell model to force. This is not really required, but it did provide a simple solution for the case of sequential cells with an inverted output as well as sequential cells with both an inverted and a non-inverted output. Always forcing a non-inverted internal node of the sequential cell provided a simple way to translate from the internal node supplied by the formal comparison map file to the appropriate node to force during GLS. In addition, the use of a VCS VPI routine to force the nodes simplified this mapping because the VPI routine was designed to be able to force pins, nets, and the state of user defined primitives (UDPs).

One obvious characteristic of our flow is that we annotate switching activity with a SAIF file instead of a VCD or FSDB file (Step 3). A SAIF, or Switching Activity Interchange Format, file describes switching activity statistics instead of the actual toggle activity. We have found that there are multiple bugs in the way PT PX processes a VCD file which can lead to dropped toggles. In one case, when using unit delay models, enough toggles were dropped that we saw about a six percentage difference in total active power between activity annotation with a VCD file versus annotation with a SAIF file. After a detailed debug, we concluded that the power value derived using a SAIF file was correct. It was possible to reduce the issue to a small test case which we have submitted to Synopsys. We have filed multiple bug reports and STARs on the issues we saw. In general, we see ordering and bus related issues leading to dropped toggles, when using VCD or FSDB files to annotate activity into PT PX. We will discuss specific cases in subsequent sections of the paper.

Figure 1 is a flow chart of the forced-gate flow which will be described in more detail in the sections on using the flow to measure power from zero-delay simulation VCD (i.e. non-glitch) and measuring glitch power with full SDF backannotation.

Key Learnings:

1) PT PX cycle accurate peak power (CAPP) flow, although it holds great promise, was not ready for production use due to name mapping issues.
2) In order to allow easy adaptation to a wider variety of requirements, the forced-gate flow should be broken into three distinct steps: create a mapping between RTL VCD and gate-level netlist; given the mapping file, RTL VCD, and the gate-level netlist, run the simulation; given the gate-level VCD, netlist, SPEF, and library collateral, calculate the active power
3) Annotating gate-level activity in PT PX with a SAIF file is a more reliable way than using an FSDB or a VCD file.

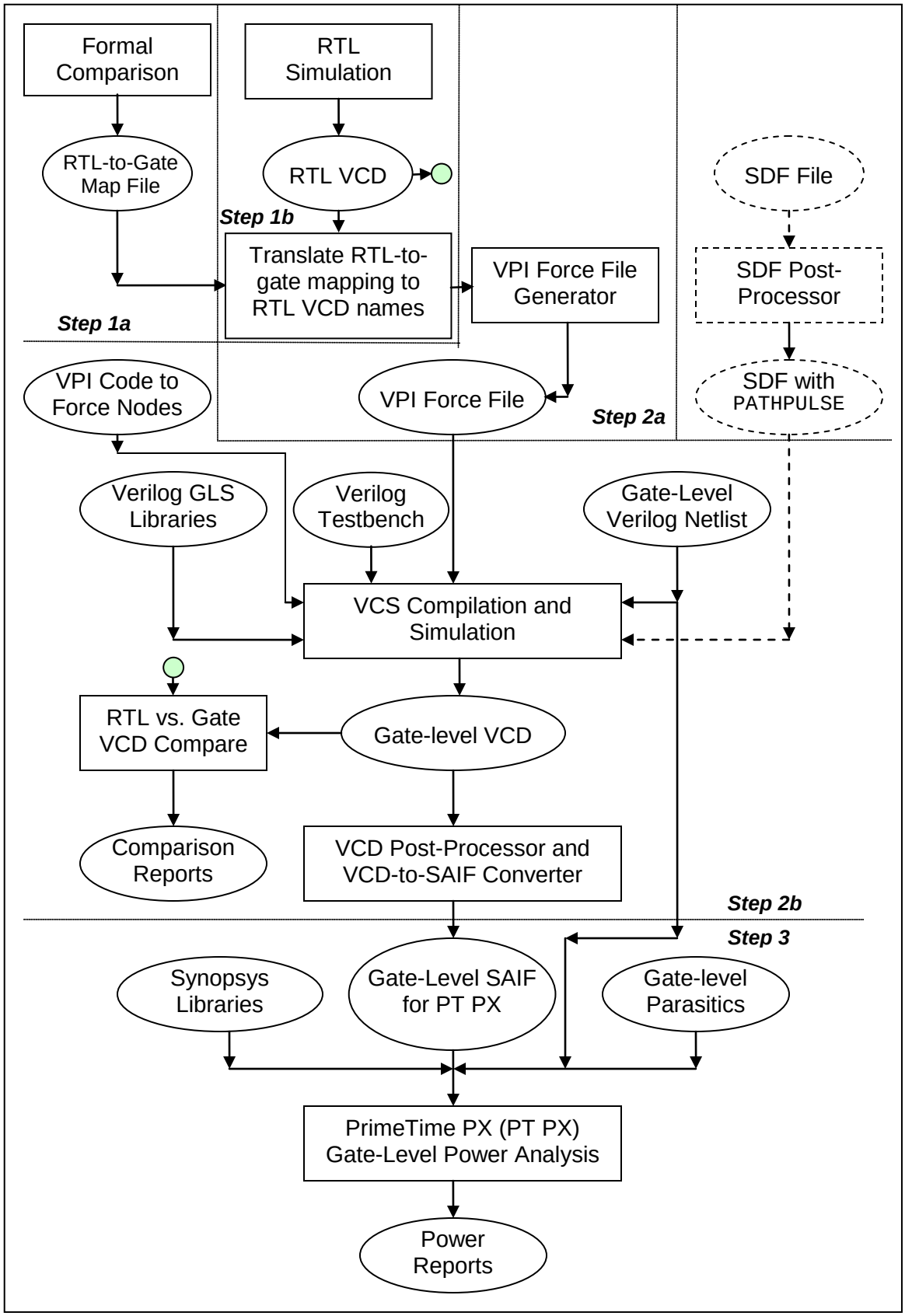

Figure 1: VPI Forced GLS Flow
(Dashed lines depict additional steps for glitch power calculation)

PERFORMING ZERO GATE DELAY POWER (NON-GLITCH POWER) MEASUREMENTS

We started with the power measurement flow using zero gate delay library models. Because it does not model the cell or interconnect delays, it is simpler to simulate, but it still exercises all the same flow steps as the full glitch power flow. The RTL inputs, the RTL VCD to gate-level mapping, and the PT PX measurement flow are exactly the same between these two simulation modes. In the zero gate delay model, we force the output pin of each sequential element; whereas, in the full gate delay model we force a node corresponding to the input of the sequential. The ramifications of this distinction will be described later in the full gate delay simulation section of this paper.

One issue that arises when comparing RTL simulation and GLS runs is that X's are often not handled the same way in RTL as they are in GLS runs [9]. Our solution to this problem is to eliminate, wherever possible, X's from both the RTL simulation and the GLS. VCS provides options (`+vcs+initmem` and `+vcs+initreg`) to initialize the state of registers (SystemVerilog types are treated like registers) and memories. However, these routines do not initialize the state of UDPs. This issue was filed with Synopsys [10]. As a workaround, a VPI routine was created which can initialize the state of all the elements in the simulation. This VPI routine is invoked at the beginning of the RTL simulation to initialize the RTL model. With this method, the RTL VCD used to determine the value of the forced-gate state nodes contains few to no X states.

A formal comparison step was used to create the mapping between the simulation RTL and the gate-level netlist. In order to allow flexibility in which formal comparison tool could be used, we could not depend on the formal comparison hints, an SVF file, which DC/DCT can write out. Instead we added automation to create an appropriate RTL to gate level seed map file.

One key technique for this mapping automation was to read the RTL into DC/DCT and assign the initial unmapped names of the sequential elements to a user attribute. A `change_names` step was then performed, and then this existing user attribute could be used to write out a file giving a direct mapping from the original to the changed sequential element names. Another complication to the mapping was the fact that the simulation modeling and implementation modeling of certain elements was substantially different. Our formal verification is from the simulation model to the gate-level output of the implementation model, so the mapping created needs to understand these different modeling methods. However, by far, the most complicated mapping issues were created by nested RTL `generate` statements and nested `for` loops. The sequential name created often gives an indication of the variables used in the `for` loop or `generate` steps, but they do not give an indication of the order of these indexes. If a coding convention was applied in which the name of the index variable indicated the order of the loop nesting relative to the order of the indexes in the variable name, this would have greatly simplified the initial seed mapping generation. A Perl program was created which reads the original-to-changed-names mapping file created, RTL nodes, and gate nodes needing to be mapped, and attempts multiple heuristics in order to create a seed mapping between the RTL and gate nodes. The mapping does not need to be complete; because if a reasonable seed mapping is supplied, then the internal mapping algorithms of the formal comparison tools can do a good job at completing the rest of the mappings.

After the appropriate mapping is established, the RTL VCD file then needs to be read to provide the appropriate states to force onto the gate-level netlist. We looked at three different ways to do this:

1. Compile the force values directly into the Verilog code using a custom Verilog test bench

2. Use Unified Command Line Interface (UCLI) forces

3. Use a VPI routine to force signals

The custom Verilog test bench ran very fast, but for large test benches it dramatically slowed down VCS compile times and required a VCS model rebuild every time the force inputs changed. If one creates a UCLI force file where each node is forced using a "`force <node-`

name> <value>" statement, the UCLI run time becomes very long due to the fact that VCS looks up the <node-name> for each force statement. A procedure written to use the VPI allows the results of this lookup to be cached and used for all subsequent force statements. As shown in Table 1 below, this allows the VPI approach to complete in nearly the same amount of time as a regular GLS. One can speed up the UCLI simulation time by forcing multiple values for each UCLI force statement. An example of this would be "force {clk} 0@10, 1@20, 0@30, 1@40". However, we decided to stick with a VPI routine due to the flexibility it provides in picking the node or pin to force and the fact that it allows for a more compact force file.

	Regular GLS Run Time	UCLI Forced GLS Run Time	VPI Forced GLS Run Time
500K instance cluster	23 min	Crashed after 24 hours	25 min

Table 1: Comparing Different GLS Run Times

The node trace output of the forced-gate GLS run is then read into PT PX to supply the gate-level node activity. We were originally capturing the trace output into a FSDB file and feeding this FSDB file directly into PT PX. However, this caused subtle issues. PT PX really depends on the ordering of the events in the trace output file to determine which input event in a gate caused the output to change. This allows PT PX to capture arc-dependent power even in a zero gate delay simulation. The order of events in a VCS VCD file corresponds to the order of events in the event-driven simulation, so the inputs which cause the output event to happen are listed before the output event. PT PX depends upon this ordering. Unfortunately, FSDB files do not preserve this ordering of events. Synopsys has talked to SpringSoft (formerly Novas) about this issue. SpringSoft has the capability to add sequence numbers to preserve the ordering in FSDB files, but its FSDB-to-VCD file converter did not consider the sequence numbers. SpringSoft has enhanced its FSDB-to-VCD converter to consider the sequence numbers in the FSDB file beginning with its early 2010 version. Nonetheless, we have standardized on using VCS generated VCD files as the source of PT PX gate-level activity input. It should be noted that the use of sequence numbers causes the FSDB file size to increase by as much as 10%.

In general, if PT PX gets confused about which input event causes the output to transition, it calculates the power of the output transition to be the average of all the potential input combinations assuming a ZERO slope on the inputs [11]. This means the library needs to give reasonable power values for the zero input slope case. One sanity check for a library is to set all the node slopes to zero and measure the power delta between the zero slope and the actual slope conditions. Synopsys has agreed that a zero slope is not a good choice, and an ESTAR [12] has been filed to switch this measurement condition to the average slope of all the controlling inputs.

Other issues encountered in, and filed against PT PX, involved processing of bussed nets and bussed EBB pins where non-existent glitching was produced after parsing the VCD file [13]. Another related issue observed with bussed nets and pins was that PT PX reported incorrect toggle counts for nets connected to bussed inputs of EBBs [14]. In general, we found that PT PX can get confused when there are multiple changes within a single time step and the bus values changed in such a manner that certain bits of the bus have multiple transitions during the time step. Our workaround for this issue is to bit-blast the VCD file, and dump out a value change for each bus bit only when the bit actually changes value. The vcdpost Synopsys utility with the +unique option can be used to accomplish this manipulation.

We have also noticed a case where even if the switching activity was annotated through PT PX set_switching_activity commands (instead of reading in a SAIF or VCD file), PT PX reports the wrong cell internal power for two identical cells with same input transition time and output load, but one cell has a bussed net feeding in and the other has a non-bussed net feeding in [15].

Finally, we converged on using a SAIF file (generated from the post-processed VCD file) in PT PX in order to get our power measurements and avoid any remaining false toggles in the VCD file. We lose the capability to observe peak power by not using a VCD file, but for average power measurements, using a SAIF file seems to be safer than using a VCD file.

One other issue involved the behavior of `report_power` omitting cells with negative internal power but positive total power from the default power report [16]. This issue was fixed in the next service pack release of PT PX (C-2009.06-SP3).

In our Synopsys libraries, output pin capacitances are specified for standard cells. PrimeTime includes the output leaf pin capacitance in the output load, when performing table lookups in the libraries to determine cell delay and output transition times. But PT PX omits this output leaf pin capacitance for both switching power and cell internal power calculations. This causes the active power correlation of PT PX to our internal transistor-level power analysis tool to be off by a reasonable amount. We worked around this issue by adding the output pin capacitances to the nets connected to these output pins. This workaround is not perfect since it affects the node slopes, which in turn affects the power calculation. The appropriate fix is to include this output leaf pin capacitance in the calculation of the power tables.

During the course of implementing our power analysis flow, we considered switching to PT PX D-2009.12-SP1 to take advantage of some of the bug fixes and new features. However, an issue with the VCD and SAIF file reader, which reported twice as many cells in the log after reading the activity file, caused us to stick with the C-2009.06-SP1 version [17]. Synopsys has scheduled the fix for this issue for its D-2010.06-SP2 PT PX release.

Key Learnings:

1) Need to use VCS generated VCD files rather than FSDB files as the starting point for PT PX gate-level activity inputs
2) Need to validate that PT PX library gives reasonable results for zero input slopes, as PT PX assumes zero input slopes when it cannot determine the input event causing an output transition
3) Attempt to get the RTL team to adopt a naming convention for the indexes of nested `generate` statements and `for` loops, which indicates the order of the nesting relative to the order of the indexes in the variable names
4) The use of VPI routines allows a forced-gate simulation to run at nearly same performance as a regular full GLS
5) Removing X's from the gate-level power simulation by initialization of the gate-level model eliminates potential RTL-to-gate simulation mismatches due to the propagation of X values

PERFORMING FULL GATE DELAY POWER (GLITCH POWER) MEASUREMENTS

Measuring the active power due to glitches, those transmitted from the input of a cell to its output (Synopsys calls these transport glitches) and those just at the input of a cell (Synopsys calls these inertial glitches), require the use of toggle data generated from an SDF file-backannotated forced-gate VCS simulation. The glitch power cannot be measured directly. In order to determine the actual power contribution of glitches, the active power from a PT PX run utilizing a VCD file from ideal (zero delay) simulation should be subtracted from the active power from a PT PX run utilizing a VCD file from a SDF backannotated forced-gate simulation. The difference between these two runs is the glitch power.

The SDF files used by VCS in forced-gate simulations are generated from PrimeTime runs with post-route data (netlist and SPEF parasitics) and timing constraints.

The Verilog simulation libraries used for the force-gate simulation were modified so that each sequential cell included a fixed internal net name which could be forced to set a value at the

input of the sequential element (we called this net "`force_d`"). The cell was also modified such that an internal net connected to the clock input of the sequential element was also given a fixed name (we called this net "`force_c`"). Internally, the clock input net was inverted appropriately such that `force_c` was always defined to be active high even if the sequential was an active low sequential cell, and was zero delay buffered from the cell clock pin, such that the `force_c` net could be toggled without changing the value of the cell clock pin. Modifying the cell description this way allowed a cell value to be forced in the same manner independent of the actual sequential cell type. Note these internal nets are not included in the Synopsys .lib files used for power analysis, so the activities on these nets are not counted by PT PX during power analysis.

In SDF-backannotated, forced-gate VCS simulations, our method of forcing sequential nodes also uses the same VPI routine as the zero delay simulation. This VPI routine has the ability to assign internal nodes within leaf cells (essentially the Verilog library descriptions) a new value through a “force” (which would retain the forced value until it was removed or another “force” was applied), or through a “deposit” (which would retain the value deposited until a new value, be it through another “deposit”, “force” or through simulation was set). The VPI routine also has options to toggle a net of a sequential cell so that values on the data inputs could be loaded into a cell at any specified time by toggling a clock net. This approach of toggling the clock net was required to initialize the sequential UDP that was used to hold the state of a sequential element.

The initialization strategy for the simulation was to force the appropriate value on the "`force_d`" net of the sequential and to toggle the sequential "`force_c`" net. This loads all the sequential with their initial value. Before every ensuing clock edge, the "`force_d`" forces are updated for any sequential which has changed value, and the clock edge propagates these values through the sequential cell. By virtue of including the SDF file, the clock-to-Q delays for sequential cells can be observed. Another advantage of forcing the sequential cell inputs instead of outputs is that the inputs can be forced before the clock network starts to propagate the new clock edge and well before this clock propagates through the clock network and arrives at the sequential cells.

Since the focus is on obtaining a VCD file for power, toggling of notifier signals for timing violations during the forced-gate simulation is turned off through the `+no_notifier` switch. Otherwise, there would be X values on the outputs of sequential cells that violate timing constraints.

PHYSICAL VS. SIMULATION GLITCH

When one runs a gate-level simulation with backannotated delays, small glitch pulses can be created due to the fact that signals now do not all change at the same time. The VCS simulator is able to propagate glitches throughout the gate level logic without regard to the width of these glitches. However, in a physical inverter this is not the case. In a SPICE simulation, delay is measured from the input at 50% to the output at 50%, and signals have a finite rise/fall transition time. A delay measurement assumes the input continues its rise/fall during the switching event as then stays at the appropriate power/ground rail. If the input node switches again (i.e. glitches) before the output has settled at its final power/ground rail, the inverter output switching will be affected. If the glitch is small enough, the output will never completely transition to the appropriate power/ground rail and the glitch will not propagate. Based on various SPICE simulations, we have come up with the approximation that a glitch will not physically propagate through a standard cell unless that glitch is 1.5 times the value of a simple inverter delay. Hence, to give a good approximation of the real “glitch” generated by a gate-level implementation, we need to remove the small non-physical glitch pulses from the VCS simulation.

Three strategies were explored to filter glitches from VCS simulations:

1. VCS percentage delay-based filtering through pulse controls,

2. Pulse filtering by post-processing the VCD to remove signal pulses of less than the given width, and
3. Pulse filtering using PATHPULSE statements in SDF file

To use VCS percentage delay-based filtering, transport delays along with pulse control switches (+pulse_r and +pulse_e switches) were employed. This filtering strategy was easy to apply as it only required users to specify VCS options to enable it. However, this strategy did not work appropriately as it did not support absolute delay values. In the physical circuit, pulse filtering is a function of the delay of each inversion stage of the circuit, not of the full circuit delay. For multiple stage cells (such as buffers), this VCS option incorrectly sets the filtering delay based on the whole delay of the cell rather than just the inversion delay in the cell. Also, in some cases, we want to filter out glitches that are greater than the delay of the cells (for example 1.5 times the delay of a simple inverter), and this method did not support filtering of delays greater than the cell delay.

The use of the PATHPULSE statements in Verilog and SDF files allow the specification, on a per cell instance basis, of rejection and error limits, which are the same as the pulse control limits used in VCS, but in this case, absolute values of rejection and error limits can also be specified. For this approach, one has to make sure that token PATHPULSE statements are included in the Verilog libraries, and the SDF file must be post-processed to include PATHPULSE statements for each leaf cell. For this method of filtering, we used a rejection limit and an error limit of approximately 1.5 times the standard delay we saw for a single inversion cell.

The use of PATHPULSE statements enabled the use of absolute delay values, but the PATHPULSE statement cannot be used to filter delays greater than the intrinsic cell delay. Hence, the original issue of not being able to filter delays greater than the delay of the standard cell is still present. Our solution to this problem was to write a utility, we call `filter_vcd [20]`, which is used to post-process VCD files. This utility allows one to post-process a VCD file and remove any logic pulse less than the specified filter delay and write out the resulting new VCD file.

Our final solution was an amalgam of the PATHPULSE and `filter_vcd [20]` methods. The VCD file was obtained from the forced-gate SDF simulation with a fixed PATHPULSE value of 1.5 times the delay of a single inversion cell. For cells where this value was greater than the cell delay, the PATHPULSE value defaulted to the delay of the cell. We then post-processed the VCD using our `filter_vcd [20]` utility to remove the glitches that could not be filtered during the simulation using the PATHPULSE property of the cell due to the fact than the glitches were greater than the delay of the associated cell. We saw this problem mainly with simple single inversion cells. However, most of the glitch filtering was performed by the PATHPULSE statement during the VCS simulation. This is desirable because we want to reduce the probability of two glitches being logically combined into one longer single glitch which would not be removed by a VCD post-processing filter step.

Through our glitch filtering experiments, we came to the conclusions that for our design-under-test and power test patterns of interest, the glitch power observed was 5% of the total active power.

One issue we found was a subtle bug in VCS where SDF backannotated delays were not being applied to certain cells due to calling the SDF backannotation Verilog task and the VPI to force node values at time t=0. We worked around this issue by delaying the SDF backannotation to a later point in the simulation (t=5 for example). We filed a STAR against Synopsys for this issue [18].

Another issue we found was an issue with PT PX and bussed ports. In all cases, the VCD file was bit-blasted using Synopsys' `vcdpost` utility before being used in PT PX. This took care of an issue in PT PX that propagated incorrect toggles for bussed nets when different delays are seen on the bussed nets and the pins to which these nets connected. PT PX does not expect delays on pins due to net SDF delays (when processing a VCD file from an SDF simulation). This issue has been reported to Synopsys.

Finally, an issue with PT PX losing toggles for the VCD file input was observed during a power analysis run of activity generated from a unit-delay GLS run (similar to an SDF-backannotated GLS run). This issue has been reproduced in a small testcase, and was recently submitted to Synopsys [19].

Key Learnings:

1) A full SDF simulation in a forced-gate flow is possible and gives good glitch power measurement. The best approach is to force the inputs of sequential cells, so the forced-gate flow mimics a conventional SDF backannotated simulation where clock signals naturally clock the data out of sequential elements (as opposed to just forcing the outputs of the sequential cells).
2) Real glitch power is approximately 5% of total active power for the given cluster and pattern
3) One needs to delay SDF backannotation in the forced-gate simulation so that SDF backannotation and the VPI to force nodes are not called at the same time. This prevents valid non-zero SDF delays from being converted to zero delays
4) It is best to use `vcdpost` to bit-blast busses so that correct toggles for bussed nets are propagated in PT PX. A fix for this issue is targeted for PT PX D-2010.06-SP2.

SUMMARY

In the paper we described a method to analyze power using a force-gate simulation of a gate-level netlist with full SDF back annotation. This method showed that glitch power is under 5% of the total active power.

Multiple STARs were filed against Synopsys for PT PX during the process of getting our forced-gate simulation flow working. When these STARs get resolved, the implementation of a forced-gate flow should be substantially easier. When running PT PX, SAIF files should be used instead of VCD or FSDB files.

The use of VCS VPI interface routines is a very effective way to speed up the forced-gate simulation flow, and any new force-gate flow should take advantage of them. The authors also proposed a partitioning of the forced-gate power estimation flow which would allow the pieces of the estimation flow to be more easily adapted to different technologies and vendors.

ACKNOWLEDGMENTS

The authors would like to acknowledge the work of Nathaniel Hazelton in writing the initial Perl scripts necessary to get the forced gate flow described working. The authors would like to acknowledge the work of the design team in implementing the design which they used to measure power. In addition they would like to thank Richard Shin for generating the RTL VCD files, which were the primary inputs for our power measurement strategy. Finally, the authors would like to thank John Geremia and Chris Spear of Synopsys for providing assistance in addressing questions and issues related to PT PX and VCS.

BIBLIOGRAPHY/REFERENCES

[1] PT PX – http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx

[2] PT PX Cycle-Accurate Peak Power Flow – https://solvnet.synopsys.com/dow_retrieve/D-2009.12/ptpx/ptpx_6.html#index004

[3] STAR 9000309751 (PT PX) – RTL VCD with 2-D array cannot be annotated; fixed in PT PX C-2009.06.

[4] STAR 9000264846 (Power Compiler) – Bogus output of `saif_map -report` on missing nodes; fixed in Power Compiler C-2009.06.

[5] STAR 9000319040 (Power Compiler) – `saif_map` creates unreadable output file; fixed in Power Compiler C-2009.06-SP1.

[6] ESTAR 9000316869 (PT PX and Power Compiler) – add `-invert` flag to `set_rtl_to_gate_name` command; fixed in PT PX C-2009.06 and Power Compiler D-2010.03.

[7] ESTAR 9000257558 (PT PX and Power Compiler) – `saif_map` to generate `-invert` flag when writing out SAIF map file; fixed in PT PX C-2009.06 and Power Compiler D-2010.03 (Related to [6]).

[8] ESTAR 9000302184 (PT PX) – Enhance switching annotation reporting; fixed in PT PX 2010.06.

[9] Galbi, Duane, et al, "RTL X's – A Treasure Trove of Trouble", SNUG Boston 2002.

[10] Case ID Unknown (VCS) – Enhance `+vcs+initreg` option to initialize UDPs; fixed expected in September patch release of VCS 2009.06 / 2009.12 / 2010.06.

[11] STAR 9000352325 (PT PX) – 0ns input transition time used for power for unmatched input events; Open issue.

[12] ESTAR 9000354389 (PT PX) – Identify related input transitions in zero-delay VCD which lists outputs first; Open issue (related to [11]).

[13] STAR 9000361500 (PT PX) – Non-existent glitching created from VCD parser and processing of bussed port; fixed in PT PX D-2010.06.

[14] STAR 9000356190 (PT PX) – Toggle counts incorrect if VCD file includes inputs to black box macro; fixed in PT PX D-2010.06 (related to [13]).

[15] STAR 9000350866 (PT PX) – Different internal power for identical cells (due to bussed port or driving cell); Open issue (related to [13]).

[16] STAR 9000338970 (PT PX) – `report_power` incorrectly omits cells with negative cell internal power, but positive total power from default power reports; fixed in PT PX C-2009.06-SP3.

[17] STAR 9000406010 (PT PX) – VCD and SAIF file reader in PT PX D-2009.12-SP1 reports twice as many cells in the design; fix expected in PT PX D-2010.06-SP2.

[18] Case ID 8000357887 (VCS) – VPI force erases SDF delays; fixed in VCS C-2009.06-6 and D-2010.06.

[19] STAR 9000401865 (PT PX) – Losing toggles when reading in unit delay VCD file; Open issue.

[20] filter_vcd (filter_vcd.cpp) code has been released and is available on www.veripool.org

[21] force_reg.c (VCS PLI code) has been released and is available on www.veripool.org