PowerTeam:™ There is more to Verilog beyond Behavioral Simulation

Mehmet A Cirit
Library Technologies, Inc.
mac@libtech.com

February 27, 2004

Abstract

Verilog has been a work-horse of digital design for many years. As the design process has evolved, however, it has been relegated to functional and behavioral simulation. Its event driven timing engine, makes it an excellent design and analysis tool as well, offering flexibility, interoperability and speed which can not be matched by any transistor level simulator. We describe in this paper, dynamic power simulation extensions to Verilog which naturally fit into existing design environments.

1 Introduction

Once upon a time, gate level timing simulation was the main digital design tool. Verilog had a venerable place among designers suit of tools. With the advent of RTL based synthesis, gate level timing simulation has lost its prominence. Instead, RTL based functional and behavioral simulation and verification became the main focus of the design activity. Timing verification and timing closure has been relegated to synthesis and timing analysis tools. In fact, J. Cooley[1] did a survey recently to find out if anybody did gate level timing simulation. A few souls turned out to be still clinging to the old reliable for the final verification. As the functional and timing verification aspects of chip design parted ways, a plethora of tools and companies emerged offering various dedicated analysis tools to address "deep" sub-micron analysis needs, which are essentially transistor level tools aimed at providing analysis of various timing effects[2].

In this paper, we shall show that event driven simulation is an excellent concept for analyzing power dissipation of digital designs. When an input changes state, it may trigger a change at the outputs of a gate. In addition, it starts a power dissipation cycle. A changing output may trigger changes at the inputs of other gates. As the signal propagates to the outputs, the cell stabilizes, and power dissipation comes to a stop. It is possible to track all of these events in an event driven simulation environment which can trigger various analysis steps in response to changing input conditions and the expected behavior of the outputs.

Using HDL's like Verilog for simulation is not unusual. However, typically the use of HDL's has been restricted to the extraction of switching activity of the cells. This author had introduced power estimation as a viable method quite a long time ago.[3] A survey of various power modeling techniques can be found in [4]. Power modeling is usually limited to estimation because of the lack of models and reliable data to support such analysis.[5] Although switching activity may be important for estimations, one can generate the actual power usage information without a need for estimation using our techniques. There are also vector based commercial products doing power analysis.[6] This attempts to analyze waveforms generated by a timing simulator and associate them with power events. Such tools though will have difficulty when there are no output changes triggered by an input change, or when there are multiple output changes triggered by the same input event.

PowerTeam provides integration of functional, timing and power simulation in Verilog. It consists of two parts, (i) a Verilog library of basic building blocks including core cells, IO's and memories, (ii) extensions to Verilog simulator to enable it to perform dynamic power simulation using PLI, programming language interface to Verilog engine. Verilog library is generated by SolutionWare™ and consists of functional, timing and power characteristics of each component. PowerTeam™ provides the power modeling capability in Verilog. It keeps a tally of various power events to track dynamic power dissipation and generates various power and energy consumption reports.

2 Power Models

Modeling timing and functional aspects of library elements has been a standard feature of Verilog library models. Functional aspects are modeled using various gate level primitives. Timing aspects are modeled by a series of listings of signal paths between the inputs and outputs. Based on what inputs change and triggered changes on the outputs, the simulator automaticly schedules a delayed assignment to the affected output. There is no such build-in mechanism to track power events, so we rolled out own power modeling using behavioral capabilities of Verilog. In order to do generate a power model, one needs to know how outputs or internal states of a cell changes as inputs change. The energy associated with such events can be measured using Spice and parameterized as function of input rate of change and output loads. Along with the energy dissipated per input event, one can measure the duration of the

```
module inv (Z, A);
 output Z; input  A;
 wire zd_Z;
 not pow_i0(zd_Z,A);
 real tickDelay;
 always @(negedge A) begin
  #(tickDelay);
  if(Z^zd_Z)
   $proppower({Z,zd_Z}===2'b01, 0);
 end
 always @(posedge A) begin
  #(tickDelay);
  if(Z^zd_Z)
   $proppower({Z,zd_Z}===2'b10, 1);
 end
 specify
 specparam
  phl0$A$Z = 'inv_a_lh_z_hl_power,
  plh0$A$Z = 'inv_a_hl_z_lh_power;
  ...
  // Pin-to-pin timing.
 (A => Z) = (0,0);
 endspecify
 // Gate-level description.
 not _i0 (Z,A);
endmodule
```

Figure 1: Fragments of a power model in Verilog for an inverter.

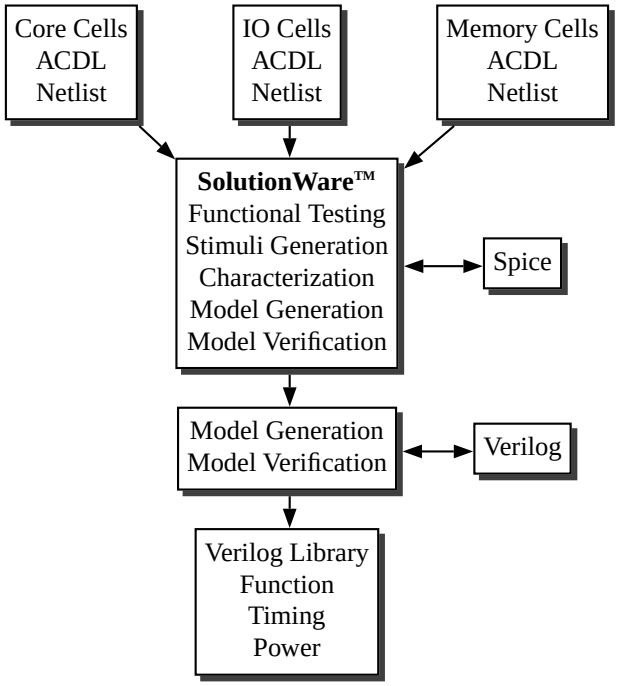

Figure 2: Characterization and Model generation flow using SolutionWare™.

event, which is the time the cell completes its change of state and stops energy consumption. Using these two, the average power use can be calculated as the cell switches states.

Figure 1 shows the parts of a Verilog model for an inverter cell which models its own power dissipation. Power modeling involves the selection of a power event when an input changes, very similar to the selection of a signal path in the case of timing modeling. The signal `zd_Z` is logically equivalent to the signal `Z`, but unlike `Z`, it is evaluated immediately as soon as the input changes. `Z` output, on the other hand, is delayed. Comparing the current state of the output, `Z` to its future value, one can conclude that there will be a signal path through the cell, and it power dissipation can be accounted for by calling `$proppower` which propagates the power event upwards through the design hierarchy. Various parameters needed for delay and power dissipation are defined using Verilog `specparam` definitions. Path polarity, output rise/fall, input/output delay and input capacitance parameters in table format are referenced in the model. In the same way, for each signal path, there are associated power dissipation parameters. `$proppower` references these parameters depending on the input and output state. The condition `Z^zd_Z` is used to differentiate between input events which cause outputs to change and events which causes only internal state changes. In the first case, one needs to account for capacitive power dissipation.

At each input event, energy dissipated by the cell is calculated and various total energy counters are updated. Duration of the power dissipation event is calculated, dynamic energy dissipation counter is increased and scheduled for decreasing the total energy dissipation after the event concludes. Since each power event is represented by a rectangular pulse, such a model is only an approximation of the actual pulse. However, as thousands of gates switch within a short span of time, the total power dissipation could be more accurate than the accuracy of the individual events.

3 Model Generation

Figure 2 shows the characterization and model generation steps involved in Verilog library generation. The flow covers the three main components of digital designs: core, IO amd memory. Actually, the system is flexible enough even to beyond these and model ADC and DAC's using analog extensions to Verilog. The primary inputs for all three types cells are functional descriptions of each component type using ACDL, and spice netlists. ACDL, Asic Cell Description Language, is a simple proprietary format good enough to describe and model basic digital building blocks. For an inverter it looks like

```
Z = !A,
```

for a flip-flop

```
Q = !RN ? 0 : rising(CLK) ? D : "p"
```

and for memories it consists of pin descriptions:

```
memory.state = iq
type.iq = 2
clock.iq = CLK
chipSelect.iq = CEN
writeEnable.iq = WEN
adr.iq = A[9:0]
```

```
dataW.iq = D[7:0]
dataR.iq  = Q[7:0]
activeLow.iq=CEN,WEN
```

which varies with different types of memories. The modeling of IO's are very similar to core cells save for the complexities of having multiple supplies. These inputs are used to generate test vectors to verify the functionality of the cells. Verification is performed using Spice. Cells are further analyzed to find out variations of timing, power and setup/hold behavior under different input and internal state conditions. Spice is used for this step as well. Finally stimuli are generated for measuring timing and power parameters. Each of these parameters are measured by varying input rise/fall times and output loads. Verilog models are generated and verified using the same test vectors used for verifying the spice netlist. For core cells, run times may range from a few minutes to hours for each cell on a single machine. For memories it may be a up to a few days on multiple machines depending on its size. For memories bigger than 1024x9, one may need to use a spice-like simulator for characterization.

Adding power simulation capability to Verilog has some run time penalties as should be expected. In addition to the obvious cost of making an expensive PLI call for every input change, there are additional costs in terms for various conditionals to enable various power dissipation modes. Static power, for example, is highly state dependent. As inputs change, power simulator must decide which static power parameter needs to be assigned to the cell which changes its state. In the same way, there could be multiple propagation and input toggle power conditionals which the simulator need to evaluate. This also entail more memory utilization compared to pure timing simulation. Typically we see approximately a factor of 10 slow down in run time compared to pure timing simulation.

4 Power Reports

One benefit of integrating power simulation inside Verilog is that one can rely on existing infra-structure build already for functional and timing verification. New test vectors, or new file formats or new user interfaces or debugging environment are not needed. All of these which make a new tool usable, are already built into existing Verilog environment.

Using a power capable library, power simulation is very easy. The following section demonstrates how to monitor power usage of a design:

```
chip chipi(...);
initial $delay_calc1("chip.load", chipi);
```

where `"chip.load"` contains interconnect load for `chipi`. `$delay_calc1` performs delay and power calculations using the load information coming from layout extraction. Timing and power characteristics of the cells as defined in the Verilog library in table look-up format. Integration of delay and power calculation inside the simulator has the obvious benefit of consistency of the timing models and delay calculation process, unlike more traditional SDF back-annotation where carrying around the delay information from tool to tool is mired with compatibility problems. In this case though, although there is no library compatibility issue, one needs to make sure that net names referenced by `"chip.load"` are actually found in the design.

The delay and power calculation is done once at the beginning of simulation, and the information is stored away for use when inputs toggle. Furthermore, dissipated power can be classified into static, internal and external components to help guide the design process. As much as the dynamic power changes as the inputs to the cells change, steady state power dissipation is also state-dependent. Being able to view internal and external load dependent parts of power dissipation may help in coming up with better power-aware layouts. A big load on a less frequently toggling net, may be less important than a smaller load on a frequently toggling net.

In addition to dynamic power, one can generate total energy used by various blocks of the design. For each block, one can divide up power dissipation further into sub-blocks. All this information could be quite useful for improving power dissipation. Figure 3 shows how one can make architectural changes in RTL based on module/instance based power dissipation reports. By going through the synthesis process, eliminates all the uncertainty about the power behavior of RTL code. Layout wire load can be estimated at this stage.

Energy use could be further broken into capacitive and internal components. This information could be used to direct P&R tools to do power oriented layout. There are various power monitor which one can use to track power dissipation of various modules:

```
real totalPower;
real totalIntEnergy;
real totalCapEnergy;
```

which get updated as cells start and stop switching. `totalPower` is the instantaneous rate of energy dissipation. `totalIntEnergy` is the internal energy dissipation of the cells, without the external capacitive component, and `totalCapEnergy` is load dependent energy dissipation. One can generate further reports using these variables, like energy consumed at each clock period, as the following example shows:

```
real pEnergy;
initial pEnergy = 0;
always @(posedge CLK)
begin
 $display("%e", totalIntEnergy
  totalCapEnergy-pEnergy);
 pEnergy=totalIntEnergy
  +totalCapEnergy;
end
```

which provides a very flexible and convenient way to monitor energy used during each clock cycle initiated by the rising edge of `CLK` signal. Such reports can be structured in such a way to easily extract them from the simulator output and pipe them to various plot generators.

One can also generate energy used by different components of the cell library. This information could be used to selectively

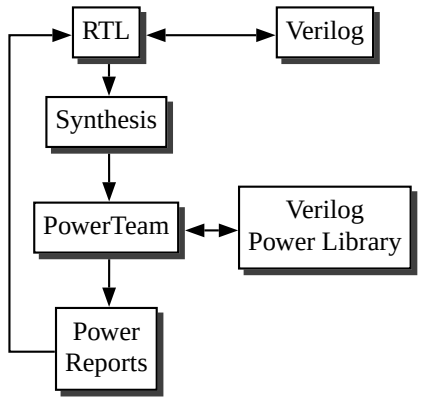

Figure 3: Use of PowerTeam in RTL design flow. Instance/module based power usage reports identify the main power consumer blocks.

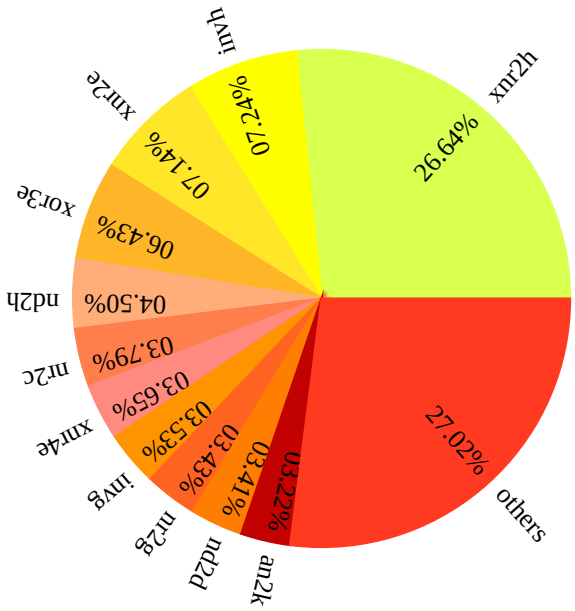

Figure 4: A pie chart of total energy dissipated by each cell type. Such fancy presentation graphics can be generated easily by simple reformatting of the PowerTeam reports and LaTeX.[7] The report was generated for a 16-bit ALU from ISCAS test suite.

optimize the design of various library elements. Figure 4 illustrates such a use of power reports generated by PowerTeam. Here the energy dissipated by each of type of cell used in a 16-bit ALU is charted after a long simulation. Clearly, in this example, certain group of cells stand out as consuming most of the energy. Proper power optimization of such cells can significantly reduce power dissipation.

5 Conclusions

One of the design criteria is to find the worst case power peaks. This information is needed for power bus sizing. PowerTeam generates this information by providing access to instantaneous power dissipation for the whole design as well as for sub-blocks.

Using PowerTeam, energy consumption profile of a design can be generated, the most power consuming sub-blocks of the design can be identified. This information can be used to re-architect the design for lower energy use. Such information could be useful even without having the full layout parasitic information. RTL synthesis followed by power analysis with estimated wireloads should be sufficiently accurate to guide architectural explorations.

Since PowerTeam works at the library level, it can also point out the impact of various library elements on the total energy use of the design. One can generate reports detailing the energy dissipation of various components, and their impact on the design. After all, energy is consumed by the library elements. Making sure that they use as little energy as possible should be the first priority of any cell design activity. Unfortunately, this has been an elusive goal in cell design, where low power come to be understood as synonymous to low speed. The ease of implementation which is usually manual activities, and time to market are the main goals of the cell library design rather than the energy efficiency of a cell implementation. There are, however, better ways of designing energy efficient libraries[8] using circuit optimization techniques.

I would like to acknowledge the support of A. Dutta, K.K Kothar, H.K. Ramasubramanya and V. Tatke of Cisco Systems, and thank them for their patience and encouragement while PowerTeam was being assembled.

References

[1] J. Cooley, “A chip designer asks what gate-level sims buy you these days,” Tech. Rep. 420, ESNUG, Oct. 2003.

[2] C.Huang and B. Zhang, “The design and implementation of powermill,” *Proc. of IEEE Symp. on Low Power Electronics*, pp. 105–110, 1995.

[3] M. A. Cirit, “Estimating dynamic power consumption of cmos circuits,” *Proc. ICCAD*, pp. 534–537, Nov. 1987.

[4] F. Najm, “A survey of power estimation techniques in vlsi circuits,” *IEEE Trans. on VLSI Systems*, vol. 2, no. 4, pp. 446–455, 1994.

[5] A. Bogliolo, L.Benini, and G. D. Micheli, “Adaptive least mean square behavioral power modeling,” *Proc ED&TC*, pp. 404–410, 1997.

[6] H. Sanghavi, “Power: A costly and critical parameter,” *Chip Design*, pp. 12–13, Oct. 2003.

[7] M. Goossens, S. Rahtz, and F. Mittelbach, *The LaTeX Graphics Companion*. Addison Wesley, 1997.

[8] M. A. Cirit, “Positional sizing: A recipe for high speed and low power cell libraries,” tech. rep., Library Technologies, Inc., Mar. 2002.