

RTL Power Estimation in an Industrial Design Flow

C. Guardiani[†] A. Macii[◊] E. Macii[◊] M. Poncino[◊]
M. Rossello[†] R. Scarsi[◊] C. Silvano[#] R. Zafalon[†]

[†] SGS-Thomson Microelectronics [◊] Politecnico di Torino [#] Università di Brescia
Agrate Brianza, ITALY Torino, ITALY Brescia, ITALY

Abstract

We analyze the integration of an RT-level power estimation tool, RTPOW, into an industrial design flow. We address important practical issues such as the extraction of structural and functional information and the use of simulation data to derive probabilistic information. We present the results obtained with RTPOW on a realistic design that demonstrate the effectiveness of our tool.

1 Introduction

With the advent of portable microelectronic devices and the consequent increase of frequency and transistor density of moderns ICs, the power dissipation of VLSI circuits has emerged as a critical concern.

Low-power design needs efficient and accurate estimation tools at all design abstraction levels. In particular, RT-level power estimation is critical in obtaining short design times and it is very important to help the designer in making the right architectural choices. Accurate RT-level power estimation allows to reduce the number of design iterations and their relative cost.

The approaches proposed in the literature (see [1] for a comprehensive survey) can be categorized into two main classes: Top-down and bottom-up methods. Methods of the former class are particularly suited for components with a fixed structure and/or design (e.g., memories, data-path units).

Bottom-up methods are based on the idea of building an abstract power model by refining an initial model template through experimental power measurements.

In this work, we analyze the integration of a power estimation tool, RTPOW, into an industrial design flow.

The environment poses additional constraints and limits the tool in two ways; first, the hardware description style and the abstraction level. In fact, different styles imply different availability of structural and functional information. Second, since the availability of probabilistic information is fundamental in power estimation, the simulation process may add constraints on what information can be obtained to drive the estimation procedure.

2 General Functionality and Environment

RTPOW is an RT-level power estimation tool that works within the Synopsys' DesignCompiler environment, as shown in Figure 1. RTPOW is implemented as a set of scripts, driven by certain user-specified variables, that outputs information about the current design structure on a set of files parsed and elaborated by an underlying C++ program.

The estimator may work in two ways. The first mode of operation is simulation mode. At the end of the simulation, a file which contains switching activity and static probability information about synthesis invariant nodes (i.e., I/O ports, sub-module boundaries, and sequential cell outputs) is written. This file has a `.saif` extension (Switching Activity Interchange Format).

The second mode of operation is probabilistic mode. In this case, it may be useful to enable the annotation of switching information on selected design nodes; this is important for sequential cell outputs, if good accuracy is required, since the switching activity propagation engine tends to be inaccurate when dealing with sequential cells. Annotation may be either performed by reading a `.saif` file (as the one produced in simulation mode) or by `set_switching_activity` commands.

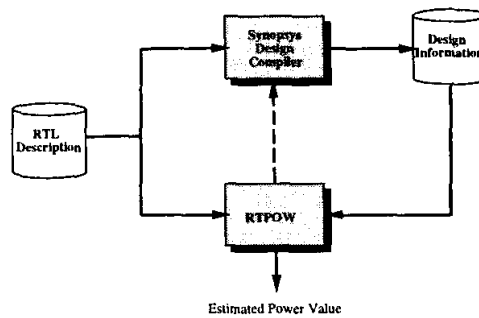


Figure 1. Conceptual Flow of RTPOW.

3 Structural Information

An important problem for an RTL power estimator is how to infer circuit functional information from a given description. Some academic works [2, 3, 4] have solved this problem by using an initial description from which circuit topology is directly imported in the estimator by extracting some functional representations (typically by means of BDDs).

Given the context of RTPOW, we need a method to get circuit topology directly from the tool used to analyze the design. After source code analysis and elaboration, a RT-level description is expressed within DesignCompiler as an interconnection of different types of primitives (e.g., Gtech ports, Generic logic blocks, Synthetic operators, DesignWare modules, Generic sequential operators). The difficulty here is to make this type of description available to the underlying C++ program. This is carried out in RTPOW with proper commands that dump the circuit as a set of equations. Then, connections between the previous components and sequential cells are recognized by parsing the file produced by the `report_cell` command and by patching their functionality into the previously created representation.

4 Implementation Details

The underlying estimation engine has been implemented in C++. A top-level class `Module` gathers all common data structures and offers several virtual methods which are implemented differently in each sub-class. Furthermore, there is a class for each kind of module. Currently, the following types are supported:

- Generic logic, including selectors and 2-to-1 Gtech multiplexors: Gtech library components can be exported from DesignCompiler directly in equation format;
- Sequential generic cells (*SEQGENs*) used by DesignCompiler to describe various kinds of sequential functionalities;
- DesignWare's RPL adder;
- DesignWare's CSA multiplier;
- Gtech multiplexor.

5 Area Models

Area and power models are different for each kind of module. All the models described in the following try to match mappings to the CORELIB HCMOS6 technology library from SGS-Thomson:

- Generic Modules: The BDD representation of a boolean function is equivalent to a 2-to-1 multiplexor mapping where controlling signals come from the primary inputs of the circuit. Since this represents a sort of library technology mapping, the area estimation engine fits the number of BDD nodes, as a measure for the area occupation, to the actual area measured on a number of benchmarks mapped onto the CORELIB library; The number of BDD nodes versus the actual area, shown in Figure 2, indicates a clear linear dependency in logarithmic scale. Least-mean squares regression can be used to extract an estimator in logarithmic scale, starting from the equation: $Y = AX + B$ where A and B are the fitting parameters, X is the

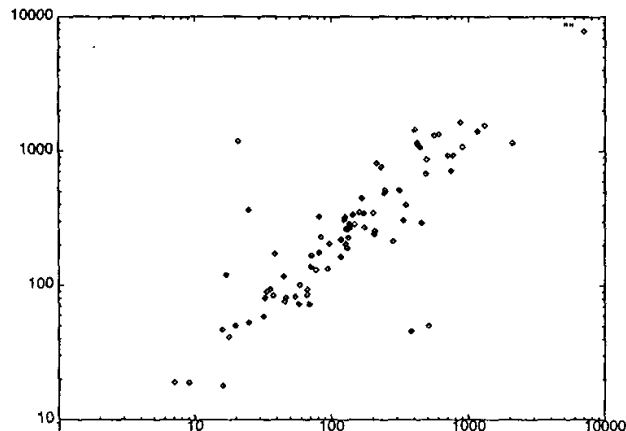


Figure 2. Estimated Area (X-axis) vs. Actual Area (Y-axis).

BDD node count, and Y is the actual area estimated by Design Compiler.

- Sequential Cells: A fixed area value which represents the area occupied on average by a sequential cell is used. This value is chosen as $250\mu m^2$.
- DesignWare Modules and Gtech MUXes: All the modules have been mapped with several widths, and the area values so obtained have been eventually fitted to get a model. Perfect area mapping is obtained for adders, since the number and type of gates on the synthesized netlist is totally predictable; average and maximum errors of 3.9% and 15.2% have been reported for 4-bit multipliers.

6 Power Models

Area estimation is used in the power model as a capacitance approximation of modules capacitance. Therefore, to get the power values both area and average switching activity estimates are needed. Switching activity estimation is done differently in simulation and probabilistic mode. While simulation mode traces the actual behavior of the circuit when stimulated by a set of patterns and simply counts the number of toggles, probabilistic mode uses the following statistical relation, valid for temporally-independent inputs, to get the toggle rate of a function f from that of its support:

$$T_i = \sum_x P\left(\frac{\partial f}{\partial x}\right) \cdot T_x \quad (1)$$

where T_i is the toggle rate of node i , x are the inputs, $\frac{\partial f}{\partial x}$ is the boolean difference of f over x , and $P(y)$ denotes the static probability of y .

6.1 Generic Logic

Both the average switching activity and the total area are estimated by processing the BDD representation as an actual 2-to-1 MUX mapping. In simulation mode all the toggle events for each equivalent multiplexor are summed up and finally divided by the total mux count. Each equivalent multiplexor output toggle is counted proportionally to the fanout of the multiplexor, so as to take into account a sort of load factor. Toggle count, when multiplied by the capacitance (area) estimation, becomes a correct estimate of the effective switched capacitance.

Similarly, in probabilistic mode the above relation is used mux by mux. If O is the output, C is the controlling input, and A and B are propagated to the output if C is high or low, respectively, the toggle rate and static probability relations for a 2-to-1 MUX are the following:

$$\begin{aligned} P_O &= P_A \cdot P_C + P_B \cdot (1 - P_C) \\ T_O &= P_C \cdot T_A + (1 - P_C) \cdot T_B + P_A \cdot (1 - P_B) \cdot T_C + (1 - P_A) \cdot P_B \cdot T_C \end{aligned}$$

Again, toggle activity is summed up (accounting for the fanout load as before) and divided by the number of nodes.

6.2 Sequential Cells

In simulation mode, sequential cell behavior is simply simulated through `if-else` C++ statements, and average output toggle count is taken.

In probabilistic mode, the switching activity evaluation is more complicated. Propagation of statistical information through sequential cells is difficult and tends to be inaccurate. For this reason, switching activity annotation on sequential cells outputs is very recommended. The complete estimation algorithm is shown in Section 7.

6.3 DesignWare Modules and Gtech MUXes

DesignWare and multiplexor's behavior in simulation mode is emulated through C++ statements, so it is possible to trace back the input and output toggle rates. When running in probabilistic mode, Equation 1 can be used to compute the toggle rate of each output. In both cases, the problem is to find the average (over all equivalent gates in the module) toggle rate from the knowledge of only the input and output toggle rates. Since the module topology is clear, it is possible to multiply every I/O toggle activity for the appropriate capacitance as in the case of the ripple carry adder.

However, the topology may be quite complex, and a simpler method should be used. For this reason, the toggle rates at the inputs and outputs are individually summed up, to get a sort of input/output entropy. The formula of Equation 2, proposed in [2], can be used to get the average module entropy, and consequently the average toggle activity since they are closely related:

$$\mathcal{H} = \frac{2H_I + 2H_O}{3(N_I + N_O)} \quad (2)$$

where H_I and H_O are input and output entropies, respectively, and N_I and N_O , are the number of circuit inputs and outputs.

The above formula relies on some simplifying topological assumptions that require specific correction factors. These are empirically determined, as:

RPL adder $\mathcal{H}_{corr} = \mathcal{H} \cdot 3.31 - 90$

CSA multiplier $\mathcal{H}_{corr} = \mathcal{H}^{1.35} / 4$

These models have been obtained through the mapping mentioned in Section 4, with equiprobable, independent inputs, at 20MHz. An average and maximum error of 0.5%(9.1%) and 1.2%(18.8%), respectively, have been experimented on adders (multipliers). Refined models for Gtech MUXes have not been developed yet, and the original value of \mathcal{H} from Equation 2 is used in these cases.

7 Algorithm

7.1 Simulation Mode

The simulation engine tracks new node values from outputs back to the inputs. All new sequential cell input values are computed later, so that they can be used in the next cycle. If forward propagation would have been done, sequential cells would then have been updated before all other signals; this could be traced correctly but larger memory occupation and more complex management would be required.

```

while (there are patterns) {
    assign values to top level inputs
    for (each top level output i)    get output value(i);
    for (each sequential cell in hierarchy) get new output value;
    for each module in hierarchy {
        update toggle counts;
        update static probability values;
    }
    setup for next pattern;
}

```

Output values for the various modules, except sequential cells, are simply computed through their combinational functions.

A virtual function called `trackCombSignal()` serves the purpose of traversing, in a depth-first fashion from each module, until a top-module input or a sequential cell is reached. The implementation of `trackCombSignal()` is different for each module type. Concerning sequential cells, `trackCombSignal()` must be explicitly written in order to take into account the already mentioned input priorities.

7.2 Probabilistic Mode

The main information obtainable in probabilistic mode is not very different from simulation mode, that is, net activities are traced back from outputs to inputs. The main difference is that instead of actual simulated values, static probabilities and toggle rates are propagated in one step.

In a circuit there are typically several cycles that make data inputs dependent on their own outputs. Moreover, since there is no time information, so there is no way to distinguish old and new input values. For example, consider the example circuit of Figure 3.

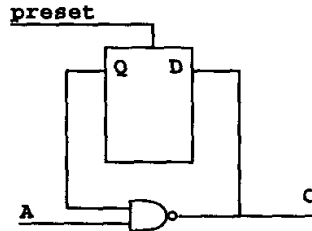


Figure 3. Example Circuit.

This simple circuit keeps C high when $preset$ is high, until A goes high. Since there is no time information, there is no way to know if $preset$ and A are synchronized so that A is up together with $preset$, and for how much time it has to stay high.

For this reason, the most viable way to find the statistical properties of a sequential output is that of finding a sort of *eigenvalue*, that is, a value that, used to compute itself, gives the same value. We have experimentally observed that, in general, it is not possible to find such toggle rate value. Conversely, a static probability value seems to be computable in quite a limited time, though we cannot be sure that this represents the only possible value. The sequential cell output toggle rates are then computed with the following formula:

$$T_O = 2 \cdot P_O \cdot (1 - P_O) \cdot \frac{T_{clock}}{4}$$

where T_{clock} is the clock toggle rate and P_O is the output static probability. The equation expresses the fact that a white noise signal can statistically toggle a sequential cell once every two clock cycles, i.e., every four clock transitions. The static probability of the output of sequential cells is computed iteratively. The whole process is repeated until the new value computed and the previous value differ for less than a defined percentage.

8 Experimental Results

To illustrate the proposed estimation tool in an existing, industrial flow, we evaluated different design realizations of a proprietary micro-controller.

The plot of Figure 4 shows the results concerning the model itself. The x-axis gives power values estimated with the model, whereas the y-axis reports the actual, simulated, power. The points in the plot represent measurements on several benchmark circuits. We can again observe the good approximation of the linear regression model, in log scale.

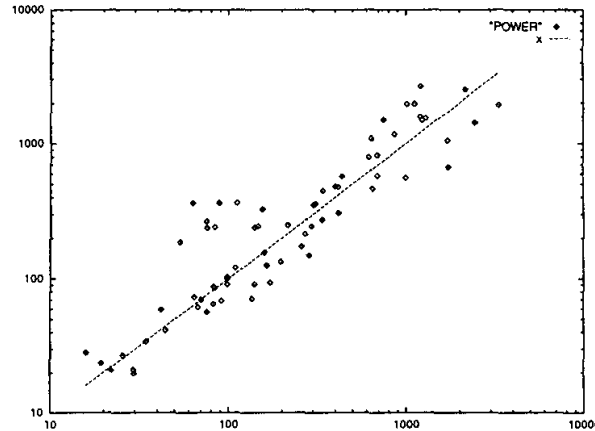


Figure 4. Estimated Power (X-axis) vs. Actual Power (Y-axis).

Two architectures have been derived from the reference one by assigning different values to some of the configuration parameters. The first solution, denoted as *micro-a*, has been obtained by increasing from 16 to 32 the bit width of the status register. The second architecture, called *micro-b*, has been obtained by decreasing from 8 to 4 the number of interrupt request lines that are managed by the interrupt controller. Clearly, while the first architecture is expected to be more power consuming than the reference one, the second alternative should be more power efficient.

The relative power results with respect to the reference solution are reported on the left-hand side of Table 1 (column *RTPOW*). The data confirm the expected trend of relative power.

In order to validate the power figures determined at the high level, each architecture has been synthesized onto the CORELIB HCMOS6 library, and the actual power dissipated has been estimated using DesignPower. The results obtained, expressed in relative terms with respect to the reference description, are reported in Table 1 (column *DesignPower*). The data in the table show that the trend of relative power has been conservatively preserved.

<i>Architecture</i>	<i>Relative Power [%]</i>	
	<i>RTPOW</i>	<i>DesignPower</i>
<i>micro-ref</i>	100.0	100.0
<i>micro-a</i>	142.0	150.8
<i>micro-b</i>	78.1	80.5

Table 1. Relative Power Results.

9 Conclusions

We have presented an RT-level power estimator, *RTPOW*, which is suitable for integration into an existing industrial design flow. Such integration needs to address important practical issues that are often of minor concern when designing a new power estimation method, but need to be taken into account when the theoretical framework has to meet the constraints of both the existing design flow and the underlying tools.

References

- [1] P. Landman, "High-Level Power Estimation," *ISLPED-96*, pp. 29-35, Monterey, CA, Aug. 1996.
- [2] M. Nemani, F. Najm, "Towards a High-Level Power Estimation Capability," *IEEE Transactions on CAD*, Vol. CAD-15, No. 6, pp. 588-598, Jun. 1996.
- [3] D. Marculescu, R. Marculescu, M. Pedram, "Information Theoretic Measures For Power Analysis," *IEEE Transactions on CAD*, Vol. CAD-15, No. 6, pp. 599-609, Jun. 1996.
- [4] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, D. Sciuto, "Power Estimation of Behavioral VHDL Descriptions", *DATE'98* pp. 762-766, Paris, France, Mar. 1998.