Fast High-level Power Estimation for Control-flow Intensive Designs

Kamal S. Khouri, Ganesh Lakshminarayana, and Niraj K. Jha

Department of Electrical Engineering
Princeton University, Princeton, NJ 08544

Abstract

In this paper, we present a power estimation technique for control-flow intensive designs that is tailored towards driving iterative high-level synthesis systems, where hundreds of architectural trade-offs are explored and compared. Our method is fast and relatively accurate. The algorithm utilizes the behavioral information to extract branch probabilities, and uses these in conjunction with switching activity and circuit capacitance information, to estimate the power consumption of a given architecture.

We test our algorithm using a series of experiments, each geared towards measuring a different indicator. The first set of experiments measures the algorithm's accuracy when compared to the actual circuit power. The second set of experiments measures the average tracking index, and tracking index fidelity for a series of architectures. This index measures how well the algorithm makes decisions when comparing the relative power consumption of two architectures contending as low-power candidates.

Results indicate that our algorithm achieved an average estimation error of 11.8% and an average tracking index of 0.95 over all examples.

1 Introduction

Evaluating metrics such as area, delay, and power consumption is an integral part of the design process. Estimation techniques at different levels of the design hierarchy provide the means for this evaluation. The importance of power consumption and its impact on both portable and desktop electronics has also been recognized by researchers, who have investigated power estimation techniques from the circuit to the system level.

In high-level synthesis, behavioral descriptions may be classified into three categories: data-dominated, control-flow intensive, and control-dominated. Control-flow intensive designs are characterized by a mix of data-flow and control-flow operations such as nested loops and conditionals. This paper tackles the problem of estimating the power consumption of control-flow intensive designs. Fast and accurate high-level power estimation tools are required to drive many of today's high-level synthesis tools. Traditional methods fail when large numbers of architectures (up to several hundreds for a single circuit) need to be evaluated during the course of synthesis - *i.e.*, they consume too much CPU time to be practical during synthesis. Unlike these methods, our technique provides a fast and relatively accurate means for power estimation, allowing iterative high-level synthesis tools to explore and evaluate hundreds of architectural trade-offs easily.

Typically, lower-level power estimation techniques, such as SPICE [1], provide the highest estimation accuracy. Extensive work has also been done at the logic level. A detailed survey of these techniques is given in [2]-[4]. A more recent gate-level technique [5] uses power macro-modeling based on a three-variable model. The shortcoming of low-level methods is that they tend to be computationally intensive and unsuitable for driving a high-level synthesis system.

Many contributions in power estimation have been made at the architecture or register-transfer level (RTL). In [6], a method known as power factor approximation (PFA) characterizes the power consumption of macro-blocks, such as adders and multipliers, by performing simulations on low-level implementations of these blocks. However, the PFA method does not account for inter-operation dependencies. This problem is accounted for in [7], where a dual bit-type model for word-level signals is presented. The dual bit-type method is most effective for data-dominated designs. In [8], a switching activity and power estimation method is presented for control-flow intensive designs, where glitches in datapath and controller signals are accounted for. This method is accurate, yet too computationally intensive to be in the inner loop of a high-level synthesis system. The above techniques are characterized by the fact that they model circuit blocks as "black boxes." In other words, low-level details of the circuit blocks are simplified into a high-level power model. Techniques which do not use the "black box" concept are based on information-theoretic models [9, 10]. In [9, 10], values such as entropy and informational energy are used to measure the switching activity. Power estimation at the behavior level is given in [11], where linear regression is used to model and characterize library macros used for implementing behavioral operators.

This paper presents a high-level power estimation method that uses behavioral information to extract accurate branch probability values which are then used to estimate the switched capacitance, based on switching activity matrices and technology dependent capacitance values, at the architecture level. The use of the behavioral information dramatically speeds up the process of branch probability extraction, eliminating the need to perform RTL simulations. This method is ideal for driving an iterative high-level synthesis system whereby comparison of the relative power consumption of hundreds of architectures may be performed quickly.

In the next section, we motivate our method with an example. In Section 3, we introduce some concepts and state the assumptions that are used in the estimation technique. We introduce the algorithm in Section 4, then present the experimental procedures and results in Sections 5 and 6, respectively. We conclude in Section 7.

2 A Motivational Example

We motivate our work with an example illustrated in Figure 1. The figure represents a control-data flow graph (CDFG)[1] of a typical, yet simplified, control-flow intensive circuit. The circuit performs a calculation within a loop using an input seed value. We compare two estimation engines driving the high-level synthesis

[1]We describe this CDFG model in more detail in Section 3.1

system, IMPACT [12], for control-flow intensive designs. The first engine is the one presented in [8] (henceforth referred to as scheme-1), while the second is our proposed technique (referred to as scheme-2).

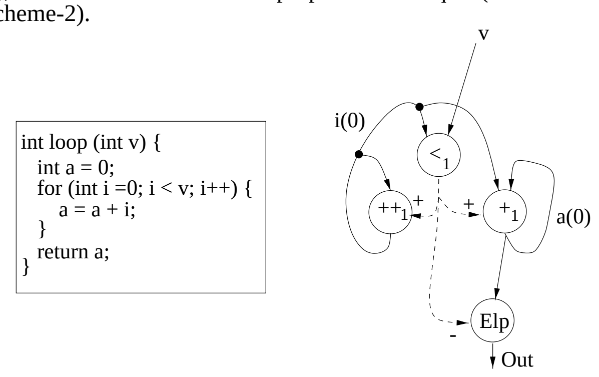


Figure 1: CDFG of Loop1 benchmark

Figure 2 illustrates the steps in the estimation process of scheme-1. Using an RTL circuit of the benchmark and a typical input trace[2], various statistics of the signals in the circuit are collected using a cycle-based simulator. Once the statistics are collected, the numbers are fed into the computational engine that uses power models from a design/module library to calculate the circuit power. The time taken by the estimation algorithm is linear in the number of traces applied to the simulator. The bottleneck in this estimation procedure is RTL simulation (Block 1 in Figure 2).

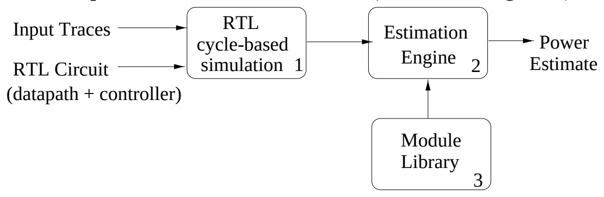


Figure 2: Block diagram of the power estimator [8]

Our proposed estimator is illustrated in Figure 3. We first perform a behavior-level simulation by providing as input the CDFG, and the set of input traces to a simulator. From this simulation, we extract branch probabilities and switching activity (SA) information which are then passed on to the estimation engine along with the RTL circuit and design library.

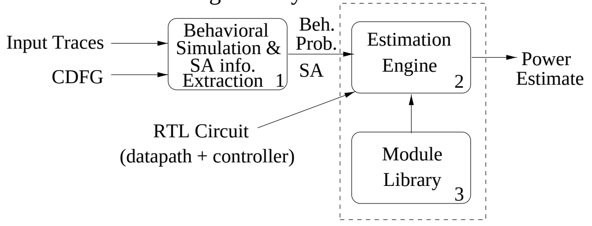


Figure 3: Block diagram of our proposed power estimator

The bottleneck in scheme-2 is also the simulation time. The time complexity of our estimator as illustrated in Figure 3, is also linear in the size of the input trace. Where is the advantage of scheme-2 then? Consider having to evaluate one hundred architectures for the behavior given above. Using scheme-1, one hundred simulations will have to performed - one for each architecture. Using scheme-2, only one behavioral simulation is required, since the estimator only executes the blocks highlighted with a dotted line in Figure 3. Experiments show that for the example in Figure 1, a tracking index[3] of 0.99 is achieved for scheme-2, and the final architecture of the synthesis run is identical using both scheme-1 and scheme-2. The speed-up in execution time of scheme-2 over scheme-1 is about 100X.

[2]The input trace is created by a designer with knowledge about the circuit's functionality and interface. For experimental purposes, the traces may also be automatically generated.

[3]Tracking index is defined formally in Section 5. It represents how well the estimator evaluates different architectures in terms of relative power consumption. It varies from 0 to 1, 1 being perfect.

3 Preliminaries

Power consumption in CMOS circuits may be divided into two categories: *static* and *dynamic*. Static power dissipation is mainly due to *leakage* and *standby* currents. With the correct choice of device technology, static power loss can be made negligible [4]. Dynamic power loss is due to the *charging* and *discharging* of transistors, and *short-circuit* currents. Modern design techniques can reduce short-circuit currents. Hence, the average power consumption of a CMOS gate, based on its charging and discharging, is a function of the gate's output capacitance, supply voltage, and transitions per time period (switching activity).

One approach for optimizing power consumption is by using iterative algorithms (simulated annealing, iterative improvement, genetic algorithm). In such systems, the inner loop of the optimization algorithm performs a set of *moves* on an architecture to produce other architectures of the design. The moves affect different high-level synthesis tasks, such as scheduling, allocation, and assignment, and hence, enables the system of examine the interaction among these tasks. The estimation engine estimates the power consumption of each architecture and helps choose which move is the most beneficial with respect to a given cost function. Clearly, the estimator has a direct impact on the above high-level synthesis tasks via the moves that are chosen.

Imagine taking a snapshot of an iterative synthesis process at a given moment in time. The picture will include a CDFG, a schedule and state transition graph (STG) from which a controller is derived, a datapath and the set of input traces. These are the components fed to the power estimator. Next, we define these components in more formal terms and explain their contribution to power estimation.

3.1 The Control-data Flow Graph Model

The high-level synthesis process starts with a hardware description of the design that is compiled into a CDFG. A CDFG is a directed graph where nodes represent operations and edges represent data and control dependencies among the nodes. The model used is formally defined in [12]. However, for the sake of completeness, we cover some of its important aspects. Referring to the CDFG in Figure 1, we represent a data dependency with a solid arc, and a control dependency with a broken arc. Unlike data dependencies, each node may, at most, be control-dependent on one other node. The control dependency may have two forms: (a) *active-high*, denoted by $+$ on the edge, meaning a node will only execute if the condition is true, and (b) *active-low*, denoted by $-$ on the edge, meaning the node in question will only execute if the associated condition is false. While nodes perform operations such as ADD, MULTIPLY and SELECT, edges carry data values which may be constants, variables, or bit-vectors. We use an *Elp* node to denote the termination of a loop construct.

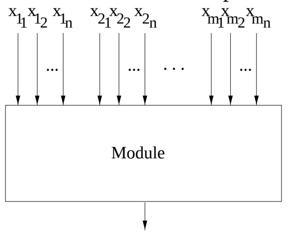
There are many types of condition constructs that may be used in a behavioral description. These include *if-then-else*, *while-do*, *case statements*, *wait statements*, and *loops*. In order to make handling conditionals uniform and as straightforward as possible, all constructs are converted to the *if-then-else* form.

3.2 The State Transition Graph

The CDFG is used for (a) deriving a datapath, (b) determining the control mechanism through scheduling, and (c) performing behavioral simulations. The controller is derived from an STG. The behavioral simulation results are used to compute the state transition probabilities for every edge-linked state pair in the STG. The probability of a state transition is computed as the product of the probabilities of the control dependency edges it represents, assuming mutual independence of these edges. For control-flow intensive designs, this assumption has been experimentally validated in [14]. By using this approach, it is only necessary to perform one behavioral simulation from which the state transition probabilities may be extracted. These probabilities are then used to derive the expected schedule length of the design. It is important to point out, however, that in [14] concurrent loop optimization was not handled. In other words, it was assumed that only one loop can execute at a given time. It has been shown in [15] that optimizing across concurrent loops is a key process in optimizing the performance of control-flow intensive designs. In the presence of such concurrent constructs, the procedure presented in [14] no longer gives accurate results for expected schedule length, and errors of up to 50% in the schedule length have been demonstrated[4]. We will illustrate the problem in Section 4.2 and present a solution to modify the method of [14] to account for these constructs.

3.3 The Module Power Model

The power consumption of a module (functional unit, register or multiplexer) is expressed as a linear weighted sum of the switching activities of its inputs. Consider an m-input, n-bit module shown in Figure 4.



$x_{1_1} x_{1_2} x_{1_n}$ $x_{2_1} x_{2_2} x_{2_n}$ $x_{m_1} x_{m_2} x_{m_n}$
Module

Figure 4: Block diagram of a generic module

The capacitance, C_{mod}, switched by the module, mod, during a state transition between source state s_1 and sink state s_2 is given by:

$$C_{mod} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i_j}^{s_1 \rightarrow s_2} \times c_{i_j} \tag{1}$$

where $a_{i_j}^{s_1 \rightarrow s_2}$ is the activity at input x_{i_j} of the module during the state transition $s_1 \rightarrow s_2$ and c_{i_j} is a module-dependent constant. The constant c_{i_j} is derived from switch-level simulation of the layout of the module, and is stored as part of the design library. Let u (v) be the variable that appears at input i of module mod in state s_1 (s_2). $a_{i_j}^{s_1 \rightarrow s_2}$ is computed as:

$$a_{i_j}^{s_1 \rightarrow s_2} = p(u_j \oplus v_j = 1) \tag{2}$$

where $p(u_j \oplus v_j = 1)$ is the probability that bit j of u differs from bit j of v. These probabilities are computed only once in a preprocessing step, shown as Block 1 in Figure 3, and are stored in a data structure known as a *switching activity matrix.*

A switching activity matrix [16] M is a matrix whose rows and columns correspond to variables in the behavior. Row i represents variable u_i and column j represents variable u_j. The entry $M[ij]$ is an ordered set (from most significant to least significant bit) of cardinality equal to the bit-width of the variables, where the k^{th} element of the set represents the probability that the k^{th} bit of u_i and the k^{th} bit of u_j differ.

To construct the switching activity matrix, we use the traces for individual variables which are produced by behavioral simulation. Let t_u (t_v) be the trace for u (v); $t_u[k]$ ($t_v[k]$) represents its k^{th} element. Then, $p(u_j \oplus v_j = 1)$ can be computed as:

$$\frac{\sum_{k=1}^{N} (t_u[k] \oplus t_v[k])_j}{N} \tag{3}$$

where N is the length of the trace. The matrix is built only once for all pairs of variables in the behavior.

Note that we automatically account for sharing, *i.e.*, when more than one operation (variable) are mapped to the same functional unit (register). For example, an adder may perform operation $+_1$ in state s_1 and operation $+_2$ in state s_2. By considering the state transition $s_1 \rightarrow s_2$, we account for the effects of the inputs of the adder changing from the values associated with $+_1$ to the values associated with $+_2$.

In the next section, we combine the concepts described above and formally define the estimation algorithm.

[4]The expected schedule length is often an ill-conditioned function of the state transition probabilities. For instance, the expected schedule length is often directly proportional to $\frac{1}{1-p_x}$ (p_x is the state transition probability). Therefore, a change in p_x from 0.99 to 0.98 would result in a 50% change in the expected schedule length.

4 The Power Estimation Algorithm

The block diagram of the estimation process is shown in Figure 3. In this section, we concentrate on Block 1 and Block 2 of the diagram. Note that Block 1 is a one-time cost. We first illustrate the algorithm through an example.

4.1 Power Estimation of a Simple Benchmark

As mentioned before, behavioral simulation provides us with probabilities of the conditional edges of the CDFG and with the switching activity matrix. We explain each step in the estimation process by using the example shown in Figure 5. The CDFG consists of two conditional branches and one loop. The edges have been annotated with names for which the probabilities have been given.

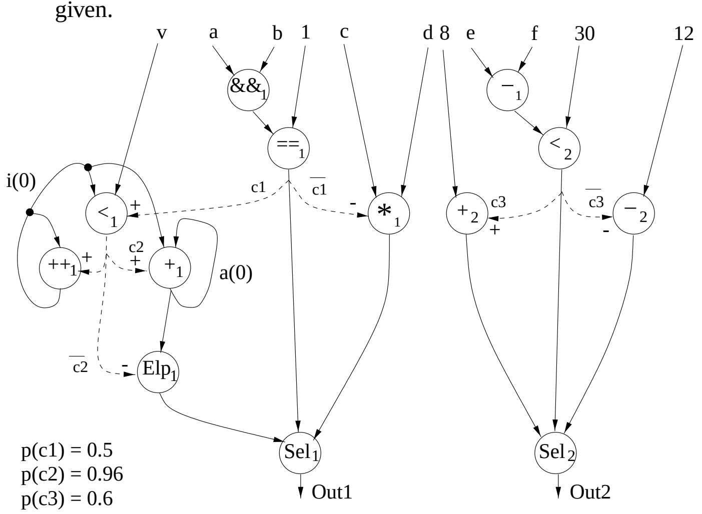


Figure 5: CDFG of an example benchmark

An STG for the CDFG is given in Figure 6. The first step in the estimation process is to annotate the STG edges with the appropriate probability values. Using the assumptions from Section 3.2, the edge probabilities for the given edges are: $p(c_1 c_3) = 0.3$, $p(c_1 \bar{c_3}) = 0.2$, $p(\bar{c_1} c_3) = 0.3$, and $p(\bar{c_1} \bar{c_3}) = 0.2$.

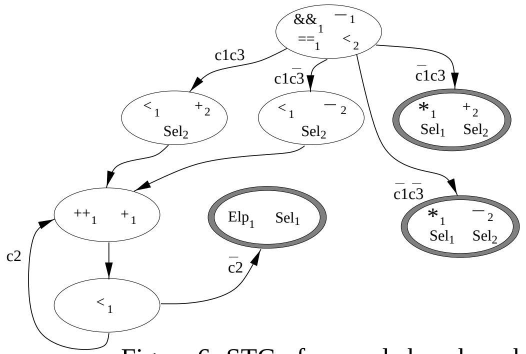


Figure 6: STG of example benchmark

Once the state transition probabilities have been established, a set of linear equations is derived for the state probabilities based on modeling the STG as a Markov process. As shown in [14], this method is valid for control-flow intensive behaviors. The equations are solved using a linear equation solver, and the state probabilities are derived. Next, the switching activity matrix in conjunction with the technology constants (c_{i_j}'s) are used to derive the switched capacitance of the RTL circuit. Unlike data-dominated applications, for which there is only one path in the STG, control-flow intensive designs have many threads of execution, each with an associated probability of execution. Using Equation (1), we find the switched capacitance associated with a state transition edge $s_1 \rightarrow s_2$ in the STG is given by:

$$C_{edge} = \sum_{\forall\ modules} p(s_1) \times P_{s_1 \rightarrow s_2} \times C_{mod} \tag{4}$$

where $P_{s_1 \rightarrow s_2}$ is the state transition probability from state s_1, which has a state probability $p(s_1)$, to state s_2. The average switched capacitance for the entire circuit is found by summing up the edge switched capacitance over all edges:

$$C_{total} = \sum_{\forall\ edges} C_{edge} \tag{5}$$

4.2 Effect of Loop Optimization

As mentioned before, special attention needs to be given to loops, especially in the case where concurrent loops are unrolled to improve schedule length. Consider the simple example of two loops executing concurrently in Figure 7. An optimizing scheduler

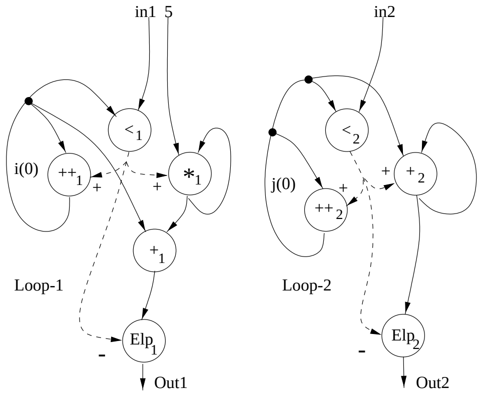


Figure 7: CDFG of two concurrent loops

will perform loop unrolling and schedule these loops concurrently. This is a clear advantage in terms of performance since these loops are independent, but may also require sharing of resources. Figure 8 is part of an STG where the loops in Figure 7 are unrolled twice. By examining the STG, it can be seen that at state 3, Loop-2 has already executed twice while Loop-1 is still in the first iteration. It is, therefore, necessary to account for the fact that at certain points in the STG, a given loop may have executed a given number of times. If the accounting is not performed, the error in expected schedule length can be significant.

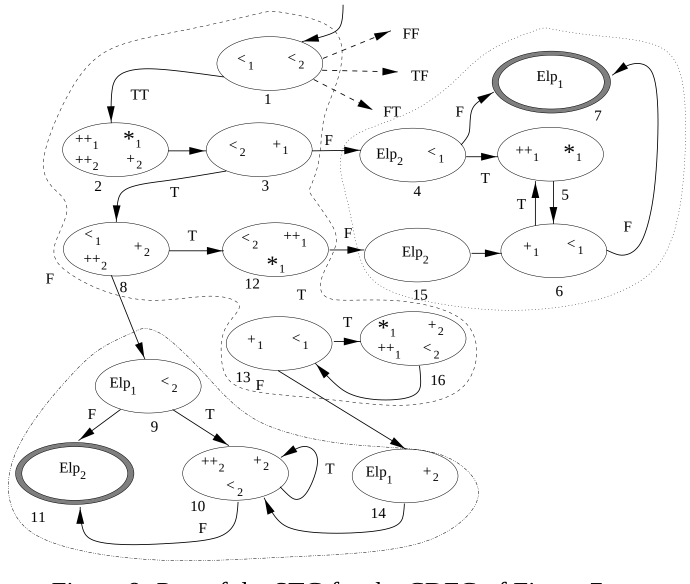


Figure 8: Part of the STG for the CDFG of Figure 7

Clearly, the length of the schedule for the CDFG of Figure 7 is sensitive to inputs in_1 and in_2. Once again, consider state 3 and, more specifically, the edge marked "T" emanating from it. Assume that the value of in_2 is 99. Without considering concurrent loop optimization, we may naively arrive at a state transition probability of 0.99 for the edge. In reality, the value of the probability is $\frac{in_2-2}{in_2+1}$, since Loop-2 has closed twice already. The state transition probability is, therefore, 0.97. The 0.02 variation in the state transition probability may result in an error of 66% in the expected schedule length (see footnote 3).

The solution is composed of two phases. The first phase is to divide the STG into regions shown by dashed arcs. The CDFG may be viewed as a hierarchy of nested loops, and this hierarchy can be used to induce a partition on the STG. Each region represents the states in which the same loops are executing simultaneously.

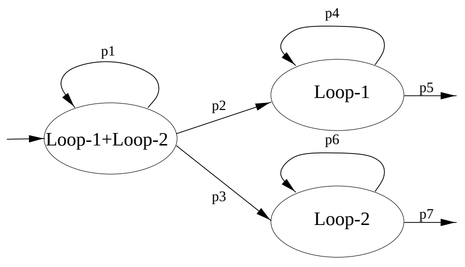


Figure 9: Simplified STG of Figure 8

In the simplified version of Figure 8, given in Figure 9, the states are compacted into those regions where (1) Loop-1 and Loop-2 execute concurrently, (2) Loop-1 executes by itself, and (3) Loop-2 executes by itself. The outgoing edges from these regions are annotated appropriately by considering how many times each loop has executed within the region. This process requires the use of the user-defined input traces. Once the regions have been defined, and the boundary probabilities assigned, the state transition probabilities are evaluated. Each state within a region is traversed and marked with the number of times a loop has been encountered. Once this step is performed, the state transition probabilities are updated by accounting for (a) the number of times a loop has been executed before entering a region, (b) the total number of times a loop is executed within a region, and (c) the total number of times the loop executes over all regions.

Let us return to the example and apply this method to see how the state transition probabilities may be derived. Now assume that both in_1 and in_2 have the same value of 99. This implies that both loops will execute simultaneously for some period of time. By examining the sequence of operations, behavioral simulation will indicate how many times Loop-2 closes before Loop-1. In this case, we have the following sequence: $(\{<_1\}, \{<_2\})$, $(\{{++}_1, *_1\}, \{{++}_2, +_2\})$, $(\{+_1\}, \{<_2\})$, and so on. Over the entire simulation run, Loop-2 will close, on an average, twice before Loop-1 closes. In other words, Loop-2 is twice as fast as Loop-1. Using this information, we annotate the probabilities in Figure 9 to account for this fact as follows:

$$p_1 = \left(\frac{\frac{in_1}{2}}{\frac{in_1}{2}+1}\right)\left(\frac{in_2}{in_2+1}\right),\ p_2 = \left(\frac{\frac{in_1}{2}}{\frac{in_1}{2}+1}\right)\left(\frac{1}{in_2+1}\right),$$

$$p_3 = 0,\ p_4 = \left(\frac{\frac{in_1}{2}}{\frac{in_1}{2}+1}\right),\ p_5 = \left(\frac{1}{\frac{in_1}{2}+1}\right)$$

Using the above state region transition probabilities, we can calculate the probabilities for the different state regions, *i.e.*, the probability of being in each region. Once the boundary conditions are evaluated, we can move down the hierarchy and evaluate the state transition probabilities within each region. For example, if we traverse the STG from state 1 to state 13, we note that Loop-2 has been encountered at least four times up to that state. The loop in the STG between states 13 and 16 introduces non-determinism into the state transition probability computation. The boundary information available indicates that Loop-2 will execute $in_2 - 4$ times between states 13 and 16 before exiting the region these states are a part of. Therefore, the probability of closure, which is the state transition probability from state 13 to 16, is $\frac{in_2-4}{in_2-3}$. The procedure can be applied in a straightforward fashion to each visited state. After the evaluation of the state transition probabilities, the state probabilities are calculated, as before, by solving a system of linear equations.

4.3 Controller Power Estimation

The controller can constitute a significant portion of circuit area and power for control-dominated circuits, which have a small datapath and a large controller. For such circuits, it is necessary to accurately estimate the controller power since it is the primary component. In control-flow intensive circuits, the controller is not as significant - results in [15] indicate that the controller size typically constitutes only 2% of the total circuit area when taken to layout. One reason for this is that the datapath in control-flow intensive circuits accommodates control-data chaining in a natural fashion. However, there is still a need to estimate the controller power.

The model used for controller power estimation is derived from [17], where the switched capacitance for the controller was found to be:

$$C_{FSM} = \alpha_1 N_{states} + \alpha_2 \tag{6}$$

where α_1 and α_2 are technology-dependent constants, and N_{states} is the number of states in the controller. This equation shows that the controller power is directly proportional to the number of states. Even though this model was derived primarily for data-dominated circuits, it is also valid for control-flow intensive circuits because the ratio of datapath to controller size is comparable.

4.4 Putting it all Together

Figure 10 represents the pseudo-code for the power estimation algorithm. First, the algorithm calls the function DERIVE_STATE_TRANS_PROB which performs the two-level sweep of the STG described in Section 4.2. The STG is first partitioned into the loop-induced regions previously defined, and the transition probabilities in and out of these regions are computed. The probabilities are based on the number of times each loop executes within a given region. Once the boundary probabilities have been assigned, a second pass of the STG visits each state to update the state transition probabilities. Similar to the first pass, the second one also keeps track of how many times individual loops have executed within each state. Once these numbers are available, the state transition probabilities are calculated taking all other (non-loop) branch probabilities into consideration. After the state transition probabilities have been computed, the updated STG is used to derive a set of linear equations. The equations are solved using Gauss-Jordan elimination in order to calculate the state probabilities.

Second, the algorithm uses the switching activity matrix and module capacitance values to calculate the total capacitance switched for each module, and simultaneously, computes the switched capacitance for the entire datapath using Equation (5). The last step is estimating and adding the controller's switched capacitance using Equation (6).

```
ESTIMATE_POWER (CDFG $G$, Datapath $D$, STG $S$, Matrix $M_{ac}$, Library $L$) {
  State_Trans_Probs $P_{s_i \to s_j}$ = DERIVE_STATE_TRANS_PROB($S$, $P_{bev}$);
  State_Probs $p(s)$ = GAUSS_JORDAN_ELIMINATION($S, P_{s_i \to s_j}$);
  Switched Capacitance $C_{total} = 0$;
  for_each_state_transition ($s_i \to s_j$ in $S$) {
    for_each_module ($mod$ in $D$) {
      $C_{mod} = 0$;
      for_each_input ($ip$ in $mod$) {
        $u$ = get_input ($ip$ of $mod$ in $s_i$);
        $v$ = get_input ($ip$ of $mod$ in $s_j$);
        //$u$ is the variable in $s_i$ and $v$ is the variable in $s_j$
        for_each_bit ($b$ in $ip$) {
          $C_{mod} = C_{mod} + M_{ac}[uv]_b \times L[mod] \to c_{ip_b}$;}}
      $C_{total} = C_{total} + p(s_i) \times P_{s_i \to s_j} \times C_{mod}$;}
    }
  }
  Switched Capacitance $C_{fsm}$ =ESTIMATE_CONTROLLER_POWER ($S$);
  return $C_{total} + C_{fsm}$ ;
}
```

Figure 10: Pseudo-code for the power estimation algorithm

5 Experimental Procedure

The estimation algorithm is implemented within the IMPACT [12] synthesis system. The benchmarks used for the experiments are: Greatest Common Divisor (GCD), a system of parallel loops (Loops) [12], the send process of the X.25 communications protocol [14], and Paulin, which is a data-dominated circuit.

IMPACT takes in as an input a CDFG of the benchmark, and performs behavioral simulation to extract the branch probabilities. The system then optimizes the design for low power using iterative improvement. During iterative improvement, the system may compare hundreds of architectures, by applying various architectural moves to each architecture.

All estimated power results obtained using our algorithm are compared to power values obtained using a switch-level simulator. From the architecture level, the MSU standard cell library with the logic synthesis tool *SIS* is used to produce a logic-level netlist. The *Octtools* suite is then used to perform layout and routing of the controller and datapath. Transistor and wiring capacitances, extracted from the layout using MAGIC, are used to annotate the netlist. *IRSIM-CAP*, a switch-level simulator, is used to calculate the switched capacitance from which power consumption is derived. The switch-level estimator takes into account the interconnect power, glitching power, clock network power, *etc*, which are difficult to account for at the higher levels[5]. The input traces for the simulation are automatically generated by creating a zero-mean normal distribution, which is then passed through an autoregressive filter to introduce a temporal correlation.

The experiments are divided into three categories. Each category is designed to measure a different aspect of the estimation algorithm. Our experiments measure both the accuracy and the fidelity of the estimation algorithm. Some terms are defined next.

Estimation Accuracy: For each benchmark, we measure how accurate the power reported by our estimation algorithm is to the power estimated using *IRSIM-CAP*. The absolute error is averaged out over the architectures produced and compared during a synthesis run of the high-level synthesis system. Note that positive and negative errors are not canceled out!

Tracking Index: This index is used to measure how well the estimation algorithm compares the relative power of a pair of architectures. In other words, given n architectures, the index measures the correctness of the order in which the architectures are placed in terms of decreasing power consumption. The equation for tracking index is given as:

$$I = 1 - \frac{k}{{}^{n}\mathrm{C}_2} \tag{7}$$

where k is the number of out-of-order pairs within n architectures, and the correct order is assumed to be defined by *IRSIM-CAP*.

This series of experiments measures the tracking index for all the given benchmarks as they are processed in the power-optimizing synthesis tool. At each step, the synthesis algorithm pauses and a "snapshot" of the architecture is taken. The power of each architectural choice is estimated using our algorithm and *IRSIM-CAP*. Ideally, each successive architecture should have less power than the previous one.

Tracking Index Fidelity: Throughout a synthesis run, the difference in power consumption of successive architectures will vary. The fidelity measure indicates the estimation algorithm's confidence level for a given change (δ) in power.

This set of experiments measures the tracking index versus the percentage separation in architecture power, ranging from 10% to 100%. To avoid introducing second-level error effects, we do not go below 10%, since *IRSIM-CAP*'s confidence level decreases beyond this point. For each 10% separation in power, we average out the tracking index for the different pairs of architectures available from all four benchmarks.

6 Experimental Results

In this section, we present the results of our experiments described above. Table 1 summarizes the results for all four benchmarks. The first column represents the architecture number. The *IC* column represents the power as estimated by *IRSIM-CAP*, the *EP* column represents the estimated power using our algorithm, and

[5]While circuit simulators such as SPICE may be very accurate in terms of estimating power consumption, they take a prohibitively long time to complete on the type of controller/datapaths targeted here. For this reason, we use *IRSIM-CAP* which is much faster than SPICE and has an average margin of error of 7% compared to SPICE.

Table 1: Summary of results for our power estimation algorithm

Arch.	Loops			GCD			X.25			Paulin		
No.	*IC* (mw)	EP (mw)	%Er	*IC* (mw)	EP (mw)	%Er	*IC* (mw)	EP (mw)	%Er	*IC* (mw)	EP (mw)	%Er
1	24.76	26.13	5.5	8.23	9.43	14.6	15.30	13.51	-11.7	21.95	24.51	11.7
2	26.78	24.01	-10.3	6.42	7.99	24.5	12.14	10.64	-12.4	21.68	23.33	7.6
3	24.18	23.35	-3.4	7.89	7.79	-1.3	8.51	7.95	-6.6	20.85	19.40	-7.0
4	23.92	22.48	-6.0	6.67	7.39	10.8	6.94	5.36	-22.8	20.53	19.25	-6.2
5	21.81	20.71	-5.0	5.98	6.96	16.4	4.51	5.21	15.5	20.12	18.73	-6.9
5	20.15	18.83	-6.6	6.59	6.10	-7.4	4.13	5.01	21.3	18.95	17.89	-5.6
6	18.10	16.37	-9.6	6.32	6.01	-4.9	3.81	4.85	27.3	18.62	17.04	-8.5
7	14.02	15.96	13.8	5.15	5.63	9.3	3.99	4.69	17.5	17.82	16.53	-7.2
8	12.15	13.62	12.1	4.62	5.01	8.4	4.82	4.58	-5.0	16.15	15.46	-4.3
9	10.73	12.28	14.4	4.25	4.55	7.1	2.89	3.55	22.8	14.34	15.32	6.8
10				3.01	3.89	29.2	2.19	2.73	24.7	13.86	14.23	2.7
11				2.72	3.18	16.9				10.93	12.10	10.7
12				2.43	2.26	-7.0						
13				1.58	1.90	20.3						
14				2.02	1.75	-13.4						
15				2.15	1.57	-27.0						
I	0.98			0.92			0.89			0.99		

%Er is the percentage error. Over all the benchmarks, the average absolute error is only 11.8%.

Results for the tracking index, I, for the benchmarks, which go through a synthesis run, are shown in the last row of the table. The number of architectures varies from one benchmark to the other since each architecture will require a different number of architectural moves to produce a power-optimized circuit. The average tracking index is 0.95. Figure 11 is a plot showing the fidelity of the tracking index with increasing separation in power between successive architectures. Above 20% separation, the tracking index rises dramatically.

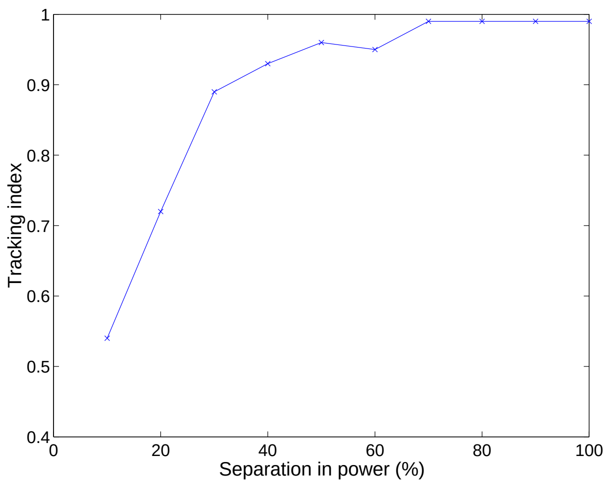


Figure 11: Tracking index vs. % separation in power

On an *SGI Challenge* workstation with $256MB$ of memory, on an average, it takes about 15 minutes of CPU time to evaluate the switching activity matrix, which is a one-time cost. Consecutive power evaluations take about 30 seconds of CPU time for each architecture.

7 Conclusions

In this paper, we have presented a power estimation algorithm for control-flow intensive designs, which can also handle data-dominated designs. The algorithm allows an iterative high-level synthesis system to evaluate architectural trade-offs without introducing a run-time bottleneck which typically results from repeated simulations and computations.

The algorithm utilizes behavioral information to extract accurate branch probabilities, and is based on the concept of switching activity matrices. We combine the probability measures and matrices with a pre-characterized design library to calculate the power of a given architecture. Results demonstrate that, on an average, our algorithm only has 11.8% error.

References

[1] A. C. Deng, "Power analysis for CMOS/BiCMOS circuits," in *Proc. Int. Wkshp. Low Power Design*, pp. 3-8, Apr. 1994.

[2] F. N. Najm, "Power estimation techniques for integrated circuits," in *Proc. Int. Conf. Computer-Aided Design*, pp. 492-499, Nov. 1995.

[3] M. Pedram, "Power minimization in IC design: Principles and applications," *ACM Trans. Design Automation Electronic Systems*, vol. 1, pp. 3-56, Jan. 1996.

[4] J. Rabaey and M. Pedram (Editors), *Low Power Design Methodologies*. Kluwer Academic Publishers, Boston, MA, 1996.

[5] S. Gupta and F. N. Najm, "Power macromodeling for high level power estimation," in *Proc. Design Automation Conf.*, pp. 365-370, June 1997.

[6] S. R. Powell and P. M. Chau, "Estimating power dissapation of VLSI signal processing chips: The PFA technique," in *Proc. VLSI Signal Processing IV*, pp. 250-259, 1990.

[7] P. Landman and J. M. Rabaey, "Architectural power analysis: The dual bit type method," *IEEE Trans. VLSI Systems*, vol. 3, pp. 173-187, June 1995.

[8] A. Raghunathan, S. Dey, and N. K. Jha, "Register-transfer level estimation techniques for switching activity and power consumption," in *Proc. Int. Conf. Computer-Aided Design*, pp. 158-165, Nov. 1996.

[9] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures of energy consumption at the register-transfer level," in *Proc. Int. Symp. Low Power Design*, pp. 81-86, Apr. 1995.

[10] F. Najm, "Towards a high-level power estimation capability," in *Proc. Int. Symp. Low Power Design*, pp. 87-92, Apr. 1995.

[11] L. Benini, A. Bogliolo, M. Favalli, and G. De Micheli, "Regression models for behavioral power estimation," in *Proc. Power & Timing Modeling Optimization & Simulation Wkshp.*, pp. 179-188, 1996.

[12] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, "IMPACT: A high-level synthesis system for low power control-flow intensive circuits," in *Proc. Design Automation & Test in Europe Conf.*, pp. 848-854, Feb. 1998.

[13] A. P. Chandrakasan, S. Sheng, and R. W. Broderson, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, pp. 473-484, Apr. 1992.

[14] S. Bhattacharya, S. Dey, and F. Brglez, "Performance analysis and optimization of schedules for conditional and loop-intensive specifications," in *Proc. Design Automation Conf.*, pp. 491-496, June 1994.

[15] G. Lakshminarayana, K. S. Khouri, and N. K. Jha, "*Wavesched*: A novel scheduling algorithm techniques for control-flow intensive behavioral descriptions," in *Proc. Int. Conf. Computer-Aided Design*, pp. 244-251, Nov. 1997.

[16] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," in *Proc Int. Conf. Computer Design*, pp. 318-322, Oct. 1994.

[17] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Broderson, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 12-51, Jan. 1995.