Activity Estimation for Field-Programmable Gate Arrays

Julien Lamoureux, *Steven J.E. Wilton*

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., Canada.
email: julienl@ece.ubc.ca, stevew@ece.ubc.ca

ABSTRACT

This paper examines various activity estimation techniques in order to determine which are most appropriate for use in the context of field-programmable gate arrays (FPGAs). Specifically, the paper compares how different activity estimation techniques affect the accuracy of FPGA power models and the ability of power-aware FPGA CAD tools to minimize power.

After comparing various existing techniques, the most suitable existing techniques are combined with two novel enhancements to create a new activity estimation tool called ACE-2.0. Finally, the new publicly available tool is compared to existing tools to validate the improvements. Using activities estimated by ACE-2.0, the power estimates and power savings were both within 1% of the results obtained using simulated activities.

1. INTRODUCTION

Advancements in process technologies, programmable architectures, and CAD tools are enabling the implementation of large applications on FPGAs. Although FPGA implementations provide significant speedups and power reductions compared to software implementations, programmable devices still consumes significantly more power and are slower than Application Specific Integrated Circuit (ASICs). Thus, careful delay and power modeling of programmable devices is critical to ensure that the system meets its speed and power requirements. In this paper, we focus on power since it is quickly becoming the limiting factor for many systems implemented on FPGAs.

In addition to being accurate, it is also important that FPGA power models are fast. FPGAs are often used because their programmability allows designers to quickly implement and debug their application. If the power models are too slow, the design cycle becomes cumbersome and FPGAs loose some of their appeal. The power models currently provided by FPGA vendors typically rely on simulation-based activity estimation techniques [1,2]. Although the accuracy inherent in these techniques is important for hardware designers, they are not as useful in FPGAs since they are slow for large system-level applications and require input vectors. In many cases, power estimates are needed when designing a system but realistic input vectors are not available.

Instead, probabilistic-based techniques provide power estimates very quickly at the expense of a small amount of accuracy. Probability-based activity estimation has been well studied [3,4,5,6,7,8], with techniques that range in terms of speed and accuracy; however, there have been no studies which determine which of the previously published techniques are appropriate in FPGAs. This is the objective of this paper. Specifically, we answer the following questions:

1. How does the accuracy of the switching activity estimates affect the accuracy of the final power estimates in FPGAs? If the accuracy of the power estimate is not overly sensitive to the accuracy of the activity estimates, then a faster technique may be sufficient.
2. How does the accuracy of the switching activity estimates affect the ability of power-aware CAD tools to minimize power in FPGAs? Knowing how sensitive the power-aware CAD tools are to the accuracy of the activities will help to determine what techniques are most appropriate.
3. Should different activity estimation techniques be used for different applications? Estimating activities for circuits with many state storage elements (flip-flops) is more difficult than for circuits with fewer state storage elements. Perhaps accurate techniques should be used for circuits with many flip-flops and faster techniques can be used otherwise.

Once these questions have been answered, we present a new activity estimation tool called ACE-2.0, which was developed based on the answers to the above three questions. The tool is compatible with the Versatile Place and Route [9] and the power model from [10], and is publicly available.

This paper is organized as follows. Section 2 summarizes existing activity estimation techniques and describes an existing power model and power-aware CAD flow for FPGAs. Section 3 then examines the accuracy of the power model and the performance of the power-aware CAD flow when various activity estimation techniques are employed. In doing so, it answers the three questions listed above. Section 4 first describes a new activity estimation tool called ACE-2.0, which incorporates the techniques found suitable in Section 3 and then compares the new activity estimation tool to an existing activity estimation tool called ACE-1.0. Finally, Section 5 describes our conclusions and future work. Instructions for downloading our new tool are provided in the appendix.

2. BACKGROUND

2.1 Terminology

Switching information is required when estimating static and dynamic power dissipated by integrated circuits. Three statistical values are commonly used to describe the switching behavior of circuit wires, namely: static probability, switching probability, and switching activity. The static probability of a wire x, denoted $P_1(x)$, is the probability that the value of a wire is logic high. This value is used in static power calculations to determine how long transistors are in a particular leakage state. The switching probability of a wire x, denoted $P_s(x)$, is the probability that the steady-state value of the wire will transition from logic high to logic low or from logic low to logic high during a given clock cycle. Since the steady-state value of a wire x transitions either once or not at all during a clock cycle, $0 \leq P_s(x) \leq 1$. This value can be used to determine the switching activity at the output of flip-flops. Finally, the switching activity of a wire x, denoted $A_s(x)$, refers to the average number of times that the wire will transition from logic high to logic low or from logic low to logic high during a clock period. Although its steady-state value can only change once per cycle, a wire can actually transition multiple times before reaching steady-state since the arrival times at the inputs of a logic gate may be different, $P_s(x) \leq A_s(x)$. These spurious transitions are called glitches. Switching activities are used in dynamic power calculations.

2.2 Simulation-Based Activity Estimation

Simulation-based activity estimation can be employed at various levels of abstraction, which include: switch-level, gate-level, and block-level. In this paper, we only consider gate-level simulation since it provides very accurate activity estimates and is significantly faster than switch-level simulation, which simulates transitions at the transistor level. Block-level simulation, which considers larger blocks such as registers, adders, multipliers, memories, and state machines, is not considered.

Gate-level simulation-based activity estimation involves simulating a Boolean network consisting of logic gates and latches while keeping track of transitions in order to determine P_1, P_s, and A_s for each node in the network. During a simulation, the value at the output of a gate is determined from the values at the input of the gate each time an input changes. Gate-level simulation is a well studied problem and much effort has been placed on improving its speed [11,12]. Despite many innovations, such as Monte Carlo techniques, gate-level simulation for large system-level designs can still take as long as days. Moreover, simulation requires input vectors, which are often not available when designing a new system.

2.3 Probabilistic-Based Activity Estimation

Probabilistic-based (or vectorless) activity estimation is typically faster than simulation since it involves a one-time calculation for each node of a circuit. The most straightforward way of calculating $P_1(n)$, $P_s(n)$ and $A_s(n)$ is to visit the nodes one at a time, starting from the primary inputs, as shown in Figure 1. For each node, the Lag-one model from [6] and the Transition Density model from [8] can be used to calculate $P_1(n)$, $P_s(n)$ and $A_s(n)$, based on $P_1(f)$, $P_s(f)$ and $A_s(f)$ of each fan-in node f of n, and the function implemented by node n. These quantities can then be used to estimate the power using standard power models [10].

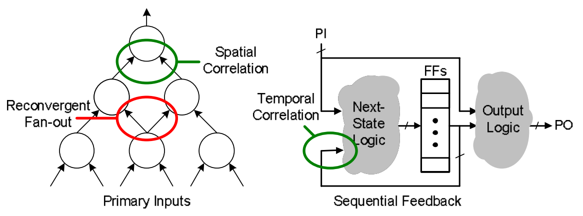
```
calculate_switching_activities( network )
{
  foreach node n ∈ network {  // topological order (PIs to POs)
    calculate $P_1(n)$, $P_s(n)$, $A_s(n)$ using techniques from [5,8]
  }
}
```

Figure 1: Pseudo-code of activity calculation.

The probabilistic method shown in Figure 1 is less accurate than simulation-based estimates for two reasons. First, it ignores the effects of wire and gate delays. These delays affect the arrival times of gate inputs. Techniques to take these delays into account have been described in [4]; however, these techniques are not considered in this paper since delay information may not be available.

The second reason that the probabilistic method above is less accurate than simulation-based estimates is that it ignores the correlation between signals. There are two types of correlation: spatial correlation and temporal correlation. Spatial correlation occurs when the logic value of a wire depends on the value of another wire. Spatial correlation can be introduced directly at the primary inputs or can occur between internal nodes when gates fan-out to multiple gates and later reconverge, as illustrated in Figure 2 (a). Temporal correlation occurs when the value of a wire depends on previous values of the same wire. This can also be introduced directly at primary inputs or can occur within sequential circuits which have feedback, as illustrated in Figure 2 (b). From [13], ignoring temporal correlation introduces between 15% and 50% error, and ignoring spatial correlation introduces between 8% and 120% error.

Probabilistic techniques that consider signal correlation have been described in previous works [4,6]. Unlike previous techniques, which estimate the activity of each gate output separately, error introduced by reconvergent fan-out can be eliminated by collapsing nodes with its predecessors into a larger node. In this technique, each time a node is visited, it is collapsed with all its predecessors to form a new (larger) node c. The inputs of c consist of some, or all, of the primary inputs of the circuit, and no other signals. The switching activity at the output of node c (and hence the output of node n) are then calculated as before. Typically, a binary decision diagram (BDD) representation of c is used, since this representation is efficient for large nodes.



(a) Example of internal spatial correlation (b) Example of internal temporal correlation

Figure 2: (a) Spatial correlation (b) Temporal correlation.

Techniques that consider temporal correlation have also been described in previous works. In [5,6,7], internal temporal correlation is captured by "unrolling" next state logic (see Figure 3) and iteratively estimating switching activities until the activities converge. By unrolling the next logic ∞ times, all temporal correlation can be captured. Although unrolling ∞ times is infeasible, [5] found that 3 times produced good results.

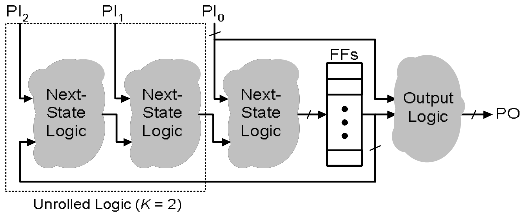


Figure 3: Example of unrolling next-state logic.

2.4 Existing Activity Estimation Tools

Despite a large number of activity estimation techniques described in the literature, very few fully functional activity estimation tools are available since many of the techniques are specific to a certain type of circuit (sequential or combinational) or are simply enhancements of existing algorithms. Commercial gate-level simulation tools can be used; however, as described above, this method requires realistic input vectors and is not fast enough for system-level designs. To our knowledge, the only available tool that are suitable are ACE-1.0 [10] and the activity estimator in Sis [7] (the estimator is called Sis-1.2 is this paper).

ACE-1.0 estimates P_1, P_s, and A_s for combinational and sequential gate-level circuits using probabilistic techniques. The static and switching probabilities are calculated using the techniques in [3], and switching activities are calculated using the Transition Density model [8] and the analytical low-pass filter model described in [14]. For circuits with sequential feedback, a simplistic iterative technique is used to update the switching probabilities at the output of flip-flops using the simplistic expressions P_1(Q) = P_1(D) and P_s(Q) = 2*P_1(D)*(1-P_1(D)) as described in [3]. ACE is fast but inaccurate for large and/or sequential circuits.

The activity estimator in Sis estimates P_1 and P_s, but does not estimate A_s. For circuits with sequential feedback, static and switching probabilities are calculated using the iterative solution described in [7]. The Sis activity estimator is only accurate for circuits that do not have a large glitching component. Furthermore, the estimator can become very slow for circuits with large BDD representations such as multipliers.

2.5 Power Model and Power-Aware CAD tools

The FPGA power model described in [10] and the power-aware FPGA CAD flow described in [15] are used to answer questions 1 and 2 from the introduction. The power model and the power-aware CAD flow are both built on top of the VPR CAD tool. VPR models FPGAs at a low-level, taking into account specific switch patterns, wire lengths, and transistor sizes. Once a circuit is routed onto the FPGA, VPR extracts the resistance and capacitance information for each net. The power model uses this resistance and capacitance information to estimate dynamic, short-circuit, and leakage power. Activity information needed for each wire in the circuit is read from file and thus can be obtained using any activity estimator. The power-aware CAD flow includes power-aware technology mapping, clustering, placement, and routing. Each power-aware algorithm minimizes power by reducing the capacitance of high-activity wires.

3. FPGA-SPECIFIC ACTIVITY ESTIMATION

This section examines the questions described in the introduction regarding the effect of switching activities on power models and power-aware CAD tools for FPGAs. This is important; the activity estimation techniques described in Section 2 are general since they do not target any specific implementation. Without using the activities in the context of FPGAs, it is impossible to determine with confidence which techniques are most suitable.

3.1 Accuracy and Speed

Before examining the effect of activities on power modeling and power-aware CAD tools, we begin by comparing the activities directly to highlight the strengths and weaknesses of the techniques employed in the ACE-1.0 and Sis-1.2 tools and to provide insight into the behavior of the power model and CAD tools when driven by these activities later on.

Figure 4 illustrates how the activities are obtained for each activity estimator. Twenty large MCNC benchmarks circuits (alu4, apex2, apex4, bigkey, clma, des, diffeq, elliptic, ex1010, ex5p, frisc, misex3, pdc, s298, s38417, seq, spla, tseng) and one ISCAS multiplier circuit (C6288) were used to empirically determine the speed and accuracy of the each activity estimation method. The multiplier was included since it is known to be a challenging circuit for activity estimation. For each circuit, static probabilities and switching probabilities are pseudo-randomly determined for each primary input of every circuit. A custom vector generator routine was then used to generate pseudo-random test vectors that matched these static and switching probabilities. All input vectors are first-order temporally correlated but not spatially correlated. The pseudo-random static and switching probabilities were then used to drive ACE-1.0 and Sis-1.2, and the corresponding vectors were used to drive the Verilog XL® gate-level simulator and circuit delay information was obtained using the VPR place and route tool.

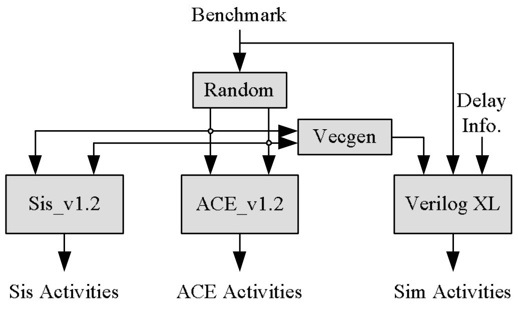


Figure 4: Activity comparison framework.

Three statistical measurements are used to examine accuracy. The first is *average relative error*, which is zero for perfectly accurate estimates. For each circuit, this measurement involves taking the average of the absolute of the relative error of every node within that circuit, as described in the expression below:

$$avg.rel.err.(circuit) = \frac{\sum_{n \in circuit} \left| \frac{A_{s_{estimated}}(n) - A_{s_{simulated}}(n)}{A_{s_{simulated}}(n)} \right|}{|circuit|} \qquad (1)$$

The second measurement is the *activity ratio*, which divides the sum of the estimated activities by the sum of the simulated activities, as described by the expression below:

$$activity\ ratio(circuit) = \frac{\sum_{n \in circuit} A_{s_{estimated}}(n)}{\sum_{n \in circuit} A_{s_{Simulated}}(n)} \qquad (2)$$

where an activity ratio of one is ideal. Finally, the third accuracy measurement is r^2 correlation, which measures the quality of a least squares fitting to the exact data, as described in the expression below:

$$r^2 = \frac{ss_{xy}^2}{ss_{xx} \cdot ss_{yy}} \qquad (3)$$

where ss_{xy}, ss_{xx}, and ss_{yy} are the sum of squared values of a set of n data points (x_i, y_i) about their respective mean [16]. For each circuit, n is the number of nodes in that circuit, x_i is the estimate activity of the i^{th} node in that circuit, and y_i is the simulated activity of the i^{th} node in that circuit. An r^2 of one indicates that there is a perfectly linear relationship between the estimated results and the simulated results; an r^2 of zero indicates no relationship.

Each measurement has a purpose. The *average relative error* gives an intuitive feel for how much error is occurring for individual nodes; however, the measurement does not give insight to the nature of the error. As an example, the *average relative error* is the same if an estimator tends to overestimate or underestimate activities by a factor of two. The *activity ratio*, on the other hand, does indicate if the estimator tends to over or under estimate activities but is not a good measure of accuracy since the error of one activity can cancel out the error of another activity. Finally, r^2 is useful for measuring the fidelity of the activities. In other words, it measures how well the estimator can determine if one node has a higher activity than another node. Intuitively, fidelity is important for power-aware CAD tools, which need to determine which wires to optimize.

Table 1 compares the accuracy and run-time of ACE-1.0 and Sis-1.2. Many observations can be made from the results. Firstly, the table shows that some results are missing for Sis-1.2 due to insufficient memory or extremely long execution time. This problem occurred in all but the smallest sequential circuit (dsip) but in only one of the combinational circuits (C6288), which is a multiplier. The likely origin of the problem is the use of BDDs, which tend to grow exponentially in size with the number of logic inputs. ACE-1.0 was able to produce results for all the circuits; however, the accuracy for sequential circuits is not acceptable for power-aware CAD tools or power models with a correlation of 0.47 and activity ratio of 2.37.

Table 1: Accuracy and speed of each activity estimator.

	Simulation		Sis-1.2		ACE-1.0	
	Comb	Seq	Comb	Seq	Comb	Seq
Activity Ratio	1.0	1.0	1.17	1.40	1.21	2.37
Avg. Rel. Error	0.0	0.0	0.02	0.09	0.07	0.08
R^2	1.0	1.0	0.88	0.72	0.89	0.47
Run-time (s)	122	191	-	-	31	108

Secondly, it can be seen that the accuracy of ACE-1.0 and Sis-1.2 are similar for the combinational circuits, with correlation close to 0.89 and average relative errors of 2% and 7%, respectively. These results are likely acceptable for power-aware CAD tools; however, the activity ratio indicates that both estimators tend to overestimate activities, which may translate to power overestimations.

Finally, the table shows the run-time of each estimator. The average Sis-1.2 run-times are not shown since it was not able to process many of the circuits, which makes the average meaningless. By inspection of the circuits that did pass, however, we observed that the tool is relatively efficient compared to simulation. ACE-1.0, on the other hand, is somewhat slower with average run-times that are approximately 4 and 2 times faster than simulation for combinational and sequential circuits, respectively.

3.2 FPGA Power Modeling

Comparing activities directly is useful since it reveals how accurate the activities are, if there are any trends such as over or underestimation, and if accuracy depends on circuit types. However, in order to determine the level of accuracy needed in the context of FPGAs, this subsection examines how the activities affect detailed power measurements of circuits implemented on FPGAs.

Figure 5 illustrates the experimental framework used to compare the effect of the activities on power estimates. The framework employs the none power-aware VPR tool to implement benchmarks circuits onto an FPGA and then uses the FPGA power model described in [10] to estimate the power of each implementation. All experiments target island-style FPGAs implemented in a 0.18μm TSMC CMOS process.

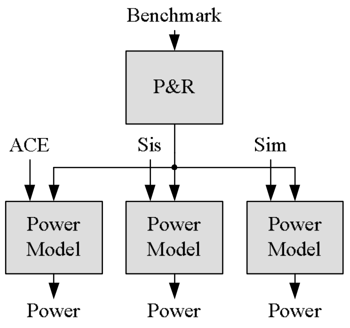


Figure 5: Power modeling framework.

Power estimates are obtained three times for the each benchmark circuit using activities produced using ACE-1.0, Sis-1.2, and gate-level simulation. Table 2 compares power estimates obtained using ACE-1.0 and Sis-1.2 to the power estimates obtained using simulation.

Table 2: Power using ACE, Sis, and Simulation.

Benchmark Type	Sim	Sis-1.2		ACE-1.0	
	Avg. Power	Avg. Power	Avg. |% Diff|	Avg. Power	Avg. |% Diff|
Comb.	1.18	1.39	19.8	1.38	18.6
Seq.	0.93	0.88	29.9	2.82	220

From the table, it is clear that both activity estimators tend to overestimate power as predicted by the *activity ratio* metric in Section 3.1. The most severe cases occur in the results obtained using ACE-1.0 for sequential circuits (by as much as 3X). This trend follows for the observation made in Section 3.1 regarding the overestimation of the switching activities. Clearly, another technique must be used in sequential circuits. The ACE-1.0 results suggest that filter function employed to reduce glitching is not accurate enough in this context. The Sis-1.2 results were higher than expected since Sis-1.2 estimates switching probabilities (ignores glitching) instead of switching activities. The most likely explanation why Sis-1.2 overestimates power is that the tool reads in static probabilities

(P_1) for primary inputs nodes and assumes that switching probabilities (P_s) are equal to $2*P_1*(1-P_1)$, which is the upper bound. The pseudo-randomly generated activities used in our framework chooses probabilities between 0 and $2*P_1*(1-P_1)$. Thus, on average, the Sis-1.2 activity estimator assumes higher activities at the primary inputs of each circuit, which lead to overestimation throughout the circuit.

3.3 Power-Aware CAD for FPGAs

This section examines how the accuracy of the various activity estimation techniques affects power-aware CAD tools that use switching activities to minimize power of circuits implemented on FPGAs. Intuitively, the most important characteristic of the activities being used to minimize power is fidelity since the tools must know which wire to optimize for power.

Figure 6 illustrates the framework used to compare the effect of the activities on the power-aware version of VPR described in Section 2, by implementing each benchmark three times using activities generated using ACE-1.0, Sis-1.2, and gate-level simulation. The power of all three implementations of the circuit is then estimated using the power model described in Section 2 and the activities obtained using simulation.

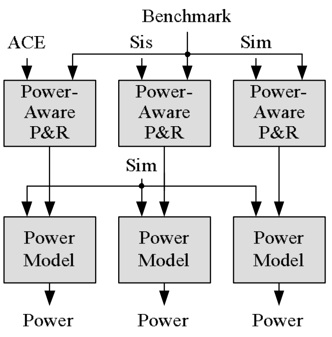


Figure 6: Power-aware CAD framework.

Table 3 reports the power savings obtained by the power-aware version of VPR using the three different activity estimators. Intuitively, the greatest power savings should be obtained when power-aware VPR is guided by the simulated activities since they are exactly the same as the activities used to estimate power.

Table 3: Power savings using each activity estimator.

Benchmark Type	Average Power Savings (%)		
	Simulation	Sis-1.2	ACE-1.0
Comb.	9.8	9.0	8.8
Seq.	13.6	-	5.3

On average, the power-aware CAD flow reduces power 9.8% for combinational circuits and 13.6% for sequential circuits when driven by the simulated activities. Although no average is available for Sis-1.2, careful inspection reveals power savings achieved using activities from Sis-1.2 and ACE-1.0 are approximately 9%, which is relatively good. This follows from the good correlation results measured in Section 3.1. For sequential circuits; however, the savings using ACE-1.0 activities are only 5.3% compared to 13.6%, which is not acceptable. These results again suggest that more accurate techniques must be employed to improve the performance of power-aware CAD tools.

4. ACE-2.0 : A NEW ACTIVITY ESTIMATOR

The previous section served to highlight the strength and weaknesses of existing activity estimation tools in the context of FPGAs. Specifically it found that the technique used in Sis-1.0 for circuits with sequential feedback does not scale well enough for larger circuits and the technique in ACE-1.0 was inaccurate, causing significant error in power estimates and degraded power savings from the power-aware CAD tool. The previous section also showed that the Transition Density model combined with a low-pass analytical filter were somewhat accurate but still caused power model to overestimate power.

Using this information, this section describes a new activity estimation tool called ACE-2.0, which addresses these weaknesses. Figure 7 outlines the ACE-2.0 algorithm, which has three phases. The first phase begins by simulating static and switching probabilities for logic within sequential feedback loops, if there are any. The second phase then employs the Lag-one model, described in [6], to calculate static and switching probabilities for the remaining logic that has not been simulated. Finally, the third phase calculates switching activities using a novel probabilistic technique that considers glitching. Each phase is described below.

```
ACE-2.0 (network, vectors, activities) {
  // Phase 1
  feedback_latch = find_feedback_latches (network);
  feedback_logic = find_feedback_logic (feedback_latches);
  if (vectors == NULL) vectors = gen_vectors (activities);
  simulate_probabilities (feedback_logic, vectors);

  // Phase 2
  foreach node n ∈ network
    if (Status(n) != SIMULATED) {
      bdd = get_partially_collapsed_and_pruned_bdd (n);
      Static_Prob(n) = calc_static_prob (bdd);
      Switch_Prob(n) = calc_switch_prob (bdd);
    }
  }
  // Phase 3
  foreach node n ∈ network {
    bdd = get_local_bdd(n);
    Switch_Act(n) = calc_switch_activity (bdd):
  }
}
```

Figure 7: ACE-2.0 pseudo-code.

4.1 Phase 1

This phase determines the static and switching probability for logic and flip-flops within sequential feedback loops in a circuit. The previous section demonstrated that sequential feedback is the greatest source of error and long execution times for existing tools. The existing probabilistic solutions proposed in the literature use BDDs or involve solving systems of non-linear equations. In either case, the techniques are not feasible for large circuits. Thus, the solution used within ACE-2.0 is to use a simplified form of simulation.

Two simplifications are made to improve the efficiency of the simulation. The first simplification is to simulate only the logic within sequential feedback loops. In most circuits with sequential feedback, the logic within feedback loops accounts for only a fraction of the circuit. Simulating this logic produces accurate probabilities within the feedback loops which can then be used to calculate probabilities for the remaining logic in the circuit.

The second simplification is to simulate switching probabilities instead of switching activities. In circuits with many levels of logic or with many exclusive-or gates, glitching accounts for a significant proportion of all transitions. As opposed to simulating activities, which involves processing each transition in each gate for every cycle, simulating probabilities only involves processing the final value of each gate for every cycle. The simulation routine used in ACE-2.0 is described in Figure 8.

```
simulate_probabilities (feedback_logic, vectors) {
  foreach vector ∈ vectors {
    update_primary_inputs (vector);
    evaluate_logic (feedback_logic);
    update_flip_flops ();
  }
}

evaluate_logic (logic) {
  foreach node n ∈ feedback_logic
    if (get_status_of_inputs (n) == NEW) {
      value = evaluate (n);
      if (value == 1) Num_ones(n)++;
      if (value != Value(n)) {
        Value(n) = value;
        Status(n) = NEW;
        Num_transitions(n)++;
      } else {
        Status(n) = OLD;
      }
    } else {
      Status(n) = OLD;
    }
  }
}
```

Figure 8: Simplified simulation pseudo-code.

Each cycle of the simulation begins by updating the values of the primary inputs with the next input vector. If vectors are not supplied, ACE-2.0 pseudo-randomly generates vectors which match the specified input activities. Once the input values are specified, the routine determines the output value of each gate in the feedback logic in topological order from the primary inputs to the primary outputs. Finally, at the end of each cycle, the routine updates the value at the output of each flip-flop based on the input value. To further improve performance, the evaluate logic routine only evaluates when at least one fanin of the gate has changed.

4.2 Phase 2

Although ACE-2.0 uses simulation to obtain static and switching probabilities for logic and flip-flops within sequential feedback loops, switching probabilities are also required for logic and flip-flops not within sequential feedback loops. These remaining probabilities are calculated using the Lag-one model [6], which produces exact switching probabilities (assuming that inputs are not correlated).

For a Boolean function f, the Lag-one model can be calculated by summing probabilities over all pairs of input states $\{x_i, x_j\}$ such that $f(x_i) = \overline{f}(x_j)$. Intuitively, you can think of an input state as a single row of the truth table representation of function f. Explicitly, the switching probability P_s can be calculated using the following expression:

$$P_s = \sum_{x_i \in X_1}\left[P_1(x_i)\cdot \sum_{x_j \in X_0} P_s(x_i,x_j)\right] + \sum_{x_i \in X_0}\left[P_1(x_i)\cdot \sum_{x_j \in X_1} P_s(x_i,x_j)\right] \qquad (4)$$

where X_1 is the set of input states such that $f(x_i) = 1\ \forall x_i \in X_1$, X_0 is the set of input states such that $f(x_j) = 0\ \forall x_j \in X_0$, $P(x_i)$ is the probability that the current input state is x_i, and $P(x_i, x_j)$ is the probability that the input state will be x_j at the end of a clock cycle given that the input state was x_i at the beginning of the clock cycle.

The probability that the current input state is x_i can be determined by taking the product of the static probabilities for each input literal, as expressed below:

$$P_1(x_i) = \prod_{k=1}^{n} P(f_k, x_i[k]) \ \Big|\ P(f_k, x_i[k]) = \begin{cases} P_1(f_k) & x_i[k]=1 \\ 1 - P_1(f_k) & x_i[k]=0 \end{cases} \qquad (5)$$

where n is the number of inputs that fan into function f, f_k is the function of the k^{th} input that fans into f, and $x_i[k]$ is the value of the k^{th} literal of input state x_i.

Similarly, the probability that the input state will change from x_i to x_j can be determined by taking the product of the switching probabilities for each input literal, as expressed below:

$$P_s(x_i,x_j) = \prod_{i=k}^{n} P(f_k,\, x_i[k],\, x_j[k]) \qquad (6)$$

$$P(f_k,\, x_i[k],\, x_j[k]) = \begin{cases} P_{0\rightarrow 1}(f_k) & x_i[k]=0,\, x_j[k]=1 \\ 1 - P_{0\rightarrow 1}(f_k) & x_i[k]=0,\, x_j[k]=0 \\ P_{1\rightarrow 0}(f_k) & x_i[k]=1,\, x_j[k]=0 \\ 1 - P_{1\rightarrow 0}(f_k) & x_i[k]=1,\, x_j[k]=1 \end{cases} \qquad (7)$$

where $P_{0\rightarrow 1}(f_k)$ is the probability that f_k will transition from 0 to 1 and $P_{1\rightarrow 0}(f_k)$ is the probability that f_k will transition from 0 to 1. These probabilities can be determined with the following expressions:

$$P_{0\rightarrow 1}(f_k) = \frac{P_s(f_k)}{2\cdot(1 - P_1(f_k))} \quad \text{and} \quad P_{1\rightarrow 0}(f_k) = \frac{P_s(f_k)}{2\cdot P_1(f_k)} \qquad (8)$$

The most efficient known implementation of the Lag-one model involves using a BDD. However, as observed in Section 3, using BDDs become infeasible for large circuits because of the exponential relationship between BDD size and the number of inputs. As a solution, ACE-2.0 combines the *partial collapsing* technique described in [17] with *BDD pruning* as an approximation to improve the speed of the calculation.

Partial Collapsing

The activity estimation techniques described in [6,7] suffer in that they require collapsing a large number of nodes. During activity estimation, each node is collapsed with all of its predecessors. Although this takes into account spatial correlation within the circuit, the technique is infeasible for large circuits. In [17], an alternative is proposed; rather than collapsing each node with all its predecessors, only smaller portions of the logic are collapsed, as shown in Figure 9. This results in smaller BDDs, leading to faster activity estimation time. The cost of doing this is that not all spatial correlation can be captured. There is a tradeoff between the amount of partial collapsing (the size of each collapsed node) and the error introduced by not taking into account all spatial correlation.

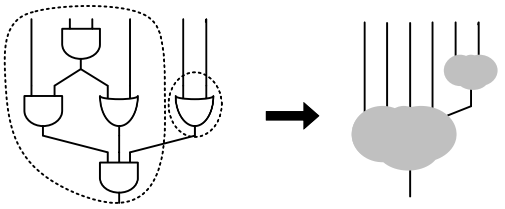


Figure 9: Example of a partially collapsed network.

The partial_collapse routine collapses nodes with only some (as apposed to all) of its predecessors such that the BDD representation of the collapsed logic contains at most *max_nodes* nodes. The *max_nodes* parameter can be used to tradeoff accuracy with run-time.

BDD Pruning

An alternative to partial collapsing is BDD pruning. By pruning low-probability branches of a BDD, the size of a BDD can be significantly reduced with minimal impact on the accuracy of the activity estimation. In logic synthesis applications (which also often use BDDs to represent logic nodes), such pruning is not possible since it would be unacceptable to change the behavior of the circuit. For activity estimation, however, the BDDs can be pruned since some degree of approximation is acceptable. Although pruning BDDs is not a new idea, pruning BDDs to improve the execution time of activity calculations is novel.

The proposed BDD pruning technique involves removing BDD branches with probabilities smaller than a pruning threshold probability, *min_prob*. An example is illustrated in Figure 10. In the example, the static probability of the function's four inputs are $P_1(a)=0.5$, $P_1(b)=0.9$, $P_1(c)=0.2$, and $P_1(d)=0.5$, and the probability of each branch is shown next to each node. The grey nodes are pruned from the BDD since their probabilities are smaller than *min_prob*=0.03.

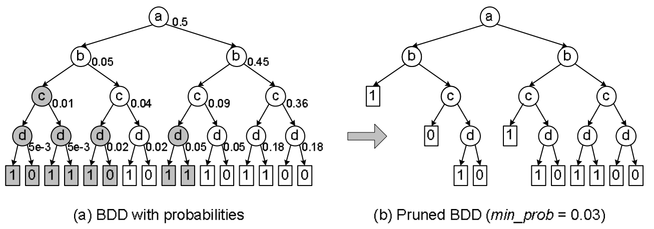

(a) BDD with probabilities
(b) Pruned BDD (*min_prob* = 0.03)

Figure 10: Example of BDD pruning.

After a branch is pruned, it must be replaced by either a '0' or a '1' terminal in order for the BDD to remain valid. A naïve approach is to arbitrarily use a '0' or a '1' terminal; however, this produces more error later on when the BDDs are used to calculate activities. A better approach is to replace the branch with a '0' terminal when the value of the branch being pruned is more likely to be '0' and a '1' terminal when the value is more likely to be '1'

Figure 11 describes the BDD pruning routine. The routine begins at the root node (top) and recursively traverses the BDD until it reaches the BDD terminals ('0' or '1' nodes) or BDD nodes with a branch probability less than the threshold value. When the probability is less than the threshold value, the branch is pruned away.

```
get_partially_collapsed_pruned_bdd (n, max_size, min_prob) {
    bdd = partially_collapse (n, max_size);  // see [17]
    pruned_bdd = bdd_prune (bdd, 1.0, min_prob);
    return (pruned_bdd);
}

bdd_prune (bdd, prob, min_prob) {
    if (bdd  is a '0' or '1' terminal) return (bdd);

    if (prob < min_prob) {
        if (Prob(bdd) < 0.5 )  {
            return ('0'); // replace branch with '0'
        } else {
            return ('1'); // replace branch with '1'
        }
    }
    n = literal (bdd);
    true_prob = prob · P1(n);
    false_prob = prob · (1.0 – P1(n));
    true_bdd = bdd_prune (bdd->true, true_prob, min_prob);
    false_bdd = bdd_prune (bdd->false, false_prob, min_prob);
    bdd ' = build_bdd (n, true_bdd, false_bdd);
    return (bdd');
}
```

Figure 11: BDD pruning pseudo-code.

The threshold probability parameter, *min_prob*, controls the tradeoff between accuracy and execution time. Increasing *min_prob* increases pruning and results in smaller BDDs, faster execution times, but reduced accuracy.

Combining Partial Collapsing and BDD Pruning

Partial collapsing and BDD pruning can be combined. In Figure 11, the BDD generated by the *partial_collapse* routine is then pruned using the *bdd_prune* routine. The resulting routine has two parameters: *max_size* and *min_prob*. Intuitively, the *max_size* parameter limits the BDD size of the collapsed node and the *min_prob* parameter controls the amount of pruning that occurs on the BDD representation of the collapsed node. A *max_size* of 125 and a *min_prob* of 0.004 were found empirically to produce good results.

4.3 Phase 3

The final phase of the ACE-2.0 algorithm addresses the issue of accurately and efficiently modeling the glitch component of switching activities. This subsection introduces the novel switching activity calculation that we used. The calculation is a simple generalization of the Lag-one model, yet performs well compared to existing techniques.

A transition at the output of a gate is normally caused by a transition occurring at a single input of that gate; however, transitions can also occur (or be canceled out) when two or more inputs transition at nearly the same time. Consider a two-input XOR gate with inputs A and B. If A is '0' and B transitions from '0' to '1', this transition will probably cause a transition at the output. Similarly, a second transition, this time of input 'A', will probably cause a second transition at the output. However, these two input transitions might not cause a transition at the output if they happen close enough together since the glitch generated by these two input transition may be filtered out by the resistance and capacitance of the gate. In other words, the amount of glitching that occurs depends on the minimum pulse width of the gate.

The new calculation adds the notion of minimum pulse width to the Lag-one model described in Section 4.2. Explicitly, the switching activity A_s is calculated using the following expressions:

$$A_s(f) = \frac{T}{\tau} \cdot P_s(f) \qquad (9)$$

$$P_{0\to1}(f_k) = \frac{P_s(f_k)}{2\cdot(1-P_1(f_k))} \cdot \frac{\tau}{T} \text{ and } P_{1\to0}(f_k) = \frac{P_s(f_k)}{2\cdot P_1(f_k)} \cdot \frac{\tau}{T} \quad (10)$$

where T is the maximum delay from the primary inputs to the output of the function f and τ is some period of time less than or equal to T. Intuitively, the calculation determines the switching probability during period τ, assuming that that input arrival times are normally distributed, and then multiplies this probability by T/τ, the number times that period τ occurs during T.

It is interesting to note that when $\tau = T$ the new switching activity model reduces to the Lag-one model since output transitions caused by any number of input transitions are equally weighted. Conversely, when $\tau \to 0$, the model reduces to the Transition Density model since only single input transitions carry any weight in the calculation. The best results are obtained when τ is set to approximately the physical delay of the gate.

4.4 ACE-2.0 Results

Table 4 summaries the results obtained using ACE-2.0. The activities are very accurate for both combinational and sequential circuits. The correlation for combinational circuits is 0.97, which is close to ideal. For sequential circuits, the correlation is 0.86, which is significantly better than ACE-1.0 with 0.47. Moreover, the activity ratio and average relative error are also close to ideal.

In terms of power estimates, the ACE-2.0 activities translated into very accurate power estimates, with less than 1% error compared to simulation. Similarly, the power-aware CAD achieved power savings that closely matches those achieved using simulated activities. Finally, the average run-times are 53 and 7.4 times faster than simulation for combinational and sequential circuits, respectively.

Table 4: ACE-2.0 results.

Circuit Type	ACE-2.0				Power Model	Power-Aware CAD
	Avg. R^2	Avg. Act. Rat.	Avg. Rel. Error	Avg. Run-time (s)	% Diff.	%Power Savings
Comb.	0.97	0.97	0.03	2.3	-0.1	8.8
Seq.	0.86	1.00	0.02	25.9	0.6	14.0

5. CONCLUSIONS AND FUTURE WORK

This paper examined various activities estimation techniques in order to determine which are most appropriate for use in the context of FPGAs. It found that existing probabilistic techniques were either too slow or too inaccurate for circuits with sequential feedback; causing inaccurate power estimations and poor power savings for power-aware CAD tools. It also found that using fully collapsed logic to calculate probabilities is not feasible for large circuits because of execution time. Finally, it found that calculating switching activities using the Transition Density and the associated low-pass filter caused the power model to overestimate power.

Given the above findings, a new activity estimation tool called ACE-2.0 that incorporates the techniques found most suitable was described. The new tool begins by calculating static and switching probabilities for every node in the circuit. For circuits with sequential feedback, a simplified simulation technique is used for the feedback logic and the Lag-one model is used for the remaining logic. To improve the speed of the Lag-one calculation with only a slight loss of accuracy, BDD sizes were reduced using partial collapsing and BDD pruning. Once the static and switching probabilities are obtained, ACE-2.0 employs a novel probabilistic-based technique to calculate the switching activities.

Finally, the new tool was validated in the context of FPGAs. Using activities estimated by ACE-2.0, power estimates and power savings were both within 1% of results obtained using simulated activities. Moreover, the new tool was 53 and 7.4 times faster than simulation for combinational and sequential circuits, respectively.

APPENDIX

Source code and instructions for downloading ACE-2.0 are available at http://www.ece.ubc.ca/~julienl/activity.htm.

REFERENCES

[1] Altera Corporation, Quartus II Handbook, Chapter : PowerPlay Power Analyzer, vol. 3, 2005.

[2] Xilinx, Inc., XPower: Online Documentation, http://www.xilinx.com/ise/power_tools/quick_start.htm, 2005.

[3] Q. Wu, M. Pedram, and X. Wu, A Note on the Relationship Between Signal Probability and Switching Activity, in Proc. Asia and South Pacific Design Automation Conf., pp. 117-120, 1997.

[4] C.Y. Tsui, M. Pedram, A.M Despain,, Efficient estimation of dynamic power consumption under a real delay model, in IEEE Intl. Conf. Computer-Aided Design (ICCAD), pp. 224-228, 1993.

[5] C.Y. Tsui, M. Pedram, A.M. Despain, Exact and approximate methods for calculating signal and transition probabilities in FSMs, in ACM/IEEE Design Automation Conference (DAC), pp. 18-23, 1994.

[6] R. Marculescu, D. Marculescu, M. Pedram, Switching Activity Analysis Considering Spatiotemporal Correlations, in the IEEE Intl. Conf. Computer-Aided Design (ICCAD), pp. 294-299, 1994.

[7] J. Monteiro, S. Devadas, A methodology for efficient estimation of switching activity in sequential logic circuits, in ACM/IEEE Design Automation Conference (DAC), pp. 12-17, 1994.

[8] F. Najm, Transition density: A new measure of activity in digital circuits, in IEEE Trans. Computer-Aided Design, vol 12, no. 2, pp. 310-323, 1993.

[9] V. Betz, J. Rose, A. Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers, 1999.

[10] K.K.W Poon, S.J.E Wilton, A detailed power model for field-programmable gate arrays, ACM Transactions on Design Automation of Electronic Systems (TODAES), April 2005, Vol. 10, Issue 2, 2005, pp. 279-302.

[11] R. Burch, F. Najm, P. Yang, T. Trick, A Monte Carlo approach to power estimation, in IEEE Trans. on VLSI Systems., vol. 1, no. 1, pp. 63-71, 1993.

[12] J.N. Kozhaya, F. Najm, Accurate power estimation for large sequential circuits, in IEEE Intl. Conf. Computer-Aided Design (ICCAD), pp. 488-493, 1997.

[13] P.H. Schneider, S. Krishnamoorthy, Effects of Correlation on Accuracy of Power Analysis – An Experimental Study, ACM/IEEE Intl. Symp. of Low Power Electronics and Design (ISLPED), pp. 113-116, 1996.

[14] F. Najm, Low-pass filter for computing the transition density in digital circuits, in IEEE Trans. Computer-Aided Design, vol. 13, no. 9, pp. 1123-1131, 1994.

[15] J. Lamoureux, S.J.E. Wilton, On the Interaction Between Power-Aware Computer-Aided Design Algorithms for Field-Programmable Gate Arrays, Journal of Low Power Electronics (JOLPE), Vol. 1, No. 2, pp. 119-132(14), Aug. 2005.

[16] Mathworld, http://mathworld.wolfram.com/Correlation Coefficient.html, 2006.

[17] B. Kapoor, Improving the Accuracy of Circuit Activity Measurement, ACM Design Automation Conference, pp. 734-739, 1994.