Accurate High Level Datapath Power Estimation *

James E. Crenshaw and Majid Sarrafzadeh

Department of Electrical and Computer Engineering
Northwestern University, Evanston, IL 60208

Abstract

The cubic switching table, is a new data structure for estimating datapath switching at a high level. It is constructed during behavioral simulation, and is used to estimate the switching for any particular datapath schedule and binding. Time to extract the estimate from the table is independent of the original simulation size. For n operations in the RTL description, it takes $O(n^3)$ time to perform the extraction. We show that an exact switching table would require exponential size, but experimental results show that the cubic table is accurate, with typical error under 5%.

1 Introduction

The requirements of portable electronics and high speed computing are driving increased interest in reducing power consumption in digital circuits. This design goal can be as important as area reduction or speed. Although many techniques exist for lowering power given a RTL model or a netlist, there is greater potential for power reduction before the RTL model has been written.

The cubic switching table allows us to exploit this fact by capturing information during behavioral simulation so that power for the same simulation on a given RTL datapath model can be estimated quickly.

The input for the general problem is a behavioral model impelementing some algorithm. We wish to characterize the power requirements of the behavioral model for the purpose of generating a low power RTL model for the algorithm. Thus in this paper we will use the cubic switching table to generate power estimates for particular RTL implementations. We will show that these estimates are very accurate (less than 6% average error), and we also show that square switching tables have unacceptably high errors (up to 44%).

1.1 CMOS Current Drain

In CMOS integrated circuits, current drain can be traced to three components. The primary cause is charging and discharging capacitative elements when nodes switch logic values. Secondarily, short-circuit current results when a temporary path from V_{DD} to Ground exists when the output of a logic gate switches. Also, a negligible leakage current is a natural part of the steady state of a CMOS circuit.

Short-circuit current can be reduced to less than 20% of total power by good circuit design, so it is generally not considered in high level power models. Similarly, leakage current is ignored [10].

Power derived from switching can be calculated from the equation $\frac{1}{2T}\sum_{i\in nodes} V_{DD}^2 s_i c_i$ where s_i is the number of times node i switches during time T; c_i is the capacitance of node i, and V_{DD} is the supply voltage.

Switching at nodes in a circuit is determined by the current state resulting from previous inputs and the next input.

1.2 Behavioral Level Modeling

The tasks involved in getting from a behavioral level model to an RTL level model are resource allocation, scheduling and resource binding.

In resource allocation, a set of functional units capable of executing all operations in the HDL model is selected from a library. Each member of the set represents one physical unit which may be instantiated in a netlist.

Most designs target a specific technology, and in many cases standard cell libraries are available in advance of the design. So even at the behavioral level, models for functional units can be very complete, and can account for glitching within the units.

Scheduling takes into account the allocation and determines at what time each operation will be performed. After scheduling, we know which inputs will be sent when, but not on which functional unit, so switching cannot be calculated directly by simulation.

Resource binding maps operations in the HDL model onto allocated resources. After resource binding and scheduling, an RTL model has been created

*Research supported in part by Motorola University Partnerships in Research and NSF grant number MIP-9527389

ED&TC '97 on CD-ROM

and clearly switching in any functional unit can be calculated.

2 Previous Work

Previous work in behavioral level power estimation has been targeted primarily at DSP applications. One such effort, described in [9], proposes a statistics based method where the power of each operation type is modeled based on past circuits. The work described in [1] and [7] seeks to reduce voltage in order to reduce power, but neither addresses the impact of signal correlations on switching. In [4] a number of behavioral level transformations and techniques are described which may be useful to reduce power, but no general high-level power estimation technique is given. In [2] an algorithm is presented which exploits signal correlation to minimize switching in registers.

Another approach to estimation was described as part of a larger synthesis system in [11]. The use of a (square) switching matrix was detailed for use in an iterative based synthesis system. The same authors improved the switching matrix in [6] and formulated an ILP solution to the binding problem. Next, they introduced a new iterative synthesis system using the improved switching matrix in [5]. Our empirical results demonstrate that the square switching matrix may introduce unacceptably high error when applied to general CDFGs rather than the simpler DSP derived DFGs shown in the papers just mentioned.

3 Definitions

A switching table is a matrix associated with a single resource type (e.g. CLA adder, ALU, bus, etc). From the HDL model we have a set of operations, $S_r = \{op_1, op_2, ..., op_k\}$ which can be performed on resource r. Each cell (i, B, j) of the matrix corresponds to the switching resulting if operation, op_i, is scheduled before op_j, with operations in the blocking set $B = b_1, \ldots, b_k$ scheduled in between. B can be any set of operations which may be scheduled on r between b_1 and b_k.

We must have a model of switching for each resource type. This can range from a simple piecewise linear model such as the DBT of [8] to a complex simulation capable of modeling glitching within the resource. Note that inputs to the resource are always presented simultaneously, and therefore the global delay model is always zero delay.

Once a table has been constructed for each resource type, the switching on each resource for any particular schedule and binding can be evaluated in $O(n^2)$ time. From the schedule and binding, each physical functional unit is associated with a total ordering of operations $op_1, op_2, ..., op_l$. Switching within the unit can be found by summing over the cells corresponding to (op_i, op_j) for $i, j <= l$ with the appropriate blocking set in between each pair.

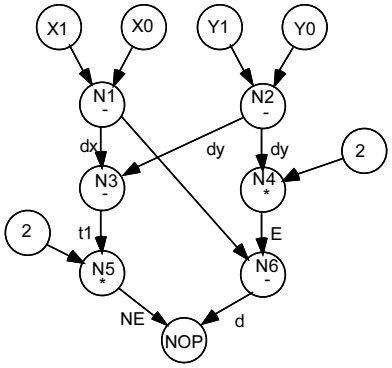

Figure 1: DFG for Line Drawing Fragment

Since the model is likely to contain loops, we must also allow for inter-iteration switching. This is done by adding another table for each loop called an inter-loop table. We refer to the collective set of tables as "the switching table".

3.1 A Switching Table Example

Consider the following fragment of Bresenham's line-rasterization algorithm, and its corresponding DFG, shown in 1.

```
dx = x1 - x0;
dy = y1 - y0;
d = 2*dy - dx;
E = 2*dy;
NE = 2*(dy-dx);
```

If the inputs shown below on the left are applied, the results are in the middle column. The value of each variable is shown in binary on the right. A temporary variable, t1, has been introduced to carry the value computed by the operation with DFG label 3.

```
x1 = 3        dx = 3                 0011
x0 = 0        dy = 2                 0010
y1 = 2        d = 1                  0001
y0 = 0        E = 4                  0100
              t1 = dy - dx = -1      1111
              NE = 2*t1 = -2         1110
```

If we use two buses, a multiplier and a subtracter to schedule the CDFG for speed without regard for power, then we might find the schedule and binding shown in 2. The bus accesses and netlist for this solution are shown in 3.

	CS1	CS2	CS3	CS4	CS5
BUS1			Val 2	t1	d
BUS2		dx	dy	E	NE
MUL			N4	N5	
SUB	N1	N2	N3	N6	

Figure 2: Schedule and Binding for Line Drawing Fragment

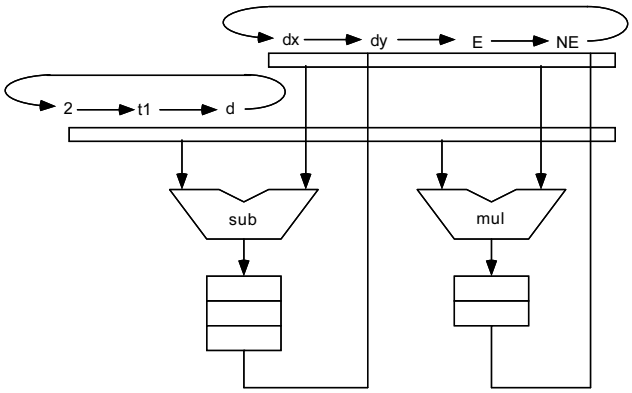

Figure 3: Block Diagram of Datapath for Line Drawing Fragment

Consider the order of events on the buses with respect to our previous example. Placing value 2 immediately before t1 causes a four bit bus to go from value 0010 to value 1111. This means three bits switch values. Similarly, going from t1 (1111) to d (0001) causes three bits to change. On BUS2, transitions dx to dy to E to NE take 1, 2, and 2 bits respectively. In total for this example, 11 bits change. But if d and NE are swapped, then BUS1 sees t1 (1111) to d (1110) taking one transition, and BUS2 sees E (0100) to d (0001) which is two transitions. Thus, this single swap saves almost 20% of power consumed. Similarly, swapping dx with value 2 realizes a savings of two more transitions.

This observation leads to the questions of how to formalize this estimation process and whether it will lead to an algorithm for selecting the best sequence for all buses. To answer the first question, between each pair of values to go on the bus, there is an associated

	2	dx	dy	d	E	t1	NE
2	-	1	0	2	2	3	2
dx	1	-	1	1	3	2	3
dy	0	1	-	1	2	3	2
d	-	-	-	-	-	3	4
E	-	2	-	2	-	3	2
t1	3	-	-	3	3	-	1
NE	-	-	-	3	2	-	-

Figure 4: Switching Matrix for Line Drawing Fragment

number of transitions representing the power cost if that particular pair is scheduled consecutively on the same bus. Considering the cost of all feasible pairs in our example, we have the table shown in 4.

From the table, we can seek a better ordering of bus accesses, or evaluate any arbitrary RTL solution without resimulating.

4 Exact Switching Tables

The purpose of a switching table is to allow fast evaluation of a particular RTL implementation of some behavioral model. We assume that the RTL is based directly on the behavioral model — that is, it is the same CDFG as in the behavioral model with a specific schedule and binding. We wish to determine, for any pair of nodes (a, b) scheduled on the same function unit, how much switching occurs in that unit during the full simulation as a result of b following immediately after a.

In the example of the previous section, we saw that when there are no conditional nodes in the HDL model, we can use a square table to capture the switching. On the other hand, consider the following code, and suppose that in some RTL solution, a, b, c are calculated in that order on the same adder.

```
while (portA > 0)
    a = x1 + x2;
    if (portB <> 0) then b = x3 + x4;
    c = x5 + x6;
endwhile;
```

Then the square table is insufficient, since a and c may be calculated one after the other on the adder when *portB* is nonzero.

In the cubic table, we find the switching in the adder of this example by adding $switching(a, \emptyset, b) + switching(b, \emptyset, c) + switching(a, b, c)$.

In general, to have a lookup table capable of giving the exact intra loop switching for a pair of nodes, (a, b), based only on a particular schedule it would be necessary to have a matrix indexed as $M(a, S, b)$ where S is a binary number representing one of the sets of conditional nodes possible to schedule between a and b.

To see that this is indeed necessary, consider the following pseudocode.

```
while (true)
    a = b + c
    d = e + f
    if (port1)
        k1 = p1 + q1
    if (port2)
        k2 = p2 + q2
...
    if (portn)
        kn = pn + qn
endwhile
```

There are no data dependencies, so we are free to schedule any of the conditional nodes between operation $b + c$ and operation $e + f$. Since the conditional nodes all depend directly on ports, any of the 2^n possible combinations of conditional nodes can be simulated. The following theorem can be shown. We omit the proof due to space limits.

Theorem 4.1 *A lookup table capable of giving the exact intra loop switching for a pair of nodes, based only on a particular schedule, has at least* $\Omega(2^n)$ *cells.*

Intuitively, the size of the table is related to the number of conditional nodes which can be scheduled between any two non-conditional nodes. It can be shown that if the size of such strings of conditional nodes is restricted to k, then a table of size $O(n^{2+k})$ is sufficient to evaluate switching in a functional unit for a particular schedule and binding. This result is stated without proof in the following theorem.

Theorem 4.2 *A set of lookup tables capable of giving the exact intra-loop and inter-loop switching for a pair of nodes, based only on a particular schedule and binding, where the schedule is guaranteed to allow no more than k conditional nodes to be scheduled one after the other on any functional unit, can be accomplished with* $O(n^{2+k})$ *cells.*

Fortunately, $k = 1$ is sufficient to get very good results and this will be shown in the remainder of the paper. However, we also show that $k = 0$ is not sufficient for good estimates.

5 Cubic Switching Table

In the previous section, we saw that we had to keep track of all sets of nodes capable of allowing switching between two nodes. But the only time an entry $M(a, S, b)$ is nonzero is if at least one iteration during the simulation exited with none of the nodes in set S executing. That is, the conditions enabling the nodes in S were all false during that iteration.

Intuitively, as the size of set S grows, the likelihood of $M(a, S, b)$ being nonzero is lower, and even if it is nonzero, we expect it to be of decreasing significance. And yet the table M is exponential in the size of S. So for most cases, we are probably keeping track of some useless information.

This leads us to a simple heuristic method for expanding the table. Instead of a third dimension of size $O(2^n)$, we introduce a third dimension of size $O(n^k)$ for user specified k where this dimension consists of the $\sum_{i=0}^{k} \binom{n}{k}$ possible blocking sets of size k between a and b.

As it turns out, it is generally to sufficient choose $k = 1$, to create an $O(n^3)$ size table which we call the cubic switching table. This has been implemented, and it will be shown in the experimental results section of this paper that this table works very well in practice. We will also show that $k = 0$ results in unacceptably large error.

Keeping a partial blocking set complicates evaluation, since two nodes may be separated by several different blocking sets. The question of which one to include is not obvious. For instance, we certainly would want the min of the blocking sets to be an upper bound on the switching, but, consider a case statement which blocks pairs split by it every time it executes. If we simply take the min, we will get some non-zero value rather than the correct zero switching.

Such problems are inevitable since we are only keeping a small fraction of the information necessary to compute exact switching. Nevertheless, the min function seems to work well in practice.

5.1 Cubic Switching Table Definition

In more formal terms, we have a cubic switching matrix for each resource r, which is a data object containing one table, M_r for all intra-iteration switching and for each while loop k, we have inter-iteration table N_{rk}.

A cell (i, b, j) in M_r represents the switching incurred when operation i is scheduled before operation j with only b in between on resource r.

A cell (i, b, j) in N_{rk} represents the switching incurred when operation j is scheduled first in loop k on resource r and cell i is scheduled afterward, with b in

between.

We use the notation $(i, \emptyset, j)$ to indicate the cell where i is scheduled immediately before j.

5.2 Cubic Table Calculation During Simulation

To calculate the intra-iteration switching matrix for resource r, during simulation, after each iteration i of each loop k, the simulation is paused, and all pairs of operations (a, c) simulated in the iteration which are computable on resource r are simulated in sequence on the switching model for the resource. Several cases arise which must be handled with additional data structures.

Before detailing the cases, let us define several functions.

$lastSeen(op)$ is the last set of inputs simulated for operation op.

$exit(l, op)$ is the last set of inputs simulated at loop l for operation op.

$entry(l, op)$ is the first set of inputs simulated in the current iteration of loop l for operation op.

Case 1 In the simplest case, neither a nor b is in any loop nested within loop k. Then $M(a,b,c) = M(a,b,c) + switching(lastSeen(a), lastSeen(c))$ iff b did not occur, where the function $switching$ is determined by the resource switching model.

Case 2 If a is in a loop l, nested within k, but c is not in any inner loop, then since a is taken to precede c in the schedule, clearly $exit(l, a) = lastSeen(a)$ so the same equation as in Case 1 still holds.

Case 3 On the other hand, if a is not in an inner loop, but c is in nested loop l, then we have that $M(a,b,c) = M(a,b,c) + switching(lastSeen(a), entry(k+1, c))$ if b didn't occur in between.

Case 4 If a is in inner loop l and c is in inner loop m, then if a and c share the same parent loop p within loop k, the switching for this case will have been computed during the iterations of p. But if the parent loop of a and c is k, then switching is computed by $M(a,b,c) = M(a,b,c) + switching(exit(k+1, a), entry(k, c)$ and since $exit(k+1, a) = lastSeen(a)$, we can use the same equation as found in Case 3.

Calculations for intra-iteration switching tables N_k are as follows.

Recall that $N_k(a,b,c)$ holds the inter-iteration switching for loop k if a is operation scheduled later in k and c is earlier. Thus we will always use the $prevExit(k, a)$ value for a whereas for c we use either $lastSeen(c)$, if c is not in an inner loop with respect to k or otherwise we use $entry(k+1, c)$.

5.3 Switching Evaluation Using the Cubic Switching Table

A particular schedule and binding can be represented by a table as shown in 2. Each row represents a physical functional unit, and each column is a timestep so that the value v_{ut} in each cell (u, t) is an operation to be executed on physical unit u during timestep t. If loop lengths are variable then column labels may also be relative.

To calculate the switching on each function unit, we have the following equation:

$$S = Intra + Inter$$

$$= \sum_{t=0}^{n} min_{b \in between}(M(v_{ut}, b, v_{u(t+1)}))$$

$$+ \sum_{k \in loops} min_{b \in sameparentloop}(N_k(last_{op\_k}, b, first_{op\_k}))$$

6 Experimental Results

For three examples, a line rasterizer, heapsort and a line clipping algorithm, we show three schedule and binding solutions with one, two and three buses. We consider only the bus resource and switching derived from communications on the buses. Each of the three examples has at least one inner loop and conditional statements.

Each algorithm was translated into the simple HDL used by our system. An accompanying vector file with 100 simulations for each algorithm was generated randomly (restricted so that the vectors make sense in the algorithmic context). The programs were run on a Sparc 10 workstation. Output of the models was verified against standalone direct implementations of each model. Actual switching was calculated using the direct implementations.

In each of the results tables, square table switching is presented along with cubic table switching. The square table is included for comparison because it is based on work described in [11] and [5]. The error produced by the square table estimate is shown to climb over 40%, while the cubic table is always under 6% error.

Switching is broken down into inter vs intra loop iteration switching to illustrate where the potential for error is greatest in the square table. Total switching for the cubic table includes the square table since it is just the set of cells described by $(a, \emptyset, b)$.

6.1 Example: Bresenham's Line Rasterizer

TABLE 1
BRESENHAM'S LINE RASTERIZATION ALGORITHM
Cubic rows are the sum of square and unblocked switching. Square error column shows previous work and cubic error shows accuracy of our approach.

		intra loop	inter loop	total	actual	square error	cubic error
1 Bus	square	208797	101051	309848			
	unblocked	0	47358	47358	345100	10.2	3.5
	cubic	208797	148409	357206			
2 Bus	square	65922	122183	188105			
	unblocked	101539	53334	154873	336824	44.1	1.8
	cubic	167461	175517	342978			
3 Bus	square	44792	143078	187870			
	unblocked	101899	41466	143365	335880	44.1	1.4
	cubic	146691	184544	331235			
average						32.8	2.2

The first example is Bresenham's line drawing algorithm.

The result of simulation of the model for three particular schedule and binding solutions (one each of one bus, two bus, and three bus solutions) was compared to results predicted by the square table and the cubic table. The exact-switching simulation was at the RTL level with exact-switching calculated for bus accesses as they occurred. The HDL model was simulated in the system and the particular schedule and binding was then evaluated according to the square table, and the cubic table. Table 1 shows percent error as calculated by $\frac{|RTLSwitching - tablePrediction|}{RTLSwitching}$.

Table 1 shows that the square table has a significant error compared to the cubic data structure for the bresenham example. This is because the square table doesn't recognize the significant contribution of inter-iteration switching between the last node in the inner loop with the middle node in the inner loop, whereas the cubic table checks for the possibility, and correctly adds the contribution.

Previous related work [11] described a square switching table, which was essentially the cells with empty blocking sets. So we can calculate the values predicted by such a $k = 0$ approach by using the base of the cubic table. The rows labelled square show those results. The rest of the cubic table is used to find switching between nonadjacent nodes which are unblocked. This is the amount shown in the unblocked rows. Thus, the total switching predicted by the cubic table is the sum of square switching plus ublocked switching.

The intra loop column shows intra loop iteration switching and the inter loop column shows inter iteration switching. The actual column shows the switching from a real switch level simulation. The square error column shows the error resulting from a square table estimation, and the cubic error is the error observed when our approach is used.

Note the variation in unblocked switching from implementation to implementation. This is precisely the portion of switching difficult to detect using a square table.

6.2 Example: Heapsort

The second example is a heapsort algorithm.

6.3 Example: 2D Clipping

The third example is a 2D clipping algorithm. It has many nested if then else statements, making it a challenging example.

7 Conclusions

We have developed a new way of characterizing switching at the behavioral level for use in driving low-power high level synthesis which has a reasonable space requirement but which is also efficient and accurate to within 6% in practice.

Limitations were established for an existing method which may have error up to 44%, and exact switching for the general class of HDL models was shown to be beyond the capability of the switching table approach.

The techniques described were implemented and their capabilities were demonstrated in the experimental results section.

References

[1] Chandrakasan, A., Potkonjak, M., Mehra, R., Rabaey, J., Brodersen, R., "Optimizing Power Using Transformations", *IEEE Transactions on CAD*, vol 14, 1995.

TABLE 2
HEAPSORT ALGORITHM
Cubic rows are the sum of square and unblocked switching. Square error column shows previous work and cubic error shows accuracy of our approach.

		intra loop	inter loop	total	actual	square error	cubic error
1 Bus	square	43530	5634	49164			
	unblocked	9718	501	10219	59133	16.9	0.4
	cubic	53248	6135	59383			
2 Bus	square	35876	9687	45563			
	unblocked	6407	333	6740	53562	14.9	2.3
	cubic	42283	10020	52303			
3 Bus	square	31972	15433	47405			
	unblocked	6146	301	6447	54569	13.1	1.8
	cubic	38118	15734	53582			
average						15.0	1.5

TABLE 3
2D CLIPPING ALGORITHM
Cubic rows are the sum of square and unblocked switching. Square error column shows previous work and cubic error shows accuracy of our approach.

		intra loop	inter loop	total	actual	square error	cubic error
1 Bus	square	11477	17	11494			
	unblocked	3130	0	3130	14284	19.5	2.4
	cubic	14607	17	14624			
2 Bus	square	7489	54	7543			
	unblocked	5393	351	5744	13484	44.1	1.5
	cubic	12882	405	13287			
3 Bus	square	6015	69	6084			
	unblocked	9723	874	10597	15803	61.5	5.6
	cubic	15738	943	16681			
average						41.7	3.2

[2] Chang, J., and Pedram, M., "Register Allocation and Binding for Low Power", *Proceedings of the ACM/IEEE Design Automation Conference*, 1995.

[3] Dasgupta, A., and Karri, R,. "Simultaneous Scheduling and Binding for Power Minimization During Microarchitecture Synthesis", *Proceedings of the International Symposium on Low-Power Design*, "1995".

[4] Musoll, E. and Cortadella, J., "High-Level Synthesis Techniques for Reducing the Activity of Functional Units", *Proceedings of the International Symposium on Low-Power Design*, 1995.

[5] Raghunathan, A., and Jha, N., "An Iterative Improvement Algorithm for Low Power Data Path Synthesis", *Proceedings of the ICCAD* 1995.

[6] Raghunathan, A., and Jha, N., "An ILP Formulation for Low Power Based on Minimizing Switched Capacitance During Data Path Allocation", *Proceedings of ISCAS*, 1995.

[7] Raje, S., and Sarrafzadeh, M., "Variable Voltage Scheduling", *Proceedings of the International Symposium on Low Power Design*, 1995.

[8] Landman, P., and Rabaey, J., "Black-Box Capacitance Models for Architectural Power Analysis", *International Workshop on Low Power Design*, 1994.

[9] Mehra, R., and Rabaey, J., "Behavioral Level Power Estimation and Exploration", *Proceedings of the International Workshop on Low-Power Design*, 1994.

[10] Pedram, M., and Rabaey, J., "Design Solutions and Challenges for Low Power Systems", ICCAD Tutorial #2, 1994.

[11] Raghunathan, A., and Jha, N., "Behavioral Synthesis for Low Power", *Proceedings of the ICCD*, 1994.