Low-Power Driven Scheduling and Binding[1]

Jim Crenshaw and Majid Sarrafzadeh

Electrical and Computer Engineering Department
Northwestern University
Evanston, Illinois
crenshaw@ece.nwu.edu, majid@ece.nwu.edu

Abstract

We investigate the problem of exploiting signal correlation between operations to find a schedule and binding which minimizes switching. We propose several heuristics to solve the problem. Experimentally, we give an algorithm for scheduling communications on a bus, which reduces bus switching up to 60%, without increasing the number of cycles required for the schedule. Low-power scheduling efforts in the literature have focused on decreasing the number of cycles in the schedule so that the voltage required to run the resulting circuit can be lowered. However, the number of voltages supplied to a chip is likely to be limited, so among the processes to be implemented, typically only a few will determine the minimum voltages, and the rest will have slack in their schedules. Therefore it is interesting to inquire about the impact of scheduling which does not reduce the number of time steps in order to decrease switching. In this paper, we show that power-aware scheduling can lead to significant decreases in switching, often without an increase in the number of time steps required. The technique is general, and can be used to schedule operations on any kind of resources.

1 Introduction

An increasing number of Design Automation tools are taking power into account as a solution metric along with the traditional goals of high speed and low area. This trend is driven primarily by the demand for battery-driven, wireless devices, so less energy is required to complete a given computational task. In addition, low-power design techniques often reduce the heat produced by a circuit, so major cost benefits can be realized by less expensive packaging requirements.

To address the power issue as effectively as possible, it is important to take it into consideration as early in the design cycle as possible. In this paper, we show that signal correlation information can be effectively captured during behavioral simulation in such a way that it can be used to decrease switching in the resultant circuit by up to 60%, without increasing the number of cycles required for the schedule. Thus, this technique can be applied in addition to popular voltage reduction methods.

The rest of the paper is organized as follows. In section 2, previous work is discussed. Section 3 covers background information and assumptions in our switching model. Next, the low-power scheduling problem is explored in section 4. Experimental results given in section 5 show that heuristic methods can result in significant improvements in power. Finally, brief conclusions are drawn in section 6.

2 Previous Work

Interest in low-power design automation has resulted in a significant body of work at many levels of design abstraction. General survey papers covering circuit level to architectural and behavioral level include [1, 2].

For high-level synthesis, a number of techniques have been demonstrated. One of the most influential papers, [3], shows the usefulness of increased parallelism to allow voltage reduction for the same computational throughput. This work has influenced several research teams, such as [4] who show how to use slack to avoid unnecessary computation, and [5]who show how a DFG might be partitioned for multiple voltages. Other papers on modeling functional units for behavioral level power estimation have been written, such as [6].

[1] Research Supported in part by Motorola University Partnerships in Research and NSF grant number MIP-9527389.

0-8186-8409-7/98 $10.00 © 1998 IEEE

The voltage idea was combined with an iterative improvement approach using a square switching matrix as the basis for a signal correlation metric in [7]. The same authors had earlier published an iterative algorithm designed to reduce switched capacitance also based on the square switching matrix, in [8]. Neither of these papers investigated the effects of scheduling changes alone. In the latter, capacitance estimates are included to drive synthesis, and in the former, the voltage reduction is key. Another switching matrix based approach used a simulated annealing technique in [9], but the square table was suitable only for scheduling DFGs without conditional statements, and the annealing itself did not produce outstanding results.

Other signal correlation techniques, based on the dual bit type method are suggested in [10, 11], and exploitation of regularity in [12]. In [13], switched capacitance is reduced by partitioning the CDFG into groups with minimized inter-group communication.

Other techniques include [14], which attempts to improve sleep-mode for memories, and for control synthesis, [15] among others has presented a method. In the remainder of this paper, we investigate the impact of scheduling general CDFGs without voltage reduction to reduce switching.

3 Background

This section describes the CMOS power model which will be used in the rest of the paper. The power derived from switching can be calculated from the equation $P = \frac{1}{2T}\sum_{i \in nodes} V_{DD}^2 s_i c_i$ 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.

In general, the switching of a node in a circuit occurs when some primary input to the circuit changes. In a non-idealized circuit model, propagation delays and re-convergent fanout can cause a node to switch more than once for each change of primary inputs. After all inputs to a combinational circuit have been steady for a sufficient time, all switching ceases, and the circuit settles into a state determined entirely by the inputs. Now, after a new input has been applied, the internal nodes may switch and after some time, the circuit comes to a new state. It is thus easy to see that the switching in a combinational circuit is not determined by the current inputs alone, but by the previous inputs in conjunction with the current inputs.

Information about possible input pairs can be captured during behavioral simulation using a data structure known as a switching table. Such tables can be used to account for switching between sequentially ordered operations, conditional operations and operations which are sequential with respect to two iterations of a loop, and have been shown to be very accurate, [16], typically to within about 6%.

The table has three dimensions, each of which corresponds to atomic operation nodes in the sequencing graph. The first axis represents the previous operation, the second axis the current operation, and the third accounts for a single conditional operation which is scheduled in between. By convention, the zeroth element of the third axis corresponds to the case where no node scheduled in between the other two.

At the end of each loop iteration during behavioral simulation, all appropriate node triples are enumerated for operations which executed during the iteration, and the switching associated with each is added to the switching table. Once the simulation is complete, the table may be used to quickly evaluate particular schedule and binding solutions, accounting for the entire simulation, but taking time independent of the simulation length.

As an example of how switching tables are constructed, consider the following HDL code.

```
while (true)
    a = b + c;
    while (port p ==1 )
          d = e + f;
    endwhile
    g = 2 * d;
    h = 5 + g;
endwhile
```

Suppose this code is simulated for two iterations of the outer loop. In the first iteration, suppose port p is 1 for one time step (so d is calculated once); and in the second outer iteration, p is 0. Suppose further that b, c, e and f are inputs, and their values are 15,14,21,11 for the first iteration, and 14, 5, *don't care, don't care*, for the second. Then the operations of a behavioral simulation are shown in Figure 1. In the first column, the first outer loop iteration is shown. Since p is 1 for exactly one inner iteration, a, d, g and h are calculated in order.

Below the calculations in the figure, the two levels of the cubic switching table are shown. Since there is a level for each potential blocking operation, plus one level for no blocking, and *d* is the only potential blocking node, we have two levels. For the first iteration of the outer loop, since *d* executes, for each cell in the *k=null* level, switching is calculated as if the row operation were scheduled before the column operation, and the result of the switching function is added to the cell (here we suppose a function s returns the zero delay switching for an adder). In the *k=d* level, since *d* executed, no switching will be added to any cell (because cell *x, y* in that level means the switching incurred by *x* following *y* if *d* is scheduled in between). Now, for the second iteration of the outer loop, the figure shows that d is not calculated, so switching for the triple *a,d,h* and *h,d,a* is added to the table in the *k = d* level, and the *k = null* level is calculated as before.

4 *Scheduling and Binding*

In this paper we investigate the impact of signal correlation on switching. To do this we wish to evaluate the switching resulting from power oblivious schedule and bindings and compare that to schedule and bindings that seek to place highly correlated operations sequentially on the same resource. We show that a very restricted case is NP-hard, and therefore, heuristic methods must be applied. Fortunately, we are able to demonstrate significant savings in switching without an optimal algorithm.

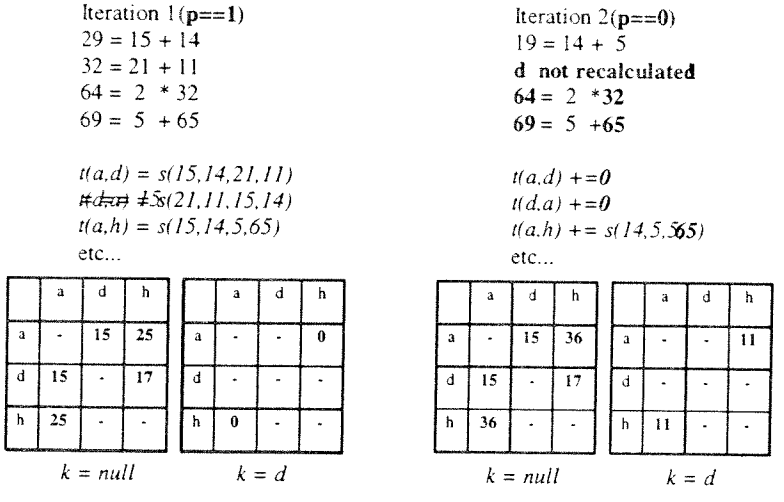

Figure 1 Cubic Switching Table Example: Calculating the table during simulation.

4.1 Single-resource low-power DFG scheduling and binding is NP-hard

It is obvious that minimum time scheduling on a single resource is a simple problem. Similarly, minimum area solutions for unbounded time are found trivially. And when the dependency partial order is empty, optimal area and time solutions can be found. But the low-power scheduling and binding problem is difficult, even for the case where only one resource is available, and no dependencies are present in the DFG.

To develop an intuition for this problem, consider the switching table for a DFG with no dependencies. Since there are no conditional nodes, the table has only two dimensions; it is symmetric about the diagonal and there are values in each cell except those on the diagonal. The table can therefore be interpreted as a graph with edge weights equal to the cell values.

Now consider the task of sequencing the operations on a single resource. This amounts to finding a minimum weight path through every node of the graph, which is almost the formulation of the Traveling Salesman Problem. The only difference is in the edge weights. For TSP, and even TSP with triangle inequality property, more freedom is allowed in the choice of edge weights than we have.

As an illustration of a possible edge weight calculation, suppose the required task is to sequence data words from an n-word wide on-chip DRAM output onto a one word bus. Since the data is presented all at once, there are no order constraints. So, if it is known that certain of the words may be correlated, this fact might be exploited to reduce switching on the bus. In fact, by simulation, we can derive the relationship of switching between the words and store that information in a switching table. Thus, the weight of a path through the corresponding graph can be calculated in *O(n)* time, and the value is the amount of

switching for the entire simulation due to the chosen sequence.

In this example, the weight of edge *(u, v)* is the sum of the hamming distance of the value of word u and word v over all iterations in the simulation. That this obeys the triangle inequality property is easy to show, but it is also more restricted. First, the size of the simulation must be assumed polynomial in the size of the sequencing graph. Also, since the switching metric is hamming distance, the graph can be thought of as a subset of the nodes of a word-size times simulation-length dimensional hypercube, with the edge weights given by the distance between the nodes in the hypercube. So we must show that the problem is difficult under such conditions.

Theorem 1. *The single-resource, no-dependency, minimum-switching scheduling problem is NP-hard.*

Proof: That the decision version of this problem is NP-complete can be shown by reduction from Hamiltonian Path. We omit the proof here due to space considerations.❑

This result shows that finding optimum low-switching schedules is a difficult task, and justifies the use of heuristic techniques. As will be shown in the remainder of the paper, despite the complexity of the problem, heuristic scheduling algorithms can significantly improve switching.

4.2 Power Oblivious Scheduling and Binding

As a benchmark against which to compare results, a power oblivious scheduling algorithm is used. The basis for it is the As Soon As Possible method. Since the primary goal of this algorithm is to reduce the number of cycles required to schedule a sequencing graph, it is consistent with design methodologies which seek to reduce required voltage.

This algorithm performed quite well at reducing the number of time steps, and was often able to generate a minimum time schedule. Note that there are several degrees of freedom for low power scheduling. First, position p is simply the first available, so clearly other positions may have the same time value but reduce switching. Also, scheduling the complex nodes first means that they cannot take into account nodes which may ultimately be scheduled before them. Finally, the ready queue does not need to be ordered, so any ready node could be scheduled first.

Lemma : *The time complexity of the power oblivious scheduling algorithm is $O(|V|^2)$ where V is the set of nodes in the sequencing graph.*

Proof: This follows from the fact that topological sort is $O(|V| + |E|)$ and placing each node may take up to $O(|V|)$ time to find a vacant position in the schedule❑

4.3 Switching table based evaluation

Consider a set of nodes scheduled in the following order, $n_1 n_2 \ldots n_k$. Then clearly a lower bound on switching in the model is obtained by summing $\sum_{i=1}^{i=k-1} cell(n_i, n_{i+1})$. If no conditional nodes are present in the model, then this provides exact zero-delay switching.

To handle conditional nodes, the cubic table augments the square table estimate by including data for switching from node n_i to node n_{i+j} where $j > 1$. To compute this, the cubic table

For iterations where *d* executed, schedule is

	Cy1	Cy2	Cy3	Cy4
Add	a	d	-	h
Mult	-	-	g	-

adder's switching is
t(a,null,d) + *t(d,null,h)* = 32
15 + 17 =

And when *d* doesn't execute, we have:

	Cy1	Cy2	Cy3	Cy4
Add	a	-	-	h
Mult	-	-	g	-

adder's switching is
t(a,d,h) = 11

Final *k=null* Level

	a	d	h
a	-	15	36
d	15	-	17
h	36	-	-

Final *k=d* Level

	a	d	h
a	-	-	11
d	-	-	-
h	11	-	-

Figure 2 Cubic table evaluation example shows how the two parts of a to d switching are summed.

cells *(a,S,b)* are used, where *S* is a node scheduled between *a* and *b*. Note that if *S* is not a conditional node then *cell(a,S,b)=0*. To get the switching for *j>2* we compute switching $s(n_i, n_{i+j}) = \min_{i+1<l<i+j-1} cell(n_i, n_l, n_j)$. Thus to the basic square table switching, the following sum is added, $\sum_{i=1}^{i=k-1} s(n_i, n_{i+1})$.

The example of Figure 1 is continued in Figure 2, showing the use of the cubic table.

In the figure, note that the square table alone does not capture switching from a to h properly, since d is in between. By using the k = d level, exact a to h switching is available. For the case where multiple conditional nodes fall in between, the table cannot give exact switching, however it has been shown empirically that taking the min over all the related cubic table cells gives a good estimate.

These evaluations can be performed while the schedule is being created, and can be used to evaluate possible schedules. The time taken to evaluate a schedule is clearly $O(|V|^3)$. The next section discusses several variants.

4.4 Greedy Low-Power Scheduling and Binding Algorithm

Three different power aware scheduling heuristics were added to the basic ASAP algorithm. The first requires the least modification to the original algorithm, consisting only of evaluating ties while placing the sorted nodes. The second performs the recursive scheduling of sub-sequencing graphs while placing the sorted nodes. In the final variant, several nodes are removed from the topological sort's FIFO queue at once, and are placed according to the best switching.

ASAP can be modified to break ties among possible equal-time positions in a greedy fashion. The outline of the algorithm is identical to the power oblivious version, except when nodes are placed. In the previous version, a node was placed in the first available position. Here, all equal-time positions are considered by performing a cubic-table switching evaluation, and selecting the best one. It was found empirically that this method often reduced power, but not a great deal.

The problem with this method is that each complex node starts out oblivious of what might be scheduled before it, so early nodes in each subgraph are not scheduled wisely. Furthermore, the number of equal-time positions available to a complex node is likely to be limited, so finding the best tie-breaker is not a significant improvement.

Lemma : *Time complexity of the greedy scheduler is $O(|R||V|^4)$.*

Proof: For each node, the schedule is evaluated for up to |R| positions, where R is the set of resources available.❑

4.5 Hybrid Low-Power Scheduling and Binding Algorithm

To improve on the greedy method, two techniques were implemented and combined serially. The first method involves scheduling complex nodes as they appear in the topological sort, and the second method can take more than one operation node from the ready queue, and thereby considers more scheduling options simultaneously.

The first improvement in the hybrid algorithm consists of allowing subgraphs to be scheduled with information about nodes scheduled before them. The power improvement from this method is quite significant, and shows up to 45% reduction of switching without adding more time steps than the power oblivious algorithm. However, in some cases only a small reduction was achieved. To improve on such tough cases, additional freedom is required for the scheduler.

Since the ready queue may have several nodes in it at once, it makes sense to consider the scheduling of those nodes simultaneously. This amounts to scheduling nodes which have no data dependencies across all available resources. However, even though this is the simplest possible form of the low-power scheduling problem, we have already seen that for the single-resource case, the problem is NP-Hard. Since the number of nodes available in the queue is typically small, we chose to test all possible schedules for up to k nodes at a time, where k is a supplied parameter.

To keep the time complexity for this variation practical, the size of *k* must be small. For practical purposes, *k* greater than 4 or 5 is unreasonable, and for this paper, k was set to 3. Since each possible configuration is evaluated, the time complexity *is* $O(((k|R|)!/(k|R|-k)!)|V|^3 + |V|^4)$.

The results from this variation are generally good. However, the added freedom results in an

increase in the number of time steps required. This is fine as long as there is slack in the schedule, but in general no such slack may be available. To address this, the hybrid scheduler considers the results of each of its parts, and selects the quicker one if the two switching values are within 5% of each other, otherwise it takes the lower switching value. In practice, this parameter would be changed or replaced by a criterion dependent on slack in the schedule.

Lemma : *The time complexity for the hybrid algorithm is* $O(((k|R|)!/(k|R|-k)!)|V|^3 + |V|^4)$ *where k is a parameter, and* $|R|$ *is the number of resources.*

Proof: The time complexity of the better subgraph scheduler is the same as the greedy scheduler, and is therefore dominated by the k at a time scheduler.❑

Although this algorithm has a worst case asymptotic time complexity which is fairly significant, the actual running time is much better because the evaluations are never over all n nodes.

5 Experimental Results

We implemented all four of the proposed scheduling algorithms listed in the previous section. Each algorithm was run on five implementations of each of three behavioral models. Switching was measured for communications over buses, and the algorithms described in the models were Bresenham's line rasterizer, 2D clipping, and heapsort. Each implementation was tested with 100 sets of input data generated randomly but in such a way as to make sense for the given problem.

5.1 Experiment Framework

During behavioral simulation, switching tables are generated. Afterward, several scheduling algorithms can be applied, and the resulting schedule simulated, with switching summed according to the implicit RTL description. The platform is written in Java, and all experiments were run on a 150MHz Pentium using the Symantec Just-In-Time bytecode compiler.

This process was applied forty-five times to generate the data summarized in Table 1. Each scheduler was applied to each HDL model with five different bus allocations. The results show that scheduling for signal correlation can provide a significant improvement in power over designs which ignore such information.

	Hybrid	Greedy	Oblivious
bres 1	345826	345826	345826
bres 2	271457	326305	371757
bres 3	133182	218422	378102
bres 4	134683	175652	377340
bres 5	134631	175512	377340
clip 1	14222	14222	14222
clip 2	13012	15118	14530
clip 3	11396	15536	15277
clip 4	10820	14219	12718
clip 5	10780	12705	14324
heap 1	53655	53655	53655
heap 2	41888	53312	52340
heap 3	41675	52755	52821
heap 4	35230	48231	50237
heap 5	35919	50412	52002

Table 1 Absolute Switching Counts

The first column shows each model's name with the number of buses in the particular implementation. So clip 5 is the 2D clipping algorithm model, with 5 buses allocated for scheduling communications. The next three columns show the raw switching counts for each of the power oblivious, greedy, and hybrid approaches respectively.

For most of the examples, improved switching does not come at the price of increased schedule time. The final number of control steps required for each schedule is shown in Table 2. The increased schedules (relative to the power oblivious version) occur

	Hybrid	Greedy	Oblivious
bres 1	14	14	14
bres 2	11	7	7
bres 3	5	5	5
bres 4	4	4	4
bres 5	4	4	4
clip 1	33	33	33
clip 2	27	22	22
clip 3	27	18	18
clip 4	26	17	17
clip 5	17	17	17
heap 1	27	27	27
heap 2	25	19	19
heap 3	22	17	17
heap 4	17	17	17
heap 5	17	17	17

Table 2 Time steps resulting from application of schedulers.

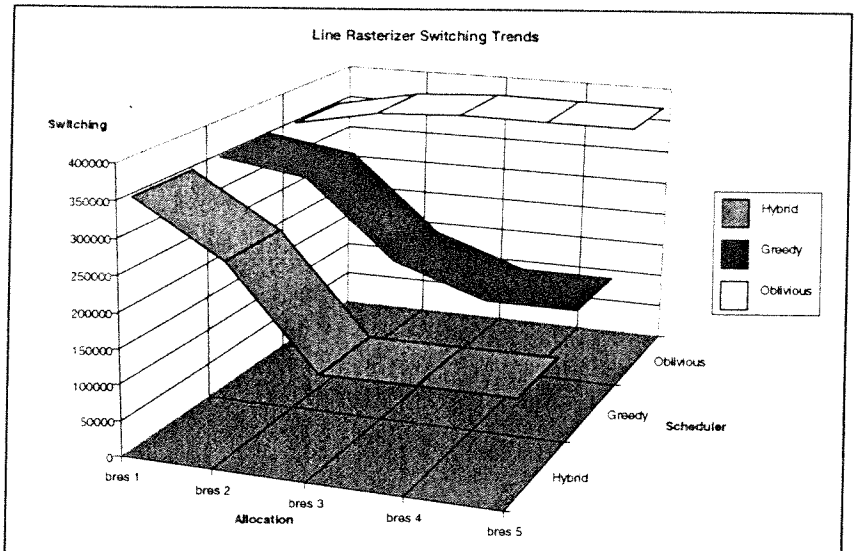

Figure 3 Bresenham's Line Rasterizer Switching Trends. As more buses are allocated, low-power schedulers can find solutions with decreased switching.

only when the hybrid algorithm chose the add-k method over the better subgraph scheduler.

5.2 Bresenham's Line Rasterizer

The line rasterizer algorithm has only 15 operations, and most of the simulation time is spent in a tight loop with only 7 operations total. Because of the high correlation between some of the operations, it is relatively easy to create a good low-switching schedule. In Figure 3, the trends of improvement with more resources, and from oblivious to best algorithm can be seen. Each ribbon shows how the scheduler does as more resources are added for scheduling communications. The oblivious scheduler actually does slightly worse as more buses are added, whereas the hybrid algorithm finds a solution over 60% lower. Even the greedy algorithm works well.

5.3 2D Clipping

The 2D clipping algorithm is a particularly hard example because it consists of many nested conditional statements, and no loops. Thus, while broad generalizations about positive and negative values may allow some switching optimization, in general when the data is drawn from a uniform distribution, it is not easy to find good solutions. Nevertheless, the hybrid scheduler managed to find solutions which are 14%-25% better than the oblivious scheduler. These results are shown graphically in Figure 4.

Note that the greedy algorithm is unable to beat the power oblivious algorithm in several instances. This is due to poor choices made by the greedy algorithm during the complex node scheduling phase, and lucky decisions by the oblivious algorithm.

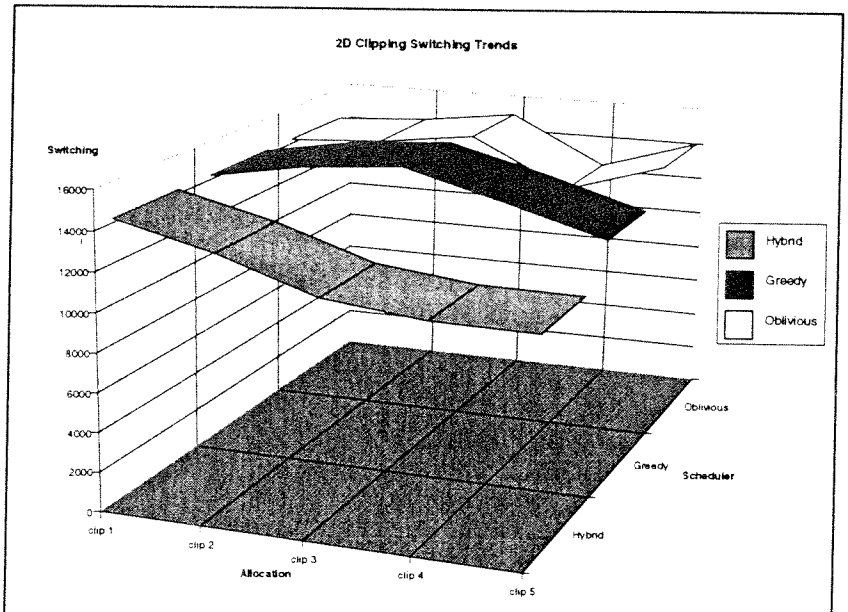

Figure 4 2D Clipping Algorithm Switching Trends. As more buses are available, hybrid algorithm consistently beats oblivious.

5.4 Heapsort Algorithm

The heapsort algorithm consists of several loops, as well as conditional statements, so it represents a fairly typical general algorithm. Again, the hybrid algorithm is best. The oblivious trend is almost flat, and the greedy heuristic barely beats it, but the hybrid manages to do 19%-30% after an extra resource is available.

6 Conclusion

We have shown that optimum switching scheduling is a difficult problem, but that heuristic efforts to reduce switching by exploiting signal correlation can improve switching significantly. This conclusion is based on experiments involving three very different algorithms, all with conditional operations, scheduled over five different allocations.

As future work, it would be interesting to find an algorithm that uses the cubic switching table to find lower bounds on the amount of switching required for a given algorithm over various allocations.

Bibliography

[1] S. Devadas and S. Malik, "A Survey of Optimization Techniques Targeting Low Power VLSI Circuits," presented at Design Automation Conference, 1995.

[2] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. on Design Automation of Electronic Systems*, vol. 1, pp. 3-56, 1996.

[3] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing Power Using Transformations," *IEEE Transactions on CAD*, vol. 14, pp. 12-31, 1995.

[4] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar, "Scheduling Techniques to Enable Power Management," presented at Design Automation Conference, 1996.

[5] S. Raje and M. Sarrafzadeh, "Variable Voltage Scheduling," presented at International Symposium on Low Power Design, 1995.

[6] S. Gupta and F. Najm, "Power Macromodeling for High Level Synthesis Power Estimation," presented at ACM/IEEE Design Automation Conference, 1997.

[7] A. Raghunathan and N. Jha, "An Iterative Improvement Algorithm for Low Power Data Path Synthesis," presented at International Conference on Computer Aided Design, 1995.

[8] A. Raghunathan and N. Jha, "Behavioral Synthesis for Low Power," presented at ICCD, 1994.

[9] A. Dasgupta and R. Karri, "Simultaneous Scheduling and Binding for Power Minimization During Microarchitecture Synthesis," presented at International Symposium on Low-Power Design, Dana Point, California, 1995.

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

[11] P. Landman and J. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method," *IEEE Trans. on VLSI Systems*, vol. 3, pp. 173-187, 1995.

[12] R. Mehra and J. Rabaey, "Exploiting Regularity for Low-Power Design," presented at ICCAD, 1996.

[13] R. Mehra, L. Guerra, and J. Rabaey, "Low Power Architectural Synthesis and the Impact of Exploiting Locality," *Journal of VLSI Signal Processing*, vol. 13, pp. 239-258, 1996.

[14] A. Farrahi, G. Tellez, and M. Sarrafzadeh, "Memory Segmentation to Exploit Sleep Mode Operation," presented at Design Automation Conference, 1995.

[15] L. Benini and G. DeMicheli, "Automatic Synthesis of Low-Power Gated-Clock Finite State Machines," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 630-643, 1996.

[16] J. Crenshaw and M. Sarrafzadeh, "Accurate High Level Datapath Power Estimation," presented at European Design and Test Conference, 1997.

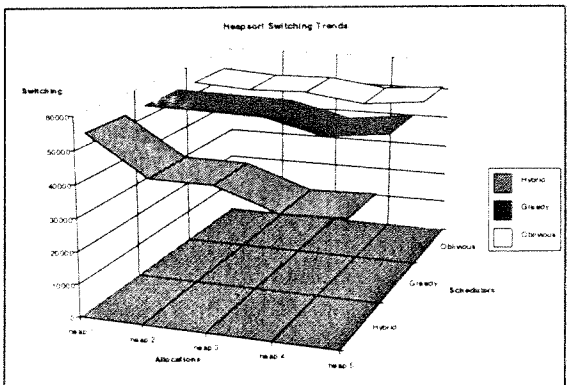

Figure 5 Heapsort Switching Trends are shown. The hybrid algorithm is up to 30% better than the power oblivious scheduler.