Scheduling Techniques to Enable Power Management

José Monteiro, Srinivas Devadas
Department of EECS, MIT
Cambridge, MA

Pranav Ashar
C&C Research Labs., NEC USA
Princeton, NJ

Ashutosh Mauskar
Synopsys, Inc.
Mountain View, CA

"Shut-down" techniques are effective in reducing the power dissipation of logic circuits. Recently, methods have been developed that identify conditions under which the output of a module in a logic circuit is not used for a given clock cycle. When these conditions are met, input latches for that module are disabled, thus eliminating any switching activity and power dissipation. In this paper, we introduce these power management techniques in behavioral synthesis. We present a scheduling algorithm which maximizes the "shut-down" period of execution units in a system. Given a throughput constraint and the number of execution units available, the algorithm first schedules operations that generate controlling signals and activates only those modules whose result is eventually used. We present results which show that this scheduling technique can save up to 40% in power dissipation.

I. Introduction

Rapid increases in chip complexity, increasingly faster clocks, and the proliferation of portable devices have combined to make power dissipation an important design parameter. The power dissipated by a digital system determines its heat dissipation as well as battery life. Power reduction techniques have been proposed at all levels – from system to device.

It has been demonstrated at the gate and system levels that large power savings are possible merely by cutting down on wasted power – commonly referred to as power management. At the system level, this involves shutting down blocks of hardware that are not being used ([3], Chapter 10). Detection and shut down of unused hardware is done automatically in current generations of Pentium and PowerPC processors. The Fujitsu SPARClite processor provides software controls for shutting down hardware.

It has been shown in recent work that similar power management techniques are effective at the sequential [1] and combinational logic [6], [9] levels also. Application of power management at the gate level involves first identifying large portions of the circuit that frequently produce information that is either not essential for determining the values on the primary outputs, or information that could have been produced by much simpler hardware. Additional hardware is then added to the circuit that detects on a per-clock-cycle basis input conditions under which such a situation arises and shuts down the corresponding portions of the circuit for that clock cycle.

The goal of our work is to introduce power management into scheduling algorithms used in behavioral level synthesis. Behavioral synthesis comprises of the sequence of steps by means of which an algorithmic specification is translated into hardware. These steps involve breaking down the algorithm into primitive operations, and associating each operation with the time interval in which it will be executed (called operation scheduling) and the hardware functional block that will execute it (called hardware allocation). Clock-period constraints, throughput constraints, hardware resource constraints and their combination make this a non-trivial optimization problem.

Decisions taken during behavioral synthesis have a far reaching impact on the power dissipation of the resulting hardware. For example, throughput-improvement by exploiting concurrency via transformations like pipelining and loop unrolling enables the hardware to be operated at lower clock frequencies, and thereby at lower voltages [2]. The lower supply voltage leads to a reduction in power dissipation. Hardware allocation also has an effect on the switching activity and thereby on the power dissipation, for example see [8].

Our work is centered around the observation that scheduling has a significant impact on the potential for power savings via power management. Based on this observation, we present a scheduling algorithm that is power-management-aware, i.e., it generates a schedule that maximizes the potential for power management in the resulting hardware. The proposed algorithm operates under a user determined combination of throughput, cycle-time and hardware resource constraints. Starting from a Silage [4] description, our implementation of the algorithm generates VHDL code for the controller as well as the datapath corresponding to the power-management-aware schedule. Validation of power reduction is done via the Synopsys power estimation tool.

II. Power Management

A. Data-dependent Power Shut Down

A system-level approach is to identify idle periods for entire modules and turn off the clock lines for these modules for the duration of the idle periods ([3], Chapter 10). At the logic level, two effective shut down techniques have been proposed recently, *precomputation* [1], [6] and *guarded evaluation* [9].

In *precomputation*, a simple combinational circuit (the precomputation logic) is added to the original circuit. Under certain input conditions, the precomputation logic disables the loading of all or a subset of input flip-flops and computes some or all of the circuit outputs by itself. Under these input conditions, no power is dissipated in the portions of the original circuit with only disabled flip-flops as inputs. *Sequential precomputation* shuts down an entire cone of logic feeding a primary output using existing flip-flop boundaries. *Guarded evaluation* and *combinational precomputation* take that approach to a finer grain by identifying cones internal to the circuit that can be shut down under certain input conditions. In the process, new transition barriers (guards) in the form of additional latches or OR/AND gates are created.

These methods can be called *Data-dependent Power Shut Down* methods since the shutting down of logic is decided on a per clock cycle basis given an input vector.

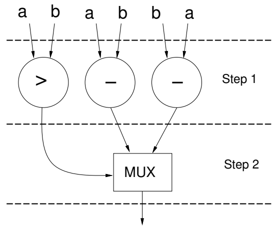

Fig. 1. CDFG and Schedule for $|a - b|$ Using 2 Control Steps.

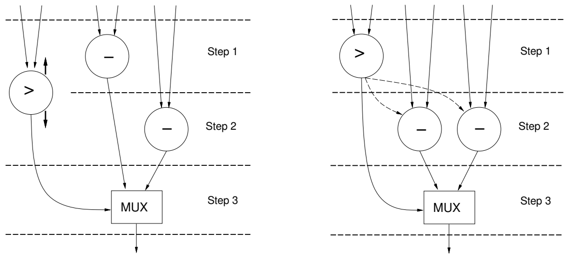

Fig. 2. Schedule for $|a - b|$ Using 3 Control Steps.

B. Scheduling for Power Management

The techniques of Section II-A are limited by the predefined logical structure of the circuit. In this paper, we move one step upwards in the CAD synthesis pipeline. We propose an algorithm that schedules operations so as to minimize the amount of unused computation.

In a typical design, the flow of data is determined at run time based on conditions derived from input values. As an example, say we need to compute $|a - b|$. One way to implement it is to do the comparison $a > b$ and if the result of this operation is true we compute $a - b$ otherwise we compute $b - a$. The Control Data Flow Graph (CDFG) for this simple example is shown in Figure 1. Assume that one control step is required for each of the three operations ($-$, $>$ and MUX). The only precedence constraint for this example is that the multiplexor operation can only be scheduled after all other three operations. Existing scheduling algorithms use this flexibility to minimize the number of execution units needed and/or the number of control steps. If we are allowed two control steps to compute $|a - b|$, then necessarily the operations $a > b$, $a - b$ and $b - a$ (we need two subtractors) have to be executed in the first control step and the multiplexor in the second control step as indicated in Figure 1. If instead we are allowed three control steps, we can get by with one subtractor and schedule operations $a - b$ and $b - a$ in different control steps, one in the first control step and the other in the second. Operation $a > b$ can be scheduled in any of these two control steps and the multiplexor will be in the third control step, as shown in Figure 2(a).

In either case, *both* $a - b$ and $b - a$ are computed although only the result of one of them is eventually used. This is obviously wasteful in terms of power consumption.

We propose a scheduling algorithm that attempts to assign operations involved in determining the data flow (in this case $a > b$) as early as possible in the initial control steps, thus indicating which computational units are needed to obtain the final result. Only those units that eventually get used are activated. The algorithm chooses a schedule only if the required throughput and hardware constraints are met. In other words, the algorithm explores any available slack to obtain a power manageable architecture.

For our example, and assuming we have available three control steps, our scheduling algorithm will assign $a > b$ to the first control step and $a - b$ and $b - a$ to the second. Depending on the result of $a > b$, only the inputs to one of $a - b$ and $b - a$ will be loaded, thus no switching activity will occur in the subtractor whose result is not going to be used. This situation is shown in Figure 2(b), where the dashed arrows indicate that the execution of the '$-$' operations depends on the result of the comparator. Here we assumed we have two subtractors available. If that is not the case, we need to assign one subtract to the first control step and another to the second, the operation in the first control step will always be computed, but we can still disable the one in the second control step when it is not needed.

If only two control steps are allowed, there is no flexibility. The solution is unique (Figure 1) and our scheduling algorithm will produce the same result as the traditional method: no power management is possible.

C. Relationship with Past Research on Identifying Mutually Exclusive Operations

Two operations are said to be mutually exclusive if the result of only one of them will be used, whatever the input. The mutual exclusiveness of two identical operations can be exploited to schedule them in the same control step and to make them share the same resource. With this in mind, a few algorithms have been proposed in the past to identify mutual exclusiveness efficiently (see [5] for a review).

Our application of mutual exclusiveness for power management is more general in that we also exploit mutual exclusiveness of two operations that are not identical to each other. Even so, we can leverage off the previous work on algorithms for identification of mutually exclusive operations.

III. Scheduling Algorithm

Given a behavioral description of the system, our objective is to schedule the operations such that operations whose result goes through some conditional branch (such as an `if` or `case` statement) are only activated if the condition for their use is met. We want to maximize the number of operations whose control signals (the signal that selects the usage of their result) are computed before they are scheduled. The pseudo-code for an algorithm that does such an optimization is shown in Figure 3. This algorithm was implemented within the HYPER framework [7]. HYPER routines were used for the parsing of the high-level description language (Silage [4]), final scheduling and VHDL output generation.

In step 1, the behavioral description of the system is converted into a Control Data Flow Graph (CDFG), where each node corresponds to an operation. This process creates all precedence conditions among operations. Also the "As Soon As Possible" (ASAP) and "As Late As Possible" (ALAP) values for each node are computed. These values indicate the earliest and latest control step a given node can be scheduled in. In other words, they represent the slack of a node for the specified throughput. After this parsing, all conditionals in the system will correspond to multiplexor nodes. Our goal is to schedule nodes in the transitive fanin of the control input of each multiplexor before nodes in the transitive fanin of inputs 0 and 1, and do so for as many nodes as possible.

The algorithm looks at each multiplexor individually and starts with those multiplexors closer to the outputs (or farther from the inputs).

```
1.   Generate CDFG ;
2.   For each multiplexor mux {
3.        Annotate nodes in fanin of the 0, 1 and control inputs of mux ;
4.        Compute new ASAP of each node in the fanin of the 0 and 1 inputs ;
5.        Compute new ALAP of each node in the fanin of the control input ;
6.        If for any node ASAP > ALAP
7.             then power management not possible for mux ;
8.             else assign new ASAP and ALAP values to nodes ;
9.   }
10.  Create control edges between last node in the control fanin and top nodes
          in 0 and 1 fanin of muxes for which power management is possible ;
11.  Execute Hyper scheduling ;
12.  Generate final Datapath and Controller circuits ;
```

Fig. 3. Pseudo-code for Power Management Scheduling Algorithm.

The reason for this is that if we are able to do power management on a multiplexor closer to the outputs then we will be able to shut down a larger number of operations in the circuit.

For each multiplexor, the algorithm identifies which nodes are in the transitive fanin of each input (step 3). If a node is both in the fanin cone of the 0 and 1 inputs of the multiplexor then no power management is attempted since the operation is needed no matter what the result of the condition is. The same applies for nodes that fanout to other nodes besides the current multiplexor.

In step 4, new ASAP values are computed for the nodes in the 0,1-input fanin assuming they are scheduled after the last node in the control input fanin of this multiplexor. Similarly, in step 5 new values ALAP values are computed for the nodes in the control input fanin assuming they are scheduled before the first node on either the 0 or 1 input fanin.

If at any point any node is assigned an ASAP value greater than the value for ALAP then no scheduling is possible for this node, meaning that with the specified throughput value no power management is possible for the current multiplexor. In that case, the ASAP and ALAP values for the nodes are reverted (step 7).

Otherwise, the multiplexor is selected to be power managed and the current ASAP and ALAP values become the new values for the nodes. In any case, the algorithm now returns to step 3 with the next multiplexor.

After all multiplexors in the circuit have been processed and those which can be power managed selected, in step 10 new precedence edges are created between the last node in the control input fanin and the top nodes in the 0,1-input fanin of each of these multiplexors. With this new edges, we allow HYPER's original scheduling algorithm to determine a complete schedule (step 11), targeting minimum hardware resources for the desired throughput.

The final step is to map the scheduled CDFG into execution units (datapath) and generate the Finite State Machine (controller) that generates the signals that control the loading of registers and the flow of data through multiplexors. For the datapath we use HYPER's algorithm directly. However for the controller we had to develop a new routine. The controller is somewhat more complex since the loading of the input registers to some of the execution units will depend on signals generated by some previous computation.

IV. Techniques to Improve Power Management

Tight constraints on throughput and hardware resources may leave very little slack for the ordering of operations thus restricting the effectiveness of our scheduling algorithm. We propose some techniques that can improve power management under tight constraints.

Circuit Name	Critical Path	Number of Operations				
		MUX	COMP	+	-	×
`dealer`	4	3	3	2	1	0
`gcd`	5	6	2	0	1	0
`vender`	5	6	3	3	3	2
`cordic`	48	47	16	43	46	0

TABLE I
CIRCUIT STATISTICS.

A. Multiplexor Reordering

The algorithm presented in Section III selects multiplexors for power management on an individual basis (cycle 2-9 in pseudo-code of Figure 3). The selection of a particular multiplexor may impede the selection of one or more other multiplexors, therefore the order in which the multiplexors are tested can play an important role on the number of total modules that can be shut down.

In our algorithm we test the multiplexors closer to the outputs first. It may happen that we have less savings from power managing the multiplexor that is closest to the outputs than the savings for another multiplexor and this multiplexor may not be selected because of the first being selected. We are currently working on a pre-processing algorithm which performs reordering of multiplexors trying to maximize the number of modules that can be shut down.

B. Pipelining

A common technique to increase the throughput of a design is to introduce pipeline stages. A two-stage pipeline means that two input samples are processed at any given time. The effective number of control steps needed to process one input sample is reduced by half.

For our purposes we can look at this through a different angle: adding control steps for pipelining increases the number of control steps and at the same time improves the throughput or leaves it unchanged. The addition of new control steps is very useful for power management since it creates the slack needed to schedule the control signals first.

The disadvantage of using pipelining is that the latency of the circuit increases. Also it may lead to some increase in the number of registers and execution units, increasing the area of the circuit.

V. Experimental Results

In this section we present some preliminary results that compare the power dissipation of circuits with and without power management. In Table I we give some statistics about the circuits we present results for. In the second column of Table I we give the minimum number of control steps needed to perform the operation. The remaining columns indicate the number of each different operations that make up each circuit.

All circuits were initially described in Silage [4]. They were read into HYPER [7] and a scheduling with power management was obtained using the algorithm described in Section III. Table II shows the results obtained.

The second column of Table II indicates the number of control steps we allowed each computation to take and under column three is the number of multiplexors that were selected for power management given this number of control steps. The fourth column gives the area increase due to the extra execution units needed to perform the desired

Circuit Name	Control Steps	P.Man. Muxs	Area Incr.	Number of Operations					Power Red.(%)
				MUX	COMP	+	-	×	
`dealer`	4	1	1.20	2.00	2.00	2.00	0.50	0.00	27.00
	5	1	1.00	2.00	2.00	2.00	0.50	0.00	27.00
	6	2	1.00	2.00	2.00	1.75	0.25	0.00	33.33
`gcd`	5	1	1.00	5.50	2.00	0.00	0.50	0.00	11.76
	6	1	1.00	5.50	2.00	0.00	0.50	0.00	11.76
	7	2	1.05	5.50	2.00	0.00	0.25	0.00	16.18
`vender`	5	4	1.04	4.50	2.50	1.50	1.00	1.00	41.67
	6	4	1.00	4.50	2.50	1.50	1.00	1.00	41.67
`cordic`	48	38	1.00	47.00	16.00	24.00	27.00	0.00	30.16
	52	46	1.17	47.00	16.00	22.00	23.00	0.00	34.92

TABLE II
AVERAGE NUMBER OF OPERATIONS EXECUTED USING POWER MANAGEMENT.

Circuit Name	Ctl Stp	Area			Power		
		Orig	New	Incr.	Orig	New	%
`dealer`	6	895	946	1.06	46.5	35.1	24.5
`gcd`	7	806	892	1.11	31.9	28.7	10.0
`vender`	6	2338	2283	0.98	106.2	71.4	32.8

TABLE III
POWER ESTIMATION USING SYNOPSYS.

power management. As it can be observed, in most cases there is no area penalty or the increase is very small.

In the next columns we show the average number of times that each of the operations is executed in one computation. Here we have assumed that each multiplexor has equal probability of selecting any of its inputs.

The last column of Table II gives the estimated power savings achieved by using power management. To obtain this estimate, we computed the power consumption of each of the operations using timing simulation with random input vectors, thus obtaining a relative weight of the operations in terms of power (MUX: 1; COMP: 4; +: 3; –: 3; and ×: 20). An 8-bit datapath was assumed for all examples. Recall that without power management all the operations given in Table I are always executed. These power savings are relative only to power dissipated in the datapath. The real power savings will be slightly less since the controller for the power managed circuit is slightly more complex. As it can be observed, it is possible to achieve power savings above 40% in the datapath using this scheduling for power management.

To further validate our power savings estimations we used the Synopsys power analysis tool. The RT level circuits, described in VHDL, were synthesized to gate-level using Synopsys Design Compiler(TM) and power estimate obtained with DesignPower(TM). The results are presented in Table III. For the allowed number of control steps, we compare the area increase and the power savings of the design without (`Orig`) and with (`New`) power management. These values agree with our predictions. Recall that the power reduction in Table II refers only to the datapath. Since the controller is more complex for the power managed circuit, the savings in Table III are slightly lower in Table II as expected.

VI. SUMMARY

We have presented a scheduling algorithm which, for a given throughput, exploits the slack available to operations to obtain a schedule that enables the use of power management techniques. When possible, controlling signals are scheduled first thus indicating which operations to activate and which operations to shut down. This more constrained scheduling process may lead to a larger number of execution units required. The algorithm obtains a solution that maximizes the ability to do power management while still meeting user specified throughput and hardware resource constraints.

VII. ACKNOWLEDGMENTS

This research was supported in part by the Advanced Research Projects Agency under contract DABT63-94-C-0053, and in part by a NSF Young Investigator Award with matching funds from Mitsubishi Corporation. This research was initiated when José Monteiro was employed at CCRL, NEC USA.

REFERENCES

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-Based Sequential Logic Optimization for Low Power. *IEEE Transactions on VLSI Systems*, 2(4):426–436, December 1994.

[2] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Broderson. HYPER-LP: A System for Power Minimization Using Architectural Transformations. In *Proc. of the ICCAD*, pages 300–303, November 1992.

[3] Anantha Chandrakasan and Robert Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[4] P. Hilfinger. A High-level Language and Silicon Compiler for Digital Signal Processing. In *Proc. of the Custom Integrated Circuits Conference*, pages 213–216, May 1985.

[5] H. Juan, V. Chaiyakul, and D. Gajski. Condition Graphs for High-Quality Behvioral Synthesis. In *Proc. of the ICCAD*, pages 170–174, November 1994.

[6] J. Monteiro, J. Rinderknecht, S. Devadas, and A. Ghosh. Optimization of Combinational and Sequential Logic Circuits for Low Power Using Precomputation. In *Proc. of the Chapel Hill Conf. on Advanced Research on VLSI*, pages 430–444, March 1995.

[7] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast Prototyping of Datapath-Intensive Architectures. *IEEE Design and Test*, 8(2):40–51, June 1991.

[8] A. Raghunathan and N. Jha. Behavioral Synthesis for Low Power. In *Proc. of the ICCD*, pages 318–322, October 1994.

[9] V. Tiwari, P. Ashar, and S. Malik. Guarded evaluation: Pushing power management to logic synthesis/design. In *International Symposium on Low Power Design*, pages 221–226, April 1995.