Reducing Switching Activity on Datapath Buses with Control-Signal Gating

Hema Kapadia, Giovanni De Micheli
Stanford University, CA

Luca Benini
DEIS University of Bologna, Italy

Abstract

This paper presents a practical technique for saving power dissipation in large datapaths by reducing unnecessary switching activity on wide buses. Control signals on a datapath module are gated by the observability don't care condition of the bus driven by that module to stop unnecessary switching activity on the bus. A methodology for automatic generation of gating conditions and synthesis of gated control signals from the RTL description of a design is presented. The technique has very low overheads in terms of area, power, and designer effort. It was applied to one of the integer execution units of a 64-bit superscalar RISC microprocessor. Experimental results of running various application programs on the microprocessor show on an average 26.6% reduction in dynamic switching power in the integer execution unit, with no increase in path delays.

1. Introduction

Power dissipation continues to grow as an important challenge in deep submicron chip design. Low power operation is important for system reliability, reducing cooling costs, and improving battery life in portable devices. Achieving low average power dissipation in a complex chip calls for employing low power techniques at all levels of design abstraction [1, 2].

A large fraction of the total area and power dissipation on a chip today is typically due to clocks, memory and datapaths [3]. Power dissipation in a large datapath is dominated by the dynamic switching power of heavily loaded wires of buses. Significant power savings can be achieved on a chip by reducing unnecessary switching activity on these buses. Dynamic power management techniques have been proposed at various levels of design abstraction [4-9] to detect and stop unnecessary switching activity on wires on a cycle-by-cycle basis.

This paper focuses on dynamic power management in datapaths at the register-transfer level. The technique presented here reduces unnecessary switching activity on datapath buses by selectively stopping propagation of switching activity through modules driving them. The emphasis is on achieving significant power savings without adding any area, delay or wiring complexity in the datapath, at very low computational cost. This is achieved by only using control signals to detect and stop unnecessary switching activity on datapath buses.

2. Related Work

Clock-gating is an effective dynamic power management technique [4, 5]. It has been shown to save power by gating clocks to large functional units on a complex chip [5]. The gating logic may introduce clock skew, which is a problem in high performance designs.

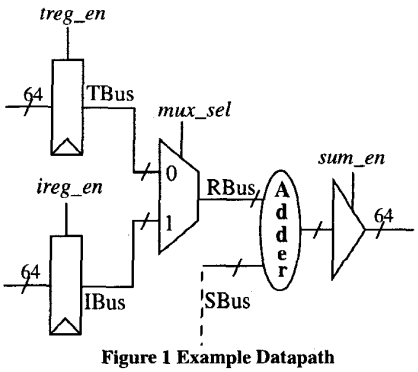

Figure 1 Example Datapath

This is avoided in [5] by making the gating logic a part of the clock generation circuitry. However, this technique does not take advantage of situations when one part of a functional unit is in use while other parts are unused. For example, Fig. 1 shows part of a 64-bit datapath. When *sum_en* is 0, *TBus* and *IBus* are unused. If some other part of the datapath is computing useful data, the clock to the entire datapath would be ungated, thereby making *TBus* and *IBus* switch unnecessarily.

Even if clocks to the registers driving *TBus* and *IBus* were gated with $\overline{sum\_en}$, *RBus* would switch unnecessarily if *mux_sel* changed while sum_en was 0. *TBus* is unused when *mux_sel* is 1 and *IBus* is unused when *mux_sel* is 0. Unnecessary switching activity due to these conditions is not controllable by clock-gating. Control-signal gating presented here takes advantage of these conditions to save power. Unnecessary switching activity is reduced on *IBus* by gating *ireg_en* with $(\overline{mux\_sel} \mid \overline{sum\_en})$ and on *TBus* by gating *treg_en* with $(mux\_sel \mid \overline{sum\_en})$. Also, *mux_sel* is held unchanged when *sum_en* is 0, to stop unnecessary switching on *RBus*.

Hold condition detection [6] finds intersections of individual hold conditions of flip-flops to create groups of flip-flops with a common clock-gating condition. It is aimed at stopping switching activity when the output of a flip-flop is going to be fed back to its input. It is not useful when enabled flip-flops are used, which is commonly the case in large datapaths. The technique in [7] collects and analyzes simulation traces to group flip-flops for clock-gating, however the grouping is trace-dependent. The gating condition needs to be determined manually, which is done automatically in our approach.

Precomputation based methods [8] use a few bits of the input bus of a module to disable unnecessary switching on the rest of the bits. Logic duplication is required for multi-output functions and multi-fanout buses, and extra wiring complexity is added in the otherwise regular structure of the datapath to route bits of buses to precomputation logic.

0-7803-4292-5/97/$10.00 © 1998 IEEE

Guarded evaluation [9] places enabled transparent latches at inputs of modules that need to be selectively turned off, using existing signals in the design as latch-enables. Automatic selection of the enable signals is based on logical implication, which is computationally complex for large designs. The area overhead of placing guard latches on wide buses is quite high.

Although precomputation and guarded evaluation are quite effective on random logic and small datapaths, they have high overheads in large datapaths. Control-signal gating does not add any logic in the datapath, resulting in very low overheads. It uses functional information available at RT-level to synthesize gated control signals, giving high computational efficiency.

3. Control-Signal Gating

The rationale of control-signal gating is that values driven on buses that are not used by the environment should be frozen in a quiescent state by stopping propagation of switching activity through their drivers. This is achieved by making a module's inputs unobservable at its output, when the output is not going to be observed at the primary outputs of the datapath. Thus, we need to compute two types of Observability Don't-care Conditions (ODCs): i) ODC_P -- the ODC of a bus at primary outputs of the datapath and ii) ODC_M -- the ODC of a module's input bus at its output. Computing and synthesizing these ODCs in large datapaths would be very expensive in terms of computational and hardware costs. Control-signal gating simplifies these ODCs by dropping parts that are dependent on datapath buses.

3.1. Computing low overhead ODC_M

Datapath modules fall into two categories: *computational modules* and *steering modules*. Arithmetic and logic modules that are not controlled by any inputs from control logic are computational modules, e.g. adders, shifters, multipliers. Steering modules steer one or none of input data buses to output, depending on the values of control signal inputs. Multiplexers, tri-state drivers and registers are steering modules.

In a computational module, input buses are observable at the output unless one of the inputs makes another unobservable. For example, if one of the inputs to a multiplier is 0, the other input is not observable at the output. Making use of this unobservability would require inserting new logic in the datapath, which would have prohibitive area and power overheads. Hence, for the purpose of control-signal gating, we will assume that inputs of a computational module are completely observable at its outputs. Thus, the ODC_M of all inputs of a computational module is 0.

On the contrary, the ODC_M of a data input of a steering module is a simple function of its control inputs. When an input bus is not being steered to the output, it is not observable at the output. Fig. 2 summarizes the ODC_M of inputs of different steering modules. A multiplexer input bus is unobservable at its output when the corresponding select line is low. A tri-state input bus is unobservable at its output when the tri-state enable signal is low. A register input bus is unobservable at its output in clock cycle T if the register enable signal is low in cycle T-1.

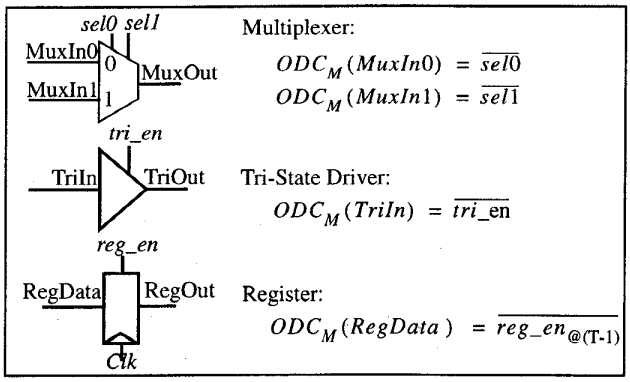

Figure 2. ODC_M for various steering modules

The ODC_M of steering module input buses can be used to compute the ODC_P of buses in a datapath.

3.2. Computing ODC_P of a bus

Computing the ODC of a signal with multiple fanouts to modules with multiple outputs requires computing ODCs along each path and combining them. Combining the ODCs of a signal at a multi-fanout point is quite complex due to terms dependent on the signal itself [10]. This is simplified in control-signal gating because the ODC of a bus contains terms dependent on control signals only. Thus, ODCs along different paths can be combined simply by taking their intersection. This is a conservative simplification that gives efficient computation of small ODC expressions. In the worst case, the simplification may result in 0 ODC_P on a bus due to data-dependent control signals. For a bus with N fanouts, if the ODC_P of each fanout is known, the ODC_P of the bus is:

$$ODC_P(BUS) = \left(\bigcap_{i=1}^{N} ODC_P(FANOUT_i) \right) \quad (1)$$

For a module with N outputs, if the ODC_P of each output is known, the ODC_P of an input bus is:

$$ODC_P(IN) = ODC_M(IN) \cup \left(\bigcap_{i=1}^{N} ODC_P(OUT_i) \right) \quad (2)$$

Traversing a datapath from primary outputs to primary inputs, the ODC_P of all buses can be computed using equations (1) and (2), and Fig. 2. Fig. 3 illustrate the computation of ODC_P of a multiple-fanout bus.

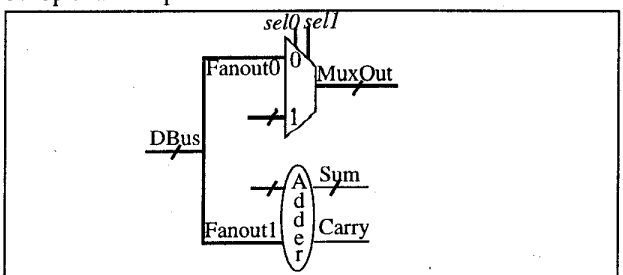

Figure 3 Computation of the ODC_P of a bus

Reconvergent fanout on a bus introduces redundant terms in the ODC_P. For example, if *MuxOut* in Fig. 3 was the second input of the Adder, the ODC_P of *MuxOut* would be the same as ODC_P of *Fanout1*, changing the ODC_P of *DBus* to:

$$\begin{aligned} ODC_P(DBus) &= \left(\overline{sel0} + ODC_P(Fanout1)\right) \bullet ODC_P(Fanout1) \\ &= ODC_P(Fanout1) \end{aligned}$$

Thus, the computed ODC_P should be optimized using a logic synthesis tool to remove these redundant terms.

3.3. Synthesizing Gated Control Signals

The ODC_M of a data input of a steering module is the inverse of one of the control inputs of that module (Fig. 2). Gating every control input by the ODC_P of the module's output would make the ODC_M of every data input true, effectively stopping unnecessary propagation of switching activity through the module. Table 1 summarizes the logic required for generating gated control signals in different types of steering modules.

Table 1. Logic for Control-Signal Gating

Module	Gated Control Signal
Tri-State	$tri\_en\_gated = tri\_en \bullet \overline{ODC_P(TriOut)}$
Register	$reg\_en\_gated = reg\_en \bullet \overline{ODC_P(RegOut)}_{@(T-1)}$
Multi-plexer	if $(ODC_P(MuxOut)_{@(T-1)})$, $sel\_gated_{@(T)} = sel\_gated_{@(T-1)}$ So, on flip-flops in the fan-in cone of select lines, $flop\_en\_gated = flop\_en \bullet \overline{ODC_P(MuxOut)}_{@(T-1)}$

In a register, the enable signal needs to be gated in cycle T-1 if the output's ODC_P is going to be true in cycle T. This is synthesized by finding the fan-in cone of the output's ODC_P up to flip-flops, and generating the ODC_P one cycle early from inputs of these flip-flops. Stopping the propagation of switching activity through a multiplexer requires two conditions: the select lines should not switch, and the selected input should not switch. Switching on a select line is stopped by applying clock-gating or control-signal gating to flip-flops feeding into its fan-in cone. The gating condition on these flip-flops is a one-cycle early version of the ODC_P of the multiplexer's output. If the fan-in cone of a select line has fanouts leaving the cone, some control logic needs to be duplicated. The required duplication is likely to be little since heavily loaded multiplexer select lines are likely to have only a few levels of logic after flip-flop outputs in order to make timing.

A multiplexer select line can not be gated if it depends on a primary input of the chip. Also, the select lines should be gated only if switching activity on multiplexer data inputs can also be stopped. If a multiplexer data input is a primary input bus that can not be gated, its select lines need not be gated.

Similarly, if the ODC_P of a bus depends on primary inputs of the chip, it would not be possible to find a one-cycle early version of it. If such a bus is the output of a multiplexer or a register, the control signals of that module can not be gated.

4. Implementation Methodology

The methodology for implementing control-signal gating requires topological ordering of the datapath, and two traversals of its steering modules in opposite topological order. We assume that the design has been partitioned into datapath and control units, and sequential feedback loops have been broken at register inputs. The datapath netlist is topologically ordered [10], treating the datapath as a DAG with modules being the vertices, and buses and control signals being the edges.

4.1. First Pass

The first pass visits all steering modules in the topologically ordered datapath from primary outputs to primary inputs. The ODC_P of the output bus of each steering module is computed using the method of Section 3.2. If the output of a module is a primary output of the datapath, its ODC is assumed to be zero unless specified otherwise. The ODC_P of a bus gets more complex as the algorithm traverses towards primary inputs. It contains OR and AND terms from all steering modules in the fanout cone of the bus up to the primary outputs. This provides opportunity for maximum removal of unnecessary switching activity looking ahead over multiple clock cycles.

However, for very deep datapaths, timing analysis must be performed after adding control-signal gating to make sure that generation of the gating logic does not become a critical path. If such a condition should arise for a bus, its ODC_P computation should be restricted to look at a smaller fanout cone, at the expense of allowing some unnecessary switching activity.

4.2. Second Pass

The second pass traverses steering modules from primary inputs to primary outputs. Depending on the steering module visited, it synthesizes gated control signals using the ODC_P of its output computed in the first pass, and the method of Section 3.3. Traversing from input to output allows checking for the case when switching activity on a multiplexer input bus can not be controlled.

Synthesizing the gating logic adds fanout to control signals used to generate the ODCs. It also adds an extra gate in the path of register and tri-state enables. Timing analysis should be performed after adding the gating logic to make sure that these control signals can make timing.

Any design database system and a synthesis tool can be used to automate control-signal gating. Due to the simplifications made in analyzing the datapath, these passes are computationally very efficient.

5. Experimental Setup and Results

We used the protocol processor (PP) of the MAGIC chip designed as a part of the Stanford FLASH multiprocessor project [11] for our experiments. The chip was designed with verilog RTL description, synthesized and laid out using Synopsys design compiler and LSI Design's physical design tools. PP is a two-way superscalar microprocessor with a RISC core, with two integer execute units: AExecuteUnit and BExecuteUnit.

Table 2. Power Savings with Control-Signal Gating

Benchmark	Runtime in clock- cycles	Total Toggle Counts		% Switching Reduction	Dyn. Switching Power (mW)		% Power Reduction
		Ungated	Gated		Ungated	Gated	
Sumup	7312	1,926,966	1,543,013	19.9	3787.8	2998.8	20.8
Saxpy	10016	2,555,114	1,918,753	24.9	3715.5	2784.9	25.0
QuickSort	23388	6,732,918	4,237,245	37.1	4339.5	2791.9	35.7
Sparse	30710	13,124,823	8,381,713	36.1	6510.4	4312.5	33.8
BubbleSort	98680	44,851,554	33,971,783	24.3	6789.7	5283.0	22.2
ProtocolProc	212798	46,334,403	36,232,823	21.8	3010.3	2346.0	22.1

Control-signal gating was applied to the BExecuteUnit. The BExecuteUnit unit consists of control and datapath sections. The control section decodes instructions and generates signals that steer data in the datapath, depending on the instruction being executed. In the final chip layout, the area of the control section is 0.37mm^2 with approximately 1.6K transistors, and of the datapath is 3.3mm^2 with approximately 45K transistors. The datapath is 64-bits wide, with 3 input and 2 output buses. The datapath consists of 7 computational and 11 steering modules. Among the steering modules, there are 3 registers for three input buses, a multiplexer that selects one of two input buses to generate an internal bus, and a register and 6 tri-state drivers generating two output buses. Control-signal gating was automated with Verilog-PLI routines. The ODC of primary outputs was assumed to be 0 (unknown). Thus, gated control signals were generated for the three input registers and the multiplexer.

Five integer programs and a cache-coherence protocol test were run on the mapped RTL of the PP that was used for final chip layout. The results are summarized in Table 2. Signal toggle counts were collected on all nodes in the BExecuteUnit for each benchmark run. Capacitive loading on each node was extracted from the chip layout. Synopsys net report was used to estimate the capacitance on the new nodes introduced by control-signal gating. Dynamic switching power was estimated by calculating the switching capacitance of all nodes.

Table 2 shows 27.4% average reduction in signal switching activity, resulting in 26.6% average reduction in dynamic switching power. The switching activity after control-signal gating reflects the cumulative effect of increased switching activity in the control section due to added gating logic, and reduced switching activity in the datapath. The area of the control section increased only by 5% after applying control-signal gating. Thus the overhead of adding control-signal gating is minimal in terms of both are and power.

The logic for the computation of ODC_P of buses did not affect any critical path timing. The enable signals of three input registers had an extra NOR2 gate in their path. After careful logic resynthesis we were able to keep the path delays of these signals the same as they were before control-signal gating.

6. Conclusions

We have presented control-signal gating, a method to stop unnecessary switching activity on datapath buses. Control-signal gating offers minimum area and power overheads, at very low computational cost. It is a technique that can be easily used by a designer to reduce unnecessary power dissipation in a datapath without having to worry about adding clock skew. We applied the technique to one of the integer execution units of a superscalar microprocessor. Running benchmarks on the microprocessor showed an average 26.6% reduction in dynamic switching power in the execution unit, with only 5% increase in area in the control section, and no reduction in speed.

Acknowledgments

We would like to thank Mark Horowitz for helpful discussions and constructive suggestions. This work was sponsored in part by NSF (NIP942119), Toshiba and ARPA (DABT63-94-C-0054).

References

[1] M. Pedram, *Power minimization in IC design: principles and applications*, ACM Trans. on Design Automation of Electronic Systems, Vol. 1, No. 1 (1996), pp. 3-56.

[2] T. Burd and R. Brodersen, *Processor Design for Portable Systems*, Journal of VLSI Signal Processing, Vol. 13, No. 2/3, Aug./Sept. 96, pp. 203-22.

[3] T. Burd and B. Peters, *A Power Analysis of a Microprocessor: A Study of an Implementation of the MIPS R3000 Architecture*, ERL Technical Report, University of California, Berkeley, 1994.

[4] L. Benini et al., *Symbolic Synthesis of Clock-Gating Logic for Power Optimization of Control-Oriented Synchronous Networks*, IEEE European Design and Test Conference, March 1997, pp. 514-20.

[5] G. Gerosa et al., *A 2.2W 80MHz Superscalar RISC Microprocessor*, IEEE Journal of Solid-State Circuits, vol. 29, no. 12, Dec. 1994, pp. 1440-54.

[6] F. Theeuwen and E. Seelen, *Power reduction through clock gating by symbolic manipulation*, Symposium on Logic and Architecture Design, Dec. 1996, pp. 184-191.

[7] M. Ohnishi et al., *A Method of Redundant Clocking Detection and Power Reduction at RT Level Design*, International Symposium on Low Power Design, 1997, pp. 131-36.

[8] M. Alidina et al., *Precomputation-Based Sequential Logic Optimization for Low Power*, Proceedings of IEEE International Conference on Computer Aided Design, Nov. 1994, pp. 74-81.

[9] V. Tiwari, S. Malik and P. Ashar, *Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design*, International Symposium on Low Power Design, April, 1995, pp. 221-6.

[10] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[11] J. Kuskin et al., *The Stanford FLASH Multiprocessor*, Proceedings of the 21st International Symposium on Computer Architecture, Chicago, IL, April 1994, pp. 302-13.