Register Transfer Level Power Optimization with Emphasis on Glitch Analysis and Reduction

Anand Raghunathan, *Member, IEEE*, Sujit Dey, *Member, IEEE*, and Niraj K. Jha, *Fellow, IEEE*

Abstract— **We present design-for-low-power techniques for register-transfer level (RTL) controller/data path circuits. We analyze the generation and propagation of glitches in both the control and data path parts of the circuit. In data-flow intensive designs, glitching power is primarily due to the chaining of arithmetic functional units. In control-flow intensive designs, on the other hand, multiplexer networks and registers dominate the total circuit power consumption, and the control logic can generate a significant amount of glitches at its outputs, which in turn propagate through the data path to account for a large portion of the glitching power in the entire circuit. Our analysis also highlights the relationship between the propagation of glitches from control signals and the bit-level correlation between data signals. Based on the analysis, we develop techniques that attempt to reduce glitching power consumption by minimizing propagation of glitches in the RTL circuit. Our techniques include restructuring multiplexer networks (to enhance data correlations and eliminate glitchy control signals), clocking control signals, and inserting selective rising/falling delays, in order to kill the propagation of glitches from control as well as data signals. In addition, we present a procedure to automatically perform the well-known power-reduction technique of clock gating through an efficient structural analysis of the RTL circuit, while avoiding the introduction of glitches on the clock signals. Application of the proposed power optimization techniques to several RTL circuits shows significant power savings, with negligible area and delay overheads.**

Index Terms— **Clock gating, controller/data path, glitch, low-power design, multiplexer restructuring, power comsumption, register-transfer level**

I. Introduction

REDUCING power consumption in very large scale integrated (VLSI) circuits has become important for several reasons. Mobile or portable electronic devices, which already account for a significant portion of all consumer electronics sold, are battery-driven. Reducing power consumption in the various components of such systems prolongs the life of the batteries, which is highly desirable. Excessive power consumption also leads to an increase in chip packaging and cooling costs, which increase the total system cost. Another benefit of reduced power consumption is increased reliability of VLSI circuits. Reducing either average power consumption or peak power consumption has its own merits. For example, average power consumption is related to battery life, while peak power consumption is related to packaging and cooling costs. In this work, our goal is to minimize average power consumption.

Most savings in power consumption can be obtained through a combination of various techniques at different levels of the design hierarchy. Several design and synthesis techniques have been proposed for power optimization at the technology [1], transistor [2], physical design [3], and logic [2] levels of the design hierarchy. In this work, we concentrate on techniques to reduce power consumption in RTL circuits. Such circuits can be either manually generated by designers, or synthesized from behavioral specifications by behavioral synthesis tools. On the architectural power estimation front, a method based on a uniform white noise model of signal statistics was presented in [4]. A more accurate estimation method based on a dual-bit-type model was presented in [5] and [6]. The use of entropy as a measure of average switching activity, and its use in high-level power estimation was suggested in [7] and [8]. More recent work on high-level power estimation has been described in [9] and [10]. Early work in architectural power optimization was presented in [1] and [11]. In [1], the use of architectural parallelism was proposed based on data path replication and pipelining to enable supply voltage scaling for power reduction. A methodology that used a variety of architectural transformations to reduce power consumption was presented in [11]. In [12], switching activity metrics were used to reduce power consumption in bit-serial digital filters. Optimizing memory-dominated computations for power consumption was addressed in [13] and [14]. Tools for power estimation and design space exploration at the behavior level were presented in [15]. In [16], module selection and pipelining were used to combat the performance degradation that results from reducing the supply voltage. Methods for performing allocation and assignment in order to minimize switching activity and switched capacitance in the data path were presented in [17]–[20]. Techniques to reduce power consumption during high-level synthesis based on reducing activity in functional units were presented in [21]. The use of limited-weight codes to minimize power consumption in buses and input–output circuitry was described in [22]. A multiphase clocking scheme for RTL circuits that reduces activity by naturally imposing shut-off for inactive parts of the circuit was proposed in [23]. An optimization tool for average and

Manuscript received June 2, 1998; revised November 24, 1998. The work of A. Raghunathan was supported by a grant from NEC C&C Research Laboratories. The work of N. K. Jha was supported by the National Science Foundation (NSF) under Grant MIP-9319269. This paper was recommended by Associate Editor R. Camposano.

A. Raghunathan is with C&C Research Laboratories, NEC USA, Princeton, NJ 08540 USA.

S. Dey is with the Department of Electrical and Computer Engineering, University of California, San Diego, CA 92093 USA.

N. K. Jha is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA.

Publisher Item Identifier S 0278-0070(99)05682-1.

peak power consumption during behavioral synthesis, based on genetic search, was described in [24]. Techniques for software power estimation and optimization were presented in [25].

A popular class of power optimization techniques, called power management [26], is based on the observation that not all parts of a circuit perform useful computations in any given clock cycle. Power management techniques at the register-transfer and behavioral levels have been proposed in [21] and [27]–[31]. While power management techniques and our RTL glitch-reduction techniques share the basic motive that they both try to eliminate unnecessary switching activity, they significantly differ in the way they achieve power reduction, when they can be applied. The main difference arises from the fact that our RTL glitch-reduction techniques do not rely upon the existence of idle components or parts of the circuit. Even when all parts of the circuit are used in each clock cycle, glitching power may be significant and hence our techniques may be applicable.

The importance of eliminating glitches in the design of digital VLSI circuits has been recognized for a long time. Avoiding glitches or hazards is known to be of great importance in asynchronous circuit design and the design of digital-to-analog and analog-to-digital converters. Several studies have reported the importance of considering glitching power during power estimation and optimization [32], [33]. The extreme sensitivity of glitching power to process variations has been pointed out in [32] and [34], where it was shown that the switching activity and power consumption due to glitches vary much more with process variations than the other components of power dissipation. The design of a multiplier with significantly reduced glitching power consumption was described in [35]. However, very few automated design and synthesis techniques exist for reducing glitching power consumption in general circuits. At the architecture and behavior levels, most previous work on power estimation and optimization ignores the effects of glitching. In particular, the effect of glitch propagation across the boundaries of blocks in the architecture has not been considered. While accurate library modeling approaches [6] can be used to account for the effect of glitches within architectural blocks, they typically assume that inputs to these blocks are glitch-free.

Most previous work at the architecture and behavior levels has also sought to focus on data-flow intensive designs, where arithmetic units like adders and multipliers account for most of the total power consumption. However, our experiments with control-flow intensive designs reveal that the power consumed by the functional units constitutes a small fraction of the total power consumption, while multiplexer networks and registers can consume a major part of the total power for such designs.

Our experiments also show that a large part of the register power consumption arises due to transitions on the register's clock input. The technique of gating clocks has been used by designers for a long time to selectively turn off parts of a system [9], [26]. Methods to automatically detect conditions under which the clock inputs to all the registers in a design can be shut off, based on identifying self-loops and unreachable states in the state transition graph (STG), were presented in [36] and [37]. However, the techniques in [36] and [37] can be applied only to the control and random logic parts of a design for which it is feasible to extract the STG. The focus on clock gating in this paper is on: 1) techniques to automatically perform clock gating efficiently using RTL information and 2) avoidance of glitching activity on the clock signals when clock gating is performed.

In this paper, we present analysis and optimization techniques to reduce glitching power consumption in RTL circuits. These techniques attempt to reduce glitching power consumption by minimizing propagation of glitches from control as well as data signals through the RTL circuit. Our techniques include restructuring multiplexer networks, clocking control signals, and inserting selective rising/falling delays, in order to kill the propagation of glitches from control as well as data signals. The key features of our glitch-reduction techniques are as follows: 1) they do not rely upon the existence of idle periods for components in a design, i.e., they are also applicable to designs with complete or near-complete resource utilization and 2) they target power consumption in all parts of the design, including multiplexer networks and registers, not just functional units.

This paper also contributes to the area of automatic clock gating insertion at the RTL. While clock gating is known to be a very effective technique for saving power, one of the associated design pitfalls it carries is the introduction of glitches on the clock signal. We present an efficient procedure to perform RTL clock gating while ensuring that there will be no glitching on the clock signal. Unlike the techniques in [36] and [37], which require the STG of the circuit, our technique derives gating conditions for register clock inputs based on a structural analysis of the RTL circuit and, hence, can be applied to the data path registers as well.

The rest of this paper is organized as follows. Section II illustrates the impact of glitch generation and propagation through detailed power/activity profiling of an example RTL circuit, and motivates the glitch-reduction techniques presented in this paper. Section III analyzes the generation and propagation of glitches in RTL circuits. This analysis leads to an understanding that forms the basis for our glitch-reduction techniques that we present in Section IV. Section IV describes techniques to reduce glitching power consumption in RTL circuits by minimizing the generation and propagation of glitches through different blocks of the circuit. For the purpose of illustration, each technique is considered separately and explained through examples. Later, a procedure is presented that automatically applies the various glitch-reduction techniques to RTL circuits. Section V presents a procedure to automatically apply clock gating at the RTL through efficient structural analysis, while avoiding the introduction of glitching on the gated clock signals. Section VI describes the experimental methodology and design flow used for evaluating the proposed techniques, and presents results on several RTL circuits.

II. Motivation

We motivate our work through the analysis of an example RTL circuit shown in Fig. 1, which computes the greatest common divisor (GCD) of two numbers. The inputs are

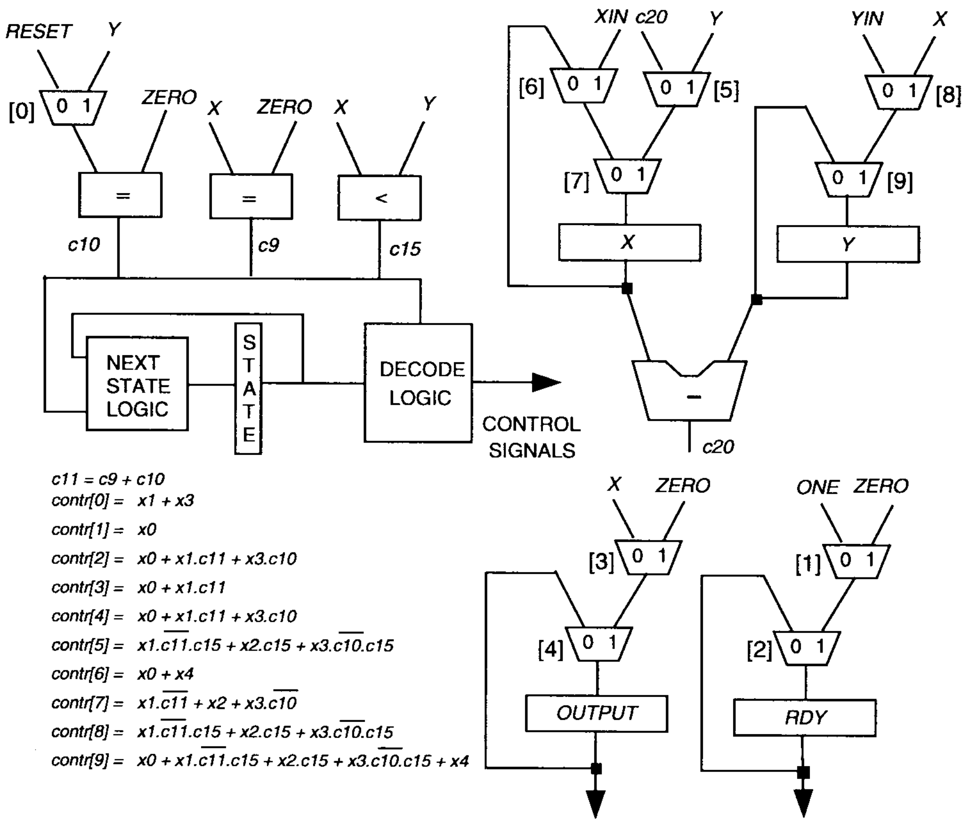

RESET
Y
[0]
0 1
ZERO
X
ZERO
X
Y
=
=
<
c10
c9
c15
NEXT STATE LOGIC
STATE
DECODE LOGIC
CONTROL SIGNALS
XIN c20
Y
YIN
X
[6]
[5]
[8]
[7]
[9]
X
Y
-
c20
X
ZERO
ONE
ZERO
[3]
[1]
[4]
[2]
OUTPUT
RDY
c11 = c9 + c10
contr[0] = x1 + x3
contr[1] = x0
contr[2] = x0 + x1.c11 + x3.c10
contr[3] = x0 + x1.c11
contr[4] = x0 + x1.c11 + x3.c10
contr[5] = x1.c̄11.c15 + x2.c15 + x3.c̄10.c15
contr[6] = x0 + x4
contr[7] = x1.c̄11 + x2 + x3.c̄10
contr[8] = x1.c̄11.c15 + x2.c15 + x3.c̄10.c15
contr[9] = x0 + x1.c̄11.c15 + x2.c15 + x3.c̄10.c15 + x4

Fig. 1. The RTL architecture of the GCD circuit.

applied at XIN and YIN, and the GCD is written into register $OUTPUT$. Since the number of cycles required for computing the GCD depends on the input values provided, an additional output signal RDY indicates when the result is available in $OUTPUT$. This circuit was derived from a behavioral description of the GCD algorithm. The high-level synthesis system SECONDS [38]–[40], was used to perform resource allocation, scheduling, and assignment to result in the RTL circuit shown in Fig. 1. The circuit consists of one functional unit—subtracter, two *equal-to* (=) comparators, one *less-than* (<) comparator, registers, multiplexer trees, the controller finite state machine (FSM), and the decode logic. The decode logic generates the control signals that configure the multiplexers in the circuit. We refer to the controller FSM and the decode logic collectively as the control logic of the circuit. The logic expressions implemented by the control logic are also shown in the figure. Literals $x0$–$x4$ represent the decoded present state lines from the controller. Literals $c9$, $c10$, and $c15$ represent results of the three comparators in the circuit.

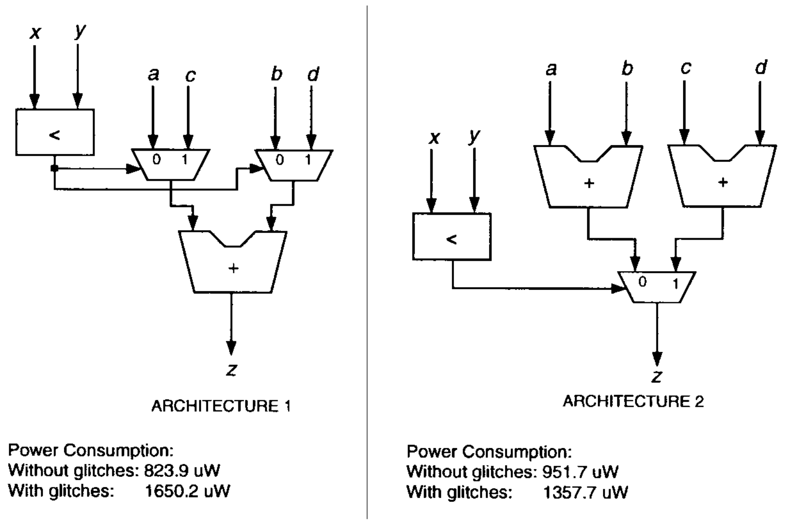
The RTL circuit shown in Fig. 1 was mapped to the NEC CMOS6 library [41]. An in-house simulation-based power calculation tool, CSIM [42], was used to measure power consumption in the various parts of the design. CSIM has been calibrated with SPICE and benchmarked within 10% of SPICE. The power and delay models for individual library cells used in CSIM were constructed using SPICE. It incorporates several state-of-the-art gate-level power simulation techniques, including state-dependent power modeling, accurate glitch filtering using inertial delay model, etc. It is currently being used for gate-level sign-off at NEC USA (Princeton, NJ).

TABLE I
POWER CONSUMPTION IN VARIOUS PARTS OF THE GCD CIRCUIT

Block	Power Consumed (% of total)
Functional units	9.08
Random Logic	4.67
Registers	39.55
Multiplexers	46.70

Table I provides the split up of the total power consumption into separate figures for the functional units (subtracter and three comparators), random logic (controller FSM and decode logic blocks), registers (including power consumed due to clock transitions), and multiplexers. It indicates that most of the power consumption is in the multiplexers and registers. Similar figures were observed for several circuits that implemented other control-flow intensive and mixed specifications.

In order to get a feel for the glitching activity in the GCD circuit, we collected data on the transition activity with and without glitches in various parts of the design. The transition activity without (excluding) glitches can be obtained by simulating the circuit under a zero-delay model. The simulations were performed using input vectors that were derived from the test bench for the behavioral specification. Table II shows the total bit transitions with and without glitches for all the control signals, and selected data path



(a) (b)

Fig. 2. Alternate architectures that implement the same function: effect of glitching.

TABLE II
ACTIVITIES WITH/WITHOUT GLITCHING FOR VARIOUS SIGNALS OF THE GCD

Signal	Activity	
	With Gl.	Without Gl.
Control Signals		
$contr[0]$	71	70.5
$contr[1]$	22	22
$contr[2]$	72	20
$contr[3]$	42	20
$contr[4]$	72	20
$contr[5]$	55.5	54
$contr[6]$	22	22
$contr[7]$	50	20
$contr[8]$	55.5	54
$contr[9]$	77	70.5
Data Path Signals		
$dp2[7..0]$	71.5	21.5
$dp4[7..0]$	92	26
$dp5[7..0]$	1124.5	247
$dp7[7..0]$	1044.5	273
$dp9[7..0]$	321.5	80.5

signals.[1] Control signal $contr[i]$ feeds the select input of the multiplexer marked $[i]$ in Fig. 1. Similarly, data path signal dpi corresponds to the output of the multiplexer marked $[i]$ in Fig. 1. Clearly, a significant portion of the total transition activity at several signals in the circuit is due to glitches. Another interesting observation is that several control signals in the GCD circuit, like $contr[2]$ and $contr[4]$, are highly glitchy. We will later analyze the generation of glitches on control signals, and illustrate that control signal glitches can have a profound effect on the glitching power consumption in the rest of the circuit.

The following example illustrates how ignoring glitches can be misleading and result in designs that are suboptimal in terms of their power consumption.

[1]CSIM counts each $0 \rightarrow 1$ or $1 \rightarrow 0$ transition as half a transition. Hence, the transition numbers that are reported throughout the paper may be fractional.

Example 1: Consider the two RTL architectures shown in Fig. 2(a) and (b). Both architectures implement the simple function: $\texttt{if}\ (\texttt{x} < \texttt{y})\ \texttt{then}\ \texttt{z} = \texttt{c} + \texttt{d}\ \texttt{else}\ \texttt{z} = \texttt{a} + \texttt{b}$. ARCHITECTURE 2 uses more resources than ARCHITECTURE 1 since the former uses two adders as opposed to one adder for the latter. Based on the number of operations performed, a metric that is commonly used to estimate power consumption at the behavior and architecture levels, it seems that ARCHITECTURE 2 would consume more power than ARCHITECTURE 1. This conclusion is supported by power estimation results which do not take glitches into account. However, when accurate power estimation that also considers glitches is performed, it turns out that ARCHITECTURE 2 actually consumes 17.7% less power than ARCHITECTURE 1.

The above observation can be explained as follows. As we shall see in Section III, the comparator generates glitches at its output though its inputs are glitch-free. In the case of ARCHITECTURE 1, these glitches then propagate through the two multiplexers to the inputs of the adder, which causes a significant increase in glitching activity and hence power consumption in the two multiplexers and the adder. In ARCHITECTURE 2, though the comparator generates glitches as before, the effect of these glitches is restricted to the single multiplexer. ■

III. GLITCH GENERATION AND PROPAGATION

In this section, we analyze the generation and propagation of glitches in RTL circuits. This analysis leads to an understanding that forms the basis for our glitch-reduction techniques that we present in Section IV. For clarity, we illustrate glitch generation in the data path blocks (functional units, comparators, and multiplexer trees) and in the control logic separately.

A. Glitch Generation in Data Path Blocks

This following example illustrates that data path blocks can generate a significant amount of glitches.

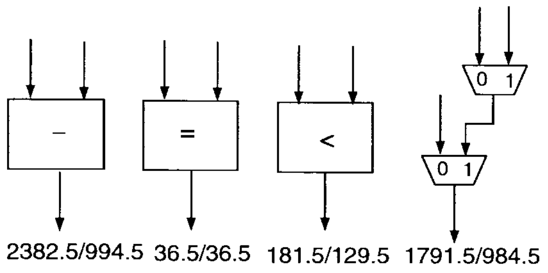


Fig. 3. Glitch generation in various data path blocks.

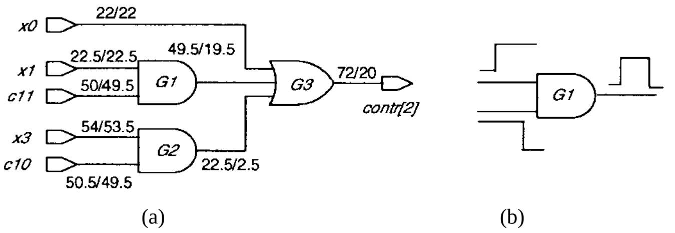


Fig. 4. (a) Implementation of control signal $contr[2]$ and (b) generation of glitches at gate $G1$.

Example 2: Consider the elements shown in Fig. 3—a subtracter, an equal-to comparator, a less-than comparator, and a 3 : 1 multiplexer tree—as representative data path blocks for studying glitch generation. Note that registers do not generate glitches at their outputs. Each block was mapped to the NEC CMOS6 library, and then simulated under long input sequences that consisted of random vectors. We measured the total number of bit-transitions (including glitches) at the block outputs and the number of $zero$-delay transitions (i.e., the number of transitions not counting glitches). The block outputs in Fig. 3 are annotated with the results, where the numbers indicate transitions with and without glitches, respectively. The results clearly indicate significant generation of glitches in various data path blocks. In the equal-to comparator, no glitches were generated due to the fact that all its paths are balanced. However, even in such cases, wiring delays can disturb the balance of delays and thus cause generation of glitches. ■

When data path blocks like those shown in Fig. 3 are connected together, the glitches generated by the various blocks propagate through the following blocks, causing in several situations, as illustrated later, an explosion in glitches and glitching power consumption. The techniques presented in this paper attempt to control the propagation and explosion of glitches by killing most, if not all, of the glitches at various locations in the circuit.

B. Glitch Generation in the Control Logic

Though the control logic itself accounts for only a small portion of the total circuit power, it has a significant role in the total circuit power due to its ability to generate glitches on the control signals, and the effects of glitchy control signals on the rest of the circuit. Hence, it is important to study how the control logic generates glitches, a topic that has not been addressed in the previous work on power optimization at the architecture or behavior levels. The inputs to the decode logic within the control logic are fed by the outputs of comparators and the state flip-flops (FF's) of the controller. The outputs of the control logic include the control signals fed to the data path. The previous subsection has already demonstrated that the outputs of comparators can be glitchy. The glitches on the comparator outputs can propagate through the control logic and cause glitches on the control signals. The control logic can also generate a lot of glitches even if its inputs are glitch-free. The aim of this section is to illustrate and analyze through examples the generation of glitches in the control logic.

Example 3: Consider the RTL circuit shown in Fig. 1 once again. Let us focus on control signal $contr[2]$, which is highly glitchy according to the statistics of Table II. The portion of the decode logic that implements this control signal is shown in Fig. 4(a). We observe that though the inputs are largely glitch-free, significant glitches are generated at AND gates $G1$ and $G2$. After careful analysis, the generation of glitches was attributed to two conditions:

C1) A rising transition on signal $x1$ was frequently accompanied by a falling transition on $c11$. Thus, the rising transition on $x1$ and the falling transition on $c11$ are highly correlated.

C2) Transitions on signal $x1$ arrive earlier than transitions on signal $c11$.

Condition C1 arises due to the functionality of the design: most of the times when state s_1 is entered (rising transition on $x1$), the comparisons evaluated by the comparators feeding $c9$ and $c10$ evaluate to zero, changing from one in the previous state. On the other hand, condition C2 is a result of the delay/temporal characteristics of the design. These conditions, captured graphically in Fig. 4(b), lead to the generation of glitches at gate $G1$, that propagate to control signal $contr[2]$. A similar explanation holds for the output of gate $G2$ being glitchy. ■

Example 4: For this example, we will use a portion of an RTL circuit that is a preprocessor for a barcode reader. We shall focus on a particular control signal, $contr[1]$, whose implementation is given in Fig. 5. Signals $state[2]$, $state[1]$, and $state[0]$ are fed by the FF's of the controller. Signals $x3$ and $x4$ represent decoded state signals, i.e., $x3$ ($x4$) assumes a logic value of one if and only if the controller is in state $s3$ ($s4$), or equivalently, $state[2] \cdots state[0]$ assume the values 011 (100). Signals $x3$, $x4$, and control signal $contr[1]$ are annotated with their transition counts including and excluding glitches. The figure indicates that the output of gate $G5$ is highly glitchy even though glitches do not occur at its inputs. In order to explain the generation of glitches at gate $G5$, consider the partial state transition graph for the controller that is shown in Fig. 6(a). The figure indicates a loop involving states s_3 and s_4. This results from a *while* loop in the VHDL behavioral specification. Since this loop is the innermost loop among all loops in the behavioral description, it is executed a large number of times. Thus, the state transitions from s_3 to s_4 and from s_4 to s_3 are frequently executed. Fig. 6(b) shows how the inputs and output of gate $G5$ vary under these two state transitions. A transition from s_3 to s_4 causes a rising transition on signal $state[2]$, and falling transitions

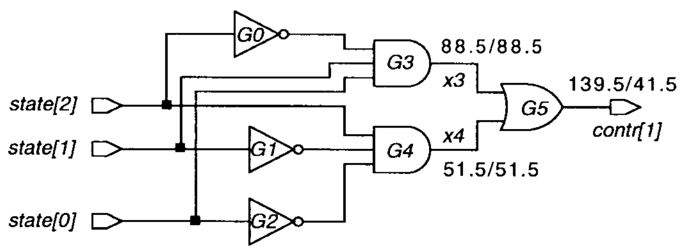


Fig. 5. Implementation of control signal $contr[1]$ in the Barcode RTL circuit.

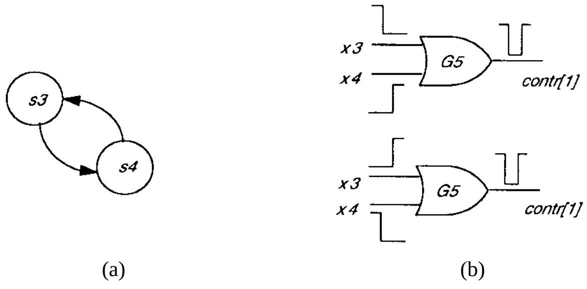


Fig. 6. (a) Partial STG for Barcode controller and (b) generation of glitches at gate $G5$.

on signals $state[1]$ and $state[0]$. These transitions in turn cause rising and falling transitions on signals x_4 and x_3, respectively. However, the rising transition on x_4 arrives later than the falling transition on x_3, since the delays of inverters $G1$ and $G2$ are reflected in the former, while the delay of inverter $G0$ is not reflected in the latter. This results in a 1-0-1 static hazard or glitch at the output of gate $G5$, as shown in Fig. 6(b). Similarly, a controller state transition from s_4 to s_3 leads to a rising transition on x_3 and a falling transition on x_4 such that the former transition arrives later. This again leads to glitches at the output of gate $G5$, as shown in Fig. 6(b). ■

In general, glitches are generated at the control signals due to the simultaneous presence of the following two conditions.

1) Functional: Correlation between rising and falling transitions at two or more signals that feed a gate.

2) Temporal: The controlling[2] to noncontrolling transition arrives earlier at the gate's input.

IV. Glitch Reduction Techniques

In this section, we describe our techniques to reduce glitching power consumption in RTL circuits by minimizing the propagation of glitches through different blocks of the circuit. Several glitch-reduction transformations/techniques suited to specific scenarios are presented in Sections IV-A and IV-B. For the sake of illustration, each technique is presented separately through examples. Section IV-C provides a unifying procedure that automatically applies all the glitch-reduction techniques to RTL circuits.

[2]A controlling value at a gate's input determines the value at the gate's output independent of the values at the other inputs to the gate.

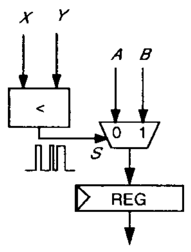


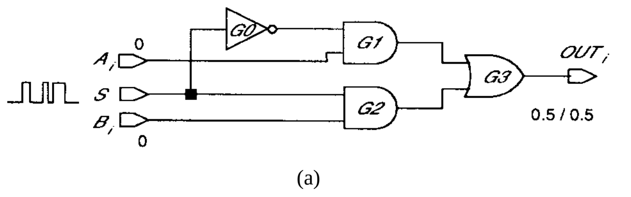
Fig. 7. Effect of data signal correlations on propagation of control signal glitches: example circuit.

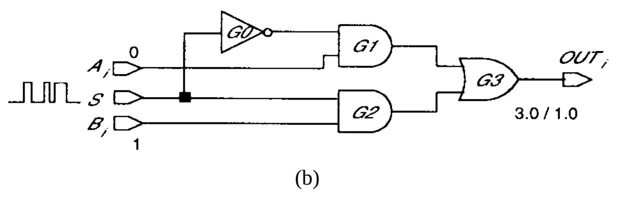
A. Reducing Glitch Propagation from Control Signals

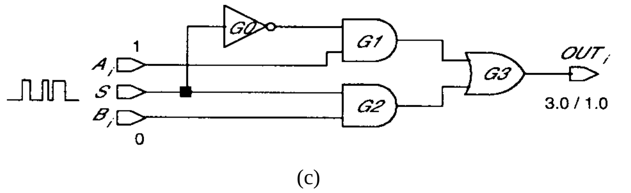
As shown in Sections II and III, significant glitches can be generated on control signals. We have also seen from Section II that these glitches can propagate through the other parts of the circuit, causing significant power dissipation. Our aim is to stop glitches on control signals as close to their source as possible in order to reap the maximum benefits in terms of power savings. We illustrate each of our techniques separately through examples in this subsection. We integrate these techniques into a single power optimization framework in Section IV-C.

1) Glitchy Control Signals and Data Correlations: Consider the circuit shown in Fig. 7. A multiplexer selects between two 8-b data signals, A and B, depending on whether the expression $X < Y$ evaluates to *true* or *false*. The output of the multiplexer is written into a register. Since we know a less-than comparator generates glitches at its output, the select signal of the multiplexer is glitchy. In control-flow intensive designs, it is often the case that the control signals are late-arriving and on the critical paths, whereas the data signals are relatively early arriving (this occurs especially in designs that contain an abundance of conditional assignment statements). In such situations, it may be possible to significantly reduce glitching activity at the output of the multiplexer by exploiting spatial correlations between its data inputs, as explained next.

The glitches on the select signal of the multiplexer in Fig. 7 propagate to its output. In order to study this propagation, consider the gate-level implementation of a bit-slice of the multiplexer that is shown in Fig. 8(a)–(d). The four figures represent the cases when the relevant bits A_i and B_i assume values of $\langle 0, 0\rangle$, $\langle 0, 1\rangle$, $\langle 1, 0\rangle$, and $\langle 1, 1\rangle$, respectively. In each figure, the multiplexer output is annotated with the number of transitions with and without glitches, in accordance with our usual notation. In the $\langle 0, 0\rangle$ case, glitches on the select signal S are killed at the AND gates $G1$ and $G2$ due to controlling side inputs that arrive early. When the data inputs are $\langle 0, 1\rangle$, glitches on S do not propagate through gate $G1$, but do propagate through gates $G2$ and $G3$. A similar explanation holds when the data inputs are $\langle 1, 0\rangle$. Finally, when the data inputs are $\langle 1, 1\rangle$, glitches on S propagate through gates $G1$ and $G2$. The output of the multiplexer is glitchy as a result of the interaction of the glitchy signal waveforms at $G1$ and $G2$. The exact manner in which the waveforms interact depends on the propagation and inertial delays of the various wires and gates in the implementation, which are modeled by the simulator that







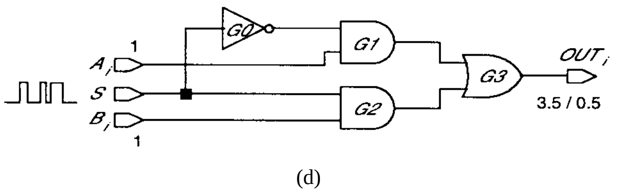


Fig. 8. Propagation of glitches on a multiplexer select signal for various values of data signals.

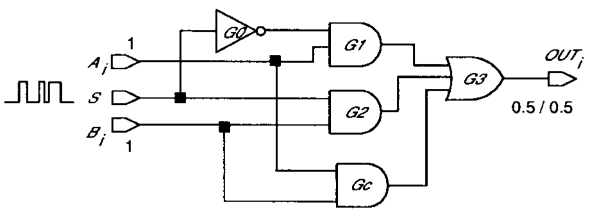


Fig. 9. Effect of adding the consensus term on glitch propagation.

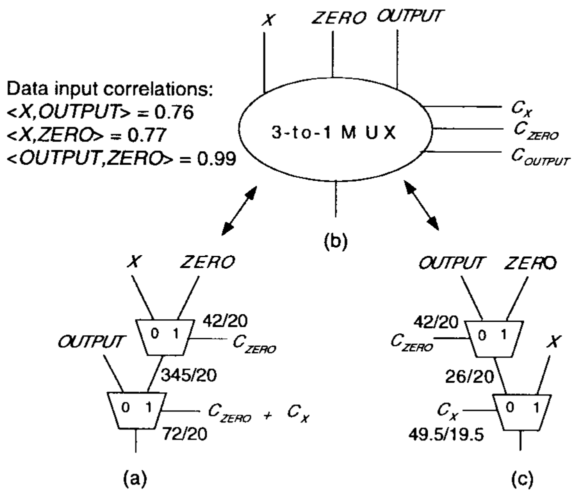


Fig. 10. Multiplexer restructuring to enhance data correlations: (a) initial multiplexer network, (b) abstract 3 : 1 multiplexer, and (c) restructured network.

we used to obtain estimates of switching activity and power consumption.

There are many ways of preventing the propagation of glitches for the $\langle 1, 1\rangle$ case. One way is to insert a buffer, whose delay is equal to the delay of inverter $G0$, at the fanout branch of S that feeds gate $G2$. Ideally, the insertion of the buffer should result in complementary waveforms at the outputs of $G1$ and $G2$ that cancel each other out, i.e., result in a steady one at the output of the multiplexer. However, this solution will depend on the exact propagation and inertial delays of the gates and wires in the circuit. Moreover, slight variations in circuit parameters due to process variations can invalidate the effect of path balancing. Another possible solution, that we consider more robust, is to add an extra gate Gc, as shown in Fig. 9. Gc realizes $A_i \cdot B_i$ which is the consensus of $\overline{S} \cdot A_i$ and $S \cdot B_i$, and hence its addition does not change the function computed by the output. When the data inputs are $\langle 1, 1\rangle$, Gc effectively kills any glitches at the other inputs of $G3$ that arrive after the output of Gc settles to a one. For all other input cases, the output of Gc evaluates to a zero and previous explanations hold. Simulating the implementation of Fig. 9 validates that glitches on the select signal do not propagate to the output any more when the data inputs are $\langle 1, 1\rangle$. For the circuit of Fig. 7 alone, adding Gc to the multiplexer implementation results in a 17.5% decrease in total power consumption.

There is a power overhead incurred in adding the gate based on the consensus term due to its own power consumption, and the overhead of having a three-input OR gate in place of a two-input OR gate. Thus, this transformation is applied to multiplexers with glitchy select inputs only when the arrival time at the data inputs is smaller than the arrival time at the select signal by a prespecified margin, and when the probability of the data inputs being $\langle 1, 1\rangle$ is above a prespecified threshold. In Section IV-C, we discuss how to judiciously choose multiplexers in the design to which we add consensus terms so that the power savings obtained are maximized.

Note that with the addition of the consensus term, glitches will not propagate from the select signal to the multiplexer output if the data values are correlated ($\langle 0, 0\rangle$ or $\langle 1, 1\rangle$), else the glitches will propagate to the multiplexer outputs. We next show how to restructure a multiplexer tree so as to maximize data correlations and, hence, minimize propagation of glitches from select signals of multiplexers.

2) Enhancing Data Correlations by Restructuring Multiplexer Networks: Consider the 3 : 1 multiplexer network feeding register $OUTPUT$ in the GCD circuit of Fig. 1. The tree of 2 : 1 multiplexers is shown in Fig. 10(a). Functionally, the multiplexer tree can be thought of as an abstract 3 : 1 multiplexer, as shown in Fig. 10(b). The conditions under which $OUTPUT$, X, and $ZERO$ are selected are represented as C_{OUTPUT}, C_X, and C_{ZERO}, respectively. Note that C_{OUTPUT}, C_X, and C_{ZERO} must be mutually exclusive. The cumulative switching activities with and without glitches are shown for various signals in the figure.

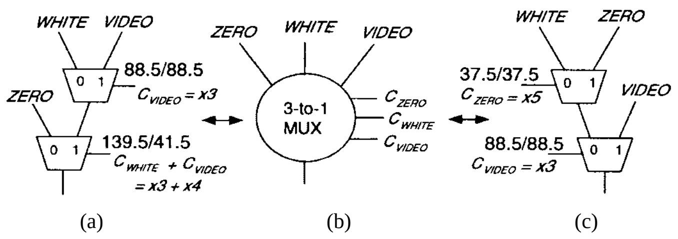


Fig. 11. Eliminating glitchy control signals: (a) initial multiplexer network, (b) abstract 3 : 1 multiplexer, and (c) restructured network.

We recall, from our earlier discussions in this section, that glitch propagation from control signals through a multiplexer is minimized when its data inputs are highly correlated (assuming the addition of the consensus gate to the multiplexer implementation when necessary). This observation can be used to reduce glitch propagation from control signals feeding a multiplexer network by restructuring it, as explained next.

Given the abstract representation of the 3 : 1 multiplexer network in Fig. 10(b), there are several possible implementations which enhance correlations of the data inputs to the multiplexers in the tree. For this example, select signal C_{ZERO} is observed to be glitchy, leading to propagation of glitches to the output of the first 2 : 1 multiplexer in Fig. 10(a). Note that data signals $OUTPUT$ and $ZERO$ are highly correlated at the bit level. Hence, in order to minimize the propagation of glitches on C_{ZERO} through the multiplexer tree, we transform the multiplexer tree to the implementation shown in Fig. 10(c), such that the highly correlated data signals $OUTPUT$ and $ZERO$ become inputs to the first 2 : 1 multiplexer. This significantly lowers the switching activity at the output of the first 2 : 1 multiplexer to 26/20 from 345/20 originally. Multiplexer restructuring can also help to eliminate glitchy select signals as discussed next.

3) Restructuring Multiplexer Networks to Eliminate Glitchy Select Signals: Consider again the 3 : 1 multiplexer tree and its representations shown in Fig. 10(a) and (b), respectively. The implementation in Fig. 10(a) uses only two of the signals from the set $\{C_{OUTPUT}, C_X, C_{ZERO}\}$. In general, in order to implement an abstract $n : 1$ multiplexer with n data inputs and n select inputs as a tree of 2 : 1 multiplexers, it can be shown that anywhere between $\lceil \log_2 n \rceil$ and $n-1$ of the select conditions can be used to generate the expressions for the select signals for the 2 : 1 multiplexers in the implementation, depending on the exact structure of the implementation. It is possible that among the set of select signals to an abstract $n : 1$ multiplexer, some carry a lot of glitches in their implementations while others do not. Similarly, as shown in Example 4 of Section III, it is possible that certain expressions involving the select signals of the abstract $n : 1$ multiplexer can be glitchy even though the individual signals are not. Our aim, therefore, should be to restructure the multiplexer tree in such a way that as few of the glitchy select inputs to the abstract $n : 1$ multiplexer (or combinations of select signals that are glitchy) as possible are used. This concept is illustrated by the next example.

Consider the 3 : 1 multiplexer network that is shown in Fig. 11(a). This network is part of the `barcode` reader RTL circuit. As illustrated in Example 4 of Section III, the select signal of the second multiplexer in Fig. 11(a) that implements the expression $x3 + x4$ is glitchy. An alternative implementation of the 3 : 1 multiplexer network, that does not require the use of any glitchy select signal expressions, is shown in Fig. 11(c).

4) Clocking Control Signals to Kill Glitches: When all the methods presented so far to reduce the effect of glitches on control signals do not help, we use the clock signal to kill glitches on control signals that feed either select inputs of multiplexers, or function select inputs of arithmetic-logic units (ALU's). Let us assume that the design is implemented using rising-edge-triggered FF's and a single-phase clock with a duty cycle of 50%. However, our methods can be extended with slight modifications to more complex clocking schemes as well. Our technique is illustrated in Fig. 12(a)–(c). Consider the 2 : 1 multiplexer shown in Fig. 12(a), that is part of the RTL circuit implementing the controller for an unmanned auto vehicle (UAV) [43]. The conditions for selecting $ZERO$ and $c21$ are $C_{ZERO} = x3 \cdot \overline{c5}$ and $C_{c21} = x3 \cdot c5$, respectively. In this case, both C_{ZERO} and C_{c21} are glitchy due to the generation of glitches in the less-than comparator that generates signal $c5$. Thus, multiplexer restructuring transformations that eliminate glitchy control signals cannot be applied here. Fig. 12(b) shows the modified circuit after clocking the select signal to the multiplexer. The original select signal is ANDed with the inverted clock. We refer to the output of the AND gate as the clocked select signal. This ensures that for the first half of the clock period, when the clock is high, the output of the AND gate is forced to zero in spite of the glitches on its other input. Fig. 12(c) shows example waveforms for the clock, the original select signal, and the clocked select signal. The select input to the multiplexer in Fig. 12(a) and (b) is annotated with the activity with and without glitches. The switching activity numbers in the figure show that clocking the control signal significantly reduces its glitching activity.

The technique of clocking control signals needs to be applied judiciously due to the following reasons. By clocking the control signal, we are preventing it from evaluating to its final value until time $T/2$, where T is the clock period. In general, this could lead to an increase in the delay of the circuit, if the control signal needs to settle to its final value before $T/2$ in order to meet the specified timing constraints at the circuit outputs. It is possible to derive a shifted clock waveform such that the required arrival time of the control signal being clocked is not violated. However, this involves exactly matching the required arrival time at the control signal and the shift imparted to the clock, which is best done after layout [35]. We apply the technique of clocking control signals conservatively, i.e., only when the required arrival time at the control signal is greater than $T/2$ by a specified margin of safety. Another problem that can be caused by clocking control signals is that of introducing extra transitions on the control signal under certain conditions. Consider a situation where the control signal remains at a steady one over a pair of clock cycles. By forcing the control signal to zero in the first half of both the clock cycles, we are actually introducing extra transitions on the control signal, which can lead to increased

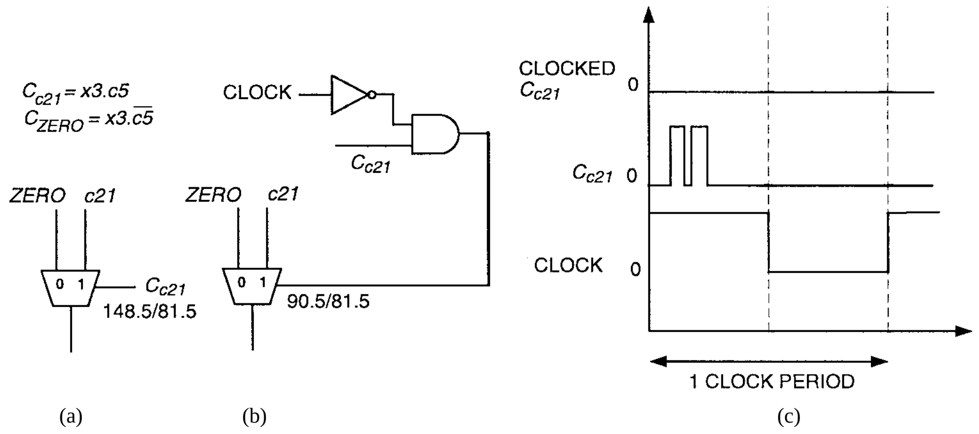


Fig. 12. Clocking control signals to kill glitches: (a) initial multiplexer network, (b) multiplexer network with clocked control signal, and (c) sample waveforms.

power consumption. Thus, the scheme presented in Fig. 12(b) leads to most power savings when the probability of the control signal evaluating to a one (signal probability) is low. On the other hand, if the probability of the control signal evaluating to a one is very high, an alternate scheme can be used to clock the control signal by ORing the original control signal with the clock. This forces the clocked control signal to a one in the first half of the clock period, as opposed to zero in the case of the first scheme, avoiding extra transitions on the clocked control signal when it evaluates to a one.

B. Minimizing Glitch Propagation from Data Signals

The previous subsection outlined several ways in which the propagation of glitches from control signals can be reduced to save power. The data signals to a circuit block can also be glitchy, as seen in Section III. In this subsection, we present several techniques to restrict propagation of glitches on data signals.

1) Glitch-Reduction Using Selective Rising/Falling Delays: Consider the example circuit shown in Fig. 13(a). A 2 : 1 multiplexer selects between the outputs of two adders, and the multiplexer's output is fed to another adder. This is a situation that occurs commonly in RTL designs that employ data chaining. As shown in Section III, adders generate glitches even when their inputs are glitch-free. Thus, the data inputs to the multiplexer have glitches, which propagate through the multiplexer and then through the third adder, causing significant power dissipation. We propose a technique called *selective delay insertion* that can be used to cut down the propagation of glitches through the circuit, as follows.

Consider the gate-level implementation of a bit-slice of the multiplexer as shown in Fig. 13(b). Both the data inputs to the multiplexer are glitchy. Consider a pair of consecutive clock cycles q_1 and q_2 such that the select signal to the multiplexer makes a $1 \rightarrow 0$ (falling) transition from q_1 to q_2. If the falling transition at S is early arriving, there will be an early rising transition at the output of gate $G0$ that implements $\overline{S}$. Consequently, the side input of $G1$ will become noncontrolling early, allowing the data input glitches to propagate through $G1$. This propagation can be minimized by ensuring that the side input to $G1$ remains controlling as long as possible,

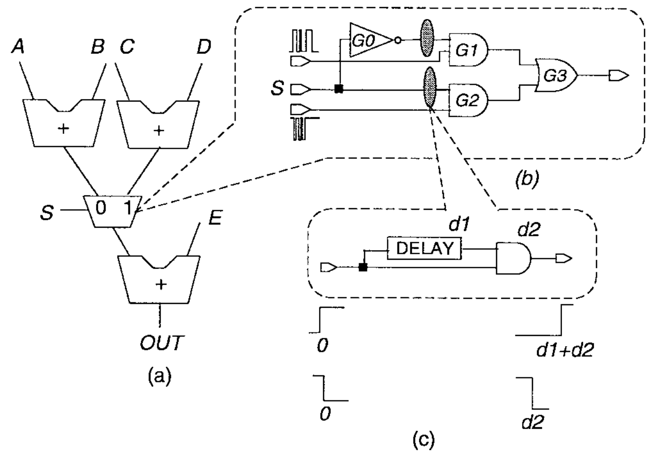


Fig. 13. (a) Example circuit, (b) multiplexer bit-slice with selective delays inserted, and (c) implementation of a rising delay block.

which can be achieved by delaying the rising transition at the output of $G0$ ($\overline{S}$). In other words, we would like to add a "rising transition delay" to the output of $G0$ ($\overline{S}$). Similarly, to minimize glitch propagation through gate $G2$ when there is an early rising transition at S, it is desirable to delay the rising transition on the fanout branch of S that feeds AND gate $G2$. Since we wish to delay selected (either rising or falling, but not both) transitions at certain signals, we refer to the technique as selective delay insertion. The selective rising delay blocks are represented by the shaded ellipses shown in Fig. 13(b). A possible implementation of a rising delay block, that uses one AND gate and a delay element, is shown in Fig. 13(c). The delay element is constructed using either a series of buffers or inverters added to the input. The implementation uses the fact that a falling transition at any one input of an AND gate is sufficient to force the output to zero, while, on the other hand, the latest arriving rising transition among all the inputs will trigger a rising transition at the output. Under a simplified delay model of d_1 ns for the delay block and d_2 ns for the AND gate, it can be seen that a rising transition at the input is delayed by $(d_1 + d_2)$ ns, while a falling transition is delayed by only d_2 ns. Since d_1 is typically large compared to d_2, the slight increase in propagation of glitches due to the additional delay of d_2 ns imparted to the falling transition is far outweighed by the savings obtained for the case of a rising transition. A selective falling delay block is similar to the circuit shown in Fig. 13(c), except that the AND gate is replaced by an OR gate.

Applying the above technique to the example circuit shown in Fig. 13(a) results in a 15.4% decrease in overall power consumption. The following three conditions need to be considered when inserting selective delays. First, a selective delay block comes at a price in terms of the power it consumes. Thus, the expected savings must be large enough to justify this overhead. Second, inserting a rising delay block leads to a reduction in the propagation of glitches through a multiplexer only in the clock cycles in which there is a rising transition at the delay block's input. Thus, the probability of a rising transition at the signal where we desire to insert a selective rising delay block should be high. Third, to ensure that inserting the selective delay block does not increase the delay of the circuit, we insert the delay block only on select signals that have sufficient slack. Note that for an entire m-b multiplexer, it suffices to have only one selective rising delay each at the select signal S and its complement $\overline{S}$. To allow this low-cost solution, we use an m-b selector instead of a multiplexer (a selector implements the function $S_1 \cdot A + S_2 \cdot B$). The two select signals (S_1 and S_2) are generated explicitly outside the selector as S and $\overline{S}$.

2) Effect of Multiplexer Restructuring Transformations on Glitchy Data Signals: Multiplexer restructuring transformations can also be used to reduce the propagation of glitches on data signals. We illustrate this concept using a small portion of the GCD RTL circuit, which is shown in Fig. 14(a). The subtracter's output, $c20$, has a lot of glitches which propagate through the multiplexer shown in the figure, and also through the logic that is fed by the multiplexer. Let us assume that signal Y is glitch-free. Fig. 14(b) shows the equivalent abstract 2 : 1 multiplexer. We utilize the fact that there might be several instances when the value of the select signal is a *don't care*, i.e., when $C_{c20} + C_Y$ is not a tautology. In Fig. 14(a), the zero-input (i.e., $c20$) is selected whenever C_Y evaluates to zero. That includes not just the off-set of C_Y, but also the set of *don't care* conditions. In this case, the behavioral synthesis tool specified the select signal to zero in all the *don't care* conditions in order to simplify the control logic. We can utilize the *don't care* conditions by selecting the less glitchy data input of the multiplexer in the *don't care* cases. The transformed implementation of the 2 : 1 multiplexer that is shown in Fig. 14(c) illustrates this idea. By having the glitchy data input $c20$ as the one-input of the multiplexer, and thus forcing the select signal to be C_{c20}, we ensure that the glitchy data input is selected as infrequently as possible, reducing the propagation of glitches to the multiplexer output.

3) Clocking Control Signals to Kill Data Signal Glitches: When the techniques presented above to handle glitchy data signals are not applicable or not adequate to reduce glitching power consumption, we utilize the concept of clocking control signals to kill data signal glitches. For example, when both the select and data inputs to a multiplexer are glitchy and multiplexer restructuring transformations fail to eliminate the glitchy select signal, we resort to clocking to solve the

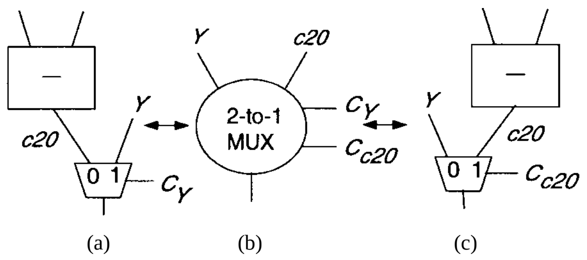


Fig. 14. Using multiplexer restructuring transformations for glitchy data signals: (a) initial multiplexer network, (b) abstract 2 : 1 multiplexer, and (c) restructured network.

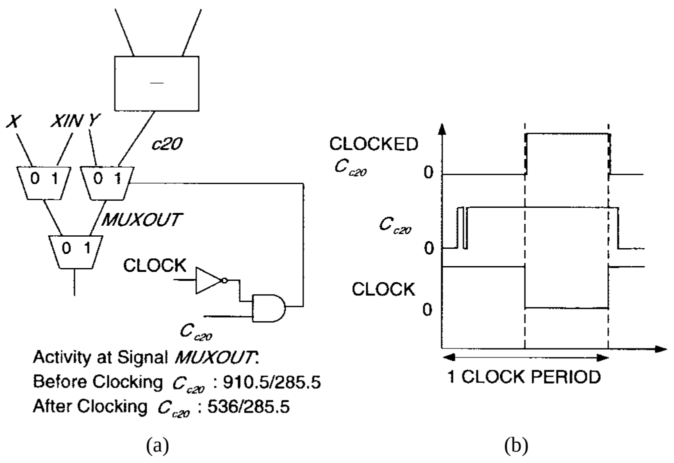

X
XIN Y
c20
0 1
MUXOUT
CLOCK
C_{c20}
Activity at Signal MUXOUT:
Before Clocking C_{c20} : 910.5/285.5
After Clocking C_{c20} : 536/285.5
CLOCKED C_{c20}
0
CLOCK
1 CLOCK PERIOD
(a)
(b)

Fig. 15. Clocking control signals to kill data signal glitches: (a) example circuit and (b) sample waveforms.

problem. As described in the previous subsection, we force the control signal to take a particular value for the first half of the clock period. In order to further illustrate this technique, consider the circuit shown in Fig. 15(a). The subtracter's output $c20$, which is glitchy, feeds the data input of a 2 : 1 multiplexer. As shown in the figure, this results in significant glitches at the output of the multiplexer. Clocking select signal C_{c20} alleviates the problem as explained next. Since the value on the select signal to the multiplexer is forced to zero for the first half of the clock period, the multiplexer selects the value of data input Y for this duration. Thus, the glitches on the subtracter's output are killed at the multiplexer for approximately the first half of the clock period. For this example, this leads to a large decrease in the glitching activity at the multiplexer output, as shown in the figure. Sample waveforms for the clock, original select signal, and the clocked select signal are shown in Fig. 15(b). Again, as mentioned before in this subsection, it is important to consider the required arrival time at the select signal to the multiplexer and the extra transitions that can be potentially introduced on the clocked select signal before applying this technique.

C. Algorithms for Application of RTL Glitch-Reduction Transformations

The previous two subsections described the various RTL transformations that we use to minimize the glitching power consumption in RTL circuits, and the conditions under which each technique is applicable, with the help of illustrative

```
Procedure RTL_GLITCH_REDUCTION($RTL\_circuit$)
{
    COLLAPSE_MUX_NETWORKS($RTL\_circuit$);
    LEVELIZE_CIRCUIT($RTL\_circuit$);
    $level \leftarrow 0$;
    while ($level \leq$ MAX_LEVEL($RTL\_circuit$)){
        $activity\_stats \leftarrow$ COLLECT_ACTIVITY_STATISTICS($RTL\_Circuit$);
        $delay\_stats \leftarrow$ COLLECT_DELAY_STATISTICS($RTL\_Circuit$);
        for (each node $RTL\_node$ at $level$){
            GR_Transform($RTL\_node$, $activity\_stats$,$delay\_stats$);
        }
        $level + +$;
    }
}

Procedure GR_TRANSFORM($RTL\_node$, $activity\_stats$, $delay\_stats$)
{
    switch ($RTL\_node.type$){
    case MUX: // multiplexers feeding FUs or Registers
        CREATE_MINGLITCH_MUX_TREE_LEVEL ($RTL\_node$, $activity\_stats$, $delay\_stats$);
    case CONTROL: // MUX select signals, ALU function selects
        CLOCK_CONTROL_SIGNALS($RTL\_node$, $delay\_stats$);
    }
}

Procedure CREATE_MINGLITCH_MUX_TREE_LEVEL ($MUX\_node$, $activity\_stats$, $delay\_stats$)
{
    $data\_input\_list \leftarrow$ list of all nodes feeding data inputs of $MUX\_node$;
    while ($data\_input\_list \neq \phi$){
        Select ($best\_inp1$, $best\_inp2$) such that
            GLITCH_ACTIVITY_ESTIMATE($best\_inp1$, $best\_inp2$) is minimum;
        $New\_2x1\_MUX \leftarrow$ CREATE_2X1_MUX($MUX\_node$, $best\_inp1$, $best\_inp2$);
        INSERT_SELECTIVE_DELAYS($New\_2x1\_MUX$, $activity\_stats$, $delay\_stats$);
        ADD_CONSENSUS_GATES($New\_2x1\_MUX$, $activity\_stats$, $delay\_stats$);
        $New\_control\_node \leftarrow$ CONTROL node corresponding to $New\_2x1\_MUX$;
        CLOCK_CONTROL_SIGNALS($New\_control\_node$, $delay\_stats$);
    }
}
```

Fig. 16. Glitch-reduction procedure overview.

examples. In this section, we describe a systematic procedure to apply these transformations in an integrated manner to an RTL circuit, in conjunction with a switching activity and delay estimator. We would like to emphasize that the focus here is not on estimation of switching activity and delay, but on the integration of the various transformations in a single framework that is driven by activity/power and delay estimators. In our framework, we use an in-house logic-level power simulator, CSIM [42], to provide activity statistics, and an RTL static timing analysis tool [40] to provide timing estimates. For the purposes of this subsection, we treat the activity analysis tool simply as a subroutine that we call in order to obtain signal statistics, including signal and transition probabilities, correlations, glitching activities, etc. Similarly, the timing analysis tool is treated as a subroutine that is called to obtain arrival and required time information at the signals of interest. However, in terms of computational efficiency of the tool implementation, some of the biggest bottlenecks arise due to the communication and iterative use of these tools. Hence, for the purpose of obtaining an efficient tool implementation, it is necessary to carefully instrument the interfaces between the tools. These integration details are provided in Section VI.

The pseudocode for the top-level procedure that applies the various glitch-reduction transformations to an RTL circuit is shown in Fig. 16. The circuit is assumed to consist of an interconnection of RTL blocks or nodes that could represent registers, multiplexers, control nodes, or functional units. Note that functional units could include arithmetic units such as adders and subtracters, comparators, or vector logic operators. Since our transformations focus on the reorganization and modification of multiplexer networks and control logic, the first step performed is to identify the multiplexer networks and collapse them into an intermediate form for further processing.

Procedure COLLAPSE_MUX_NETWORKS traverses the RTL circuit and collapses all multiplexer networks into abstract $n:1$ multiplexers, as described in Section IV. The circuit is levelized by ordering blocks in forward topological order, from primary input/register output to primary output/register input.

The activity and delay estimators are first called in order to compute signal statistics and arrival/required times at selected signals.

Application of the glitch-reduction techniques to a node in the RTL circuit affects signal statistics and glitching activities at all other nodes in its transitive fanout. In order to take the above dependency into account, we traverse the RTL circuit in increasing order of levels in order to apply the glitch-reduction transformations. The netlist update and execution of the activity/delay estimator is performed in an incremental manner in order to avoid the high computation time to perform activity/delay analysis from scratch. Details are provided in Section VI. At each level in the circuit, we call procedure GR_TRANSFORM() on each node or component at that level.

Procedure GR_TRANSFORM reduces glitch generation and propagation at a given node in the RTL circuit. As mentioned in the earlier sections, our focus is on transforming the multiplexer and control nodes to reduce power consumption. For multiplexer nodes, we have developed a procedure to apply all the transformations described in the previous sections in an integrated manner, which we call procedure CREATE_MINGLITCH_MUX_TREE_LEVEL. If the given node is a control node that generates one or more control signals, we apply the consensus gate insertion and control signal clocking transformations to reduce glitching activity at the control signal.

Procedure CREATE_MINGLITCH_MUX_TREE_LEVEL works as follows. If the current node is an $n:1$ multiplexer node, procedure CREATE_MINGLITCH_MUX_TREE_LEVEL decomposes the $n:1$ multiplexer by extracting a set of $2:1$ multiplexers that constitute one level of the corresponding multiplexer tree such that glitching activities at the outputs of the created $2:1$ multiplexers are minimized. In other words, procedure CREATE_MINGLITCH_MUX_TREE_LEVEL decomposes an abstract $n:1$ multiplexer into $\lfloor n/2 \rfloor$ $2:1$ multiplexers feeding an $\lceil n/2 \rceil:1$ abstract multiplexer. We attempt to minimize the glitching activity at the output of the various created $2:1$ multiplexers by first grouping data inputs so as to eliminate glitchy control signals, maximizing data input correlations, and using select glitchy data inputs as infrequently as possible. After that, procedure CREATE_MINGLITCH_MUX_TREE_LEVEL automatically determines which bit-slices, if any, of each created $2:1$ multiplexer to add the consensus term to, which bit-slices to add selective delays to, and whether to clock the control signals that feed the select inputs of the created $2:1$ multiplexers.

The complexity of the RTL glitch-reduction procedure may be determined as follows. The complexity of collapsing the multiplexer networks, levelizing the RTL circuit, and traversing it in order to apply the transformations to each component are all linear in the number of vertices and edges of the graph representation of the RTL circuit. The complexity of applying the procedure GR_TRANSFORM at a multiplexer node is $O(I^2)$ where I is the largest number of inputs to an abstract multiplexer (this is due to the search for the pair of inputs that results in a new $2:1$ multiplexer with minimum glitching activity at its output). The number of times the delay estimator and activity estimator are called is equal to the number of levels in the final circuit. This is because all nodes at a level may be transformed independently using the delay and activity information obtained from the previous call to the respective estimators. The RTL delay estimator that we employ is based on a topological traversal that is linear (in practice) in the number of components in the RTL circuit, since it uses look-up table techniques to compute the delay of chained components. The power simulator that we use to report activity statistics has a complexity that is linear in the number of events encountered during the simulation. While there exist pathological cases of circuits in which a large number of events could be generated, in practice we have observed from our experiments that its run time also scales close to linearly with the size of the circuit being simulated.

The above complexity analysis, however, is of limited value in our scenario since the constant overheads involved in intertool communication, creation/updating of interface files, etc., may dominate the total computation time. We use two techniques to achieve efficient intertool communication in our framework—incremental netlist update, and restricting the analysis tools to only work on the relevant portion of the circuit. These are described in further detail with the experimental methodology in Section VI.

V. REDUCING REGISTER POWER CONSUMPTION BY GATING CLOCK INPUTS TO REGISTERS

We observed in Section II that registers are responsible for a significant fraction of the total power consumption. A large part of the register power consumption, in turn, is due to the transitions on the clock inputs to registers. We present techniques to automatically perform clock gating efficiently using RTL information, while avoiding glitching activity on the clock signals and satisfying timing constraints.

Methods to automatically detect conditions under which the clocks can be shut off based on identifying self-loops and unreachable states in the state transition graph were described in [36] and [37]. The distinguishing features of our work with respect to the previous technique of gating clocks are as follows: 1) we identify separate gating conditions for each register in the circuit, which can lead to greater opportunities for gating clocks and 2) our procedures, which are based on a structural analysis of the given RTL circuit, are applicable to all the registers of a design, including the data path registers. The techniques presented in [36] and [37] require the STG of the circuit and hence can be applied only to the control logic and random logic parts of a design.

The basic technique of gating clocks to registers is illustrated in Fig. 17. In the circuit in Fig. 17(a), we note that the register reloads its previous value when the less-than comparator's output is zero. Hence, whenever the comparator output evaluates to zero, the clock input to the register can be suppressed from making a transition. For this example, we assume that the design is based on single-phase rising-edge-triggered FF's. Fig. 17(b) shows two candidate schemes to gate the clock input to the register. We refer to these schemes as *scheme 1* and *scheme 2*, respectively. The rationale behind *scheme 1* is that the register clock input would be forced to

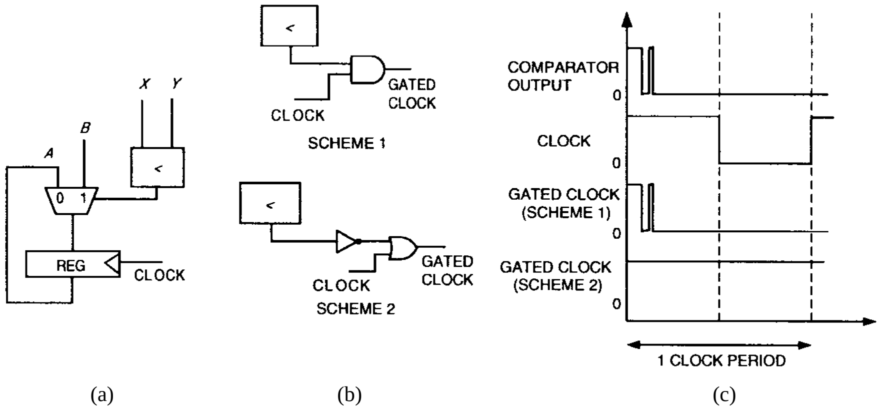


Fig. 17. Gating clock inputs to registers: (a) example circuit, (b) two candidate schemes implementing gated clocks, and (c) sample waveforms.

a zero whenever the output of the comparator evaluates to a zero, thus suppressing the unnecessary transitions on the clock. The reasoning for *scheme 2* is the same as for *scheme 1*, except that the register clock input is forced to a one whenever the comparator's output evaluates to zero. Thus, an initial analysis suggests that both the schemes are equivalent. However, the schemes impose different timing constraints on the clock gating signal. In order to illustrate why, consider the sample waveforms shown in Fig. 17(c) for both the schemes. For *scheme 1* to work, we require that the comparator's output evaluate to zero *before the clock edge rises,* i.e., at $t = 0$. This is not possible since the new inputs to the comparator are applied only at $t = 0$, and the comparator obviously requires a finite nonzero delay before its output is stable. Hence, *scheme 1*, when implemented exactly as shown in Fig. 17, does not work when timing considerations are taken into account. On the other hand, *scheme 2* will work as long as the gating condition stabilizes before half the clock period.

Scheme 1 of Fig. 17 can be enhanced by inserting a transparent latch at the signal representing the gating condition before ANDing it with the clock signal [26]. While this incurs the overhead of an additional latch, it enables us to apply clock gating as long as the gating condition settles within the complete clock period. Thus, in some situations, the enhanced *scheme 1* may be more advantageous than *scheme 2*. The techniques presented later in this section to derive time-constrained clock gating conditions are equally applicable with both enhanced *scheme 1* and *scheme 2*; the only difference is the timing constraint passed to the procedure. One possible strategy that can be used to combine the two schemes is to: a) apply *scheme 2* when either the gating condition derived without regard to timing constraints arrives before half the clock period, or when it is possible to derive a reduced gating condition (as described later in the section) without significantly reducing the gating signal probability and b) apply the enhanced *scheme 1* in other situations.

In order to gate the clock input to a register, we first need to compute the set of conditions under which the register does not need to load a new value. These gating conditions can be easily obtained during behavioral synthesis if the design is synthesized from a behavioral description. Lifetime analysis, that is used during behavioral synthesis for resource sharing, reveals the exact set of conditions under which each register needs to load a new value. In general, the gating conditions thus obtained are in the form of expressions involving the present state of the controller and also the outputs of comparators that evaluate the various conditions in the behavioral description.

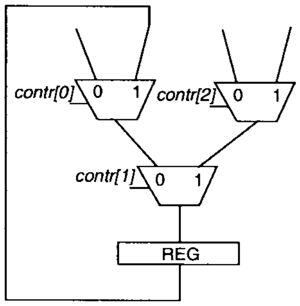


Fig. 18. Deriving clock gating conditions for registers.

Alternatively, if only the RTL description of the design is given, we use the following procedure to derive the gating conditions. For each register, we analyze the multiplexer network that feeds it to determine whether the register's output is fed back as one of the data inputs to the multiplexer network. Note that the presence of such a self-loop from a register's output back to its data input is typical in manually designed RTL circuits as well as RTL circuits produced by high-level synthesis tools. The conditions under which this self-loop is logically activated are also those under which the register retains its previous data value. We traverse the path through the multiplexer network starting at the identified data input to the multiplexer network and ending at the output of the multiplexer network that is connected to the input of the register. We then compute the condition for this path to be activated, in terms of the select signals connected to the individual multiplexers along the path. The condition that the path is activated can be written as the *conjunction* of the conditions that each multiplexer along the path selects the on-path input.

Example 5: Consider the register and multiplexer tree feeding it shown in Fig. 18. Assuming that we are using *scheme 2* shown in Fig. 17, the gating condition for the clock input to the register is $contr[0] \cdot \overline{contr[1]}$. ■

Now consider the general case, where a register in an RTL circuit has a self-loop passing through n $2:1$ multiplexers in a multiplexer network. Let Sel_1, Sel_2, $\ldots$, Sel_n represent the conditions under which the multiplexers in the path that forms the self-loop select the on-path inputs. Note that Sel_i is either equal to the control signal that feeds the select input of the corresponding multiplexer, or its complement, depending on whether the on-path input is the one-input or zero-input to the multiplexer. The gating condition for the register clock can then be written as $Sel_1 \cdot Sel_2 \cdots Sel_n$.

Since the logic to compute the select signals to the various multiplexers in the multiplexer network is already implemented, we only need to add the logic required to invert the control signals where necessary, and compute the conjunction of the individual conditions for each multiplexer in the path. The above procedure to derive gating conditions does not guarantee that the required timing constraint (the gating condition should stabilize within the first half of the clock period for *scheme 2*, or within the complete clock period for enhanced *scheme 1*) is met. Failure to meet the required timing constraints can lead to the generation of spurious transitions on the clock inputs to registers. This not only causes additional power consumption, but can also cause the registers to *load incorrect values.* One possible solution to get around this problem is to clock the design slower. Alternatively, it is possible to explore the possibility of changing the duty cycle of the clock while maintaining the same clock period. However, these schemes involve either a performance penalty or a change in the initial clocking scheme, both of which may often be undesirable.

In order to ensure that the required timing constraints are met, we augment the above procedure as follows. After computing the expression for the gating condition as explained in the previous paragraph, we first check using an initial implementation whether the arrival time at the gating condition is less than half the clock period. If this condition is not met, we derive a *reduced* gating condition which is guaranteed to satisfy the timing constraint. The expression for the gating condition is first converted to a sum-of-products $\bigcup_{i=1}^{k} \text{Prod}_i$. The high-level delay estimator, FEST [40], is used to determine the arrival times at the signals representing each product term. A subset of the product terms Prod_i is identified such that the largest arrival time among the product terms plus the delay of the logic required to compute the OR of the selected terms is less than half the clock period. We would like to mention here that heuristic methods to obtain a reduced cover for the gating condition were presented in [36]. However, the aim there was to minimize the overhead required to synthesize the logic implementing the gating condition, while our primary goal is to eliminate terms that cause the initial expression to violate the timing constraint.

While our procedures derive a separate gating condition for each data path register, it is possible to combine the gating conditions for a group of registers into a single gating condition that can be used to gate the clock input to all the registers in the group. The benefit of such merging is that it is possible to suppress unnecessary transitions in parts of the clock distribution network as well. However, the clock transition to multiple registers can be suppressed only if all of the individual gating conditions are satisfied. Hence, the number of transitions suppressed at the clock inputs to some of the registers, when we use a merged gating condition, may be less than the number of transitions that could have been suppressed by using individual gating conditions.

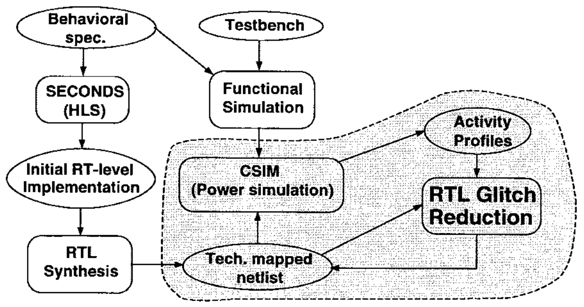


Fig. 19. Experimental methodology.

VI. Experimental Results

We present results of the application of the proposed power-reduction techniques to seven RTL circuits implementing: GCD, a barcode reader preprocessor (`Barcode`) [44], the controller for an unmanned auto vehicle (UAV) [43], a vending machine controller (`Vendor`) [45], a line-drawing process that is part of a graphics controller chip [46] (`Graphics`), a transmitter process that implements part of the X.25 communications protocol (`X.25`) [38], and a circuit that computes the dot product of two vectors (`Dot_Prod`). All the circuits except the last one are control-flow intensive designs, whose behavioral descriptions contain nested data-dependent loops and conditionals. The last circuit (`Dot_Prod`) is a data-flow intensive design, and has been included to demonstrate the applicability of our glitch-reduction techniques to such designs as well. The gate-level implementations of these circuits range in size from around 550 gates to around 2000 gates, and contain between 28 and 85 FF's.

The design flow used for our experiments is shown in Fig. 19. The initial RTL circuits were obtained by synthesizing VHDL behavioral descriptions using the SECONDS high-level synthesis system [38]–[40]. The high-level synthesis system optimizes performance (average or expected number of clock cycles) during scheduling as well as the clock period during resource sharing. Thus, all the RTL circuits that were synthesized can be considered to be optimized for performance. Moreover, we did not allow our power optimization tool any slack over the minimum clock period that the initial RTL circuit could satisfy. With looser performance specifications, we believe that the power savings attained by our glitch-reduction transformations could be higher, since we would be able to apply transformations to larger parts of the circuit.[3]

[3]However, it should be mentioned that since most of our techniques are quite low-overhead, and add only a minimal amount of circuitry, applying our RTL glitch-reduction techniques even with an unconstrained clock period would not increase the clock period of the original circuit very significantly. Conversely, even if a large slack in the clock period is available, only a small part of it is utilized for power *versus* performance tradeoffs using our glitch-reduction techniques. Thus, other power optimization techniques, such as supply voltage scaling or gate sizing, can also be used to exploit power *versus* performance tradeoffs in such cases.

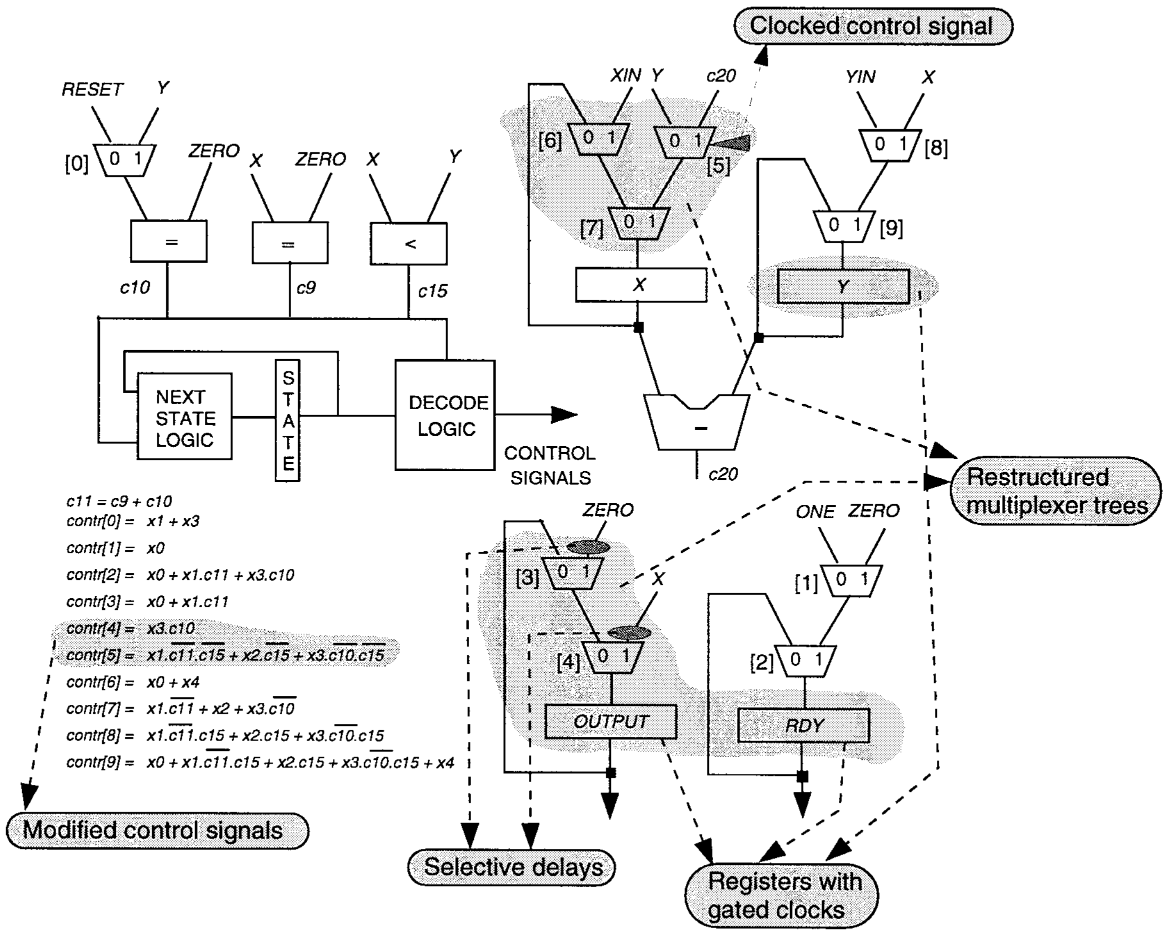

Clocked control signal
RESET
Y
[0]
0 1
ZERO
X
ZERO
X
Y
=
=
<
c10
c9
c15
NEXT STATE LOGIC
STATE
DECODE LOGIC
CONTROL SIGNALS
XIN
Y
c20
YIN
X
[6]
[5]
[8]
[7]
[9]
X
Y
-
c20
Restructured multiplexer trees
ZERO
ONE
ZERO
[3]
[1]
[4]
[2]
OUTPUT
RDY
$c11 = c9 + c10$
$contr[0] = x1 + x3$
$contr[1] = x0$
$contr[2] = x0 + x1.c11 + x3.c10$
$contr[3] = x0 + x1.c11$
$contr[4] = x3.c10$
$contr[5] = x1.\overline{c11}.\overline{c15} + x2.\overline{c15} + x3.\overline{c10}.\overline{c15}$
$contr[6] = x0 + x4$
$contr[7] = x1.\overline{c11} + x2 + x3.\overline{c10}$
$contr[8] = x1.\overline{c11}.c15 + x2.c15 + x3.\overline{c10}.c15$
$contr[9] = x0 + x1.\overline{c11}.c15 + x2.c15 + x3.\overline{c10}.c15 + x4$
Modified control signals
Selective delays
Registers with gated clocks

Fig. 20. The resulting GCD circuit after the application of our power optimization techniques.

The RTL circuits were mapped to the NEC CMOS6 [41] library, and the mapped gate-level netlists were simulated using CSIM [42] to determine glitching activity and signal statistics of various signals in the circuit. As mentioned earlier, the CSIM power estimator incorporates several state-of-the-art gate-level power simulation techniques, including state-dependent power modeling, accurate glitch filtering using inertial delay model, etc. The simulation results are passed back to our RTL power optimizer that transforms the initial RTL design to reduce the generation and propagation of glitches. As mentioned in Section IV-C, the application of glitch-reducing transformations can impact glitching activities and signal statistics in a global manner. It is necessary to incrementally recompute the information used to drive the transformations in order to ensure maximal power savings. Hence, we apply our glitch-reduction transformations to all RTL nodes at one level in the RTL circuit, incrementally modify the technology-mapped netlist to reflect the transformations, and recompute glitching activities and signal statistics for guiding the application of glitch-reduction transformations to RTL nodes at other levels. Both the original and the optimized RTL circuits were evaluated for area and delay using NEC's VARCHSYN synthesis system [47], and for power consumption using CSIM.

The vectors used for simulation were obtained as follows. For each design, we used the behavioral test bench for the design to simulate the scheduled behavioral description using the VHDL simulator, VSIM [48], and obtain a cycle-by-cycle input vector trace. The above step is especially important for control-flow intensive designs where, unlike data-flow intensive specifications, the number of clock cycles required to perform the computation varies depending on the input values. The cycle-by-cycle input vector trace was used both for collecting information about glitching activity and signal statistics and for evaluating the initial and optimized designs for power consumption.

Example 6: Before proceeding to provide the quantitative results of our experiments, we would like to illustrate the effect of our power optimization techniques on the GCD RTL circuit. The final GCD RTL circuit that results from the application of our tool is shown in Fig. 20.

The multiplexer trees consisting of multiplexers {[3], [4]}, and {[5], [6], [7]} were restructured to reduce glitch propagation from the data signal $c20$ (the subtracter's output) and from control signal $contr[4]$, and the control signals feeding them were redesigned as necessary. Multiplexers [3] and [4] were modified by inserting selective delay gates to minimize glitch propagation from their control signals. Control signal $contr[5]$ was clocked in order to further reduce glitch propagation from $c20$ (the subtracter output). Finally, the clock inputs to registers Y, $OUTPUT$, and RDY were gated. It was found that the exact clocking condition for register X violated the

TABLE III
EXPERIMENTAL RESULTS

Circuit	Original			Optimized			Power Red.	CPU
	Power	Area	Delay	Power	Area	Delay	(%)	
GCD	8.738mW	1037	32.29ns	7.229mW	1034	31.94ns	17.27%	427s
Barcode	9.409mW	1945	49.73ns	7.770mW	1968	47.29ns	17.42%	671s
UAV	10.878mW	1954	83.48ns	8.023mW	1967	83.15ns	26.25%	619s
Vendor	10.471mW	1595	70.13ns	7.725mW	1617	71.63ns	26.22%	545s
Graphics	9.653mW	3865	132.60ns	6.751mW	3914	132.68ns	30.06%	1,128s
X.25	2.056mW	2409	129.53ns	1.658mW	2434	135.11ns	19.36%	858s
Dot_Prod	19.704mW	3341	98.69ns	15.521mW	3347	97.78ns	21.23%	1,017s

timing constraint. In addition, the reduced gating condition that satisfied the timing constraint derived by our tool was found to have a very low probability, hence the tool decided not to gate the clock input to register X. ■

Table III reports the quantitative results of our experiments. The power consumption, area (# of transistor pairs), and delay of the original circuits are reported under column *Original*. The corresponding numbers for the optimized circuits (obtained after applying the glitch-reduction techniques presented in this paper) are reported under column *Optimized*. The column *Power Red* provides the percentage power-reduction obtained through the application of the techniques proposed in this paper. The CPU times required by our tool are presented under column CPU. The CPU times presented indicate the time taken to perform the glitch-optimizing transformations after the initial RTL synthesis has been completed (i.e., the CPU time required to perform the tasks corresponding to the shaded region in Fig. 19).

The results indicate that the proposed glitch-reduction transformations can significantly reduce power consumption of RTL designs (up to 30.06% and 22.54% on the average). Note that our glitch-reduction techniques target power reduction solely by reducing the propagation of glitches between various blocks in the RTL circuit. Thus, our techniques can be combined with other power-reduction techniques that attempt to suppress transitions that do not correspond to glitches, or techniques that optimize power by also changing the physical capacitance. For the example circuits considered in our experiments, the most common glitch-reduction transformation applied was multiplexer restructuring, followed by selective delay insertion, and then clocking of control signals. Clock gating was applicable to all the example circuits considered. Another point worth mentioning is that while our glitch-reduction techniques are applicable to control-flow intensive as well as data-flow intensive designs, there is a subtle difference in the source of power savings. In control-flow intensive designs, control signals are often late arriving and glitchy, in addition to data signals. Hence, the transformations that reduce glitch propagation from control signals are extensively applicable. In data-flow intensive designs, most of the glitching power is due to data chaining, hence the most commonly employed transformations are those that reduce glitch propagation from data signals (Section IV-B).

In some cases, the area or delay of the circuit after applying our techniques is slightly lower than the original circuit, while in other cases there are minor area and/or delay overheads. Upon analysis of these variations, we found that they could be attributed to the fact that multiplexer restructuring transformations can lead to a modification of the control logic, which can result in area and delay fluctuations due to logic synthesis optimizations. At the RTL, the control logic is represented as a set of Boolean expressions. As mentioned in the description of the algorithm (Section IV-C), our tool uses a static RTL delay estimator [40] to check whether the delay of the circuit is within the given clock period constraint. The delay estimator works with the Boolean expressions for the control logic, assuming a straightforward implementation. However, the control logic is typically modified during the logic synthesis process (due to optimizations such as factorization, technology mapping, etc.). Thus, although the RTL estimator may tell the glitch-reduction procedure that a given transformation does not affect the circuit delay, the changes in the control logic, which lead to variations in scope for logic synthesis, could result in minor delay variations in the gate-level netlist. Sometimes, the changes to control logic introduced by restructuring multiplexer trees actually improve the post-logic-synthesis area and delay, while for other examples it increases area and/or delay. However, as can be seen from Table III, the area and delay overheads, when incurred by our power-reduction techniques, are nominal, and do not show any consistent trend (positive or negative) in our experience.

The total CPU times required for the examples shown in Table III varied from 427–1128 s on a SPARCstation 20 with 128 Mbytes main memory. In order to efficiently realize the flow of Fig. 19, we exploited the incremental nature of the circuit modifications performed by the proposed techniques to perform incremental technology mapping and netlist update. As mentioned in Section IV-C, the run-time of our tool is dominated by the requirement to perform activity/power estimation at the gate-level. We had to do this only due to the lack of availability of a sufficiently accurate RTL glitching activity/power estimator. For the purposes of comparison, the time required to run a typical optimizing logic synthesis script on the initial RTL circuits varied from 185–1181 s. We believe that the CPU times can be significantly reduced through the use of efficient RTL power estimators [9], [10] to drive the power-reduction techniques presented in this paper.

VII. Conclusions

We presented several techniques to reduce power consumption in RTL designs. The key features of our techniques are as follows: 1) we focus on power consumption due to the propagation of glitches across the various blocks in the circuit and 2) we target power consumed by not just functional units, but also multiplexers and registers in the design, which may consume a major part of the total power in control-flow intensive designs. Our glitch-reduction techniques are based on an analysis of generation and propagation of glitches in RTL circuits. Based on the observation that registers can consume a significant part of the total power and most of the register power is in turn caused by transitions on the clock input, we gate clock inputs to registers with conditions derived by an analysis of the RTL circuit, ensuring that glitches are not introduced on the clock signals. Experimental results demonstrate the efficacy of the proposed techniques in providing significant power reductions with nominal area and delay overheads.

References

[1] A. P Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power CMOS digital design,” *IEEE J. Solid-State Circuits,* vol. 27, pp. 473–484, Apr. 1992.

[2] S. Devadas and S. Malik, “A survey of optimization techniques targeting low-power VLSI circuits,” in *Proc. Design Automation Conf.,* June 1995, pp. 242–247.

[3] H. Vaishnav and M. Pedram, “PCUBE: A performance driven placement algorithm for low-power designs,” in *Proc. European Design Automation Conf.,* Sept. 1993, pp. 72–77.

[4] S. R. Powell and R M. Chau, “Estimating power dissipation of VLSI signal processing chips: The PFA technique,” in *Proc. VLSI Signal Processing IV,* Sept. 1990, pp. 250–259.

[5] P. E. Landman and J. M. Rabaey, “Power estimation for high level synthesis,” in *Proc. European Design Automation Conf.,* Feb. 1993, pp. 361–366.

[6] ——, “Black-box capacitance models for architectural power analysis,” in *Proc. Int. Workshop Low-Power Design,* Apr. 1994, pp. 165–170.

[7] D. Marculescu, R. Marculescu, and M. Pedrarn, “Information theoretic measures for energy consumption at the register-transfer level,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 81–86.

[8] F. N. Najm, “Toward a high-level power estimation capability,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 87–92.

[9] J. Rabaey and M. Pedram, Eds., *Low-Power Design Methodologies.* Norwell, MA: Kluwer Academic, 1996.

[10] A. Raghunathan, N. K. Jha, and S. Dey, *High-Level Power Analysis and Optimization.* Norwell, MA: Kluwer Academic Publishers, 1998.

[11] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, “Optimizing power using transformations,” *IEEE Trans. Computer-Aided Design,* vol. 14, pp. 12–31, Jan. 1995.

[12] A. Chatterjee and R. K. Roy, “Synthesis of low-power DSP circuits using activity metrics,” in *Proc. Int. Conf. VLSI Design,* Jan. 1994, pp. 265–270.

[13] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, and H. DeMan, “Global communication and memory optimizing transformations for low-power systems,” in *Proc. Int. Workshop Low-Power Design,* Apr. 1994, pp. 203–208.

[14] D. Lidsky and J. Rabaey, “Low-power design of memory intensive functions,” in *Proc. Symp. Low-Power Electronics,* Oct. 1994, pp. 16–17.

[15] R. Mehra and J. Rabaey, “Behavioral level power estimation and exploration,” in *Proc. Int. Workshop Low-Power Design,* Apr. 1994, pp. 197–202.

[16] L. Goodby, A. Orailoglu, and P. M. Chau, “Microarchitectural synthesis of performance-constrained, low-power VLSI designs,” in *Proc. Int. Conf. Computer Design,* Oct. 1994, pp. 323-326.

[17] A. Raghunathan and N. K. Jha, “Behavioral synthesis for low power,” in *Proc. Int. Conf. Computer Design,* Oct. 1994, pp. 318–322.

[18] ——, “An ILP formulation for low power based on minimizing switched capacitance during datapath allocation,” in *Proc. Int. Symp. Circuits and Systems,* May 1995, pp. 1069–1073.

[19] J. M. Chang and M. Pedram, “Register allocation and binding for low power,” in *Proc. Design Automation Conf.,* June 1995, pp. 29–35.

[20] A. Dasgupta and R. Karri, “Simultaneous scheduling and binding for power minimization during microarchitecture synthesis,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 69–74.

[21] E. Musoll and J. Cortadella, “High-level synthesis techniques for reducing the activity of functional units,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 99–104.

[22] M. Stan and W. P. Burleson, “Limited-weight codes for low-power I/O,” in *Proc. Int. Workshop Low-Power Design,* Apr. 1994, pp. 209–214.

[23] C. Papachristou, M. Spining, and M. Nourani, “A multiple clocking scheme for low-power RTL design,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 27–32.

[24] R. S. Martin and J. P, Knight, “Power profiler: Optimizing ASIC’s power consumption at the behavioral level,” in *Proc. Design Automation Conf.,* June 1995, pp. 42–47.

[25] V. Tiwari, S. Malik, and A. Wolfe, “Power analysis of embedded software: A first step toward software power minimization,” in *Proc. Int. Conf. Computer-Aided Design,* Nov. 1994.

[26] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools.* Norwell, MA: Kluwer Academic Publishers, 1997.

[27] J. Monteiro, P. Ashar, and S. Devadas, “Scheduling techniques to enable power management,” in *Proc. Design Automation Conf.,* June 1996, pp. 349–352.

[28] C. Papachristou, M. Spining, and M. Nourani, “An effective power management scheme for RTL design based on multiple clocks,” in *Proc. Design Automation Conf.,* June 1996, pp. 337–342.

[29] G. Tellez, A. Farrahi, and M. Sarrafzadeh, “Activity driven clock design for low-power circuits,” in *Proc. Int. Conf. Computer-Aided Design,* Nov. 1995, pp. 62–65.

[30] M. Aldina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, “Precomputation-based sequential logic optimization for low power,” *IEEE Trans. VLSI Systems,* vol. 2, pp. 426–436, Dec. 1994.

[31] V. Tiwari, S. Malik, and R Ashar, “Guarded evaluation: Pushing power management to logic synthesis/design,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 221–226.

[32] M. Favalli and L. Benini, “Analysis of glitch power dissipation in CMOS IC’s,” in *Proc. Int. Symp. Low-Power Design,* Apr. 1995, pp. 123–128.

[33] S. Rajagopal and G. Mehta, “Experiences with simulation-based schematic-level power estimation,” in *Proc. Int. Workshop Low-Power Design,* Apr. 1994, pp. 9–14.

[34] F. N. Najm and M. Y. Zhang, “Extreme delay sensitivity and the worst-case switching activity in VLSI circuits,” in *Proc. Design Automation Conf.,* June 1995, pp. 623–627.

[35] C. Lemonds and S. S. Mahant-Shetti, “A low-power 16 by 16 multiplier using transition reduction circuitry,” in *Proc. Int. Workshop Low-Power Design,* Apr. 1994, pp. 139–142.

[36] L. Benini, P. Siegel, and G. De Micheli, “Saving power by synthesizing gated clocks for sequential circuits,” *IEEE Design Test Comput.,* Winter 1994, pp. 32–41.

[37] L. Benini and G. De Micheli, “Automatic synthesis of gated-clock sequential circuits,” *IEEE Trans. Computer-Aided Design,* vol. 15, pp. 630–643, June 1996.

[38] S. Bhattacharya, S. Dey, and F. Brglez, “Performance analysis and optimization of schedules for conditional and loop-intensive specifications,” in *Proc. Design Automation Conf.,* June 1994, pp. 491–496.

[39] ——, “Effects of resource sharing on circuit delay: An assignment for clock period optimization,” *ACM Trans. Design Automat. Electron. Syst.*, vol 3, no. 2, pp. 285–307, April, 1998.

[40] ——, “Fast true delay estimation during high level synthesis,” *IEEE Trans. Computer-Aided Design*, vol 15, pp. 1088–1105, Sept. 1996.

[41] *CMOS6 Library Manual.* NEC Electronics, Inc., Princeton, NJ, Dec. 1992.

[42] *CSIM Version 5 Users Manual.* Systems LSI Division, NEC Corporation, Princeton, NJ, 1993.

[43] K. Hintz and D. Tabak, *Microcontrollers: Architecture, Implementation, and Programming.* New York, NY: McGraw Hill, 1992.

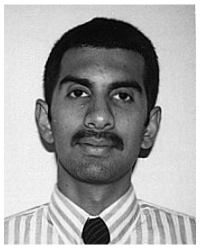
[44] “High-level synthesis benchmarks,” CAD Benchmarking Laboratory, Research Triangle Park, NC, [Online]. Available: http://WWW.cbl.ncsu.edu.

[45] D. L. Perry, *VHDL.* New York, NY: McGraw-Hill, 1991.

[46] A. Raghunathan, S. Dey, N. K. Jha, and K. Wakabayashi, “Controller re-specification to minimize switching activity in controller/data path circuits,” in *Proc. Int. Symp. Low-Power Electronics and Design,* Aug. 1996, pp. 301–304.

[47] *VARCHSYN Version 2.0 Users Manual.* Advanced CAD Development Laboratory, NEC Corporation, Princeton, NJ, Nov. 1993.

[48] *VSYSTEM Users Manual*, Model Technology, Inc., Beaverton, OR.



Anand Raghunathan (S'93–M'97) received the B.Tech. degree in electrical and electronics engineering from the Indian Institute of Technology, Madras, India, in 1992, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 1994 and 1997, respectively.

He is currently a Research Staff Member at the Computers and Communications Research Laboratories, NEC USA, Princeton, NJ, where he is involved in the research and development of high-level and system-on-chip design tools and methodologies with emphasis on low-power design, testing, and design-for-testability. He has coauthored *High-Level Power Analysis and Optimization,* (Norwell, MA: Kluwer Academic, 1998). He has presented conference tutorials on low-power design, and considering testability during high-level design. He holds or has filed for seven U.S. patents in the areas of digital testing and low-power design.

Dr. Raghunathan received the Best Paper Award at the 11th IEEE International Conference on VLSI Design (1998) and Best Paper Award nominations at the Design Automation Conference (1996 and 1997). He has served on the technical program committees of the IEEE VLSI Test Symposium (1998–1999), and the International Test Synthesis Workshop (1998–1999). He is currently Organizational Vice-Chair of the Tutorials and Education Group at the IEEE Computer Society Test Technology Technical Council, and serves as an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.

Sujit Dey (S'90–M'91), for a photograph and biography, see. p. 630 of the May 1999 issue of this TRANSACTIONS.

Niraj K. Jha (S'85–M'85–SM'93–F'98), for a photograph and biography, see. p. 281 of the March 1999 issue of this TRANSACTIONS.