Glitch Analysis and Reduction in Register Transfer Level Power Optimization

Anand Raghunathan *
Department of EE
Princeton University
Princeton, NJ 08544

Sujit Dey
C&C Research Labs
NEC USA, Inc.
Princeton, NJ 08540

Niraj K. Jha †
Department of EE
Princeton University
Princeton, NJ 08544

ABSTRACT: We present design-for-low-power techniques based on glitch reduction for register-transfer level circuits. We analyze the generation and propagation of glitches in both the control and data path parts of the circuit. Based on the analysis, we develop techniques that attempt to reduce glitching power consumption by minimizing generation and propagation of glitches in the RTL circuit. Our techniques include restructuring multiplexer networks (to enhance data correlations, eliminate glitchy control signals, and reduce glitches on data signals), clocking control signals, and inserting selective rising/falling delays. Our techniques are suited to control-flow intensive designs, where glitches generated at control signals have a significant impact on the circuit's power consumption, and multiplexers and registers often account for a major portion of the total power. Application of the proposed techniques to several examples shows significant power savings, with negligible area and delay overheads.

I. Introduction

Most savings in power consumption can be obtained through a combination of various techniques at different levels of the design hierarchy. We focus on techniques to reduce average power consumption in register-transfer level (RTL) circuits. Power estimation techniques for RTL designs, and high-level synthesis techniques for reducing power consumption have been previously investigated [1, 2]. Several studies have reported the importance of considering glitching power during power estimation and optimization [3, 4]. However, very few automated design and synthesis techniques exist for reducing glitching power consumption. At the architecture and behavior levels, previous work on power estimation and optimization ignores the effects of glitch generation and propagation across the boundaries of blocks in the architecture. While accurate library modeling approaches 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 functional units may constitute a much smaller fraction of total power than multiplexer networks and registers.

In this paper, we analyze the generation and propagation of glitches in both the control and data path parts of the circuit, and propose techniques to reduce glitch power consumption. In order to minimize the generation and propagation of glitches from control as well as data signals, we propose several techniques including restructuring multiplexer networks, clocking control signals, and inserting selective rising/falling delays. These techniques 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. In addition, they target power consumption in all parts of the design, including multiplexer networks and registers, not just functional units.

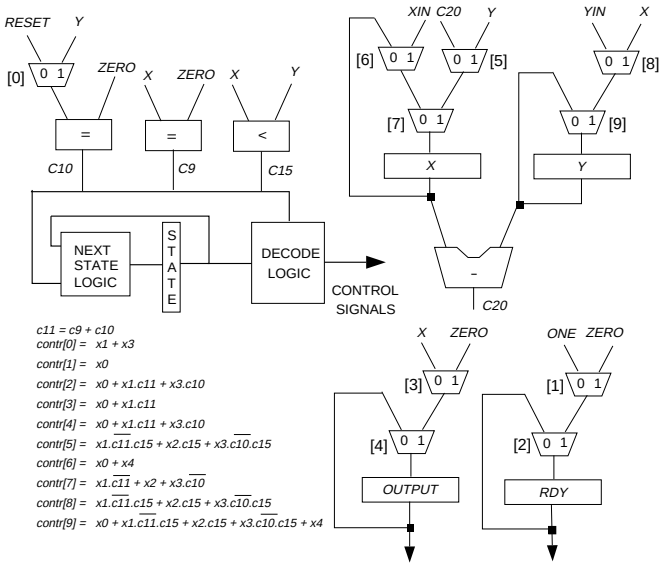


Figure 1: The RTL Architecture of the GCD circuit

II. Motivation

We motivate our work through the analysis of an example RTL circuit shown in Figure 1, which computes the *greatest common divisor* (GCD) of two numbers. The inputs are applied at XIN and YIN, and the result 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. The circuit consists of one subtractor, 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 for the decode logic are also shown in the figure. Literals $x0$ through $x4$ represent the decoded controller present state lines, while literals $c9$, $c10$, and $c15$ represent results of the three comparators.

The RTL circuit shown in Figure 1 was mapped to the NEC CMOS6 library [5]. An in-house simulation-based power estimation tool, CSIM [6], was used to measure power consumption in the various parts of the design. Table 1 provides the break up of the total power consumption into separate figures for the functional units (subtractor and three comparators), random logic (controller FSM and

* Supported by NEC C&C Research Labs

† Supported by NSF under Grant No. MIP-9319269.

DAC 96 - 06/96 Las Vegas, NV, USA ©1996 ACM, Inc. 0-89791-833-9/96/0006..$3.50

Table 1: Power consumption in various parts of the GCD circuit

Block	% of total power
Functional units	9.08%
Random Logic	4.67%
Registers	39.55%
Multiplexers	46.70%

decode logic blocks), registers, 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 specifications.

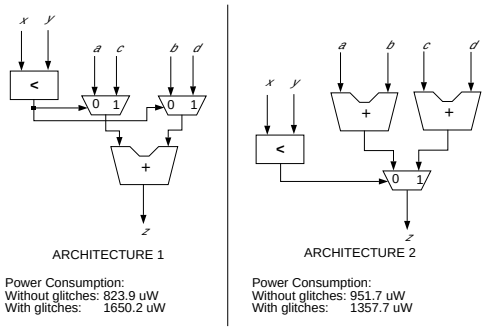
In order to get a feel for the glitching activity in the GCD circuit, we collected data on the transition activities with and without glitches in various parts of the design. Table 2 shows the total bit transitions with and without glitches for all the control signals, and selected data path signals [1]. Control signal *contr[i]* feeds the select input of the multiplexer marked *[i]* in Figure 1. Similarly, data path signal *dpi* corresponds to the output of the multiplexer marked *[i]* in Figure 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 later illustrate that control signal glitches can have a significant effect on the glitching power consumption in the rest of the circuit. We would like to point out here that while CSIM, being a discrete simulator, does not model effects like partial transitions, it does model the attenuation or suppression of glitches due to inertial delays of gates.

Table 2: Activities with/without glitches for various signals of the GCD circuit

Control signal	Activity		Datapath signal	Activity	
	Total	W/O Gl.		Total	W/O Gl.
$contr[0]$	71	70.5	$dp2[7..0]$	71.5	21.5
$contr[1]$	22	22	$dp4[7..0]$	92	26
$contr[2]$	72	20	$dp5[7..0]$	1124.5	247
$contr[3]$	42	20	$dp7[7..0]$	1044.5	273
$contr[4]$	72	20	$dp9[7..0]$	321.5	80.5
$contr[5]$	55.5	54			
$contr[6]$	22	22			
$contr[7]$	50	20			
$contr[8]$	55.5	54			
$contr[9]$	77	70.5			

The following example illustrates how ignoring glitches can be misleading and result in designs that are sub-optimal in terms of their power consumption. Consider the two RTL circuits shown in Figures 2(a) and 2(b) that implement the simple function: `if(x < y) then z = c + d else z = a + b` in two different ways. ARCHITECTURE 2 uses two adders as opposed to one adder in the case of ARCHITECTURE 1. Power estimation methods that do not take glitches into account would report that ARCHITECTURE 2, in which both adders perform computations in each cycle, consumes more power than ARCHITECTURE 1. 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. 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.

[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



(a) (b)

Figure 2: Alternate architectures that implement the same function: Effect of glitching

III. Glitch Generation in the controller and data path

In this section, we analyze the generation 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

Consider the elements shown in Figure 3 — a subtractor, an equal-to comparator, a less-than comparator, and a 3-to-1 multiplexer tree — as representative data path blocks for studying glitch generation. Each block was mapped to the technology library, and then simulated under long input sequences that consisted of random vectors. Figure 3 is annotated with the total number of bit-transitions including and excluding glitches that were observed at the output of each block. 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 Figure 3 are connected together, the glitches generated by the various blocks propagate through the following blocks, often causing an explosion in glitches and glitching power consumption.

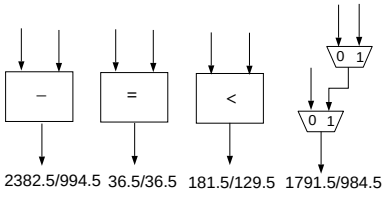
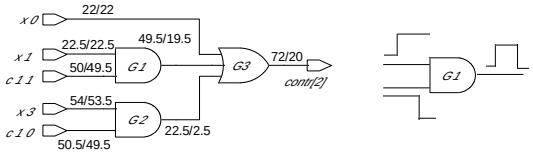


Figure 3: Glitch generation in various data path blocks

B. Glitch generation in control logic

Though the control logic itself accounts for only a small portion of the total circuit power, it plays an important role in determining the total circuit power because it is responsible for the generation of glitches at the control signals, which in turn have a significant impact on the glitching activity in the rest of the circuit. The inputs to the

decode logic (see Figure 1) are fed by the outputs of comparators and the state flip-flops of the controller. The previous subsection has already demonstrated that outputs of comparators can be glitchy. The glitches at comparator outputs can propagate through the decode logic and cause glitches on the control signals. In addition, the decode logic can itself generate a lot of glitches, as shown next. Let us focus on control signal *contr[2]* in the GCD RTL circuit, which is highly glitchy according to the statistics of Table 2. The portion of the decode logic that implements this control signal is shown in Figure 4(a). We observe that though the inputs are nearly glitch-free, significant glitches are generated at AND gates $G1$ and $G2$. After careful analysis,



(a) (b)

Figure 4: (a) Implementation of control signal *contr[2]*, and (b) generation of glitches at gate $G1$

the generation of glitches at $G1$ was attributed to two conditions that are depicted graphically in Figure 4(b):

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 $c11$.

Condition C1 arises due to the functionality of the design: most of the times when state s_1 is entered (rising transition on $x1$), comparators feeding $c9$ and $c10$ produce a 0, changing from 1 in the previous state. On the other hand, condition C2 is a result of the delay/temporal characteristics of the design. A similar explanation holds for the output of gate $G2$ being glitchy. Generation of glitches in the control logic has been described in detail in [7].

IV. Glitch Reduction Techniques

In this section, we describe our techniques for reducing glitch power consumption in RTL circuits, by minimizing the generation and propagation of glitches through different blocks of the circuit.

A. Reducing glitch propagation from control signals

As shown before, control signals to the data path can be very glitchy. Our aim is to stop glitches on control signals from propagating 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, and later integrate these techniques into a single power optimization framework.

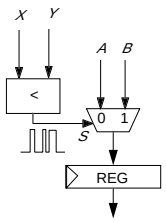
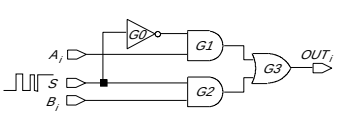


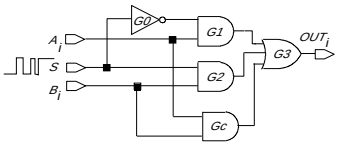
Figure 5: Example circuit used to illustrate the effect of data signal correlations on control signal glitches

Glitchy control signals and data correlations. Consider the circuit shown in Figure 5. A multiplexer selects between two 8-bit data



A	B	W/O Gl.	With Gl.
0	0	0.5	0.5
0	1	1.0	3.0
1	0	1.0	3.0
1	1	0.5	3.5

(a)



A	B	W/O Gl.	With Gl.
0	0	0.5	0.5
0	1	1.0	3.0
1	0	1.0	3.0
1	1	0.5	0.5

(b)

Figure 6: (a) Effect of data correlations on select signal glitches, and (b) use of the consensus term to reduce glitch propagation

signals, A and B, depending on whether the expression $X < Y$ evaluates to $True$ or $False$. Its output is written into a register. Suppose that the less-than comparator generates glitches at its output, and that data inputs to the multiplexer are not glitchy and settle to their final value well before the select signal settles. The glitches on the select signal of the multiplexer 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 Figure 6(a). The table shown in Figure 6(a) reports the glitches at the multiplexer output for all possible values of the data signal bits A_i and B_i. In the $< 0, 0 >$ case, glitches on select signal S are killed at AND gates $G1$ and $G2$ due to controlling side inputs that arrive early. When data inputs are $< 0, 1 >$ ($< 1, 0 >$), glitches on S propagate through gates $G2$ and $G3$ ($G1$ and $G3$). Finally, when data inputs are $< 1, 1 >$, 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. There are many ways of preventing the propagation of glitches for the $< 1, 1 >$ case. One way is to add an extra gate Gc, as shown in Figure 6(b). Gc realizes $A_i.B_i$ which is the *consensus* of $S.A_i$ and $S.B_i$. When data inputs are $< 1, 1 >$, Gc effectively kills any glitches at the other inputs of $G3$ that arrive after the output of Gc settles to a 1, as shown in the table of Figure 6(b). Maximum benefits are derived from the addition of the consensus term when the select signal is very glitchy, the data inputs arrive early compared to the select signal, and the probability of the data inputs being $< 1, 1 >$ is high.

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

Enhancing data correlations by restructuring multiplexer networks. Consider the 3-to-1 multiplexer tree shown in Figure 7(a), that feeds register $OUTPUT$ in the GCD RTL circuit. The select signals are annotated with their cumulative transition counts including and excluding glitches. Functionally, the multiplexer tree can be thought of as an abstract 3-to-1 multiplexer, as shown in Figure 7(b). The conditions under which $OUTPUT$, X and $ZERO$ are selected are represented as C_{OUTPUT}, C_X, and C_{ZERO}, respectively (which must be mutually exclusive). Select signal C_{ZERO} is observed to be glitchy, leading to propagation of glitches to the output of the first 2-to-1 multiplexer in Figure 7(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

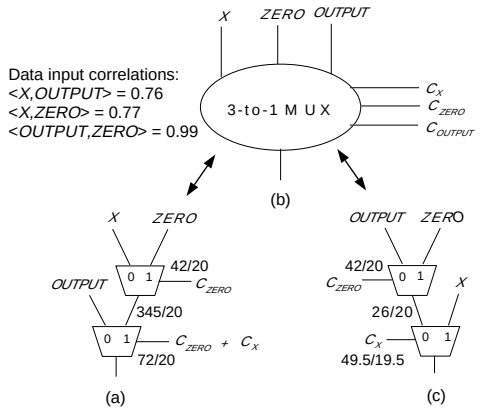


Figure 7: Multiplexer restructuring to enhance data correlations: (a) initial multiplexer network, (b) abstract 3-to-1 multiplexer, and (c) restructured network

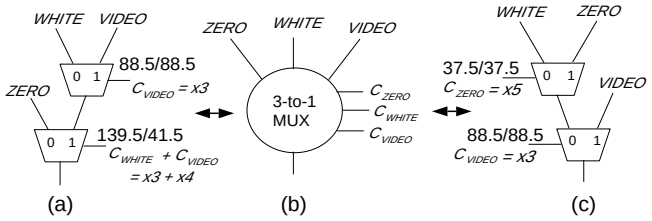


Figure 8: Eliminating glitchy control signals: (a) initial multiplexer network, (b) abstract 3-to-1 multiplexer, and (c) restructured network

tree, we transform the multiplexer tree to the implementation shown in Figure 7(c), such that the highly correlated data signals $OUTPUT$ and $ZERO$ become inputs to the first 2-to-1 multiplexer. This significantly lowers the switching activity at the output of the first 2-to-1 multiplexer to $26/20$ from $345/20$ originally.

Restructuring multiplexer networks to eliminate glitchy select signals. In order to implement an abstract n-to-1 multiplexer with n data inputs ($d_1 \ldots d_n$), and n select inputs ($c_1 \ldots c_n$) as a tree of 2-to-1 multiplexers, it can be shown that depending on the exact structure of the implementation, anywhere between $\lceil log_2 n \rceil$ and $n-1$ select expressions of the form $\bigcup_i c_i$ can be used, where $\bigcup$ represents the Boolean OR operation. It is possible that some of $c_1 \ldots c_n$ are glitchy, while others are not. Similarly, it is possible that some of the disjunctive expressions in $c_1 \ldots c_n$ are glitchy. Our aim is to restructure the multiplexer tree so that as few as possible of the select expressions used are glitchy. This concept is illustrated using the 3-to-1 multiplexer network shown in Figure 8(a) that is part of the RTL circuit implementing a `Barcode` preprocessor [8]. The select signal of the second multiplexer in Figure 8(a) ($x3 + x4$) is glitchy. An alternative implementation of the 3-to-1 multiplexer network, that does not require the use of any glitchy select signal expressions, is shown in Figure 8(c).

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 to suppress glitches on control signals. For the following example, we assume that the design is implemented using rising-edge-trigerred flip-flops and a single phase clock with a duty cycle of 50%. Consider the 2-to-1 multiplexer shown in Figure 9(a), that is part of an Unmanned Auto Vehicle controller (UAV) circuit [9]. 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 to eliminate glitchy control

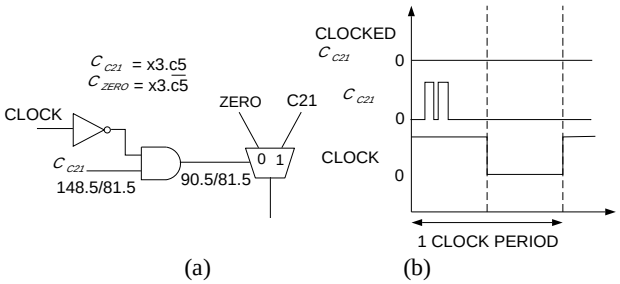


Figure 9: Clocking control signals to kill glitches: (a) multiplexer network with clocked control signal, and (b) sample waveforms

signals cannot be applied here. As shown in Figure 9(a), the original select signal is ANDed with the inverted clock to result in the *clocked select signal*. For the first half of the clock period, when the clock is high, the clocked select signal is forced to 0 in spite of the glitches on the original select signal. Figure 9(b) shows example waveforms for the clock, the original select signal and the clocked select signal. The switching activity numbers shown in Figure 9(a) demonstrate 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 $\frac{T}{2}$, where T is the clock period. This could lead to an increase in the delay of the circuit, if the control signal needs to settle to its final value before $\frac{T}{2}$ in order to meet the specified timing constraints at the circuit outputs. It should also be noted that clocking control signals may introduce extra transitions on the control signal under certain conditions. Consider a situation where the control signal remains at a steady 1 over a pair of clock cycles. By forcing the control signal to 0 in the first half of both the clock cycles, we are actually introducing extra transitions on the control signal, which can lead to increased power consumption. Thus, the scheme presented in Figure 9(b) leads to most power savings when the probability of the control signal evaluating to a 1 (signal probability) is low. On the other hand, if the signal probability of the control signal is very high, the control signal can be clocked by ORing the original control signal with the clock.

B. Minimizing glitch propagation from data signals

The previous subsection outlined several ways in which the generation and propagation of glitches from control signals can be reduced to save power. The data inputs 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 from data signals.

Glitch reduction using selective rising/falling delays. Consider the 2-to-1 multiplexer shown in Figure 10(a). Both the data inputs to the multiplexer have glitches, which propagate through the multiplexer and then through the adder, causing significant power dissipation. Consider the gate-level implementation of a bit-slice of the multiplexer as shown in Figure 10(b). Consider a pair of consecutive clock cycles c_1 and c_2 such that select signal S makes a $1 \rightarrow 0$ (falling) transition from c_1 to c_2. If this transition 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 non-controlling early, allowing the data input glitches to propagate through $G1$. This propagation can be minimized by delaying the rising transition at the output of $G0$ ($\overline{S}$), *i.e.*, by adding a "rising transition delay" to it. 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 $G2$. Since we wish to delay selected (either rising or falling, but not both) transitions at

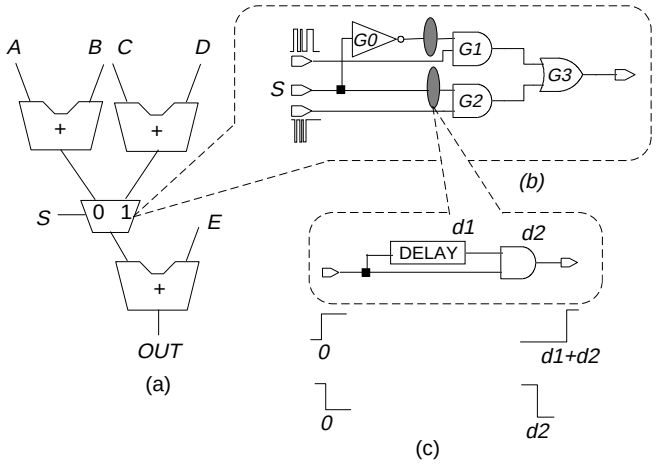


Figure 10: (a) Example circuit, (b) multiplexer bit-slice with selective delays inserted, and (c) implementation of a rising delay block

certain signals, we refer to the technique as *selective delay insertion*. The selective rising delay blocks are represented by the shaded ellipses shown in Figure 10(b). One possible implementation of a rising delay block, that uses one AND gate and a delay element, is shown in Figure 10(c). 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 $d_2\ ns$. A selective falling delay block is similar, except that the AND gate is replaced by an OR gate.

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. In addition, to ensure that the circuit delay does not increase, we insert selective delay blocks only at signals that have sufficient slack.

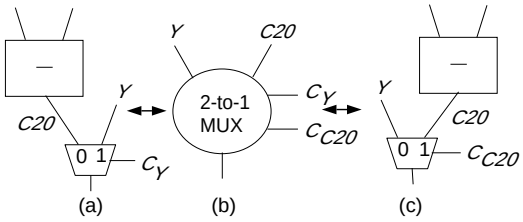


Figure 11: Using multiplexer restructuring transformations for glitchy data signals: (a) initial multiplexer network, (b) abstract 2-to-1 multiplexer, and (c) restructured network

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, that is shown in Figure 11(a). The subtractor's output, $C20$, has a lot of glitches which propagate through the multiplexer shown in the figure, and also through the logic that it feeds. Let us assume that signal Y is glitch-free. Figure 11(b) shows the equivalent abstract 2-to-1 multiplexer. We utilize the fact that there might be several instances when the value of the select signal is a *don't care* ($C_{C20}+C_Y$ is not a tautology). In the implementation of Figure 11(a), the glitchy operand $C20$ is selected in the don't care cases as well. The transformed implementation of the 2-to-1 multiplexer that is shown in Figure 11(c) ensures that the glitchy data input is selected as infrequently as possible, thus reducing

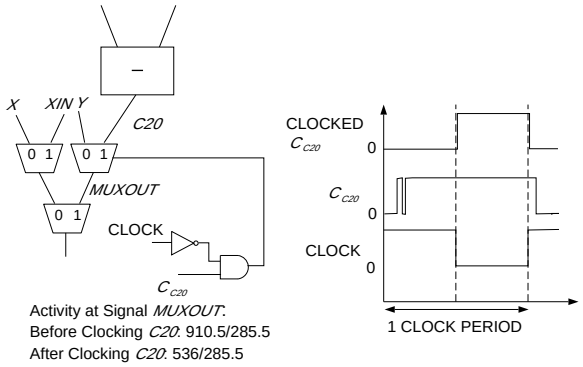


Figure 12: Clocking control signals to kill data signal glitches: (a) example circuit, and (b) sample waveforms

the propagation of glitches to the multiplexer output.

Clocking control signals to kill data signal glitches. When the techniques presented above to handle glitchy data signals are either not applicable or not adequate, we utilize the technique of clocking control signals to kill data signal glitches as well. Consider the part of the GCD circuit shown in Figure 12(a). The subtractor's output $C20$, which is glitchy, feeds the data input of a 2-to-1 multiplexer. As shown in the figure, this results in significant glitches at the output of the multiplexer. Clocking select signal C_{C20} alleviates this problem. Since the clocked select signal is forced to 0 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 subtractor's output are killed at the multiplexer for approximately the first half of the clock period. This leads to a significant 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 Figure 12(b).

C. Algorithm

The previous two subsections described the various techniques that we use to minimize the glitching power consumption in RTL circuits. Also, the conditions under which each technique is applicable and most beneficial have been stated. In this section, we give a brief overview of the order of application of the various glitch reduction techniques. The pseudo-code for power optimization procedure is shown in Figure 13. In order to apply each of the techniques, we need information about signal statistics and glitches at various signals in the circuit, including the control signals and the outputs of each RTL unit like functional unit, register, comparator, 2-to-1 multiplexer, *etc.* We first obtain an initial technology mapped gate-level implementation of the RTL circuit, and use a simulator to collect the required information.

A block in the RTL circuit is defined as a functional unit, comparator, or register, together with the multiplexer networks that feed it. We partition the RTL circuit into constituent blocks, and levelize the blocks through a single traversal starting from primary inputs or register outputs to primary outputs or register inputs. We then visit the circuit blocks in increasing order of levels (since applying glitch reduction techniques to a block can affect glitching at other blocks which are at later levels) and use applicable glitch reduction techniques at each step. Within a block, we first attempt to apply multiplexer network restructuring transformations to either eliminate glitchy select signals, or else maximize bit-level correlation between the data inputs of multiplexers whose select signals have a lot of glitches. We also selectively determine which bit-slices, if any, of each multiplexer to add the consensus term to, based on the probability of the data inputs

```
Procedure RTL_POWER_REDUCTION(RTL Circuit R)
    EXTRACT_BLOCKS(R);
    LEVELIZE_CIRCUIT(R);
    for each block B in levelized order {
        (*eliminate glitchy control signals,
        enhance data signal correlations,
        select glitchy data signals as infrequently as possible*)
        RESTRUCTURE_MUX_NETWORK(B);
        if (significant glitches remain) {
            ADD_SELECTIVE_DELAYS(B);
            CLOCK_CONTROL_SIGNALS(B);
        }
    }
```

Figure 13: Procedure Overview

taking on values of $< 1, 1 >$ and the glitchiness of the select signal. If significant glitches are present at the inputs of any RTL unit (functional unit, register, comparator, multiplexer) in the block even after the application of multiplexer restructuring transformations, we attempt to add selective rising/falling delays to multiplexers in order to reduce the propagation of glitches on data signals, or clock control signals as described in Subsections A and B in order to kill glitches on both control signals as well as data signals that feed multiplexers.

The data presented in Section II revealed that register power constituted a significant portion of the total circuit power. Upon further analysis, we found that a major portion of the register power was in turn consumed due to transitions at the clock inputs to registers. The technique of gating clocks has been used by designers to selectively turn off parts of a system. 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 [10]. However, the techniques in [10] can be applied only to the control and random logic parts of a design for which it is feasible to extract the STG. We have developed a procedure, that is based on a structural analysis of the RTL circuit, to determine the conditions under which transitions on the clock input to a register can be suppressed. Our procedure consists of identifying (structural) self-loops involving a register in the RTL circuit, and analyzing the conditions under which it is logically enabled. Gating the clock input to a register can lead to glitches on the gated clock signal, that not only cause unnecessary power consumption, but may also cause the design to function incorrectly. Our procedure ensures that glitches are not introduced while gating clock signals. Details of the procedure are provided in [7].

V. Experimental Results and Conclusions

We present results of the application of the proposed power reduction techniques to four RTL circuits implementing: `GCD`, a barcode reader preprocessor (`Barcode`) [8], the controller for an Unmanned Auto Vehicle (`UAV`) [9], and a vending machine controller (`Vend`) [11]. The initial RTL circuits were obtained by synthesizing VHDL behavioral descriptions using the SECONDS high-level synthesis system [9, 12]. Both the original and optimized RTL circuits were mapped to NEC's CMOS6 library [5], and evaluated for area and delay using the logic synthesis system VARCHSYN [13], and for power consumption using the simulation based power estimation tool, CSIM [6]. The vectors used for simulation were obtained for each design by simulating the scheduled behavioral description with a test bench written to generate typical input cases, using a VHDL simulator, to result in a cycle-by-cycle input vector trace. The above step is important for control-flow intensive designs where the number of clock cycles required to perform the computation varies depending on the input values. Table 3 reports the results of our experiments. The power, area (# of transistor pairs), and delay numbers are obtained after mapping to the technology library used.

Table 3: Experimental Results

Circuit	Original			Optimized			Pow Red (%)
	Pow (mw)	Area	Del (ns)	Pow (mw)	Area	Del (ns)	
`GCD`	8.74	1037	32.3	7.23	1034	31.9	17.27
`Bar-code`	9.41	1945	49.7	7.77	1968	47.3	17.42
`UAV`	10.89	1954	83.5	8.02	1967	83.2	26.25
`Vend`	10.47	1595	70.1	7.72	1617	71.6	26.22

The results shown in Table 3 demonstrate that our glitch reduction techniques can significantly reduce power consumption in RTL circuits. Note that these techniques target power reduction solely by reducing the propagation of glitches between various blocks in the RTL circuit. Hence, they can be combined with other power reduction techniques that attempt to suppress transitions that do not correspond to glitches. The area and delay overheads incurred by our power reduction techniques can be seen to be nominal. In some cases, the area and delay are slightly reduced due to the fact that multiplexer restructuring transformations can lead to a simplification in control logic.

References

[1] J. Rabaey and M. Pedram (Editors), *Low Power Design Methodologies*. Kluwer Academic Publishers, Boston, MA, 1996.

[2] M. Pedram, "Power minimization in IC design: principles and applications," *ACM Trans. Design Automation of Electronic Systems*, vol. 1, Jan. 1996.

[3] M. Favalli and L. Benini, "Analysis of glitch power dissipation in CMOS IC's," in *Proc. Int. Symp. Low Power Design*, pp. 123–128, Apr. 1995.

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

[5] *CMOS6 Library Manual*. NEC Electronics, Inc., Dec. 1992.

[6] *CSIM Version 5 Users Manual*. Systems LSI Division, NEC Corp., 1993.

[7] A. Raghunathan, S. Dey, and N. K. Jha, "Register-transfer-level power optimization techniques with emphasis on glitch analysis and optimization," Tech. Rep., NEC C&C Research Labs, Princeton, NJ, Oct. 1995.

[8] High-level synthesis benchmarks, CAD Benchmarking Laboratory, Research Triangle Park, NC. Benchmarks can be downloaded anonymously from http://www.cbl.ncsu.edu.

[9] S. Bhattacharya, S. Dey, and F. Brglez, "Clock period optimization during resource sharing and assignment," in *Proc. Design Automation Conf.*, pp. 195–200, June 1994.

[10] L. Benini, P. Siegel, and G. DeMicheli, "Saving power by synthesizing gated clocks for sequential circuits," *IEEE Design & Test of Computers*, pp. 32–41, Winter 1994.

[11] D. L. Perry, *VHDL*. New York, NY 10020: McGraw-Hill, 1991.

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

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