GlitchLess: Dynamic Power Minimization in FPGAs Through Edge Alignment and Glitch Filtering

Julien Lamoureux, Guy G. F. Lemieux, and Steven J. E. Wilton

***Abstract*—This paper describes GlitchLess, a circuit-level technique for reducing power in field-programmable gate arrays (FPGAs) by eliminating unnecessary logic transitions called glitches. This is done by adding programmable delay elements to the logic blocks of the FPGA. After routing a circuit and performing static timing analysis, these delay elements are programmed to align the arrival times of the inputs of each lookup table (LUT), thereby preventing new glitches from being generated. Moreover, the delay elements also behave as filters that eliminate other glitches generated by upstream logic or off-chip circuitry. On average, the proposed implementation eliminates 87% of the glitching, which reduces overall FPGA power by 17%. The added circuitry increases the overall FPGA area by 6% and critical-path delay by less than 1%. Furthermore, since it is applied after routing, the proposed technique requires little or no modifications to the routing architecture or computer-aided design (CAD) flow.**

***Index Terms*—Field-programmable gate arrays (FPGAs), low-power, switching activity minimization.**

I. Introduction

With POWER dissipation of field-programmable gate arrays (FPGAs) increasing each generation, power reduction is quickly becoming the main challenge for implementing large applications. FPGAs dissipate significantly more power than application-specific integrated circuits (ASICs) because of the added circuitry needed to make them programmable. Although static power dissipation has received significant attention recently due to its sharp increase, dynamic power still accounts for 62% of total power [1].

There are a number of ways to reduce dynamic power in FPGAs. Techniques that can be used at the physical level include lowering the supply voltage [2] or increasing the threshold voltage [3]. At the circuit level, device features can be sized less aggressively for speed to reduce capacitive loading and therefore dynamic power [4]. At the architecture level, power management [5] and clock network design are also helpful [6]. At the computer-aided design (CAD) level, grouping together high-activity logic reduces dynamic power [7]. A summary of techniques to reduce power is described in [8].

This paper introduces *GlitchLess*, a circuit-level technique that reduces dynamic power in FPGAs by actively preventing each logic output from toggling until all of its inputs have fully resolved. Although there are a few possible implementations, the one explored in this paper adds programmable delay elements to the configurable logic blocks (CLBs). These delay elements programmably align the arrival times of early-arriving signals to the inputs of the lookup tables (LUTs) to prevent the generation of glitches. Additionally, the delay elements also behave as filters that eliminate other glitches generated by upstream logic or off-chip circuitry. Since it is applied after routing, GlitchLess requires little or no modifications to the FPGA routing architecture or CAD flow. Furthermore, it can be combined with other low-power techniques.

In theory, GlitchLess offers the potential to *eliminate all glitching* in FPGAs, thereby saving significant amounts of power. In practice, however, we must tradeoff the power saved with the area and speed overhead incurred by the additional circuitry required to implement it. Fortunately, the impact on circuit speed is not significant (other than increased parasitic capacitance) because only the early-arriving signals need to be delayed. However, the programmable delay elements do consume chip area, so we should expect a modest increase in the area of the device. The tradeoff between glitch reduction (hence, power), area, and speed will be quantified in this paper. Specifically, this paper examines the following questions.

1) How should the programmable delay elements be connected to the logic? The programmable delay elements could conceivably be connected to the inputs or outputs of each CLB or they could be connected to the inputs of the LUTs within the CLBs.
2) How many programmable delay elements are needed within each CLB? Intuitively, adding more programmable delay elements to the CLBs eliminates more glitches since more signals can be aligned; however, it also increases the area overhead.
3) How flexible should the programmable delay elements be? The more flexible each delay element is, the better it will be able to align the arrival times of signals. However, there is a tradeoff between this flexibility and the area overhead of the added circuits.
4) Does the delay insertion technique work when there is process, voltage, and/or temperature (PVT) variation? PVT affects both the delay of the existing FPGA logic and the delay of the programmable delay elements. Special measures must be taken to ensure that the delay insertion technique can tolerate variation well enough to eliminate glitches without introducing timing violations.

A preliminary version of this work appeared in [9]. This paper introduces a new delay insertion scheme which reduces

Manuscript received January 23, 2007; revised August 20, 2007. Current version published October 22, 2008. This research was supported in part by Altera and the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: lemieux@ece.ubc.ca; julienl@ece.ubc.ca; stevew@ece.ubc.ca).

Digital Object Identifier 10.1109/TVLSI.2008.2001237

the area overhead and proposes new techniques and a new programmable delay element that tolerates PVT variation more effectively than the previous work.

This paper is organized as follows. Section II presents terminology used in this paper to describe glitching and PVT variation and then summarizes existing techniques that can be used to minimize glitching. Section III examines how much glitching occurs within FPGAs. Section IV presents the proposed delay insertion schemes. Section V describes the experimental framework used to estimate power savings and area and delay overhead. Section VI describes how each scheme is calibrated and Section VII presents the overall power savings and overhead. Finally, Section VIII summarizes the results and presents our conclusions.

II. Background

A. Switching Activity Terminology

There are two types of transitions that can occur on a signal. The first type is a functional transition, which is necessary in order to perform a computation. A functional transition causes the value of the signal to be different at the end of the clock cycle than at the beginning of the clock cycle. In each cycle, a functional transition occurs either once or the signal remains unchanged. The second type of transition is called a glitch (or a hazard) and is not necessary in order to perform a computation. These transitions can occur multiple times during a clock cycle.

B. PVT Variation

Process variation refers to manufacturing imprecision, leading to variability in characteristics like device geometry or even placement and concentration of dopant atoms. Similarly, *voltage variation* refers to the variability of the power supply and *temperature variation* refers to variability of the temperature of the surrounding environment. Collectively, these are called *PVT variation*. Variations can either be *die-to-die* (different dies have different properties) or *within-die* (similar circuit elements within the same chip have different properties). In either case, variations can affect both the timing and power dissipation of the devices.

C. Existing Glitch Minimization Techniques

Several techniques to reduce power have been proposed, including logic decomposition [10], loop folding [11], high-level compiler optimization [12], technology mapping [13], and clustering [7]. These techniques reduce switching activity, which eliminates some glitching, but they typically incur area and delay penalties as they reorganize the structure of the circuit. Another approach to reduce glitches adds flip-flops (pipelining) [14] to reduce the combinational path length. However, this increases the latency of the circuit. To preserve latency, alternatives include adding flip-flops that use the opposite (e.g., negative) clock edge [15] or relocating the flip-flops by retiming [16]. The *gate freezing* technique [17] suppresses 1-0 transitions on selected gate outputs using an nMOS footer controlled by a fixed-delay circuit. Similarly, a delay insertion technique described in [18] reduces glitching by aligning the input arrival times of gates using delay elements with a fixed delay. These last two techniques are applied to ASIC-style circuits where the location and amount of delay to insert can be tailored for each circuit by analyzing which nodes have high activity and large capacitance. However, these techniques are not suitable for FPGAs since the applications are not known until *after* fabrication, making it is impossible to determine, at fabrication time, where the extra delay circuitry should be located or how much delay to add. In this paper, we target FPGAs by adding *programmable* delay elements to the architecture. The design and location of these elements must be considered carefully, since their overhead can overwhelm any power savings obtained from glitch removal.

III. Glitching in FPGAs

This section begins with a breakdown of functional versus glitching activity to determine how much glitching occurs within FPGAs. It then examines the width of typical glitches and determines how much power is dissipated by a single glitch. Finally, it indicates how much power could be saved if glitching could be completely eliminated. These statistics are important, not only because they help motivate our work, but also because they provide key numbers (such as typical pulse widths) that will be needed in Section VI when the delay insertion schemes are calibrated.

A. Switching Activity Breakdown

Table I reports the switching activities for a suite of benchmark circuits implemented on FPGAs. These activities are gathered using gate-level simulation of a post-place and route implementation for a set of benchmark circuits (see Section V for more details). Gate-level simulations provide the functional and total activity; the glitching activity is computed as the difference between these two quantities. In general, the amount of glitching is greater in circuits with many levels of logic, uneven routing delays, and exclusive-or logic. As an example, C6288 is an unpipelined 16-bit array multiplier that has four times more glitch transitions than functional transitions.

B. Pulse Width Distribution

In FPGAs, glitches are generated at the output of a LUT when the input signals transition at different times. The *pulse width* of these glitches depends on how uneven the input arrival times are. Intuitively, we would expect FPGA glitches to be wider than ASIC glitches since FPGA interconnect introduces larger delays. Fig. 1 plots the pulse width distribution of the C6288 circuit. The distribution was obtained using event-driven simulation and delays from the versatile place and route (VPR) tool [19], as described in Section V. The graph shows that the majority of glitches have a pulse width between 0 and approximately 10 ns. Although this range varies across the benchmark circuits, we have found that the shape of the distribution is similar for every circuit.

C. Power Dissipation of Glitches

The parasitic resistance and capacitance of the routing resources filters out very short glitches. To measure the impact of

TABLE I
BREAKDOWN OF SWITCHING ACTIVITY

Circuit	Logic Depth	Activity	Func. Activity	Glitch Activity	% Glitch
C1355	4	0.32	0.23	0.09	27.5
C1908	10	0.26	0.17	0.09	34.6
C2670	7	0.27	0.21	0.06	22.2
C3540	12	0.42	0.23	0.19	45.2
C432	11	0.26	0.18	0.08	29.3
C499	4	0.34	0.23	0.11	31.9
C5315	10	0.40	0.25	0.15	36.7
C6288	28	1.56	0.29	1.27	81.1
C7552	9	0.39	0.23	0.16	42.0
C880	9	0.23	0.19	0.05	19.8
alu4	7	0.08	0.07	0.01	13.1
apex2	8	0.05	0.04	0.01	13.7
apex4	6	0.04	0.03	0.01	32.3
des	6	0.27	0.17	0.10	36.8
ex1010	8	0.03	0.01	0.02	52.9
ex5p	7	0.17	0.08	0.09	51.0
misex3	7	0.06	0.05	0.01	20.9
pdc	9	0.03	0.02	0.01	31.8
seq	7	0.05	0.04	0.01	16.0
spla	8	0.05	0.03	0.02	42.7
Geomean	**8.1**	**0.024**	**0.019**	**0.047**	**30.8**

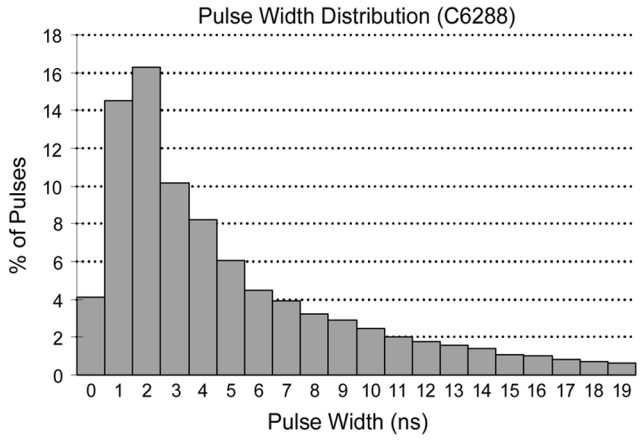


Fig. 1. Pulse width distribution of glitches.

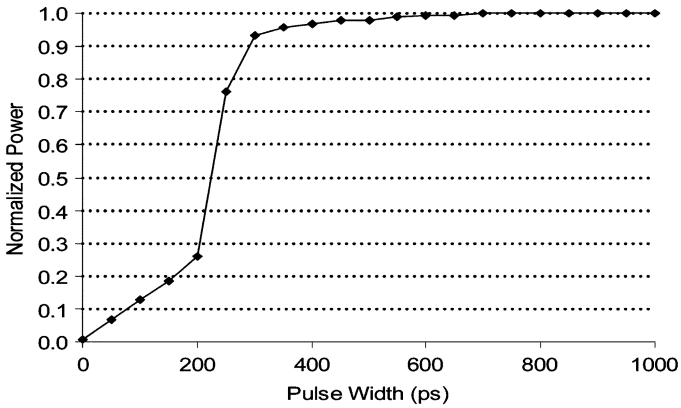


Fig. 2. Normalized power versus pulse width.

this, HSPICE was used to determine power with respect to pulse width. Fig. 2 illustrates the relative power dissipated when pulse

TABLE II
FPGA POWER WITH AND WITHOUT GLITCHING

Circuit	Power (mW)		% Difference
	With Glitching	Without Glitching	
C1355	9.5	6.7	28.8
C1908	6.2	4.9	21.1
C2670	21.5	18.6	13.4
C3540	21.3	14.6	31.7
C432	4.6	3.8	17.1
C499	8.7	5.7	34.6
C5315	34.7	26.8	22.8
C6288	41.6	11.2	73.1
C7552	39.9	29.8	25.5
C880	5.8	5.3	9.6
alu4	39.2	37.8	3.6
apex2	41.2	39.4	4.3
apex4	24.5	22.0	10.1
des	88.2	72.4	17.9
ex1010	51.4	41.9	18.4
ex5p	29.7	21.4	28.1
misex3	41.6	38.3	8.1
pdc	35.8	31.0	13.3
seq	38.3	36.0	6.1
spla	45.5	35.8	21.4
Geomean	**24.3**	**18.8**	**22.6**

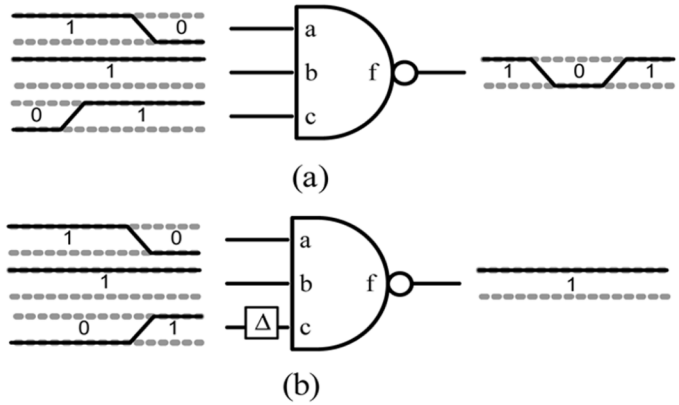


Fig. 3. Delaying early-arriving signal removes glitch.

widths ranging from 0 to 1 ns are applied to an FPGA routing track that spans four CLBs. A 180-nm process was assumed.

The graph illustrates that pulses less than or equal to 200 ps in duration are mostly filtered out by the routing resources. All pulses that are 300 ps or longer in duration dissipate approximately the same amount of power. Thus, if the input signals of a gate arrive within a 200-ps window, the glitching of that gate is effectively eliminated.

D. Potential Power Savings

Table II reports the average total power dissipated by circuits when implemented in an FPGA. The second column reports the power of the circuits in the normal case, when glitching is allowed to occur. The third column reports the power in the ideal case, when glitching is eliminated with no overhead. The fourth column shows the percent difference between the two; this number indicates how much power could be saved if glitching was completely eliminated without any overhead.

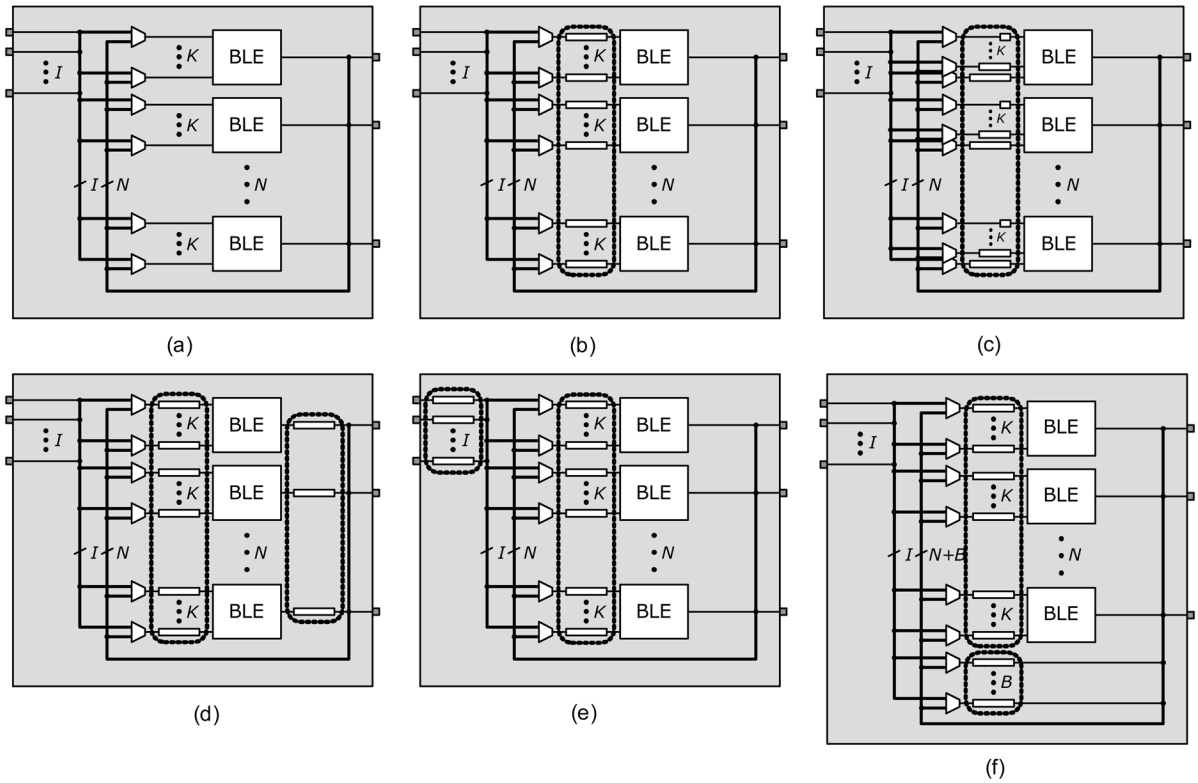


Fig. 4. Delay insertion schemes. Original VPR logic block. (b) Scheme 1: LUT inputs. (c) Scheme 2: Gradual LUT inputs. (d) Scheme 3: LUT inputs + outputs. (e) Scheme 4: CLB and LUT inputs. (f) Scheme 5 LUT inputs + bank.

Depending on the circuit, the potential power saving ranges between 4% and 73%, with average savings of 22.6%. These numbers motivate a technique for reducing glitching in FPGAs.

IV. Glitch Elimination

This section describes the techniques used in this paper to eliminate glitching. It begins by describing our proposed technique and discusses other possible techniques as well. It then presents five variations (or schemes) of the proposed technique, which employ delay elements in different locations within the FPGA logic blocks. It then describes the programmable delay element that is used to align the arrival times and the CAD algorithms that are used to configure these programmable delay elements. Finally, it describes techniques that can be used to make programmable delay insertion more tolerant to PVT variation.

A. Glitch Elimination Techniques

Our proposed technique involves adding programmable delay elements to the CLBs of the FPGA. Within each CLB, the programmable delay elements are configured to delay early-arriving signals so as to align the arrival times on each LUT input to eliminate glitching. The technique is shown in Fig. 3; by delaying input c, the output glitch can be eliminated. Note that the overall critical-path of the circuit is not increased since only the early-arriving inputs are delayed.

Another technique that we considered involved modifying the placement and routing algorithms to be glitch-aware. By placing CLBs at even distances from common sources and/or routing connections to balance arrival-times, the amount of glitching could likely be reduced. The inherent problem with this approach is that it is difficult to balance arrival-times by making the late-arriving fan-ins faster since the CAD algorithms have already been optimized to minimize critical-path delay. The other alternative is to balance arrival-times by making the early-arriving signals slower. This approach; however, would not minimize power as efficiently as the proposed technique since the routing resources, which would effectively be used to add delay to early arriving signals, dissipate more dynamic power than the proposed programmable delay element, which uses a large resistance (as opposed to capacitance) to delay signals.

B. Architectural Alternatives

We consider five alternative schemes for implementing the delay insertion technique; the schemes differ in the location of the delay elements within the CLB. Fig. 4(a) illustrates the baseline CLB. A CLB consists of LUTs, flip-flops, and local interconnect. The LUTs and FFs are paired together into basic logic elements (BLEs). Three parameters are used to describe a CLB: I specifies the number of input pins, N specifies the number of BLEs and output pins, and K specifies the size of the LUTs. The local interconnect allows each BLE input to choose from any of the I CLB inputs and N BLE outputs. Each BLE output drives a CLB output. The five schemes we consider for adding delay elements to a CLB are illustrated in Fig. 4(b)–(f), each of which are described in the following.

In Scheme 1, the programmable delay elements are added at the input of each LUT, as shown in Fig. 4(b). This architecture allows each LUT input to be delayed independently.

We describe the architecture using three parameters: *min_in, max_in*, and *num_in*. The *min_in* parameter specifies the precision of the delay element connected to the LUT inputs. Intuitively, more glitching can be eliminated when *min_in* is small since the arrival times can be aligned more precisely. On the other hand, there is more overhead when *min_in* is small since each programmable delay element requires more stages to provide the extra precision. The *max_in* parameter specifies the maximum delay that can be added to each LUT input. Intuitively, more glitching can be eliminated when *max_in* is large since wider glitches can be eliminated. However, there is more overhead when *max_in* is large. Finally, the *num_in* parameter specifies how many LUT inputs have a programmable delay element, between 1 and K (the number of inputs in each LUT). Increasing *num_in* reduces glitching but increases the overhead. In Section VI, we quantify the impact of these parameters on the power, area, and delay of this scheme.

The disadvantage of Scheme 1 is that, since some inputs need very long delays for alignment, large programmable delay elements are required. Since *num_in* delay elements are needed for every LUT, this technique has a high area overhead if *num_in* is large. In Scheme 2, shown in Fig. 4(c), the programmable delay elements are in the same location as Scheme 1; however, the maximum delay of the elements is gradually decreased for each LUT input (by a factor of 0.5). Intuitively, the arrival times of the inputs most likely vary with one another; therefore the area overhead can be reduced by reducing the maximum delay of some of the delay elements without a significant penalty on glitch reduction. The same parameters used to describe Scheme 1 are used to describe Scheme 2, with *max_in* specifying the maximum delay of the largest delay element.

In Scheme 3, shown in Fig. 4(d), additional programmable delay elements are added to the outputs of LUTs (we refer to these new delay elements as *LUT output delay elements*). With this architecture, a single LUT output delay element could be used to delay a signal that fans out to several sinks, potentially reducing the size and the number of delay elements required at each LUT input. We describe the LUT output delay elements using two parameters *min_out* and *max_out*, which specify the minimum and maximum delay of the output delay elements. The LUT input delay elements are described using the same parameters as Scheme 1.

Scheme 4, shown in Fig. 4(e), is another way to reduce the area required for the LUT input delay elements. Here, additional delay elements, which we call *CLB input delay elements*, are added to each of the I CLB inputs. Since there are typically fewer CLB inputs than there are LUT inputs in a CLB, this could potentially result in an overall area savings. The parameters *min_c* and *max_c* specify the minimum and maximum delay of the CLB input delay elements. We assume every CLB input has a delay element, in order to maintain the equivalence of each CLB input.

Finally, Scheme 5, shown in Fig. 4(f), reduces the size of the LUT input delay elements by adding a bank of delay elements which can programmably be used by all LUTs in a CLB. We refer to these delay elements as *bank delay elements*. Signals that need large delays can be delayed by the bank delay elements, while signals that need only small delays can be delayed by the LUT input delay elements. In this way, the LUT input delay elements can be smaller than they are in Scheme 1. These bank delay elements are described using two additional parameters: *max_b* and *num_b*. The *max_b* parameters specify the maximum delay of the bank delay elements and the *num_b* parameter specifies the number of programmable delay elements in the bank. Note that we assume that the minimum delay of the bank delay element is equal to the maximum delay of the LUT input delay element since only one of delay elements needs to add precision.

TABLE III
DELAY INSERTION PARAMETERS

Scheme	Parameter	Meaning
1-5	*min_in*	Min delay of LUT input delay element
	max_in	Max delay of LUT input delay element
	num_in	# of LUT input delay elements / LUT
2	*max_in**	Max delay of LUT input delay element (gradually decreases by 50% for each input)
3	*min_out*	Min delay of LUT output delay element
	max_out	Max delay of LUT output delay element
4	*min_c*	Min delay of CLB input delay element
	max_c	Max delay of CLB input delay element
5	*max_b*	Max delay of bank delay element
	num_b	# of bank delay elements / CLB

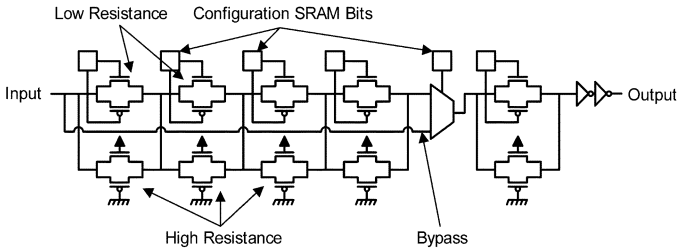


Fig. 5. New programmable delay element.

Table III summarizes the parameters used to describe each scheme. The area and delay overhead for each scheme, as well as their ability to reduce glitches, will be quantified in Sections VI and VII.

C. Programmable Delay Element

Fig. 5 illustrates an example of the programmable delay element used in each of the delay insertion schemes. The circuit has multiple delay stages (five in this example), each consisting of two transmission gates and an SRAM cell. Each stage has a fast and a slow mode, which is controlled by the value stored in that SRAM cell. In the slow mode, the signal must pass through the slow transmission gate, consisting of pass-transistors with long channel lengths. In the fast mode, the signal is allowed to pass through fast a transmission gate consisting of a minimum sized transistor. By approximately doubling the resistance

```
calc_needed_delays (circuit) {
  // in topological order beginning from the primary inputs
  foreach node n ∈ circuit {
    Arrival_Time(n) = 0.0;
    foreach fanin f ∈ n
      if (Arrival_Time(f) + Delay(n, f) > Arrival_Time(n))
        Arrival_Time(n) = Arrival_Time(f) + Fanin_Delay(n, f);
    }
    foreach node n ∈ circuit {
      foreach fanin f ∈ n
        Needed_Delay(n, f) = Arrival_Time(n) -
          Arrival_Time(f) - Fanin_Delay(n, f);
  }}}
```

Fig. 6. Calculating the delay needed to align the inputs.

```
scheme1 (circ, min_in, max_in, num_inl)
{
  config_LUT_input_delays (circ, min_in, max_in,
                           num_in);
}
config_LUT_input_delays (circ, min_in, max_in, num_in) {
  foreach LUT n ∈ circ {
    count = 0;
    foreach fanin f ∈ n {
      if (Needed_Delay(n, f) > min_in &&
        Needed_Delay(n, f) ≤ max_in && count < num_in)
      {
        Needed_Delay(n, f) = Needed_Delay(n, f) –
          min_in * floor(Needed_Delay(n, f) / min_in);
        count++;
}}}}
```

Fig. 7. Assigning delays for Scheme 1.

of each successive stage, the circuit can be configured using n bits to produce one of 2^n different delay values with even increments. Specifically, the circuit can be configured to produce any delay $\Delta. \in \{k, \tau+k, 2\tau+k, 3\tau+k, \ldots, (2^n-1)\tau+k\}$, where τ is the minimum delay increment and k is the delay produced by the (nonzero) bypass resistances and the inverters. Note that this binary approach is more efficient than a straightforward linear arrangement of equal-delay elements since it requires significantly less multiplexing to select the needed delay.

In addition to n delay stages, the programmable delay element has a 2-to-1 multiplexer and a buffer. The multiplexer is required to bypass the first $n-1$ stages when a very small delay is needed. Without this, the minimum delay of the circuit (k) would be too large. The buffer consists of two inverters with long channel lengths to minimize short-circuit power.

This is the circuit we use to obtain the area, power, and delay overhead for the proposed delay insertion technique. The programmable circuit produces the required delays and careful consideration was taken to minimize the area and power dissipation of the circuit. This being said, there are likely other circuit-level techniques that can be used to align input edges and filter glitches that may be even more efficient. Our main goal is to validate the overall technique and to give a reasonable account of the tradeoffs between power savings and area/delay overhead.

D. CAD Algorithms

This section describes the algorithms used to determine the configuration of each programmable delay element. This configuration occurs *after* placement and routing, when accurate delay information is available.

For all architecture schemes, the quantity *Needed_Delay* is first calculated for each LUT input using the algorithm in Fig. 6. This quantity indicates how much delay should be added to the LUT input so that all LUT inputs transition at the same time. Since the LUT inputs can have different speeds, the delay difference accounted for when the arrival times are calculated. Specifically, the *Fanin_Delay*(n, f) value represents the propagation delay from *f to n*, including both the interconnect delay and the precise logic delay determined from a detailed timing analysis.

The next step is to implement a delay as close to *Needed_Delay* as possible for each LUT input. In all but the first scheme, signals can be delayed in more than one way. Hence, the technique used to determine and to implement the needed delay for each scheme is different.

The algorithm used to calculate the configuration of each LUT input delay element in Scheme 1 is shown in Fig. 7. In this case, there is only one way to insert delays, so the algorithm is straightforward. Note that the granularity of the delay elements (*min_in*) and the number of delay elements attached to each LUT (*num_in*) will affect how closely the inserted delays match the desired values (determined by the algorithm described in Fig. 6).

The algorithm for Scheme 2 is similar to the algorithm for first scheme except that it begins by sorting the delay elements and the fan-ins based on delay. Both are sorted to ensure that the fan-ins that need small delays use the smaller delay elements, which leaves the larger delay elements to the fan-ins that need larger delays.

The algorithm for Scheme 3 first visits each LUT in topological order from inputs to outputs and determines the minimum delay needed by all the fan-outs of that LUT. It then configures the output delay element to match this delay and then updates the needed delay value of each fan-out. It then configures the LUT input delays as in Scheme 1. Similarly, the algorithm for Scheme 4 first visits each CLB input to determine the minimum delay needed by the LUT inputs that are driven by that input. After configuring each CLB input delay element, it then updates the needed delay of the affected LUT inputs to reflect the change and then configures the LUT input delays as in Scheme 1.

Finally, the algorithm for Scheme 5, which incorporates a bank of programmable delay elements in addition to those at the LUT inputs, first visits each CLB in the circuit and configures the bank circuits to delay signals that need to be delayed by more than *max_in* and smaller or equal to *max_b*. When the algorithm finds a signal that requires a delay that is greater than *max_in*, it calculates the amount of delay that it can add to a signal (by a delay element in the bank) and then updates the needed delay for the subsequent LUT input algorithm.

E. PVT Variation Techniques

PVT variations can have a significant impact on circuit delays, which is problematic for the proposed delay insertion technique. Our technique requires accurate estimates of path delays in order to calibrate the programmable delay elements. If the estimates are not accurate, and the delay elements are not configured properly, they may be ineffective at reducing

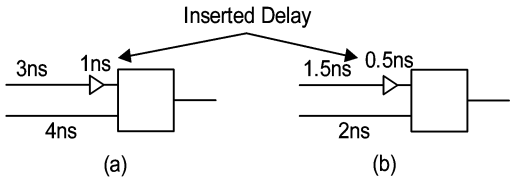


Fig. 8. Inserted delays must scale with remaining delays.

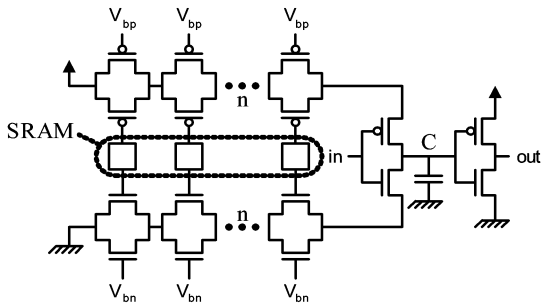


Fig. 9. Schematic of programmable delay element [9].

glitches. Techniques for minimizing the effect of both die-to-die and within-die PVT variation on the proposed delay insertion technique are described in the following.

1) Die-to-Die Variation: Die-to-die variation occurs when circuits on different chips have different delay properties. A common practice used by FPGA vendors to deal with variation is *speed binning*, which involves grouping a product based on the maximum speed of that product. Because of PVT variation, some FPGAs are faster than other FPGAs. Grouping the FPGAs into different speed bins allows the vendors to sell FPGAs with different speed grades. This practice tends to reduce die-to-die variation for FPGAs within each speed bin which improves the feasibility of the proposed technique.

Although speed-binning can help reduce the die-to-die variations, this may not be sufficient to provide the accuracy required to obtain significant power savings. Within a speed grade, we can tolerate variations if the programmable delay element is designed to react the same way as the existing FPGA logic and routing resources. As an example, consider an input signal that arrives 1 ns before the slowest input under normal conditions, as illustrated in Fig. 8(a). In order to eliminate glitches, the corresponding programmable delay element would be configured to add 1 ns to that input. Now, consider some variation that causes that same input to arrive only 0.5 ns before the slowest input [see Fig. 8(b)]. In this case, adding 1 ns would be too much and possibly cause a timing violation. However, if the programmable delay element is affected the same way as the remaining circuitry, the added delay would actually be 0.5 ns, producing the desired effect.

For this to be effective, PVT variation must affect the delay of the programmable delay element in the same way as the existing FPGA routing and logic circuitry. In the remainder of this section, we show that this is *not* true in the delay element presented in prior work, however, it is partially true in the delay element presented in Section IV-A.

First, consider the delay element proposed in [9]. The circuit, which is illustrated in Fig. 9, is composed of two inverters. The first inverter has programmable pull-up and pull-down resistors to control the delay of the circuit. The second inverter has large channel lengths to minimize short-circuit power. The pull-up and pull-down resistors of the first inverter have n stages. Each stage has a resistor and a bypass transistor controlled by an SRAM bit. The resistor in each stage consists of a pass-transistor that is only partially turned on (though biasing) to produce a large resistance.

The circuit has two major drawbacks related to variation. The first drawback is that is uses gate biasing to produce the large resistances. As we will show in the following, this tends to react differently to variation compared to the existing FPGA circuitry. The second drawback is that, since the nMOS and pMOS transistors can react differently to variation, the rise and fall times of the delay element become unbalanced when there is variation. This is less of a concern in conventional buffers and logic gates which also use pMOS pull-up networks and nMOS pull-down networks, since the effect is reduced when gates are cascaded.

To illustrate these effects, Fig. 10 shows the rise and fall times of the programmable delay element for every possible delay configuration. For the black, white, and grey bars, the X-axis represents intended delay and the Y-axis represents actual delay. Results from three experiments are shown. The white bars are the delays of the programmable delay element simulated in HSPICE assuming typical-typical (TT) process parameters. Similarly, the grey and black bars are the delays assuming slow-slow (SS) and fast-fast (FF) process parameters, respectively.

In addition to the programmable circuit delays, the graphs also include lines that show the effect of process variation on the delay of the existing FPGA routing resources, which were obtained by simulating a chain of buffered routing resources as described in [19] in HSPICE. For the black and white lines, the x-axis represents the delay of the FPGA routing resources when typical-typical (TT) process parameters are assumed and the y-axis represents the delay of the same resources when other process parameters are assumed. Specifically, the black line (SS-Routing) indicates the delay of the FPGA routing assuming SS process parameters and the white line (FF-Routing) indicates the delay of the FPGA routing assuming FF parameters.

The two graphs in Fig. 10 highlight the drawbacks described previously. In the first graph, the rise times are less affected by process variations than are the FPGA routing circuitry. In the second graph, however, the fall times are more affected by process variations. On average, the fall times assuming the FF process corner is 47% faster than TT values, while the fall times assuming the SS process corner are 137% slower. The process variation has a greater impact on the fall times than the rise times of this delay element because it changes the effective on-resistance of the biased nMOS transistors in the pull-down network more than the biased pMOS transistors in the power-up network.

Now consider the new programmable delay element described in this paper (in Section IV-C). In this circuit, nMOS and pMOS transistors were used in parallel in order to average out their response to variation. The rise times of the new delay circuit are shown in Fig. 11. Similar results were obtained for the fall time. On average, the actual delays are 19% faster and

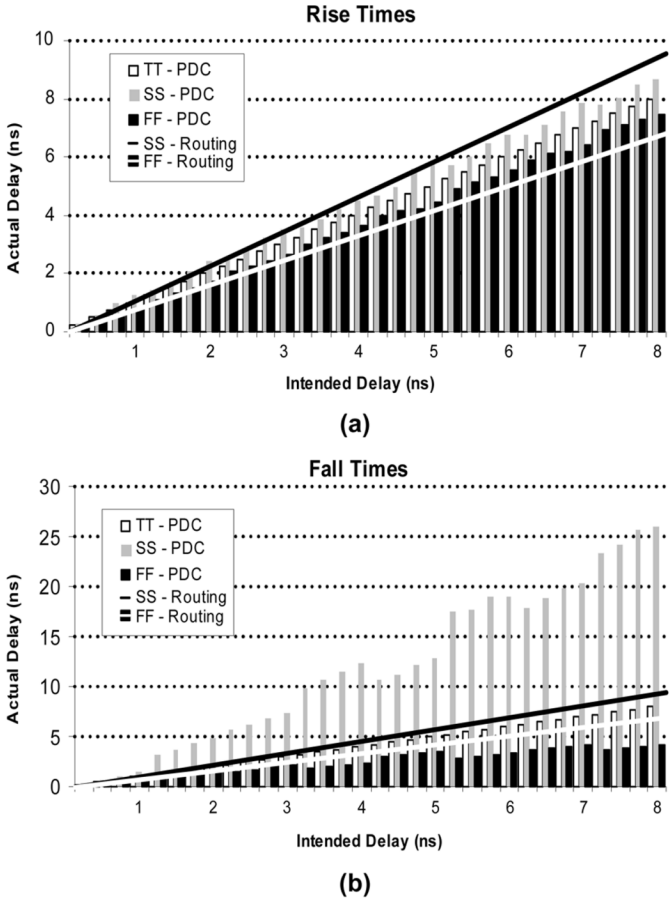


Fig. 10. Rise and fall times of delay element from [9] considering process variation. (a) Rise times. (b) Fall times.

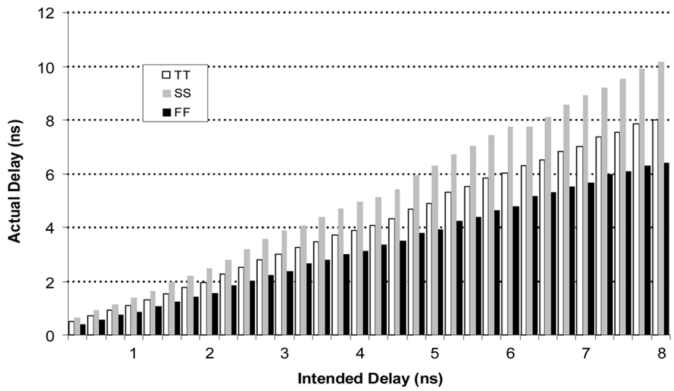


Fig. 11. Rise times of new programmable delay element considering process variation.

26% slower for the FF and SS process corners, respectively. The response of the new delay circuit varies more than the response of the FPGA routing resources since wires do not vary as much as transistors; however, the new delay circuit responds significantly better than the previous delay circuit which makes it more suitable.

2) Within-Die Variation: In the case of within-die variation, speed binning and proportional scaling may not be sufficient. Since the inputs of a LUT can come from any part of the chip, within-die variation can affect the delay of one input differently from another input. Although most connections are local (since the FPGA clustering, placement, and routing tools minimize the routing distance between connections), within-die variation is still a problem for large nets that span the entire chip.

A naïve solution to within-die variation is to reconfigure the programmable delay elements of each FPGA individually. This solution, however, is impractical since it is difficult to obtain PVT variation information for individual FPGAs and it would be time consuming to reconfigure each FPGA with different delays.

Another, more practical solution, is to pessimistically reduce the delay added by each programmable delay element. We first determine D the inserted delay assuming no PVT variation. Then, if the nature of the expected variations are known, we can estimate the approximate worst-case impact of the variation d. We then configure the programmable logic element to insert the delay $D - d$. This ensures that the delay inserted by the delay element does not lengthen the overall delay of the circuit. However, it also means that the actual delay that is inserted may be shorter than the delay that is needed to eliminate the glitch. This will reduce glitch elimination; however, even in cases where the glitch is not eliminated, the width of the glitch is reduced. These shorter pulses are then more likely to be filtered out by other delay elements that are downstream.

Note that a more complete approach to this technique would involve using statistical timing analysis to determine the maximum delays that can safely be added without increasing the critical path delay. However, statistical timing analysis is not supported within our current experimental framework. Nonetheless, the results for this static approach, presented in Section VII-E, still serve to demonstrate the tradeoff between the power savings and the uncertainty introduced by PVT variation.

V. Experimental Framework

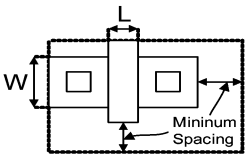
This section describes the experimental framework that is used to obtain the switching activity information and the FPGA area, delay, and power estimates that are presented in Section VI and VII.

A. Switching Activity Estimation

The switching activities are obtained by simulating circuits at the gate level and counting the toggles of each wire. The simulations are driven by pseudorandom input vectors and circuit delay information from the VPR place and route tool [19]. To capture the filtering effect of the FPGA routing resources and of the programmable delay elements, the simulator uses the *inertial delay* model. Furthermore, to replicate an FPGA routing architecture consisting of length 4 routing segments, the VPR delays are divided into chains of 300-ps delay.

B. Area, Delay, and Power Estimation

Area, delay, and power estimates are obtained from the VPR place and route tool and HSPICE simulations. VPR is used to model the existing FPGA circuitry and HSPICE is used to model the added delay element circuitry.



$$Area(W) = \frac{1}{2} + \frac{W}{2 \cdot W_{min}} \quad (1)$$

$$Area(L) = \frac{4}{5} + \frac{L}{5 \cdot L_{min}} \quad (2)$$

Fig. 12. Extension of MTE Area model from [19].

The VPR models are detailed, taking into account specific switch patterns, wire lengths, and transistor sizes. After generating a specified FPGA architecture, VPR places and routes a circuit on the FPGA and then models the area, delay, and power of that circuit. VPR models area by summing the area of every transistor in the FPGA, including the routing, CLBs, clock network, and configuration memory. The area of each transistor is approximated using the minimum transistor equivalents (MTE) metric from [19], which calculates the layout area occupied by a minimum sized transistor plus the minimum spacing as illustrated in Fig. 12.

The model from [19] was augmented slightly in this paper to consider transistors with longer than minimum channel length. Expression (1) models the layout area of a transistor with respect to its channel width (W) and Expression (2) models the area with respect to its length (L). The models were derived by observing the relative area increase when either W or L is increased. The expressions differ slightly since the minimum width of a transistor accounts for approximately one half of the y-component of the layout area, whereas the minimum length accounts for approximately one-fifth of the x-component of the layout area.

The delay and power are modeled after routing occurs, when detailed resistance and capacitance information can be extracted for each net in the benchmark circuit. The Elmore delay model is used to produce delay estimates and the FPGA power model described in [20] is used to produce power estimates. The power model uses the VPR capacitance information and simulated switching activities to estimate dynamic, short-circuit, and leakage power. Note, however, that the leakage power estimates for both the existing FPGA circuitry and the programmable delay elements do not account for PVT variation (typical process, voltage, and temperature are assumed).

C. Architecture Assumptions and Benchmarks

We gathered results for three LUT sizes: 4, 5, and 6 inputs. In all cases, we assumed that each CLB contains 10 LUTs and that the CLBs have 22, 27, and 33 inputs for architectures with 4, 5, and 6 input LUTs, respectively. In each case, we assume that the crossbar that programmably connects the CLB inputs and LUT outputs to the LUT inputs with each CLB is fully populated as described in [19]. Furthermore, for routing, we assumed two segmented routing fabrics, one consisting of buffered length 1 and another of length 4 routing segments and a channel width that is 20% wider than the minimum channel width (a separate value was found for each benchmark). Since the results were similar for both segment lengths, only the length 4 results are presented in Section VI and VII unless stated otherwise.

In each experiment, we used 20 combinational benchmarks including the 10 largest combinational circuits from the MCNC and ISCAS'89 benchmark suites. Before placement and routing, each circuit is mapped to LUTs using the Emap technology mapper [7] and packed into clusters using the T-VPack clusterer [19].

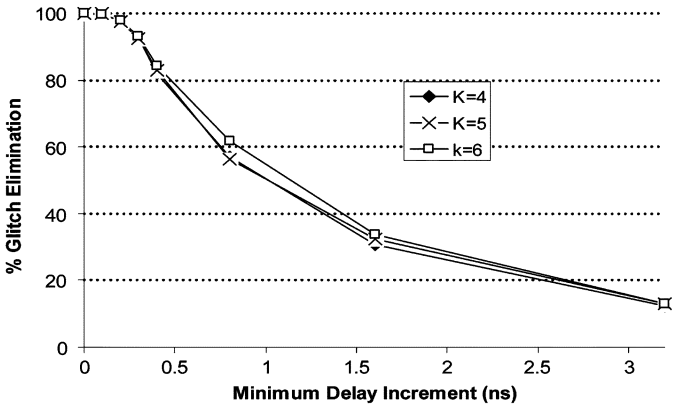


Fig. 13. Minimum LUT input delay for Scheme 1.

VI. Scheme Calibration

Before we examine the overall power savings and area and delay overhead of the delay insertion technique, we need to find suitable values for the parameters of each scheme (listed in Table III). In each case, the value is chosen to eliminate as much of the glitching as possible, while minimizing the area and delay overhead.

A. Scheme 1 Calibration

We first consider the *min_in* parameter, which defines the minimum delay increment of the programmable delay element at the inputs of the LUTs. Intuitively, a smaller delay increment reduces glitching but increases area. Fig. 13 shows how much glitching is eliminated for minimum delay increments ranging between 0.1 and 3.2 ns. To isolate the impact of the *min_in* parameter, the graph assumes that every LUT input has a programmable delay element with an infinite maximum delay (*max_in* is ∞ and *num_in* is K).

The graph illustrates that most of the glitching can still be eliminated when the minimum delay increment is 0.25 ns. This corresponds to the fact that narrow glitches are filtered away by the routing resources and that the majority of glitches have a width greater than 0.2 ns, as described in Section III. The same conclusion holds for FPGAs that use 4, 5, or 6 input LUTs.

The second parameter, denoted *max_in*, defines the maximum delay of the programmable delay element at the inputs of the LUTs. Intuitively, increasing the maximum delay reduces glitching but increases area. Fig. 14 shows how much glitching is eliminated as a function of the maximum delay. The graph illustrates that over 90% of the glitching can be eliminated when the maximum delay of the programmable delay element is 8.0 ns. This corresponds with Fig. 1, which illustrates that the majority of glitches have a width that is less than 10.0 ns.

Finally, *num_in* defines the number of LUT inputs that have a programmable delay element. Intuitively, increasing the number of inputs with delay elements reduces glitching since the arrival times of more inputs can be aligned. Fig. 15 shows how much

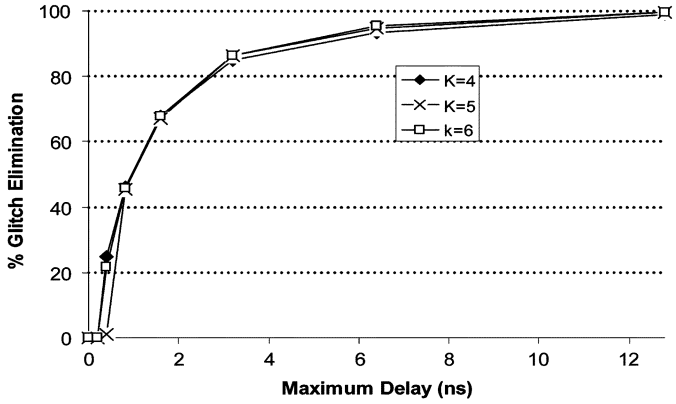


Fig. 14. Maximum LUT input delay for Scheme 1.

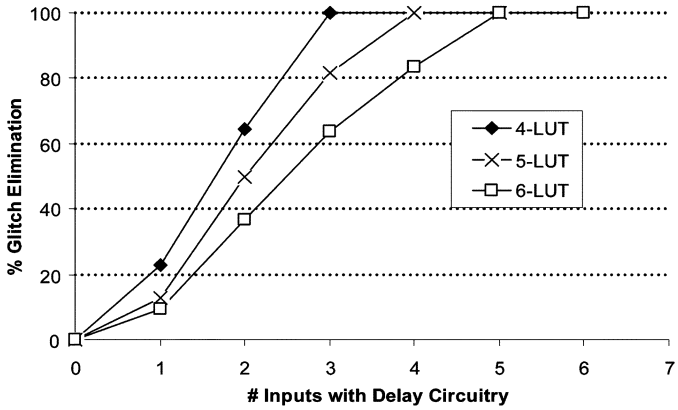


Fig. 15. Number of delay elements/LUT for Scheme 1.

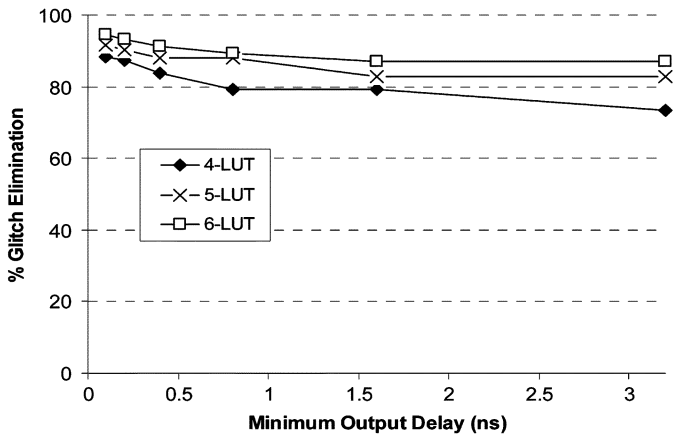


Fig. 16. Minimum LUT output delay for Scheme 3.

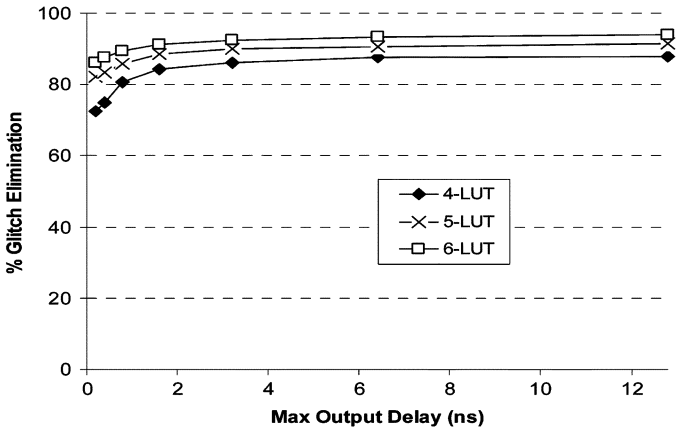


Fig. 17. Maximum LUT output delay for Scheme 3.

glitching is eliminated when the number of inputs with programmable delays is varied. The graph assumes that the *min_in* is $1/\infty$ and *max_in* is ∞.

The graph illustrates that each LUT should have a programmable delay element on every input minus one $(K-1)$. Intuitively, adding delay circuitry to every input is not necessary since each LUT has at least one input that does not need to be delayed (the slowest input). However, adding fewer than $K-1$ delay elements significantly reduces the amount of glitching that can be eliminated. Note also that, since LUTs tend to have uneven input-to-output propagation delays, the $K-1$ delay elements should be added to the slowest inputs so as not to impede the slowest (critical-path) input signal.

B. Scheme 2 Calibration

Scheme 2 has the same three parameters as Scheme 1 and the same values are used for each parameter. Specifically, *num_in* is $K-1$, *min_in* is 0.25 ns, and *max_in* is 8ns. However, to minimize overhead, the maximum delay of the LUT input delay elements (*max_in*) is gradually decreased by half (or by 1 delay stage) per LUT input. As an example, the maximum delay values for a four-input LUT would be 8, 4, and 2 ns.

C. Scheme 3 Calibration

Scheme 3 has five parameters, namely: *min_in, max_in, num_in, min_out*, and *max_out*. The first three parameters control the delay elements at the inputs of the LUTs; the last two parameters control the delay elements at the output of the LUTs. Although the *min_in, max_in*, and *num_in* parameters were already calibrated for Scheme 1, they must be recalibrated for Scheme 2 since the output delay elements change how much delay is needed by LUT input delay elements. Intuitively, however, the value of the *min_in* parameter can be reused since the LUT input delays are still used to perform the final alignment of each signal.

The same technique is used to recalibrate *max_in* and *num_in* but with assumption that *min_out* is infinitely precise $(1/\infty)$ and *max_out* is ∞. The results are similar to those in Scheme 1 except that some glitching is eliminated even when there are no delay elements on the LUT inputs since the output delay elements are able to align some of the inputs and filter out narrow pulses on their own. For Scheme 3, setting *max_in* to 8.0 ns and *num_in* to $K-2$ eliminates most of the glitching.

The remaining output delay element parameters are calibrated assuming *min_in* is 0.25 ns, *max_in* is 8.0 ns, and *num_in* is $K-2$. Fig. 16 shows the glitch elimination for *min_out* from 0 to 3.2 ns assuming that *max_out* is ∞ and Fig. 17 shows the glitch elimination for *max_out* from 0 to 12 ns assuming that *min_out* is $1/\infty$. The graphs illustrate that a 0.25 ns and 8.0 ns are also suitable for *min_out* and *max_out*, respectively.

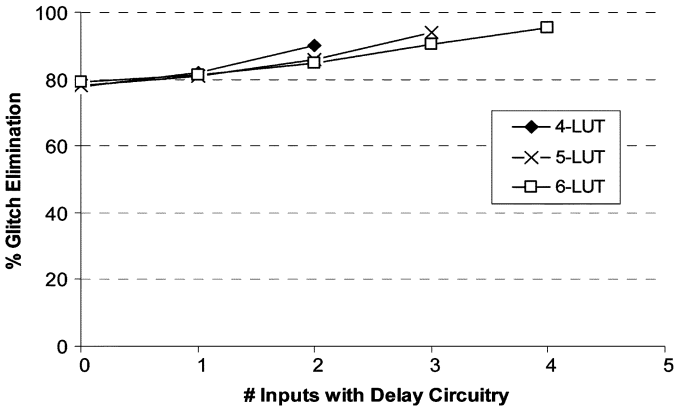


Fig. 18. Number of input delay elements per LUT for Scheme 4.

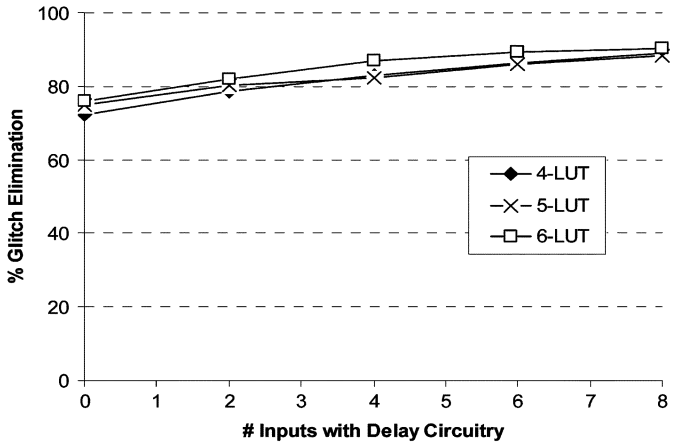


Fig. 19. Number of bank delay elements for Scheme 5.

D. Scheme 4 Calibration

Scheme 4 has five parameters, namely: *min_in, max_in, num_in, min_c*, and *max_c*. The first three parameters control the delay elements at the inputs of the LUTs; the last two parameters control the delay elements at the input of the CLBs. The *min_in, max_in*, and *num_in* parameters are again recalibrated to account for the affect of the CLB input delay elements. The same procedure used in Scheme 1 was used. The results for *min_in* and *max_in* were similar to the previous cases, which indicated that 0.25 and 8.0 ns, respectively, were suitable.

The results for *num_in*, which are plotted in Fig. 18 were different than in the previous cases. To isolate the impact of *num_in*, the graph assumes that *min_in* is $1/\infty$, *max_in* is ∞, *min_c* is $1/\infty$, and *max_c* is ∞. The results indicate that *num_in* should be 1, 2, and 2, for 4, 5, and 6-LUTs, respectively. Intuitively, fewer LUT input delay elements are needed since the CLB input delay elements account for most of the delay. Only in cases where the CLB inputs fan-out to multiple LUTs within that CLB and those fan-outs need different delays are the LUT input delay elements required.

E. Scheme 5 Calibration

Finally, Scheme 5 has five parameters, namely: *min_in, max_in, num_in, max_b*, and *num_b*. The first three parameters control the delay elements and the inputs of the LUTs; the last two parameters control the bank of delay elements in the CLB. The bank of programmable delay elements are only used for signals that need more delay than can be added by the LUT input delay elements, therefore this scheme uses the same *min_in* and *num_in* values as Scheme 1: 0.25 ns and $K-1$, respectively. Suitable values for *max_in* and *max_b* were found empirically to be 4.0 and 8.0 ns, respectively. Finally, Fig. 19 shows glitch elimination with respect to the number of bank delay elements per CLB (*num_b*) assuming *min_in* is 0.25 ns, *num_in* is $K-1$, *max_in* is 4.0 ns, and *max_b* is 8.0 ns. The results show that 4 is a suitable value for *num_b* for CLBs with 10 LUTs.

F. Summary

Table IV summarizes the values that were selected for each of the five delay insertion schemes. The first two columns specify the scheme number and the programmable delay element location. The third and fourth columns specify the minimum delay increment and the maximum delay of the programmable delay element at that location. The fifth column specifies the corresponding number of delay stages needed to implement the programmable delay element. Finally, the sixth column specifies the number of programmable delay elements needed per LUT (rows 2, 3, 5, and 7) and per CLB (rows 4, 6, and 8).

TABLE IV
SUMMARY OF DELAY ELEMENT VALUES

Scheme	Location	Delay Incr. (ns)	Max. Delay (ns)	# Stages	# Circuits
1	LUT Inputs	0.25	8	5	K-1
2	LUT Inputs	0.25	8, 4, 2, …	5, 4, 3, …	K-1
3	LUT Inputs	0.25	8	5	K-2
	CLB Outputs	0.25	8	5	N
4	LUT Inputs	0.25	8	5	1, 2, 2
	CLB Inputs	0.25	8	5	K(N+1) / 2
5	LUT Inputs	0.25	4	4	K-1
	Bank	4.0	8	1	4 (N=10)

VII. RESULTS

This section presents the overall results. It begins by presenting the area, delay, and power overhead of each delay insertion scheme. It then presents the overall power savings assuming there is no PVT variation. Finally, it presents the overall power savings assuming there is PVT variation.

A. Area Overhead

The area overhead is determined by summing the area of the added delay circuitry in each CLB. This area includes the area of the delay elements and the added configuration memory. Table V reports how much area is needed in the CLBs and Table VI reports the percent area overhead taking the CLB and routing area into account. More precisely, the percent area overhead was calculated by dividing the total area occupied by the added programmable delay circuitry by the total area occupied

TABLE V
CLB AREA OVERHEAD (NO GLOBAL INTERCONNECT)

LUT Size	Original CLB Area (MTE)	CLB Area Overhead (MTE)				
		Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	6938	2460	2020	2460	2568	3184
5	10361	3280	2430	3280	3368	3808
6	15228	4100	2720	4100	4282	4494

TABLE VI
OVERALL AREA OVERHEAD

LUT Size	Overall Area Overhead (%)				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	8.0	6.6	8.0	8.4	10.4
5	7.6	5.3	7.6	7.8	8.8
6	6.7	4.4	6.7	7.0	7.3

TABLE VII
AVERAGE POWER OVERHEAD (%)

LUT Size	Power Overhead: $P_{overhead}$ / ($P_{overhead}$ + P_{FPGA}) * 100				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	0.94	0.79	1.02	1.16	0.97
5	0.97	0.84	1.12	1.28	0.99
6	1.02	0.94	1.14	1.10	0.93

TABLE VIII
AVERAGE DELAY OVERHEAD

LUT Size	Average Delay Overhead (%)				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
4	0.21	0.19	2.4	2.3	0.21
5	0.13	0.14	2.2	2.1	0.13
6	0.14	0.15	2.1	1.9	0.14

by the FPGA logic and routing resources, which we determined using VPR.

The tables show that Scheme 2 has the lowest area overhead, followed by Schemes 1, 3, and 4, and finally, Scheme 5 has the highest overhead. Scheme 5 requires the most area because of the large multiplexers needed to select which CLB input or LUT output uses the bank delay elements. Schemes 1, 3, and 4 have a similar area overhead since they use the same size delay elements and roughly the same number of them. Scheme 2 has the lowest area overhead since it uses smaller delay elements. The tables also show the area overhead decreases as the LUT size increases. This occurs since the area of the LUTs and multiplexers increases exponentially with K, while the area of the delay elements only increases linearly.

B. Power Overhead

Even if all the glitches could be eliminated, the programmable delay elements still dissipate power. This overhead is modeled by summing the power dissipated by the added circuitry in each CLB of the FPGA using the following expression:

$$P_{\text{overhead}} = \frac{\sum_{n \in \text{dnodes}} E_{\text{toggle}} \cdot \alpha(n)}{T_{\text{crit}}} + P_{\text{static}}. \quad (3)$$

In the expression, *dnodes* is the set of nodes in the circuit that can be delayed, E_{toggle} is the energy dissipated by one programmable delay element during one transition, $\alpha(n)$ is the switching activity of the delayed node n, and T_{crit} is the critical path delay of the circuit. The energy and leakage power of the programmable delay element is determined using HSPICE, the switching activity is determined using gate-level simulation, and the critical-path delay is determined using the VPR place and route tool. Note, however, that the leakage power estimates assume typical process, voltage, and temperature conditions.

Table VII reports the average overhead power (as a percentage) dissipated by the added delay circuitry for each scheme. The power of the remaining FPGA circuitry is calculated using the power model described in [20]. The table shows that the power overhead is approximately 1% for all the schemes and that Scheme 2 has the lowest power overhead.

C. Delay Overhead

Although the delay elements are programmed to only add delay to early arriving edges, a small delay penalty may be incurred even if the delay element is bypassed because of parasitic resistance and capacitance. To model delay overhead, HSPICE was used to determine the parasitic delay incurred by the delay element. The critical-path delay of each circuit was then recalculated, taking these parasitic delays into account. Finally, the overhead was calculated by comparing the new critical-path delay to the original critical-path delay.

Table VIII reports the average delay overhead for each scheme. Schemes 1, 2, and 4 have the smallest overhead since both have *fast-paths* with no delay elements (no parasitics) to slow down the critical-path. Schemes 3 and 4 have a larger overhead, since neither scheme offer a *fast-path* for critical-path connections. Specifically, the parasitic capacitance of the programmable delay elements at the output of the CLBs for Scheme 3 and at the inputs of the CLBs for Scheme 4 imposes a small delay on any signal that bypasses them (see Fig. 4).

D. Overall Power Savings (Without Variation)

Table IX presents the average glitch elimination for each scheme and Table X presents the corresponding overall power savings. Both tables indicate that Scheme 1 produces the best results, with 91.8% glitch elimination and overall power savings of 18.2%. The power savings are close to the ideal savings of 22.6%. Note also that the results in both tables are for FPGAs with 4-input LUTs and length 4 routing segments; the results for 5 and 6-input LUTs and for FPGAs with length 1 routing segments were similar. As an example, using Scheme 1 for FPGAs the 6-input LUTs and length 1 routing segments reduced glitching by 92.9% and the overall power by 16.8%. In general, the power savings for larger LUTs are slightly smaller because there tends to be less glitching to begin with since the netlists have fewer levels of logic. Moreover, the segment length distribution has little affect because the *needed delays* tend to be quite dispersed even for buffered routing architectures with only one segment length. The timing of a signal is affected not only by the number of LUTs and routing

TABLE IX
% GLITCH ELIMINATION OF EACH SCHEME

Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
91.8%	87.3%	83.3%	81.8%	85.4%

TABLE X
OVERALL POWER SAVINGS

Circuit	Power Saving (%)				
	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5
C135	25.4	25.4	25.0	25.0	25.8
C1908	18.1	17.5	18.4	16.1	17.0
C2670	11.6	11.4	11.3	10.2	11.7
C3540	27.5	25.4	22.9	23.5	26.3
C432	13.0	11.0	10.7	10.6	10.6
C499	31.8	31.8	30.9	32.3	32.4
C5315	18.2	16.8	16.2	16.0	17.9
C6288	52.1	41.3	43.2	40.0	46.1
C7552	22.6	21.0	18.9	19.7	22.3
C880	7.2	6.5	6.5	8.0	7.1
alu4	2.5	2.5	2.4	3.3	2.7
apex2	3.6	3.6	3.2	3.8	3.6
apex4	9.5	9.5	9.1	9.4	9.3
des	15.1	14.9	12.1	14.2	14.4
ex1010	16.8	16.8	16.4	16.5	15.9
ex5p	23.8	23.3	23.4	21.5	25.0
misex3	7.6	7.6	7.3	7.3	7.2
pdc	11.1	10.8	10.1	10.7	11.3
seq	5.3	5.2	5.9	5.7	5.6
spla	20.3	20.1	19.8	20.0	20.2
Average	18.2	16.8	16.3	16.2	17.4

segments it passes through, but also by where it taps on to and off of those segments.

E. Overall Power Savings (With Variation)

The results presented in the previous sections assumed no PVT variation. The following results present the overall power saving when the technique described in Section IV-E is applied to cope with the timing uncertainty introduced by PVT variation. Specifically, we repeated the experiments from Section VII-D, using the same delay element parameter values as before, but we reduced the delay inserted by each delay element by a factor β. We varied β from 0.7 (meaning each delay element is programmed to provide a delay of 70% of the value predicted assuming no process variations) to 1.0 (which is the same as the results in Section VII-D). Fig. 20 shows the results. In this figure, β is shown on the X-axis. The lower line indicates the amount of glitching removed compared to the case when programmable delay elements are not used. As the results show, when β is 0.7, the glitch savings are reduced to 56% (compared to 91% when process variations are not considered). The upper line shows the resulting decrease in power; as expected, the power reduction is proportional to the number of glitches removed. Overall, these results indicate that the delay insertion technique still works when the added delays are reduced, but with diminished glitch and power savings as the timing uncertainty increases.

VIII. CONCLUSION AND FUTURE WORK

This paper proposed GlitchLess, a glitch elimination technique to reduce dynamic power in FPGAs. The implementation

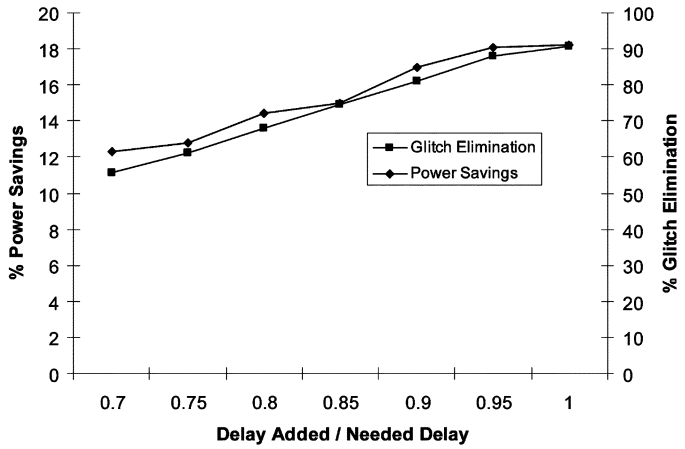


Fig. 20. Glitch elimination and power savings versus β.

investigated here adds programmable delay elements to the CLB architecture to align the edges of each LUT input, thereby preventing formation of glitches on the LUT outputs. The delay elements can also filter some glitches produced by the upstream logic. Five alternative schemes were considered for delaying the logic inputs. Scheme 1, which uses delay elements on $K-1$ inputs of each LUT, produced the greatest power savings, reducing power by 18.2%. However, Scheme 2, which uses $K-1$ delay elements that gradually decrease in size, produced similar power savings with less area. On average, Scheme 2 eliminates 87% of all glitching, which reduces overall FPGA power by 16.8%. The added circuitry increases overall area by 6.6% and critical-path delay by less than 1%.

There are a number of interesting issues that were not fully explored in this paper that merit further research. First, a more complete approach to the proposed delay insertion technique would involve using statistical timing analysis to determine the maximum delays that can safely be added without increasing the critical path delay. Second, investigation using newer process technologies that tend to dissipate more leakage power is also needed. Finally, further research of circuit-level implementations for delaying the inputs or preventing the output from toggling prematurely may yield lower overhead, increased power savings, and/or improved PVT tolerance. As an example, a self-calibrating delay element that tunes itself to the latest arriving transition of a LUT (relative to the clock) would be ideal since it would be more tolerant to variation. Furthermore, this delay element could be used to gate all the early arriving inputs *or* to suppress output transitions until the last input arrives. Such an implementation may reduce area since it requires only one delay element per LUT.

REFERENCES

[1] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, "A 90 nm low-power FPGA for battery-powered applications," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2006, pp. 3–11.

[2] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton, "Dynamic voltage scaling for commercial FPGAs," in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, 2005, pp. 173–180.

[3] A. Kumar and M. Anis, "Dual-Vt FPGA design for subthreshold leakage tolerance," in *Proc. Int. Symp. Quality Electron. Des. (ISQED)*, 2006, pp. 735–740.

[4] R. R. Rao, D. Blauw, D. Sylvester, C. J. Alpert, and S. Nassif, "An efficient surface-based low-power buffer insertion algorithm," in *Proc. ACM Int. Symp. Phys. Des.*, 2005, pp. 86–93.
[5] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "A dual-Vdd low power FPGA architecture," in *Proc. Int. Conf. Field-Program. Logic Appl. (FPL)*, 2004, pp. 145–157.
[6] J. Lamoureux and S. J. E. Wilton, "FPGA clock network architecture: flexibility vs. area and power," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2006, pp. 101–108.
[7] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware computer-aided design algorithms for field-programmable gate arrays," *J. Low Power Electron. (JOLPE)*, vol. 1, no. 2, pp. 119–132, 2005.
[8] J. H. Anderson, "Power optimization and prediction techniques for FPGAs," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2005.
[9] J. Lamoureux, G. G. Lemieux, and S. J. E. Wilton, "GlitchLess: An active glitch minimization technique for FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2007, pp. 156–165.
[10] J. C. Monteiro and A. L. Oliveira, "Finite state machine decomposition for low power," in *Proc. Des. Autom. Conf. (DAC)*, 1998, pp. 758–763.
[11] D. Kim and K. Choi, "Power conscious high-level synthesis using loop folding," in *Proc. Des. Autom. Conf. (DAC)*, 1997, pp. 441–445.
[12] M. Kandemir *et al.*, "Influence of compiler optimizations on system power," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 6, pp. 801–804, Dec. 2001.
[13] D. Chen, J. Cong, F. Li, and L. He, "Low-power technology mapping for FPGA architectures with dual supply voltages," in *Proc. Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2004, pp. 109–117.
[14] S. Wilton, S.-S. Ang, and W. Luk, "The impact of pipelining on energy per operation in field-programmable gate arrays," in *Proc. Int. Conf. Field-Program. Logic Appl. (FPL)*, 2004, pp. 719–728.
[15] T. Czajkowski and S. Brown, "Using Negative edge triggered FFs to reduce glitching power in FPGA circuits," in *Proc. Des. Autom. Conf. (DAC)*, 2007, pp. 324–329.
[16] J. C. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. Des. Autom. Conf. (DAC)*, 1993, pp. 398–402.
[17] L. Benini *et al.*, "Glitch power minimization by selective gate freezing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 3, pp. 287–298, Jun. 2000.
[18] A. Raghunathan, S. Dey, and N. K. Jia, "Register transfer level power optimization with emphasis on glitch analysis and reduction," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 8, pp. 1114–1131, Aug. 1999.
[19] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron*. Norwell, MA: Kluwer, 1999.
[20] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Trans. Des. Autom. Electron. Syst. (TODAES)*, vol. 10, no. 2, pp. 279–302, Apr. 2005.

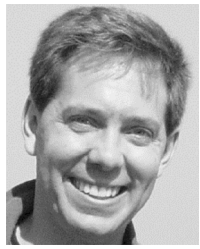
Julien Lamoureux received the B.Sc. degree in computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2001, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of British Columbia (UBC), Vancouver, BC, Canada, in 2003 and 2007, respectively.

In between his studies at UBC, he completed an internship at the Altera Toronto Technology Centre, where he worked on FPGA power modeling. In 2007, he started a post doctoral in the Department of Computing, Imperial College, London, U.K., where his research focuses on architecture and CAD for FPGAs.

Guy G. F. Lemieux received the B.A.Sc. degree in engineering science, and the M.A.Sc., and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada.

In 2003, he joined the Department of Electrical and Computer Engineering, the University of British Columbia, Vancouver, BC, Canada, where he is an Assistant Professor. He is the coauthor of the book *Design of Interconnection Networks for Programmable Logic* (Kluwer, 2004). His research interests include computer-aided design algorithms, VLSI and SoC circuit design, FPGA architectures, and parallel computing. His specialization is in interconnection network design and routing algorithms.

Dr. Lemieux was a recipient of the Best Paper Award at the IEEE International Conference on Field-Programmable Technology in 2004.



Steven J. E. Wilton received the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 1992 and 1997, respectively.

In 1997, he joined the Department of Electrical and Computer Engineering, the University of British Columbia, Vancouver, BC, Canada, where he is now an Associate Professor. During 2003 and 2004, he was a Visiting Professor with the Department of Computing at Imperial College, London, U.K., and at the Interuniversity MicroElectronics Center (IMEC), Leuven, Belgium. He has also served as a consultant for Cypress Semiconductor and Altera Corporation. His research focuses on the architecture of FPGAs and the CAD tools that target these devices. In 2005, he was the Program Chair for the ACM International Symposium on Field-Programmable Gate Arrays and the Program Cochair for the International Conference on Field Programmable Logic and Applications.

Dr. Wilton was a recipient of Best Paper Awards at the International Conference on Field-Programmable Technology in 2003 and 2005, at the International Conference on Field-Programmable Logic and Applications in 2001, 2004, and 2007, and the Douglas Colton Medal for Research Excellence for his research into FPGA memory architectures in 1998.