PGR: Period and Glitch Reduction Via Clock Skew Scheduling, Delay Padding and GlitchLess

Xiao Dong, Guy G.F. Lemieux

Department of Electrical and Computer Engineering, University of British Columbia
Vancouver, Canada
{*xdong* | *lemieux*}*@ece.ubc.ca*

***Abstract*—This paper describes PGR, an architectural technique to reduce dynamic power via GlitchLess or to improve performance via clock skew scheduling (CSS) and delay padding (DP). It is integrated into VPR 5.0, and is invoked after the routing stage. We use programmable delay elements (PDEs) as a novel architecture modification to insert delay on FF clock inputs, enabling all optimization steps to share it, avoiding multiple architecture modifications. The central theme of this paper is considering the trade-off between power and performance, and finding an appropriate compromise considering process variation and timing uncertainties. Overall, an average of 15% speedup can be achieved via CSS alone, or up to 37% for individual circuits. Although delay padding only benefits several circuits, the average improvement of those circuits is an additional 10% of the original period, or up to 23% for individual circuits. In addition, a new model to estimate glitching power is proposed, taking into account the analog behavior of glitch pulse width reduction as it travels along FPGA routing tracks. We show that the original glitch estimation method can underestimate glitching power by up to 48%, and overestimate by up to 15%. GlitchLess is performed on both the original VPR and post-CSS solutions. We are able to eliminate on average 16% of glitching power, and up to 63% for individual circuits.**

I. Introduction

Power and performance are two very important issues in FPGA design. FPGA applications typically consume more power per operation, and run at slower speeds than their ASIC counterparts, due to circuitry needed for programmability.

There is much research effort addressing these two topics. On the performance front, two popular techniques are retiming and clock skew scheduling (CSS). The former method changes the positions of sequential elements (SEs) to shorten effective critical path while maintaining functionality [1], and has been applied to FPGAs ([2], [3], [4]). CSS achieves the same goal by assigning intentional clock skews to SEs instead of moving them physically ([5], [6]), and has also been applied to FPGAs ([7], [8], [9]).

On the power front, dynamic power consumption is significant due to large capacitive loading on the interconnect. Recent advances in process technology have seen a decreasing trend in the rate of increase of dynamic power versus static power. However, total dynamic power still accounts for about 50% of total power [10]. In this paper, "dynamic power" excludes clock network power. Dynamic power arises from two kinds of logic transitions produced by combinational look-up tables (LUTs), functional and glitch. The former causes the data to be different at the end of a clock period, a result of user logic functions. The latter results from input data signals arriving at different times during the period, causing the output to fluctuate before settling down. Several existing examples to reduce glitching power include techniques at the architecture level [11], or at the CAD level during technology mapping [12] and routing [13].

This paper makes the following contributions:

1) The programmable delay element (PDE) proposed in [11] is used to provide discrete delays on flip-flop (FF) clock inputs. This unified architecture change, shared by CSS, DP and glitch reduction, avoids the need for multiple architecture modifications. For glitch reduction, we use the concept of GlitchLess (GL) [11], but with a different implementation. Previously, delay elements needed to be very precise to eliminate glitching. This can be difficult with increasing process variations. The new approach is much more resistant to variation.
2) Integrated delay padding scheme with CSS to further optimize performance. Past work ([14], [15], [16], [17]) uses either LP or graph theory to solve CSS. However, these techniques apply only to ASICs, and assume padded delays are continuous. However, a PDE can only provide discrete delays. We adapt the algorithms to use discrete delays as well as margin for process variation.
3) An integrated tool flow that uses the same physically realizable architectural change to reduce power and increase performance. CSS, delay padding and GlitchLess are combined with VPR 5.0 [18] into a single framework. This is important for getting a final result that considers both delay and power at the same time.
4) Improvement on vector based activity estimation [19], which used a threshold to determine whether a glitch does not propagate at all or propagates indefinitely. Our work models the analog behavior of the gradual decrease in width of a narrow glitch as it travels along FPGA interconnect, and calculates glitch power accordingly.

The central theme of this paper highlights the major difference of this work: previous research has focused purely on either performance or power. Our work shows performance optimization adds to power, while 100% glitch reduction is not possible without impacting performance. Therefore it is important to achieve an appropriate compromise between the two. Furthermore, we motivate better PDE designs by putting

PDE power overhead in perspective with total dynamic power consumption before and after glitch reduction. We show that while there is potential for good savings, a power-efficient PDE is crucial to the attractiveness of both period reduction and glitch reduction.

The rest of the paper is organized as follows. Section 2 introduces basic concepts. Section 3 describes architecture changes and its adaptation by the optimization steps. Section 4 details the modification to glitch estimation. Section 5 outlines our algorithm. Section 6 provides results and discussion, and section 7 concludes our work.

II. Background and Past Work

A. Clock Skew Scheduling

Clock networks suffer from clock skew due to variation [20]. CSS uses it as a resource for improving performance, instead of treating it as an unavoidable burden. In the following example, Fig. 1, a zero-skew clock network means the circuit has a minimum period of 14ns assuming zero setup/hold times. If a skew of 4ns is applied to FF_B, the circuit is able to operate at a minimum period of 10ns. This effect can be viewed as time borrowing: shortening the effective delay of long paths, at the expense of increased delay for short paths. Indeed, the path from FF_B to FF_C now has an effective delay of 10ns.

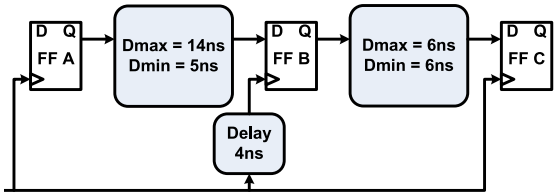

Fig. 1. Intentional Clock Skew

The relative skew assigned to two neighboring FFs is bound by a setup time (T_s) constraint (Eq. 1) and a hold time (T_h) constraint (Eq. 2) to avoid zero-clocking and double-clocking conditions, respectively. T_i and T_j are clock arrival time at FFs i and j, $D_{max}(i,j)$ and $D_{min}(i,j)$ is the maximum and minimum combinational delay (due to variation or reconvergence) between FFs i and j, respectively [5]. Delay M is a user-defined safety margin to compensate for process variation, and allows T_i, T_j and the path delays to vary up to M without violating the constraints [6].

$$T_j - T_i \geq T_s + D_{max}(i,j) - P + M \quad (1)$$

$$T_i - T_j \geq T_h - D_{min}(i,j) + M \quad (2)$$

The above system of equations is an optimization problem for period P subject to $|T_i| < P$, and can be solved by Linear Programming (LP). A more efficient method [6] uses graph theory [21] and binary search to find the optimum P between upper and lower bounds (Eq. 3), where $G(V,E)$ is the graph constructed with a set of constraints, with vertex v_i corresponding to T_i.

$$P_{max} = \max_{G(V,E)} (T_s + D_{max}(i,j) + M)$$

$$P_{min} = \max_{G(V,E)} (T_s + T_h + D_{max}(i,j) - D_{min}(i,j) + 2M) \quad (3)$$

Architecture changes required to implement intentional clock skew varies. One way is to use multiple global clock lines available in the FPGA to implement different skews [7]. An alternative approach [8] uses a single global H-tree with ribs on the H-tree for local routing. PDEs are inserted into branching points of the clock tree. The clock goes through a trail of PDEs before arriving at each FF node, and there are more choices for skew values because of this levelized structure. In [9], 4 PDEs are inserted at each rib of the H-tree, producing 4 skewed version of the global clock for each row. A statistical model is used to model process variation. All of the above approaches focus on CSS only. Our delay padding scheme requires extra skews to be available in the clock line in addition to those for CSS. While our method may sometimes use more power than previous work, it allows extra flexibility for delay padding (further performance gains), and also for GlitchLess (power reduction).

B. Delay Padding

The setup/hold constraints can limit the range of skews that can be assigned to SEs, and therefore the smallest obtainable period. In Eq. 1 and 2, larger D_{max} and smaller D_{min} will decrease the permissible range of assigned skews [22]. Nothing can be done to decrease $D_{max}(i,j)$, but an increase in $D_{min}(i,j)$ will widen the permissible range, allowing skew assignment to be more flexible. This *short path* optimization effectively reduces hold time violations, allowing a smaller period. We call this step delay padding.

C. Glitch Reduction

GlitchLess reduces glitching by delaying early arriving signals to prevent the output from fluctuating [11]. To realize this, PDEs are added to LUT inputs. In [11], only combinational circuits are included. In this paper, we extend this work to sequential circuits as well so CSS can be applied. Other work done to reduce glitching includes [13], which uses routing techniques, and [12], which proposes a new glitch-driven technology-mapping tool.

D. Power Calculation

Dynamic power is defined by $P = \alpha \times C \times V_{dd}^2 \times f$, where α is switching activity, C is capacitance, V_{dd} is supply voltage and f is operating frequency. For 65nm technology, V_{dd} is 1V. The power figure we will refer to in this work is the power per operation, namely $P_{op} = \alpha \times C$. We define a power unit P_{op} as 1 femto-Farad of capacitance switching once per clock cycle ($\alpha = 1$).

III. Architecture

A major contribution of this work is the proposal of a unified architecture change that can be shared by CSS, delay padding and GlitchLess. This section will detail this architecture as well as its adaptation by each of the 3 optimization steps. We assume that newer FPGAs such as the Stratix III and Virtex 6, have 2 flip flops per LUT.

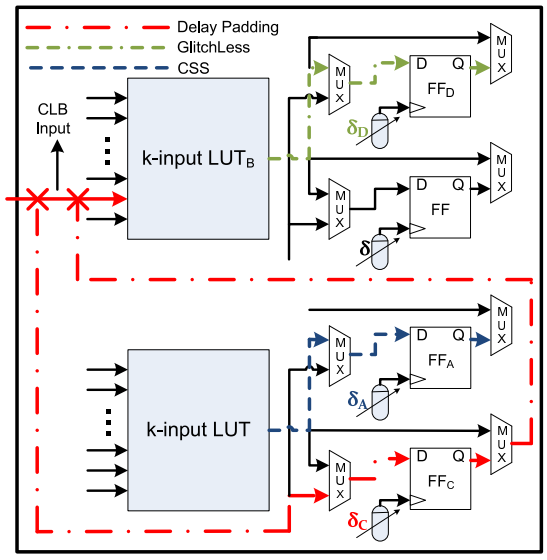

Fig. 2. Unified Architecture Modification

TABLE I
SEQUENTIAL CIRCUIT CHARACTERISTICS AND GLITCHING POWER

circuit	depth	%FF	pre-CSS %	post-CSS %
bigkey	5	42.5	1.9	4.2
clma	18	2.8	6.2	11.5
diffeq	16	52.4	3.4	2.7
dsip	5	48.9	1.3	1.7
elliptic	20	59.3	21.0	24.0
frisc	25	51.0	13.0	15.1
s298	17	1.7	21.7	29.5
s38417	13	46.0	26.8	28.3
s38584.1	11	43.4	7.8	9.4
tseng	15	66.7	15.3	16.1
average			11.8	14.2

A. CSS and Delay Padding

Architecture changes are highlighted in Fig. 2 with legends shown to distinguish optimization steps. CSS can be done by adding delay δ_A to FF_A. For delay padding, we use local rerouting within CLBs. The CLB input (solid arrow line) in Fig. 2 goes to LUT_B originally. We reroute it (dash-dotted line) to unused FF_C in another BLE, then back to the original LUT. Properly adjusting the skew assigned to δ_C, any desired delay can be achieved provided there is enough slack for it.

B. Motivation for Glitch Reduction

Glitching can account for a large portion of dynamic power. CSS perturbs glitching. All SEs have the same signal departure time in zero-skew circuits, but skew assigned to SEs effectively delays that time, changing the amount of glitching created downstream. In Table I, the pre-CSS and post-CSS columns show the amount of dynamic power due to glitching before and after CSS and delay padding has been performed, respectively. An architecture with 4-input (k=4) LUTs, 10-LUT clusters with 22 inputs per cluster is used. In general, the amount of glitching increases by a fair margin after CSS. This further motivates the need for glitch reduction.

C. Architecture for Glitch Reduction

To eliminate glitching on a combinational node, we use a circuit-level architecture change different from that analyzed in [11]. Instead of inserting a PDE at LUT inputs, we achieve glitch reduction by intentional clock skew (Fig. 2). The LUT output is directed to FF_D, whose clock skew δ_D will be set to the latest arrival time of all LUT inputs plus setup time and safety margin. The LUT output fluctuates, but the FF will block all glitches until the final functional evaluation is known. Our approach requires only one PDE to eliminate the glitching for each LUT, compared to at least k-1 PDEs for each LUT used in [11]. One disadvantage of this approach is the fact that clock has an activity of 1. Compared to PDEs inserted into the data lines with relatively low activity, this approach may introduce a significant power overhead. We will show how this affects the results in section VI.

IV. IMPROVED GLITCH ESTIMATION

The ACE tool [19] filters out fluctuations of short pulse widths since the routing resource can damp them out. Originally, simulation determined this maximum pulse width that can be filtered out by a single stage of length-4 routing segment. A glitch longer than this threshold is assumed to go on indefinitely, otherwise it is assumed to consume no power. Neither of these assumptions is true in reality: as long as the pulse width of a glitch is below a different threshold (short glitch), it will be *gradually* filtered out after propagating down a certain number of wire segments. All glitches longer than the threshold can propagate indefinitely.

We try to address the above issue by first modifying ACE to group glitches of different pulse widths into bins (for example, glitches ranging from 15ps to 20ps is bin #1, etc), and this histogram is printed for each net into an output file.

Cadence (Spectre) simulations are done for glitches of varying pulse widths, propagating down a routing track of n wires or stages. A short glitch of a particular pulse width, travelling down a routing track of n stages and being gradually filtered out, will consume a certain amount of power. This power can be expressed as a percentage normalized to the power consumed by a long glitch propagating down the same n stages. Simulation results are summarized in Fig. 3. A converging trend is observed as the lines get closer together for increasing number of stages. Therefore it is assumed that any net longer than 10 stages (wire segments) will behave the same as a 10-stage net.

To calculate total dynamic power consumption, ACE output and Cadence simulation results are read into VPR as separate input files. For a glitch generated at the source node of a net in the circuit, the length and capacitance of the routing track for that net is determined with the VPR routing graph, the glitch activity for each bin is read from ACE, and the amount of glitching power can be calculated by multiplication of capacitance, glitch activity, and the percentage found in Fig 3 via indexing by net length and bin #. There are other components that consume dynamic power, namely intra-CLB routing and MOSFETs that make up LUTs and SEs.

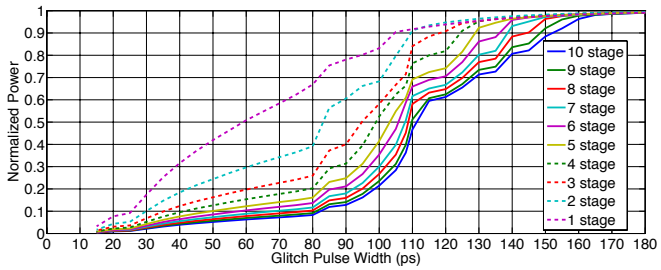

Fig. 3. Segment Length to Power Lookup (65nm Technology)

TABLE II
GLITCH POWER (P_{op}) OF ORIGINAL ACE AND ACE WITH BINNING

circuit	k = 4			k = 6		
	Bins	Original	% diff	Bins	Original	% diff
bigkey	913	471	48.4	560	629	-12.4
clma	3794	3407	10.2	2955	3303	-11.8
diffeq	136	129	4.9	63	58	6.7
dsip	698	512	26.6	574	557	3.2
elliptic	11607	10462	9.9	6408	6944	-8.3
frisc	1185	1096	7.5	1045	1088	-4.1
s298	5350	3906	27.0	4956	5585	-12.7
s38417	29292	19195	34.5	7036	8111	-15.3
s38584.1	10455	9246	11.6	4052	4395	-8.5
tseng	1334	1326	0.6	590	608	-3.2

The former is dominant because a feedback wire from LUT output to LUT input MUXes in the same CLB carries much more capacitance than the latter, which we neglect in our calculations.

The results from the original ACE, and those obtained from binning, are compared in Table II for circuits produced by VPR (all pre-CSS). Units are P_{op} described in section II-D. All circuits are simulated using 5000 pseudo-random input vectors. A positive percentage difference means the original ACE underestimates glitching. The original ACE can underestimate glitching power as much as 48%, for k=4, and overestimate as much as 15% for k=6. Generally, original ACE underestimates glitch power for k=4 because arrival time differences for a small LUT tend to be smaller and get dropped (below threshold).

Although our glitch power modelling is improved, there is still work to be done, such as comparing ACE results against HSPICE power simulations. In addition, glitch filtering creates two issues: glitch generation and propagation. The former is created at the output of a gate generated by the combined effect of its logic function and different input arrival times. The latter addresses the fact that a short glitch becomes narrower as it travels along a routing path, including its possible elimination. Our work better estimates the effect of pulse narrowing on power consumed in the interconnect immediately following glitch generation. However, it does not propagate the narrowed glitch through each LUT sink; instead, it propagates the original pulse width. For a complete analysis, we need to account for the change of glitching activity on downstream LUTs caused by glitch narrowing. This requires tight integration of VPR and ACE so that logic evaluation (ACE) and routing RC trees (VPR) can be obtained concurrently.

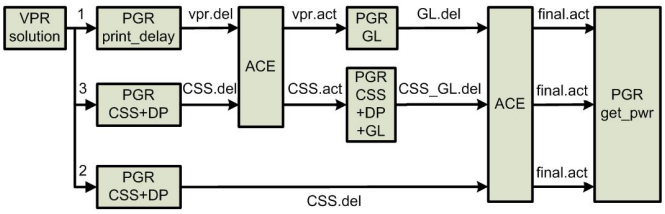

Fig. 4. Top Level Algorithm

V. ALGORITHM

The overall approach is illustrated in Fig. 4. It offers three approaches to glitch reduction. The first (route "1") uses the original VPR placement and routing solution to generate a net delay file for ACE, which produces an activity file for PGR to do glitch reduction only. The resulting net delays are analyzed by ACE again to produce final activities, and the power analysis routine of PGR is used to determine power savings. Alternatively, the P&R solution can be used directly by PGR to do CSS and DP, followed either by activity simulation to determine power (route "2"), or by ACE simulation, a full run of PGR that includes CSS, DP and GL, and final analysis by ACE to get power results (route "3"). CSS and DP are done twice since the delays affect ACE output.

A. CSS and Delay Padding

Our post-CSS delay padding algorithm offers the following novelties. While the traditional approach [15] considers an ASIC environment where any arbitrary delay is realizable, our algorithm targets FPGAs, is aware of discrete delay steps, process variation margins that limit both the minimum and maximum delay that can be assigned to a node, and the possibility that delay padding may fail due to these margins. To our knowledge, this is the first time delay padding has been applied to FPGAs. Our algorithm is outlined in Fig. 5.

In each iteration of assign_skew(), the optimum period and skews determined with the approach in [6] are stored in a solution array, and all critical hold time edges are appended into a list of currently deleted edges [15]. In each iteration, the combinational LUTs on each critical edge are identified and put into arrays with find_deleted_edge_nodes(). When the lowest possible period is attained, the algorithm attempts to pad delays for all deleted critical edges from the most current iteration. In case delay padding is not successful, additional attempts will be made for earlier iterations until a valid padding solution is achieved.

The detailed delay padding algorithm is shown in Fig. 6. For each deleted edge, the algorithm attempts to pad delays for each combinational node on the critical *hold time* short path, starting with the LUT immediately following the source node. Timing analysis is done to safeguard the setup time constraint. On lines 6 to 8 of Fig. 6, the skew is first set to the arrival time of the fanin signal plus the setup time and variation margin, rounded up to the nearest discrete time unit specified by the

```
1: iteration = 0;
2: initialization();
3: solution[iteration] = assign_skews(P_max, P_min);
4: num_edges = find_crit_hold_edges(edges[iteration]);
5: while num_edges > 0 do
6:    find_deleted_edge_nodes();
7:    recalc_binary_bounds(P_max, P_min);
8:    iteration++;
9:    solution[iteration] = assign_skews(P_max, P_min);
10:   num_edges = find_crit_hold_edges(edges[iteration]);
11: end while
12: while iteration > 0 do
13:   success = pad_delay(edges[iteration],solution[iteration]);
14:   if success then
15:      break;
16:   end if
17:   iteration−−;
18: end while
```

Fig. 5. CSS and Delay Padding Algorithm

```
1: success = 1;
2: for all edge "iedge" in deleted edges list do
3:    needed_delay = calculate_needed_delay(iedge);
4:    for all node "n" on deleted edge "iedge" do
5:       analyze_timing();
6:       max_padding = get_max_possible_padding(n);
7:       skew = roundup(fanin→arrival + Ts + MARGIN
8:          + fanin_delay(n, fanin), PRECISION);
9:       delay = skew − fanin→arrival
10:         − fanin_delay(n, fanin);
11:      while delay<(needed_delay && max_padding) do
12:         increment skew and delay by PRECISION
13:      end while
14:      needed_delay −= delay;
15:      if needed_delay ≤ 0 then
16:         edge_done = 1; break;
17:      end if
18:   end for
19:   if edge_done == 1 then
20:      check_other_paths();
21:   else
22:      success = 0;
23:   end if
24: end for
25: if !success then
26:   roll_back_delays();
27: end if
28: return success;
```

Fig. 6. Detailed Delay Padding Algorithm, pad_delay()

```
1: for all levels in timing graph do
2:    rank_nodes(&list, threshold);
3:    for all node "n" in list do
4:       skew = roundup(n→arrival
5:          + Ts + MARGIN, PRECISION);
6:       needed_slack = skew − n→arrival + MARGIN;
7:       if needed_slack < n→slack then
8:          for all fanin "f" of node "n" do
9:             needed_delay = n→arrival − f→arrival
10:               − fanin_delay(n, f);
11:            fanin_delay(n, f) +=
12:               needed_delay + needed_slack;
13:         end for
14:         analyze_timing();
15:      end if
16:   end for
17: end for
```

Fig. 7. Glitch Reduction Algorithm

parameter "PRECISION". It is then incremented in quantized steps (lines 9 to 11), until either the node's slack runs out, or the needed delay is satisfied. When delay padding for an edge finishes, check_other_paths() is used to check whether other short delay paths (with the same source and sink) are violated. If found, a recursive call to pad_delay() will be invoked until setup and hold time constraints for all combinational paths with the same source and sink are satisfied.

In this work, "PRECISION" and "MARGIN" in Fig. 6 are chosen to be 0.1ns, or 1 discrete step. Combined with rounding up (line 7), each PDE has at least 0.1ns of uncorrelated variation tolerance between its assigned skew and path delay, for early clock signals. An additional 0.1ns is added to the needed_slack (line 6 in Fig. 7) for both delay padding and GlitchLess to account for late clock signals. It is omitted in Fig. 6 to save space. The margin M for CSS is 0.2ns, e.g., allowing T_i and T_j (Eqs. 1,2) to each shift 0.1ns away from each other in the worst case. For larger delays, this margin may be too small, and future work will investigate performance impact of different margins.

B. Glitch Reduction

One significant difference of this work compared to [11] is added consideration of process variation. In [11], all added delays are shortened by an amount d so that variation will not increase the critical path. This may result in increased glitching power in practise, since narrow pulses are not "zero power" as assumed previously. Our approach eliminates this issue by stopping all glitching at the FF input, and only clocks through the data when the last signal has arrived. The algorithm is shown in Fig. 7.

The circuit is traversed in a breadth first fashion so that the extra delays assigned upstream do not invalidate those assigned downstream. For each node, GlitchLess delay is assigned based on calculated slack (lines 6 to 13). A threshold parameter is specified to filter out nodes with small glitch

activity. In each level (line 1), all combinational nodes are ranked according to their glitching power, so that nodes with high loading get priority during PDE delay assignment. Care is taken to give each PDE extra margin for process variation in addition to setup time.

VI. Results and Discussion

The largest 10 MCNC sequential circuits are used as benchmarks, and they are simulated for k = 4 and 6, N = 10, and I = 22 and 33, respectively. 65nm technology is used. All results are normalized with respect to the original solution found using VPR 5.0. Architecture files are from the iFAR repository [23], and routing resource capacitance and resistance values are calculated from the PTM website [24] assuming CLBs are 125μm squares. All circuits are simulated using timing-driven placement and routing, with a channel width of 100. Activity estimation is produced with the modified ACE described in section IV, simulated with 5000 input vectors. CSS+DP runtime ranges from a few seconds to minutes for each circuit, and GlitchLess requires a few seconds for each circuit. A Xeon X5355 2.66GHz CPU is used. ACE is run on a UltraSPARC-III CPU at 900MHz, and it needs about 45 minutes to run 10 circuits.

A. CSS Only Results and Power Overhead Estimation

In Table III, we demonstrate the period reduction (as a percentage of original critical path) we were able to obtain from CSS and delay padding, the increase in power as a result, and the impact of PDE power overhead. An average speedup of 13% and 16% are obtained for k=4 and 6, for CSS alone. Delay padding further improves 4 circuits (bold results). For elliptic, frisc and tseng, delay padding is very helpful. Combined CSS and delay padding reduces period by up to 37.7% for individual circuits.

The percentage power normalized to pre-CSS results is shown in the "no PDE" column. The increase averages 3-4% for both LUT sizes. In fact, the total dynamic power increase is much larger due to the PDEs, and this is shown in the total dynamic power, "PDE" column. We used the PDE designed in [11], extending it to 6 stages. As a result of more FETs as well as progressively larger FETs needed to provide large delays, each PDE adds roughly 45 units of power (45fF, activity of 1).

We compare the power overhead due to CSS for our approach with that used in [7], which used 4 phase-shifted global clocks. The power due to an extra global clock is estimated as follows. The placement size (rows $\times$ columns of CLBs) is obtained from VPR, and we assume these CLBs are laid over a spine-and-ribs clock network. For example, if a circuit is 10$\times$10 CLBs, then the power due to an extra clock is $C_{clk} = 10 \cdot C_{rib} + C_{spine} + C_{clb}$, and $C_{rib} = 10 \cdot C_{int}$, $C_{spine} = 10 \cdot C_{global}$, and $C_{clb} = 10 \times 10 \cdot C_{local} \cdot \%_{FF}$. C_{global}, C_{int} and C_{local} are the capacitance of the wire that spans the length of 1 CLB for global, intermediate and local type routing wire, calculated from [24]. $\%_{FF}$ is the percent of the CLBs that contain a flip flop. The power overhead of 3 extra global clocks is then $3 \cdot C_{clk}$. This is shown in the 4-clk column. Circuits with lower density of user FFs incur less overhead with our approach, while a higher density of FFs favors the approach in [7].

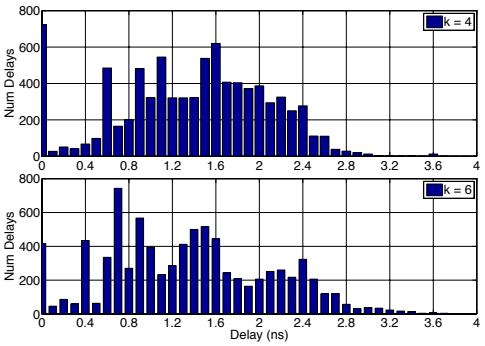

Fig. 8. Skew Histogram

We can decrease power overhead by decreasing the number of PDEs, e.g., by limiting each CLB to 1 PDE. In addition, there are certain nodes in the CSS solution that require zero skew. However, due to the nature of the PDE, zero skew cannot be achieved, so the entire schedule is shifted by the min-skew to maintain functionality. We can limit the CSS algorithm to use only skews higher than the min-skew, therefore avoiding the need to provide zero-skew nodes with PDEs. We may also decrease the power used by each PDE, using a more power efficient circuit design. In the PDE_{imp} column, we show the power overhead of our approach if we can decrease the number of PDEs by 10%, and the power used by each PDE by 10%. With these minor improvements, our approach usually uses less power than [7].

A histogram of the total number of PDEs used at each discrete delay for all circuits is shown in Fig. 8. The data is relatively spread out, indicating it is beneficial to use PDEs that can provide a large range of delays. This provides greater flexibility for CSS and delay padding beyond what is available with just 4 global clock lines. Area calculation is done with the model used in [11]. With 1 PDE assigned to each FF, 20 PDEs are needed per CLB, and the overhead is 11.7% for k=4 and 7.6% for k=6. This is the maximum area overhead with our approach. Future work will investigate PDE reduction techniques to achieve lower overhead.

B. GlitchLess Only Results

Next, we show that 100% glitch reduction is not possible without impacting performance. In Fig. 9, we show glitch power reduction vs. threshold. The y-axis is the normalized glitching power, and the x-axis is the ranking threshold percentage with respect to the node with the maximum glitching power in each circuit. The lines represent power savings as threshold is gradually decreased and more nets are de-glitched. The trend is rather gradual for threshold values larger than 20%, and it makes sense since there are only a few nodes with high glitching power consumption. As the threshold is lowered below 20%, more nodes became eligible for skew assignment

TABLE III
CSS PERFORMANCE GAINS AND POWER OVERHEAD

circuit	k = 4						k = 6					
	clock period (%)		dynamic power (%)				clock period (%)		dynamic power (%)			
	CSS	CSS+DP	no PDE	PDE	4-clk	PDE_{imp}	CSS	CSS+DP	no PDE	PDE	4-clk	PDE_{imp}
bigkey	92.2	92.2	103	162	287	150.4	65.6	65.6	101	182	348	166.8
clma	91.2	91.2	106	119	206	116.5	96.9	96.9	101	118	191	114.7
diffeq	78.3	78.3	99.3	636	567	533.6	84.8	**83.3**	99.6	812	475	675.6
dsip	73.6	**72.5**	100	153	264	142.9	62.3	62.3	100	172	316	158.0
elliptic	91.1	**68.1**	104	235	193	212.1	88.6	**74.5**	108	275	209	243.3
frisc	78.6	**74.4**	105	618	557	515.3	81.3	**68.3**	103	736	571	620.4
s298	99.7	99.7	111	114	168	113.6	100	100	107	111	156	109.8
s38417	97.6	97.6	102	168	167	155.3	97.9	97.9	113	261	199	232.5
s38584.1	87.7	87.7	102	156	165	145.5	88.1	88.1	102	174	178	159.8
tseng	78.3	**67.9**	99.8	391	332	337.3	76.6	**73.1**	104	463	384	393.7
geomean	86.5	**82.2**	103	224	261	204.3	83.2	**80.0**	104	260	276	234.1

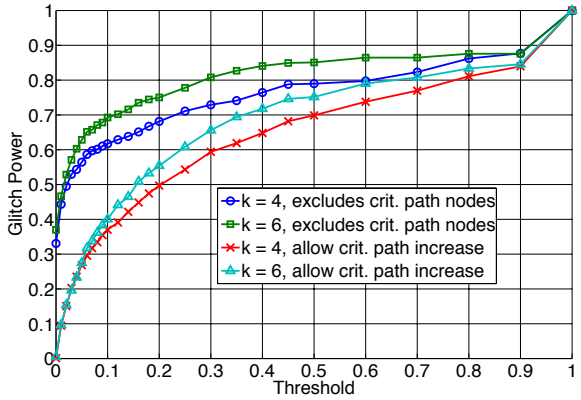

Fig. 9. Glitch Power Reduction Excluding PDE Overhead

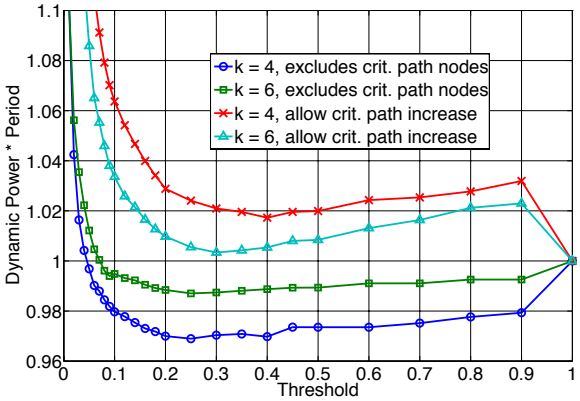

Fig. 10. GlitchLess Only Power Plot with PDE Overhead

and the rate of savings increases more dramatically. The square and circle lines show that even at a threshold of 0, total glitch elimination is not possible because critical and near-critical paths do not tolerate PDEs. All glitching can be eliminated only if an increase of the critical path is allowed, as the triangle and cross lines point out.

The impact of increasing critical path is shown in Fig. 10, where the y-axis is the normalized product between power and period. It's clear that the extra glitch power savings is not worth the increase in critical path, as the power-period product goes above 1.0, and is never better than the case where critical path is not allowed to increase. Note that since the period stays the same for the square and circle lines, they basically represent the power savings.

Also noteworthy is the number of PDEs necessary to achieve glitch reduction, and power overhead implications. In Fig. 11 the number of PDEs used is plotted on a log scale against threshold. Referring to Fig. 9, it is interesting to see that roughly a third of the savings can be obtained using less than 10 PDEs on average, for both k=4 and 6. When increasing the critical path, more PDEs are needed. As the threshold is lowered below 20%, the number of PDEs used increases dramatically, and the savings from the diminishing rate of return of each added PDE is over-compensated by the PDE's own power overhead. As a result, savings at low thresholds are gone, as was shown earlier in Fig. 10.

The best savings (P_{final}) for each circuit, where period is not allowed to increase, is presented in Table IV. The limit (P_{lim}) refers to the best possible savings achievable (assuming 100% glitch elimination with no PDE overhead), and the percentage of the limit achieved is also shown (%). While some circuits cannot be improved, their limiting case is close to 100% because glitching is only a small fraction of total dynamic power. Overall, the technique is able to remove on average 13% to 20% of glitching power, or up to 63% for individual circuits.

VII. CONCLUSION AND FUTURE WORK

This paper proposed an architecture change and associated tool flow to consider both power and performance optimization. A programmable delay element (PDE) added to each flip-flop clock input can be used to satisfy CSS, delay padding and GlitchLess simultaneously. The results are summarized in Fig. 12. CSS is able to improve performance by an average of 15%, or up to 37% for individual circuits. Some circuits can further benefit from delay padding, and their period can be reduced by 10% of the original period, or up to 23% of the original period for individual circuits, in addition to CSS improvements. We then discussed an improved method to estimate power due to glitching, and showed that the original

TABLE IV
DYNAMIC POWER AFTER GLITCHLESS

circuit	k = 4			k = 6		
	P_{final}	P_{lim}	%	P_{final}	P_{lim}	%
bigkey	100	98.2	0	100	98.4	0
clma	98.0	93.8	32.3	98.6	93.9	23.0
diffeq	100	96.6	0	100	97.9	0
dsip	100	98.8	0	100	98.6	0
elliptic	86.7	79.0	63.3	92.2	84.1	49.1
frisc	94.6	87.0	41.5	95.1	86.6	36.6
s298	100	78.3	0	98.3	76.8	7.3
s38417	85.7	73.2	53.4	98.8	85.6	8.3
s38584.1	99.5	92.2	6.4	99.6	96.0	10.0
tseng	99.6	84.8	2.6	100	91.6	0
average			20.0			13.4
geomean	96.3	87.8		98.2	90.7	

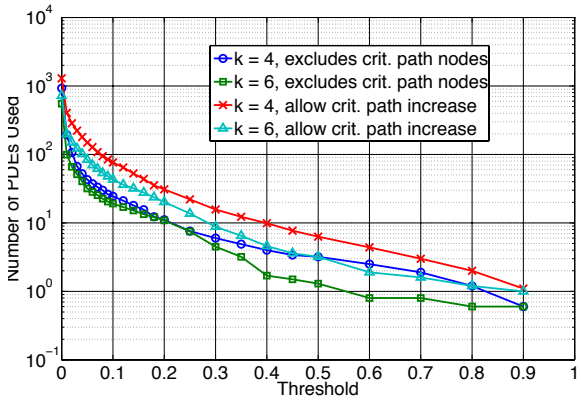

Fig. 11. Number of GlitchLess PDEs Used

method can underestimate glitching by as much as 48% while overestimating by 15%. The power overhead due to PDEs are compared to adding 3 extra global clocks, and our method is estimated to be 45% more power efficient on average. Lastly, we investigate the effect of glitch reduction on dynamic power. We were able to eliminate on average 16% of glitching power, and up to 63% for individual circuits.

As future work, it is important to evaluate the performance trade-off of using fewer PDEs to save power. Architecturally, moving PDEs up the clock distribution network saves both power and area ([8], [9]), but the restriction this imposes on performance (with delay padding) is not clear. Instead, algorithmically penalizing each PDE instance saves power as well, but retains flexibility for performance when needed. In addition, integration with retiming, with CSS added as fine tuning, can be done to exploit power overhead benefits.

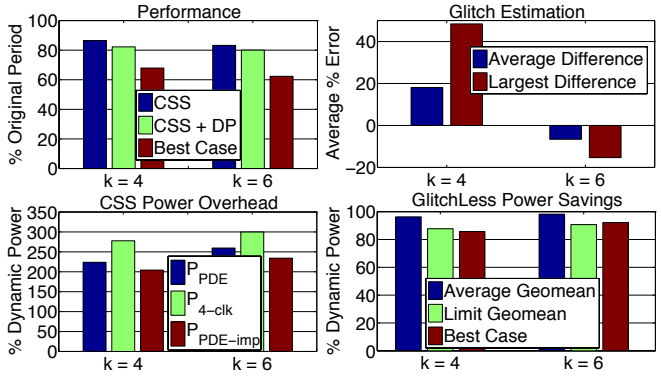

Fig. 12. Summarized Results

REFERENCES

[1] C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, Jun. 1991.

[2] J. Cong and C. Wu, "FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits," in *DAC*. Anaheim, California, United States: ACM, 1997, pp. 644–649.

[3] P. Pan and C. L. Liu, "Optimal clock period FPGA technology mapping for sequential circuits," *ACM TDAES*, vol. 3, no. 3, pp. 437–462, 1998.

[4] D. P. Singh and S. D. Brown, "Integrated retiming and placement for field programmable gate arrays," in *International Symposium on FPGAs*. Monterey, California, USA: ACM, 2002, pp. 67–76.

[5] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, 1990.

[6] R. Deokar and S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *IEEE International Symposium on Circuits and Systems*, 1994, pp. 407–410 vol.1.

[7] D. P. Singh and S. D. Brown, "Constrained clock shifting for field programmable gate arrays," in *International Symposium on FPGAs*. Monterey, California, USA: ACM, 2002, pp. 121–126.

[8] C. Yeh and M. Marek-Sadowska, "Skew-programmable clock design for FPGA and skew-aware placement," in *International Symposium on FPGAs*. Monterey, California, USA: ACM, 2005, pp. 33–40.

[9] S. Sivaswamy and K. Bazargan, "Statistical generic and Chip-Specific skew assignment for improving timing yield of FPGAs," in *ICFPL*, 2007, pp. 429–434.

[10] Altera Corporation, "Achieving low power in 65-nm Cyclone III FPGAs." [Online]. Available: http://www.altera.com/literature/wp/wp-01016.pdf

[11] J. Lamoureux, G. Lemieux, and S. Wilton, "GlitchLess: dynamic power minimization in FPGAs through edge alignment and glitch filtering," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 11, pp. 1521–1534, 2008.

[12] L. Cheng, D. Chen, and M. D. F. Wong, "GlitchMap: an FPGA technology mapper for low power considering glitches," in *DAC*. San Diego, California: ACM, 2007, pp. 318–323.

[13] Q. Dinh, D. Chen, and M. D. Wong, "A routing approach to reduce glitches in low power FPGAs," in *International Symposium on Physical Design*. San Diego, California, USA: ACM, 2009, pp. 99–106.

[14] B. Taskin and I. S. Kourtev, "Delay insertion method in clock skew scheduling," in *International Symposium on Physical Design*. San Francisco, California, USA: ACM, 2005, pp. 47–54.

[15] S. Huang, Y. Nieh, and F. Lu, "Race-condition-aware clock skew scheduling," in *Conference on Design Automation*. Anaheim, California, USA: ACM, 2005, pp. 475–478.

[16] S. Huang, C. Cheng, C. Chang, and Y. Nieh, "Clock period minimization with minimum delay insertion," in *Conference on Design Automation*. San Diego, California: ACM, 2007, pp. 970–975.

[17] C. Lin and H. Zhou, "Clock skew scheduling with delay padding for prescribed skew domains," in *ASP-DAC*, 2007, pp. 541–546.

[18] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *International Symposium on Field Programmable Gate Arrays*. Monterey, California, USA: ACM, 2009, pp. 133–142.

[19] J. Lamoureux and S. Wilton, "Activity estimation for Field-Programmable gate arrays," in *ICFPL*, 2006, pp. 1–8.

[20] P. Sedcole, J. Wong, and P. Cheung, "Modelling and compensating for clock skew variability in FPGAs," in *ICFPT*, 2008, pp. 217–224.

[21] T. H. Cormen, *Introduction to Algorithms*. MIT Press, 2001.

[22] J. Neves and E. Friedman, "Optimal clock skew scheduling tolerant to process variations," in *DAC*, 1996, pp. 623–628.

[23] (2008) iFAR - intelligent FPGA architecture repository. [Online]. Available: http://www.eecg.utoronto.ca/vpr/architectures/

[24] (2008) Predictive technology model (PTM) website. [Online]. Available: http://www.eas.asu.edu/ ptm/