A Routing Approach to Reduce Glitches in Low Power FPGAs

Quang Dinh, Deming Chen, Martin D. F. Wong
Department of Electrical and Computer Engineering
University of Illinois at Urbana Champaign
{qdinh2, dchen, mdfwong}@illinois.edu

ABSTRACT
Glitches (spurious transitions) are common in electronic circuits. In this paper we present a novel approach to reduce dynamic power in FPGAs by reducing glitches during the routing step. This approach involves finding alternative routes for early-arriving signals, so that signal arrival times at LUTs are aligned and no glitches are generated. This approach does not require additional circuitry to balance signals as done in previous work, but uses the available programmable routing resources instead. We develop an efficient algorithm to find routes with target delays. Based on this algorithm, we then build a glitch-aware router, named GlitchReroute, aiming at reducing dynamic power. To the best of our knowledge, this is the first glitch-aware routing algorithm for FPGAs. Experiments show that an average of 23% reduction in glitch power is achieved, which translates into a 9.8% reduction in dynamic power, compared to the glitch-unaware VPR router.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids – *Placement and Routing.*

General Terms
Algorithms, Experimentation.

Keywords
FPGAs, Low Power, Glitch Reduction, Path Balancing, Routing.

1. INTRODUCTION
The use of Field-Programmable Gate Arrays (FPGA) has become increasingly more wide-spread, due to progress in process technologies, architectures, and CAD tools. FPGAs are also well suited to changing requirements and short design cycles. FPGAs, however, are not power efficient. Reducing FPGA power consumption is important, as this lowers packaging and cooling costs and helps reliability.

ISPD'09, March 29–April 1, 2009, San Diego, California, USA.

There are two types of power sources in FPGAs: *static power* and *dynamic power*. Static power is the power consumed when there is no signal transition in a gate. Dynamic power, on the other hand, occurs whenever there is a signal transition, and this power includes *switching power* and *short-circuits power*. Dynamic power is a major power source for FPGAs. For example, in Altera's Stratix II FPGAs, 67% of the total power consumption is dynamic power [1].

Signal transitions, which directly determines dynamic power, are classified into two types. One type is *functional transitions*, the transitions required to perform the logic functions between two consecutive clock cycles. The other type is *spurious transitions*, or *glitches*. These are the unnecessary signal transitions caused by unbalanced arrival times of the inputs of the same LUTs: the output may switch when an early-arrival signal switches, and then may switch again when the critical-path signal switches. The dynamic power caused by these glitches is called *glitch power*. In FPGAs, glitch power can be a significant portion of total dynamic power. According to [2], glitch power can be 60% of the dynamic power. Therefore, it is very important to reduce glitches for power reduction.

In this paper, we present a routing approach to reduce dynamic power by balancing the paths to inputs of LUTs, so that signals of the same LUTs arrive at the same time and no glitches are generated. We are not aware of any works that look at reducing glitches through routing for FPGAs. Our approach involves finding alternative routes for early-arriving signals, such that the delays of the new routes causes the signals to arrive at the balanced times. Because only early-arriving signals are delayed more, the overall critical-path delay is not affected.

The uniqueness of this work is that this is a CAD-only approach, no modifications to existing FPGA architecture are required. Because FPGAs have a fixed architecture, traditional path-balancing techniques used in ASIC design, such as gate sizing and buffer insertion, are not applicable. Instead, we have to exploit the rich routing resources available to find alternative paths that have the desired delays. We use the real delay model to accurately balance paths. We also consider the power overhead when paths are lengthened.

Overall, our contributions can be summarized as follows:

- We propose a routing approach to reduce dynamic power by reducing glitches through path-balancing.
- We describe a novel, efficient algorithm to find a path with a desired delay between a source node and a sink node in a routing-resource graph.

- We develop GlitchReroute, which integrates the VPR router with the above algorithm to reroute the connections in FPGAs, while considering overhead issues so that dynamic power is reduced through minimizing glitches.

The rest of the paper is organized as follows. Section 2 provides some background information, including related works. Section 3 presents our detailed algorithm. The results are shown in section 4, and we conclude the paper in section 5.

2. BACKGROUND AND RELATED WORKS

Dynamic power can be modeled by the following formula:

$$P_{Dynamic} = 0.5 \cdot f \cdot {V_{dd}}^2 \cdot \sum_{i=1}^{n} C_i S_i \qquad (1)$$

where n is the total number of gates, f is the clock frequency, V_{dd} is the supply voltage, C_i is the load capacitance for gate i, and S_i is the switching activity for gate i. Switching activity is the average number of transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) a signal switches per unit time. The switch activity includes both functional transitions and glitches. Therefore, minimizing glitches is one way to reduce dynamic power.

Several glitch minimizing techniques have been published. CAD techniques includes logic decomposition [15] and RTL synthesis [16], etc. The gate freezing technique in [17] suppresses transitions until the frozen gate is enabled. This is only applicable to ASIC, because FPGA architecture is fixed at fabrication but the gates which should be frozen (which requires additional circuit) are not known until applications are implemented. The same difficulty in applying glitch minimization techniques in ASIC to FPGA is encountered with the delay insertion technique in [16].

There have been some studies to reduce glitches specifically targeted to FPGAs. GlitchMap [7] is an FPGA technology mapper that is glitch aware. Because no physical delay information is available at this mapping stage, GlitchMap uses the unit delay model to estimate delays. Glitches are minimized by favoring mapping solutions that balance LUT levels between different paths. Another study is GlitchLess [6], in which the authors proposed enhancing the FPGA architecture with programmable delay elements within the CLBs. After the glitch-unaware routing stage, these delay elements are used to align the arrival times of early-arriving signals. The authors report that the additional delay elements in the FPGA fabric take up to 5% area overhead and less than 1% delay overhead, while reducing power consumption by about 18%. The uniqueness of our routing approach to glitch reduction presented in this paper is that in FPGAs, interconnect delay is more significant than logic delay. Therefore, glitch reduction during or after routing is important. Our CAD-only approach can also be applied to existing commercial FPGAs, without the need of changing the architecture.

An FPGA consists of programmable logic blocks (CLBs), programmable I/O pads, and programmable interconnects. The interconnect structure is modeled as 2-D segmented wire channels connected through programmable switch boxes. The most popular FPGA architecture is the island-style FPGA, in which each CLB is surrounded by interconnects on all four sides. A high-level view of an island-style FPGA is illustrated in Figure 1 [10].

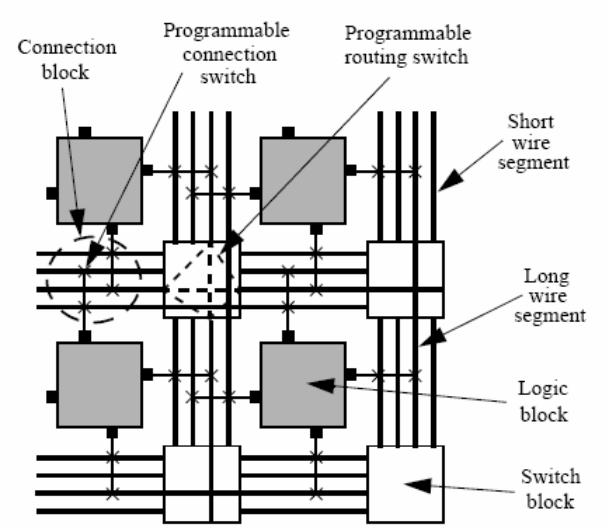

Figure 1. An Island-Style FPGA [10]

The routing resources of FPGAs consist of prefabricated wire segments and programmable switches. Once the location for all the logic blocks in a circuit have been chosen, the role of the router is to determine which programmable switches should be turned on to connect all the nets in the circuit. Many recently published FPGA routing algorithms are based on the negotiated congestion mechanism proposed in the PathFinder paper [8]. The main idea is to repeatedly rip-up and re-route every net until all congestion is resolved. A *routing iteration* is one pass of rip-up and re-route of every net once. The VPR [9][10] is a well known FPGA placement and routing system, which is open to the research community. The VPR router is based on the PathFinder algorithm, with a more careful cost scheduling. Another recent work used Lagrangian relaxation technique to guide the router towards a minimum critical path delay solution [11].

Most published works on FPGA router focus on congestion issues and timing performance. That is, they try to find paths that have minimum delay, while honoring the exclusivity constraints that no wire segments are used by more than one net. There is one study that considers short-path timing constraints [12]. The algorithm described in that paper extends the VPR router cost function to handle minimum delay budget, so that hold time requirements are met. There is no path balancing considered.

The routing architecture of an FPGA is represented by the routing resource graph $G_r = (V_r, E_r)$, which is a directed graph that describes available routing resources and the connections between them. The set of vertices V_r represents the input and output pins of logic blocks, and the wire segments. The set of edges E_r represents the feasible connections between the nodes. Figure 2 provides an example of a routing resource graph.

In this paper, we assume that the FPGA architecture only contains buffered switches for routing and does not contain pass transistor-based switches. This assumption matches well with current commercial FPGAs, where extensive amount of buffers are used to boost interconnect performance. Thus, the delay of each path

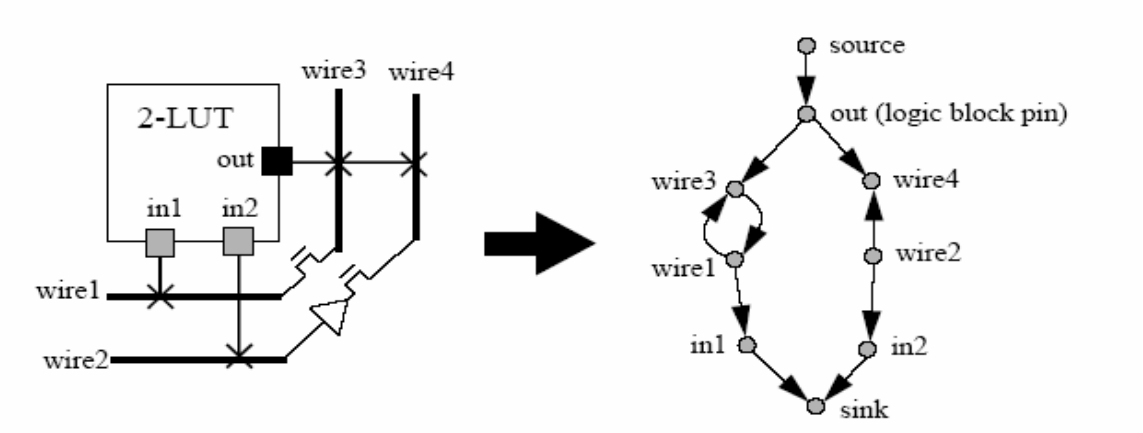

Figure 2. Modeling FPGA routing architecture using a routing resource graph [10]

can be linearly summed from each routing resource element under the real delay model.

3. ALGORITHM DESCRIPTION

3.1 Overview

To balance path delays during routing, our approach is to lengthen the short-delay paths to the same delay of the long-delay path when they all connect to the same LUT. We propose a final path-balancing routing pass, after a routing solution is found using any existing FPGA routers. This path-balancing routing pass has a similar structure to a normal routing iteration, in which some un-balanced nets are selected, ripped-up and re-routed to have balanced delays. To avoid congestion problem and to ensure that all nets are routed, our algorithm rips-up and re-routes one source-sink pair before balancing the next pair. In the following sections, we address two issues: how to select which pairs to be balanced, and in what order these selected pairs are to be balanced.

3.2 Selection Criteria

If we consider each path separately, lengthening its delay leads to increased power consumption by this particular path. This is because we need to select more wires with more capacitances and maybe more buffers as well. Thus, we should not try to balance every LUT inputs, but focus on some selected inputs. A simple, but effective selection scheme is to select the inputs to the first level clusters (those clusters that are the direct fan-outs of the primary inputs). This is because a glitch generated at this level could propagate to many other LUTs downstream. So it is more effective to balance the inputs of this 1st-level CLBs.

Results from our experiments (20 benchmarks as in section 4), shown in Table 1, confirm this approach. Comparing to the case when we balance inputs at all levels, for 1st-level balancing we increase the total estimated capacitance due to lengthened wires by only 14%, but we can reduce 53% of switching activity. This obviously represents a very good tradeoff, given that both capacitance and switching activity contribute to the dynamic power in a linear proportion (formula 1). If we balance all the 2nd-level CLBs as well, we observe that the total increase of the estimated capacitance is 2.6x compared to the 1st-level value but the reduction of switching activity is only 1.3x. Note that these values are normalized against the case, where all the levels are balanced shown in the last row of Table 1. Therefore, we conclude that we would focus on the 1st-level CLBs in this study.

Table 1. Switching Activity vs. Capacitance Tradeoff

Balanced Inputs	Normalized Reduction in Switching Activity (%)	Normalized Increase in Capacitance (%)
1st-Level	53	14
2nd-Level	71	36
All	100	100

3.3 Ordering Criteria

LUT Input Weighting: Because some inputs of a LUT have smaller influence on the LUT output than the other inputs, they are less likely to generate glitches when arrival times are not balanced. Therefore, we should spend less effort trying to balance the paths to these less critical inputs, by using the signal probabilities of the Boolean differences as path weights.

Assume that a LUT has n inputs x_1, x_2,… , x_n, and the function of the LUT is $f(x_1, x_2, \ldots, x_n)$. The Boolean difference of f with respect to x_i is defined as:

$$\frac{\partial f}{\partial x_i} = f_{x_i=0} \oplus f_{x_i=1} \qquad (2)$$

where $f_{x_i=0}$ (or $f_{x_i=1}$) is the function f when $x_i = 0$ (or $x_i = 1$).

Signal probability is the ratio of the time the signal is in logic 1 to the total observation time. Since a LUT has only a limited number of inputs, we use the famous Parker-McCluskey algorithm [13] to compute signal probabilities. Assuming that all inputs to the function f have signal probability of 0.5 (because these are the primary inputs), we can determine the signal probability $P\left(\frac{\partial f}{\partial x_i}\right)$ of the function $\frac{\partial f}{\partial x_i}$. The smaller this value is, the less influence of input x_i on the LUT output (a transition on x_i is

less likely to cause a transition on the LUT output). This signal probability is then used as the weight for this LUT input x_i.

Balancing Overhead: Paths that need more increased delays to be balanced generally introduced more dynamic power overhead. They also potentially use more wiring segments, which may prevent other paths to be balanced, due to the exclusivity constraints. Therefore, we should balance paths that need only small increases in delays first.

Path Ranking: We combine the two observations above to derive our final ranking for each path as follows:

$$Path\_Rank = \frac{LUT\_Input\_Weight}{Increased\_Delay} \quad (3)$$

By this formula, paths that are more likely to cause the output to switch, and paths that require smaller increases in delays to be balanced, are ranked higher and are selected to be balanced first.

3.4 Path-Finding Algorithm

In this section, we propose an algorithm to solve the single path balancing problem. We have an FPGA routing-resource graph $G_r = (V_r, E_r)$, with delay information for each node. For a source node *s* and a sink node *t*, we would like to find an (s,t) path such that the delay of the path falls within a specified target ($d \pm \Delta$), which is the range required to balance this path. Glitch filtering is handled by setting proper Δ.

3.4.1 Motivation

The basic idea is to generate a number of paths between *s* and *t*, and then check these candidates to see if any paths have the desired delay. To keep the runtime under control, we can not try every possible path, but need to have some effective heuristic to limit the search space. The Dijkstra's algorithm [14], used in most routers, is very efficient at finding the shortest path between a source node and a sink node in a routing-resource graph. To take advantage of this, we limit our search to only certain (s,t) paths that have some shortest-length properties.

One possible approach is to only consider paths that are the combinations of two shortest paths. That is, for each node $u \in V_r$, we look at the (s,t) path that is formed by the shortest path from s to u and the shortest path from u to t. These paths can be quickly enumerated, because only one run of the Dijkstra's algorithm is needed to find the shortest paths from s to every $u \in V_r$, and similarly for paths to t. However, the shortest path from s to u and the shortest path from u to t may overlap, making the combined path invalid, i.e. the delay of the path is wrong. To ensure that no overlapping occur, we select our candidate paths more carefully as follows.

3.4.2 Two Disjoint Sets

Let $shortest(u,v)$ be the shortest path from u to v. Let $d(u,v)$ be the distance (the length of the shortest path) between u and v.

We divide V_r into two disjoint sets S and T.

S is the set of nodes s' such that $d(s,s') < d(s',t)$. S is the set of nodes that are closer to s than to t.

T is the set of nodes t' such that $d(s,t') \geq d(t',t)$. T is the set of nodes that are closer to t than to s.

Clearly, S and T are disjoint, and $V_r \setminus S = T$. V_r is divided into two disjoint groups, S and T. We observe that S and T have the following property, which allow us to combine a shortest path in S with a shortest path in T to form a valid (s,t) path:

LEMMA 1: $shortest(s,s')$ lies entirely in S. Similarly, $shortest(t',t)$ lies entirely in T.

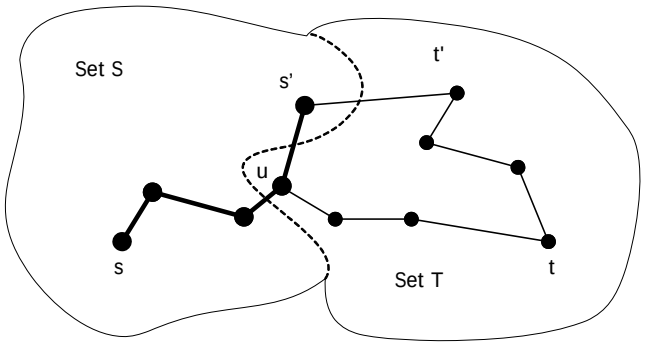

Figure 3. Shortest(s, s') lies entirely in Set S

Proof: Let u be the node next to s' in $shortest(s,s')$ (Figure 3). We will show that $u \in S$. Then by induction, all the remaining nodes of $shortest(s,s')$ are also in S.

By contradiction, assume that $u \in T$: $d(s,u) \geq d(u,t)$. Consider the (s',t) path from s' to u, then $shortest(u,t)$. The length of this path is $l(s',u) + d(u,t)$, where $l(s',u)$ is the length of edge (s',u). Then because $d(s',t)$ is the length of $shortest(s',t)$, we have:

$$l(s',u) + d(u,t) \geq d(s',t) \quad (4)$$

On the other hand:

$$d(s',t) > d(s,s') \quad (5)$$

$$d(s,s') = d(s,u) + l(s',u) \quad (6)$$

$$d(s,u) \geq d(u,t) \quad (7)$$

Thus:

$$d(s',t) > d(u,t) + l(s',u) \quad (8)$$

(4) and (8) contradict, therefore $u \notin T$, or $u \in S$. ■

3.4.3 Candidate Paths

For each pair of nodes $s' \in S$, $t' \in T$ such that the edge (s', t') exists, we form a candidate path $CP(s, s', t', t)$ by joining together $shortest(s, s')$, the edge (s', t'), and $shortest(t', t)$. An example is illustrated in Figure 4.

THEOREM 1: The candidate path $CP(s, s', t', t)$ is a valid (s, t) path, which means it does not overlap itself.

Based on Lemma 1, this theorem can be easily derived.

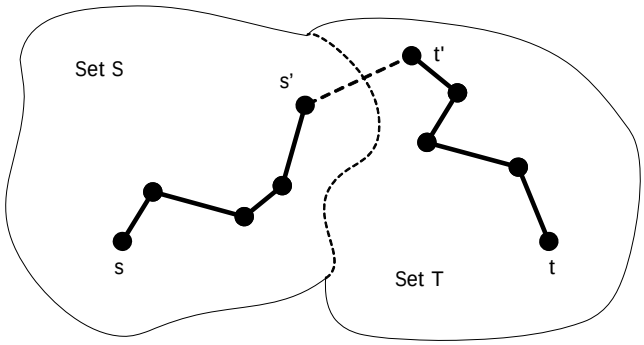

Figure 4. Candidate Path

Candidate Path Length: The length of the candidate path is:

$$D(s', t') = d(s, s') + l(s', t') + d(t', t) \quad (9)$$

where $l(s', t')$ is the length of edge (s', t').

The candidate paths have the following property, which allows us to significantly reduce the number of nodes to be searched.

THEOREM 2: We have $d(s, s') \le \frac{1}{2} D(s', t')$, and similarly $d(t', t) \le \frac{1}{2} D(s', t')$.

Proof: The path formed by joining edge (s', t') with $shortest(t', t)$ is an (s', t) path, therefore:

$$l(s', t') + d(t', t) \ge d(s', t) \quad (10)$$

But $d(s', t) > d(s, s')$ (as $s' \in S$), and from (9) we have:

$$D(s', t') = d(s, s') + [l(s', t') + d(t', t)] > d(s, s') + d(s, s')$$

■

With this result, when finding paths with desired length d, we only need to consider nodes $u \in V_r$ such that $d(s, u) \le \frac{1}{2} d$ and $d(u, t) \le \frac{1}{2} d$. This means that we do not need to find shortest paths for all nodes in V_r, but we can stop the Dijkstra's algorithm as soon as distances grow more than $\frac{1}{2} d$. We also need to examine fewer s', t' pairs for candidate paths.

3.4.4 Power Overhead Cost

When there are multiple candidate paths that meet the desired delay, we would like to select the path that has the least power overhead. Power overhead of a path is determined by the number of switch buffers (which consume both switching power and short-circuits power) and the total wire capacitance (which consumes only switching power). Therefore, a simple estimate of power overhead cost is the number of wire segments in the path, which is directly related to the number of switch buffers used. Note that different wire segments can have different lengths, based on the FPGA architecture specification. If two paths have the same number of wire segments, then the total wire capacitances are used to select the less overhead path.

3.4.5 Overall Algorithm

The overall path-finding algorithm is presented in Figure 5.

Input: Routing-resource graph V_r with a source node s and a sink node t, desired delay d and allowed margin Δ.

Output: Min-cost path *best_path* with delay within $d \pm \Delta$.

```
// Finding Shortest Paths and Distances
for every u ∈ V_r, initialize d(s,u) and d(u,t) to ∞
Find shortest(s,u) and d(s,u) up to distance ½d for u ∈ V_r.
Find shortest(u,t) and d(u,t) up to distance ½d for u ∈ V_r.
// Checking Candidate Paths
min_cost = ∞, best_path = ∅
for every s' ∈ V_r
  if d(s,s') ≤ ½d and d(s,s') < d(s',t)
    for every neighbor t' of s'
      if d(t',t) ≤ ½d and d(t',t) ≤ d(s,t')
        // Found a Candidate
        Calculate D_{s't'}
        if |D_{s't'} − d| ≤ Δ
          Calculate Path Cost cost
          if cost < min_cost
            min_cost = cost
            best_path = CP(s,s',t',t)
          end
        end
      end
    end
  end
end
```

Figure 5. Path-Finding Algorithm

Note that Δ is the gate inertial delay (glitches with pulse width less than Δ are filtered out), therefore it is very small comparing to the delays of the routing resources. Under this condition, the algorithm is guaranteed to find the best path from all candidate paths.

With the routing resource graph being a sparse graph (each routing resource can connect to only a limited number of other routing resources), the complexity of our algorithm is dominated by the shortest path search. This means the complexity of the Path-Finding Algorithm is $|V_r| \log |V_r|$.

3.4.6 Multi-sink Net Enhancement

For nets with multiple sinks, when we try to balance the path to a sink, the other sinks are already routed. We should take into account the existing paths that leads to the others sinks, by initializing the shortest path search with these paths in the routing resource graph.

3.5 Glitch-Reducing Framework

Our overall glitch-reducing framework is presented in Figure 6. It is the combination of the original VPR router with our path-balancing pass, named GlitchReroute. Any other FPGA routers can be used before the GlitchReroute pass.

Input: A placed netlist.

Output: A routed netlist with path-balancing.

```
Run timing-driven VPR router to get a routing solution

// GlitchReroute Path-balancing Pass
for each 1-st level CLB input
   get desired balanced delay
   compute path rank according to (3)
end

Sort these inputs by their rank

for each input
   rip-up current path
   use path-finding algorithm to find a path with the desired delay
   if can not find such a path
      restore the ripped-up path
   end
end
```

Figure 6. Glitch-Reducing Framework

4. EXPERIMENTAL RESULTS

To evaluate the effectiveness of our algorithm, we use 20 benchmarks in our experiments. They include 10 largest combinational circuits from the MCNC and ISCAS89 benchmark suites, respectively. Each circuit goes through technology independent logic optimization using SIS [4] and is technology-mapped to *K*-LUTs using DAOmap [5]. The mapped netlist is packed using T-VPACK into *N*-LUT clusters. This is then timing-driven placed and routed using VPR [10]. We then carry out the GlitchReroute algorithm on top of the VPR routing solution. Finally, we use the FPGA power simulator fpgaEVA-LP2 [3] to estimate dynamic power, glitch power, and switching activity.

This CAD flow is flexible, we can choose various parameters for LUT size *K* and cluster size *N*. In our experiment, we use $K = 4$ and $N = 4$. We assume a routing fabric containing buffered wires of two types, length-1 and length-4 (interconnect wires that span 1 CLB block and 4 CLB blocks, respectively). The FPGA array size is selected by VPR, which fits the given circuit well. The routing channel width W is selected as $W = 1.2W_{min}$, where W_{min} is the minimum channel width required to successfully route the circuit. This is to reflect the low-stress routing situation usually found in average circuits on commercial FPGAs.

To evaluate our proposed approach, we compare the results obtained with and without running GlitchReroute. The overall dynamic power results are presented in
Table 2. *Without* column is the dynamic power when the CAD flow is run without our GlitchReroute algorithm. This is the baseline result as obtained from the original VPR router. *With* column is the dynamic power when our GlitchReroute is included in the flow. The *Impr.* column shows the percentage of dynamic power that the GlitchReroute is able to reduce. We see that while the average improvement is nearly 10%, some circuits can have up to 19% improvement.

Table 2. Dynamic Power Consumption and Comparison

Circuits	Dynamic Power (mW)		Impr. (%)	Runtime (seconds)	
	without	with		without	with
alu4	38.20	35.29	6.45	22.8	42.7
apex2	40.13	39.75	1.26	34.6	82.4
apex4	17.65	16.20	8.10	20.9	22.2
des	67.53	59.95	9.92	25.9	101.9
ex1010	39.04	32.47	18.60	187.4	194.3
exp5p	19.78	15.83	16.87	15.1	16.8
misex3	32.33	29.19	7.67	19.2	32.4
pdc	33.11	29.35	12.28	212.3	243.4
seq	34.92	32.92	5.42	28.6	41.1
spla	33.80	31.09	8.76	139.2	148.1
C1355	3.71	3.24	12.04	0.6	1.1
C1908	7.82	7.13	8.94	1.0	1.5
C2670	12.64	11.87	5.24	4.0	8.3
C3540	21.54	19.05	11.45	3.5	3.8
C432	2.68	2.44	7.81	0.4	0.6
C499	3.59	3.07	13.78	0.6	0.8
C5315	34.08	31.03	8.82	7.2	14.0
C6288	73.20	59.39	18.84	4.9	6.1
C7552	43.51	38.55	11.40	9.1	17.9
C880	3.49	3.42	3.33	0.7	0.8
average			**9.85**		

Detailed performance of the GlitchReroute algorithm is presented in Table 3, where we report individual reduction in glitch power, reduction in switching activity, and the overhead of increased wire length (from those balanced paths). On average, GlitchReroute is able to reduce 23% of glitch power by reducing 13% of switching activity through glitch minimization. The overhead in increased wire length for path-balancing is about 8%. Because our algorithm only lengthens paths of early-arrival signals, the critical-path delays remain unchanged.

Table 3. Detailed Improvements and Overhead

Circuits	Reduction in Glitch Power (%)	Reduction in Switching Activity (%)	Increase in Wire Length (%)
alu4	38.89	8.01	5.51
apex2	7.48	5.23	9.45
apex4	20.35	10.18	7.10
des	20.81	11.94	8.44
ex1010	28.50	22.94	9.41
exp5p	25.79	21.02	8.43
misex3	28.43	9.35	6.36
pdc	31.63	15.26	7.29
seq	26.77	6.96	4.50
spla	16.17	10.96	7.50
C1355	34.76	15.22	7.80
C1908	21.25	11.52	7.62
C2670	19.09	6.57	4.68
C3540	20.92	14.21	9.99
C432	21.82	9.93	6.62
C499	33.45	17.75	8.62
C5315	18.82	11.10	7.42
C6288	18.05	22.84	9.23
C7552	21.32	13.98	9.74
C880	13.82	6.67	8.27
average	**23.41**	**12.58**	**7.70**

5. CONCLUSIONS

In this paper, we present a novel routing approach to reduce dynamic power in FPGA. By utilizing the routing fabric, we try to find paths that help balancing signal arrival times, therefore minimizing glitches. This unique CAD-only approach does not require changes to existing FPGA architectures, and it also does not affect critical-path delay. We develop an efficient path-finding algorithm that can find such balancing paths in the routing resource graph. We then build a framework around this algorithm to try to reduce glitches in a routed circuit, while taking into consideration the overhead associated with lengthening paths. The results from our GlitchReroute show that on average, 9.8% of dynamic power is reduced.

GlitchReroute can be used together with other glitch-minimization techniques for FPGA, such as technology mapping (GlitchMap [7]) or architectural enhancement (GlitchLess [6]), to reduce dynamic power even further.

6. ACKNOWLEDGMENTS

This work is partially supported by a grant from Altera Corp, by NSF grant CCF-0746608, and by NSF grant CCF-0701821. We wish also to thank Chen Dong, ECE, University of Illinois at Urbana-Champaign for his support on fpgaEVA-LP2.

7. REFERENCES

[1] Altera, Stratix II vs. Virtex-4 Power Comparison & Estimation Accuracy White Paper. http://altera.com/literature/wp/wp_s2v4_pwr_acc.pdf

[2] F. Li, D. Chen, L. He, and J. Cong. Architecture Evaluation for Power-Efficient FPGAs. *Intl. Symp. on Field-Programmable Gate Arrays*, pp. 175–184, 2003.

[3] F. Li, Y. Lin, L. He, D. Chen, and J. Cong. Power Modeling and Characteristics of Field Programmable Gate Arrays. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, Issue 11, pp. 1712-1724, 2005.

[4] E. M. Sentovich et. al. *SIS: A System for Sequential Circuit Synthesis*, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 1992.

[5] D. Chen and J. Cong, DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs. *Intl. Conf. on Computer-Aided Design*, pp. 752–759. 2004.

[6] J. Lamoureux, G. Lemieux, and S. Wilton. GlitchLess: An active glitch minimization technique for FPGAs. *Intl. Symp. on Field-Programmable Gate Arrays*, pp. 156–165, 2007.

[7] L. Cheng, D. Chen, and D.F. Wong. GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches. *Proc. Design Automation Conference*, 2007.

[8] L. McMurchie and C. Ebeling. PathFinder: A negotiation-based performance-driven router for FPGAs. *Intl. Symp. on Field-Programmable Gate Arrays*, 1995.

[9] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. *Proc. of the 7th Intl. Workshop on Field Programmable Logic*, 1997.

[10] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[11] S. Lee and M. D. F. Wong. Timing-driven routing for FPGAs based on Lagrangian relaxation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2003.

[12] R. Fung, V. Betz, and W. Chow. Simultaneous Short-Path and Long-Path Timing Optimization for FPGAs. In *Intl. Conf. on Computer-Aided Design*, 2004.

[13] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE trans. on Computers*, 1975.

[14] E. Dijkstra. A Note on Two Problems in Connextion with Graphs. *Numer. Math.*, Vol. 1, 1959, pp. 269–271.

[15] J. C. Monteiro and A. L. Oliveira. Finite state machine decomposition for low power. In *Proc. Design Automation Conference*, 1998.

[16] A. Raghunathan, S. Dey, and N. Jha. Register transfer level power optimization with emphasis on glitch analysis and reduction. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 18(8):1114-1131, Aug 1999.

[17] L. Benini et al. Glitch power minimization by selective gate freezing. *IEEE trans. VLSI*, 2000.