Special Section Short Papers

A Routing Approach to Reduce Glitches in Low Power FPGAs

Quang Dinh, *Student Member, IEEE,* Deming Chen, *Member, IEEE,* and Martin D. F. Wong, *Fellow, IEEE*

***Abstract*—This paper presents a novel approach to reduce dynamic power in field-programmable gate arrays (FPGAs) by reducing glitches during routing. It finds alternative routes for early-arriving signals so that signal arrival times at look-up tables are aligned. We developed an efficient algorithm to find routes with target delays and then built a glitch-aware router 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 27% reduction in glitch power is achieved, which translates into an 11% reduction in dynamic power, compared to the glitch-unaware versatile place and route's router.**

***Index Terms*—FPGA, glitch reduction, low power, path balancing, routing.**

I. Introduction

Signal transitions, which directly determine 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 look-up table (LUT, the basic logic cell of FPGAs). 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 [10], [11], glitch power can be 40% 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 LUT inputs, 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 cause the signals to arrive at the balanced times. Because only early-arriving signals are delayed, the overall critical-path is not affected.

II. Background and Related Work

Dynamic power can be modeled by the following formula:

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

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 switching activity includes both functional transitions and glitches. Therefore, minimizing glitches is one way to reduce dynamic power, as is done, for example, in the work presented in [14].

There have been some studies to reduce glitches specifically targeted to FPGAs. GlitchMap [4] is an FPGA technology mapper that is glitch aware. Glitches are minimized by favoring mapping solutions that balance LUT levels between different paths. Another study is GlitchLess [8], in which the authors proposed inserting programmable delay elements into the configurable logic blocks (CLBs). After a glitch-unaware routing stage, these delay elements are used to align signal arrival times. The additional delay elements take up to 6% area overhead and less than 1% delay overhead, while reducing power consumption by about 17%. The strength of our glitch reduction routing approach is that in FPGAs, interconnect delay is more significant than logic delay. Therefore, routing could introduce delays that cause unaligned signal arrival times and glitches. Our CAD-only approach can also be applied to existing commercial FPGAs.

Many recentlypublished FPGA routing algorithms are based on the negotiated congestion mechanism proposed in the PathFinder paper [12]. 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 versatile place and route (VPR) toolset [1], [2] is a well-known FPGA placement and routing system, with a PathFinder-based router. Another recent work used Lagrangian relaxation to find a minimum critical path delay solution [9].

Most published works on FPGA routers 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 [7]. The algorithm described in that paper extends the VPR router cost function to handle minimum delay budget. Path balancing is not considered.

III. Algorithm Description

A. 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 unbalanced nets are selected, ripped up, and re-routed to have balanced delays. To avoid congestion 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

Manuscript received June 13, 2009; revised August 19, 2009. Current version published January 22, 2010. This work was supported in part by Altera Corporation, San Jose, CA, by National Science Foundation (NSF) Grant CCF-0746608, and by NSF Grant CCF-0701821. This work is a transaction version of the authors' work in the Proceedings of International Symposium on Physical Design, 2009 [6]. This paper was recommended by Associate Editor P. Saxena.

The authors are with the Department of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: qdinh2@illinois.edu; dchen@illinois.edu; mdfwong@illinois.edu).

Digital Object Identifier 10.1109/TCAD.2009.2035564

TABLE I
SWITCHING ACTIVITY VERSUS CAPACITANCE TRADEOFF

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

two issues: how to select which pairs to be balanced, and in what order these selected pairs are to be balanced.

B. 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 use more wires with more capacitances and maybe more buffers as well. Thus, we should focus on only some selected paths. 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 these 1st-level CLBs.

Results from our experiments (20 benchmarks as in Section V), shown in Table I, confirm this approach. Comparing to the base case when we balance inputs at all levels, for 1st-level balancing, we increase the estimated overall circuit capacitance due to lengthened wires by only 14%, but we can reduce 53% of the overall circuit 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 2.6× compared to the 1st-level value but the reduction of switching activity is only 1.3×. Note that these values are normalized against the case where all the levels are balanced shown in the last row of Table I. Therefore, we conclude that we would focus on the 1st-level CLBs in this paper.

C. Ordering Criteria

1) *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 $x_1, x_2, \ldots, x_n$, 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

$$\frac{\partial f}{\partial x_i} = f_{x_i=0} \oplus f_{x_i=1} \tag{2}$$

where $f_{x_i=0}$ (resp. $f_{x_i=1}$) is f when $x_i = 0$ (resp. $x_i = 1$).

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 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 of the Boolean difference. The smaller this value, the less the influence of input x_i on the LUT output (a transition on x_i is less likely to cause a transition on the output). This signal probability is the weight for this input x_i.

2) *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 require small increases in delays first. Note that we do not need to balance paths with arrival time differences less than the inertial delay.

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

$$Path\_Rank = \frac{LUT\_Input\_Weight}{Increased\_Delay}. \tag{3}$$

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.

D. 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 due to inertial delay is handled by setting proper Δ.

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 cannot try every possible path, but need to have some effective heuristic to limit the search space. Dijkstra's algorithm [5], used in most routers, is very efficient at finding the shortest path between a source node and a sink node in a graph. To take advantage of this, we limit our search to only certain (s, t) paths that have some shortest-length properties.

One approach is to consider only paths that are 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 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 occurs, we select our candidate paths more carefully as follows.

1) *Two disjoint sets:* Let $shortest(u, v)$ be a shortest path from u to v. Let $d(u, v)$ be the distance (length of shortest path) between u and v.

   We divide V_r into two disjoint sets S and T as follows.

   a) 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.

   b) 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.

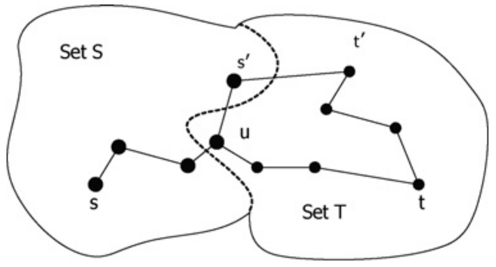

Fig. 1. Proof that *shortest* (s, s') lies entirely in set S (by contradiction).

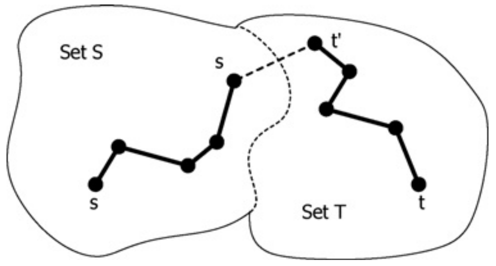

Fig. 2. Example of a candidate path by joining two shortest-path segments.

Clearly, S and T are disjoint, and $V_r \backslash S = T$. 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.

Proof: Let u be the node next to s' in *shortest* (s, s') (Fig. 1). 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

$$l(s', u) + d(u, t) \geq d(s', t). \tag{4}$$

On the other hand

$$d(s', t) > d(s, s') \tag{5}$$

$$d(s, s') = d(s, u) + l(s', u) \tag{6}$$

$$d(s, u) \geq d(u, t). \tag{7}$$

Thus

$$d(s', t) > d(u, t) + l(s', u). \tag{8}$$

Equations (4) and (8) contradict each other; therefore, $u \notin T$, or $u \in S$.

2) *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 Fig. 2.

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.

Candidate path length: The length of the candidate path is

$$D(s', t') = d(s, s') + l(s', t') + d(t', t) \tag{9}$$

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 $d(s, s') \leq (1/2)\,D(s', t')$, and similarly $d(t', t) \leq (1/2)\,D(s', t')$.

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

$$l(s', t') + d(t', t) \geq d(s', t). \tag{10}$$

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

$$\begin{aligned} D(s', t') &= d(s, s') + \left[l(s', t') + d(t', t)\right] \\ &> d(s, s') + d(s, s'). \end{aligned}$$ ■

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

2) *Overall Algorithm:* The overall path-finding algorithm is presented in Fig. 3.

Note that Δ is the gate inertial delay (glitches with pulse width less than Δ are filtered out); therefore, it is very small compared 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) *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 lead to the other sinks by initializing the shortest path search with these paths in the routing resource graph. Similarly, wire segments in the existing paths do not contribute to power overhead consideration.

E. Glitch-Reducing Framework

Our overall glitch-reducing framework is presented in Fig. 4. 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.

IV. Extended Candidate Paths

The candidate paths presented in Section III-D allow one edge between the two shortest-path segments. In this section, we discuss two extensions to the construction of the candidate paths, aiming at generating more paths with different lengths, thereby increasing the likelihood that we can find a path with the desired delay. First, we consider paths that have two edges between the two shortest-path segments. Second, we recursively look for better alternatives for the two shortest-path segments.

A. Two-Edge Extension

We consider a candidate path $CP_2(s, s', m, t', t)$ that is formed from three parts: *shortest* (s, s'), the two edges (s', m)

TABLE II
DYNAMIC POWER AND RUNTIME COMPARISON

Circuits	Size		VPR Dynamic Power (mW)	Dynamic Power Reduction (%)			VPR Runtime (s)	Runtime Overhead (%)			Glitch Power Reduction (%)	SA Reduction (%)	Wire Length Increase (%)	Track Width Usage		
	LUTs	Nets		A	B	C		A	B	C				Alloc.	VPR	Glitch-Reroute
alu4	1277	1291	38.20	6.45	8.19	8.27	22.8	87	126	137	45.42	9.76	5.95	31	31	31
apex2	1609	1647	40.13	1.26	1.34	1.35	34.6	138	204	227	7.76	5.78	10.04	35	35	35
apex4	1147	1156	17.65	8.10	9.22	9.25	20.9	6	39	45	24.11	12.46	8.07	40	39	40
des	1263	1519	67.53	9.92	10.89	11.44	25.9	293	382	399	24.36	14.80	9.38	17	17	17
ex1010	4052	4062	39.04	18.60	23.41	24.30	187.4	4	21	29	38.94	30.18	11.64	41	41	41
ex5p	954	962	19.78	16.87	18.30	18.36	15.1	11	30	36	33.78	24.94	10.36	38	38	38
misex3	1177	1191	32.33	7.67	8.36	8.85	19.2	69	82	105	31.74	11.29	7.27	33	32	33
pdc	3901	3917	33.11	12.28	15.45	15.82	212.3	15	42	50	40.98	18.89	8.64	50	48	50
seq	1427	1468	34.92	5.42	5.77	6.04	28.6	44	83	95	30.87	8.02	4.76	36	36	36
spla	3330	3346	33.80	8.76	9.55	9.87	139.2	6	22	28	19.62	13.38	8.42	43	43	43
bigkey	1818	2047	92.31	11.31	13.05	13.65	43.7	37	59	85	36.84	14.76	8.78	15	15	15
clma	6784	6846	69.90	5.19	7.62	8.53	235.3	15	25	30	21.83	14.87	9.94	42	42	42
diffeq	993	1057	9.12	7.35	8.23	9.84	10.6	78	96	104	23.13	5.93	4.41	13	12	13
dsip	1370	1599	78.44	2.16	3.59	3.77	30.6	64	87	108	12.35	6.04	8.41	18	18	18
elliptic	2146	2277	25.68	8.92	9.07	9.21	50.3	129	177	197	20.39	11.90	6.25	16	16	16
frisc	2335	2355	20.06	10.55	13.24	13.42	93.1	74	101	112	27.09	10.95	7.02	19	18	19
s298	1686	1690	13.83	4.91	6.91	7.33	47.9	16	43	52	19.95	13.92	7.20	16	16	16
s38417	5158	5187	32.91	9.03	9.73	9.89	115.8	37	51	65	22.85	27.18	9.90	16	16	16
s38584.1	4692	4730	68.85	14.52	16.02	16.49	77.2	41	60	75	22.65	17.98	11.10	15	15	15
tseng	782	834	10.07	12.88	14.31	14.53	10.2	53	62	68	43.36	21.25	8.61	11	11	11
Avg.				**9.11**	**10.61**	**11.01**		**61**	**90**	**102**	**27.40**	**14.71**	**8.31**			

```
Input: Routing-resource graph $V_r$ with a source node $s$ and a sink node $t$, desired delay $d$ and allowed margin $\Delta$.
Output: Min-cost path best_path with delay within $d \pm \Delta$.

// Finding Shortest Paths and Distances
for every $u \in V_r$, initialize $d(s,u)$ and $d(u,t)$ to $\infty$
Find $shortest(s,u)$ and $d(s,u)$ up to distance $\frac{1}{2}d$ for $u \in V_r$.
Find $shortest(u,t)$ and $d(u,t)$ up to distance $\frac{1}{2}d$ for $u \in V_r$.

// Checking Candidate Paths
min_cost = $\infty$, best_path = $\varnothing$
for every $s' \in V_r$ {
  if $d(s,s') \le \frac{1}{2}d$ and $d(s,s') < d(s',t)$ {
    for every neighbor $t'$ of $s'$ {
      if $d(t',t) \le \frac{1}{2}d$ and $d(t',t) \le d(s,t')$ {
        // Found a Candidate
        Calculate $D(s',t')$ as in (9)
        if $|D(s',t') - d| \le \Delta$ {
          Calculate Path Cost cost
          if cost < min_cost {
            min_cost = cost
            best_path = $CP(s,s',t',t)$
} } } } } }
```

Fig. 3. Path-finding algorithm.

```
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 formula (3)
}
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
} }
```

Fig. 4. Glitch-reducing framework.

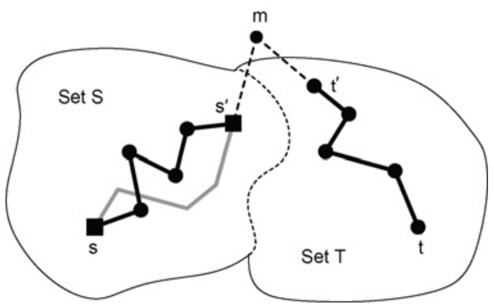

Fig. 5. Recursively find better path for *shortest* (s, s').

and (m, t'), and *shortest* (t', t). Fig. 5 illustrates such a path. Note that there are no restrictions on the node m: it can be inside set S, or inside set T, or neither (here S and T refer to the reduced sets of nodes within $(1/2)d$ away from s and t, respectively). The length of this candidate path is

$$D(s', m, t') = d(s, s') + l(s', m) + l(m, t') + d(t', t). \quad (11)$$

If *shortest* (s, m) goes through s', then $CP_2(s, s', m, t', t)$ is also $CP(s, m, t', t)$. This means the set of candidate paths described in Section III-D is a subset of the set of 2-edge candidate paths.

The overlapping problem would occur if either *shortest* (s, s') or *shortest* (t', t) goes through m. We could apply more constraints to the path by requiring that $d(s, m) > d(s, s')$ and $d(m, t) > d(t', t)$. This, however, would reduce the number of candidate paths. We instead decide to check on the overlapping condition for each combination of (s', m, t') by tracing through *shortest* (s, s') and *shortest* (t', t) looking for m.

B. Recursive Improvement

If there are no candidate paths with the desired delay, we can select the candidate path with the length closest to, but smaller than, the desired length. Then we can try to increase the length of this path by applying the same technique recursively on the two shortest-path segments. This is illustrated in Fig. 5.

In this example, we find a candidate path $CP_2\left(s, s', m, t', t\right)$ that has the closest length to the desired delay, but the difference is more than the allowed margin Δ. We would like to increase the delay a little more. Our recursive approach is to rip up the shortest path *shortest* $\left(s, s'\right)$, then apply our path-finding algorithm to the source-sink pair $\left(s, s'\right)$ to find a longer, more balanced path from s to s'. The path from s' to t is kept unchanged. Then we find a better alternative for *shortest* $\left(t, t'\right)$ similarly.

V. Experimental Results

To evaluate the effectiveness of our algorithm, we use 20 benchmarks in our experiments. They include the 20 largest combinational and sequential circuits from the Microelectronics Center of North Carolina benchmark suite. The sizes of the circuits are shown in Table II. Each circuit goes through technology-independent logic optimization using SIS [15] and is technology-mapped to K-LUTs using DAOmap [3]. The mapped netlist is packed using T-VPACK into N-LUT clusters. This is then timing-driven placed and routed using VPR [2]. We then carry out the GlitchReroute algorithm on top of the VPR routing solution. Finally, we use the FPGA power simulator fpgaEVA-LP2 [11] to estimate dynamic power, glitch power, and switching activity. Similar to [11], the process technology is 100 nm from [16]. All buffers are of minimum size.

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 one CLB block and four 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 a common practice in VPR [2] to reflect the low-stress routing situation usually found in circuits on commercial FPGAs.

To evaluate our proposed approach, we compare the results obtained with baseline VPR and three different algorithm combinations: (A) uses candidate paths presented in Section III-D; (B) uses two-edge candidate paths (Section IV-A); (C) further includes recursive path-finding (Section IV-B). The results are presented in Table II. We see that while the average dynamic power reduction is 11%, some circuits can have more than 20% reduction. We can also see that the most improvement is achieved by two-edge candidate paths (10.6% reduction). Recursive path-finding does not improve the results as much. The runtime overhead of GlitchReroute is also reported, which is very reasonable.

The next set of columns shows detailed performance of the GlitchReroute algorithm for combination (C), where we report individual reduction in glitch power, reduction in switching activity, and the overhead of increased wire length (due to the balanced paths). On average, GlitchReroute is able to reduce 14.7% of total switching activity (SA) by reducing 27.4% of glitch power through glitch minimization. The overhead in increased wire length for path-balancing is about 8.3%. Because our algorithm only lengthens paths of early-arrival signals, the critical-path delays remain unchanged.

The last set of columns show the routing track usage comparison. Both VPR and GlitchReroute are provided with the same number of tracks. We can see that GlitchReroute is able to exploit all available track widths for path balancing and glitch reduction. In most cases, VPR also use up all available track widths for routing. There are only a few circuits in which VPR uses one or two fewer track widths.

VI. Conclusion

In this paper, we present a novel routing approach, GlitchReroute, to reduce dynamic power in FPGA. By utilizing the routing fabric, we try to find paths that help to balance signal arrival times, thereby 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 show that, on average, dynamic power is reduced by 11%.

GlitchReroute can be used together with other glitch-minimization techniques for FPGAs, such as technology mapping (GlitchMap [4]), to reduce dynamic power even further.

References

[1] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. Int. Workshop Field Programmable Logic Applicat.*, 1997, pp. 213–222.

[2] V. Betz, J. Rose, and A. Marquardt, "Introduction," in *Architecture and CAD for Deep-Submicron FPGAs*. Boston, MA: Kluwer Academic, 1999, pp. 1–10.

[3] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. Int. Conf. Comput.-Aided Design*, 2004, pp. 752–759.

[4] L. Cheng, D. Chen, and D. F. Wong, "GlitchMap: An FPGA technology mapper for low power considering glitches," in *Proc. Design Autom. Conf.*, 2007, pp. 318–323.

[5] E. Dijkstra, "A note on two problems in connexion with graphs," in *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[6] Q. Dinh, D. Chen, and D. F. Wong, "A routing approach to reduce glitches in low power FPGAs," in *Proc. Int. Symp. Physical Design*, 2009, pp. 99–106.

[7] R. Fung, V. Betz, and W. Chow, "Simultaneous short-path and long-path timing optimization for FPGAs," in *Proc. Int. Conf. Comput.-Aided Design*, 2004, pp. 838–845.

[8] J. Lamoureux, G. Lemieux, and S. Wilton, "GlitchLess: Dynamic power minimization in FPGAs through edge alignment and glitch filtering," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 11, pp. 1521–1534, Nov. 2008.

[9] S. Lee and M. D. F. Wong, "Timing-driven routing for FPGAs based on Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 4, pp. 506–510, Apr. 2003.

[10] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, 2003, pp. 175–184.

[11] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 11, pp. 1712–1724, 2005.

[12] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[13] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Comput.*, vol. C-24, no. 6, pp. 668–670, Jun. 1975.

[14] A. Raghunathan, S. Dey, and N. K. Jha, "Register transfer level power optimization with emphasis on glitch analysis and reduction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 8, pp. 1114–1131, Aug. 1999.

[15] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: a system for sequential circuit synthesis," Dep. Electr. Eng. Comput. Sci., Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M92/41, 1992.

[16] Device Group, University of California, Berkeley. (2004, Jan.). *Predictive Technology Model* [Online]. Available: http://www-device.eecs.berkeley.edu/~bsim3

A Metal-Only-ECO Solver for Input-Slew and Output-Loading Violations

Chien-Pang Lu, *Member, IEEE,* Mango Chia-Tso Chao, *Member, IEEE,* Chen-Hsing Lo, *Member, IEEE,* and Chih-Wei Chang, *Member, IEEE*

***Abstract*—To reduce the time-to-market and photomask cost for advanced process technologies, metal-only engineering change order (ECO) has become a practical and attractive solution to handle incremental design changes. Due to limited spare cells in metal-only ECO, the new added netlist may often violate the input-slew and output-loading constraints and, in turn, delay or even fail the timing closure. This paper presents a framework, named metal-only ECO slew/cap solver (MOESS), to resolve the input-slew and output-loading violations by connecting spare cells onto the violated nets as buffers. MOESS performs two buffer-insertion schemes in a sequential manner to first minimize the number of inserted buffers and then resolve timing violations, if any. The experimental results based on industrial designs demonstrate that MOESS can resolve more violations with fewer inserted buffers and less central processing unit runtime compared to an electronic design automation vendor's solution.**

***Index Terms*—Engineering change order (ECO), physical design, slew/loading violation.**

I. Introduction

For current process technologies, the cost of photomasks increases dramatically per generation [1], [2]. To reduce this expensive cost of photomasks, the incremental design changes are enforced to be implemented by changing only the metal layers while the base layers (for cells) remain the same. As a result, the original photomasks used for printing the cells can be reused in the next tape-out. This reuse of the base-layer photomasks can not only save the cost of photomasks themselves but also reduce the tape-out turn-around time since the base layers could be manufactured in advance. This type of incremental design changes is referred to as metal-only engineering change order (ECO).

To realize metal-only ECO, some design techniques have to be developed. First, spare cells need to be spread all over the design so that the change can be made at every possible location. This spare-cell allocation determines the affordable ECO size and its area overhead. Electronic design automation (EDA) vendors already provide some solutions to it. Second, a more complicated router is required to efficiently handle a large number of existing obstacles and design rules in ECO. Some previous work addressed these issues by using an implicit connection graph [3], a graph-reduction technique [4], or a timing-aware router [5]. Third, the violations of timing factors may significantly increase after metal-only ECO. Thus, a solver which can automatically remove those timing-related violations is needed to shorten the timing closure of metal-only ECO. Unfortunately, the current solutions provided by EDA vendors are not effective enough.

Manuscript received June 12, 2009; revised August 14, 2009. Current version published January 22, 2010. This paper was recommended by Associate Editor P. Saxena.

C.-P. Lu and C.-H. Lo are with Mstar Semiconductor, ChuPei, Hsinchu Hsien 302, Taiwan (e-mail: knuth.lu@mstarsemi.com; jack.lo@mstarsemi.com).

M. C.-T. Chao and C.-W. Chang are with the Department of Electronics Engineering, National Chiao-Tung University, Hsinchu 300, Taiwan (e-mail: mango@faculty.nctu.edu.tw; xesuschang@gmail.com).

Digital Object Identifier 10.1109/TCAD.2009.2040011

Input slew and output loading are two important timing factors to sign off the timing closure, which are limited by the slew constraint and loading constraint, respectively. Any violation to these two constraints may lead to a wrong timing estimation of the design, and in turn, degrade its performance and yield.

Several buffer-insertion techniques [6]–[10] are proposed to resolve the violation of the slew, loading, and timing constraint. However, most previous works assume that its gate placement is able to change, and hence cannot be applied to metal-only ECO.

In metal-only ECO, solving the timing-related violation relies on the utilization of pre-placed spare gates. [11] proposed a technology-remapping technique to fix timing violations, which may require more pre-placed spare cells to support the desired remapping. [12] inserts constant values to the inputs of spare cells and applies a technology-mapping technique to replace the original cells with spare cells. It may require more universal but larger-area spare cells, such as and-or-invert and multiplexer.

Some commercial tools also provide options to support buffer insertions in metal-only ECO. However, the final location of the inserted buffers often deviates from the ideal location due to the lack of physical information on spare cells and routing resources during the buffer insertion.

This paper presents a metal-only-ECO framework, named metal-only ECO slew/cap solver (MOESS), which resolves slew and loading violations by using pre-placed spare gates as inserted buffers. MOESS also can resolve the timing violations, implicitly or explicitly. For each violation, the proposed framework first finds the best buffer candidates from all spare gates and then utilizes a commercial back-end tool to insert the selected buffer through the tool's interface. Therefore, the focus of this framework is on how to accurately estimate the output loading (input slew) of the buffers newly inserted with the adopted back-end tool. This framework can be applied based on any commercial back-end tool as long as the design database can be queried through an open interface.

II. Proposed ECO Slew/Loading Solver

A. Overall Flow of MOESS

After the metal-only ECO is done by using netlist difference and spare cell mapping, tools will report the pins violating slew or loading constraints. Based on this timing report, MOESS will insert buffers through a commercial automatic placement and routing (APR) tool [15] to resolve each slew or loading violation. In MOESS, the slew constraint of a gate g can be transferred into a corresponding loading constraint, denoted as OAL_g (output available loading). The definition of OAL_g is the maximum output loading of g which can generate a output