A Small Survey on Glitch Minimization Approaches in FPGAs

Jérémie Dumas

École Normale Supérieure de Lyon

December 2011

This report will cover the topic of reducing the dynamic power consuption in FPGAs through the elemination of glitches, or spurious transitions, that occur in a circuit. Several approaches have been designed to tackle the issue at different stage of the FPGA conception (see either [LLW07], [DCW10] and [CCW07]). This document will focus on first two approaches, for they take place after the routing process.

Contents

1 Introduction

1.1 Architectural Background

We remind that a field-programmable gate array (FPGA) consists of configurable logic block elements interconnected through a net of wires running across the chip. The usual process of compiling a FPGA from its high-level description to the low-level transistors revolves around these three steps :

Technology Mapping The behavior of the circuit is expressed as a graph of gates (LUT).

Placement The gates are physically put on the chip.

Routing The switch boxes are configured to connect the logic block elements.

Wires' lengths and channels width is chosen so as to facilitate routing via approximated heuristics, because the underlying problem is usually NP-hard. Routing is usually devised to minimize essentially the delay of the critical path in the obtained architecture, where the critical path represent the longest path a signal can go through in the network.

While of primary importance, the length of the critical path in a FPGA is not the only criterion to be concerned about. Power consumption is also becoming more and more detrimental to reduce the cost of computing with FPGAs.

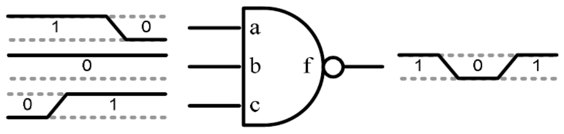
1.2 Power Dissipation and Glitches

There are two kinds of power dissipation in a FPGA. *Static power* dissipation occurs inevitably due to the nature of the transistors and current leaking through the components. *Dynamic power* dissipation is however inherent to individual signals toggling between two values. Lamoureux et al. [LLW07] indicate in their introduction that dynamic power account for 62% of the total power, and as such is a major source of power dissipation in FPGAs.

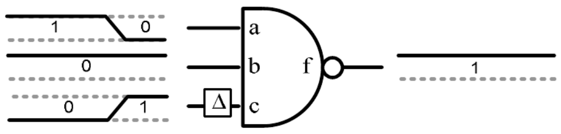
Transitions causing dynamic power dissipation can be distinguished between the *functional transitions* — which indicates a change in the gate input signal between two clock cycle — and *spurious transitions* or *glitches* — which do not change the output of the gate at the end of the cycle. This second kind of transition is an unwanted hazard we would like to suppress.

We denote by S_i the switching activity of gate i, i.e. the average number of signal transitions per unit time. Dinh et al. in [DCW10] used a dynamic power dissipation model that grow linearly in the S_i. The goal of both reviewed papers is to balance signal path, by delaying early-arriving signals, so as to align functional transitions and thus avoiding unnecessary glitches. A illustration can be seen in Figure 1.

Narrow shifts won't affect the output of the gate : the signal have to be separated by a minimal delay δ, called *intertial delay*, in order to create a significant glitch ; otherwise the glitch is filtered out naturally. Typical inertial delay is hinted to be $0.2ns$ in [LLW07].



(a) Original circuit with glitch



(b) Glitch removed by delaying input c

Figure 1: The idea behind glitch removal.

2 GlitchLess: Glitch Removal using Programmable Delay Element

2.1 The Programmable Delay Element

The cornerstone of the approach designed by Lamoureux et al. in [LLW07] is the programmable delay element illustrated in Figure 2. The idea is delay the signal passing trhough the circuit by an adjustable value Δ, controlled with the SRAM bits. They can be introduced in the logic block element of a FPGA in several ways, as hinted by Figure 3.

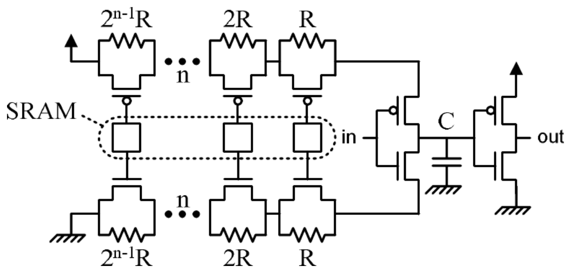
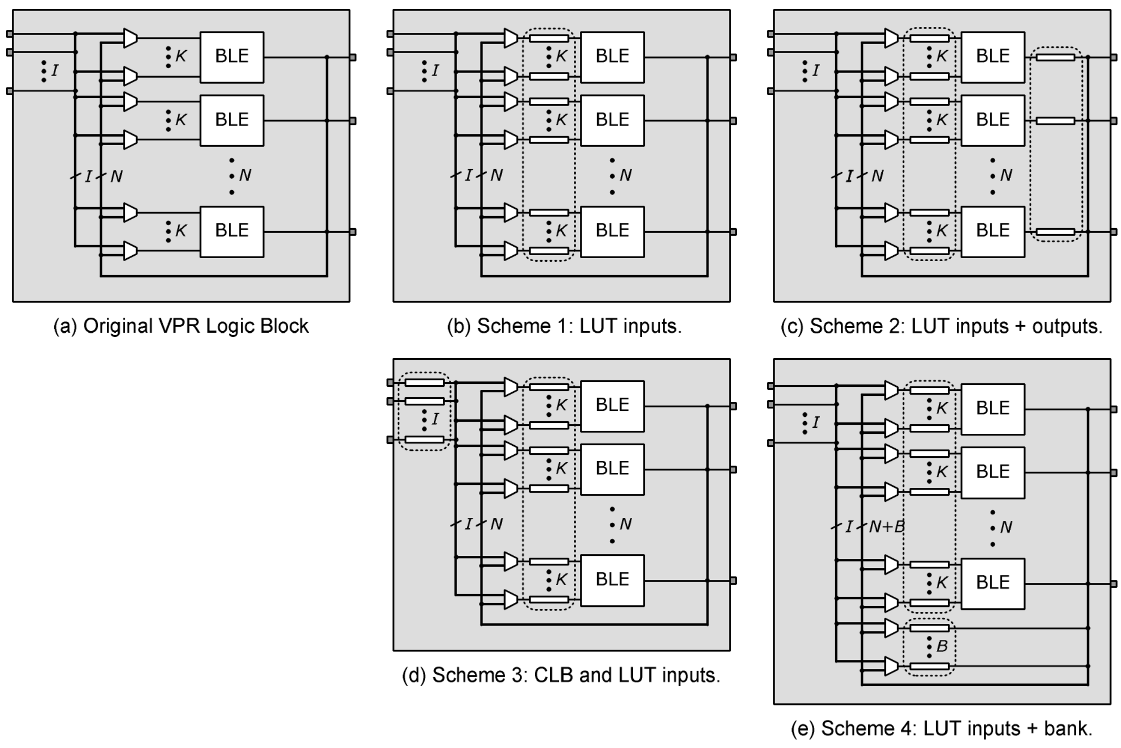


Figure 2: Programmable delay element.

In Figure 3, the Basic Logic Elements (BLEs) designate the combination of look-up tables (LUTs) and flip-flops. The LUTs have K input signals. The delay elements are controlled by several parameters, such as minimum and maximum delay they can incur, the number of delay elements at the input of a BLE, etc. When adjusting theses parameters, the more flexibility we get, the more important the area overhead will be.

2.2 Configuration Algorithms

Once the delay elements are placed, and the routing in the FPGA is done, we have to take advantage of the programmable delays in order to align input signals of the different



(a) Original VPR Logic Block (b) Scheme 1: LUT inputs. (c) Scheme 2: LUT inputs + outputs. (d) Scheme 3: CLB and LUT inputs. (e) Scheme 4: LUT inputs + bank.

Figure 3: Delay insertion schemes.

BLEs of each cell. In scheme 1 there is only one way to delay each input signal, but scheme 2 to 4 offer multiple ways to do so. The first step of the algorithm is to compute the ideal $Needed\_Delay(n, f)$ for each input signal f of node n. This is simply the difference between the arrival time of the two signals and the delay incurred by signal f when it goes through node n.

The configuration algorithm for scheme 1 boils down to setting the $Added\_Delay(n, f)$ for each input f of node n to be as close to $Needed\_Delay(n, f)$ as possible. For scheme 2, the configuration process makes use of the delay elements placed at the output of the BLEs to somehow factorize the minimum delay needed for each other BLEs' input it is connected to ; the algorithm visit each gate in the topological order of its gates from the primary inputs up to the final outputs of the circuit.

The configuration of scheme 3 and 4 works the same way, with a first step adjusting the I input delays in the logic block element (CLB) or the blank delay elements. When this is done, the delay element at the entrance of each BLE is configured the same way as with scheme 1.

2.3 Simulation Process

The experimentation protocol followed by Lamoureux et al. [LLW07] works as follow. The CAD tools used for placement and routing is the widespread Versatile Place and

Route (VPR) tool. Simulation of the generated FPGA is done using the HSPICE software, which is a comprehensive simulation tool for integrated circuits. It includes models for power dissipation, and allows to compute critical path overheads due to the insertion of the programmable delay elements.

Thorough calibration of each scheme parameters (such as minimum and maximum delays offered by the programmable delay elements) is done by simulation. Calibration is done over a wide range of benchmark circuits, using LUT with $K = 4$, 5 and 6 entries. To calibrate one parameter, ideal values for all the others are set, and empirical simulation yields the optimal value we are looking for. This multi-dimensional optimization is done quite empirically but it seems to bode well and gives coherent results — which will be presented in Section 4.

3 GlitchReroute: A Path Balancing Rerouting Algorithm

3.1 A Rerouting Heuristic

Instead of adding new delay elements which increase area and power consumption in a slight way, the principle of the rerouting algorithm proposed by Dinh et al. [DCW10] is to perform a new re-routing step to balance signal arrival at the input gates. As the algorithm only delay the early arriving signals, the critical path is not modifying and the new routing is still optimal from the graph-diameter point of view.

As with the *rip-up and re-route* approach of the VPR tool, this algorithm select a pair of node (s, t) and try to find a new route from s to t, whose length falls into a target range $[d - \Delta, d + \Delta]$, where Δ is the inertial delay mentioned in Section 1.2.

The pair (s, t) is chosen according to a certain heuristic, which favors the input that most influence the corresponding LUT's output on one hand, and that need smaller lifts on the other hand. Intuitively, inputs that generate more transitions of their LUT's are responsible for more glitches. And paths that need to be delayed by a long shot will increase significantly the number of wires needed for the routing, in a way impeding other paths to be lifted later in the process.

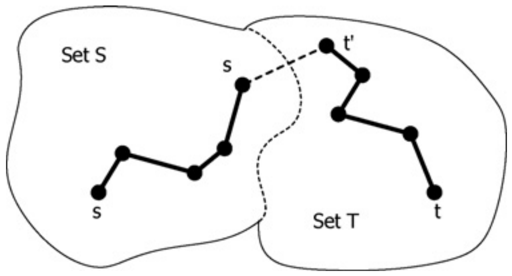
Remark. Their selection process consider only the first-level logic blocks, which are the fan-outs of the primary inputs of the FPGA. This is justified because it represents a very good trade-of between reduction of the switching activity and the increased capacitance caused by the use of more wire to route signals. Intuitively unaligned primary input signals will cause glitches that can ripple to more gates since they where created early in the graph.

3.2 Source to Sink Balancing

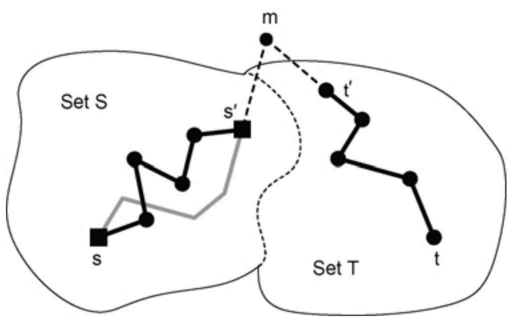
We are now interested in finding a new path between pair of nodes (s, t) through the wire network, with a delay in a window of $d \pm \Delta$. Because there are many paths between two node in a graph, the heuristic restricts its search to specific ones, composed of shortest paths from s to s', and shortest paths from t' to t, where s', t' are arbitrary nodes.

As the computed path needs to be simple (node wire may be used more than once), the following heuristic is used : split the graph in two sets S and T, where S contains nodes which are closer to s than to t, and T contains nodes which are closer to t than to s. Then any path of the form $s \rightsquigarrow s' - t' \rightsquigarrow t$ — where $s' \in S$, $t' \in T$ and $s \rightsquigarrow s'$ (resp. $t' \rightsquigarrow t$) is a shortest path from s to s' (resp. t' to t) — is a valid path with no loop (proof in [DCW10], see Figure 4a for an illustration).

Set S and T also avoid nodes distant from s or t by more than $\frac{1}{2} \cdot (d + \Delta)$, as they would induce a path from s to t that falls out of the desired window. Finally, since the delay are positive Dijsktra's algorithm can be used to find efficiently the shortest paths emanating from s and the shortest paths terminating in t.



(a) Single-edge path



(b) Two-edge and recursive

Figure 4: Finding a balanced path composed of two shortest path segments.

The authors also suggested two extensions of their path-finding heuristic (see Figure 4b):

Two-edge extension The candidate paths explored are of the form $s \ldots s', m, t' \ldots t$, where m can be any node in the graph.

Recursive improvement If no path of the desired length is found, choose an arbitrary path $s \ldots s', t' \ldots t$ with maximum length $\leqslant d - \Delta$, and lift recursively the paths associated to pairs (s, s') and (t', t).

Their experimental results indicated that the two-edge extension were the most efficient, while the recursive method did not improve the overall gain that much.

4 Results Comparison

4.1 Power Savings

Both GlitchLess [LLW07] and GlitchReroute [DCW10] methods claims to achieve close dynamic power savings : on average 18% for GlitchLess and 11% for GlitchReroute, but with respective reduction of the glitching activity by 91% and 27%. This highlight that primary inputs signals are responsible for a major part of the spurious transition in a circuit.

Benchmarks showed that the simpler scheme 1 for GlitchLess's insertion of programmable delay elements lead to the most satisfactory results.

4.2 Overhead

Power Overhead A consequence to both method is a slight power consumption overhead. For GlitchLess, the extra power dissipation comes from the use of the programmable delay elements themselves, but the overhead is around 1% in average for each scheme. For GlitchReroute, the extra consumption is explained by the increased wire length used in the new paths found.

Delay Overhead As GlitchLess inserts new delay elements in the logic blocks, an inevitable small increase in the critical path is expected. This is especially true with scheme 2 and 3, as every signal in a logic block must pass through a programmable delay element (there is no *fast-path* for critical signals). The GlitchReroute method is not affected by this issue since it only reroute signals. Still, delay overhead is only $\approx 0.2\%$ for scheme 1 and 4, whereas $\approx 2\%$ for scheme 2 and 3.

Area Overhead Yet an issue present in the GlitchLess method, but absent with the GlitchReroute approach. Indeed, the programmable delay element take place, and increase the global chip area by a factor of $\approx 5\%$.

4.3 Conclusion

The GlitchReroute algorithm has several advantages over GlitchLess method, as there is no area and delay overhead. However, it cancels less glitches when considering only primary input signals, and GlitchLess seems closer to the ideal dynamic power saving of 22.6%. Both methods take place after the routing process, are not incompatible with each other, and could be combined with other techniques such as GlitchMap [CCW07].

References

[CCW07] Lei Cheng, Deming Chen, and Martin D. F. Wong. GlitchMap: an FPGA technology mapper for low power considering glitches. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 318–323. ACM, 2007.

[DCW10] Quang Dinh, Deming Chen, and Martin D. F. Wong. A Routing Approach to Reduce Glitches in Low Power FPGAs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(2):235–240, 2010.

[LLW07] Julien Lamoureux, Guy G. Lemieux, and Steven J. E. Wilton. GlitchLess: an active glitch minimization technique for FPGAs. In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, FPGA '07, pages 156–165. ACM, 2007.