# Performance driven Routing for Modern FPGAs Invited Paper

Parivallal Kannan and Satish Sivaswamy
Xilinx Inc.
2100 Logic Dr.
San Jose, CA, USA
{parik, ssivasw}@xilinx.com

## ABSTRACT

FPGA routing is a well studied problem. Basic point-to-point routing of nets on FPGA fabrics can be done optimally using well known shortest path algorithms like Dijkstra's and A-star. Practical rip-up and reroute algorithms like PathFinder have been very influential in various academic and industrial routers. The relaxed version of the routing problem, that ignores congestion, has a polynomial time-complexity. This allows us to easily determine the best-case timing performance of a placed design and the degradation introduced by the router due to congestion alleviation on the datapath. Is it possible for an industrial router to actually improve upon the best case timing performance? Modern FPGAs like the Xilinx Ultrascale+ family, have introduced major changes to the clocking architecture to help improve clock skews and also to support a large number of user clocks. The new clocking architectures also allow the router to perform low-cost time-borrow optimization and efficient high fanout net routing, to alleviate fabric congestion and simplify timing-closure. We describe various methods by which industrial FPGA routers take advantage of the clocking features to improve timing performance of placed circuit designs beyond the traditional best-case scenarios.

## Keywords

FPGA, Routing, Skew, Hold, Congestion, Time-borrow, Performance

## 1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) have gone through a dramatic explosion in capacity and capability [7]. The simple Sea-of-Gates architectures of the earliest FPGAs have evolved to the modern Multi-Processing SOCs like the Xilinx Ultrascale+ family [16], with embedded multi-core CPUs, high-speed transceivers, DSPs and high-capacity memory blocks. Applications of FPGAs have likewise progressed from simple glue-logic to complete high performance programmable systems.

*ICCAD '16, November 07-10, 2016, Austin, TX, USA*
© 2016 ACM. ISBN 978-1-4503-4466-1/16/11...$15.00
DOI: http://dx.doi.org/10.1145/2966986.2980082

Timing closure has always been a major challenge for FPGA based high performance designs. Every aspect of the FPGA architecture and the tool-chain has to be optimized to extract the maximum possible performance. Logic-synthesis and placement have traditionally been the main determining factors for timing performance. In contrast, routing has largely been viewed as a QoR (quality of results) neutral step. As we explain in the next section, it is very straightforward to compute the best-case timing performance of a placed design and measure the gap, if any, versus a legally routed design.

It is during the routing phase, that we get really accurate delays, factoring in congestion and other factors. The real critical parts of the design get identified and optimized. While high-fidelity estimation models exist for both timing [10] and congestion [13], in practice it is not always possible or very efficient to account for all factors like non-linear delays, high fanout routing, hold fixing and local congestion. These issues can cause timing failures during routing, especially on high speed designs, with very low timing margins. Traditionally, this would require multiple iterations of the tools, or higher levels of margining in the timing and congestion models used by the tools, to guarantee timing closure during routing.

Recent advances in the FPGA clocking architectures, allow new avenues for performance improvements across the tool flow, including the router. Modern clocking architectures feature a very large mesh of global clock resources that can also be used for high fanout signal routing, to alleviate congestion. The clock buffers feature programmable delays that can be used by the router to perform *time borrow* [12] from the next sequential stage, to improve timing performance. The clock buffers can also convert the clocks into fixed-width pulses, which allow the router to optionally change user flip-flops into pulsed-latches. This introduces an alternate method to perform time-borrow on critical paths. Both these features allow the router to improve timing in a low cost manner, after the real critical paths have been identified. In the following sections, we describe the architectural advances in detail and the methods employed in industrial FPGA routers to push the performance envelope beyond what has been traditionally possible.

## 2. ROUTING FOR FPGAS

The FPGA routing problem is commonly modeled as a graph search problem. The FPGA interconnect is represented as a directed graph $G(V, E)$, where $V$ is the set of routing elements (nodes) and $E$ the programmable inter-

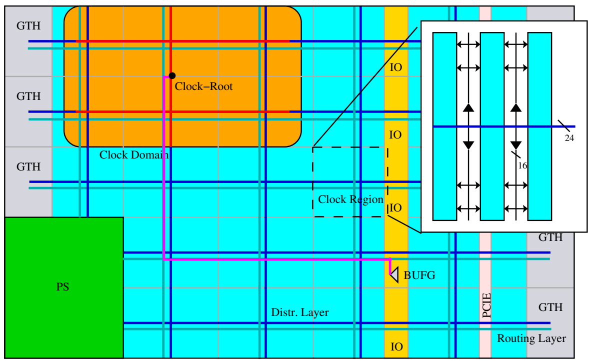

**Figure 1: Flexible FPGA Clocking Architecture**

connect switches (arcs). An edge $e_{ij} \in E$ is incident on exactly two vertices $(v_i, v_j), i \neq j$. The nodes and arcs are associated with a cost function that usually includes propagation delay and congestion. Each net $n_i$ of a placed circuit design has a driver (source) node $s_i \in V$ and a set of load (sink) nodes $t_i^j \in V$.

The best-cost path from the driver of a net to its loads can be found using well known algorithms like Dijkstra [3] and A-star [6], which guarantee optimal routes for every load. For higher fanout nets, its desirable to optimize the tree cost in addition to the individual driver-load costs. This is known to be a hard problem. Several practical algorithms based on Steiner approximations and Arborescence trees [1] [2] can be applied. Very high fanout net routing is a particularly challenging problem for FPGAs. These nets can span the entire device, cause congestion by consuming too many routing resources and adversely affect timing performance. Other approaches to net routing include SAT based formulations [9] and multi-commodity flow approximations [14].

Hold-time violations traditionally have not been viewed as a significant challenge for tools and are addressed in most tool-flows with post processing steps once the design is legal and has met all long-path constraints. With increasing process variations and faster technology nodes, this paradigm is no longer true. Hold problems are exacerbated by delay variation on the clock and data paths, in addition to the data paths themselves getting faster. In FPGAs, some common hold mitigation techniques such as buffer insertion and transistor sizing are not viable and we rely on the router to add enough routing delays on datapaths to fix hold violations.

Congestion is a major complicating factor for FPGA routing, owing to the fact that the routing fabric is pre-fabricated. Most FPGA routers use iterative ripup and reroute methods similar to [8]. The routes are allowed to overlap initially. The cost for overlaps, or congestion cost, is then gradually increased and the overlapping nets are ripped up and rerouted. Its easy to see that the initial solution is the best from timing point of view. A simple comparison of the post-legalization timing, with respect to this initial best-case timing gives a strong metric to measure the impact of congestion on timing. This gap is typically a very small value and has resulted in the expectation that FPGA routers are, or be, QoR neutral.

## 3. CLOCK ROUTING FOR FPGAS

Similar to the general purpose routing fabric, FPGA clocking resources are also pre-designed and programmable. The clock resources typically consist of a number of special clock drivers, leaf level distribution networks for logic cells and dedicated low latency clock tracks that can support a number of distinct user clocks. The topology of the clock tracks is typically chosen to allow low skew routing to the logic cells. Modern applications require FPGA architectures to support a large number of user clocks with flexible low skew routing. Most industrial designs feature a few truly global, system-level clocks and a large number of smaller module-level clocks. Clock skew is a major concern with FPGA clocks as it can reduce the cycle time available for the user-logic and can also cause hold-time violations. The clock resources are expensive in terms of metal area and it would be preferable to re-purpose any unused clock lines. Traditional FPGA clocking architectures had limited flexibility with respect to skew and the type of nets that can be routed on them. Clock routing has been largely viewed as a legalization problem and not an optimization problem. In the next section, we describe recent advances in FPGA clocking architectures, that address these short-comings and also open up new avenues for timing and congestion optimization.

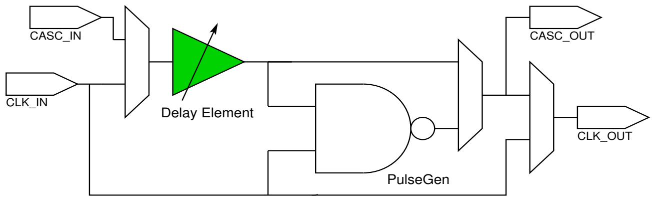

**Figure 2: Leaf Clock Delay and Pulse Generator Circuit**

### 3.1 Flexible Clocking Architecture

Figure-1 illustrates all the major components of a modern flexible clocking architecture for FPGAs. Additional details of the architecture and the components are available in [16]. The device is divided into a 2-D grid of rectangular sub-regions called *Clock-Regions*. Each clock region consists of any combination of columns of fabric (logic) blocks like CLBs, DSPs or BRAMs. The fabric columns are interleaved with Interconnect columns which contain the general purpose routing resources and also leaf level clock routing resources. The clock regions are served by a set of dedicated clock routing tracks called the *Distribution Layer*, that allow clocks to be routed to the clock leafs in every fabric column in the clock region. Logic placed inside a clock region will feature the best clock skew possible. Smaller module level clocks can typically fit in a clock region, thereby allowing the device to support a large number of clocks.

Multiple adjacent clock-regions can be combined to create a larger *clock domain region*, with the central region serving as the *clock root*. An additional layer of dedicated clock routing tracks called the *Routing Layer* allows clocks to be routed from the clock drivers (Eg: BUFG) to the clock-root. For system level clocks, the clock root can be the center of the device. The flexibility offered by this architecture allows the tools and designers to place system level and module level clocks at the most optimal location, virtually anywhere on the device. Placing the clock root near the centroid of the functional blocks and balancing the skew addresses a major problem facing high speed designs. Moreover, the regular

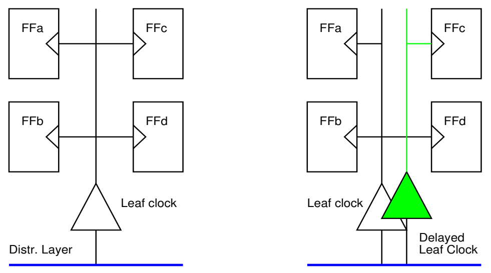

**Figure 3: Skewing clock using leaf clock delay**

mesh structure of the clock routing and distribution layers allow them to be useful for general purpose routing as well, especially, very high fanout signals.

## 3.2 Clock Leaf Architecture

The clock leafs are the last hop on the dedicated clock network and offer a convenient location for skew manipulation and optimization. Clock skews may have to be adjusted to help improve hold or as we describe in later sections, to improve the timing performance. [5] describes an architecture where leaf clock buffers provide fine grained control over the clock delays and hence the skews. Figure-2 shows the details of the architecture. The clock leaf buffers feature programmable delay elements that can be set to a specific delay value by selecting one among $n$ delay tap-settings. The delay elements can optionally be cascaded across multiple leaf buffers to achieve larger delay values.

Figure-3 shows an example of how the leaf clock delays can be used to skew clocks. The implementation on the left shows the original routing of the clock signal, to 4 sequential elements $FF_a$, $FF_b$, $FF_c$ and $FF_d$. All 4 are routed using a single leaf clock buffer. To skew the clock to $FF_c$ alone, a new leaf clock buffer, with its delay tap-setting set to a delayed value, is used to route the same clock signal, coming form the distribution layer. This is illustrated on the right of the Figure-3.

The leaf clock buffers also feature a NAND pulse generator to convert the incoming clock signal into a fixed-width pulse sequence. Existing flip-flops in the circuit can be converted into pulsed-latches, clocked by the pulse sequence. Latches allow time-borrow based timing optimization, as described in the next section. The advantage of pulsed-latch conversion over skewed clock approach above is that it can be applied even if there are no available free leaf buffers and the borrow amount can be of any value within the maximum pulse width.

## 4. CLOCK SKEW OPTIMIZATION

Time borrowing (or time stealing) is a well known concept [12] where a critical sequential stage borrows cycle time from the next stage, to help with timing closure. We can accomplish this either by delaying the capture-clock edge for edge-triggered flip-flops or by converting the capture flip-flop into a level-sensitive pulsed-latch and borrowing cycle time from the next stage that has a greater slack.

Figure-4 shows an example of time borrow using clock skew optimization. The sequential stage $\{FF_i, FF_j\}$ has a datapath delay of 2.5ns, which violates the cycle time of 2ns by 500ps. The next stage $\{FF_j, FF_k\}$, however has

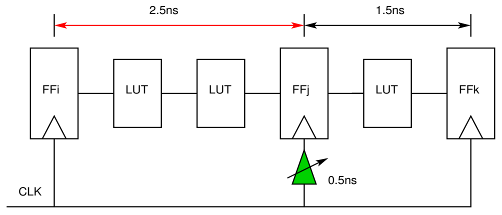

**Figure 4: Time borrowing using clock skew**

a positive slack of 500ps. By delaying the clock to $FF_j$, or in other words, by adding a skew of 500ps to the clock of $FF_j$, both stages now meet timing. The same can be achieved by converting $FF_j$ into a pulsed latch with a time-borrow amount of 500ps. The difference between these two approaches is that for delaying the clock edge, we may have to replicate the clock leaf and for pulsed-latches we need to configure the leaf clock as a pulse generator.

In practice, there might be paths ending on $FF_j$ with very little hold slack. Delaying the clock to $FF_j$ might result in hold fails. This can be corrected by either increasing the datapath delay of the hold failing paths, or by skewing (delaying) the clocks of their launching points.

## 4.1 ILP Model

We model the clock skew optimization problem as a global ILP formulation [4], that determines the optimal clock skews and pulse-latch settings for all sequential elements of the design, subject to all constraints, in order to maximize the performance improvement $f_{max}$.

Given a placed design with, $i = 1, 2, .., n$ path-groups, the objective function of the clock skew optimization problem can be stated as,

$$min \sum_{i}^{n} w_i * T_i \tag{1}$$

where $w_i$ is the relative weight of the path-group and $T_i$ is the period requirement for the path-group. Each flip flop $i$ in the design can be assigned a specific delay tap setting $j = 0, 1, ..., P$. We introduce a decision variable $d_{ij}$ to represent the assignment. $d_{ij} = 1$ implies, flop $i$ is assigned tap setting $j$. Since, each flop can take only one delay tap-setting,

$$\sum_{j=1}^{P} d_{ij} = 1 \tag{2}$$

## 4.2 Setup Constraints

Consider two flops a and b that form a setup constrained timing path. If $D_{ab}$ is the datapath delay, $T$ the period requirement for the constraint and $S_a, S_b$ are the clock-skews, then the following expression captures the setup constraint:

$$T + S_b - S_a \geq D_{ab} \tag{3}$$

The Delays for the tap-setting are known constants determined by the process.

$$S_b = \sum_{j=1}^{P} d_{bj} * DelayMin[j] \tag{4}$$

$$S_a = \sum_{j=1}^{P} d_{aj} * DelayMax[j] \tag{5}$$

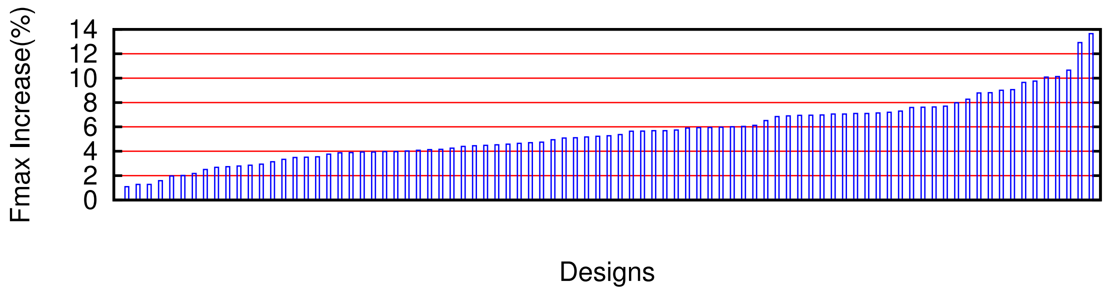

**Figure 5: Performance Improvement from Skew Optimization**

### 4.3 Hold Constraints

We use a similar method to model the hold constraints. We insert a margin $M_{ab}$ to represent the amount of hold that the router can fix at a later stage. This is determined using a heuristic based on the congestion in the neighborhood of the flops.

$$D_{ab} + M_{ab} \geq S_b - S_a \tag{6}$$

$$S_b = \sum_{j=1}^{P} d_{bj} * DelayMin[j] \tag{7}$$

$$S_a = \sum_{j=1}^{P} d_{aj} * DelayMax[j] \tag{8}$$

### 4.4 Pulsed Latch Constraints

Any flip flop can be converted to a pulsed-latch. Let binary variable $L_i$ represent if a flop $i$ is converted to a pulsed-latch and $B_i \in R^+$ represent the amount of delay borrowed from the next stage. The following equations summarize the properties of these variables.

$$\begin{aligned} B_i &\leq MaxPulseWidth \\ B_i - MaxPulseWidth * L_i &\leq 0 \\ B_i - L_i &\geq 0 \end{aligned} \tag{9}$$

Similar to Equations 3 and 6, we formulate setup and hold constraints, Equations 10 and 11 respectively, for potential latch candidates as follows:

$$S_b + B_b + L_b * \Delta_{setup} - S_a + L_a * \Delta_{c2q} - B_a + T \geq D_{ab} \tag{10}$$

$$-S_a + L_a * \Delta_{c2q} + S_b + L_b * MaxPulseWidth \leq D_{ab} \tag{11}$$

The constants $\Delta_{setup}$ and $\Delta_{c2q}$ represent the differences in timing parameters between flop and latch primitives, for setup and clock-to-q respectively. To ensure that a flop cannot be both flop and pulsed-latch simultaneously, we add the following exclusion constraints.

$$\forall_i \sum_{j} d_{ij} + L_i = 1 \tag{12}$$

### 4.5 Flop Graph Model

The formulation relies on efficient computation of the datapath delays $D_{ab}$ between sequential elements for all timing corners. A Flop Graph [15], is a specialized datastructure that was designed for this purpose. The vertices of a Flop Graph represent the sequential elements in the design. An edge between two vertices indicates the existence of a constrained timing path between them. The weight of each edge is the total data-path delay of the worst timing path between the two vertices. Timing critical endpoints are first identified and the fanin-cone reaching all of their start points are extracted. Arrival time propagation is started from the set of start points with unique timing tags and traversal is limited to the extracted fanin subgraph. When the critical endpoints are reached, the start and endpoint pair along with the datapath delays are extracted forming the flop-graph.

### 4.6 Experimental Results

We perform this optimization after all timing optimization steps in placement and routing are complete. This ensures that clock skew optimization is working on the set of timing critical endpoints that cannot be optimized further using conventional techniques. We extract the flop-graph, formulate and solve the ILP problem described above using a standard MIP solver. Finally, we implement the skew schedule returned from the optimization problem by configuring the programmable delays on the clock leafs to the appropriate delay values. Any hold violations introduced due to the margin $M_{ab}$ are fixed by adding routing delays to the datapath. We ensure setup is not violated while fixing hold. This is described in the next section.

Figure 5 presents the performance improvements from clock skew optimization on a suite of 89 large customer designs. Details of the designs can be found in [5]. On average, we see a 5.5% Fmax improvement with gains of up to 13.7% seen on high-speed designs.

## 5. SETUP AWARE HOLD ROUTING

High performance designs require hold constraints to be modeled along with setup and co-optimized as part of the negotiated congestion driven routing process. These designs also present a different dimension of the problem due to timing interactions between setup and hold critical paths which, if not handled properly can affect timing performance. This is illustrated in Figure 6.

In the figure, $FF_i$ are flops acting as start/end points of timing paths and $L_i$ are combinational elements. Hold failing paths are shown in red, while the paths marked in blue are setup critical timing paths. $FF_1$ is a start-point for hold path $FF_1 \rightsquigarrow FF_3$ and setup paths $FF_1 \rightsquigarrow FF_4, FF_5, FF_6$. $FF_2$ is the start-point for hold paths $FF_2 \rightsquigarrow FF_4, FF_5$. The connections $(FF_1, L_1)$, $(L_2, L_3)$, $(L_3, FF_5)$, $(L_3, FF_4)$ are part of both setup and hold critical paths and fixing hold on these connections would cause setup degradation. This can be avoided if we fix the hold violations only on connections $(L_1, FF_3)$, $(FF_2, L_2)$ leaving the connections on setup critical paths untouched.

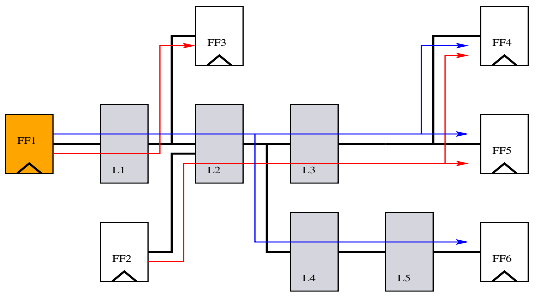

**Figure 6: Interaction between setup (blue) and hold (red) failing paths**

These types of path interactions are becoming increasingly common for ultra-high speed designs. Pushing design performance limits setup-margin on long-paths and the reduction in delays due to scaling makes short-paths longer, creating complex path interactions. The challenge here is to add routing detours to connections to satisfy hold time requirements with as little timing degradation as possible.

## 5.1 Setup and Hold Delay Budgeting

Timing engines treat setup and hold times as independent constraints and analyze them to provide slacks on design pins. Optimization tools would typically try to reduce delays on setup critical connections and increase those of hold failing connections to achieve timing closure. However, connections participating in both setup and hold failing paths will have conflicting delay requirements. We use an algorithm similar to the one outlined in Algorithm 1 to identify the connections to add delays for fixing hold violations. The algorithm gives maximum and minimum delay requirements, $(MaxDelay(v), MinDelay(v))$, for every pin in the design to achieve the best performance. These are used as constraints to guide the router's search space exploration.

```
Algorithm 1: Setup and Hold Budgeting
Input: Timing Graph, $G(V,E)$
Result: MaxDelay($v$), MinDelay($v$) $\forall v \in G$
begin
    Update Setup, Hold Slacks $(S(v), H(v))$, $\forall v \in G$
    Zero-Slack Delay Budgeting [11] to assign setup
      budgets, MaxDelay($v$), $\forall v \in G$
    foreach input pin, $v_i, H(v_i) < 0$ and $S(v_i) > 0$ do
        if $S(v_i) < abs(H(v_i))$ then
            $DlyAdded(v_i) = S(v_i)$
        else
            $DlyAdded(v_i) = abs(H(v_i))$
    while Worst Hold Slack(WHS) $< 0$ do
        $Delay(e_{ij}) = Delay(e_{ij}) + DlyAdded(v_j)$
        Incrementally update $H(v)$
        /* Relax setup to meet hold                    */
        if WHS $<$ 0 then
            Find unprocessed input pin, $v_i$, $H(v_i) < 0$
              with maximum setup slack
            $DlyAdded(v_i) = abs(H(v_i))$
    $\forall v_i$ where $DlyAdded(v_i) > 0$,
      MinDelay($v_i$) = $Delay(e_{ij})$
```

The results of the algorithm described in this section is presented in Figure 8. We use a suite of 16 large high-speed customer designs that exhibit complex setup and hold path interactions and we observe that on average, the algorithm gives a 1.7% $F_{max}$ improvement.

## 6. HIGH FANOUT ROUTING

Modern FPGA designs have a proliferation of high-fanout (few thousands of loads) signals stemming from the use global resets, address lines for wide memories etc. Such nets pose several challenges to implementation tools. Optimizing them is hard for placement tools because their connectivity and utilization demands will typically dictate spreading the net across the entire design footprint. From the routing perspective, these nets consume a large number of general purpose routing resources causing routing congestion and timing closure related issues.

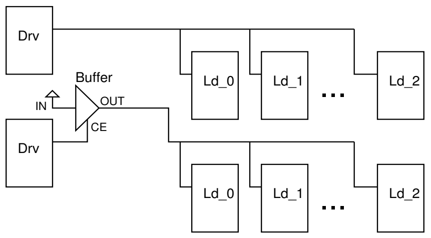

**Figure 7: High-Fanout routing with Global Buffers**

As noted in Section 3.1, modern FPGA clocking architectures can be used to route data signals as well, especially, very high fanout signals. The clock network is designed to reach pins with a minimal number of routing hops. This reduces the search space for the router, giving a significant runtime improvement. In addition, the clock network has guaranteed bounds on routing delays to all the pins in the device, thereby improving predictability and timing closure. Finally, migrating these nets from general interconnect to the dedicated clock network alleviates routing congestion by reducing routing demand and reserving the general interconnect for other nets in the design.

Using the clock network for routing high-fanout nets can be done either by instantiating clock buffers in the netlist or by the router configuring the buffers opportunistically during it's search process. Dynamic configuration of these buffers guarantees that placement results are not adversely impacted by the additional constraints and allows for greater optimization of true global clocks. For maximum flexibility, the clock buffers are provided with various levels of access to the clock network. These include global buffers that access the entire device, region level buffers that can access an entire clock region and leaf level buffers that can access a single column within a clock region.

Figure 7 illustrates how clock buffers are used by the router to route high fanout nets. The upper part of the figure shows a high fanout net driven by $Drv$ with loads $ld_0$, $ld_1$,...$ld_n$. The lower part shows the same net routed through a clock buffer. The router connects the driver, $Drv$ to the $CE$ pin of the buffer and ties off the $IN$ pin to $V_{cc}$. This allows the signal to appear on the $OUT$ pin of the buffer which is then routed to all the loads through the clock network. This operation effectively eliminates most of the routing demand due to the net, in addition to producing

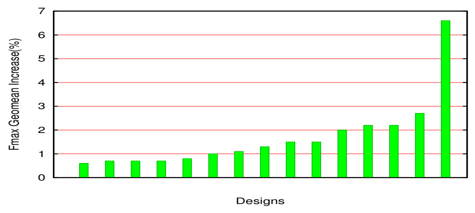

**Figure 8: Performance Improvement from Hold Budgeting**

more predictable timing. Suitable clock buffers are modeled as part of the interconnect graph, by introducing pseudo edges. This allows the automatic discovery of these paths by the router during search space exploration.

On a suite of large customer designs, routing high-fanout data signals on the global clock network results in a 1% QoR improvement on average, along with a 3.5% reduction in routed wirelength.

## 7. CONCLUSIONS

Routing is the last step in the physical design process for FPGAs. The most accurate timing and congestion picture is available only during routing. Any timing failures detected during routing are expensive and sometimes disruptive to correct. Recent advances in FPGA architectures, especially with respect to clocking, have opened up multiple opportunities to improve timing and congestion in the router. We have demonstrated practical methods to simplify timing closure problems, by optimizing clock skews and alleviating routing congestion using clocking resources. Our globally optimal, hold neutral, clock skew optimization algorithm improves $F_{max}$ by 5.5%. Reusing clocking resources for high fanout routing, reduces routed wirelength by 3.5% and improves $F_{max}$ by 1%. We also showed that high speed designs can see $F_{max}$ improvements of 1.7% by careful treatment of intersecting setup and hold paths. Our experience shows that contrary to popular notion, modern FPGA routers can actually be QoR positive.

The performance improvement methods described here can be applied during placement well. Placement tools can either apply these methods directly to improve performance or identify paths that can be easily improved by the router and focus on other parts of the design that truly limit the maximum achievable performance.

## 8. ACKNOWLEDGMENTS

Thanks to Nitin Deshmukh, Ilya Ganusov, Walt Manaker, Atul Srinivasan and Jindrich Zejda for enabling this work.

## 9. REFERENCES

[1] M. J. Alexander, , and G. Robins. New performance driven FPGA routing algorithms. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And System*, 15(12), December 1996.

[2] J. Cong, A. B. Kahng, and K.-S. Leung. Efficient algorithms for the minimum shortest path steiner arborescence problem with applications to vlsi physical design. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And System*, 17(1), January 1998.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGrawHill, second edition, 2001.

[4] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7), July 1990.

[5] I. Ganusov and B. Devlin. Time-borrowing platform in Xilinx Ultrascale+ family of FPGAs and MPSoCs. *FPL2016, International Conference on Field-Programmable Logic and Applications*, September 2016.

[6] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 1968.

[7] I. Kuon, R. Tessier, and J. Rose. FPGA architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2), February 2007.

[8] L. McMurchie and C. Ebeling. Pathfinder: a negotiation-based performance-driven router for FPGAs. *FPGA95, ACM third international symposium on Field-programmable gate arrays*, 15(12), 1995.

[9] G.-J. Nam, K. A. Sakallah, and R. A. Rutenbar. Satisfiability-based layout revisited: detailed routing of complex FPGAs via search-based boolean sat. *FPGA99, International symposium on Field programmable gate arrays*, 1999.

[10] E. S. Ochotta, P. J. Crotty, C. R. Erickson, C.-T. Huang, R. Jayaraman, and et.al. A novel predictable segmented FPGA routing architecture. *FPGA98, International symposium on Field programmable gate arrays*, 1998.

[11] R.Nair, C. Berman, P. S. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):860–874, August 1989.

[12] S. S. Sapatnekar. *Timing*. Kluwer, 2004.

[13] P. Saxena, R. S. Shelar, and S. Sapatnekar. *Routing Congestion in VLSI Circuits: Estimation and Optimization*. Springer US, 2007.

[14] E. Shragowitz and S. Keel. A global router based on a multicommodity flow model. *Integration, the VLSI Journal*, 5, March 1987.

[15] A. Srinivasan. A practical approach to clock skew optimization for FPGAs. http://www.tauworkshop.com/2016/slides/19_TAU_2016_Atul_ClockSkewOpt_invited.pdf, March 2016.

[16] Xilinx. Ultrascale Architecture and Product Overview. http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf, June 2016.