Open access • Proceedings Article • DOI:10.1109/FCCM.2019.00017

CRoute: A Fast High-Quality Timing-Driven Connection-Based FPGA Router — Source link

Dries Vercruyce, Elias Vansteenkiste, Dirk Stroobandt

Institutions: Ghent University

Published on: 01 Apr 2019 - Field-Programmable Custom Computing Machines

Topics: Router, Routing (electronic design automation), Physical design, Stratix and Clock rate

CRoute: A Fast High-quality Timing-driven Connection-based FPGA Router

Dries Vercruyce, Elias Vansteenkiste and Dirk Stroobandt
Department of Electronics and Information Systems
Computer Systems Lab, Ghent University
Ghent, Belgium

***Abstract*—FPGA routing is an important part of physical design as the programmable interconnection network requires the majority of the total silicon area and the connections largely contribute to delay and power. It should also occur with minimum runtime to enable efficient design exploration. In this work we elaborate on the concept of the connection-based routing principle. The algorithm is improved and a timing-driven version is introduced. The router, called CROUTE, is implemented in an easy to adapt FPGA CAD framework written in Java, which is publicly available on GitHub. Quality and runtime are compared to the state-of-the-art router in VPR 7.0.7. Benchmarking is done with the TITAN23 design suite, which consists of large heterogeneous designs targeted to a detailed representation of the Stratix IV FPGA. CROUTE gains in both the total wire-length and maximum clock frequency while reducing the routing runtime. The total wire-length reduces by 11% and the maximum clock frequency increases by 6%. These high-quality results are obtained in 3.4x less routing runtime.**

I. INTRODUCTION

An FPGA is an integrated circuit that can be programmed after fabrication to implement a digital design as the functionality is not fixed during the production process. For this purpose, the FPGA fabric consists of programmable logic blocks that are interconnected by a programmable interconnection network. Although this programmability introduces an overhead [1], it facilitates the design cycle, thereby reducing the non-recurrent engineering cost and time to market. Physical design translates the description of a digital circuit in a hardware design language to an FPGA configuration. Key metrics for efficient physical design are fast runtimes and high quality configurations. The configuration should efficiently use the available resources while maximizing the clock frequency. As FPGAs grow in size, the relative area of the routing infrastructure increases. Routing is therefore an important step. Many attempts have been employed to speed-up the routing process [2]–[10]. Some of them use algorithmic improvements and new techniques to speed up routing [2]–[7], while others rely on a multithreaded approach [7]–[10]. Improving the total wire-length and maximum clock frequency is difficult to obtain. The wire-length-driven CONR router [2] reports a wire-length gain of 5.8% and is further improved to a gain of 8% with the ideas proposed in [3]. The timing-driven RORA router [4] achieves a gain of 2.5% and 1.4% in maximum clock frequency and wire-length respectively.

Dries Vercruyce is supported by a PhD grant of the Research Foundation Flanders (FWO).

In this work we elaborate on the connection-based routing principle [2] and the ideas proposed in [3]. The result is a fast high-quality timing-driven router, called CROUTE, with a single threaded implementation. It has a finer granularity in routing connections [2], [4], instead of nets [11], [12]. The method is orthogonal to the multithreaded acceleration techniques and can thus be further accelerated. Algorithmic enhancements are proposed and a timing-driven version is introduced. Firstly, we adapt the cost of the wire segments and the calculation of the estimated remaining cost to the sink of a connection to cope with the heterogeneity of modern FPGAs. Furthermore, the negotiated sharing mechanism is improved by adding a bias cost, relative to the geometric mean of a net. The timing-driven version uses a wire-length and timing-driven direction factor to improve the runtime-quality trade-off. It also better estimates the initial connection criticality to ensure that the router does not unnecessarily focus on timing in the first routing iteration. Also, uncongested highly critical connections are rerouted to improve timing as they might have been routed with less timing importance in the first iteration. The rerouting of highly critical connections is similar to the work of Wang *et al.* [4]. Finally, the router detects illegal routing trees and fixes them after all congestion is resolved.

Quality and runtime are compared to VPR 7.0.7 [12] for a set of large heterogeneous benchmark designs, the TITAN23 design suite [13]. They are targeted to a detailed representation of the Stratix IV FPGA. CROUTE achieves a gain for both the quality of results and the required runtime. Total wire-length and maximum clock frequency are improved by 11% and 6% respectively, while being 3.4x faster. The router is implemented in an easy to adapt FPGA CAD framework, written in Java, and is publicly available on GitHub [14]. The framework also consists of a packing (MULTIPART [15]) and placement (LIQUID [16]) tool. With the addition of CROUTE, the framework is complete. It is now able to pack, place, and route the large heterogeneous TITAN23 benchmark designs faster, while gaining in quality of results when compared to the state-of-the-art VTR framework [12].

The remaining part of this article is organized as follows. In Section II the basic concept of the connection-based router is described. The enhancements to the original algorithm are explained in Section III. Section IV contains the timing-driven extension. Experiments are conducted in Section V and Section VI contains the conclusion.

II. Background

FPGA routers are typically based on the negotiated congestion mechanism, introduced by Pathfinder [17]. For all nets (signal between a source and one or multiple sinks), the router iteratively tries to find a disjunct routing tree in the routing-resource graph (RRG). This graph represents the routing resources and interconnectivity of the FPGA's programmable interconnection network. In each iteration, the pathfinder algorithm rips up and reroutes all nets until no resources are illegally shared. To ensure a legal solution, the cost of illegally shared resources is gradually increased.

In our work we further build on the concept of a connection-based router [2]. Instead of routing nets, each net is split up in a set of source-sink connections and these are routed independently. The routing problem then reduces to finding a simple path in the RRG for each connection in the circuit. These paths should only share nodes if the corresponding connections have the same source. The remaining paths should be disjunct in order to avoid short circuits. The connection-based router saves runtime as congested connections are rerouted instead of nets. This is in contrast with a net-based router that reroutes all (including uncongested) connections in a net if a single source-sink connection of the net is congested.

A. Routing a Connection

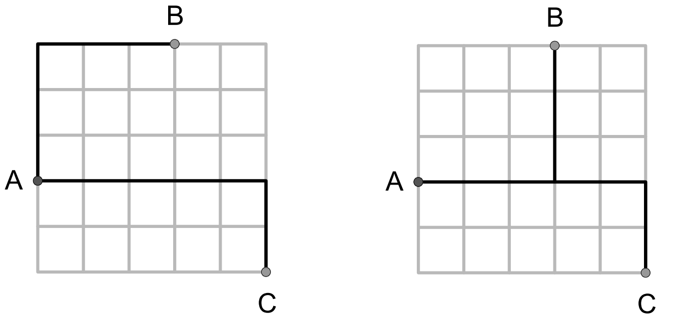
Starting from the source, a connection is routed by iteratively expanding nodes until the connection's sink is reached. In each expansion step, the node that results in the smallest estimated path cost $f(n)$ is analyzed by exploring all its downstream neighbors. The path cost of a node (1) consists of three parts: an upstream node cost, the congestion cost of the node (2) and an expected cost to the target sink (3), which is multiplied by a direction factor α.

$$f(n) = c_{prev}(n) + c(n) + \alpha \cdot c_{exp}(n) \tag{1}$$

The upstream path cost $c_{prev}(n)$ is the sum of the congestion costs of all route nodes along the upstream path from the current node to the source. The congestion cost $c(n)$ of a node is the product of its base cost $b(n)$, the present congestion penalty $p(n)$, and a historical congestion penalty $h(n)$. It is divided by a sharing factor $share(n)$, because multiple connections in a net can share the same node, as explained in Section II-B.

$$c(n) = \frac{b(n) \cdot p(n) \cdot h(n)}{1 + share(n)} \tag{2}$$

The expected cost $c_{exp}(n)$ enables a directed search to the target sink of a connection. Instead of expanding the node with the lowest congestion cost $c(n)$, which was done in the first routability-driven routers targeted to small FPGAs [11], the node that leads to the lowest path cost is expanded [18]. This results in a narrow wavefront that expands in the direction of the target sink, controlled by the direction factor α. The direction factor determines how aggressively the router explores towards the target sink. The expected cost is based on the expected number of wire segments that are required from the current node to the sink, multiplied by its base cost. The equation of the expected cost (3) is split up in three parts: one part for wire segments in the same direction as the wire segment under consideration, a second part for the orthogonal direction, and a third part that consists of the base cost of an input pin (b_{ipin}) and a sink pin (b_{sink}) as a routing path always ends with these pins. Splitting up the cost of the wires for wires in the same direction and for wires in the orthogonal direction is required because the wire segments in these directions may have different base costs (b_{same} and b_{ortho}). The expected cost in both directions is divided by the sharing factor $share(n)$ to allow an A* search [2].



(a) After iteration 1 (b) After iteration 2

Fig. 1. An example of (a) a suboptimal and (b) an optimal routing solution for a net with three terminals.

$$c_{exp}(n) = \frac{n_{same} \cdot b_{same}}{1 + share(n)} + \frac{n_{ortho} \cdot b_{ortho}}{1 + share(n)} + b_{ipin} + b_{sink} \tag{3}$$

B. Negotiated Sharing Mechanism

Connections can legally share routing nodes if they are driven by the same source. For this reason, the cost of a node for a connection should be lower in case it is already being used by other connections in the same net. It cannot be zero, because that would force the router to explore these nodes. Instead, the cost of a node in a connection is divided by a sharing factor $share(n)$, which is equal to the number of connections through this node that share the same source. The reason lies in the fact that the cost of a connection is the sum of the base costs of the nodes that realize the connection. If a node is shared between a number of connections that share the same source then the cost of that node has to be shared equally by all connections using it. By ripping up a connection at a time, the rest of the routing tree remains and influences the cost of the nodes through the $share(n)$ devision. This effectively encourages sharing routing resources over multiple pathfinder routing iterations.

In Fig. 1a a suboptimal routing tree is depicted. It is the result obtained after the first pathfinder routing iteration. The main cause why a suboptimal tree is generated, is that a large number of equivalent shortest paths are possible. For example, connection A-B has 20 equivalent paths with a minimum cost of 6 wire segments from which one is arbitrarily chosen. In the case of Fig. 1a this path does not allow any resource sharing for connection A-C. However, there are other possibilities that would enable more routing resource sharing and result in a routing tree closer to a minimum Steiner tree, as shown in Fig. 1b. This problem worsens if the manhattan distance between the source and sink of a connection increases, because the number of equivalent shortest paths increases exponentially. The sharing mechanism overcomes this problem. It is further improved in this work by introducing a bias cost relative to the geometric center of the net (Section III-C).

C. Routing Schedule

The present congestion penalty $p(n)$ of a node is updated whenever a connection is rerouted (4). Its value is based on the capacity, $cap(n)$, and occupancy, $occ(n)$, of a node. The occupancy is the number of nets that are currently using the node. It is thus equal to one if multiple connections in a net share the same node. The factor p_f is used to increase the illegal sharing cost as the algorithm progresses.

$$p(n) = \begin{cases} 1, & cap(n) > occ(n) \\ 1 + p_f(occ(n) - cap(n) + 1), & \text{otherwise} \end{cases} \tag{4}$$

The historical congestion penalty $h(n)$ is updated after every pathfinder routing iteration (5). The impact of $h(n)$ on the total resource cost is controlled by the factor h_f.

$$h^i(n) = \begin{cases} 1, & i = 1 \\ h^{i-1}(n), & cap(n) \geq occ(n) \\ h^{i-1}(n) & \\ \quad +h_f(occ(n) - cap(n)), & \text{otherwise} \end{cases} \tag{5}$$

The way the congestion factors p_f and h_f change as the algorithm progresses is called the routing schedule.

III. Algorithmic Enhancements

In this section we discuss the enhancements added to the connection-based routing algorithm. The aim is to improve the quality of results and to deal with the heterogeneity of modern FPGAs. First we explain how the cost functions are adapted to cope with a heterogeneous architecture that contains multiple wire segment types. Secondly a bias cost is used to improve the negotiated sharing mechanism.

A. Base Cost of the Wire Segments

Modern FPGA architectures have routing networks with multiple wire segment types [13]. Short wires enable the routability-driven routing of short connections, while long wires are added to improve the delay of the necessary long connections and hence improve the maximum clock frequency. The initial version of the connection-based router uses the same base cost for all wire segment types. However, the results in the experiments section show that introducing multiple wire segment types with different lengths requires a base cost adapted to its length. Long wires should have a larger cost to ensure that short wires are used if this reduces the wire-length without influencing the maximum delay. The base cost of the wire segments is therefore multiplied with its actual length.

B. Expected Distance to the Sink of a Connection

The expected remaining cost to the sink of a connection in equation (3) is based on an estimated number of wire segments that are required to reach that node. However, in case there are multiple wire segment types available, it is not possible to provide a good estimation as the length of the used wire segments is not known. Therefore, we do not estimate the number of wire segments, but use an estimation of the total wire-length instead. The estimation is based on the manhattan distance from the current node to the sink. It is split up in a same direction and orthogonal part, similar to equation (3). The distance is multiplied with an average cost per distance in both directions. The cost per distance ($\bar{c}_{same}$ and $\bar{c}_{ortho}$) is the average of a unit distance cost over all wire segments types, taking into account the number of wire segments of each type.

$$c_{exp}(n) = \frac{\delta_{same} \cdot \bar{c}_{same}}{1 + share(n)} + \frac{\delta_{ortho} \cdot \bar{c}_{ortho}}{1 + share(n)} + b_{ipin} + b_{sink} \tag{6}$$

C. Bias Cost

The negotiated sharing mechanism only works if one of the other connections is routed on a part of one of the shortest paths. When the router is routing the first connection of a net with Dijkstra, it is clueless about the location of the other sinks of the net. In order to help the router with initially choosing a good path from the equivalent shortest paths, a bias cost is added towards the geometric center of the net (7).

$$\begin{aligned} c(n) &= \frac{b(n) \cdot p(n) \cdot h(n)}{1 + share(n)} + c_{bias}(n) \\ c_{bias}(n) &= \frac{b(n)}{2 \cdot fanout} \cdot \frac{\delta_{m,c}}{HPWL} \end{aligned} \tag{7}$$

The bias cost must have a smaller influence than the wire cost as it is only meant to be a tie breaker. The minimum cost of a node is $b(n)/fanout$ in case a wire is shared by all of the connections in a net. The bias cost will thus be maximally half of the minimum wire cost. The bias cost depends on the manhattan distance to the geometric center of the net ($\delta_{m,c}$), which is normalized against its half perimeter wire-length (HPWL). As we close in on the geometric center, the effect reduces. During the negotiated congestion mechanism, the cost of the nodes can only increase. So the effect of the bias cost becomes smaller towards the later routing iterations.

IV. Timing-driven Connection Router

The connection-based routing principle is extended with a timing-driven implementation to optimize for minimum wire-length and maximum clock frequency simultaneously. A criticality (8) is assigned to all connections in the design during a static timing analysis in each pathfinder routing iteration. The criticality of a connection determines if it should be routed with minimum delay, in case the criticality is large, or with minimum wire-length, if the criticality is low.

$$f_{crit} = min\left[\left(1 - \frac{slack}{D_{max}}\right)^{\phi}, f_{crit,max}\right] \tag{8}$$

The slack of a connection (9) is calculated from the connection's delay (T_{del}), the arrival time of the source (T_{arr}), and the required time of the sink (T_{req}). The arrival and required times are calculated in a forward and backward traversal of the timing graph respectively.

$$slack = T_{req} - T_{arr} - T_{del} \tag{9}$$

The forward traversal calculates the arrival time of all nodes in the timing graph and gives us the maximum delay in the circuit (D_{max}). This maximum delay is set as the required time of the timing path leaf nodes in the backward traversal. The slacks are normalized to D_{max} (8). The result is a criticality between 0 and 1 for all connections in the design. It is larger as the delay of a connection is more critical and is capped at $f_{crit,max}$ to prevent deadlock in case a congested wire is occupied by several critical connections. Typically $f_{crit,max}$ is equal to 0.99.

In case a design has multiple clock domains, the traversals are repeated for each clock domain separately [12]. Paths between two clock domains are cut to ensure that the router optimizes for each clock domain separately [13]. The benchmark I/Os are constrained to a virtual I/O clock. To ensure that the router can not unrealistically ignore I/O timing, the paths between the netlist clock domains and the I/O domain are included [13].

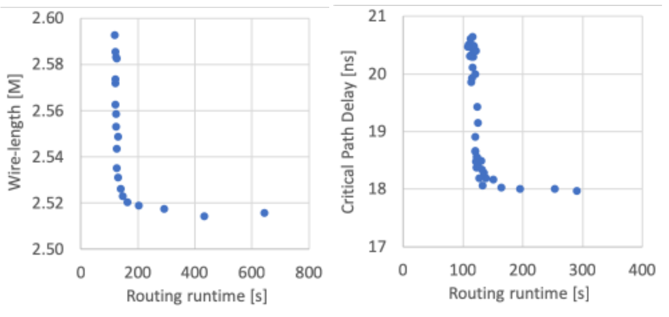
A. *Buffered Routing Switches*

The current implementation of the timing-driven router is designed for architectures with wire segments that are driven by buffered routing switches. A more detailed timing analysis is required to allow the routing of architectures with pass transistors. The capacitance and resistance of a wire is then dependent on the downstream capacitance and upstream resistance of routing resources along the connection. In a buffered architecture, the delay of a wire is only dependent on its own resistance, capacitance, and driving route switch. This simplification in the first timing-driven version is acceptable as many architectures are buffered. The detailed representation of the Stratix IV FPGA in the TITAN23 design suite [13] is fully buffered as well as the flagship architecture in the VTR framework [12].

B. *Timing-driven Cost Function*

The timing-driven router uses an adapted node cost (10) and expected cost (12) to ensure that critical connections focus more on reducing delay than on resolving congestion. The adapted node cost is the sum of a wire-length-driven cost and a timing-driven cost. The wire-length-driven cost of a node is set to its congestion cost (2). The timing-driven cost is equal to the delay of that routing resource. The relative importance of the wire-length-driven and the timing-driven part is determined by the criticality of the connection.

$$c(n) = (1 - f_{crit}) \cdot c(n)_{wld} + f_{crit} \cdot T_{del} \tag{10}$$



(a) Direction factor α: 2 to 1.05 (b) Direction factor β: 1.5 to 0.5

Fig. 2. Runtime-quality trade-off for (a) the wire-length-driven direction factor α and (b) the timing-driven direction factor β. The results are the average over the TITAN23 benchmark designs.

The adapted expected cost (12) consists of two parts: a wire-length-driven part, which is adapted from the wire-length-driven router (6), and a timing-driven part, which is based on an estimation of the remaining delay to the sink (11). The estimated delay to the sink is equal to the estimated distance to that node, multiplied by an average delay per distance in the same ($\bar{T}_{same}$) and orthogonal ($\bar{T}_{ortho}$) direction as the wire under consideration. The delay per distance for a direction is based on the average delay over all wire segments in that direction.

$$c_{exp,td}(n) = \delta_{same} \cdot \bar{T}_{same} + \delta_{ortho} \cdot \bar{T}_{ortho} \tag{11}$$

C. *Timing-driven and Wire-length-driven Direction Factors*

We assign a direction factor to the wire-length-driven and timing-driven expected costs (12) to enable a better runtime-quality trade-off. The direction factor α of the wire-length-driven part is large and allows a fast and aggressive search towards the target sink. Since the critical path delay is more important, a second, smaller direction factor β is used for the timing-driven part. This increases the runtime because more nodes are expanded, but leads to a lower maximum delay.

$$\begin{aligned} c_{exp}(n) &= (1 - f_{crit}) \cdot \alpha \cdot c_{exp,wld}(n) \\ &+ f_{crit} \cdot \beta \cdot c_{exp,td}(n) \end{aligned} \tag{12}$$

The runtime increase introduced by the small timing-driven direction factor β is minimized by using an exponent ϕ in the calculation of the criticality (8). We rely on the fact that the maximum clock frequency of a design is only determined by the longest paths. Paths with a small delay can therefore optimize for wire-length without affecting the maximum delay. The larger the value of ϕ, the smaller the criticality of the non-critical connections will be, resulting in a fast wire-length-driven routing of these connections with the aggressive direction factor α. This way, only the highly critical connections are routed with the slow timing-driven direction factor β.

The exact values of the direction factors α and β are important for a good runtime-quality trade-off. Small changes can largely increase runtime or reduce quality (Fig. 2). First we analyze the timing-driven direction factor β. The geomean runtime and critical path delay of CROUTE are shown relative to each other in Fig. 2b for a β varying from 1.5 to 0.5. Both the geomean runtime and critical path delay are highly sensitive to the exact value of β. The runtime is equal to 108 seconds if β is equal to 1.5 and increases by a factor of 2.7x to 291 seconds if β is reduced to 0.5, thereby gaining 13.9% in critical path delay. The wire-length is not sensitive to β, with a difference of only 0.4%. An optimal value for β is between 0.6 and 0.9. It is set to 0.7 in the experiments section.

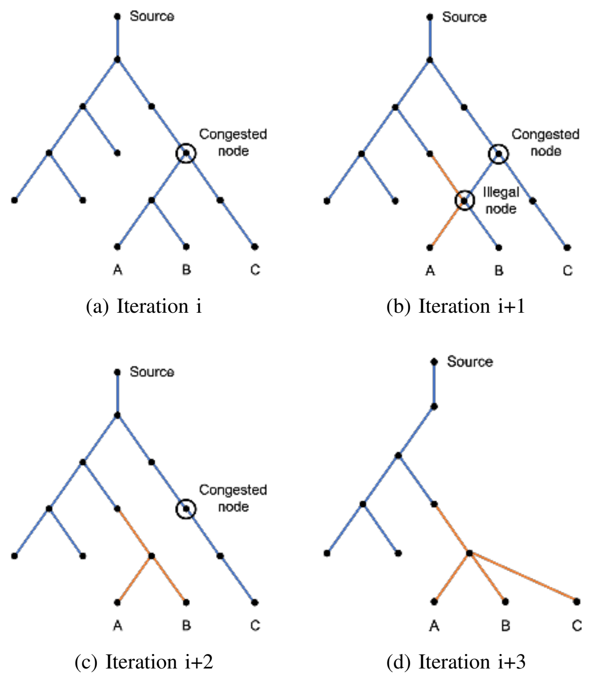
The value of the wire-length-driven direction factor α should be larger to enable an aggressive search towards the sink of a connection. Its value affects both the wire-length and critical path delay, but the influence is small. If α varies from 2 till 1.05, then the geomean wire-length and critical path delay improve by 3% and 1.2% respectively, at the cost of a 5.4x runtime increase (Fig. 2a). Further reducing α results in extremely long runtimes. The optimal runtime-quality trade-off is chosen at an α value of 1.4.

D. Initial Connection Criticality

In the first routing iteration, it is not possible to exactly calculate the criticality of the connections as the delay of the yet unrouted connections is not known. A possible solution is to perform a congestion oblivious first iteration by setting the criticality of all connections equal to one. The exact delay and the according criticality of the connections can then be calculated after the first iteration. We do not use this method as it stresses too much on reducing delay instead of congestion in the first routing iteration, even for the non-critical connections in the design. Therefore we use an estimated delay for the connections in the first iteration. This delay is equal to the optimistic congestion oblivious minimum delay used by placement tools. The minimum delay should be calculated only once for a given FPGA architecture and is already available as placement precedes routing. This way we relax the first routing iteration by only stressing on delay for the long paths in the design.

E. Reroute Critical Connections

The first routing iteration uses an optimistic estimation for the connection delays to calculate their criticalities. This leads to a non-minimal delay for connections with a low criticality. These connections, in turn, can affect the maximum clock frequency in later iterations if the delay of other connections is reduced. In each routing iteration, we therefore reroute all uncongested connections with a criticality larger than a predefined value θ_f. This allows previously routed uncongested connections to be rerouted if their delay limits the maximum clock frequency. The influence of rerouting connections is however small. The total runtime spend to reroute uncongested critical connections is only 2.5% of the total routing runtime as only the highly critical connections are rerouted. In case a design has many connections with a criticality larger than the threshold θ_f, its value is increased so that a maximum of 3% of the connections are rerouted.



(a) Iteration i (b) Iteration i+1

(c) Iteration i+2 (d) Iteration i+3

Fig. 3. Intermediary illegal routing tree

F. Illegal Routing Trees

The routing of a source-sink connection of a net is typically sped up by expanding from the (partial) routing tree of the already routed connections in that net [4], [12]. In CROUTE, each connection is routed from scratch, starting from the source, to maximally exploit the negotiated sharing mechanism. A drawback of this methodology is that illegal routing trees can occur. In case a node is congested, the router will try to circumvent the congestion. It is possible that the routing graph will be temporary illegal in-between iterations. An example is given in Fig. 3. Connections with sink A, B, and C use a congested node in iteration i (Fig. 3b) and the congestion mechanism is gradually solving the connection. In iteration $i+1$ (Fig. 3b) the routing graph is not a tree as it contains an illegal node that is driven by two nodes. The illegal tree is resolved in iteration $i+2$ (Fig. 3c) and all congestion is resolved in iteration $i+3$ (Fig. 3d).

In case there are remaining illegal routing trees after all congestion is resolved, the connections containing these illegal nodes are rerouted. An illegal routing tree occurs if a net consists of connections with a low criticality and connections with a high criticality. The connections with a low criticality use the lowest cost path in terms of congestion, while the connections with a high criticality use the lowest cost path in terms of delay. This problem is solved by a forced rerouting of all illegal connections in an illegal routing tree along the path of the connection with the highest criticality. Although this might slightly increase wire-length, the path of the highest criticality connection is used because the maximum clock frequency of a design is more important than its wire-length.

V. Experiments

The connection-based CRoute router is compared with VPR 7.0.7 (r75b47d3) in terms of quality of results and required time for routing. The Titan23 designs [13] are used for benchmarking. The target device is a model of the Stratix IV FPGA [13]. The FPGA dimensions are sized for each design separately. The RRG used by CRoute is extracted from VPR as a file containing the data of all routing resources and their interconnectivity. This enables a fair comparison between CRoute and VPR as it is hard to exactly duplicate the RRG generator of VPR. The routing runtimes of CRoute and VPR are the actual times required for routing and exclude the generation of the RRG as this should be done only once for each FPGA.

The Titan23 designs are packed by MultiPart [15] and are placed by VPR [12]. MultiPart is used for packing as it enables the routing of many Titan23 designs with the default channel width of 300 [15]. However, a few Titan23 designs have to be omitted: routing congestion problems arise for *bitcoin_miner* with the default channel width of 300, an error occurs for *LU_network* and *gaussianblur* in VPR 7.0.7, and *LU230* fails during the packing with MultiPart. Experiments are performed on a workstation with an Intel E5-2660v3@2.6GHz and 128 GB memory. All important results are given in Table I. They are split up in three categories: runtime, wire-length and critical path delay.

The routing schedule is based on the net router of VPR and experimental results of CRoute. The value of p_f is equal to 0.5 in the first two routing iterations. From then on it is multiplied by 2 in each iteration. The historical congestion penalty uses the same factor h_f of 1 in all iterations.

A. Runtime Improvement

The routing runtime of CRoute is 3.4x smaller when compared to VPR (Table I). This large gain in runtime is due to several reasons. Firstly, the total number of rerouted connections largely reduces (8.5x) if the connection-based routing principle is used. If a connection is congested in the net-based router, then all connections in that net are rerouted. Moreover, only rerouting uncongested connections also leads to a faster convergence. With CRoute, 6.2x less pathfinder routing iterations are required to achieve a congestion free solution. The detailed runtime of VPR and CRoute is shown in Fig. 4. The runtime of the first iteration is approximately equal for VPR and CRoute. The time to reroute connections, however, largely reduces. Furthermore, a static timing analysis is performed in each routing iteration. The total required time for static timing analysis thus largely reduces as less iterations are required. The remaining time is needed to initialize data and to calculate statistics in each iteration. Wang *et al.* [4] report a runtime gain of 3.2x when compared to VPR. They used, however, a set of small benchmark designs and only slightly gain in quality of results with a gain of 2.5% and 1.4% in critical path delay and wire-length, respectively.

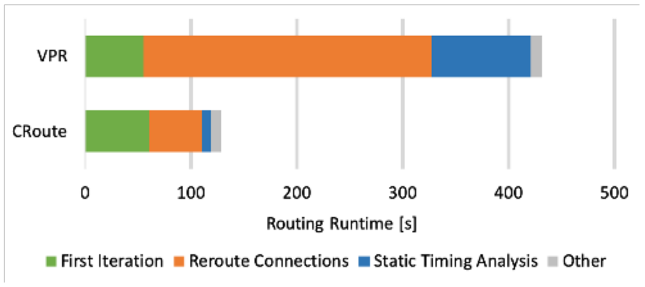


Fig. 4. Detailed geomean routing runtime of VPR and CRoute. The runtimes are the average over the Titan23 benchmark designs.

B. Total Wire-length Reduction

CRoute requires less routing resources to route all connections in a design. On average 11% less wire-length is required (Table I). This result is important as the interconnection network consumes a large fraction of the total FPGA area. With the new router, the designs can be implemented on FPGAs that contain fewer routing resources.

The detailed representation of the Stratix IV FPGA contains length 4 (L4) and length 16 (L16) wire segments. It is important to note that the wire-length gain is mainly due to the lower usage of the L16 wire segments. Routing the designs with CRoute leads to the same number of required L4 wire segments when compared to VPR. The number of used L16 wire segments, however, is reduced by 58%. The reason is that we increased the cost of using long wire segments, relative to their length. Therefore, the L16 wire segments are only used if they are really required, for example to reduce the delay of connections on the critical path. If the L4 and L16 wire segments would have the same cost, then the router is unable to know that using an L16 wire segment for a short connection leads to a large overhead from the unused part of the wire. Also note that the reduction in L16 wire segments does not lead to an increase in the number of L4 wire segments. In general the router is thus able to efficiently route the connections with a minimum amount of required wire segments.

C. Maximum Clock Frequency Increase

The maximum clock frequency can be analyzed by three performance metrics. Firstly, the critical path delay (CPD) is the delay of the longest path in the design. This can be a path inside a clock domain or a path between a netlist clock domain and the I/O clock domain. This metric for the maximum clock frequency is reported by most related work and is therefore given in Table I. The geomean reduction when compared to VPR is equal to 6%. Other metrics are the geomean critical path delay over all netlist clock domains and the fanout-weighted geomean critical path delay over all netlist clock domains. For some designs (*sparcT1_core*, *sparcT2_core* and *sparcT1_chip2*) an error prevents a fair comparison for the latter two metrics as some clock domains are left out in VPR 7.0.7. The clock domains are not analyzed by VPR because they are considered as constant generators. If these designs are left out, we obtain a geomean gain of 5% for both metrics.

TABLE I

Runtime and Quality Comparison of VPR and CRoute Expressed in Number of Routing Iterations (It), Number of Rerouted Connections (ReRout), Routing Runtime (RT), Total Wire-length (All), the Wire-length of the Length 4 (L4) and Length 16 (L16) Wire Segments, and the Critical Path Delay (CPD). The CPD Is the Maximum Delay Over All Clock Domains.

	VPR [12]							**CRoute**						
	It	**ReRout**	**RT**	**Wire-length [M]**			**CPD**	**It**	**ReRout**	**RT**	**Wire-length [M]**			**CPD**
Circuit	**#**	**[K]**	**[s]**	**All**	**L4**	**L16**	**[ns]**	**# (rel)**	**[K] (rel)**	**[s] (rel)**	**All (rel)**	**L4 (rel)**	**L16 (rel)**	**[ns] (rel)**
neuron	52	272	127	0.81	0.66	0.15	10.1	7 (0.13)	25 (0.09)	31 (0.25)	0.74 (0.91)	0.68 (1.02)	0.06 (0.42)	8.90 (0.88)
sparcT1_core	60	985	125	1.13	0.94	0.19	9.39	8 (0.13)	96 (0.10)	40 (0.32)	0.97 (0.86)	0.90 (0.96)	0.07 (0.38)	9.56 (1.02)
stereo_vision	60	235	89	0.72	0.57	0.15	8.30	8 (0.13)	30 (0.13)	29 (0.33)	0.66 (0.92)	0.60 (1.04)	0.06 (0.42)	8.47 (1.02)
cholesky_mc	55	487	233	1.11	0.92	0.19	8.35	8 (0.15)	43 (0.09)	52 (0.22)	0.98 (0.88)	0.90 (0.98)	0.08 (0.40)	7.74 (0.93)
des90	50	1319	359	2.52	2.08	0.45	12.1	11 (0.22)	175 (0.13)	165 (0.46)	2.33 (0.92)	2.07 (1.00)	0.26 (0.58)	10.3 (0.86)
SLAM_spheric	67	2272	261	2.03	1.72	0.31	80.4	10 (0.15)	216 (0.09)	73 (0.28)	1.76 (0.87)	1.65 (0.96)	0.11 (0.35)	79.9 (0.99)
segmentation	46	2212	271	2.33	1.92	0.41	788	12 (0.26)	375 (0.17)	120 (0.44)	2.10 (0.90)	1.96 (1.02)	0.14 (0.35)	798 (1.01)
bitonic_mesh	62	2411	694	5.02	4.03	0.99	15.6	10 (0.16)	326 (0.14)	235 (0.34)	4.55 (0.91)	4.07 (1.01)	0.48 (0.49)	12.8 (0.82)
dart	73	1277	297	2.30	1.92	0.38	16.5	8 (0.11)	145 (0.11)	83 (0.28)	2.06 (0.90)	1.91 (1.00)	0.15 (0.40)	16.0 (0.96)
openCV	71	1471	671	3.70	3.06	0.64	11.1	9 (0.13)	247 (0.17)	212 (0.32)	3.31 (0.90)	3.05 (1.00)	0.26 (0.40)	11.7 (1.05)
stap_qrd	41	1274	378	2.37	1.87	0.50	7.54	10 (0.24)	99 (0.08)	105 (0.28)	2.08 (0.88)	1.87 (1.00)	0.21 (0.41)	7.00 (0.93)
minres	73	1063	504	2.93	2.39	0.55	8.85	9 (0.12)	136 (0.13)	145 (0.29)	2.62 (0.89)	2.40 (1.01)	0.22 (0.40)	8.01 (0.91)
cholesky_bdti	58	1053	445	2.56	2.10	0.46	9.00	9 (0.16)	115 (0.11)	119 (0.27)	2.29 (0.90)	2.07 (0.99)	0.22 (0.47)	8.03 (0.89)
sparcT2_core	76	3224	474	4.14	3.40	0.74	11.9	10 (0.13)	426 (0.13)	166 (0.35)	3.59 (0.87)	3.33 (0.98)	0.27 (0.36)	11.5 (0.96)
denoise	65	4510	585	4.81	3.96	0.85	789	12 (0.18)	752 (0.17)	265 (0.45)	4.36 (0.91)	4.00 (1.01)	0.36 (0.43)	798 (1.01)
gsm_switch	55	2418	797	5.48	4.22	1.26	10.5	11 (0.20)	287 (0.12)	215 (0.27)	4.83 (0.88)	4.31 (1.02)	0.52 (0.41)	9.94 (0.95)
mes_noc	66	5398	2751	6.15	5.24	0.91	12.5	11 (0.17)	507 (0.09)	406 (0.15)	5.34 (0.87)	5.02 (0.96)	0.33 (0.36)	12.5 (1.00)
sparcT1_chip2	59	5120	1066	7.00	5.51	1.49	21.7	12 (0.20)	468 (0.09)	288 (0.27)	6.21 (0.89)	5.51 (1.00)	0.69 (0.47)	21.4 (0.99)
directrf	76	4297	3022	12.8	9.91	2.86	13.1	12 (0.16)	667 (0.16)	774 (0.26)	11.8 (0.92)	10.1 (1.02)	1.72 (0.60)	10.6 (0.80)
geomean	**60**	**1573**	**443**	**2.84**	**2.31**	**0.52**	**19.4**	**10 (0.16)**	**185 (0.12)**	**131 (0.30)**	**2.53 (0.89)**	**2.31 (1.00)**	**0.22 (0.42)**	**18.3 (0.94)**

VI. Conclusion

We propose a connection-based timing-driven router, called CRoute. It is originates from [2] and the ideas proposed in [3]. We propose algorithmic enhancements and introduce a timing-driven version. The router is analyzed by comparing it with the router in VPR 7.0.7 for the Titan23 benchmark designs. A large gain is achieved for both the quality of results and the required runtime. On average, 11% less wire-length is required with a 6% larger maximum clock frequency, while reducing the total routing runtime by 3.4x. Routing the large *directrf* design reduces from a runtime of 50 minutes to just 13 minutes, enabling faster design cycles. The gain in quality leads to a higher performance of the FPGA implementation with less required resources. The source code is publicly available in our FPGA CAD framework on GitHub [14].

In the future, we would like to analyze if the finer granularity of a connection-based router is beneficial for multithreaded acceleration techniques. The number of conflicts might be reduced as the source-sink connections have a smaller covering area than multi-sink nets. The workloads could also be better balanced as each net is split up in a set of smaller connections. The method to solve illegal routing trees at the end could also be improved by resolving these trees during the pathfinder routing iterations.

Acknowledgment

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government - department EWI.

References

[1] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[2] E. Vansteenkiste, K. Bruneel, and D. Stroobandt, “A connection-based router for FPGAs,” in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 326–329.

[3] E. Vansteenkiste, “New FPGA design tools and architectures,” Ph.D. dissertation, Ghent University, 2016.

[4] D. Wang, Z. Duan, C. Tian, B. Huang, and N. Zhang, “A runtime optimization approach for FPGA routing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1706–1710, 2018.

[5] S. Mukherjee and S. Roy, “Graph colouring based multi pin net detailed routing for FPGA using SAT,” in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE, 2013, pp. 308–312.

[6] M. Gort and J. H. Anderson, “Combined architecture/algorithm approach to fast FPGA routing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 6, pp. 1067–1079, 2013.

[7] M. Gort and J. H. Anderson, “Accelerating FPGA routing through parallelization and engineering enhancements special section on PAR-CAD 2010,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 61–74, 2012.

[8] M. Stojilović, “Parallel FPGA routing: Survey and challenges,” in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*. IEEE, 2017, pp. 1–8.

[9] Y. Moctar, M. Stojilović, and P. Brisk, “Deterministic parallel routing for FPGAs based on galois parallel execution model,” in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 21–214.

[10] C. H. Hoo and A. Kumar, “ParaDiMe: A distributed memory FPGA router based on speculative parallelism and path encoding,” in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 172–179.

[11] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in *International Workshop on Field Programmable Logic and Applications*. Springer, 1997, pp. 213–222.

[12] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, “VTR 7.0: Next generation architecture and CAD system for FPGAs,” *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[13] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, “Timing-driven Titan: Enabling large benchmarks and exploring the gap between academic and commercial CAD,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, p. 10, 2015.

[14] “FPGA CAD Framework: MultiPart, Liquid and CRoute” https://github.ugent.be/UGent-HES/FPGA-CAD-Framework, 2019.

[15] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, “How preserving circuit design hierarchy during FPGA packing leads to better performance,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 3, pp. 629–642, 2018.

[16] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, “Liquid: High quality scalable placement for large heterogeneous FPGAs,” in *Field Programmable Technology (ICFPT), 2017 International Conference on*. IEEE, 2017, pp. 17–24.

[17] L. McMurchie and C. Ebeling, “Pathfinder: a negotiation-based performance-driven router for FPGAs,” in *Field-Programmable Gate Arrays, 1995. FPGA’95. Proceedings of the Third International ACM Symposium on*. IEEE, 1995, pp. 111–117.

[18] J. S. Swartz, V. Betz, and J. Rose, “A fast routability-driven router for FPGAs,” in *Proc. of the 1998 ACM/SIGDA sixth int. symposium on Field programmable gate arrays*. ACM, 1998, pp. 140–149.