A Connection-based Router for FPGAs

Elias Vansteenkiste, Karel Bruneel and Dirk Stroobandt
Hardware and Embedded Systems Team, Computer Systems Lab
Department of Electronics and Information Systems, Ghent University, Belgium
Email: {Elias.Vansteenkiste, Karel.Bruneel, Dirk.Stroobandt}@UGent.be

***Abstract*—The FPGA's interconnection network not only requires the larger portion of the total silicon area in comparison to the logic available on the FPGA, it also contributes to the majority of the delay and power consumption. Therefore it is essential that routing algorithms are as efficient as possible. In this work the connection router is introduced. It is capable of partially ripping up and rerouting the routing trees of nets. To achieve this, the main congestion loop rips up and reroutes connections instead of nets, which allows the connection router to converge much faster to a solution. The connection router is compared with the VPR directed search router on the basis of VTR benchmarks on a modern commercial FPGA architecture. It is able to find routing solutions 4.4% faster for a relaxed routing problem and 84.3% faster for hard instances of the routing problem. And given the same amount of time as the VPR directed search, the connection router is able to find routing solutions with 5.8% less tracks per channel.**

I. Introduction

The routing infrastructure contributes to the majority of the FPGA's silicon area, delay and power consumption [1]. Therefore it is important that a router assigns the routing resources to nets in a highly efficient manner, with a minimum number of tracks per channel. In case the routing requirements of the design are the bottleneck, a more efficient router can significantly reduce the cost of the design, if the design could be realised with a smaller FPGA device. Efficient routing algorithms are particularly interesting for FPGA manufactures, because a router that can achieve routing solutions with lower track width, allows an FPGA vendor to provide less routing facilities, thereby saving precious silicon area.

This work aims to improve the state-of-the-art routing algorithms. In [2], the authors designed a reconfiguration-aware router for tuneable circuits. Due to the nature of the routing problem, the negotiated congestion mechanism of this router routes tuneable circuits on a per connection basis instead of on a per net basis. In this work the idea to route on a connection basis is elaborated for conventional circuits. We introduce an optimized router with a directed search instead of a breadth-first search, called the Directed Search Connection router. By ripping up and rerouting on a per connection basis, the routing trees of the nets are partially ripped up and rerouted. The effects of partially ripping up and rerouting the routing trees on the quality of the routing solutions is investigated in this work by comparing the Directed Search Connection router with the VPR Directed Search routing algorithm [3] on the basis of the benchmark circuits available in the state-of-the-art Verilog-To-Routing project [4] and we found that it is faster and has better results.

The remaining part of this article is organized as follows. In Section II, the background concepts, such as the routing problem, are explained. The VPR breadth-first router and the VPR directed search router are also described in this section. The connection router is presented in detail in Section III. In Section IV the experiments and results are described. This paper concludes with future work and a conclusion in Section V.

II. Background

A. Routing problem

If the routing architecture of the FPGA is represented as a routing-resource graph (RRG), the routing problem reduces to finding a subgraph of the RRG, called a routing tree, for each net in the input circuit. The routing trees should be disjoint in order to avoid short circuits. Each routing tree should contain at least the source node and sink nodes of its associated net and enough wire nodes so that source and sink nodes are connected. This problem definition can also be expressed in terms of connections, where a connection is an ordered pair containing a source and a sink. Each net can be seen as a set of connections and thus all interconnection between logic blocks can be defined as a set of connections. The routing problem then reduces to finding a simple path in the RRG for each connection in the circuit. Each path starts at the source node and ends at the sink node of its associated connection. These paths should only share nodes if the corresponding connections have the same source. Allowing other connections to share nodes would lead to short circuits.

B. VPR Breadth-First Search Routing Algorithm

The main structure of the VPR Breadth-First router is the negotiated congestion loop, adopted from Pathfinder [5]. In every routing iteration, the algorithm rips up and reroutes all the nets in the circuit. These iterations are repeated until no resources are shared illegally. This is achieved by gradually increasing the cost of illegally shared resources. The cost of a node is modulated with congestion penalties.

$$f(n) = b(n) \cdot h(n) \cdot p(n) \tag{1}$$

where $b(n)$ is the base cost, $p(n)$ is the present congestion penalty, and $h(n)$ is the historical congestion penalty. The present congestion penalty $p(n)$, is updated whenever a net is rerouted. The update is done as follows,

$$p(n) = \begin{cases} 1 & \text{if } cap(n) > occ(n) \\ 1 + p_f \cdot \\ (occ(n) - cap(n) + 1) & \text{otherwise} \end{cases} , \tag{2}$$

where $cap(n)$ represents the capacity of the node and $occ(n)$ is the occupancy of the node. The occupancy of a

```
routeNet(Net n):
    routingTree = {source}
    **for each** Sink s of n:
        path = dijkstra(routingTree, s)
        routingTree = routingTree$\cup$path
```

Fig. 1. Pseudo code of the maze router that the VPR routers use as net router heuristic.

node is the number of nets that are presently using the node. The factor p_f is used to increase the illegal sharing cost as the algorithm progresses. The historical congestion penalty is updated after every routing iteration i, except for the first iteration. The update is done as follows

$$h^i(n) = \begin{cases} 1 & \text{if } i = 1 \\ h^{(i-1)}(n) & \text{if } cap(n) \geq occ(n) \\ h^{(i-1)}(n) + & \\ h_f(occ(n) - cap(n)) & \text{otherwise} \end{cases} . \quad (3)$$

Again, the factor h_f is used to control the impact of the historical congestion penalty on the total resource cost. The way the congestion factors p_f and h_f change as the algorithm progresses is called the routing schedule.

To route one net, the embedded net router tries to find a minimum cost routing tree in the RRG. This problem is called the minimum steiner tree problem and is NP-complete.The VPR routers use a variant of the maze router heuristic [6]. The pseudo code for the VPR net router heuristic is shown in Fig. 1 The heuristic loops over all the sinks of the net and extends the already found routing tree with the shortest path from the routing tree to the sink under consideration. This heuristic is fast but does not always come up with an optimal solution, certainly in case of nets with a high fanout. It forces the router to extend paths that were previously added to the routing tree.

C. VPR Directed Search Routing Algorithm

A breadth-first search first expands the least-cost nodes, this leads to a circular wavefront. This guarantees a shortest path, but a lot of nodes are expanded in the wrong direction. For large architectures the search space can become very large. As the source node and the sink nodes are farther apart, the number of nodes that must be visited increases quadratically and hence the run-time and memory requirements increase. To remedy this issue the locations of the sink nodes, found in the placement of the circuit, are used to reduce the number of nodes visited during the search. The directed search routing algorithm, first introduced in [3] does not expand the least-cost node but the node that leads to a least-cost path. This results in a narrow wavefront that expands in the direction of the target pin to be connected. The path cost is calculated as follows,

$$f(n) = c_{prev} + b(n) \cdot h(n) \cdot p(n) + \ \alpha \cdot c_{exp}, \quad (4)$$

with α, the direction factor, which determines how aggressively the router explores towards the target sink, c_{prev} is the cost of the previous wire nodes on the path from the source to this wire node and c_{exp} is the expected cost of the path from this wire node to the sink node. The expected cost is calculated based on the Manhattan distance from this wire node to the sink node.

$$c_{exp} = n_{ortho} \cdot b_{ortho} + n_{same\ dir} \cdot b_{same\ dir} + b_{ipin} + b_{sink} \quad (5)$$

with n_{ortho}, the number of expected wire segments orthogonal to the wire segment under consideration, b_{ortho}, the base cost of an orthogonal wire segment, $n_{same\ dir}$, the number of expected wire segments in the same directions as the wire segment under consideration, $b_{same\ dir}$, the base cost of an wire segment in the same direction as the wire segment under consideration, b_{ipin} the base cost of and b_{sink} the base cost of a sink node.

III. Connection Router

A. Rip up and Reroute Connections

To partial rip up and reroute the routing tree of a net, the connection router rips up and reroutes connections in the main congestion loop instead of nets, as can be seen in the pseudocode in Fig. 3. The pseudocode can be compared with the pseudocode of the VPR router in Fig. 2. The connection router rips up each connection, recalculates the occupation of the nodes in the old routing path and if the occupation of a node decreases, then the present congestion penalty is updated according to Equation 2. In VPR the occupation ($occ(n)$) is the number of different nets that use the node. In the connection router, however, the occupancy is the number of different sources that drive the connections that use the node. The Dijkstra's algorithm is then used to find a path for each connection. If a new path is found, the occupation of each node in the new path is recalculated if the occupation of a node increases, then the present congestion penalty of that node is again updated according to Equation 2. At the end of each routing iteration the present congestion multiplier, historical congestion penalties and present congestion penalties are updated according to the Equations 3 and 2 respectively, in the same way as in VPR .

So the negotiated congestion mechanism in the connection router is applied on a smaller scale. First, the present congestion penalties are updated along with adding the nodes to the routing path of a connection in the connection router. In VPR, the present congestion penalties are only updated after the whole routing tree is found. And second, nodes in the path of one connection will be reconsidered when the Dijkstra's algorithm is searching a path for another connection of the same net. In VPR, once a node is added to a routing tree of a net, then the embedded net router is forced to make use of the node, if it is along the way to the sink of another connection of the same net.

To make ripping up and rerouting connections possible, a new node cost model had to be developed to take the nodes that will be shared between connections driven by the same source, into account.

B. A New Node Cost Model

The total wire-length is the sum of the base costs of the different nodes used in the routing solution. This total cost can be partitioned according to the nets in the circuit. In a legal routing solution, the nets are disjoint so no nodes can be

```
while (IllegalSharedResourcesExist()):
    for each Net n do:
        n.ripUpRouting()
        for each Sink s in n do:
            path = Dijkstra(RoutingTree,s)
            RoutingTree = RoutingTree ∪ path
        n.resources().updatePresentCongestionCost()
    allResources().updateHistoryCost()
    updatePresentCongestionMultiplier()
    allResources().updatePresentCongestionCost()
```

Fig. 2. Pseudo code of the VPR Router.

shared between nets. The cost of a net is the sum of the cost of each node used in net. Analogous to the partitioning of the total wire-length according to the nets, it is also possible to partition the cost according to the source-sink connections in the circuit. However, in a legal solution the connections do not have to be disjoint. Connections can legally share routing nodes if they are driven by the same source. So if the total wire-length is partitioned by connections, the cost of a connection is the sum of the base costs of the nodes that realise the connections, but if a node is shared between a number of connections, $share(n)$, then the cost of a node has to be shared by all the connections using it. Consequently, the cost of a node is divided by the number of connections that use the node and are driven by the same source as the connection under consideration.

C. Directed Search Connection-based Routing Algorithm

In Fig. 3 the pseudo code for the main loop of the negotiated congestion mechanism can be found. In one routing iteration all the connections are ripped up and rerouted. Dijkstra's algorithm searches the lowest cost path between the sink and the source of the connection. The cost of a node is calculated as follows

$$f(n) = c_{prev} + \frac{b(n) \cdot h(n) \cdot p(n)}{share(n)} + \alpha \cdot c_{exp}, \tag{6}$$

b(n), h(n) and p(n) are calculated in the same way as in VPR, see Equation 2 and 3, except for the occupation ($occ(n)$), which is the number of different sources that drive the connections that use the node. Another important difference is that the cost of the node is now divided by $share(n)$, the number of connections that legally share the node with the connection that Dijkstra's algorithm is currently searching a path for.

To calculate share(n), the router keeps a map for each net. The map contains the used nodes and for each node the number of connections that use the node. The values are loaded in, in the rip up method, before routing the connection and loaded out after a path is found by Dijkstra's algorithm. This implementation limits the extra data that has to be kept per node in the routing resource graph, and is therefore scalable with respect to the size of the FPGA.

To allow an A* search, the heuristic to calculate the expected cost has to be admissible. An admissible heuristic may never overestimate the cost of reaching the sink node. To keep the expected cost heuristic in the connection router admissible, the cost of the segments has to be divided by the number of connections that use the node under consideration

```
while (IllegalSharedResourcesExist()):
    for each Connection c do:
        c.ripUpRouting()
        c.routingPath = Dijkstra(c.source,c.sink)
        c.resources().updatePresentCongestionCost()
    allResources().updateHistoryCost()
    updatePresentCongestionMultiplier()
    allResources().updatePresentCongestionCost()
```

Fig. 3. Pseudo code of the Connection Router.

and are driven by the same source, because it is possible that all the wire segments up to the input pin of a connections are shared with other connections that have the same source.

$$c_{exp} = \frac{n_{ortho} \cdot b_{ortho}}{share(n)} + \frac{n_{same\ dir} \cdot b_{same\ dir}}{share(n)} + b_{ipin} + b_{sink}, \tag{7}$$

IV. Experiments and Results

A. FPGA Architecture

To test the connection router we used an architecture[1] based on Altera's Stratix IV and Xilinx' Virtex 7. The routing infrastructure of this architecture is built up by wires that span four logic blocks. Each logic block contains ten 6-input LUTs. The wires are interconnected via Wilton switch blocks.

B. Methodology

The Connection-based directed search router, hereafter mentioned as the connection router, is compared with the directed-search VPR router and not the timing-driven VPR router because the timing-driven mode is not yet implemented for the connection router. The comparison was made on the basis of the benchmarks included in the VTR project [4]. The placements were generated with the VPR wire-length-driven placer with default settings. For each benchmark, the following properties of the routing implementation were measured;

- The **Minimum Channel Width** (MCW) is the minimum number of tracks per channel that have to be present on the FPGA for the router to find a solution in a given maximum number of routing iterations.
- The **Total Wire-length** (TWL)
- The **Critical path delay** (CP) in nanoseconds
- The **Runtime** is the time the router needs to find a solution for an FPGA architecture with a given channel width. The routing algorithms were executed on a workstation with a 3.4 GHz quad-core Intel Core i7 processor and 32 GB memory.

C. Minimum Channel Width

We try to give both routers the same amount of time and measure what they achieve in that time. In a first attempt both routing algorithms were allowed 50 routing iterations to find a routing solution, this resulted in a 14.23% reduction in

[1]A description of this architecture is provided with the VTR project in `k6_N10_memDepth16384_memData64_40nm_timing.xml`.

TABLE I. THE MINIMUM CHANNEL WIDTHS FOR THE VTR BENCHMARKS.

Circuit	VPR		ConR	
Max routing its	100	50	50	40
bgm	150	154	132	132
blob_merge	96	100	88	88
boundtop	68	68	66	66
ch_intrinsics	48	48	46	48
diffeq1	52	56	54	56
diffeq2	48	50	48	50
LU8PEEng	130	136	116	118
LU32PEEng	200	204	172	178
mcml	142	142	128	128
mkDelayWorker32B	92	92	86	86
mkPktMerge	48	48	50	50
mkSMAdapter4B	64	70	64	64
or1200	84	84	76	80
raygentop	72	72	72	72
sha	62	62	60	60
stereovision0	70	70	66	66
stereovision1	112	114	108	108
stereovision2	166	170	154	154
stereovision3	30	30	30	30
Total	1756	1846	1616	1634
Relative difference		+5.1%	-8.0%	-5.8%
Runtime	3h11m	56m	5h13m	3h7m

total number of tracks per channel for the connection router in comparison with VPR, but the connection router needed roughly 4 hours more time to achieve those track widths than VPR, as can be seen in Table I. To allow a more fair comparison we increased the maximum number of routing iterations for the VPR router to 100. VPR still used 2 hours less than the connection router. Giving VPR more than 100 routing did not lead to significant decrease in channel width, so we limited the number of routing iterations of the connection router to 40. In that case the routers used about the same amount of time, 3h11m for VPR and 3h7m for the connection router and the connection router still found routing solutions for an FPGA with 5.8% less tracks.

D. Runtime

In this section both routers are given an FPGA with a fixed channel width and the runtime and quality of the routing solutions is compared. The channel width was fixed from MCW's that VPR found in 100 routing iterations up to 20% over provisioning on the MCW's that VPR found in 50 routing iterations. Table II contains the relative difference in quality of the routing solutions. The connection router is able to find routing solution on average 4.37% faster for a relaxed routing problem (MCW VPR for 50 routing iterations with an over provisioning of 20%) up to 84.28% faster for non-relaxed routing problems (MCW VPR for 100 routing iterations). The quality of connection router solutions is better with an average reduction in total wire length of 3.4% for relaxed routing problems up to 6.9% for harder routing problems. The critical path delay decreases slightly with 0.5% to 2.0% decrease.

The Connection router converges much faster to a solution than the VPR router. The number of routing iterations decreases for all the benchmark circuits, except *mkPktMerge*. The connection router needs on average, 44.28% less iterations to find a routing solution. This leads to a decrease in runtime for all the benchmark circuits except the smaller ones (less than 500 LUTs), such as *mkPktMerge*, *diffeq2* and *ch_intrinsics*

TABLE II. THE RELATIVE DIFFERENCE IN AVERAGE RUNTIME AND QUALITY OF THE ROUTING SOLUTIONS

Channel width	**Runtime**			**CP(%)**	**TWL(%)**
	VPR	**Conr**	**Δ (%)**		
$\text{CW}^{\text{50 routing its}}_{\text{min, VPR}}$+20%	6m24s	6m7s	-4.37	-0.5	-3.4
$\text{CW}^{\text{50 routing its}}_{\text{min, VPR}}$+15%	7m7s	5m57s	-16.23	-1.7	-5.6
$\text{CW}^{\text{50 routing its}}_{\text{min, VPR}}$+10%	8m3s	6m56s	-13.82	-1.5	-5.8
$\text{CW}^{\text{50 routing its}}_{\text{min, VPR}}$+5%	13m30s	9m20s	-30.89	-2.0	-5.9
$\text{CW}^{\text{50 routing its}}_{\text{min, VPR}}$	57m19s	17m14s	-69.92	-0.7	-6.6
$\text{CW}^{\text{100 routing its}}_{\text{min, VPR}}$	3h11m19s	30m5s	-84.28	-1.8	-6.9

for the MCW's found by VPR in 100 routing iterations. The cause of the increase in runtime for the smaller circuits, is the overhead to calculate and save the number of connections per net that use a node in the routing tree (see the division by $share(n)$ in the node cost in Equation 6 and 7). For the larger circuits the rapid convergence gains the upper hand, resulting in a net decrease in runtime.

V. Conclusion and Future Work

In this work a new routing algorithm is proposed, the Directed Search Connection router. The Connection router rips up and reroutes parts of the routing tree of a net, by ripping up and rerouting connections in the main congestion loop instead of nets. The Directed Search Connection router is more pareto efficient than the VPR directed search router. Given an FPGA with a certain channel width, it is able to find routing solutions 4.37% up to 84.28% faster, the quality of the routing solutions is slightly better and it is able to find routing solutions with 5.8% fewer tracks per channel, given the same amount of time.

In the future we will build a timing-driven router capable of partially ripping up and rerouting the routing trees of the nets, similar to the connection router presented in this work.

References

[1] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, Feb. 2008.

[2] E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "Maximizing the reuse of routing resources in a reconfiguration-aware connection router," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, 2012, pp. 322–329.

[3] J. S. Swartz, V. Betz, and J. Rose, "A fast routability-driven router for FPGAs," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, ser. FPGA '98. New York, NY, USA: ACM, 1998, pp. 140–149.

[4] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR project: architecture and CAD for FPGAs from verilog to routing," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 77–86.

[5] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, ser. FPGA '95. New York, NY, USA: ACM, 1995, pp. 111–117.

[6] C. Y. Lee, "An algorithm for path connections and its applications," *Electronic Computers, IRE Transactions on*, vol. EC-10, no. 3, pp. 346–365, Sept. 1961.