A Novel Connection-Based Multicasting Router for Programmable Photonic Circuits

Xiaoke Wang, *Graduate Student Member, IEEE*, Ferre Vanden Kerchove, Raveena Raikar, *Graduate Student Member, IEEE*, Mario Pickavet, *Senior Member, IEEE*, Wim Bogaerts, *Fellow, IEEE, Fellow, Optica*, and Dirk Stroobandt, *Member, IEEE*

***Abstract*—In the rapidly evolving domain of Photonic Integrated Circuits, reconfigurability is making strides through tunable waveguide elements, facilitating 'general-purpose' programmable waveguide grids. Routing in modern programmable photonic networks is challenging due to the numerous possibilities that exist for assigning photonic circuits in the grid. Especially the necessary scaling to devices with many more photonic elements calls for more advanced routing heuristics. We can leverage on the existing routers for electronic reconfigurable systems (FPGAs), such as the PathFinder. However, it is crucial to connect network elements while adhering to optical signals' physical restrictions. This complexity requires careful planning for smooth, error-free connections in the network infrastructure. This paper proposes a novel algorithm addressing routing challenges in programmable photonic circuits, specifically multicasting (single-source-multiple-sink) scenarios. Efficiently conserving the overall utilized routing resources stands as a crucial objective in programmable photonics routing. Our algorithm adeptly tackles multicasting routing problems with a particular focus on shortest pathlength multicasting routing. In contrast to the PathFinder, our algorithm demonstrates notable performance in optimization speed and the utilization of routing resources.**

***Index Terms*—Electronic photonic design automation, routing, programmable photonics.**

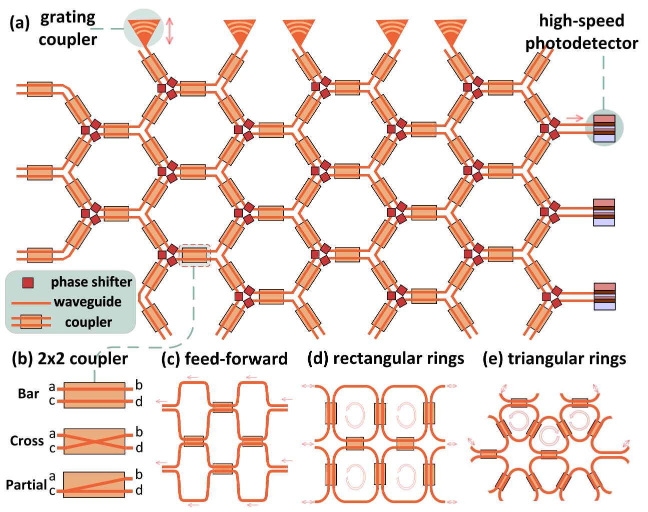

Fig. 1. Programmable photonic circuits [9]: (a) the circuit is composed of basic units and connects inputs, outputs, and functional blocks (modulators, detectors). (b) Internally it has couplers that can be put in different states. The topology can be feed-forward (c) or use feedback rings, these can be organized and tiled in different shapes, e.g. hexagons (a) or squares (d) and triangles (e).

I. Introduction

In recent decades, the field of silicon photonics and photonics integrated circuits (PICs) has experienced rapid advancements, paving the way for the realization of programmable photonic circuits through components such as tunable couplers, phase shifters and others... [1], [2]. These circuits will be more general-purpose than today's application-specific PICs (ASPICs). Such programmable photonic circuits are illustrated in Fig. 1. The 2×2 tunable coupler is the key component that enables reconfigurability, which is analogous to the switching blocks and connection blocks in field-programmable gate arrays (FPGAs). In contrast to switching blocks, these tunable couplers can also be used as photonic elements like resonators when using feedback rings. This component can enable three connection states: the bar state (no coupling), the cross state (full coupling), or the partial coupling state. Nowadays there are two groups of architectures commonly in use. The first one is the forward-only mesh [3] shown in Fig. 1(c), which facilitates the calculation of the transfer matrix for signals driven from one side (inputs) towards the opposite side (outputs). The alternative one is the recirculating mesh that tiles the same polygon unit and encompasses various topologies (hexagons [4] and squares [5] and triangular [6]) exhibited in Fig. 1(a)–(e). Koh [7] demonstrated that the hexagonal mesh is more efficient than the rectilinear and triangular meshes regarding wire length and congestion in VLSI routing architectures. This routing benefit also fits programmable photonic architectures [8]. In this article, our attention is devoted exclusively to the recirculating

Received 7 May 2024; revised 18 October 2024; accepted 19 November 2024. Date of publication 21 November 2024; date of current version 17 March 2025. The work of Ferre Vanden Kerchove was supported by Flemish Research Foundation (FWO-Vlaanderen) under Grant 11O0923N and Grant G020421N(GRAPHSPAY). *(Corresponding author: Xiaoke Wang.)*

Xiaoke Wang, Raveena Raikar, and Dirk Stroobandt are with the Hardware and Embedded Systems group, Department of Electronics and Information Systems, Ghent University, 9052 Ghent, Belgium (e-mail: xiaoke.wang@ugent.be; raveenaramanand.raikar@ugent.be; dirk.stroobandt@ugent.be).

Ferre Vanden Kerchove and Mario Pickavet are with the IDlab, Department of Information Technology, Ghent University-IMEC, 9052 Ghent, Belgium (e-mail: ferre.vandenkerchove@ugent.be; mario.pickavet@ugent.be).

Wim Bogaerts is with the Photonics Research Group, Department of Information Technology, Ghent University-IMEC, 9052 Ghent, Belgium (e-mail: wim.bogaerts@ugent.be).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JLT.2024.3504279.

Digital Object Identifier 10.1109/JLT.2024.3504279

hexagonal mesh. With the same physical restrictions in place, the routing algorithm can be adapted to other models with minimal adjustments.

With the advancement of PIC technologies, the integration of programmable photonic circuits will enable more light paths within the circuit. But compared to the mature ecosystem of FPGAs, the development of computer-aided design tools for programmable photonics is still in its infancy. Our work was inspired by state-of-the-art FPGA routers. We focus on adopting the connection-based router (CRoute) [10], [11] to adhere to the hexagonal grid programmable photonic circuits' physical restrictions and improve the scalability and efficiency.

The remainder of this paper is organized as follows: The problem definition and the constructed hexagonal grid graph are given in Section II. Section III introduces existing optical routers and some multicasting routing theories and concepts. In section III-C, we consider the well-known FPGA routers and borrow their innovations. This leads to our connection-based router for programmable photonics hexagonal grid which is detailed introduced in Section IV. Section V covers the measurements and results, starting with the generation of synthetic benchmarks and the tuning of parameters in order to perform a comparison of runtime and resource usage. We substantiate the superior time-saving and resource utilization advantages of our router through generated benchmarks. Finally, Section VI states the conclusion and potential for future research.

II. Problem Definition

We first clarify the routing requirements for programmable photonic circuits, including physical restrictions. Optical signal paths are distinct from the common electrical signals on the chip and exhibit unique properties. Additionally, the tunable coupler serves as a unique device within the routing process. Therefore, a customized model needs to be developed, and specific rules must be established to adhere to the physical restrictions.

A. Physical Restrictions

a) No Loop Condition: Avoiding loops (closed paths) in the circuit is essential to prevent undesirable interference effects, in line with the routing requirements.

b) Avoid Opposite Edges: Each photonic connection (or edge) has a certain directionality. That means light can only propagate in one direction along the waveguide during routing. Hence, an edge can't be used in the opposite direction to its defined direction, this restriction is known as avoiding using the "opposite" edge except for measuring the reflection.

c) Path Length Constraints: In certain applications like optical true time delay (OTTD) for beamforming networks, and some optical components like Mach-Zehnder interferometer (MZI), the pathlength plays a critical role in functionality. Each routed path must meet the precise desired pathlength requirements to ensure correct operation. However, this constraint is not yet addressed in this article.

d) Not in Phase: During the routing process of a net, variations in pathlength can lead to undesired interference, effectively performing as an unintended Mach-Zehnder interferometer (MZI). This interference results from the time delay difference between different paths, which causes the light waves to be out of phase.

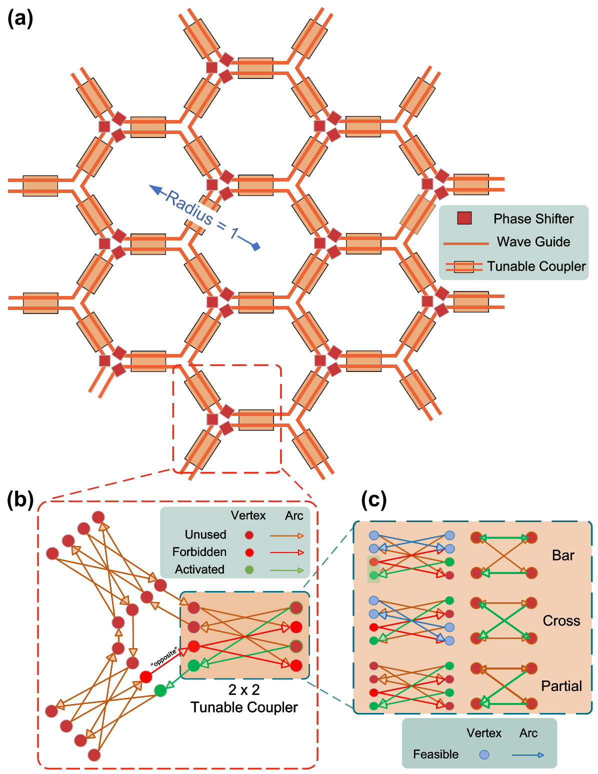

Fig. 2. Hexagonal circuit with radius $r = 1$ consists of components: characterized as a directed graph where (b) a unit is shown and (c) three states interpretation of the 2×2 tunable coupler

B. Grid

Relying on the brief restrictions listed above, the model and rules were interpreted in Fig. 2. Here, the radius r, indicating the scale of the hexagonal mesh, is set to 1 for the circuit shown in Fig. 2(a). Similar to some previous work [12], a directed routing resources graph $G = (V, E)$ was used to simplify the routing problem while also enabling a better degree of freedom to control the direction of the light path flow. The set of vertices V that represents the photonic nodes include all optical ports and **IO** pins. The set of edges E corresponds to the optical paths between and inside the tunable couplers, which are represented by all of the arcs (directed edges) shown in Fig. 2(b). As explained in Fig. 2(b) and (c) since it is a directed graph, one physical port in a coupler is represented by two nodes in the graph. We hypothesize that when the green edge between couplers is in use, the corresponding nodes inside the coupler, and the linked edges, inherit the same status. By the same principle, the red forbidden edges also infect connected edges in the coupler. The blue nodes and edges indicate that they are feasible and free for use in the current state. Consequently, all feasible linkages correspond to the three coupler states depicted on the right side.

III. Background

A connection represents a source-sink pair, while a net denotes a network with one source and multiple sinks (destinations) representing the basic interconnection format for multicasting. One multicasting net can be considered as consisting of multiple connections. Routing connections in programmable photonic circuits, aimed at linking optical components, is a common task. Some existing programmable photonics routers will be introduced in this section. Most of them aim to route only connections. This problem is effectively addressed by the negotiated congestion-based router $Aurora$ with a high success rate and speed [9].

However, multicasting (multi-sink net) will also play a significant role in programmable PIC designs, for example, the optical true time delay lines, wavelength-division multiplexing and optical splitting... Similarly, multi-sink nets routing is of paramount importance in modern very large-scale integration (VLSI) designs due to its direct impact on the circuit's performance, power efficiency, and overall chip area. Multi-sink routing builds on the well-known Steiner tree problem [13]. Popular FPGA routers already offer some innovations to deal with the Steiner tree problem [11], [14], [15], [16], [17]. These FPGA routers' approaches provide valuable insights for addressing the Steiner tree problem and optimizing wire-length-driven and routability-driven routing in programmable photonic circuits. In this article, we mainly focus on routing nets and optimizing the total pathlength of nets in programmable photonic circuits.

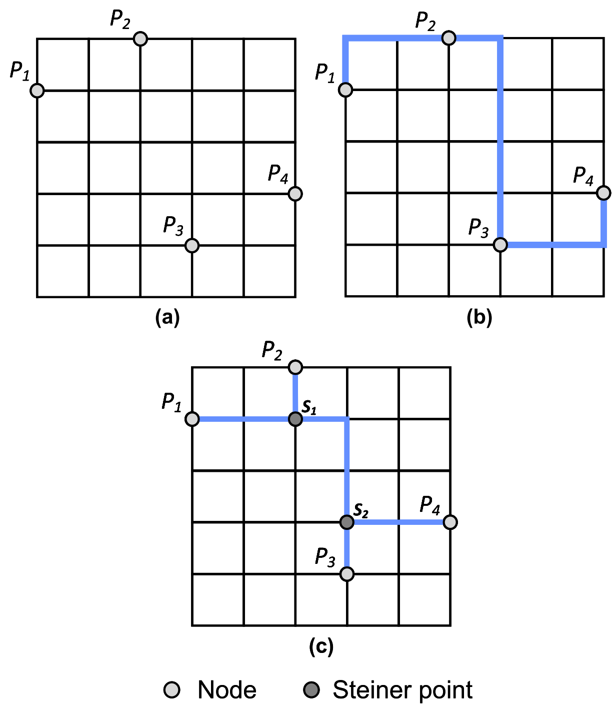

Fig. 3. A 4 pins net routed by trees [20]: (a) A 4 pins net. (b) A minimum spanning tree (MST). (c) A minimum rectilinear Steiner tree (MRST).

A. Existing Programmable Photonic Routers

a) Sequential Routing: This is a straightforward method firstly employed to address routing 2-pin nets (connections) challenges by routing one connection at a time along its shortest path, without using resources already allocated to previously routed connections [18]. Also employed to route multicasting nets [19] by considering one net into multiple connections. This process repeats for each connection until all are routed or no viable path remains for a connection, a situation that becomes increasingly likely with a large number of connections. Because the routing orders are fixed here, not only the success rate but also resource utilization is low with this approach.

b) Aurora: Aurora is a high-performance negotiated congestion based router only for solving connections [9], and the negotiated congestion mechanism is further elucidated in Section III-C. This router performed well in runtime, accuracy, and path length even in large meshes (e.g. with radius $r = 13$).

c) Integer Programming: An integer programming solver was used to solve the multi-sink routing problems in a $radius = 1$ mesh [12]. Though integer programming can always find the optimal solution, its time complexity is typically exponential in routing. In meshes with $radius \geq 3$, it will even take hours to return multi-sink results, indicating a lack of scalability.

B. Steiner Tree Problem

Breadth-first Search (BFS) [21], A^* [22] and $Dijkstra$'s [23] algorithms are commonly used to find the shortest path between the source and sink pins. But apart from 2 pins networks, also multi-pin networks commonly exist. The nets can be decomposed into sets of two-pin connections and these connections are then routed one by one. This decomposition is conventionally done by the minimum spanning tree (MST) algorithms, which is a minimum length tree of edges connecting pins. Fig. 3(b) demonstrates an example of a 4-pin rectilinear MST. For example, using Prim's algorithm, the connection between the source p_1 and its closest sink (p_2) is routed firstly and then arbitrarily from the source and searched sinks (p_1 and p_2) to the next closest sink (and so on).

An improved method, known as the minimum rectilinear Steiner tree (MRST), was employed to minimize the total wire length of the tree. Additional points, referred to as Steiner points, were introduced to the net. By finding a minimum spanning tree (MST) including these Steiner points, the result is an optimized MRST. The same 4-pin net example was depicted in Fig. 3(c), with S_1 and S_2 two Steiner points in the net. The MST total wire length from (b) is clearly decreased. When including a Steiner points S, there is a relationship between the MST and MRST of the net:

$$MRST(P) = MST(P \cup S)$$

Hanan's theorem [24] simplifies the minimum spanning tree construction by proving that MRST can be formed using Steiner points S exclusively from the grid points P of the Hanan grid, derived by extending vertical and horizontal lines through pins in the given nodes set. A MRST always exists in a Hanan grid and a minimum hexagonal Steiner tree also always exists in the hexagonal grid [7]. Despite this, MRST construction remains NP-hard, so there is also a huge challenge in real large-scale circuits to find the Steiner points and close-to-optimal practical solutions while still being cost-effective.

C. FPGA Net Routing Algorithms

1) PathFinder and Negotiated Congestion: PathFinder is a well-established routing algorithm that employs a congestion negotiation mechanism to resolve routing congestion and conflicts in FPGA circuits [25]. PathFinder includes two parts: a global router and a signal router. The signal router utilizes the Breadth-first search (BFS) algorithm as its shortest path strategy, effectively searching for the minimal spanning tree. Meanwhile, the global router adjusts the routing weights by updating the congestion penalty.

The global router finds the solution by iteratively ripping up the illegal nets and dynamically incorporating costs considering congestion within, then rerouting, until no more routing conflicts (congestions and violations) remain. This approach is known as the negotiated congestion mechanism that forms the core of the PathFinder. In this mechanism, each routing path is treated as an 'agent' that negotiates for routing resources. At the start of the process, each agent attempts to find a route from its source to its sink. This initial routing often leads to congestion, as multiple agents might choose the same routing resource. However, PathFinder deals with this congestion iteratively. After the initial routing, the algorithm identifies the congested routing resources (those used by more than one agent) and increases their cost. In the next iteration, the routes are recomputed with the updated costs, which discourages agents from using the congested resources. The cost of a given node n during the process is:

$$c(n) = (b(n) + h(n)) * p(n) \tag{1}$$

where b is the base cost of the node, h is the history cost related to the history of congestion on node n during previous iterations and p is known as the congestion penalty. The schedule is defined as follows:

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

$$h^i(n) = \begin{cases} 1, & i = 1 \\ h^{i-1}(n), & cap(n) \leq occ(n) \\ h^{i-1}(n) + h_f(occ(n) - cap(n)), & \text{else} \end{cases} \tag{3}$$

where the term occ represents the number of signals occupying a current node, while cap denotes the capacity as well as the maximum allowable usage limit. The term $occ - cap$ can be interpreted as the overuse in the current node. The history factor, h_f, determines the impact of past congestion (overcrowding) on the total cost of an edge. The change of congestion factors p_f and h_f throughout the algorithm's execution is what is called the routing schedule. Typically, in the FPGA routing, h_f is a fixed value ranging from 0.2 to 1, while p_f is initialized at 0.2 and then multiplied by 1.5 or 2 with each iteration to increase the penalty [14].

In Fig. 4, to find the MRST, PathFinder initiates a Breadth-first search from the net's source until it encounters the first sink, D_1,

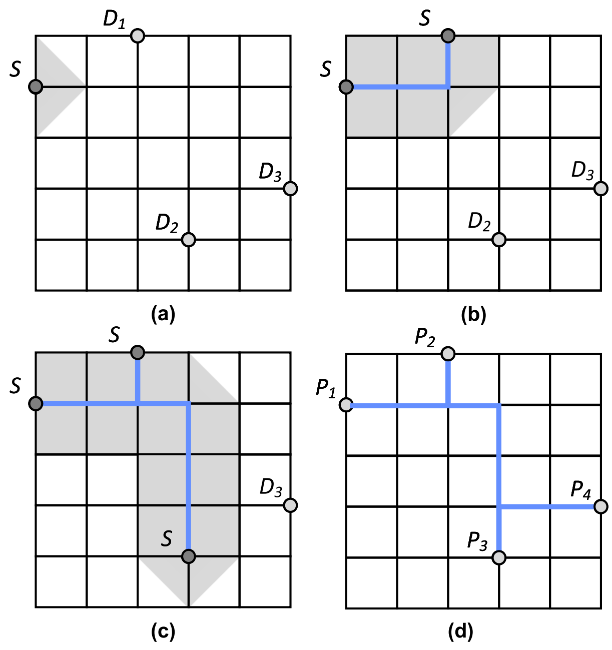

Fig. 4. PathFinder's routing procedure for multiple sinks depicted across four stages: (a) initiation of the search for the first sink, (b) identification of the first sink and path reinitialization with priority zero, (c) connection to the second sink with path extension and reinitialization, and (d) results upon locating all sinks.

thus acquiring a partial net path. Subsequently, every node along this partial path is assigned a zero priority, indicating that each becomes a starting point for the subsequent search. This results in a wavefront expansion as depicted in Fig. 4(b) and (c). The procedure continues until all sinks are located. Consequently, the path tree is designed to promote the reuse of paths from earlier partial paths.

2) VPR Router: The cost function of the router from the Versatile Place and Route (VPR) project [14], [26] slightly differs from PathFinder (1):

$$c(n) = b(n) \cdot h(n) \cdot p(n) \tag{4}$$

The addition formulation in PathFinder requires the base cost b and history cost h to be properly normalized to similar ranges of magnitude so that both can have an effect. in VPR, the multiplicative form (4) no longer necessitates normalization. The VPR router also employs a directed search, which will be introduced in the next section.

IV. Connection-Based Router for PICs

The Connection-based Router (CRoute) is a highly effective tool for FPGA routing. It simplifies routing (and makes it faster) by routing each connection of a net separately. However, to promote the reuse of already existing routes of the same net, it introduces a sharing factor. With this factor and the Aurora model as our base, we have tailored the connection-based router to address the challenges of net routing in programmable photonics. Our algorithm, presented in Algorithm 1, proposes a new length estimator for customized hexagonal meshes and a new rip-up and reroute mechanism (from lines 10 to 15) to enlarge the *share* effect. The connection-based router consists of two parts: a local router for source-to-sink connections (CRoute function in line 6) and a global router that follows a negotiated congestion mechanism (from lines 3 to 15) to solve congestions and violations. Algorithm uses the local router to route unrouted sinks (UNSINKS) sequentially, with a legality check (LEGAL) and best cost check (BEST) to update the best paths. Thus, the *not-in-phase* restriction introduced in the Section II can be solved by assigning the two *not-in-phase* connections to a "violation". As the connection router is based on a directed search aiming to find the shortest path, ensuring that loops will not appear in the path of a connection, also the recombination of two paths is set as a "violation" inside the global router as part of the negotiated congestion mechanism. The setting and updating of the cost function, incorporating the length estimator and direction factor, along with the rip-up and reroute mechanism, will be detailed and illustrated in the following subsections.

A. Weights

We use edge weights to calculate the path lengths rather than assigning weights to nodes in the routing resource graph (RRG). This difference facilitates a more straightforward representation of the inherent disparities between waveguides and couplers, encompassing factors like insertion loss (IL), power consumption (PC), and basic unit length (BUL). So in the graph the weight $b(e)$ of the edge e can be expressed as a summation of linear terms:

$$b(e) = c_1 \cdot \text{IL}(e) + c_2 \cdot \text{BUL}(e) + c_3 \cdot \text{PC}(e) \tag{5}$$

where $c_{1,2,3}$ are coefficients. In this article, we follow the same base weight for the router and define the basic unit length BUL $=$ 1 only for edges in between two different couplers. Edges inside the coupler equal to length 0. This approach is predicated on the understanding that an edge connecting two couplers will invariably link to an internal edge of a coupler on each side. Thus it simplifies the routing resources graph and decreases the computation complexity for routing algorithms.

B. Routing a Connection

The connection-based router simplifies the multi-sink problem (net) by dividing it into multiple source-sink pairs (connections) and routing these connections independently. Paths are allowed to share edges only if they originate from the same source. This approach also adheres to the restrictions of photonic nets. Relying on the negotiated congestion mechanism, only the remaining illegal connections need to be ripped up and rerouted rather than the entire net, consequently, facilitating notable reductions in runtime expenditure. The cost function of an edge is given by

$$c(e) = \frac{b(e) \cdot h(e) \cdot p(e)}{1 + share(e)} \tag{6}$$

where the numerator is the same as in VPR (4) and includes the base weight, history cost and penalty cost. For the base weight $b(e)$ we use the same definition in (5) as Aurora, the denominator $share$ is the number of paths originating from the same source that are legally sharing the current edge (node). Incorporating

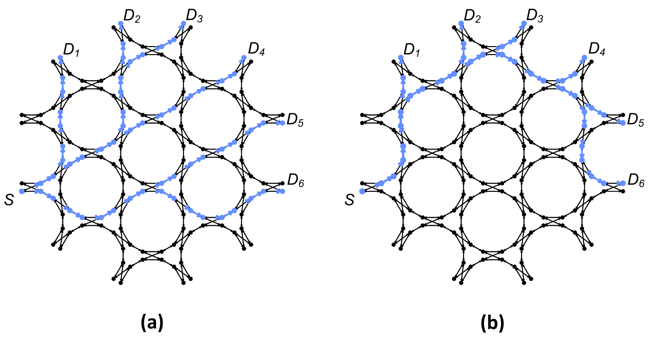

Fig. 5. A 7 pins (one source S and 6 sinks $D_{1\text{-}6}$) optical multicasting network (blue) on a radius 1 size programmable photonic hexagonal mesh: (a) A spanning tree multicasting net. (b) A minimum Steiner tree is routed by the connection-based router.

this term encourages the router to promote maximal path sharing among nets, the cost of an edge decreases as more connection paths legally share it. This strategy aims to approximate the results to those of the minimum Steiner tree more closely. And this cost function is embedded in the line 6 of the Algorithm 1. As presented in Fig. 5, S denotes the source of the net and all D_i are sinks. The result of the negotiated congestion implemented router is a spanning tree and the result of the connection-based router is a minimum hexagonal Steiner-tree. The comparison in terms of total wire length (TWL) is significant: the TWL for the spanning tree is 33, whereas it is 23 for the Steiner tree.

C. A^* Search and Distance Estimator

The conventional Breadth-first search (BFS) algorithm forms a circular wavefront around the source (diamond wavefront in a Manhattan grid) to ensure the exploration of all possible paths equally, guaranteeing the discovery of the shortest path to the sink. However, this approach may expend computational resources exploring directions that do not lead to the goal. By employing an estimator to calculate the distance between the current node and the destination, the A^* search algorithm can limit the search space.

In a Manhattan grid, a BFS algorithm will traverse $M^{bfs}_{\text{Man}} = 1 + 2n(n+1)$ nodes, where n represents the depth of the searched region. In the example shown in Fig. 6(a), with depth $n = 3$, there are 25 nodes in the light grey region. Conversely, for a hexagonal mesh, the search region will encompass $M^{bfs}_{\text{hex}} = 1 + \frac{3n}{2}(n+1)$ nodes, and 46 nodes shown in Fig. 6(b) with $n = 5$. When using the A^* search algorithm with a given depth n, the worst-case scenario in the Manhattan grid involves searching a square area until reaching the destination, including more nodes than a rectangular region. The number of nodes in this region can be expressed as $M^{a^*}_{\text{Man}} = (\frac{n}{2}+1)^2$. In contrast, the maximum search space in the hexagonal grid forms a diamond-shaped hexagon-distributed region, comprising $M^{a^*}_{\text{hex}} = (\frac{n+1}{2}+1)(\frac{n-1}{2}-1) + (\frac{n-1}{2}-3)$ nodes, as illustrated in Fig. 6(d). The search region's limit compared between BFS and A^* in two grids is given by:

$$\frac{A^*_{\text{Man}}}{\text{BFS}_{\text{Man}}} \leq \lim_{n\to\infty} \frac{M^{a^*}_{\text{Man}}}{M^{bfs}_{\text{Man}}} = \frac{1}{8}$$

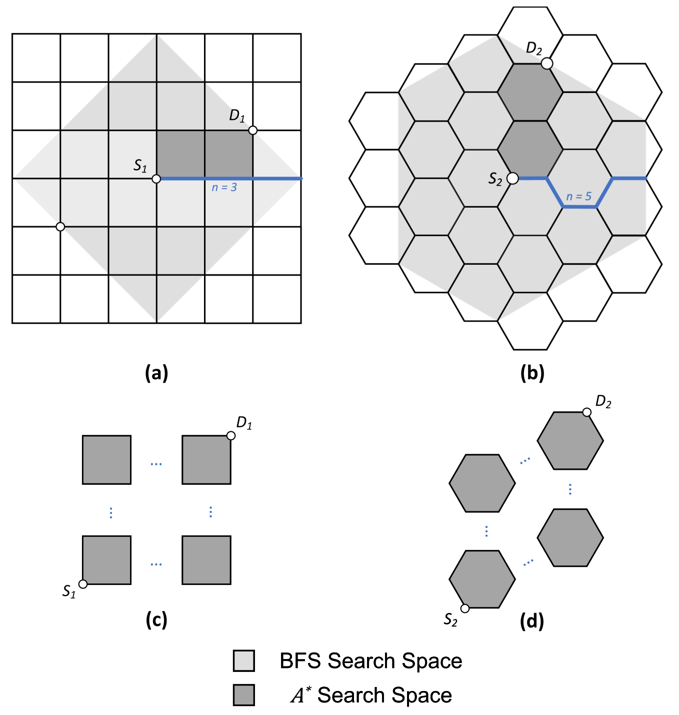

Fig. 6. (a), (b) A^* and BFS search region in the unweighted Manhattan grid and hexagonal grid between one connection, as shortest paths only exist inside the grey regions. (c), (d) the maximum search region for A^* search finding a connection in the Manhattan grid and hexagonal grid.

$$\frac{A^*_{\text{hex}}}{\text{BFS}_{\text{hex}}} \le \lim_{n\to\infty} \frac{M^{a^*}_{\text{hex}}}{M^{bfs}_{\text{hex}}} = \frac{1}{6}$$

As three primary routing directions exist in hexagonal mesh, saving at least $\frac{5}{6}$ search space is slightly worse than the Manhattan grid but still considerable to upgrade.

An accurate distance estimator is required for such approaches. Unlike the Manhattan grid's straightforward coordinate difference distance calculation, the hexagonal mesh we employ a distinct formula to compute the shortest path distance D between two nodes $\mathbf{n_1} = (x_1, y_1, z_1)$ and $\mathbf{n_2} = (x_2, y_2, z_2)$, given by:

$$D_{\mathbf{n_1}\text{-}\mathbf{n_2}} = \sum_{i\in\{x,y,z\}} \left[\frac{|i_1 - i_2|}{\delta}\right] \qquad (7)$$

where $\delta = \frac{3}{2}$ is the normalization factor, and $[\cdot]$ denotes the round function. The coordinates $\{x, y, z\}$ are defined in the coordinate system as presented in Fig. 7(a) and (b), where three axes were used to locate the nodes in the grid. The value $3/2$ is the projected length on the axes of a unit, colored green and yellow, as shown in Fig. 7(c) and (d). By dividing by $\delta = \frac{3}{2}$, the parallel-only segments are counted along the corresponding axis. All cases are illustrated in (c-d), with the counted segments labeled in c. The rounding function includes the edge parallel segment (the blue one in (c)) and ignores the edge non-parallel segment (the blue one in (d)). Finally, the summation across all three axes includes all segments, and thus the shortest distance is obtained. Equation (7) is also valid for both Manhattan and

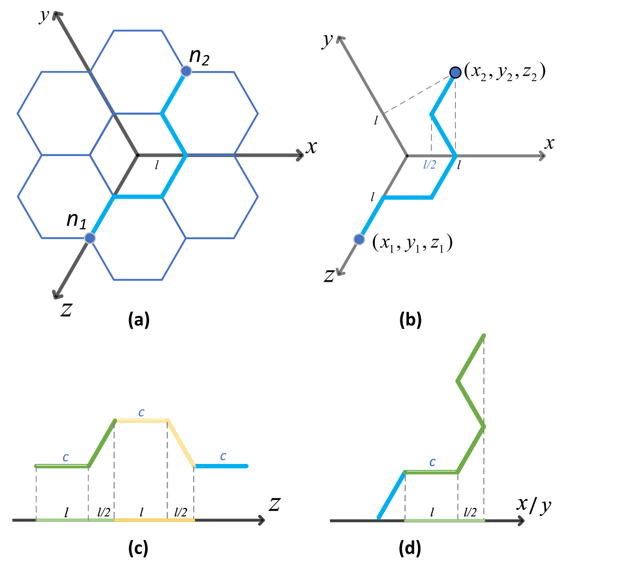

Fig. 7. The coordinate system of the hexagonal grid (a) and the shortest distance between two nodes in the grid (b). (c), (d) explanation for the projecting normalization method.

triangular grids, but with $\delta = 1$, as the two axes have fully matched projections.

The shortest distance between two nodes n_1, n_2 is plotted in blue in Fig. 7(a) with length 5, and the correct estimated distance is calculated by:

$$\begin{aligned} D_{n_1-n_2} &= \left[\frac{|-1-1|}{\frac{3}{2}}\right]_x + \left[\frac{|-1-1|}{\frac{3}{2}}\right]_y + \left[\frac{|2-(-2)|}{\frac{3}{2}}\right]_z \\ &= 1 + 1 + 3 = 5 \end{aligned}$$

In this example, only one parallel edge counts in x and y directions, and three edges count in the z direction. The A^* search requires the estimator to return the distance no more than the actual length, this estimator returns the minimum distance thereby meeting the requirement.

D. Directed Search

The concept of the directed search was introduced to accelerate the routing process [27], particularly for multicasting nets which necessitate restarting the search. Maintaining a wavefront property in such scenarios leads to excessive computational resource consumption.

The directed search algorithm is similar to the A^* algorithm, incorporating a predictive look-ahead feature to estimate the distance from the current node to the target sink. The difference is that directed search uses a greater than one direction factor multiplied by the length estimation item. Because of this direction factor, it will bring a lesser impact on nodes which are already closer to the destination (lower length estimation) compared with earlier found nodes with higher estimation. Thus, this approach gives up the wavefront property, behaving more like a Depth-first search. Consequently, the directed search algorithm consumes fewer computational resources; however, it sacrifices some total length if it is used in weighted graphs with a length-only estimator. This enables the router to quickly identify a short path, which is the primary reason directed search is employed in FPGA EDA tools like VPR for routing large-scale circuits. The cost is calculated by

$$f(e) = c_{prev} + c(e) + \alpha \cdot c_{exp}$$

for the weighted graph. Where the directional factor α refers to how aggressively the router will explore the current connection. The current edge cost $c(e)$ is calculated by using (6). In the CRoute version of the directed search, the distance D can be used to calculate the expected cost C_{exp} from the current node $\mathbf{c} = (x_c, y_c, z_c)$ to the destination $\mathbf{d} = (x_d, y_d, y_d)$. And the expected cost is given by

$$c_{exp} = \frac{D_{\mathbf{c}\text{-}\mathbf{d}}}{1 + share(e)}$$

where we also need to normalize the cost with the $share$ factor, just as in the overall cost function. The expected cost consistently represents the minimum distance obtained from the estimator (7).

E. Rip up and Reroute

A connection-based structure that focuses solely on ripping up and rerouting illegal connections instead of the entire nets, will bring huge benefits to computational speed, which is also used by other state-of-the-art routers [16]. However, this approach entails a trade-off, typically resulting in longer total wire lengths, as uncongested connections might attract other connections of the same net, forcing them too far away from the Steiner tree in an effort to avoid congested regions.

So, to amplify the effect of the sharing factor and optimize resources usage, a slightly different rip-up mechanism is implemented. During the initial λ iterations, the router will consistently perform a rip-up on all connections of nets that partly violate constraints. The routing order can impact the net length, as a connection routed first without encountering congestion or violations retains its path, compelling subsequent connections to adapt, even if this diverges from the optimal RMST solution. Implementing a mandatory rip-up reduces this impact. After λ iterations, the rip-up strategy reverts to CRoute's original "Lazy" approach. This step was shown in the Algorithm 1 from line 11 to 15, where ILSINKS refers to the set of illegal sinks.

V. Measurements and Results

A. Methodology

The measurements will cover two terms: the runtime and the total wire length between different routing algorithms to show their performance.

- Runtime: The routers were executed on a computer with a 4.7GHz AMD Ryzen 9 processor and 64GB memory.
- The Total Wire length (TWL): As with the routing resources utilization, this will not repeat counting the legal sharing of edges inside a multi-sink tree. A lower TWL for the same circuit means a better ability to save resources.

Algorithm 1: Connection-Based Router for PICs.

```
Require: $nets$ to route
Ensure: $best$ routing found
 1: function CRoutenets, λ
 2:   $best \leftarrow \emptyset$, $cur \leftarrow \emptyset$
 3:   for $iter \in 1 \ldots$ max_iters do
 4:     for all $net \in nets$ do
 5:       for all $sink \in$ UNSINKS$(net, cur[net])$ do
 6:         $cur[net] \leftarrow$ CRoute$(net, cur[net], sink)$
 7:         if LEGAL$(cur)$ then
 8:           $best \leftarrow$ BEST$(best, cur)$
 9:           UPDATE(ILSINKS)
10:     for all $net \in nets$ do
11:       for all $sink \in$ ILSINKS$(net, cur[net])$ do
12:         if $iter \leq \lambda$ then
13:           RIP_UP$(net, cur[net])$
14:         else
15:           LAZY_RIP_UP$(net, cur[net], sink)$
16:   return best
```

B. Synthetic Benchmarks

In FPGA domain, there are multiple well-established benchmark suits based on realistic functional circuits. Real circuits-based benchmarks are still a blank area for evaluating routing algorithms for programmable photonics. To address this, we generate custom benchmarks of various sizes. These benchmarks include multiple $1 \times N_i$ multicasting networks, with sources and sinks randomly located at the edge ports of a hexagonal mesh. As the scaling behaviour requires benchmarks to maintain the complexity with only varying sizes. The size of the hexagonal mesh is defined by its radius r, where $r = 0$ corresponds to a single hexagon surrounded by outer **IO** ports. For instance, in Fig. 5, the size of the mesh is $r = 1$. Consequently, for a circuit with radius r, there will be $6r$ hexagonal cells in the outer layer, and a total of $24 + 12(r - 1) = 12r + 12$. To accurately measure the scaling behaviour, it is important to maintain consistent complexity and density across varying sizes. The parameters that need to be determined are the number of networks and the number of sinks, N, for each multicasting network. Based on this, we generate $\lfloor \frac{12r+12}{N_{\max}+1} \rfloor$ multicasting networks for each size r, where $N_{\max}$ represents a set parameter, maximum number of sinks for a single network among all $1 \times N$ networks. Then, the maximum number of ports allocated to a single network is $N_{\max} + 1$. And $\lfloor \cdot \rfloor$ refers to floor function. Each network is then assigned a random number of sinks, ranging from 1 to $N_{\max}$.

C. Parameters Tuning

1) Routing Schedule: The capacity cap of each waveguide is set to 1, as we only consider routing $1 \times N$ multicasting networks in the benchmarks of this paper. The penalty factor, denoted as p_f, the initial value doesn't influence the results a lot based on measurements. So we take the initialization value 0.5 at the first iteration and increase it by multiplying it with a multiplicative factor in each subsequent iteration which can speed up the routing progress. This factor we tested from 1 to

TABLE I
COMPARISON OF THE VPR ROUTER, PATHFINDER IMPLEMENTATIONS, AND CONNECTION-BASED ROUTER FOR PICS BETWEEN PROBLEM SET SIZES FROM RADIUS $r = 1$ TO 8

Problem	Mesh	**VPR**		**PathFinder**		**Connection-based**	
Set	Size (radius)	CPU Time (s)	TWL	CPU Time (s)	TWL	CPU Time (s)	TWL
P1	1	0.029	29.8	0.013	28.8	0.020	30.8
P2	2	0.087	68.2	0.088	62.0	0.076	63.8
P3	3	0.377	113.4	0.247	102.8	0.158	107.4
P4	4	0.466	179.8	0.516	155.8	0.419	169.0
P5	5	2.251	296.8	1.334	260.8	1.013	281.8
P6	6	2.430	315.8	3.937	277.7	1.449	304.0
P7	7	4.598	427.6	22.119	361.4	1.723	397.8
P8	8	8.203	542.2	81.352	458.8	2.962	506.4

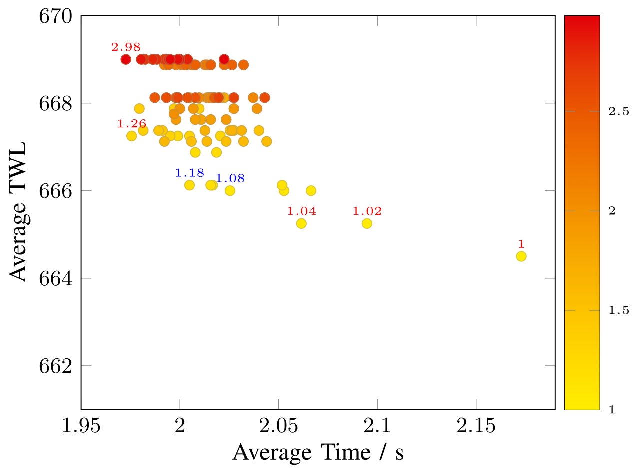

Fig. 8. Runtime-TWL trade-off for the penalty factor's multiplier: 1 to 3 based on large size synthetic benchmarks ($r \geq 10$).

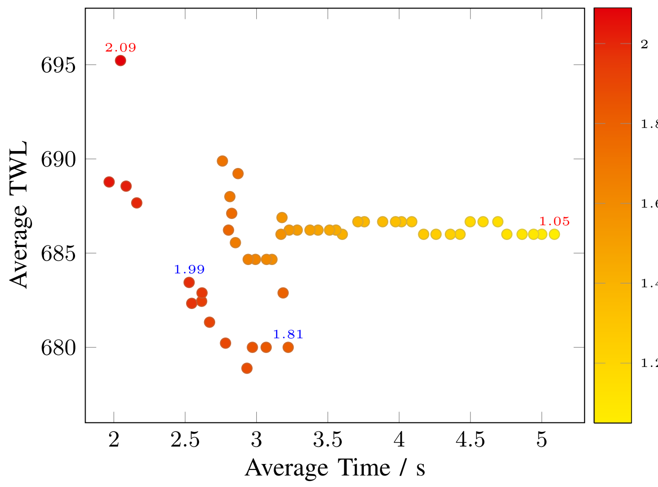

Fig. 9. Runtime-TWL tradeoff for the direction factor α : 1.05 to 2.09 based on large size synthetic benchmarks ($r \geq 10$).

2.98 is shown in Fig. 8 based on a large-scale benchmark, there is an obvious Pareto front showing the trade-off between TWL and runtime. We pick the values between blue labeled nodes which are from 1.08 to 1.18. For much simpler benchmark circuits, a larger factor can also be picked to speed up the routing.

Programmable photonic circuits are limited to a single channel width for routing (width in two if considering two waveguides as a channel). Furthermore, the hexagonal mesh topology offers fewer shortest path options between two nodes, making it challenging to quickly avoid congestion with acceptable extra length. Therefore, the rip-up and reroute process requires "patience". This necessitates the use of those relatively smaller multiplier values compared to the ones used in the original FPGA schedule.

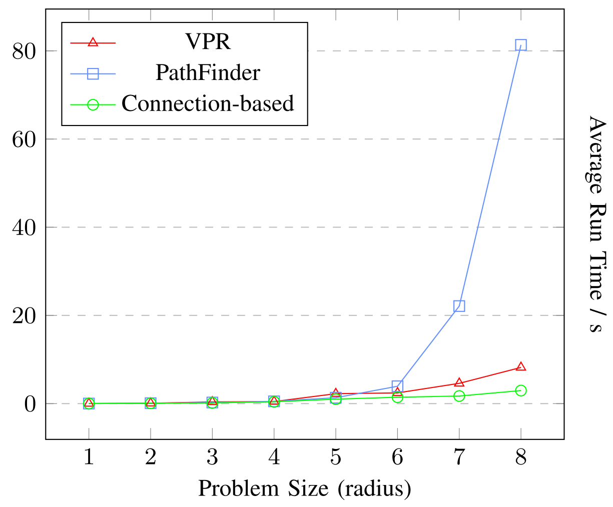

Fig. 10. Average runtime comparison between VPR, PathFinder implementations, and Connection-based on scaling synthetic benchmarks (radius: 1 to 8 hexagonal grid).

2) Direction Factor: For the directed search, a group of well-performed direction factors $\alpha = 1.8$ to 2 are obtained from the test results shown in Fig. 9. Among the scattered results, there is also an approximate Pareto front showing the trade-off between TWL and runtime. The corresponding α values are notably high. A large direction factor will prefer to keep the earlier searched path waiting for the "slowly" increasing penalty value for congestions and violations. This is also matched with the analysis of the routing schedule discussed earlier.

D. Results

Utilizing the previously mentioned synthetic benchmark generator, we generated multicasting benchmark suites spanning from a small mesh to a large one, with sizes ranging from radius $r = 1$ to 8. Each size has 5 problems, and each problem is run 10 times, with the average taken. We implemented two comparative routers for PICs, following the PathFinder and VPR router's cost function and iterative net routing mechanism

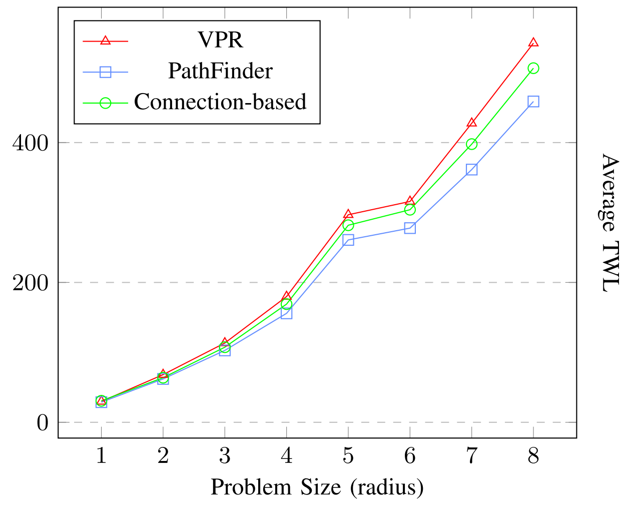

Fig. 11. Average TWL comparison between VPR, PathFinder, and Connection-based on scaling synthetic benchmarks (radius: 1 to 8 hexagonal grid).

introduced in Section III, respectively, in order to compare with our connection-based router. The results obtained on runtime and TWL are shown in Table I.

1) Runtime Comparison: The Router's runtime scaling results are plotted in Fig. 10. Due to the property, which will restart Breadth-first wavefront searching from its temporary paths after meeting one sink, constraining its scaling capabilities. The other two utilize directed search, with the connection-based router achieving significantly better performance thanks to its connection-based rip-up and reroute strategy. The connection-based router saved around 50% runtime in $r \geq 5$ benchmarks which performed a good scalability.

2) Total Wire Length Comparison: The scaling results for total wire length are presented in Fig. 11, where PathFinder achieved the best performance. Additionally, the connection-based router's utilization of its *share* strategy to encourage path reuse resulted in approximately 5% lower TWL compared to VPR.

VI. Conclusion

We have presented a high-performance routing solution capable of addressing large-scale, multicasting routing challenges within a programmable photonic hexagonal grid. This connection-based router innovated from the CRoute's connection routing structure, the sharing mechanism, directed search, and customized routing schedule modifications are added aiming at the photonic hexagonal grid. Utilizing synthetic benchmarks in radius from 1 to 8, compared with the VPR router and the Pathfinder implementations on the hexagonal grid, our router has showcased the practicality. Particularly noteworthy is its efficacy in handling large-scale circuits, delivering swift processing without compromising wire lengths and optimizing the utilization of routing resources.

This paper only covers $1 \times N$ multicasting networks as benchmarks. However, the router we proposed is more generic. By tuning the parameter cap value in (3) and (2), and adjusting the criteria for a violation (line 7 in Algorithm 1), the algorithm can route $N \times M$ multicasting networks with the same advantages.

For future work, considering that current results and measurements are based on synthetic benchmarks with random connections, developing a realistic benchmark suite would significantly enhance router development and optimization efforts. And for even large-scale mesh, a bounding box is necessary to increase time efficiency. Additionally, the precise wire length of each connection is crucial for effective phase control. Achieving length matching for routers will substantially impact the performance of on-chip resonators and interferometers.

References

[1] W. Bogaerts et al., "Programmable photonic circuits," *Nature*, vol. 586, no. 7828, pp. 207–216, Oct. 2020. [Online]. Available: https://www.nature.com/articles/s41586-020-2764-0

[2] D. Pérez-López et al., "General-purpose programmable photonic processor for advanced radiofrequency applications," *Nature Commun.*, vol. 15, no. 1, 2024, Art. no. 1563.

[3] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, "Optimal design for universal multiport interferometers," *Optica*, vol. 3, no. 12, pp. 1460–1465, Dec. 2016. [Online]. Available: https://opg.optica.org/optica/abstract.cfm?URI=optica-3-12-1460

[4] D. Pérez et al., "Multipurpose silicon photonics signal processor core," *Nature Commun.*, vol. 8, no. 1, Sep. 2017, Art. no. 636. [Online]. Available: https://www.nature.com/articles/s41467-017-00714-1

[5] L. Zhuang, C. G. H. Roeloffzen, M. Hoekman, K.-J. Boller, and A. J. Lowery, "Programmable photonic signal processor chip for radiofrequency applications," *Optica*, vol. 2, no. 10, pp. 854–859, Oct. 2015. [Online]. Available: https://opg.optica.org/optica/abstract.cfm?URI=optica-2-10-854

[6] D. Pérez, I. Gasulla, J. Capmany, and R. A. Soref, "Reconfigurable lattice mesh designs for programmable photonic processors," *Opt. Exp.*, vol. 24, no. 11, pp. 12093–12106, May 2016. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-24-11-12093

[7] C.-K. Koh and P. H. Madden, "Manhattan or non-Manhattan? A study of alternative VLSI routing architectures," in *Proc. 10th Great Lakes Symp. VLSI*, 2000, pp. 47–52.

[8] F. V. Kerchove, D. Colle, W. Tavernier, W. Bogaerts, and M. Pickavet, "Routing impact of architecture and damage in programmable photonic meshes," *Photon. Res.*, vol. 12, no. 9, pp. 1999–2011, 2024.

[9] F. V. Kerchove, X. Chen, D. Colle, W. Tavernier, W. Bogaerts, and M. Pickavet, "An automated router with optical resource adaptation," *J. Lightw. Technol.*, vol. 41, no. 18, pp. 5807–5819, Sep. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10122952/

[10] E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "A connection-based router for FPGAs," in *Proc. 2013 Int. Conf. Field-Programmable Technol.*, Dec. 2013, pp. 326–329. [Online]. Available: https://ieeexplore.ieee.org/document/6718378/

[11] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, "CRoute: A fast high-quality timing-driven connection-based FPGA router," in *Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, Apr. 2019, pp. 53–60. [Online]. Available: https://ieeexplore.ieee.org/document/8735564/

[12] X. Chen, P. Stroobant, M. Pickavet, and W. Bogaerts, "Graph representations for programmable photonic circuits," *J. Lightw. Technol.*, vol. 38, no. 15, pp. 4009–4018, Aug. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9056549/

[13] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, no. 1, pp. 55–89, 1992.

[14] V. Betz, J. Rose, and A. Marquardt, "Routing tools and routing architecture generation," in *Architecture and CAD for Deep-Submicron FPGAS* (Engineering and Computer Science Series), V. Betz, J. Rose, and A. Marquardt, Eds. Boston, MA, USA: Springer, 1999, pp. 63–103. [Online]. Available: https://doi.org/10.1007/978-1-4615-5145-4_4

[15] Y. Zhou, P. Maidee, C. Lavin, A. Kaviani, and D. Stroobandt, "RWRoute: An open-source timing-driven router for commercial FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 1, pp. 1–27, Mar. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3491236

[16] K. E. Murray, S. Zhong, and V. Betz, "AIR: A fast but lazy timing-driven FPGA router," in *Proc. 25th Asia South Pacific Des. Automat. Conf.*, Jan. 2020, pp. 338–344. [Online]. Available: https://ieeexplore.ieee.org/document/9045175/

[17] K. E. Murray et al., "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, pp. 1–55, 2020.

[18] A. López, D. Pérez, P. DasMahapatra, and J. Capmany, "Auto-routing algorithm for field-programmable photonic gate arrays," *Opt. Exp.*, vol. 28, no. 1, pp. 737–752, Jan. 2020. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?uri=oe-28-1-737

[19] Z. Xie, D. Sánchez-Jácome, L. Torrijos-Morán, and D. Pérez-López, "Software-defined optical networking applications enabled by programmable integrated photonics," *J. Opt. Commun. Netw.*, vol. 16, no. 8, pp. D10–D17, 2024.

[20] H.-Y. Chen and Y.-W. Chang, "CHAPTER 12 - Global and detailed routing," in *Electronic Design Automation*, L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, Eds. Boston, MA, USA: Morgan Kaufmann, 2009, pp. 687–749. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123743640500199

[21] E. F. Moore, "The shortest path through a maze," in *Proc. Int. Symp. Theory Switching*, 1959, pp. 285–292.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[24] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, 1966, doi: 10.1137/0114025.

[25] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. 3rd Int. ACM Symp. Field-Programmable Gate Arrays*, 1995, pp. 111–117. [Online]. Available: https://ieeexplore.ieee.org/document/1377269/

[26] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. Int. Workshop Field Programmable Log. Appl.*, 1997, pp. 213–222. [Online]. Available: http://link.springer.com/10.1007/3-540-63465-7_226

[27] J. S. Swartz, V. Betz, and J. Rose, "A fast routability-driven router for FPGAs," in *Proc. ACM/SIGDA 6th Int. Symp. Field Programmable Gate Arrays*, 1998, pp. 140–149. [Online]. Available: http://portal.acm.org/citation.cfm?doid=275107.275134

Xiaoke Wang (Graduate Student Member, IEEE) received the M.Sc. degree in electrical engineering from Ghent University, Ghent, Belgium, in 2023. He is currently working toward the Ph.D. degree with Hardware and Embedded Systems Group, Computer System Lab, Ghent University. His research interests include FPGA architecture and EDA tools.

Ferre Vanden Kerchove received the M.Sc. degree in mathematics from Ghent University, Ghent, Belgium, in 2021. He is currently working toward the Ph.D. degree in computer science with IDLab, Ghent University - IMEC, Leuven, Belgium. His research interests include algorithms, graph theory, logic, and computability.

Raveena Raikar (Graduate Student Member, IEEE) received the Master of Engineering degree in embedded systems from BITS Pilani, Pilani, Rajasthan, India, in 2018. She is currently working toward the Ph.D. degree with the Hardware and Embedded Systems Group, Computer System Lab, Ghent University. Her research focuses on optimising CAD tool flows for multi-die FPGA architectures.

Mario Pickavet (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from Ghent University, Ghent, Belgium, in 1996 and 1999, respectively. Since 2000, he has been a Professor with Ghent University, where he is currently teaching courses on discrete mathematics and network modeling. He is also Co-Leading the research cluster on network modeling, design, and evaluation (NetMoDeL). He has authored or coauthored about 500 international publications, both in journals and proceedings of conferences. He is coauthor of the book *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. His research interests mainly include fixed internet architectures and optical networks, green ICT, and the design of network algorithms. He is also involved in several European and National Projects. He was the recipient of a Bronze Medal at the International Mathematical Olympiad (Sweden, 1991).

Wim Bogaerts (Fellow, IEEE) received the Ph.D. degree in the modeling, design, and fabrication of silicon nanophotonic components from Ghent University, Ghent, Belgium, in 2004. During this work, he started the first silicon photonics process on IMEC's 200 mm pilot line, which formed the basis of the multi-project-wafer service ePIXfab. In 2014, he co-founded the spin-off company Luceda Photonics to further develop unique software solutions for silicon photonics design, using the IPKISS design framework. Since 2016, he has been again a full-time Professor with Ghent University, looking into novel topologies for large-scale programmable photonic circuits, supported by a consolidator grant from the European Research Council (ERC). His research interests include the challenges for large-scale silicon photonics: Design methodologies and controllability of complex photonic circuits, telecommunications, information technology, and applied sciences.

Dirk Stroobandt (Member, IEEE) received the Ph.D. degree in electrotechnical engineering from Ghent University, Ghent, Belgium, in 1998. He was a Visiting Researcher with the University of California at Irvine, Irvine, CA, USA, in 1997, and the University of California at Los Angeles, Los Angeles, CA, USA, from 1999 to 2000. He is currently a Full Professor with Computer Systems Laboratory, Department of Electronics and Information Systems, Ghent University, where he also leads the Research Group Hardware and Embedded Systems. His research interests include semiautomatic hardware design, runtime field-programmable gate array reconfiguration, and reconfigurable multiprocessor networks.