An Optimized GIB Routing Architecture with Bent Wires for FPGA

KAICHUANG SHI, XUEGONG ZHOU, HAO ZHOU, and LINGLI WANG*, State Key Laboratory of ASIC and System, Fudan University, China

Field Programmable Gate Arrays (FPGAs) are widely used because of the superiority in flexibility and lower non-recurring engineering cost. How to optimize the routing architecture is a key problem for FPGA architects because it has a large impact on the FPGA area, delay and routability. In academia, the routing architecture is mainly based on the connection blocks (CBs) and switch blocks (SBs), while most researches have focused on the SB architectures, such as Wilton, Universal and Disjoint SB patterns. In this paper, we propose a novel unidirectional routing architecture, general interconnection block (GIB) to improve the FPGA performance. With GIB architecture, logic block (LB) pins can directly connect with the adjacent GIBs without programmable switches. Inside a GIB, LB pins can connect to the routing channel tracks on the four sides of a GIB. In particular, the logic pins from different neighboring LBs that connect to the same GIB can connect with each other with only one programmable switch. Besides, we enhance VTR to support GIB with bent wires and develop a searching framework base on simulated annealing algorithm to search for a near-optimal distribution of wire types. We evaluate the GIB architecture on VTR 8 with the provided benchmark circuits. The experimental results show that the GIB architecture with length-4 wires can achieve 9.5% improvement on the critical path delay and 11.1% improvement on the area-delay product compared to the VTR CB-SB architecture with length-4 wires. After exploring mixed wire types, the optimized GIB architecture can further improve the delay by 16.4% and area-delay product by 17.1% compared to the CB-SB architecture with length-4 wires.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; **Programmable interconnect**; *Physical design (EDA).*

Additional Key Words and Phrases: routing architecture, connection block, switch block

1 INTRODUCTION

FPGAs are widely used due to low non-recurring engineering cost, fast time-to-market and their superiority in flexibility and reconfigurability. However, the flexibility of FPGAs comes at the cost of area, delay and power consumption when designs are implemented in FPGAs compared to application-specific integrated circuits (ASICs) [18]. The flexibility heavily relies on the programmable routing architectures, which consist of wire segments and programmable switches. The routing architecture has a great impact on the area, delay and routability [15][16]. Experimental results [1] show that the routing area accounts for about 52% of the total area and the global routing delay accounts for about 55% of the critical path delay. So how to optimize the interconnect architecture is a top question for FPGA architects. Besides, interconnect RC delay has increased rapidly with the process scaling. Despite that the wire distances shrink and transistor delay decreases, total delay actually increases because wire cross-sectional area shrinks quadratically with more advanced process nodes [29].

*Corresponding author.

[1]The area and delay parameters are extracted from COFFE 2 [50] at 22 nm technology node, the LB local interconnect is half-populated and the wire segments are all length-4. The benchmarks we use are VTR benchmarks [30] and the results are reported by VTR.

Authors' address: Kaichuang Shi, 19112020058@fudan.edu.cn; Xuegong Zhou, zhouxg@fudan.edu.cn; Hao Zhou, zhouhao@fudan.edu.cn; Lingli Wang, llwang@fudan.edu.cn, State Key Laboratory of ASIC and System, Fudan University, No.825 Zhangheng Road, Shanghai, China, 201203.

1936-7406/2022/3-ART $15.00
https://doi.org/10.1145/3519599

In academic researches, the routing architecture is mainly based on the CBs and SBs [2][10], where CBs are used to connect LB pins with routing channel tracks while SBs are used to connect horizontal and vertical tracks. In the early years, most researches focused on the SB design to improve the performance and routability, such as Wilton [49], Universal [4] and Disjoint which is also known as Subset SB pattern [21]. References [51][31] propose CS-Box and GSB architectures respectively which rely on bidirectional wires to improve the FPGA performance instead of the CB-SB architecture. In this paper, we propose a novel unidirectional interconnection to explore the routing architecture efficiency. This paper is an extension version of [43]. Our contributions include:

- We propose a novel unidirectional interconnection architecture, GIB. An LB can connect to four adjacent GIBs directly and a GIB has four adjacent LBs. Both LB input and output pins can directly connect to the adjacent GIB without programmable switches. Inside a GIB, LB pins and wire segments can connect with each other flexibly. An LB pin can connect to the routing channel tracks on the four sides of a GIB, while an LB pin can connect to one adjacent channel only in the CB-SB architecture. In addition, the pins from different neighboring LBs that connect to the same GIB can connect with each other with only one programmable switch.
- To further improve GIB architecture, we also enhance VTR to support bent wires which can achieve great improvement in delay in CB-SB architecture. Besides, we develop a searching framework based on simulated annealing algorithm to search for an near-optimal distribution of wire types which is similar to [28].
- In order to evaluate the performance of GIB architecture, we enhance the architecture description format and the Routing Resource Graph (RRG) generator in the latest VTR 8 [36]. We evaluate the performance of GIB architecture based on the area and delay parameters extracted from COFFE 2 [50] with VTR benchmarks [30]. Experimental results show that GIB architecture with all length-4 wires can improve the critical path delay by 8.3% and the area-delay product by 9.9% on average compared to CB-SB architecture with equivalent *fc* and *fs* values. After exploring different *fc* and *fs* values, GIB architecture can improve the critical path delay by 9.5% and achieve area-delay product savings by 11.1% on average. In addition, we evaluate the GIB architecture with bent wires and CB-SB architecture, results show that the GIB architecture with bent wires can achieve greater area-delay product improvement than GIB architecture with straight wires only. We also explore the GIB architecture with mixed wire types. Experimental results show that the optimized architecture can further improve the critical path delay by 16.4% and the area-delay product by 17.1% on average compared to the CB-SB architecture with length-4 wires.

The rest of this paper is organized as follows. Section 2 introduces the academic CB-SB routing architecture and the related work. The GIB architecture is proposed in Section 3. Section 4 gives the GIB architecture with bent wires and the algorithm enhancement and Section 5 introduces a searching framework for exploring mixed wire types. Section 6 presents the experimental results compared with the CB-SB architecture and the results of the searching framework. Section 7 concludes this paper with the future work.

2 BACKGROUND AND RELATED WORK

2.1 FPGA routing architecture

The modern island-style FPGA is composed of an array of LBs (or memory, DSP block) which are connected with routing channels through programmable switches as shown in Fig. 1. Most of modern FPGAs have unidirectional routing architectures which are mux based [3]. The routing channel width is described by W. The wire length L is defined as the number of LBs that the wire spans. The LB pins connect to channel wire segments via CBs. The horizontal and vertical wires can be interconnected with programmable switches inside SBs. CB flexibility is described by F_c which includes $F_{c,in}$ and $F_{c,out}$, and SB flexibility is represented by F_s. The value of $F_{c,in}$ and $F_{c,out}$ define the fraction of wires in routing channels that an LB input and ouput pin can connect to respectively,

while F_s defines the number of other wires to which an incoming wire can connect inside an SB [2] as shown in Fig. 2. These connections are all controlled by programmable switches which take up most of the delay and area. Hence, reducing the number of programmable switches on the critical path can improve the FPGA performance.

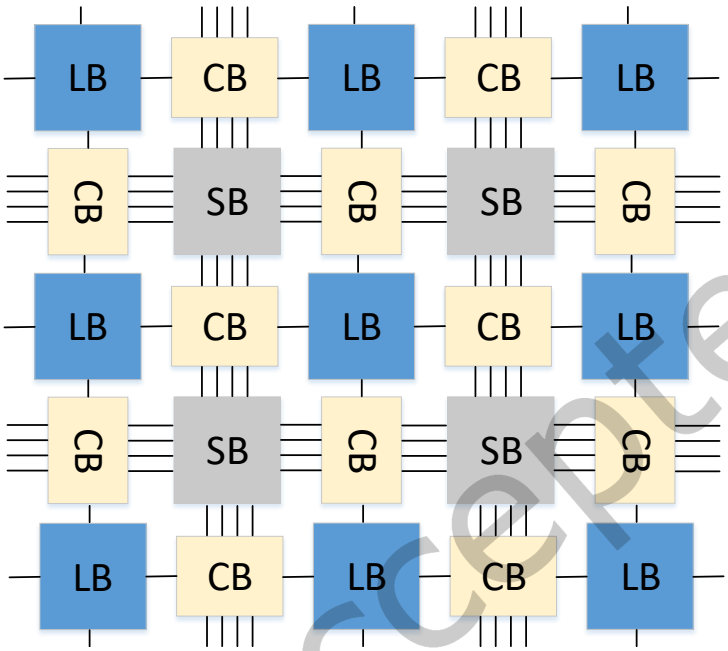


Fig. 1. Island-style FPGA architecture.

2.2 VTR platform

In this paper, the platform to explore the GIB architecture is VTR [36], which is an open-source framework to conduct FPGA architecture and CAD algorithm research. Currently, VTR is based on CB-SB routing architecture as shown in Fig. 2. The input pins and output pins are represented by black and red circles respectively. As unidirectional routing has only one driver per wire, the SB and output connection block are combined. The output pins can connect to the adjacent routing channel tracks through SB muxes directly. Horizontal and vertical tracks are connected with each other through SB muxes as well. The input pins can connect to the adjacent routing channel tracks through CB muxes.

The placement problem is to map a netlist of logic blocks (IOs or other macro blocks) onto their legal locations in an FPGA and attempt to minimize the wirelength and critical path delay. The placement algorithm in VTR is timing-driven based on simulated annealing [32]. In timing-driven mode, a delay look-up matrix is calculated to establish the delay between two blocks which are distance apart, then the timing analysis can be performed during FPGA placement. The router in VTR will be called to perform the routing between the two blocks.

The routing problem is to assign each net to wire segments and switches in an FPGA to route all the signals successfully while minimizing the wirelength and critical path delay. VTR 8 uses a timing-driven FPGA router [37] which is based upon the PathFinder negotiated-based algorithm [34]. The router uses a A*-like search algorithm to search for the paths from a source to the corresponding sink in the routing resource graph. The cost of a node

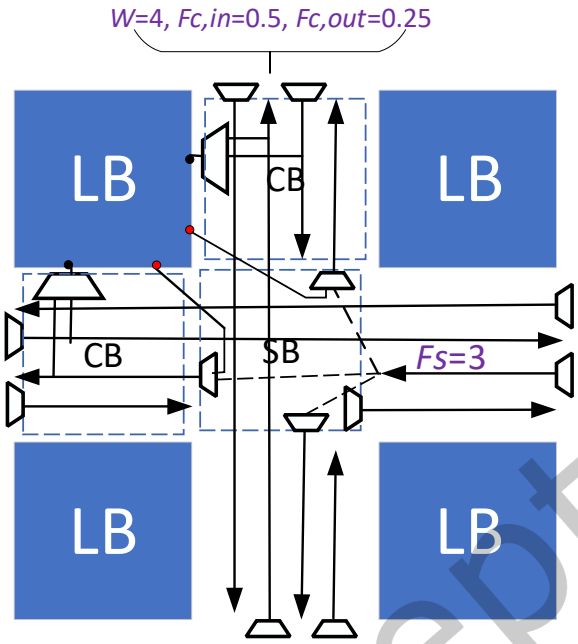


Fig. 2. The CB-SB routing architecture.

n is expressed by (1).

$$Cost(n) = C_{prev}(n) + \alpha \cdot C_{exp}(n) \tag{1}$$

$C_{prev}(n)$ is the sum of the cost from the source to the current node, while $C_{exp}(n)$ is the expected cost from the current node to the sink, and α is a weight factor. The router lookahead in VTR uses an undirected Djikastra-based search to quickly profile a large number of different routes through the routing resource graph, and estimates the cost of reaching the target sink through the current node being explored accurately [37]. The lookahead stores a cost table for each source wire type and orientation.

2.3 Related work

The routing architecture has a great impact on FPGA routability and performance. Routing accounts for more than 50% of the area and critical path delay [2]. Academic researchers have focused on the SB design to improve the routing architecture in the past several decades. There are several popular SB patterns such as Wilton, Universal and Disjoint which are developed for length-1 bidirectional architecture. Then, several SB designs focus on optimizing connections of wire segments that span more than one LBs, such as the Imran [33] and shifty [22] SB patterns. Reference [46] proposes a new switch box pattern for tileable FPGAs that achieves 12% improvement in the minimum routable channel width and 2% improvement in area-delay product. In [20], G. Lemieux et al. explore the use of sparse crossbars between the LB inputs and LUT inputs, and experimental results show that the sparse crossbars can reduce area by 10%-18% with no degradation to critical path delay. W. feng et al. [11] propose input interconnect block (IIB) which can connect signals from wire segments and LB feedbacks to LUT inputs. Experimental results show that 2 level IIBs can achieve big area savings with no routability overhead. Reference [51] proposes Connection-Switch Box (CS-Box) where CB and SB are simply combined. In CS-Box, an LB pin can connect to the wire segments on the CS-Box sides except for the side it belongs to. Experimental

results show that the number of programmable switches decreases by 11.81% at the cost of small increasing in the minimum channel width and the critical path delay. Reference [31] proposes a similar architecture, the general switch box (GSB), where an LB pin can connect to the wire segments on the four sides which can achieve better flexibility than CS-Box. The results show better improvement in delay with a small reduction in routing switches. Both CS-Box and GSB rely on bidirectional wires with tristate drivers which are not efficient in modern FPGAs. Experimental results show the unidirectional wiring can achieve 32% area-delay product savings compared to the bidirectional wiring [19]. Besides, unidirectional routing architecture consumes less energy and power in most instances than bidirectional architecture [14]. O. Mutukuda et al. [38] explore the effect of multi-bit connections using unidirectional routing on FPGA area efficiency and results show that unidirectional multi-bit routing architectures can achieve 8.6% reduction compared to the conventional routing architecture. Recently, Hu et al. [13] propose a tile-based interconnect model INTB to describe complex interconnect, like curve wires and two-level local muxes which cannot be described in VTR CB-SB model, but play an important role in interconnect performance. Routing architecture is also well designed in commercial FPGAs, such as general routing matrix (GRM) in Xilinx Virtex-5 Family which provides an array of routing switches between each internal component [35]. Each programmable element is tied to one GRM to improve the FPGA performance. In [41], Petersen et al. develop NetCracker, a flexible framework for extracting the characteristics of FPGA routing architectures and analyzing the routing network statistically. In addition, they describe and analyze the routing architecture of Xilinx 7-Series FPGAs in detail, including the routing channel width, all the wire-to-wire connections and CLB connections which have a large difference on the CB-SB architecture. The routing architecture is also well described in Intel Stratix family FPGAs [23–27].

There are several papers focusing on the wire segment patterns and channel segmentation design. Betz et al. [1] explore the best distribution of routing segment lengths and the best mix of pass transistor and tri-state buffer routing switches through an experimental method. TORCH [28] uses a stochastic method to quickly locate near-optimal solutions in designing FPGA channel segmentation and switch patterning without exhaustively enumerating all design points. To improve the FPGA performance, references [44, 48] propose a hard-wired routing pattern to reduce the number of programmable switches and achieves great improvement in delay, area and power dissipation. Reference [6] presents an architectural exploration of two diagonal tracks in FPGAs in addition to the vertical and horizontal tracks and achieves good improvement in the routing channel width and delay at the cost of area because of the added switch count, and more loading of tracks. Recently, reference [45] proposes a bent routing pattern based on VTR to enhance the routing architecture which can achieve 11% area-delay product savings. Reference [42] proposes the nearest neighbor interconnect architecture where direct connections are used between two LBs, bypassing all the intermediate routing switches and achieves 6.4% improvement in performance. Reference [39] enhances classical FPGA architecture with direct connections between intra- and intercluster LUTs to mitigate the delay of programmable routing and achieve 2.77% improvement in the critical path delay. In commercial FPGAs, Xilinx's Virtex-5 family implements a new diagonally-symmetrical interconnect architecture instead of the traditional wire segments [35]. In Versal architecture, a harden NoC is implemented as a separate level of interconnect to improve on the efficiency of bit level interconnect and a dedicated local interconnect is designed to support more versatile intra-CLB connectivity [12]. In Intel's Arria 10 FPGA [47], to improve the long line utilization these are made bi-directional via tri-stated driver input muxes (DIMs) replacing the direct-drive wires in most preceding devices. A highly pipelined routing architecture is proposed to address the problem that the RC delay per physical distance increases as the process geometry shrinks and fast output pins are added, connecting the LUT outputs directly to the routing in Intel's Agilex FPGAs [8].

This paper is largely inspired by [51][31]. LB pins and wire segments can connect with each other flexibly inside a GIB. To the best of our knowledge, there is no academic paper applying similar ideas to the unidirectional routing architecture in modern FPGAs. Besides, mixed wire types including straight and bent wires [45] are explored to make the GIB architecture perform better.

3 GIB ROUTING ARCHITECTURE

In this section, we propose the GIB architecture as shown in Fig. 3, where an LB can connect to four adjacent GIBs without programmable switches and a GIB has four adjacent LBs. Inside a GIB, LB pins and wire segments can connect with each other with significant flexibility improvement. In addition, three parameters are defined to model the GIB architecture.

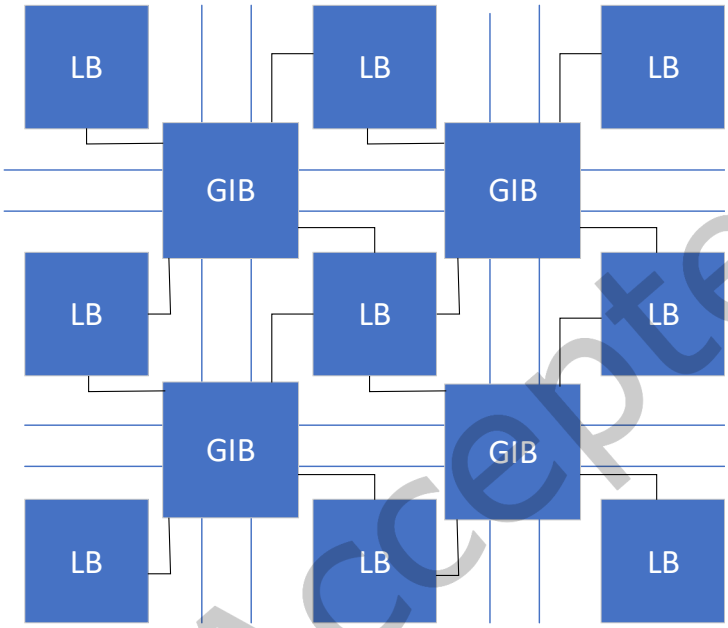


Fig. 3. GIB architecture.

3.1 GIB architecture

In GIB architecture, an LB pin can connect to the routing channel tracks on the four sides of a GIB while an LB pin can only connect to the one adjacent routing channel in classical CB-SB architecture. In the latest VTR 8, the pin location specifications are enhanced to allow LB pins to access both horizontal and vertical routing wires by creating two physically equivalent pins for each logical pin. In GIB architecture, each LB can connect to four adjacent GIBs and each LB can connect to more routing channels at different locations. Besides, output pins and input pins can connect with each other inside a GIB with only one programmable switch. In classical CB-SB architecture, output pins need to connect to the wire segments through SB muxes firstly, then connect to input pins through CB muxes. These connections between output pins and input pins are called neighbor interconnects below. The neighbor interconnects are similar to the direct link interconnects in Stratix IV FPGAs which allows the logic array block (LAB) to drive into the local interconnect of its neighbors [9]. In the latest VTR 8, it uses <direct> tag in the FPGA architecture description file to describe a dedicated connection between two block pins, such as carry chains. In this paper, we use a parameterized approach to describe the neighbor interconnects in GIB architecture which will be discussed in Section 3.3. A comparison of classical CB-SB and GIB architecture is shown in Fig. 4. Wire segments are denoted by blue circles while input pins and output pins are denoted by black circles and red circles respectively. Neighbor interconnects are shown in red lines in Fig. 4 (b). It can be seen

that GIB architecture can achieve much more routing flexibility than the CB-SB architecture. The wire segment connections between each other are not shown because they have the same definitions in CB-SB architecture and GIB architecture.

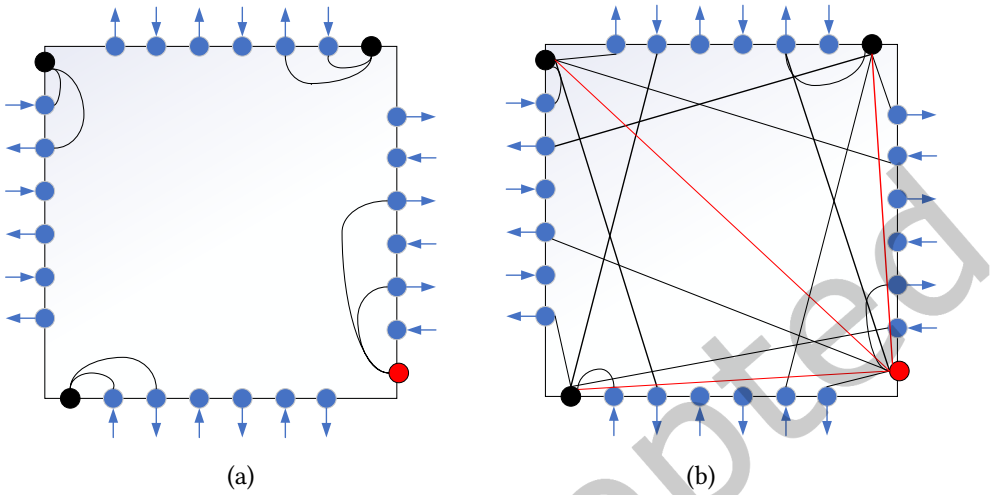


Fig. 4. Comparison of CB-SB and GIB architecture, (a) CB-SB architecture and (b) GIB architecture.

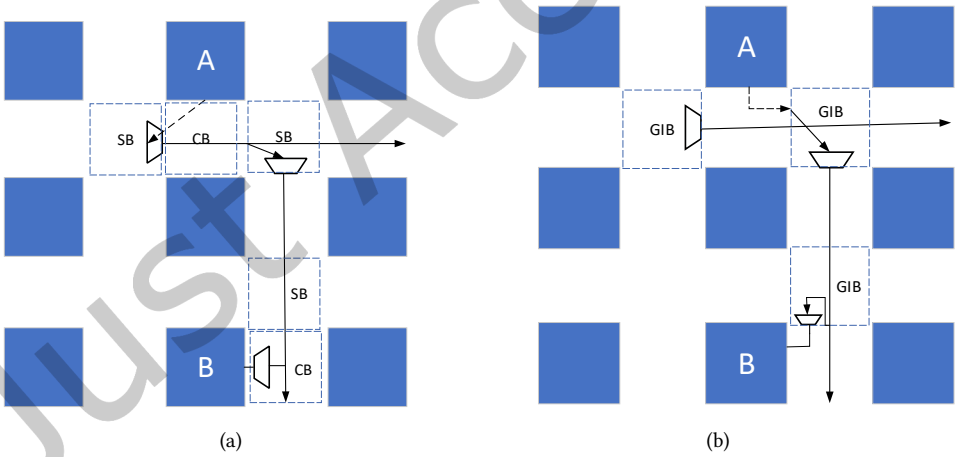


Fig. 5. The routing path compared CB-SB with GIB architecture, (a) CB-SB architecture and (b) GIB architecture.

The routing path generally starts from an LB output pin, and terminates at an LB input pin. In CB-SB architecture, an LB output pin passes through programmable switches in SBs, which eventually connects to LB input pins or IOs through CBs. With GIB architecture, LB pins can connect to the routing channel tracks on the four sides of a GIB through programmable switches and even connect to other LB pins with neighbor interconnects inside a GIB. Hence, GIB architecture can possibly reduce the number of programmable switches on the critical path.

As shown in Fig. 5 (a), an LB output pin connects to a target LB input pin through two SB muxes and one CB mux. With the GIB routing architecture as shown in Fig. 5 (b), one buffered mux can be saved. Through reducing the number of the programmable switches on the critical path, delay can be reduced. Evidence shows that the number of segments (switches), instead of wirelength, used by a net is the most dominating factor in determining routing delay in an FPGA [5]. We can also compute the interconnect delay approximatively in Fig. 5 using Elmore delay model in VTR. Assuming that the delay of LB pin connects to GIB is approximate to one length-1 wire, the net delay of Fig. 5 can be calculated in (2) and (3).

$$T_{CB-SB} = 2 \cdot T_{SB\_mux} + T_{CB\_mux} + 1.5 \cdot R_{metal} \cdot C_L + 2 \cdot R_{switch} \cdot C_L \tag{2}$$

$$T_{GIB} = T_{GIB\_smux} + T_{GIB\_imux} + 1.5 \cdot R_{metal} \cdot C_L + R_{switch} \cdot C_L \tag{3}$$

$$T_{CB-SB} - T_{GIB} = T_{SB\_mux} + R_{switch} \cdot C_L \tag{4}$$

T_{CB_mux} and T_{SB_mux} are the delays of the signal propagates through an SB mux and a CB mux respectively. T_{GIB_smux} and T_{GIB_imux} are the delay of the signal propagates through the driving mux of a wire segment and the mux of an input pin respectively. Assuming that $T_{SB\_mux} = T_{GIB\_smux}$, $T_{CB\_mux} = T_{GIB\_imux}$, R_{metal} and R_{switch} are the resistance per unit length of this wire segment and the resistance of the switch. C_L gives the total capacitance (metal plus parasitic capacitance) of the wire segment. Obviously, the delay in CB-SB is longer than that in GIB from (4).

3.2 GIB architecture enhanced in VTR

To model GIB architecture in VTR [14], we enhance the FPGA architecture description format. The connections in GIB can be divided into three types:

1. The connections between LB pins and wire segments,
2. The connections between different wire segments,
3. The connections between LB output pins and LB input pins.

To describe the above GIB connections, three parameters are defined accordingly, *fc*, *fs* and *fn*: *fc* defines the fraction of wire segments in routing channels that an LB pin can connect to; *fs* defines the number of other wires that a wire can connect to; *fn* defines the number of input pins that an output pin can connect to through neighbor interconnects inside a GIB.

As shown in Fig. 6, we enhance the *fc* tag in the XML architecture description format [2]. For every tile, we set four *fc* values for LB input pins and four *fc* values for LB output pins which correspond to four sides in a GIB. For an LB input pin, each of four *fc* values corresponds to the routing channel tracks that a pin can connect to one side of the GIB model. It can connect to different unidirectional wire pairs modeled by VTR as shown in Fig. 7 (a), where each pair corresponds to two wire segments with opposite directions. The 1st side is defined as the side that the pin belongs to. And the 2nd, 3rd and 4th sides correspond to the opposite side, the left side and the right side respectively as shown in Fig. 7. For LB output pins, they can only connect to those wires that go out of the GIB because unidirectional wires can only be driven in the start point as shown in Fig. 7 (b). So, the connected routing channel tracks are the half of *fc* values. For symmetry, the first and the second *fc* values have to be the same. So are the third and the fourth *fc* values. Supposing the value of F_c is set to 0.1 for both input pins and output pins in CB-SB architecture, *fc* values from four sides should add up to 0.1 for input pins and 0.2 for output pins in GIB architecture to achieve the equivalent pins flexibility. The *fs* value defines the number of other wires that a wire can connect to inside the GIBs which is similar to the F_s in CB-SB architecture. Each wire segment

[2]https://github.com/shikc/GIB_xml.

can connect to three other wire segments when the value of *fs* is set to 3. The value of *fn* is set to 3 which means an LB output pin can connect to three adjacent LB input pins inside a GIB through neighbor interconnects. It will be described in detail in Section 3.3.

```
<! - - fc value is divided into four parts, corresponding to four GIB sides respectively - ->
<fc in_type="frac" in_val="fc_in1 fc_in2 fc_in3 fc_in4" out_type="frac" out_val="fc_out1 fc_out2 fc_out3
fc_out4"/>
<! - - fn value defines the number of connections between output pins and input pins through neighbor
interconnect - ->
<neighbor_interconnect fn=3 />
```

Fig. 6. Example of enhanced XML tags for *fc* and *fn* values.

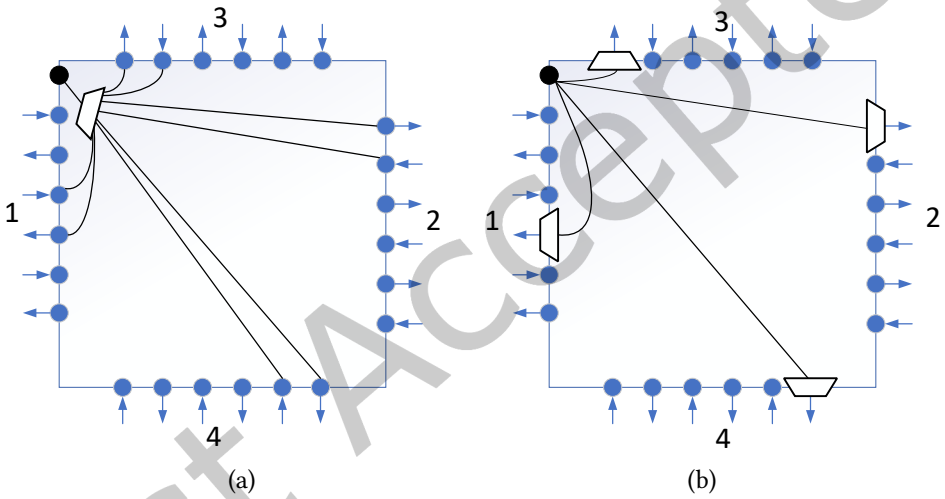


Fig. 7. An example of LB pins connections in GIB, (a) for input pins and (b) for output pins.

3.3 Neighbor interconnect

To improve GIB architecture, the neighbor interconnects are added which can be described by *fn*. The neighbor interconnects are similar to the direct link interconnects in Stratix IV FPGAs [9]. The radius [42] is defined to represent the distance of two LBs that connect with each other through neighbor interconnects. Notice that the radius between two diagonal LBs is also defined as 1. For example, the radius between the LB in the upper left corner and the LB in the bottom right corner in Fig. 8 is 1. To illustrate the advantage of neighbor interconnects, we run the VTR flow with the provided Stratix IV-like FPGA architecture and benchmarks. Then, the percentage of net connections with a radius of 1 is counted as shown in Table 1. where Δx and Δy stand for the distances between the source node and sink node in the x and y directions respectively. Experimental results show that the number of net connections with a radius of 1 account for 28.7% of the whole net connections on average. These nets connections can be connected through neighbor interconnects in GIB architecture instead of wire segments.

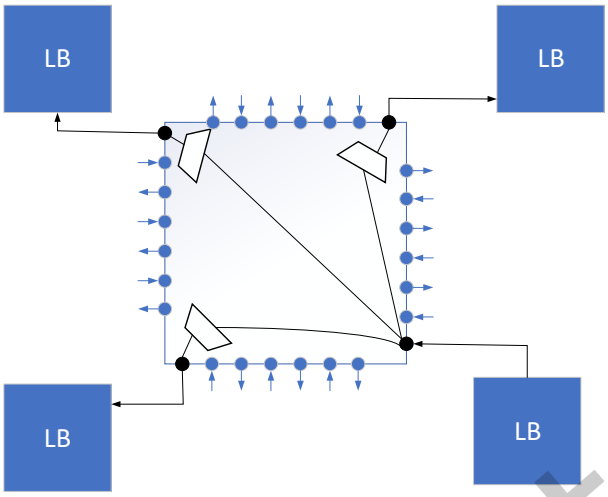


Fig. 8. Neighbor connects in GIB architecture.

Besides, the percentage of the inter-cluster delay that the radius is 1 in the total critical path delay is counted as shown in Table 1. Experimental results show that the inter-cluster delay that the radius is 1 accounts for 11.4% of the critical path delay on average. With neighbor interconnects, the critical path delay can be improved by reducing the programmable switch number.

We specify that an LB output pin can connect to three adjacent LB input pins which come from three LBs as shown in Fig. 8. These three input pins are located in different sides of the GIB and they are from different LBs. These pins are connected through buffered muxes. With these neighbor interconnects, an output pin can connect to three adjacent LB input pins with only one mux without passing through any wire segment. It can save two programmable switches compared with the CB-SB architecture. Hence, the critical path delay can be well reduced. However, too many neighbor interconnects will bring extra area cost and redundancy because LB input pins are logically equivalent with crossbar. Hence, we only set fn = 3 for an LB output pin.

Table 1. Percentage of net connections and inter-cluster delay with radius = 1.

$(\Delta x, \Delta y)$	percentage with radius = 1	
	net connections	inter-cluster delay
(1, 0)	8.90%	2.80%
(0, 1)	10.50%	4.20%
(1, 1)	9.30%	4.40%
Sum	**28.70%**	**11.40%**

3.4 CB-SB modeling in GIB

GIB architecture can be simplified as classical CB-SB architecture. When the first fc value is set to non-zero and the other three fc values are set to zero in the enhanced XML tags as shown in Fig. 6, the pin can only connect to one side which it belongs to as shown in Fig. 9. Besides, the fn value is set to 0 which means there is no neighbor

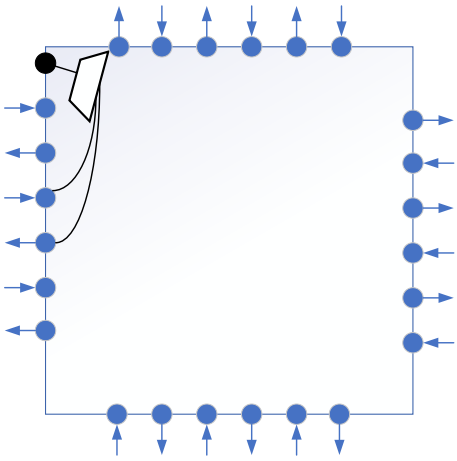


Fig. 9. CB-SB modelling in GIB.

interconnect. Under this parameter setting, as far as the routing architecture connections are concerned, GIB architecture and classical CB-SB architecture are equivalent.

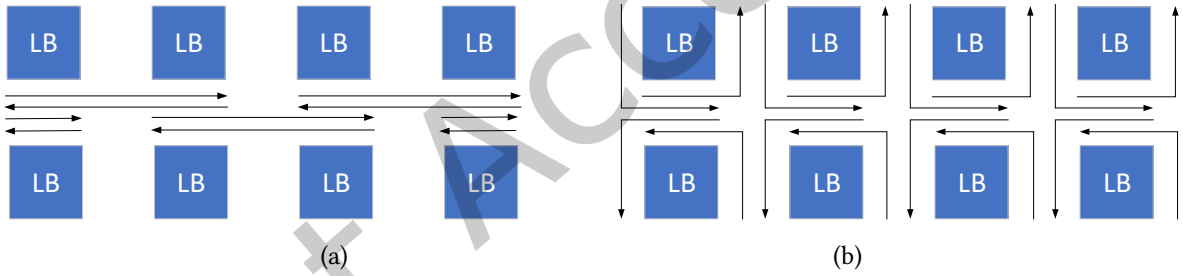


Fig. 10. Straight and bent wires in FPGA architecture with length-2, (a)straight wires, (b) bent wires.

4 GIB WITH BENT WIRES

4.1 Bent (Diagonal) wires

To explore the GIB architecture with more complex interconnects, we enhance VTR to support bent wires [45] which are also called diagonal wires in Xilinx FPGAs [35]. Experimental results have shown that bent wires can achieve great area-delay product savings [45]. Each length-L wire segment contains a start point, $(L-1)$ middle points and an end point. We use $\{CC, CW\}$ to describe the bent types at each middle point of the wire segment, where CC represents the counterclockwise type and CW represents the clockwise type. Besides, ST is used to describe the straight type. For instance, <CC> can be used to describe the length-2 bent wires in Fig. 10 (b). Typically, we stagger the starting and ending points of the wire segments whose length is greater than 1 so that each LB has the chance to connect to the beginning of a wire. Then, the routing tile can be replicated to form the whole FPGA routing fabric. A duplicable group of length-L wires is the smallest unit in the staggering loop

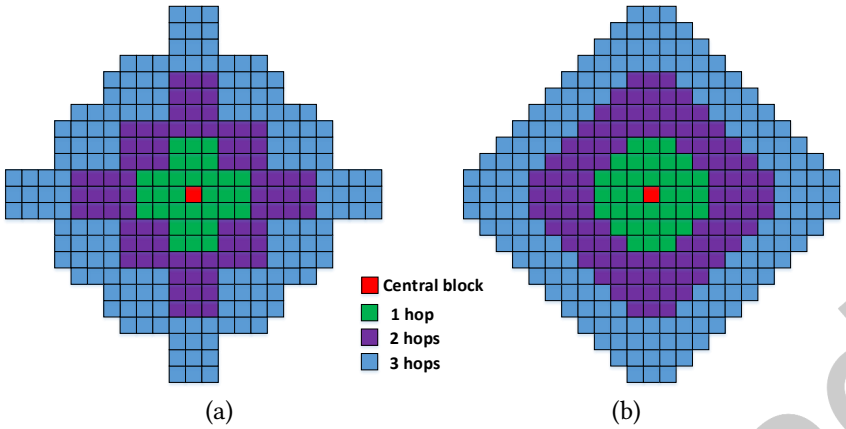


Fig. 11. The hops distribution in CB-SB architecture, (a) with length-4 straight wires, (b) with length-4 bent wires.

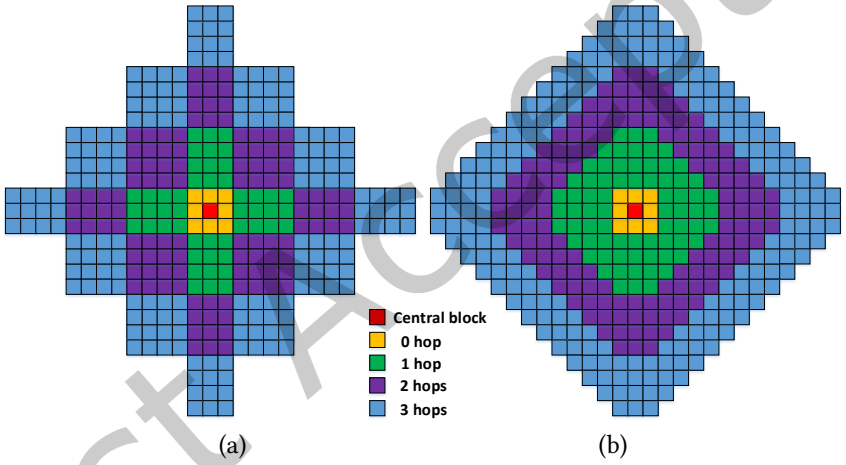


Fig. 12. The hops distribution in GIB architecture, (a) with length-4 straight wires, (b) with length-4 bent wires.

and consists of $2L$ tracks. Fig. 10 (a) shows a duplicable group of length-2 straight wires, and Fig. 10 (b) shows a duplicable group of length-2 bent wires. Other details about bent wires can be found in [45]. Fig. 11 (a) and (b) show the CB-SB routing architecture with length-4 straight wires and the CB-SB routing architecture with length-4 bent wires respectively. The number of hops means the number of wire segments used to connect to two LBs. Fig. 12 (a) and (b) show the GIB architecture with length-4 straight wires and the GIB architecture with length-4 bent wires respectively. GIB architecture makes more LBs accessible with the same number of hops. Besides, some LBs can be reached with 0 hop from the central LB, this is because of the existence of neighbor interconnects. Table 2 shows the exact number of LBs that can be reached from the central LB with different hops in CB-SB and GIB routing architecture.

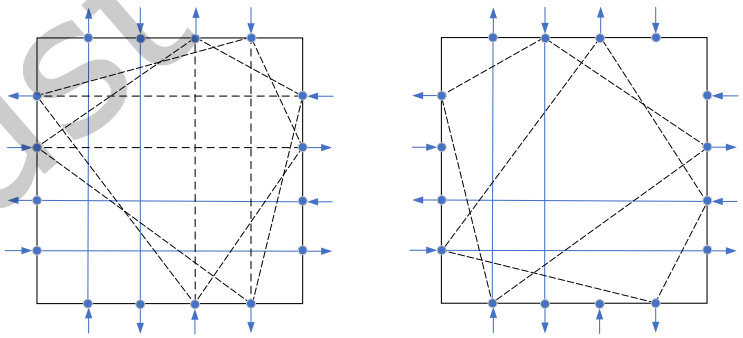
Table 2. The routing resources compared GIB with CB-SB architecture.

Hops	Number of LBs reachable with length-4 wires			
	CB-SB		GIB	
	straight wires	bent wires	straight wires	bent wires
0	0	0	8	8
1	32	36	48	72
2	84	104	112	120
3	164	168	176	200
Total	280	308	344	400

4.2 Connection enhancement

In this paper, we still use SB to describe the wire-to-wire connections in GIB. The commonly used SB patterns contain Wilton, Disjoint and Universal as we mentioned earlier in this paper. Wilton is the most efficient for single-length routing architectures. As we enhance VTR to support GIB and bent wires, the SB pattern needs to be optimized. Firstly, the connections between different wires need to be adjusted because of the appearance of bent wires. Fig. 13 shows the modified Wilton SB pattern for multi-length wires in VTR 8. The solid lines stand for the wire segments that pass through the SB and the dashed lines denote the programmable switches in SB. In classical routing architecture with straight wires, for all segments that start/end at that switch block, the connections follow the Wilton SB pattern as Fig. 13 (a) shows. For segments that pass through the switch block that can also turn there, the Wilton SB pattern cannot be used because unidirectional wire segments can only be driven at the start point. So the connections are assigned to starting segments following a round-robin scheme to balance mux size in VTR as Fig. 13 (b) shows.

In the routing architecture with bent wires, the connections also follow the Wilton SB pattern for all segments that start/end at that switch block as Fig. 14 (a) shows. For segments that pass through the SB, the connections are assigned to starting segments following a round-robin scheme which is similar to how VTR does. However, due to the appearance of bent wires, the connections have changed a lot as Fig. 14 (b) shows.



(a) (b)

Fig. 13. The SB pattern with straight wire segments, (a) ending segments to starting segments, (b) passing segments to starting segments.

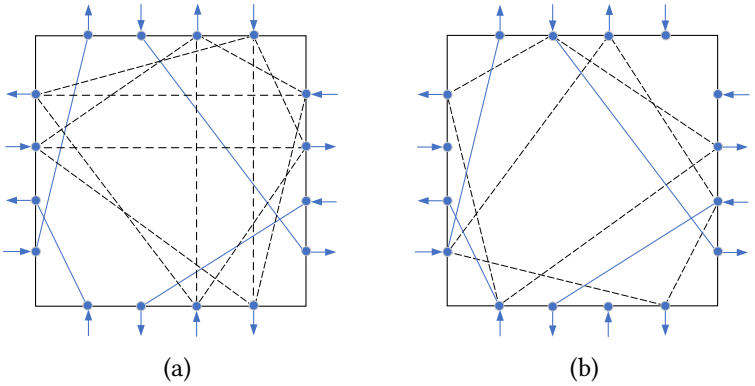


Fig. 14. The SB pattern with bent wire segments, (a) ending segments to starting segments, (b) passing segments to starting segments.

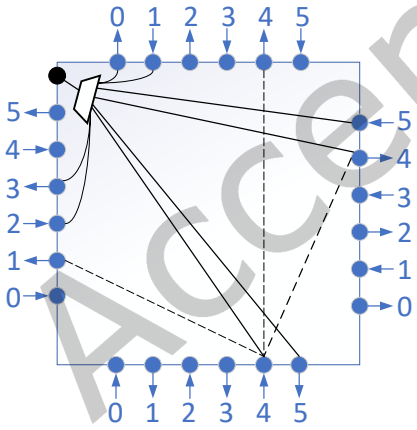


Fig. 15. The redundant connections in GIB.

Besides, because each LB pin can connect to the wire segments on the four sides of the GIB, there may exists some redundant connections as shown in Fig. 15 which may cause the loss of routability. The LB input pin can connect to track 4 on the bottom side and track 4 on the right side of the GIB through the mux. However, track 4 on the bottom side and track 4 on the right side can connect with each other through SB mux which means there exists one redundant connection of the pin. To solve the problem, we enhance the connections between the LB pins and the wire segments. When creating the pin-to-wire connections, we make the algorithm aware of the SB patterns. The connections between the pin and the wire segments on the 1st side follow the established way in VPR which uses a round-robin scheme to keep the track diversity. Then, the algorithm will also use a round-robin scheme to create the connections between the pin and the wire segments on the 2nd side. If the pin connects to the wire segments on the 2nd side which can connect to the wire segments on the 1st side through SB and the wire segments on the 1st side have been connected by the pin, the pin will reject this connections and connect to a different unidirectional wire pair. Similarly, the pin connects to the wire segments on the 3rd side

which cannot connect to the connected wire segments of the pin on the 1st and 2nd side through SB. We use the same method to connect the pin with the wire segments on the 4th side.

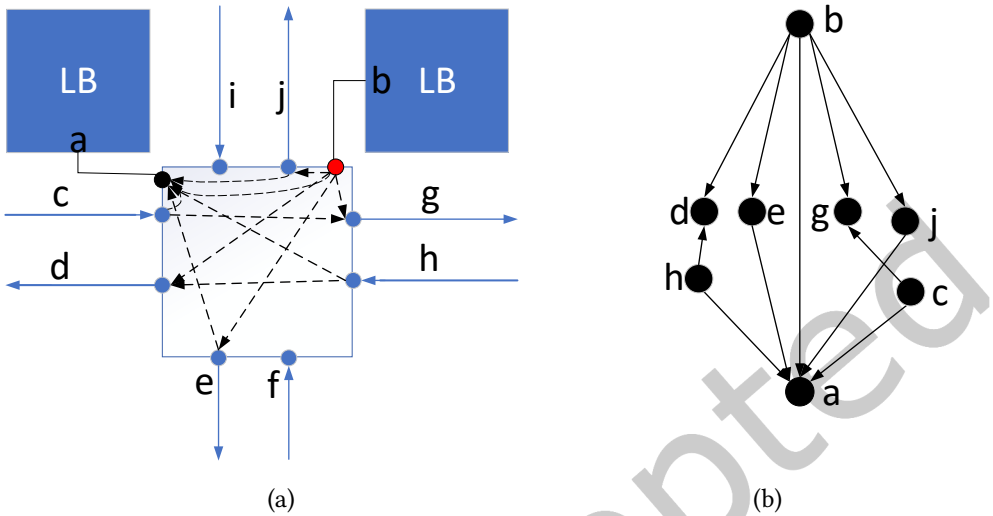


Fig. 16. Routing Resource Graph, (a) GIB architecture, (b) the corresponding RRG.

4.3 RRG generator

To support the GIB architecture, we enhance the RRG generator in VTR which is modelled to describe the FPGA routing architecture. FPGA routing resources are presented by a directed graph $G = (V, E)$, where V denotes routing nodes which can be LB pins or wire segments, and E stands for the programmable connections between different nodes. And the router tries to find routing paths to implement the connectivity of the circuit while minimizing the wirelength and the critical path delay. For the CB-SB architecture supported by VTR, one pin can only connect to the adjacent channel tracks. For the GIB architecture, a pin can connect to different channel tracks from four sides in GIB and there are neighbor interconnects between LB input pins and output pins which can achieve better flexibility as shown in Fig. 16. In addition, it still supports different wire segments including straight and bent wires. Bent wires are divided into several parts according to the bent points. For example, the length-2 bent wires are divided into two length-1 parts in Fig. 10 (b). The new RRG generator treats each part of the bent wire as a routing node. And the adjacent parts are connected by a non-configurable delayless switch.

4.4 Router lookahead enhancement

The router lookahead algorithm is enhanced to support the bent wire segments. Each bent wire type is also treated as one wire type which is similar to straight wires. Similar to VPR, the router lookahead run an undirected Djikastra-based search from each wire type to record the node cost required from the source coordinate to the sink coordinate. Because the router algorithm in VTR is based on PathFinder which routes signals on a generic routing resource graph, there is no need to do any algorithm modification for the GIB architecture.

4.5 Area and delay modeling

VTR measures the whole FPGA area in *minimum-width transistor areas* [7]. One *minimum-width transistor area* is the area of the smallest possible contactable transistor plus the spacing to neighboring transistors for a specific

process technology. The drive-strength of a transistor can be increased by either widening its diffusion region or by adding parallel diffusion regions. In other words, increasing the drive-strength of a transistor will increase its area. The delay is estimated with the Elmore delay model in VTR. The GIB architecture enhances the connections between LB pins and wire segments as well as the connections between LB pins. Hence, the default area and delay models in VTR are still suitable for GIB architecture.

5 A SEARCHING FRAMEWORK TO EXPLORE MIXED WIRE TYPES

5.1 The architecture searching flow

Since modern FPGAs often contain a variety of wire segments with different lengths for a tradeoff between routability and delay [23–27]. Short wires can achieve better routability while long wires can be used to improve the delay of the necessary long connections. So, we develop a searching framework based on simulated annealing algorithm to search for an approximate optimal distribution of wire types which is similar to [28, 45]. The simulated annealing (SA) algorithm is a stochastic optimizing algorithm which mimics the annealing process used to gradually cool molten metal to produce high-quality metal structures [17]. Fig. 17 shows the architecture searching flow. It starts with an initial architecture, and then iterates the process of generating new architecture and evaluating the cost until exiting the outer loop. As the annealing temperature drops, the probability of changing the architecture and the difference of the architectures decreases. The cost function is defined in (5), k means the number of the benchmark circuits we use to evaluate the cost, $Area_{i,base}$ and $Delay_{i,base}$ denote the area and critical path delay of the benchmarks on the baseline architecture respectively, $Area_{i,a}$ and $Delay_{i,a}$ represent the area and critical path delay of the benchmarks on the new architecture respectively. We use $Cost(b)$ to stand for the cost of the best architecture currently. If $\Delta c = Cost(a) - Cost(b) < 0$, the architecture is accepted, otherwise, it has some chance of being rejected. The purpose is to prevent the SA algorithm from being trapped in a local minimum. In the following experiments, k is set to 17 which includes all VTR benchmarks except *LU32PEEng*, *mcml* and *stereovision2* to save the run-time since they are exceptionally slow. In addition, we use 17 threads to run VTR simultaneously to further reduce the run-time.

$$Cost(a) = \frac{1}{k}\sum_{i=1}^{k}\frac{Area_{i,a}}{Area_{i,base}} \times \frac{Delay_{i,a}}{Delay_{i,base}} \tag{5}$$

5.2 The architecture changing strategy

The probability of changing the architecture and the difference of the architectures is decided by the annealing temperature. As the annealing temperature drops, the probability of changing the architecture and the difference of the two architectures decreases. There are four parameters can be changed, including the length, rate, bent pattern of each wire segment and the number of wire types.

- The length of each wire segment: the set of segment lengths to be explored is {1~8}. During the updating, the change in length is random. Then, it will also generate a bent pattern randomly and keep the rate of this wire segment unchanged.
- The rate of each wire segment: for each selected wire segment, the number of this wire segment is going to be added or subtracted by one duplicable group of this kind of wire segment which means the rate of each wire segment can be added or subtracted by $(2L)/W$, where L is the wire length and W is the routing channel width.
- The bent pattern of each wire segment: the change is done by randomly selecting a bent point which is then set to a random bent type from {*ST*, *CC*, *CW*}.

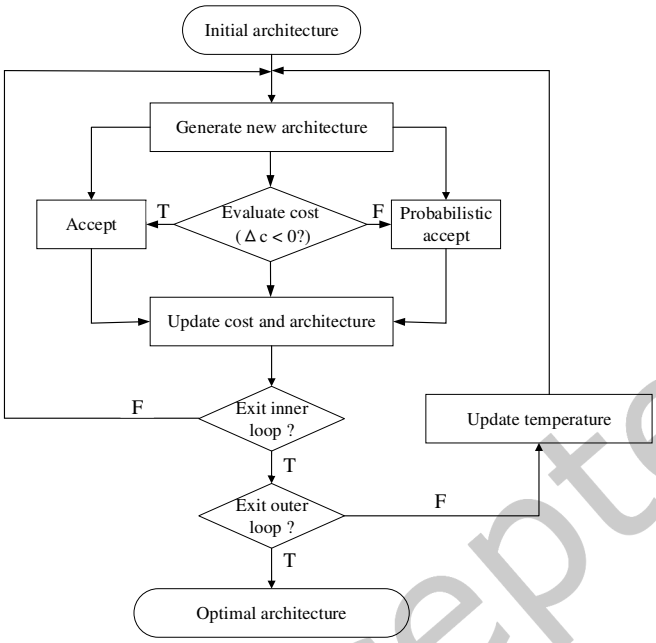


Fig. 17. The architecture searching flow.

- The number of wire types: an existing segment type can be deleted, or a new segment type can be added where the length, rate, and the bent pattern of the wire segment are generated randomly.

6 EXPERIMENTAL RESULTS

In this section, we introduce the FPGA baseline architecture, and then compare GIB architecture with CB-SB architecture based on VTR with provided benchmarks. Finally, we explore mixed wire types to make GIB architecture perform better.

6.1 Baseline architecture

In this paper, we use an island-based FPGA architecture whose area and delay parameters are extracted from COFFE 2 [50] at the 22 nm technology node which is the same as in [45]. COFFE 2 is a fully automated transistor sizing tool for FPGAs which measures delay and area by relying on HSPICE simulation. An LB is composed of ten 6-input fracturable LUTs and a local routing architecture with 50% connectivity. Memories are configurable 32K block RAMs that can operate in either single-port mode or dual-port mode. The memory has a configurable aspect ratio ranging from 32Kbits × 1 to 1K × 32 in dual-port mode and 32K × 1 to 512 × 64 in single-port mode. DSP modules are 36 × 36 fracturable multipliers which can operate as two 18 × 18 fracturable multipliers, and each 18 × 18 multiplier can be configured as two 9 × 9 multipliers. The IOs of this architecture are all on the perimeter and each IO contains 8 IO pins which can be configured to be input or output pins. Segments are all length-4 wires which can achieve the best area-delay tradeoff [40]. The F_c value is set to 0.1 for input pins and output pins which can achieve good performance [40], and F_s = 3. Experiments with single-driver routing

architecture [19] have confirmed that F_s = 3 is appropriate for this architecture. The SB pattern is Wilton, and the routing channel width is set to 300 which is reasonable in prior works [40, 45].

6.2 GIB architecture with symmetry *fc* values

In this section, we compare GIB architecture with symmetry *fc* values with CB-SB architecture. At the same time, other parameters are fixed. We divide F_c value in CB-SB into four equal values for four sides in GIB to achieve the same number of connections for pins. For example, F_c = 0.1 in CB-SB architecture is equivalent to *fc* = (0.025 0.025 0.025 0.025) for input pins and *fc* = (0.05 0.05 0.05 0.05) for output pins in GIB architecture. Four groups of different *fc* values are chosen to evaluate GIB architecture respectively as shown in Table 3. The experimental results show that GIB architecture can achieve about 8.3% improvement in the critical path delay and 9.9% improvement in area-delay product on average compared with CB-SB architecture as shown in Table 3, where the ratio is obtained by normalizing to CB-SB architecture. Besides, there is a small reduction in area (1.8%). One of the reasons for area reduction is that the IOs or LB pins in the perimeter of the FPGA device cannot connect to all four sides of GIBs. As shown in Fig. 18, any IO in the right boundary can only connect to three sides of the GIB, because there is no channel track on the right side of the GIB which costs less area. Another reason is that an input pin may connect to a wire segment repeatedly. For example, the value of *fc* is 0.68 for input pins in CB-SB architecture and the routing channel width is set to 6. That means an input pin can connect to 4 adjacent routing channel tracks since $6 \times 0.68 = 4$. Similarly, the value of *fc* is set to (0.17 0.17 0.17 0.17) for input pins in GIB architecture. That means that an input pin can connect to 4 routing channel tracks distributed in four sides of GIB since $6 \times 0.17 + 6 \times 0.17 + 6 \times 0.17 + 6 \times 0.17 = 4$. However, the wire segments in opposite sides may be the same wire segment. As shown in Fig. 19, the input pin connects to one horizontal wire segment and two vertical wire segments which are shown in red lines. The size of mux decreases because of the reduction in fan-ins which leads to the area reduction.

Table 3. Results of GIB compared with CB-SB architecture with symmetry *fc* values

Fc	Area Ratio	Critical Path Delay Ratio	Area-Delay Ratio
0.1	98.20%	90.50%	88.90%
0.12	98.10%	92.70%	91.00%
0.16	98.00%	92.90%	91.00%
0.2	98.40%	90.80%	89.40%
Avg. improvement	**1.80%**	**8.30%**	**9.90%**

6.3 GIB architecture with asymmetry *fc* distribution

To find whether different *fc* value distributions affect FPGA performance, we set six groups of different *fc* values for GIB as shown in Table 4. With the constraints that the four *fc* values add up to 0.1 for input pins and 0.2 for output pins, we compare the results with CB-SB architecture. The *fc* distribution in group 1 is symmetry. As shown in Fig. 20, the experimental results show that the FPGA architecture with symmetry *fc* distribution can achieve the best improvement in the area-delay product.

6.4 GIB architecture with different *fc* values

In this section, we will explore more combinations of *fc* values to achieve better improvement in area-delay tradeoff. Because too large *fc* values will bring extra area cost, the upper bound of *fc* values is (0.05 0.05 0.05 0.05) for input pins and (0.10 0.10 0.10 0.10) for output pins. The experimental results are compared with the baseline

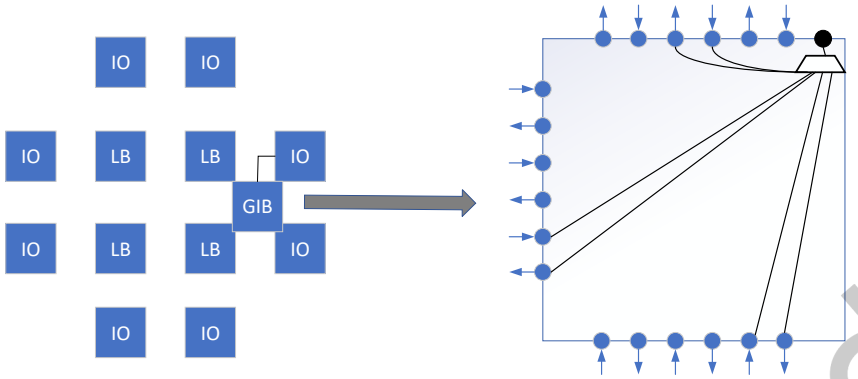


Fig. 18. An example of IO connects to GIB.

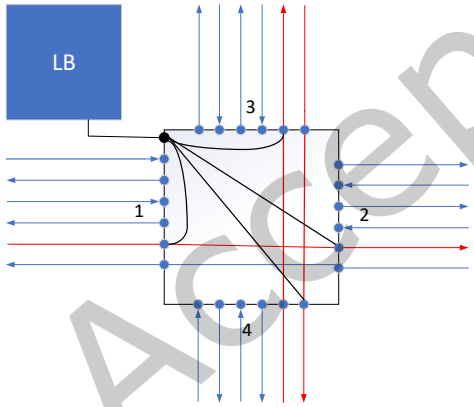


Fig. 19. Connections between input pin and wire segments.

Table 4. Groups of GIB architecture with different *fc* distributions

Group	*fc*, in_val	*fc*, out_val
1	(0.025 0.025 0.025 0.025)	(0.05 0.05 0.05 0.05)
2	(0.04 0.01 0.04 0.01)	(0.05 0.05 0.05 0.05)
3	(0.03 0.02 0.025 0.025)	(0.08 0.08 0.02 0.02)
4	(0.025 0.025 0.025 0.025)	(0.07 0.07 0.03 0.03)
5	(0.03 0.02 0.025 0.025)	(0.05 0.05 0.05 0.05)
6	(0.03 0.02 0.03 0.02)	(0.05 0.05 0.05 0.05)

FPGA architecture. As shown in Table 5, when *fc* values are set to (0.025 0.025 0.025 0.025) for input pins and (0.05 0.05 0.05 0.05) for output pins, it can achieve the best improvement by 11.1% in area-delay product. The ratio in Table 5 is obtained by normalized to the CB-SB baseline architecture. The detailed evaluation can be seen in

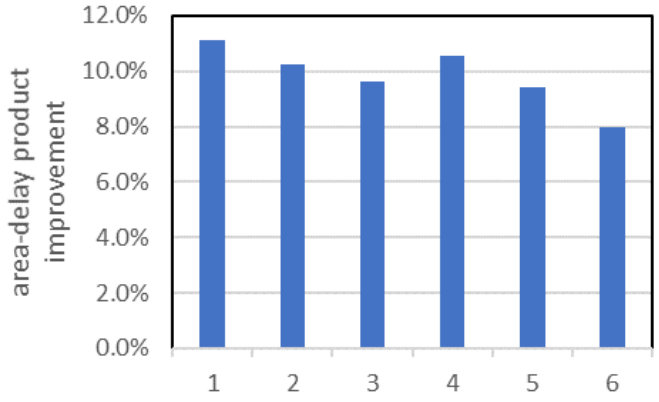


Fig. 20. The area-delay product improvement in GIB with different *fc* distributions compared to CB-SB architecture.

Table 6, where "*" means that the circuit can not go through VTR to evaluate power consumption. Results show that it can also achieve improvement by 9.6% in the power consumption on average. Particularly for those large benchmark circuits for which critical path delay is more than 20 ns such as *arm_core*, *bgm*, *LU8PEEng*, *LU32PEEng*, *mcml* as shown in Table 6, results show that it can achieve 13% improvement in area-delay product on average. There are also several small circuits whose performances become worse as shown in Table 6. After analyzing the critical path of these circuits, we find that the placements are changed which affects the routing results. During the placement, the placer [32] in VTR will call for the router [37] to estimate the inter-cluster delay which relies on the routing architecture. After enhancing the RRG generator in VTR to support GIB architecture, the placer leads to different placements. For these circuits whose performances become worse, there exist some regions getting congested which leads to longer critical path delay.

Table 5. Comparison of GIB and CB-SB baseline architecture

fc, in_val	*fc*, out_val	Area Ratio	Delay Ratio	Area-Delay Ratio
(0.03 0.03 0.03 0.03)	(0.05 0.05 0.05 0.05)	99.30%	92.70%	92.10%
(0.025 0.025 0.025 0.025)	(0.04 0.04 0.04 0.04)	97.00%	92.60%	89.80%
(0.025 0.025 0.025 0.025)	(0.05 0.05 0.05 0.05)	98.20%	90.50%	88.90%
(0.025 0.025 0.025 0.025)	(0.06 0.06 0.06 0.06)	99.40%	91.50%	90.90%
(0.03 0.03 0.03 0.03)	(0.06 0.06 0.06 0.06)	100.50%	92.40%	92.80%
(0.04 0.04 0.04 0.04)	(0.05 0.05 0.05 0.05)	101.70%	89.80%	91.30%
(0.04 0.04 0.04 0.04)	(0.08 0.08 0.08 0.08)	105.30%	91.70%	96.60%
(0.05 0.05 0.05 0.05)	(0.10 0.10 0.10 0.10)	110.90%	90.60%	100.50%

6.5 Routability comparision of GIB and CB-SB architecture

In this section, we compare the routability of GIB architecture and CB-SB architecture with W_{min}, the minimum routing channel width of each circuit. We run the VTR router repeatedly to find the W_{min} of each circuit. The *fc* values are set to (0.025 0.025 0.025 0.025) for input pins and (0.05 0.05 0.05 0.05) for output pins in GIB architecture

Table 6. Results of GIB compared with CB-SB architecture

Circuit	Total Area(10^6)			Critical Path Delay(ns)			Area-Delay(10^6)			Power(W)		
	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio
arm_core	52.65	53.58	98.3%	18.43	21.22	86.9%	970.3	1136.7	85.4%			
bgm	107.15	109.08	98.2%	19.17	21.94	87.4%	2054.3	2392.7	85.9%	0.3180	0.3467	91.7%
blob_merge	24.96	25.41	98.2%	9.86	11.28	87.4%	246.1	286.7	85.8%	0.0723	0.0832	86.9%
boundtop	3.59	3.66	98.1%	2.01	2.30	87.6%	7.2	8.4	86.0%	0.0082	0.0093	88.2%
ch_intrinsics	3.01	3.05	98.6%	2.38	2.57	92.8%	7.2	7.8	91.5%	0.0093	0.0100	92.7%
diffeq1	4.51	4.59	98.3%	15.41	16.21	95.0%	69.5	74.4	93.4%	0.0146	0.0160	91.4%
diffeq2	7.78	7.93	98.1%	13.09	13.52	96.8%	101.9	107.2	95.0%	0.0123	0.0148	83.0%
LU8PEEng	83.02	84.53	98.2%	77.37	89.26	86.7%	6422.9	7544.8	85.1%	0.3400	0.3647	93.2%
LU32PEEng	278.52	283.79	98.1%	77.02	87.19	88.3%	21451.5	24744.2	86.7%	1.4100	1.5130	93.2%
mcml	239.76	244.16	98.2%	60.57	64.55	93.8%	14522.0	15760.8	92.1%	*	*	*
mkDelayWorker32B	68.89	70.17	98.2%	6.98	8.26	84.5%	480.6	579.6	82.9%	0.1550	0.1696	91.4%
mkPktMerge	20.24	20.58	98.3%	4.29	4.00	107.1%	86.8	82.4	105.3%	0.0703	0.0716	98.2%
mkSMAdapter4B	9.88	10.04	98.4%	5.38	6.47	83.1%	53.1	65.0	81.7%	0.0270	0.0330	81.9%
or1200	18.90	19.25	98.2%	13.94	14.82	94.1%	263.5	285.4	92.3%	0.0523	0.0593	88.1%
raygentop	12.11	12.32	98.3%	4.74	5.31	89.3%	57.4	65.4	87.8%	0.0308	0.0327	94.1%
sha	8.72	8.88	98.1%	13.05	15.10	86.4%	113.8	134.1	84.8%	0.0280	0.0304	92.0%
stereovision0	32.56	33.16	98.2%	4.38	4.73	92.6%	142.5	156.7	90.9%	*	*	*
stereovision1	38.91	39.60	98.2%	4.25	4.33	98.3%	165.4	171.3	96.5%	*	*	*
stereovision2	295.22	300.73	98.2%	16.26	18.05	90.1%	4801.0	5428.5	88.4%	*	*	*
stereovision3	0.75	0.76	98.1%	2.34	2.87	81.7%	1.8	2.2	80.1%	*	*	*
Av. Improvement		**1.8%**			**9.5%**			**11.1%**			**9.6%**	

and the F_c value is set to 0.1 for input pins and output pins in CB-SB architecture. The results are shown in Table 7, where the ratio is obtained by normalized to the CB-SB baseline architecture. The area and delay are reported with the channel width set to 1.3 times W_{min} which is fairly common to create a low-stress routing. The increase in area is due to the increase in routing channel width which leads to larger mux size. Experimental results show that GIB architecture increases the minimum routing channel width by 7.5%. It is reasonable as the Wilton SB pattern is designed for CB-SB architecture which may not achieve the best routability in GIB architecture. So, design the SB pattern for GIB architecture is the main direction in the future. Besides, the IOs and LBs in the perimeter of FPGA can connect to less routing wires as shown in Fig. 18 is another reason for the increase in W_{min}. In particular, results shows that the larger circuits whose W_{min} are more than 80 (including 14 circuits except *boundtop*, *ch_intrinsics*, *diffeq2*, *mkDelayWorker32B*, *mkPktMerge*, *stereovision3*) have only 2.6% increase in W_{min} in GIB architecture in Table 7.

6.6 The effect of wire length and SB pattern

To explore the effect of wire length on GIB architecture, a single type of wire segments with different lengths are used. To obtain accurate and convincing experimental results, the timing and area parameters of wire segments and muxes are extracted from COFFE 2. Besides, different SB patterns are used to explore whether SB patterns influence the performance of GIB architecture. We use four kinds of wire segments with length-{2, 3, 4, 6}. Experimental results show that the longer wire segments can achieve more improvement as shown in Table 8. The GIB architecture with length-6 wire segments can improve critical path delay by 12.6% and improve the area-delay product by 13.5% on average compared to CB-SB architecture with length-6 wire segments. Besides, we run the VTR flow with three different SB patterns where the value of *fs* is set to 3. Experimental results show that Wilton SB pattern can achieve the most improvement on delay and area-delay product in GIB architecture

Table 7. W_{min} of GIB compared with CB-SB architecture

Circuit	W_{min}			Total Area (10^6)			Critical Path Delay (ns)		
	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio
arm_core	180	180	100.0%	47.65	48.14	99.0%	16.93	20.92	80.9%
bgm	144	142	101.4%	89.93	89.71	100.2%	18.26	19.98	91.4%
blob_merge	130	128	101.6%	20.09	20.01	100.4%	9.83	11.37	86.5%
boundtop	60	50	120.0%	2.26	2.15	105.4%	2.30	2.30	100.0%
ch_intrinsics	80	68	117.6%	2.01	1.92	104.4%	2.57	2.66	96.6%
diffeq1	94	86	109.3%	3.24	3.17	102.1%	15.63	16.38	95.5%
diffeq2	84	72	116.7%	5.49	5.24	104.7%	12.72	13.66	93.1%
LU8PEEng	146	146	100.0%	69.85	70.07	99.7%	74.13	88.60	83.7%
LU32PEEng	188	182	103.3%	257.15	255.86	100.5%	72.78	87.63	83.1%
mcml	144	136	105.9%	201.35	197.79	101.8%	57.18	64.59	88.5%
mkDelayWorker32B	62	50	124.0%	46.53	43.85	106.1%	7.40	7.87	94.1%
mkPktMerge	62	54	114.8%	13.38	12.83	104.3%	4.71	4.34	108.7%
mkSMAdapter4B	110	126	87.3%	7.46	7.85	95.1%	5.53	6.23	88.7%
or1200	140	138	101.4%	15.58	15.60	99.8%	13.37	14.66	91.3%
raygentop	98	94	104.3%	9.02	8.77	102.9%	4.74	5.31	89.3%
sha	114	114	100.0%	6.64	6.67	99.5%	12.82	14.98	85.6%
stereovision0	110	104	105.8%	24.75	24.57	100.8%	3.81	4.73	80.6%
stereovision1	140	130	107.7%	32.16	31.57	101.9%	4.19	4.45	94.2%
stereovision2	126	116	108.6%	235.91	230.66	102.3%	15.27	17.60	86.8%
stereovision3	58	48	120.8%	0.42	0.40	105.6%	2.64	3.26	81.2%
Av. Improvement		**-7.5%**			**-1.8%**			**10.0%**	

as shown in Fig. 21. The results are obtained by normalizing to CB-SB baseline architecture. This conclusion is consistent with CB-SB architecture[49].

Table 8. Results of GIB compared with CB-SB architecture with wire segments of different lengths

Length	Area Improvement	Critical Path Delay Improvement	Area-Delay Improvement
2	0.70%	3.30%	4.00%
3	2.40%	6.60%	8.80%
4	1.80%	9.50%	11.10%
6	1.00%	12.60%	13.50%

6.7 GIB architecture with bent wires

To explore the effect of GIB architecture with bent wires, we design nine GIB architectures with bent wires as shown in Fig. 22, in which straight means straight wires. The bent wire type and the corresponding bent type at each switchpoint of this type can be seen on the right of Fig. 22. The routing channel width is set to 300, and we set *fc* = (0.025 0.025 0.025 0.025) for input pins and *fc* = (0.05 0.05 0.05 0.05) for output pins which can achieve great area-delay product savings as discussed in Section 6.4. Besides, the wire length is restricted to 4 which aims to eliminate performance improvements due to different wire lengths. The results are obtained by normalizing to the baseline CB-SB architecture. Experimental results show that GIB architecture with bent wires can further improve the performance than GIB architecture with straight wires only. In addition, we use the

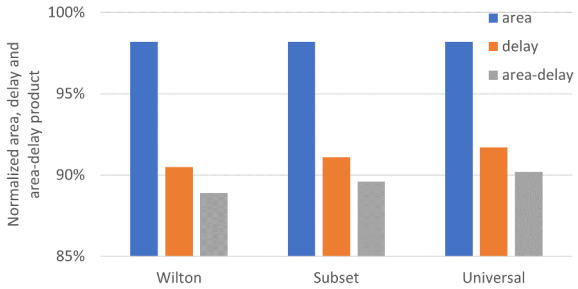


Fig. 21. The normalized area, delay and area-delay in GIB of different SB patterns.

searching framework introduced in Section 5 to search for a better GIB architecture with length-4 bent wires. After exploring the bent wire distributions, it can achieve 13.18% improvement on the critical path delay and 14.6% improvement on the area-delay product as shown in the last column of Fig. 22 and the wire type distributions can be seen in Table 9 in which "length" means the wire length, "ratio" means the proportion of the number of such wires to the total routing channel width and "<bend> list" includes the bent type at each switchpoint of this wire segment type. In addition, to explore the impact of SB pattern on the performance of GIB architecture with bent wires, we evaluate the nine GIB architectures in Fig. 22 with different SB patterns including Wilton, Subset and Universal. The average performance improvement can be seen in Fig. 23 by normalizing to the baseline CB-SB architecture. Experimental results show that Wilton SB pattern can still achieve the best area-delay product savings in GIB architecture with bent wires.

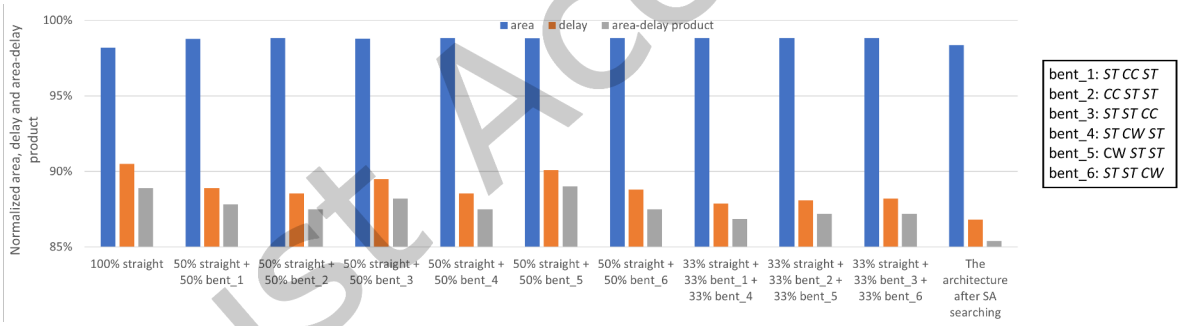


Fig. 22. The normalized area, delay and area-delay in GIB with bent wires.

Table 9. The SA searching segmentation result for bent wires

Length	Ratio	<Bend> List
4	52%	ST ST ST
4	21%	ST CC ST
4	27%	ST ST CW

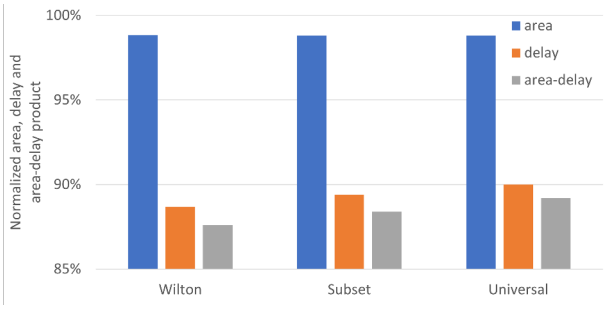


Fig. 23. The normalized area, delay and area-delay in GIB with bent wires of different SB patterns.

6.8 Mixed wire type distribution

To explore routing architecture with mixed wire types, we design four GIB architectures with single wire length and five GIB architectures with mixed wire lengths to compare with the baseline CB-SB architecture. The routing channel width is set to 300. The results are shown in Fig. 24 which are obtained by normalizing to the baseline CB-SB architecture. Results show that length-4 wiring can still the best area-delay tradeoff in the GIB architecture with single wire length which is consistent of previous work in CB-SB architecture [40]. Besides, the mix of length-3 and length-6 wiring can achieve better improvement in area-delay product than the single length wiring.

Then, we use the searching framework as mentioned in Section 5 to search for an approximate optimal distribution of wire types including straight and bent wires. The optional segment lengths contain {1,2,3,4,5,6,7,8}. The baseline architecture is the CB-SB architecture clarified in Section 6.1. The timing and area parameters of each wire type and mux are extracted from COFFE 2 at the 22 nm technology node. The routing channel width is fixed at 300. Fig. 25 shows the searching process where the x-axis is the number of iterations and the y-axis is the area-delay product improvement compared to the baseline CB-SB architecture. The total number of updated architecture points is 500 during the SA process. It takes about 5 days with one machine with Intel Xeon CPU E5-2620 @ 2.10GHz to accomplish the SA process. Experimental results show that the optimized architecture with mixed wire types can improve critical path delay by 16.4% and achieve 17.1% area-delay product savings on average compared to the baseline CB-SB architecture. The detailed performance evaluation are shown in Table 10. The ratio is obtained by normalizing to the baseline CB-SB architecture. Table 11 shows the wire distributions of the near-optimal architecture in area-delay tradeoff. "Ratio" means the proportion of the number of such wires to the total routing channel width and "<bend> list" includes the bent type at each switchpoint of this segment type. The disappearance of length-1 wire is the existence of neighbor interconnects in GIB architecture. The wire distributions are similar to some commercial FPGAs [26][47] which also contain length-{3, 4, 6} wires. The experimental results show that the GIB architecture with mixed wire types can improve the FPGA performance better.

In addition, we explore the effect of random seeds on the final optimized architecture as SA is a stochastic algorithm. We run the SA searching flow with three different seeds, and the experimental results are shown in Table 12. Results show that the optimized architecture with different seeds only have minor differences with similar improvement on the area-delay product. In the future, we will enhance the searching framework to explore larger searching space such as SB pattern, *fc* values and optimize the parameters in the SA framework.

7 CONCLUSION

In this paper, we propose a novel unidirectional routing architecture for modern FPGAs. Compared with VTR CB-SB architecture, GIB architecture with the equivalent *fc* and *fs* values achieves 8.3% improvement on the

Table 10. Results of the optimized GIB compared with CB-SB architecture

Circuit	Total Area(10^6)			Critical Path Delay(ns)			Area-Delay(10^6)		
	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio	GIB	CB-SB	Ratio
arm_core	53.55	53.58	100.0%	16.41	21.22	77.3%	878.62	1136.72	77.3%
bgm	104.52	109.08	95.8%	17.11	21.94	78.0%	1788.24	2392.73	74.7%
blob_merge	25.35	25.41	99.8%	8.88	11.28	78.7%	225.17	286.68	78.5%
boundtop	3.62	3.66	98.9%	1.86	2.30	80.8%	6.71	8.41	79.9%
ch_intrinsics	3.02	3.05	99.0%	2.30	2.57	89.5%	6.94	7.83	88.6%
diffeq1	4.54	4.59	99.0%	14.85	16.21	91.6%	67.43	74.36	90.7%
diffeq2	7.88	7.93	99.3%	12.26	13.52	90.7%	96.59	107.25	90.1%
LU8PEEng	84.53	84.53	100.0%	70.88	89.26	79.4%	5991.80	7544.76	79.4%
LU32PEEng	282.80	283.79	99.7%	72.81	87.19	83.5%	20589.34	24744.22	83.2%
mcml	242.10	244.16	99.2%	53.51	64.55	82.9%	12955.38	15760.76	82.2%
mkDelayWorker32B	70.13	70.17	99.9%	6.16	8.26	74.6%	431.93	579.56	74.5%
mkPktMerge	20.53	20.58	99.8%	3.97	4.00	99.3%	81.61	82.37	99.1%
mkSMAdapter4B	10.00	10.04	99.5%	5.24	6.47	81.0%	52.37	64.97	80.6%
or1200	19.19	19.25	99.7%	12.83	14.82	86.5%	246.13	285.37	86.2%
raygentop	12.26	12.32	99.5%	4.26	5.31	80.2%	52.22	65.39	79.9%
sha	8.83	8.88	99.4%	12.42	15.10	82.2%	109.61	134.15	81.7%
stereovision0	33.10	33.16	99.8%	3.25	4.73	68.9%	107.71	156.73	68.7%
stereovision1	39.54	39.60	99.9%	4.29	4.33	99.1%	169.54	171.32	99.0%
stereovision2	298.10	300.73	99.1%	15.19	18.05	84.1%	4525.45	5428.54	83.4%
stereovision3	0.74	0.76	96.8%	2.38	2.87	83.0%	1.76	2.19	80.4%
Av. Improvement		**0.8%**			**16.4%**			**17.1%**	

Table 11. Segmentation result of the SA searching framework

Length	Ratio	<Bend> List
3	37%	CW CC
4	42%	ST ST ST
6	21%	ST ST ST ST ST

Table 12. The final optimized architectures with different random seeds

Seed	Seed = 1			Seed = 2			Seed = 3		
	Length	Ratio	<Bend> List	Length	Ratio	<Bend> List	Length	Ratio	<Bend> List
Segmentation results	2	3%	CW	2	5%	CW	3	37%	CW CC
	3	22%	CW ST	3	20%	ST ST	4	42%	ST ST ST
	4	33%	ST ST ST	4	40%	ST ST ST	6	21%	ST ST ST ST ST
	6	42%	ST ST ST ST CW	6	35%	ST ST ST ST CC			
Area improvement	0.40%			-0.50%			0.80%		
Delay improvement	16.80%			17.00%			16.40%		
Area-delay improvement	17.10%			16.50%			17.10%		
Run-time	5.1 days			5.05 days			5.03 days		

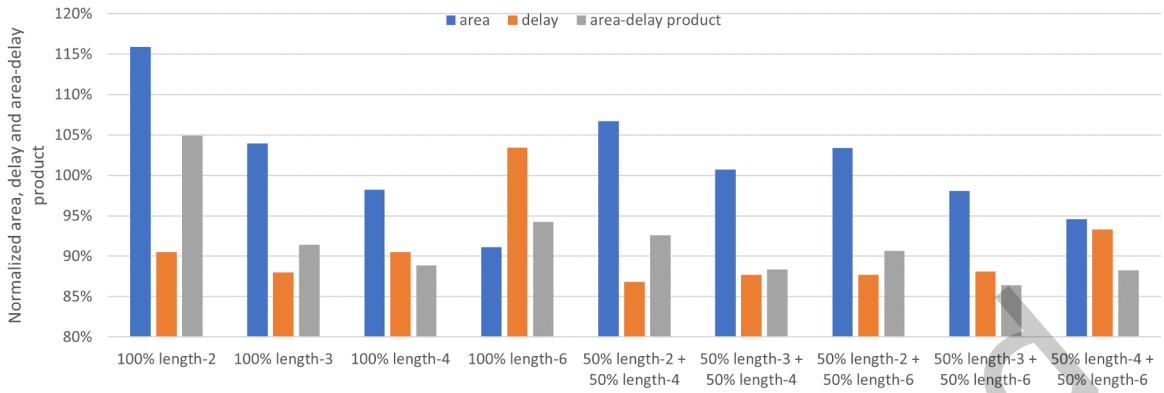


Fig. 24. The normalized area, delay and area-delay in GIB with mixed wire types.

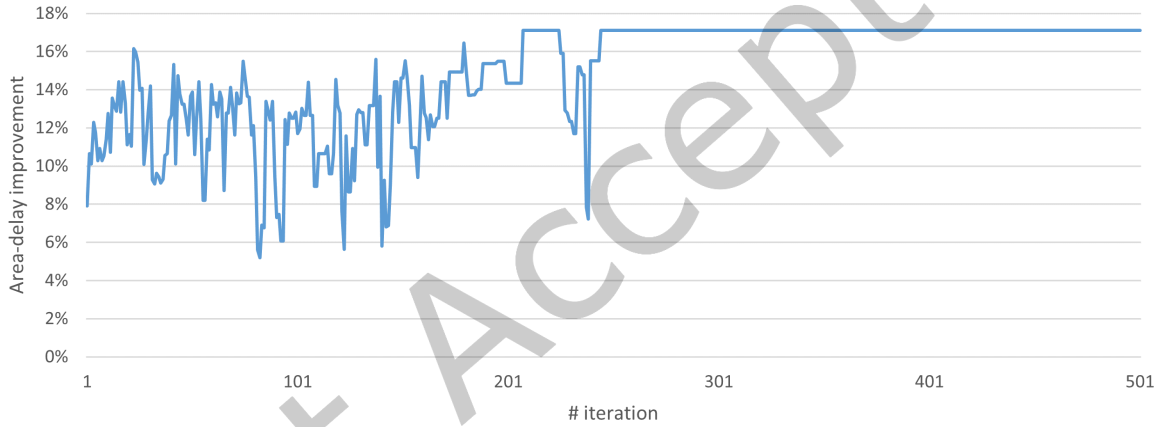


Fig. 25. The SA researching process.

critical path delay and 9.9% improvement on the area-delay product on average compared with CB-SB architecture. The GIB architecture with *fc* = (0.025 0.025 0.025 0.025) for input pins and *fc* = (0.05 0.05 0.05 0.05) for output pins can improve the critical path delay by approximately 9.5% and the area-delay product by around 11% on average. After exploring GIB architecture with mixed wire types including straight and bent wires, the optimized GIB architecture can improve the delay by 16.4% and area-delay product by 17.1% compared to the CB-SB architecture with length-4 wires. In the future, we will enhance the searching framework to explore larger searching space. In addition, we will also focus on optimizing GIB architecture for neural network, data communication and other specific applications.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation of China under Grant No. 61971143.

REFERENCES

[1] Vaughn Betz and Jonathan Rose. 1999. FPGA routing architecture: Segmentation and buffering to optimize speed and density. In *Proceedings of the ACM/SIGDA seventh international symposium on Field programmable gate arrays*. 59–68.

[2] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. 1999. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic Publishers.

[3] Alexander Brant and Guy GF Lemieux. 2012. ZUMA: An open FPGA overlay architecture. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 93–96.

[4] Yao-Wen Chang, DF Wong, and Chak-Kuen Wong. 1996. Universal switch modules for FPGA design. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 1, 1 (1996), 80–101.

[5] Yao-Wen Chang, Kai Zhu, and DF Wong. 2000. Timing-driven routing for symmetrical array-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 5, 3 (2000), 433–450.

[6] Sumanta Chaudhuri. 2009. Diagonal tracks in FPGAs: a performance evaluation. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 245–248.

[7] Charles Chiasson. 2013. *Optimization and modeling of FPGA circuitry in advanced process technology*. Ph.D. Dissertation.

[8] Jeffrey Chromczak, Mark Wheeler, Charles Chiasson, Dana How, Martin Langhammer, Tim Vanderhoek, Grace Zgheib, and Ilya Ganusov. 2020. Architectural Enhancements in Intel® AgilexâĎć FPGAs. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 140–149.

[9] Altera Corporation. 2011. Stratix IV Device Handbook, Volume 1. (2011).

[10] Soumya Eachempati, Arthur Nieuwoudt, Aman Gayasen, Narayanan Vijaykrishnan, and Yehia Massoud. 2007. Assessing carbon nanotube bundle interconnect for future FPGA architectures. In *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 1–6.

[11] Wenyi Feng and Sinan Kaptanoglu. 2008. Designing efficient input interconnect blocks for LUT clusters using counting and entropy. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 1, 1 (2008), 1–28.

[12] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. 2019. Xilinx adaptive compute acceleration platform: VersalâĎć architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 84–93.

[13] Chengyu Hu, Qinghua Duan, Peng Lu, Wei Liu, Jian Wang, and Jinmei Lai. 2020. A Tile-based Interconnect Model for FPGA Architecture Exploration. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. 113–118.

[14] Peter Jamieson, Wayne Luk, Steve JE Wilton, and George A Constantinides. 2009. An energy and power consumption analysis of FPGA routing architectures. In *2009 International Conference on Field-Programmable Technology*. IEEE, 324–327.

[15] Tanay Karnik and Sung-Mo Kang. 1995. An empirical model for accurate estimation of routing delay in FPGAs. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*. IEEE, 328–331.

[16] Muhammad Khellah, Stephen Brown, and Zvonko Vranesic. 1993. Modelling routing delays in SRAM-based FPGAs. In *Canadian Conference on VLSI*. Citeseer, 6B.

[17] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science* 220, 4598 (1983), 671–680.

[18] Ian Kuon and Jonathan Rose. 2007. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2 (2007), 203–215. https://doi.org/10.1109/TCAD.2006.884574

[19] Guy Lemieux, Edmund Lee, Marvin Tom, and Anthony Yu. 2004. Directional and single-driver wires in FPGA interconnect. In *Proceedings. 2004 IEEE International Conference on Field-Programmable Technology (IEEE Cat. No. 04EX921)*. IEEE, 41–48.

[20] Guy Lemieux and David Lewis. 2001. Using sparse crossbars within LUT. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*. 59–68.

[21] Guy G. Lemieux, Stephen D Brown, and Daniel Vranesic. 1997. On two-step routing for FPGAs. In *Proceedings of the 1997 international symposium on Physical design*. 60–66.

[22] Guy G. Lemieux and David M. Lewis. 2002. Analytical framework for switch block design. In *International Conference on Field Programmable Logic and Applications*. Springer, 122–131.

[23] David Lewis, Elias Ahmed, Gregg Baeckler, Vaughn Betz, Mark Bourgeault, David Cashman, David Galloway, Mike Hutton, Chris Lane, Andy Lee, et al. 2005. The Stratix II logic and routing architecture. In *Proceedings of the 2005 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 14–20.

[24] David Lewis, Elias Ahmed, David Cashman, Tim Vanderhoek, Chris Lane, Andy Lee, and Philip Pan. 2009. Architectural enhancements in Stratix-IIIâĎć and Stratix-IVâĎć. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 33–42.

[25] David Lewis, Vaughn Betz, David Jefferson, Andy Lee, Chris Lane, Paul Leventis, Sandy Marquardt, Cameron McClintock, Bruce Pedersen, Giles Powell, et al. 2003. The stratixâĎć routing and logic architecture. In *Proceedings of the 2003 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 12–20.

[26] David Lewis, David Cashman, Mark Chan, Jeffery Chromczak, Gary Lai, Andy Lee, Tim Vanderhoek, and Haiming Yu. 2013. Architectural enhancements in Stratix VâĎć. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 147–156.
[27] David Lewis, Gordon Chiu, Jeffrey Chromczak, David Galloway, Ben Gamsa, Valavan Manohararajah, Ian Milton, Tim Vanderhoek, and John Van Dyken. 2016. The StratixâĎć 10 highly pipelined FPGA architecture. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 159–168.
[28] Mingjie Lin, John Wawrzynek, and Abbas El Gamal. 2010. Exploring FPGA routing architecture stochastically. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 10 (2010), 1509–1522.
[29] Lee-Chung Lu. 2017. Physical design challenges and innovations to meet power, speed, and area scaling trend. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*. 63–63.
[30] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, et al. 2014. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7, 2 (2014), 1–30.
[31] Kejie Ma, Lingli Wang, Xuegong Zhou, Sheldon X-D Tan, and Jiarong Tong. 2010. General switch box modeling and optimization for FPGA routing architectures. In *2010 International Conference on Field-Programmable Technology*. IEEE, 320–323.
[32] Alexander Marquardt, Vaughn Betz, and Jonathan Rose. 2000. Timing-driven placement for FPGAs. In *Proceedings of the 2000 ACM/SIGDA international symposium on Field programmable gate arrays*. 203–213.
[33] M Imran Masud and Steven JE Wilton. 1999. A new switch block for segmented FPGAs. In *International Workshop on Field Programmable Logic and Applications*. Springer, 274–281.
[34] Larry McMurchie and Carl Ebeling. 2008. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Reconfigurable Computing*. Elsevier, 365–381.
[35] Petar Borisov Minev and Valentina Stoianova Kukenska. 2009. The Virtex-5 routing and logic architecture. *Annual Journal of Electronics, Technical University of Sofia* 3 (2009), 107–110.
[36] Kevin E. Murray, Oleg Petelin, Sheng Zhong, Jia Min Wang, Mohamed Eldafrawy, Jean-Philippe Legault, Eugene Sha, Aaron G Graham, Jean Wu, Matthew JP Walker, et al. 2020. VTR 8: High-performance cad and customizable FPGA architecture modelling. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13, 2 (2020), 1–55.
[37] Kevin E. Murray, Sheng Zhong, and Vaughn Betz. 2020. AIR: A Fast but Lazy Timing-Driven FPGA Router. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 338–344.
[38] Omesh Mutukuda, Andy Ye, and Gul Khan. 2010. The effect of multi-bit based connections on the area efficiency of FPGAs utilizing unidirectional routing resources. In *2010 International Conference on Field-Programmable Technology*. IEEE, 216–223.
[39] Stefan Nikolić, Grace Zgheib, and Paolo Ienne. 2020. Straight to the point: Intra-and intercluster LUT connections to mitigate the delay of programmable routing. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 150–160.
[40] Oleg Petelin and Vaughn Betz. 2016. The speed of diversity: Exploring complex FPGA routing topologies for the global metal layer. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 1–10. https://doi.org/10.1109/FPL.2016.7577326
[41] Morten B Petersen, Stefan Nikolić, and Mirjana Stojilović. 2021. NetCracker: A peek into the routing architecture of Xilinx 7-Series FPGAs. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 11–22.
[42] A. Roopchansingh and J. Rose. 2002. Nearest neighbour interconnect architecture in deep submicron FPGAs. In *Proceedings of the IEEE 2002 Custom Integrated Circuits Conference (Cat. No.02CH37285)*. 59–62. https://doi.org/10.1109/CICC.2002.1012766
[43] Kaichuang Shi, Hao Zhou, Xuegong Zhou, and Lingli Wang. 2020. GIB: A Novel Unidirectional Interconnection Architecture for FPGA. In *2020 International Conference on Field-Programmable Technology (ICFPT)*. 174–181. https://doi.org/10.1109/ICFPT51103.2020.00032
[44] Satish Sivaswamy, Gang Wang, Cristinel Ababei, Kia Bazargan, Ryan Kastner, and Eli Bozorgzadeh. 2005. HARP: hard-wired routing pattern FPGAs. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. 21–29.
[45] Xibo Sun, Hao Zhou, and Lingli Wang. 2019. Bent Routing Pattern for FPGA. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 9–16. https://doi.org/10.1109/FPL.2019.00012
[46] Xifan Tang, Edouard Giacomin, Aurélien Alacchi, and Pierre-Emmanuel Gaillardon. 2019. A study on switch block patterns for tileable FPGA routing architectures. In *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 247–250.
[47] Jeffrey Tyhach, Mike Hutton, Sean Atsatt, Arifur Rahman, Brad Vest, David Lewis, Martin Langhammer, Sergey Shumarayev, Tim Hoang, Allen Chan, et al. 2015. ArriaâĎć 10 device architecture. In *2015 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–8.
[48] G. Wang, S. Sivaswamy, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh. 2006. Statistical Analysis and Design of HARP FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 10 (2006), 2088–2102. https://doi.org/10.1109/TCAD.2005.859485
[49] Steven JE Wilton et al. 1997. *Architectures and algorithms for field-programmable gate arrays with embedded memory*. PhD thesis, University of Toronto.
[50] Sadegh Yazdanshenas and Vaughn Betz. 2019. COFFE 2: Automatic modelling and optimization of complex and heterogeneous FPGA architectures. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 1 (2019), 1–27.

[51] Catherine L. Zhou, Ray C. C. Cheung, and Yu-Liang Wu. 2004. What if merging connection and switch boxes-an experimental revisit on FPGA architectures. In *2004 International Conference on Communications, Circuits and Systems (IEEE Cat. No. 04EX914)*, Vol. 2. IEEE, 1295–1299.