electronics

MDPI

Article

Improving Seed-Based FPGA Packing with Indirect Connection for Realization of Neural Networks

Le Yu [1,*], Baojin Guo [1], Tian Zhi [2] and Lida Bai [3]

[1] School of Artificial Intelligence, Beijing Technology And Business University, Beijing 100048, China; 2030602043@st.btbu.edu.cn
[2] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[3] Shandong Cwise Microelectronics Technology Co., Ltd., Jinan 250102, China
* Correspondence: yule@btbu.edu.cn

Abstract: FPGAs are gaining favor among researchers in fields including artificial intelligence and big data due to their configurability and high level of parallelism. As the packing methods indisputably affect the implementation performance of FPGA chips, packing techniques play an important role in the design automation flow of FPGAs. In this paper, we propose a quantitative rule for packing priority of neural network circuits, and optimize the traditional seed-based packing methods with special primitives. The experiment result indicates that the proposed packing method achieves an average decrease of 8.45% in critical path delay compared to the VTR8.0 on Koios deep learning benchmarks.

Keywords: FPGA; EDA; packing; seed-based; indirect connections

Citation: Yu, L.; Guo, B.; Zhi, T.; Bai, L. Improving Seed-Based FPGA Packing with Indirect Connection for Realization of Neural Networks. *Electronics* **2023**, *12*, 2691. https://doi.org/10.3390/electronics12122691

Academic Editors: Akash Kumar, Andres Upegui, Laurent Gantel and Andrea Guerrieri

Received: 11 April 2023
Revised: 10 June 2023
Accepted: 14 June 2023
Published: 15 June 2023

1. Introduction

In recent years, Field Programmable Gate Array (FPGA) chips are being widely used in the acceleration of neural networks (NNs). NN applications such as image classification [1], object detection [2], and natural language processing [3] can take full advantage of the reconfigurable parallelism of FPGA architectures.

FPGAs typically consist of two-dimensional reconfigurable arrays, including Configurable Logic Blocks (CLBs), Block RAMs (BRAMs), Digital Signal Processing blocks (DSPs) [4,5], etc., and all these tiles are connected through programmable wires and switches. The back-end optimization of FPGAs is restricted by the pre-placed computing primitives and the pre-routed clock tree. As the packing methods indisputably affect the implementation performance of FPGA chips, packing techniques play an important role in the design automation flow of FPGAs.

Nowadays, the most commonly used packing algorithms are Seed-based algorithms, which pack the look up tables (LUTs) and flip-flops (FFs) together to implement the designated logic function. Seed-based algorithms construct new tiles by seeding each with an unpacked primitive and greedily absorbing its surrounding primitives according to attraction functions. However, emerging heterogeneous FPGA architectures present a new challenge to the traditional packing methods; heterogeneous IP Blocks, such as the BRAMs and DSPs, make traditional packing methods inefficient due to the unevenly distributed wiring topology.

In this paper, we propose an improved packing algorithm. The main contributions of this paper are as follows: (1) A quantitative rule for packing priority of neural network circuits is proposed. (2) The traditional seed-based packing methods with special primitives is optimized. Compared with Verilog-To-Routing(VTR) [6], the proposed packing method achieves an average reduction of 8.45% in latency at the cost of a 0.58% increase in resource consumption and a 7.55% increase in runtime for the optimized circuits without affecting other circuits.

2. Related Work

Previous algorithms for FPGA packing can be loosely categorized into seed-based packing and partitioning-based packing.

VPACK [7] is the first seed-based packing approach. It packs LUTs and FFs into BLE, and then into CLBs. T-VPACK [8] reduces the critical path delay of the circuit by modifying the attraction function. DPACK [9] adds the Manhattan distance to the attractive function of T-VPACK, which reduces the bus length by 16% and the critical path delay by 8% after placement and routing. For more complex logic blocks, ref. [10] proposes the AAPACK algorithm, which packs primitives into molecules and assemble clusters from a set of molecules. RSVPACK [11] is a packing algorithm for the XILINX V6 architecture that bridges the gap between academia and industry. DPPACK [12] adopts distributed parallel packing, which shortens the runtime by 1.4–3.2 times with acceptable quality degradation compared to AAPACK. [6] is an update to AAPack, optimized for seed selection and attraction functions.

PPFF [13] applies partition-based packing to FPGAs as a sub-step of placement. PPack [14] explores partition-based packing, which adds a significant amount of runtime compared to T-VPACK. PPack2 [15] is an improved version of PPack. Compared to T-VPack, PPack2 has an 11.2% reduction in critical path delay. PartSA [16] is a multi-threaded parallel packing algorithm that reduces runtime but increases wire length by 26%.

A summary of FPGA packing algorithms is shown in Table 1. Partitioning-based algorithms are effective at packing simple FPGAs, but they can struggle to handle the constraints present in commercial devices. Conversely, seed-based algorithms perform better in packing heterogeneous FPGAs. The seed-based algorithm adopts the same packing rule for tiles with different areas, which will affect the wire length around some tiles and even the delay of the critical path. This is more prominent in neural network circuits. This paper proposes improved seed-based packing algorithms for neural networks, which can reduce the critical path delays in the packing process.

Table 1. Summary of FPGA Packing Methods.

Packing Methods	Advantages	Disadvantages
partitioning-based packing [13–16]	effective	struggle to handle packing of heterogeneous clusters
seed-based packing [6–12]	excel at packing heterogeneous FPGAs	the influence of tiles of different sizes on wirelength was ignored

3. User Netlist

After circuit synthesis and mapping, a primitive-level user netlist is generated. It is composed of primitives and necessary connectivity.

3.1. Primitives

Primitives are fundamental units that cannot be separated, and their internal structure is usually treated as a black box. Each primitive contains one or more ports, including input and output ports, with each port having one or more pins. For example, the input port of a LUT usually has four to eight pins. In modern FPGAs, three frequently used primitives are DSPs, RAMs, and adders, which were not present in earlier generations of FPGA products. The features of these three types of primitives in application circuits are described below.

3.1.1. DSPs and RAMs

DSPs and RAMs are embedded reconfigurable IP blocks in FPGAs. However, the use of primitives inherently introduces significant latency. In advanced neural networks, about 80-90% of the operations are matrix multiplication [17], which means that the critical path often

includes DSPs or RAMs. Consequently, reducing the delay of the network connected by these primitives has become a pressing issue that requires immediate attention in the field.

3.1.2. Adders

Adders typically implement carry chains and are generally located in CLBs. In real-world applications, adders typically consist of multiple cascaded CLBs. Since adders re-use the LUT routing pins, a smaller number of LUT pins is required for CLBs with adders. Taking the Intel Stratix_10 architecture as an example, a CLB without adders can absorb a 6-bit LUT, while for CLBs with adders, the maximum number of input pins is four.

3.2. Connectivity

The connectivity between primitives in the user netlist is implemented through networks. Primitives are connected to a network through pins. A connected network is usually interconnected to the pins of multiple primitives, among which only one pin is an output pin. A network with a large number of primitives connected to it is commonly referred to as a high fan-out network, while a network with a smaller number of primitives is typically known as a low fan-out network.

There are three types of connectivity between primitives: direct connectivity, indirect connectivity and high fan-out connectivity. Direct connectivity refers to the connectivity between primitives through a low fan-out network. Indirect connectivity is the connection between primitives through different networks of the same tile. A high fan-out connectivity refers to the connectivity between primitives through a high fan-out network. The modes of connection are shown in Figure 1.

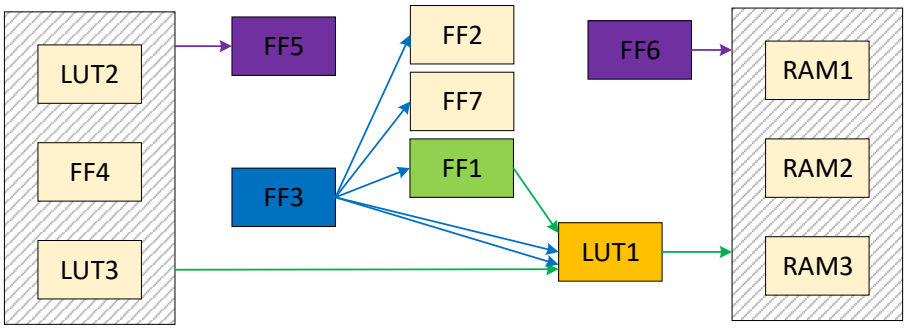

Figure 1. There are three types of connectivity between primitives. LUT1 is the seed, FF1 is a directly connected primitive, FF5 and FF6 are indirectly connected primitives, and FF3 is a high fan-out connected primitive.

Packer is designed to absorb the primitives of direct connections in order to reduce the number of external networks on the tile, which serves to reduce the overload of routing and computing. For primitives of indirect connections, packing them into the tiles can reduce the number of external connections of the tiles, which increases the relevancy of the connected tile.

4. Packing Methods

The path delay of FPGAs is affected by two factors: the internal delay of logic blocks and the delay introduced by programmable routing. The delay of logic blocks is fixed, while the delay of programmable routing is influenced by the wire length of nets. In order to minimize the delay of the nets that are connected to DSPs and RAMs, it is essential to reduce the length of these nets. However, since the area occupied by DSPs and RAMs is much larger than that of CLBs, some pins on DSPs and RAMs may be located far apart. As a result, even tiles that are situated around a DSP or RAM may be distant from the pins of the corresponding connection. Figure 2 shows the result of placement and routing in VTR8.0. In the figure, o1 stands for the DSP, o2 stands for the CLB. And two

primitives indirectly connected through o1 are grouped in o2, resulting in two nets between o1 and o2. Figure 2a shows two routing networks connected with DSP pins at the same coordinates, while Figure 2b shows two routing networks connected with DSP pins at different coordinates. It can be seen from Figure 2 that absorbing primitives indirectly connected through pins in the same location can reduce wire length. This is in contrast to absorbing primitives indirectly connected through pins in different locations. To minimize the delay of the nets that are linked to DSPs and RAMs, the packer prioritizes absorbing the primitives that are indirectly connected through pins in the same location based on the pin distribution of tile. This approach avoids absorbing primitives that are indirectly connected at different locations, thereby reducing the length of the nets connected by DSP and RAM after placement and routing.

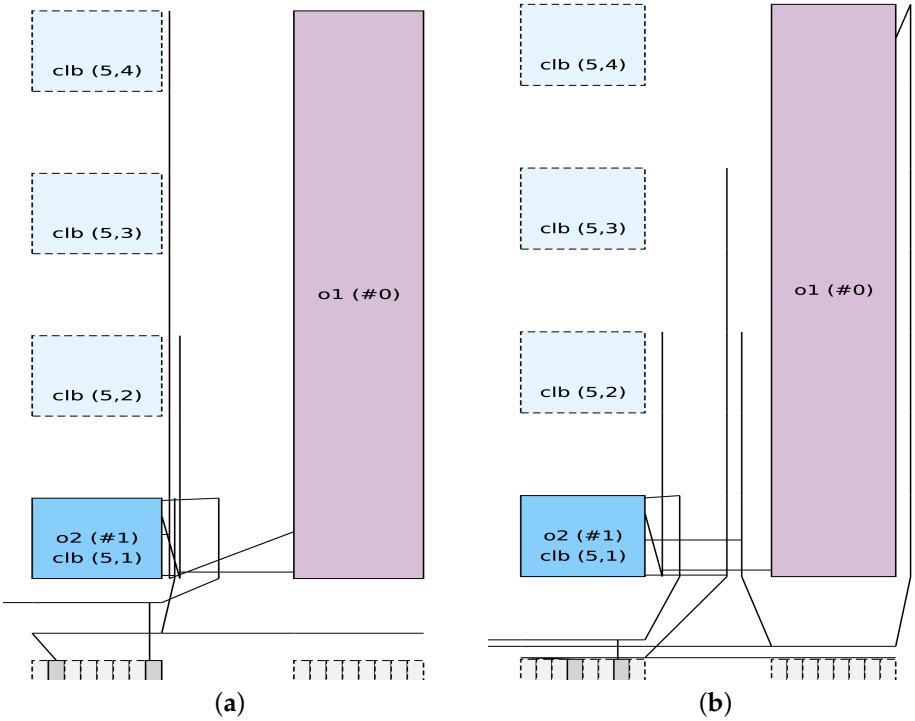

Figure 2. Connections between a CLB and a DSP via two nets (VTR8.0 visualization).

When a CLB is connected to multiple DSPs or RAMs, the average wire length between the CLB and these tiles tends to be high, as shown in Figure 3. This issue becomes more prevalent if there is a higher proportion of RAMs and DSPs in the circuit, as it increases the likelihood of multiple connections between the same CLB and these tiles. In contrast, if the circuit design incorporates a high proportion of adders, there will be fewer options for packers, and primitives in different positions will be absorbed. Moreover, the cascading of adders considers multiple CLBs as a single unit, which is often connected to multiple DSPs or RAMs. This interconnection can significantly impact the overall packing result.

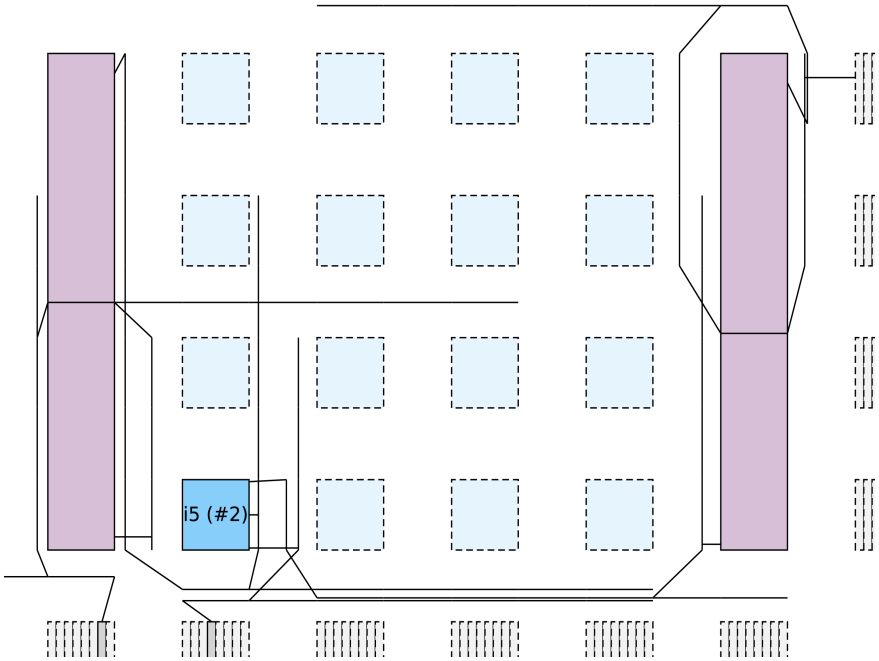

Figure 3. A CLB connects multiple DSPs (VTR8.0 visualization).

Table 2 presents a comparison between wire lengths and wiring segments utilized in connecting DSPs and CLBs. As the table illustrates, the connection relationships between CLBs, DSPs, and RAMs significantly impact wire length and wiring segment consumption. Therefore, it is essential to give priority to circuits that are indirectly connected via DSPs and RAMs. In this study, we refer to DSPs and RAMs that satisfy the specified requirements as special primitives, while referring to other primitives as normal primitives.

Table 2. Three types of connections between DSP and CLB.

Types of Connections	Total Wirelength	Maximum Net Length	Total Wiring Segments Used	Maximum Segments Used by a Net
Figure 2a	14	5	6	2
Figure 2b	21	9	8	3
Figure 3	47	12	17	5

The process of the packing algorithm in this paper is shown in Algorithm 1. First, the packer analyzes the proportion of various primitives in the user netlist to determine whether to use DSP and RAM as special primitives or not. The packer then groups the primitives into molecules, calculates the seed gain for each molecule and selects the molecule with the highest gain as the seed. After the seed is selected, the packer will absorb the molecules around the tile until the constraints of the tile are no longer satisfied or the surrounding molecules are all packed. The above process is repeated until all molecules are packed. Our packing algorithm is composed of three stages: primitive classification, seed selection, and molecule selection.

Algorithm 1 Pack algorithm.

Input: $fpga\_architecture$ and $netlist$
Output: $packed\_netlist$

if $number(DSP, RAM, adder) < threshold$ **then**
  $DSP, RAM \in special\ tiles$
else
  $DSP, RAM \in normal\ tiles$
end if
$moles \leftarrow atom2molecule(netlist)$
while $unpacked\_mol \neq \emptyset$ **do**
  $seed \leftarrow get\_highest\_seed\_gain(moles)$
  $cur\_clus \leftarrow creat\_cluster(seed)$
  while $have\_spcace(cur\_clus)$ **do**
    $next\_mol \leftarrow get\_highest\_gain(moles)$
    if $next\_mol = \emptyset$ **then**
      $break$
    end if
    $cur\_clus \leftarrow add\_mole(moles, cur\_clus)$
  end while
end while
return $packed\_netlist$

4.1. Primitive Classification

In this paper, the algorithm considers the proportion of DSPs, RAMs, and adders in the circuit as the quantitative rule for special primitives. The formula used for this purpose is as follows:

$$SP = \begin{cases} \{DSPs, RAMs\}, & \dfrac{num(DSPs) + num(RAMs) + num(adders)}{num(total)} < thre \\ \emptyset, & \text{otherwise} \end{cases} \tag{1}$$

where SP is a set of special primitives, $num(DSPs)$ is the number of DSPs in the netlist, $num(RAMs)$ is the number of RAMs in the netlist, $num(adders)$ is the number of adders in the netlist, $num(total)$ is the total number of primitives in the netlist, and $thre$ is the threshold.

4.2. Seed Selection

The selection of the seed impacts the order in which different parts of the netlist will be clustered. In VTR8.0, the criteria of the selection of seed are determined by the number of primitives in the molecule, the number of molecular pins, and the delay information of the molecule. When the molecules are packed, the packer can determine the type of connectivity between the pins of the tile and the network. In this paper we seek to raise the priority of molecules with special primitives as seeds through the use of *seed_gain* as the criterion for seed selection. The molecule with large *seed_gain* is preferentially selected as the seed. The model of *seed_gain* is as follows:

$$\begin{aligned} seed\_gain = {} & w1 \times num(in) + w2 \times num(used\_in) \\ & + w3 \times num(block) + w4 \times crit + w5 \times i(SP) \end{aligned} \tag{2}$$

where $num(used\_in)$ is the normalized number of input pins used, $num(in)$ is the normalized number of input pins, and $num(block)$ is the normalized number of primitives in the molecule, $crit$ is the delay of the primitive pins, $i(SP)$ is used to determine whether the current primitive is a special primitive, $w1$~$w5$ is the weight.

4.3. Molecule Selection

Once a seed molecule has been chosen and a new tile is opened, the packer begins searching for unclustered molecules to add. The next molecule to be grouped into the current tile is determined by attraction functions, which are influenced by the connectivity between the molecule and the current tile.

For the attraction function of direct connectivity, our packing algorithm adopts the same attraction function as VTR8.0.

$$Aff(p, B) = (1 - \beta) \times c\_gain(p, B) + \beta \times t\_gain(p, B) \tag{3}$$

where $c\_gain(p, B)$ is the connection benefit of the molecule p to the tile B, and $t\_gain(p, B)$ is the Criticality of the network connection between p and B. $c\_gain(p, B)$ formula is as follows.

$$c\_gain(p, B) = \frac{(1 - \alpha) \times nets(p, B) + \alpha \times con(p, B)}{num\_pins(p)} \tag{4}$$

where $nets(p, B)$ is the number of shared nodes between the molecule p and the tile B, and $con(p, B)$ and the pins of p are closely related to the connection relationship of B; the formula is as follows.

$$con(p, B) = \frac{1}{ext(p, B) + packed(p) + 1} \tag{5}$$

where $ext(p, B)$ is the number of pins of p that are not connected to tile B, and $packed(p)$ is the number of pins of p that are connected to the packed molecule.

Our packing algorithm considers two types of indirect connectivity: indirect connectivity through special tiles and indirect connectivity through normal tiles. The packer should preferentially absorb molecules that are indirectly connected to the current tile via short-distance pins. Therefore, the molecules that are indirectly connected to the current tile through the special tile are divided into three categories. The first type is molecules that are indirectly connected through pins on the same side and at the same location, as shown in Figure 4a. The second type is molecules that are indirectly connected on the same side but at different locations, as shown in Figure 4b. The third type is molecules indirectly connected by pins on different sides, as shown in Figure 4c. During packing, the packer prioritizes first-type molecules into the same tile, and then considers second-type molecules, and finally third-type molecules. The cost of the attractive function is as follows.

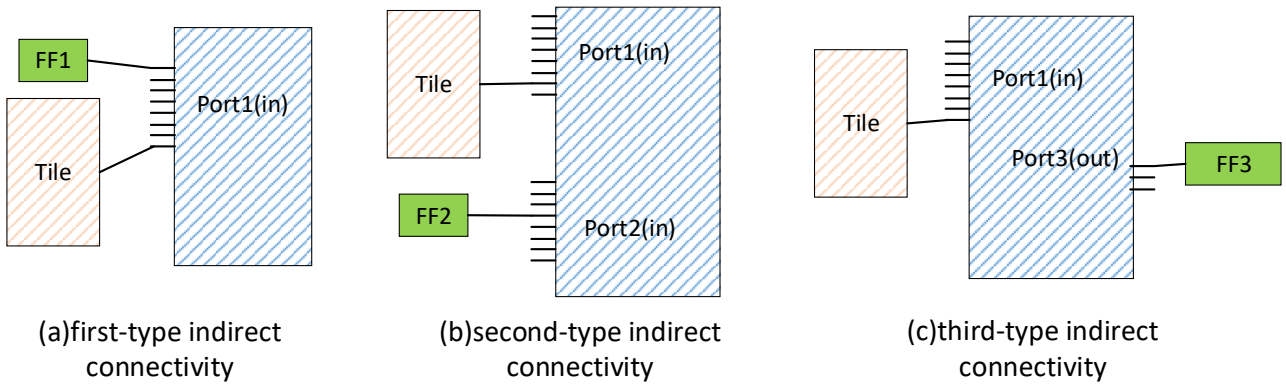

Figure 4. Three models of indirect connectivity.

$$Aff(p, B) = \sum_{T_i \in Tiles} ind\_gain(p, B, T_i) \tag{6}$$

Among them, *Tiles* is the set of tiles around tile B. $ind\_gain(p, B, T_i)$ is the attraction of p indirectly connected to molecule tile B through Ti.

$$ind\_gain(p, B, T_i) = \begin{cases} w_{port} \times num_{port} + w_{dir} \times num_{dir} + w_{rev} \times num_{rev}, T_i \in SP \\ w_{nor} \times n_{nor}, \quad \text{otherwise} \end{cases} \tag{7}$$

Among them, w_{port} is the weight of the first type of molecules, w_{dir} is the weight of the second type of molecules, w_{rev} is the weight of the third type of molecules, num_{port}, num_{dir} and num_{rev} are the connection times of the three indirect connection molecules, w_{nor} is the weight of molecules indirectly connected through normal tiles, and n_{nor} is the number of times primitives are indirectly connected through normal tiles. The formula for w_{dir} is as follows:

$$w_{dir} = \frac{w_{port}}{n_{dir}} \tag{8}$$

Among them, n_{dir} is a positive integer.

If both directly connected and indirectly connected molecules are packed, and the tile to be packed does not meet the constraints, then the molecules connected by the high fan-out network are selected for clustering.

5. Experimental Results

The experiments are performed on a workstation with an AMD EPYC 7302P (16 cores, 3 GHz) with 64 G of memory. The FPGA architecture used in this paper is the k6FracN10LB_mem20K_complexDSP_customSB_22nm architecture provided by VTR. Its blocks are Agilex-like, but the routing architecture is Stratix-IV-like [18]. The circuits used in this paper are from the Koios benchmark [19]. The Koios benchmark contains 20 deep learning-related circuits, all of which are medium- or large-size circuits, suitable for architecture research and EDA algorithm research. This paper runs Koios with a channel width of 200 for medium-size circuits and 300 for large-size circuits.

Table 3 shows some parameters and parameter values used in this experiment, and the values are obtained through verification in VTR8.0.

Table 3. Values of parameters.

Parameters	Value
$w1$	0.5
$w2$	0.2
$w3$	0.2
$w4$	0.1
w_{nor}	0.003
w_{rev}	0.001
α	0.6
β	0.2

Figure 5 shows the impact of the proportion of DSPs, RAMs and adder on the critical path delay in the Koios benchmark. It can be seen from Figure 5 that when the proportion of DSPs, RAMs and adder in the circuit exceeds 20%, the packer uses DSPs and RAMs as special primitives to cluster. This clustering results in a higher possibility of increasing the critical path delay, as shown in the red zone on the left in Figure 5. If the proportion of DSP in the circuit is less than 20%, eleven out of twelve benchmark circuits have successfully reduced critical path delays, as shown in the blue zone on the right in Figure 5. So the algorithm in this paper sets the *thre* as 20%.

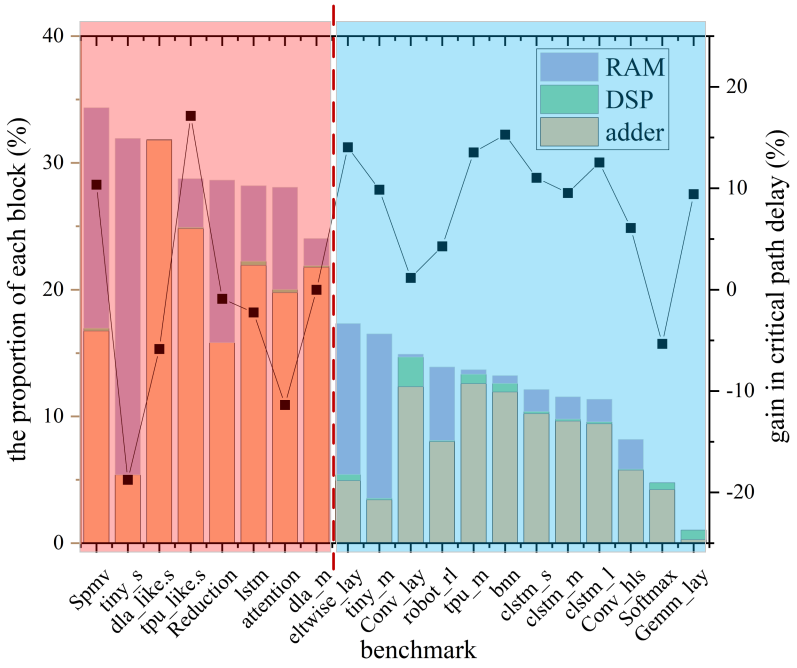

Figure 5. The influence of the proportion of DSPs, RAMs and adder in the circuit on the critical path delay. The histogram is the proportion of DSPs, RAMs and adder in the circuit, corresponding to the coordinates on the left. The line graph is the optimization rate of critical path delay, corresponding to the coordinates on the right.

In the seed selection stage, the effect of different w5 in the *seed_gain* on the algorithm of this paper was tested. For testing purposes, the medium circuits of the Koios benchmark that meet the special primitive conditions are used as the test circuits, and the results of these tests are shown in Figure 6. From the figure, it can be seen that the critical path delay is optimized best when w5 is 0.1. In this paper, we set w5 to 0.1.

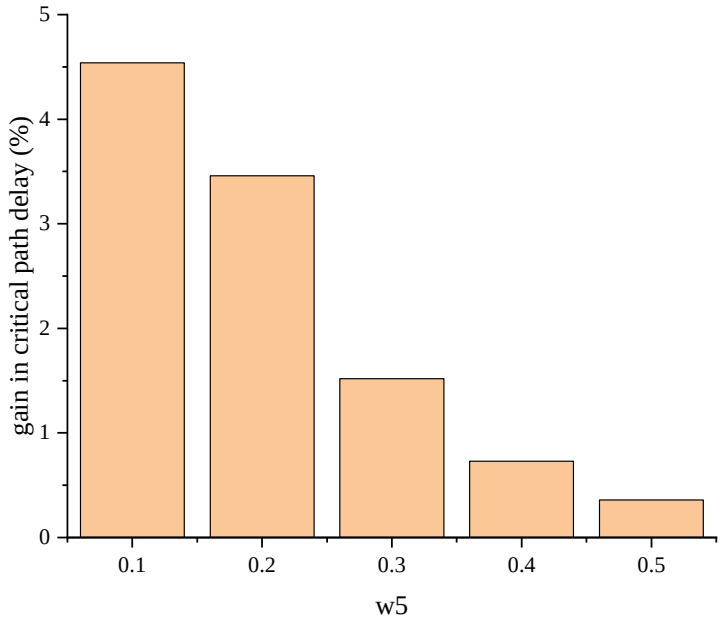

Figure 6. Effect of variation of w5 in seed selection stage on critical path delay.

In the molecule selection stage, the attraction of directly connected molecules should be greater than that of indirectly connected molecules. The attraction functions of indirectly connected molecules are shown in Equations (7) and (8). In indirect connection, the

attraction should meet the condition w_{port}>$max\{w_{dir}, w_{nor}\}$ and $min\{w_{dir}, w_{nor}\}$> w_{rev}. With the parameters of Table 3, the attraction of direct connection molecules is greater than 0.1. Therefore, this paper sets w_{port} as 0.009, 0.03, 0.06 and 0.09, respectively, and observes the impact of changes in w_{port} on the critical path delay. As can be seen from Figure 7, in the Koios benchmark, the circuits that meet the special primitive conditions achieve better results when w_{port} is 0.03 for critical path delay.

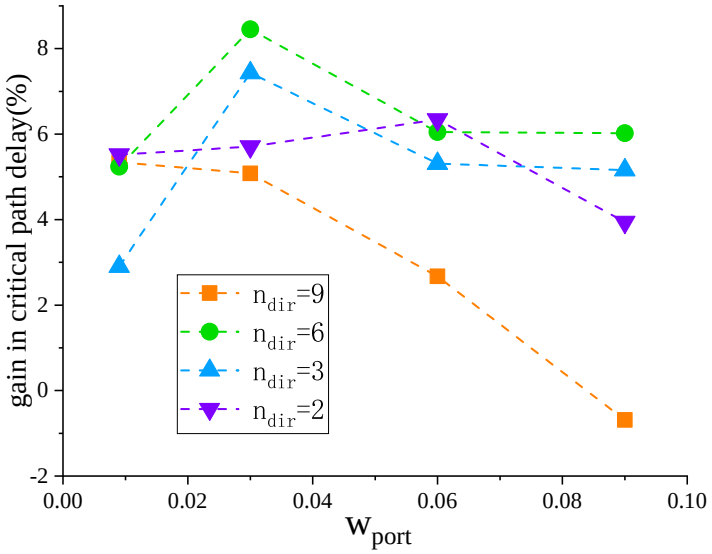

Figure 7. The impact of changes in w_{port} on critical path delays.

When dealing with molecules indirectly connected through special tiles, the first type of indirect connectivity should exhibit greater attraction than the second type. However, if the attraction of the second type of indirectly connected molecules is too small, the packer may end up neglecting these molecules and instead absorb those that are indirectly connected through another special tile. In this paper, n_{dir} is set to 2, 3, 6, and 9 respectively, and the critical path delay changes are observed. It can be seen from Figure 8 that in the Koios benchmark, the circuits that meet the special primitive conditions achieve better results in critical path delay when n_{dir} is 6.

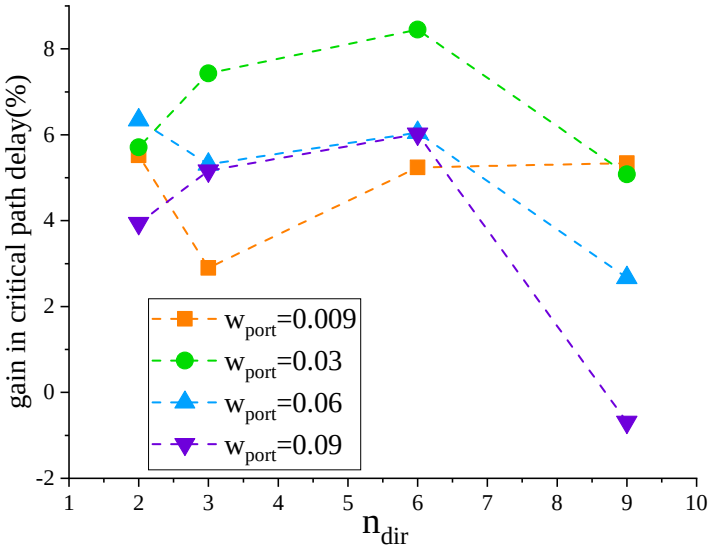

Figure 8. The impact of changes in n_{dir} on critical path delay.

From Figures 9 and 10, it can be inferred that resource consumption is scarcely affected by variations in w_{port} and n_{dir}. The variation range is within 0.24%. After rebalancing resource consumption and critical path delay, this paper sets w_{port} to 0.03 and n_{dir} to 6.

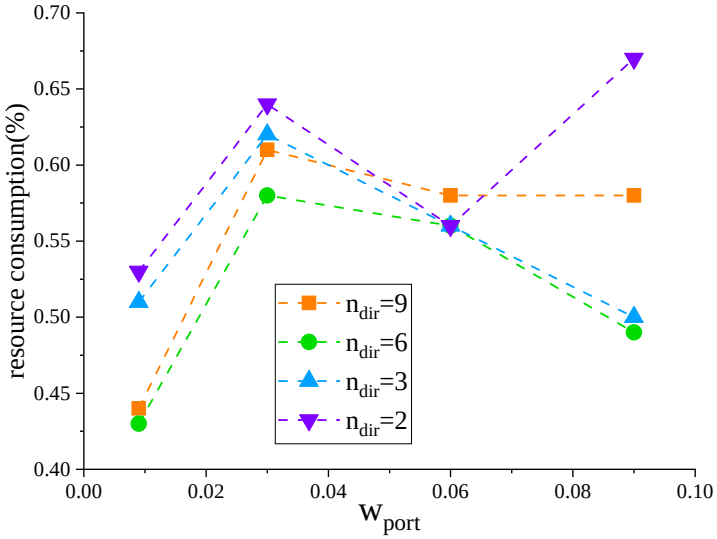

Figure 9. The impact of changes in w_{port} on resource consumption.

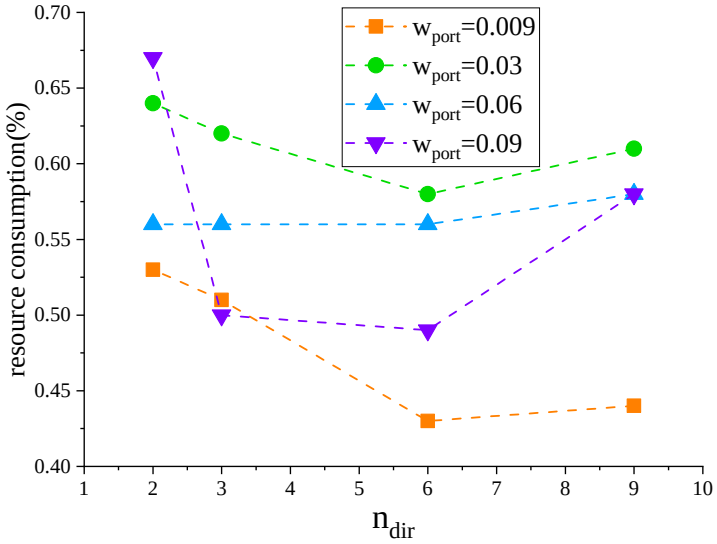

Figure 10. The impact of changes in n_{dir} on resource consumption.

Our proposed algorithm contrasts with the packing algorithm in VTR8.0 through a modified packing rule for indirectly connected molecules. This modification results in a rise in computational demand and extends the algorithm's runtime. However, while fewer options during the packing process translate into a slight increase in resource consumption, the algorithm's refinement leads to a shortened wirelength for nets around DSPs and RAMs. The end result is a reduction in the critical path delay.

As can be seen from Table 4, compared with VTR8.0, our packing reduces the critical path delay by 8.45% on average at the cost of a 0.58% increase in resource consumption and a 7.55% increase in runtime. Among the circuits in the Koios benchmark suite that meet the special primitive criteria, eleven out of twelve have successfully reduced critical path delays. For circuits that do not meet the conditions of special primitives, the algorithm in this paper does not divide the primitives around DSP and RAM, so resource consumption and critical path delay are the same as VTR8.0.

Table 4. The comparison between the packing method of the present invention and the results of VTR8.0 after placement and routing.

Circuits	Blocks		Runtime		Crit Path Delay	
	VTR8.0	Ours	VTR8.0	Ours	VTR8.0	Ours
Eltwise_layer.v	478	478	6.48	6.08	6.69	5.75
Conv_layer.v	1323	1326	42.47	43.24	6.9	6.82
Softmax.v	570	570	10.13	10.44	8.62	9.08
Gemm_layer.v	2217	2217	17.54	17.73	6.05	5.48
Robot_rl.v	1438	1443	19.19	19.46	12.66	12.12
Conv_layer_hls.v	1749	1748	115.18	137.12	6.9	6.48
bnn.v	1409	1452	378.2	575.76	7.98	6.76
Tiny_darknet.med.v	18,315	18,380	3164.24	3890.84	13.68	12.33
Tpu_like.medium.v	5344	5436	787.39	682.35	12.34	10.67
Clstm_like.small.v	10,078	10,130	308.05	326.07	8.06	7.17
Clstm_like.med.v	19,087	19,172	617.4	629.25	9.22	8.34
Clstm_like.large.v	28,118	28,232	985.77	992.84	10.51	9.19
Average	1.000	1.0058	1.00	1.0755	1.00	0.9155
improve	−0.58%		−7.55%		8.45%	

6. Conclusions and Future Work

This paper proposes a packing algorithm for FPGA improved by indirect connection, which refines the packing guideline in two aspects. (1) It proposes the quantitive rules of the special primitives by the proportion of DSPs, RAMs and adders. (2) It optimizes the traditional seed-based packing methods with special primitives, such as the modified criteria for seed and molecule selection. For circuits with special primitives, the proposed packing algorithm reduces the critical path delay by an average of 8.45% compared to VTR8.0. For circuits without special primitives, the critical path delay of our packing is the same as that of VTR8.0.

For future work, we will primarily aim at utilizing parallel computing methods to curtail the runtime of our proposed algorithm. We also plan to investigate the feasibility of porting these algorithms to commercial FPGAs.

Author Contributions: Conceptualization, L.Y. and B.G.; methodology, L.Y. and B.G.; software, B.G. and L.B.; validation, L.Y. and B.G.; formal analysis, B.G.; writing—original draft preparation, B.G.; writing—review and editing, L.Y. and T.Z.; supervision, L.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The benchmark and architecture files used in this paper are both from VTR (Verilog-to-Routing) https://github.com/verilog-to-routing/vtr-verilog-to-routing.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ghani, A.; Hodeify, R.; See, C.H.; Keates, S.; Lee, D.J.; Bouridane, A. Computer Vision-Based Kidney's (HK-2) Damaged Cells Classification with Reconfigurable Hardware Accelerator (FPGA). *Electronics* **2022**, *11*, 4234. https://doi.org/10.3390/electronics11244234.
2. Zhang, N.; Wei, X.; Chen, H.; Liu, W. FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection. *Electronics* **2021**, *10*, 282. https://doi.org/10.3390/electronics10030282.
3. Han, Z.; Jiang, J.; Qiao, L.; Dou, Y.; Xu, J.; Kan, Z. Accelerating Event Detection with DGCNN and FPGAs. *Electronics* **2020**, *9*, 1666. https://doi.org/10.3390/electronics9101666.
4. Betz, V.; Rose, J.; Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*; Springer: Boston, MA, USA, 1999; pp. 11–18.
5. 7 Series FPGAs Configurable Logic Block User Guide. Available online: https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB (accessed on 9 June 2023).

6. Murray, K.E.; Petelin, O.; Zhong, S.; Wang, J.M.; Eldafrawy, M.; Legault, J.P.; Sha, E.; Graham, A.G.; Wu, J.; Walker, M.J.P.; et al. VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling. *ACM Trans. Reconfig. Technol. Syst.* **2020**, *13*, 1936–7406. https://doi.org/10.1145/3388617.
7. Betz, V.; Rose, J. Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size. In Proceedings of the Custom Integrated Circuits Conference (CICC 97), Santa Clara, CA, USA, 5–8 May 1997; pp. 551–554. https://doi.org/10.1109/CICC.1997.606687.
8. Marquardt, A.S.; Betz, V.; Rose, J. Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density. In Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 1999; pp. 37–46. https://doi.org/10.1145/296399.296426.
9. Chen, D.T.; Vorwerk, K.; Kennings, A. Improving Timing-Driven FPGA Packing with Physical Information. In Proceedings of the 2007 International Conference on Field Programmable Logic and Applications, Amsterdam, The Netherlands, 27–29 August 2007; pp. 117–123. https://doi.org/10.1109/FPL.2007.4380635.
10. Luu, J.; Anderson, J.H.; Rose, J.S. Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect. In Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 27 February–1 March 2011; pp. 227–236. https://doi.org/10.1145/1950413.1950457.
11. Haroldsen, T.; Nelson, B.; Hutchings, B. Packing a modern Xilinx FPGA using RapidSmith. In Proceedings of the 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 30 November–2 December 2016; pp. 1–6. https://doi.org/10.1109/ReConFig.2016.7857180.
12. Chen, Q.; Shen, M.; Xiao, N. DP-Pack: Distributed Parallel Packing for FPGAs. In Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT), Naha, Japan, 10–14 December 2018; pp. 282–285. https://doi.org/10.1109/FPT.2018.00054.
13. Maidee, P.; Ababei, C.; Bazargan, K. Timing-driven partitioning-based placement for island style FPGAs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2005**, *24*, 395–406. https://doi.org/10.1109/TCAD.2004.842812.
14. Feng, W. K-way partitioning based packing for FPGA logic blocks without input bandwidth constraint. In Proceedings of the 2012 International Conference on Field-Programmable Technology, Seoul, Republic of Korea, 10–12 December 2012; pp. 8–15. https://doi.org/10.1109/FPT.2012.6412103.
15. Feng, W.; Greene, J.; Vorwerk, K.; Pevzner, V.; Kundu, A. Rent's Rule Based FPGA Packing for Routability Optimization. In Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 26–28 February 2014; pp. 31–34. https://doi.org/10.1145/2554688.2554763.
16. Vercruyce, D.; Vansteenkiste, E.; Stroobandt, D. Runtime-quality tradeoff in partitioning based multithreaded packing. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–9. https://doi.org/10.1109/FPL.2016.7577300.
17. Arora, A.; Wei, Z.; John, L.K. Hamamu: Specializing FPGAs for ML Applications by Adding Hard Matrix Multiplier Blocks. In Proceedings of the 2020 IEEE 31st International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Manchester, UK, 6–8 July 2020; pp. 53–60. https://doi.org/10.1109/ASAP49362.2020.00018.
18. AN 519: Stratix IV Design Guidelines. Available online: https://www.intel.com/programmable/technical-pdfs/654680.pdf (accessed on 9 June 2023).
19. Arora, A.; Boutros, A.; Rauch, D.; Rajen, A.; Borda, A.; Damghani, S.A.; Mehta, S.; Kate, S.; Patel, P.; Kent, K.B.; et al. Koios: A Deep Learning Benchmark Suite for FPGA Architecture and CAD Research. In Proceedings of the 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, 30 August–3 September 2021; pp. 355–362. https://doi.org/10.1109/FPL53798.2021.00068.