A Placement and Routing Algorithm for a Reconfigurable 1-bit Processor Array

Mitsuru Tomono[1], Masaki Nakanishi[1], Shigeru Yamashita[1], and Yasuhiko Nakashima[1]

Nara Institute of Science and Technology, Ikoma Nara 630-0192, Japan,
{mitsu-to,m-naka,ger,nakashim}@is.naist.jp,
WWW home page: http://www.naist.jp/index_en.html

Abstract. In recent decades, reconfigurable devices have been extensively researched to improve computer system performance, with a reconfigurable 1-bit processor array (1-bit RPA) having been proposed as an outcome of this research. Its architecture is mainly composed of processor elements (PEs) that have bit-serial data paths. The interconnection networks among PEs are determined according to certain parameters, and its structure makes flexible mapping of applications possible. However, due to its unique wiring structure, a dedicated method is required to place and route target applications for the architecture. In this paper, we present an efficient and effective placement and routing algorithm for a 1-bit RPA. Preliminary experimental results using the algorithm are promising.

1 Introduction

Recent advances in reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices [1] has promoted their use in many computing systems to improve their performance. In particular, FPGAs are widely used due to the flexibility of reconfiguring their functionality as well as their enhanced development environment.

However, some flaws of FPGAs limit the fields to which they can be applied. First, since wiring areas are dominant in chip areas of FPGAs, causing their logic density to decrease, an implementation of a target application might not fit into their logic areas. Second, the clock frequency of FPGAs is relatively low due to their fine-grained structure. As a result, desired performance cannot be obtained.

To overcome these drawbacks, some devices have been proposed. For example, Ohta et al. proposed a FPGA architecture with bit-serial pipeline data paths [2]. However, their architecture's base is still similar to conventional FPGAs. Among devices proposed, we consider a reconfigurable 1-bit processor array (1-bit RPA) [3] to be one of the most significant results. A 1-bit RPA has the structure of a bit-serial data path, but its features are reduced wiring area, flexible routability, high logic density, and high clock frequency.

A well developed design environment is an essential factor for a new type of architecture, since it is impossible for a large-scale design to be implemented by hand. In this field, commercial FPGAs have arisen in an advanced development environment, and many effective methods have been proposed such as versatile place and route [4]. On the other hand, there has been no design environment for

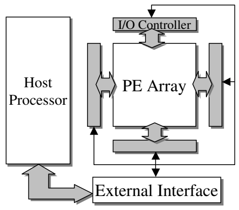

Fig. 1. Block diagram of a 1-bit RPA

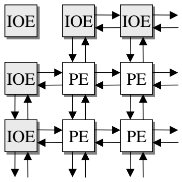

Fig. 2. Structure of short wires

a 1-bit RPA yet proposed. However, we cannot simply apply FPGAs' existing methods to a 1-bit RPA since the structures of 1-bit RPAs and FPGAs are substantially different. In the architecture, a processor element (PE) can be used to bridge wires for more flexible mapping, making placement and routing integrated and complicated. Therefore, we need an efficient way to map target applications to the architecture.

In this paper, we propose an efficient and effective placement and routing algorithm for a reconfigurable 1-bit processor array. A 1-bit RPA features a unique wiring structure that makes possible flexible mapping of applications. Our placement and routing algorithm achieves compact implementation through various optimization steps by taking advantage of the 1-bit RPA's characteristic routing structure.

2 Architecture of a Reconfigurable 1-bit Processor Array

In this section, we present a brief explanation of the 1-bit RPA's architecture. Please refer to [3] for more details. As mentioned earlier, a 1-bit RPA has the features of a bit-serial data path and a unique wiring network. The wiring areas have been reduced compared to conventional FPGAs due to its structure, and its performance is superior, demonstrated by an example application such as a discrete cosine transform [3]. However, owing to these features, the mapping process of a 1-bit RPA is different to that of conventional methods.

Figure 1 shows a block diagram of a 1-bit RPA. In the figure, a processor element array is the main component of the architecture and I/O controllers are functional units for controlling input and output data from and to external components. The processor element array is composed of processor elements and I/O elements (IOEs) that are primal processing units and controllers to feed and receive data to and from PEs, respectively. PEs are capable of basic arithmetic operations such as addition, subtraction, and shift. IOEs are arranged at the periphery of PEs.

Next, we describe the details of the 1-bit RPA's interconnection network. The architecture's wiring structure has two types of wiring resources (short wires and long wires), and PEs can be used to bridge wires. We use short wires to transfer data between neighboring PEs, with Fig. 2 showing the structure of short wires. A PE has four input and output short wires, respectively. Long wires are used to transfer data among distant PEs. The structure of long wires is depicted in Fig. 3. The architecture also includes two static parameters, *distance* and *step*, which determine the configuration of long wires and are set before fabrication. Distance denotes the farthest PE that a PE can access through a long wire,

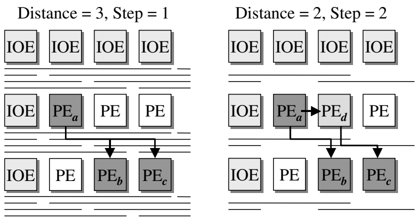

Fig. 3. Structure of a long wire

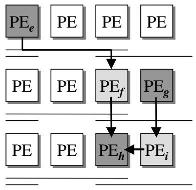

Fig. 4. Insertion of PEs for delay

and step represents intervals of long wires. In Fig. 3, the left-hand figure depicts the case where distance is 3 and step is 1. This means a PE can access PEs within a distance of 3. Here, only wires in rows are shown for clarity. The figure shows that PE_a has access to PE_b and PE_c using a long wire. The right side of the figure shows an example of the case where distance is 2 and step is 2. In the right-hand figure, PE_a uses PE_d as a bridge to connect to PE_c due to the shortness of long wires. As these examples indicate, the parameters of long wires affect the mapping flexibility.

Before proceeding further, we must describe one more feature of the architecture. Because a 1-bit RPA employs the bit-serial data path structure, delays may occur due to certain operations such as multiplication or shift. Using a PE for bridging wires also leads to delay. Here, a delay means extra clock cycles. Therefore, it is necessary to adjust the data timings. We depict a demonstrative example of this in Fig. 4. In the figure, PE_g can directly access PE_h using a long wire located immediately above PE_h and PE_i. However, PE_h's input from PE_e has a delay through PE_f. Therefore, it is necessary for PE_h's input from PE_g to have the same number of delays and, here, it has a delay through PE_i. These specific structures of the architecture require us to research and develop a dedicated placement and routing method to map applications.

3 Placement and Routing Algorithm for a Reconfigurable 1-bit Processor Array

In this section we explain in detail the processes of our placement and routing algorithm. We start with an overview of the mapping processes.

3.1 Overview of Mapping Algorithm

The process of our mapping algorithm comprises three steps: initial placement, initial routing, and iterative optimizations. We employ a control data flow graph (CDFG) as an input of our algorithm. An output is placement and routing information for the 1-bit RPA's configuration data. In this strategy, we purposely position empty PEs that are used to route nodes. The number of empty PEs is dealt with as a parameter that can be changed according to need. This scheme is one of the techniques in our placement and routing algorithm.

The first step of the algorithm is the initial placement stage. Before proceeding to the placement stage, the number of empty PEs placed between nodes is determined in the initial setting stage. Initial placement with no empty cells

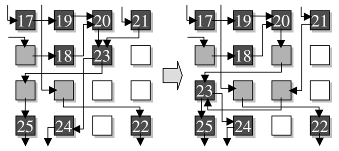

Fig. 5. Adjustment of CPEs

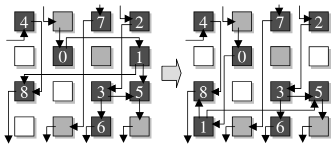

Fig. 6. Utilizing empty PEs

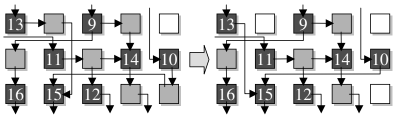

Fig. 7. Tuning input delays

	CPEs	Areas
Initial	44 $CPEs$	11×17
Optimized	27 $CPEs$	7×13

Table 1. Preliminary experimental results

might lead to invalid mapping outputs; therefore, we raise the possibility of producing valid mapping results by placing these empty PEs between each node.

In the placement step, we have three strategies: P_0, Square; P_1, TGFF; and P_2, Manhattan distance (MD). The first strategy, (P_0), is the placement whereby nodes are placed in the square-shaped form. In the second one, (P_1), nodes of a CDFG are arranged in the same form as an output of the task graph for free (TGFF) [5], in which no backward flows are structured. The third one, (P_2), is the scheme in which the placement is arranged to minimize the average MD between two connected nodes.

The next step is the initial routing stage. In this stage, we also have three strategies: R_0, Degree; R_1, MD; and R_2, Degree + MD. In the first strategy (R_0), the routing begins with the node that has the largest input and output degrees. The routing of the second one (R_1) is done in descending order from the node that has the longest MD. The last one, (R_2), is the combined method of R_0 and R_1. The routing of R_2 starts with the node whose sum of degree and MD is the largest.

In every routing step, our algorithm searches for a path along which the number of PEs used for connection (CPEs) is the smallest and adopts it for wiring two connected nodes. In the initial routing stage, empty PEs that were placed between nodes in the previous stage are utilized as CPEs, which makes possible various routings among nodes. If there are multiple possible paths found, an arbitrary one is selected, but if no possible path is found, a backtrack to the previous step occurs and another possible path is chosen.

In the initial placement and routing stage, our algorithm employs one of the above-described strategies, and the strategy selection is performed in the initial setting stage before entering optimization processes. Since strategies of these stages are naive ones, we need to improve the placement quality by the following optimization stages described in the next section.

3.2 Optimization Schemes of Our Engine

Here, we present optimization schemes of our mapping algorithm, which is composed of three steps. The following cost function is evaluated in each step.

$$Cost \ = \ \alpha \cdot \#CPEs \ + \ \beta \cdot \sum_i center(P_i) \qquad (1)$$

The first term represents the total number of CPEs, and the second represents the sum of $node_i$'s MD to the central point; α and β are user-defined parameters. The central point is the center-most position of the most congested area, which is determined before entering the optimization steps. If this cost function becomes smaller, each optimization is performed. Iterating the following three optimization steps an arbitrary number of times, our placement and routing engine attempts to improve the quality of the mapping result.

Optimization 1: *Adjustment of CPEs*
In this optimization step, our algorithm attempts to perform swapping between each node and CPEs that are connected to the node. If it is possible to re-route the nodes and CPEs and the cost function decreases, this optimization is performed. We illustrate this step using an example. In the following figures, the architectures on the left-hand side show the unoptimized state while those on the right-hand side show the optimized state. Dark gray, light gray, and white squares respectively denote occupied, connection, and empty PEs. We assume that the central point exists in the lower part of each figure. For illustrative purposes, only the intended figures are shown.

Let us focus on *node* 23 and a CPE that is placed between *node* 23 and *node* 25 in the left-side of Fig. 5. If *node* 23 is moved to where the CPE is placed and can be re-routed there, the cost function is evaluated. If the cost becomes smaller, this optimization is performed. Although the number of CPEs increases in this example, *node* 23 comes closer to the central point. Therefore, it depends on the parameters whether this optimization is performed.

Optimization 2: *Utilizing empty PEs*
The second optimization is an approach to utilizing empty PEs. In this step, the algorithm investigates whether each node can be moved to empty PEs. If it is possible to displace a node and, as a result, the cost decreases, this optimization step is performed. In the beginning, our algorithm examines the outer boundary nodes to reduce the mapping area's size. Next, the remaining nodes are examined. In Fig. 6, we can move *node* 1 to the bottom-left PEs, leading to *node* 1's proximity to the central point.

Optimization 3: *Tuning input delays*
Adjusting input delays of nodes is the last optimization step. As we mentioned in the previous section, PEs can be employed to relay a signal. However, if one of the node's inputs goes through CPEs, the node's other inputs also need to be routed through CPEs due to the timing of data. The naive scheme of the initial stages leads to using redundant CPEs. In this step, therefore, our method investigates whether CPEs for input delays can be removed. Whereas the previous three methods only focused on one-to-one wiring of PEs, this scheme deals with one-to-many connections of PEs. Figure 7 shows a case in which we can eliminate CPEs of *node* 15's inputs through *nodes* 10 and 13. As a result, the number of CPEs decreases and the input delays of *node* 15 are reduced to 0.

4 Experimental Results

We use CDFGs obtained by TGFF to evaluate the results. Figure 8 shows an example of TGFF's output. We apply this CDFG to show the effectiveness of our algorithm. Figure 9 displays the initial placement and routing result, in which the numbered dark gray cells are occupied PEs as CDFG's nodes and light gray cells are CPEs. Wires and IOEs are abbreviated in this figure and that following it. The parameters of distance and step are set to 4 and 1, respectively. Figure 10

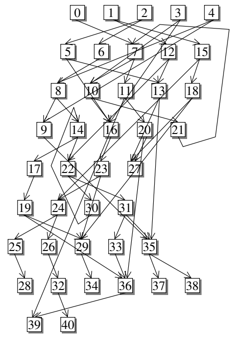

Fig. 8. CDFG example

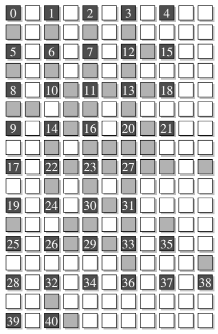

Fig. 9. Initial

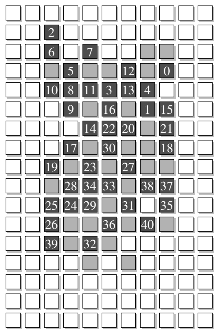

Fig. 10. Optimized

shows the result in which we manually applied our algorithm to the initial state using the above-mentioned steps. We employed the strategies P_1 at the initial placement stage and R_2 at the initial routing stage. The number of iterations and empty PEs is 1, respectively.

Table 1 presents the experimental results, where it is clear that the size of the mapping output has shrunk to 48.6% from that of the initial state, and the number of CPEs has decreased to 61.3% of the initial state. Although this experiment is a preliminary one and conducted manually, it seems reasonable to conclude that the results are promising.

5 Conclusions and Future Work

In this paper, we proposed an efficient and effective placement and routing algorithm for a reconfigurable 1-bit processor array. Since we cannot simply utilize the existing placement and routing methods due to the 1-bit RPA's unique architecture, a dedicated and effective way to map target applications automatically is needed. In our mapping algorithm, initial placement and routing stages employ a naive method, but this method attempts to achieve compact mapping of applications through various iterative optimization steps.

As future work, we will conduct more detailed experiments using various parameter settings. Through experiments, attempts will be made to improve our placement and routing engine and to derive the optimal parameter settings for wide-ranging applications.

Concurrently with this research, we are investigating a hardware-software codesign method for a 1-bit RPA. As further future work, we will integrate our placement and routing algorithm and hardware-software codesign method, with the aim of achieving a comprehensive development environment for the architecture.

References

1. S. Brown and J. Rose: Architecture of FPGAs and CPLDs: A Tutorial. IEEE Design and Test of Computers, Vol. 13, No. 2, pp. 42–57, 1996.

2. A. Ohta, T. Isshiki, and H. Kunieda: New FPGA architecture for bit-serial pipeline datapath. In Proc. of IEEE Symp. on FPGAs for Custom Computing Machines, Napa Valley, CA, Apr. 1998, pp. 58–67.
3. N. Nakai, M. Nakanishi, S. Yamashita, and K. Watanabe: Reconfigurable 1-Bit Processor Array with Reduced Wiring Area. In Proc. of International Conf. on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, NV, June, 2005, pp. 225–231.
4. V. Betz and J. Rose: VPR: A New Packing, Placement and Routing Tool for FPGA Research. In Proc. of the 7th International Workshop on Field-Programmable Logic and Applications, London, UK, 1997, pp. 213–222.
5. R. P. Dick, D. L. Rhodes, and W. Wolf: TGFF: task graphs for free. In Proc. of the 6th international workshop on Hardware/software codesign, Seattle, WA, March, 1998, pp. 97–101.