PAPER Special Section on Reconfigurable Systems

A Physical Design Method for a New Memory-Based Reconfigurable Architecture without Switch Blocks

Masatoshi NAKAMURA[†a)], ***Nonmember***, **Masato INAGI**[†b)], **Kazuya TANIGAWA**[†], **Tetsuo HIRONAKA**[†], **Masayuki SATO**[††], ***Members***, ***and*** **Takashi ISHIGURO**[††], ***Nonmember***

SUMMARY In this paper, we propose a placement and routing method for a new memory-based programmable logic device (MPLD) and confirm its capability by placing and routing benchmark circuits. An MPLD consists of multiple-output look-up tables (MLUTs) that can be used as logic and/or routing elements, whereas field programmable gate arrays (FPGAs) consist of LUTs (logic elements) and switch blocks (routing elements). MPLDs contain logic circuits more efficiently than FPGAs because of their flexibility and area efficiency. However, directly applying the existing placement and routing algorithms of FPGAs to MPLDs overcrowds the placed logic cells and causes a shortage of routing domains between logic cells. Our simulated annealing-based method considers the detailed wire congestion and nearness between logic cells based on the cost function and reserves the area for routing. In the experiments, our method reduced wire congestion and successfully placed and routed 27 out of 31 circuits, 13 of which could not be placed or routed using the versatile place and route tool (VPR), a well-known method for FPGAs.

key words: *reconfigurable device, physical design, placement, routing, MPLD, FPGA, EDA*

1. Introduction

In recent years, programmable logic devices (PLDs), such as field programmable gate arrays (FPGAs) [1], have been used in various fields, such as prototyping, networking, and high-performance computing. An FPGA consists of look-up tables (LUTs) as logic elements, and switch blocks (SBs) (including connection blocks) as routing elements (Fig. 1 (a)). LUTs and SBs are connected with wires. To realize a circuit on an FPGA, a LUT functions as a logic cell (*e.g.*, NOT, AND, OR, and more complex logic functions) and a SB determines the connections among LUTs (logic cells). To our knowledge, however, the area required for routing can reach about 90% of the total area required for FPGAs (*e.g.*, [2]), and this degrades the area efficiency. In other words, in FPGAs, the number of logic elements per area is small. Thus, we have proposed a new reconfigurable device [3] composed of *multiple-output look-up tables* (*MLUTs*) that can be used as either logic or routing elements. Hereafter, we refer to this device as a *memory-based programmable logic device* (*MPLD**). Figure 1 (b) illustrates the basic structure of an MPLD. By this structure, an MPLD can flexibly control the ratio of logic and routing elements (*i.e.*, MLUTs function as both logic and routing elements), both globally and locally. We can therefore realize a circuit more efficiently on an MPLD than on standard FPGAs. In particular, regularly structured circuits (*e.g.*, adders, multipliers, multiplexers *etc*.) can efficiently be mapped to MPLDs as IP modules adjusted to MPLDs, thereby increasing the ratio of logic elements. Figure 2 illustrates an example of a 4-bit multiplier on an MPLD. Moreover, it has been estimated that multipliers are more efficiently mapped on MPLDs than on standard FPGAs [4].

To map a circuit that involves typical random logic components on an MPLD, it is necessary to use the the same procedure used for FPGAs. In other words, it is necessary to first determine the logic design of the desired circuit in HDL, synthesize a netlist of the circuit (logic design and synthesis), and then perform placement and routing (physical design) in order to generate configuration data for MLUTs. During this process, placement and routing strongly depend on the desired device. In addition, because IP modules mapped on an MPLD are efficient as shown in Fig. 2, the placement and routing of random logic circuits is essential to efficiently map an entire circuit on an MPLD. In this paper, we therefore focus on the placement and routing of random logic circuits, and confirm an MPLD's ability to realize random logic circuits.

Placement and routing techniques of FPGAs have been studied for more than two decades. The versatile place and

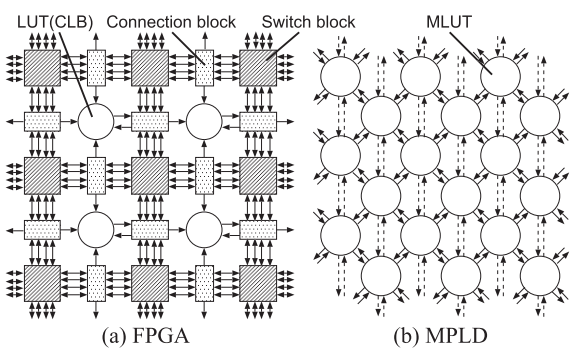


Fig. 1 (a) FPGA's fabric and (b) MPLD's fabric.

Manuscript received May 2, 2011.
Manuscript revised September 5, 2011.

[†]The authors are with the Graduate School of Information Sciences, Hiroshima City University, Hiroshima-shi, 731–3194 Japan.
[††]The authors are with Taiyo Yuden Co., Ltd, Tokyo, 110–0005 Japan.
a) E-mail: nakamura@ca.info.hiroshima-cu.ac.jp
b) E-mail: inagi@hiroshima-cu.ac.jp

DOI: 10.1587/transinf.E95.D.324

*MPLD is a trademark of Taiyo Yuden Co., Ltd.

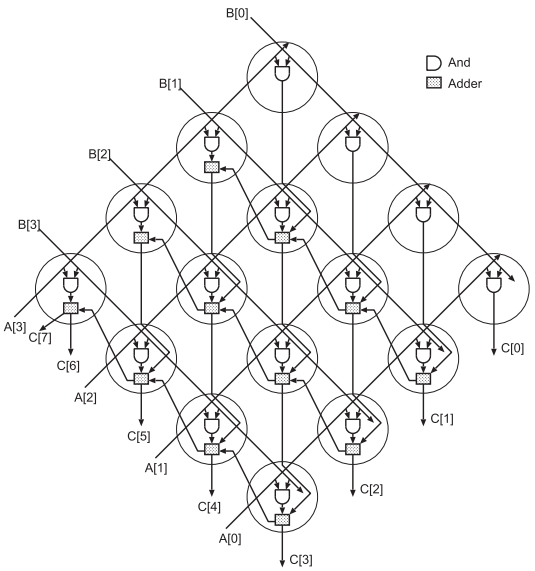


Fig. 2 Example of efficiently mapped IP module (4-bit multiplier).

route tool (VPR) [5], [6] is a well-known implementation of the placement and routing techniques of FPGAs. In placement and routing techniques, logic cells must be connected by wires. However, in an FPGA, the number of routing elements and the connections between them are fixed. Therefore, in an FPGA, the degree of routing freedom is ensured by preparing many routing lines in the device. On the other hand, in an MPLD, this degree can be ensured by choosing not to differentiate between logic and routing elements. Hence, the degree is determined by the dispersion of placed logic cells as obstacles. Thus, directly applying the existing placement and routing algorithms of FPGAs to MPLDs may possibly lead to the overcrowding of the placed logic cells and cause shortage in the number of routing elements between logic cells. As a result, the target circuit may fail to be mapped into the MPLD.

In this paper, we propose and implement a placement method for MPLDs on the basis of simulated annealing (SA) [7], which is also used in some methods for FPGAs. To prevent logic cells from overcrowding and to preserve MLUTs for routing, our proposed method uses the cost function that considers the detailed congestion of nets and also the nearness between cells. We also implement a routing method that has been modified for MPLDs. To evaluate the effectiveness of our physical design method, and to demonstrate the MPLD's ability, we performed several experiments, which showed that MPLDs realize sequential circuits from ISCAS'89 benchmark suite [8]. In addition, 13 circuits that could not be mapped onto an MPLD using VPR were successfully mapped by our method.

The rest of this paper is organized as follows. Section 2 introduces the basic structure of an MPLD. Section 3 introduces the entire process of our EDA tool for MPLDs. In Sects. 4 and 5, our placement and routing methods are presented. Section 6 presents the experimental results, and Sect. 7 summarizes this paper.

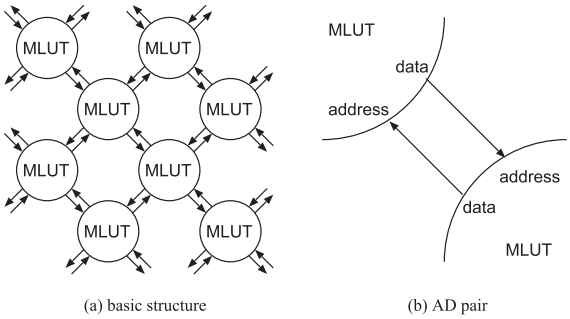


Fig. 3 Basic structure of an MPLD.

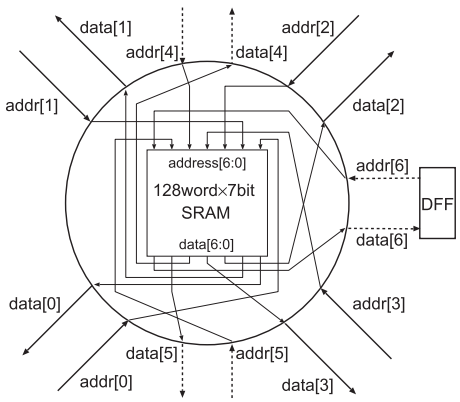


Fig. 4 Structure of an MLUT ($N = 7$).

2. Structure of MPLDs

One of the characteristics of MPLDs is that there is no difference between logic and routing elements. Standard FPGAs have switch blocks, connection blocks, and so on, which route the connecting wires between logic cells. In contrast, an MPLD consists of diagonally connected MLUTs (Fig. 3), each of which can be used as either a logic element or a routing element. An MLUT has multiple input and output terminals so that some logic cells or segments of wires can be realized in the MLUT.

An MLUT is realized by a memory module as shown in Fig. 4. An *address/data* (*AD*) *pair* (AD[i], $0 \le i \le N-1$) refers to a pair of bits of its address line (addr[$N-1, 0$]) and its data line (data[$N-1, 0$]). An MLUT with N AD pairs is realized using a $2^N \times N$ bit memory module.

As shown in Fig. 5, there are two types of interconnections between AD pairs. One is for the diagonal connection of adjacent MLUTs (*adjacent line*), and the other is for the connection of distant MLUTs (*distant line*).

In our current implementation, each MLUT has four AD pairs (AD[0]-AD[3]) for adjacent lines and three AD pairs (AD[4]-AD[6]) for distant lines (Figs. 4 and 5). Two of the three distant lines (from AD[4] and AD[5]) are connected to the horizontally adjacent MLUTs, and one line (from AD[6]) is connected in several ways. One third of the AD[6]s of MLUTs connect to flip-flops, and the remaining

two thirds connect to distant MLUTs by skipping five adjacent lines (including horizontal distant lines that connect horizontally adjacent MLUTs), both diagonally and horizontally. (Since we are still in the process of determining the best architecture for distant lines, in this paper, we considered only adjacent lines in our placement algorithm. In additional experiments, we also ensured that the absence of a full understanding of distant lines used in the placement in the current implementation did not degrade routing results presented here.)

A target circuit is synthesized as memory data and is configured to the MPLD.

In FPGAs, logic cells and nets are realized by different resources (*i.e.*, LUTs and SBs), and the location and ratio of LUTs and SBs are fixed. In contrast, in MPLDs, both logic cells and nets are realized using identical resources (*i.e.*, MLUTs), and the location and ratio of MLUTs working as logic and routing elements can be configured depending on the situation.

Note that an MLUT can function as both logic and routing elements simultaneously. Figure 6 illustrates an example of one such MLUT with four AD pairs. The MLUT in Fig. 6 (a) functions as a wire segment between addr[3] and data[2], and a 3-input AND gate whose inputs are addr[0], addr[1], and addr[2], and output is data[3]. The wire segment and 3-input AND gate are represented by the truth tables shown in Fig. 6 (b). The configuration data for the MLUT (Fig. 6 (c)) is generated by integrating those tables, and the wire segment and 3-input AND gate are realized.

This structure potentially results in lower manufacturing cost. Unlike FPGAs, the number of wiring layers required to constitute an MPLD is the same as that of an SRAM module, and MPLDs can be manufactured employing the regular CMOS ASIC manufacturing process, because MPLDs contain no SBs, which require additional metal wiring layers to efficiently route nets. Thus, MPLDs are expected to have lower costs than FPGAs, and to be easily integrated in SoCs.

As mentioned above, in an MPLD, there is no difference between logic and routing elements. If this characteristic is not considered in its placement and routing processes, the area efficiency may degrade and the processes may also fail. In other words, the efficiency of MPLDs depends on the placement and routing method.

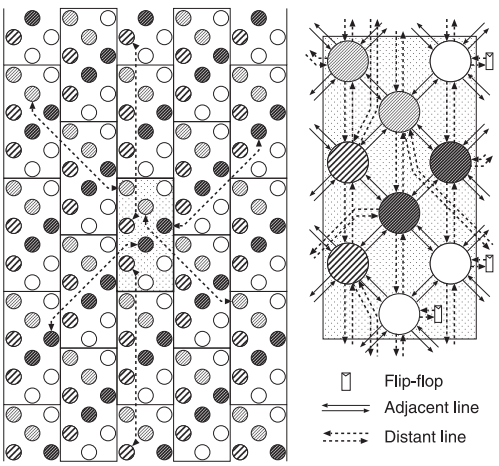
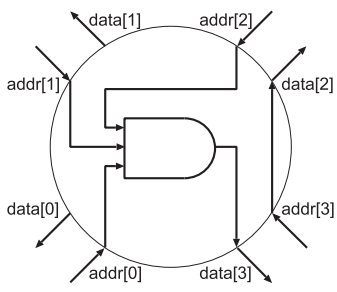


Fig. 5 Adjacent lines and distant lines.



(a) example of mapped logic and wire

addr 2 1 0	data[3]
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

addr[3]	data[2]
0	0
1	1

(b) truth tables of logic and wire

addr 3 2 1 0	data 3 2 1 0
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 0
0 0 1 0	0 0 0 0
0 0 1 1	0 0 0 0
0 1 0 0	0 0 0 0
0 1 0 1	0 0 0 0
0 1 1 0	0 0 0 0
0 1 1 1	1 0 0 0
1 0 0 0	0 1 0 0
1 0 0 1	0 1 0 0
1 0 1 0	0 1 0 0
1 0 1 1	0 1 0 0
1 1 0 0	0 1 0 0
1 1 0 1	0 1 0 0
1 1 1 0	0 1 0 0
1 1 1 1	1 1 0 0

(c) SRAM data for configuration

Fig. 6 Example of logic and wire mapped to an MLUT.

3. Flow of Our EDA Tool for MPLDs

In this paper, we propose a placement and routing method for MPLDs. This method is implemented as a part of our EDA tool for MPLDs. The flow of the EDA tool is shown in Fig. 7.

First, in logic synthesis, an architecture-independent gate-level netlist (Fig. 8 (b)) is synthesized from the RTL de-

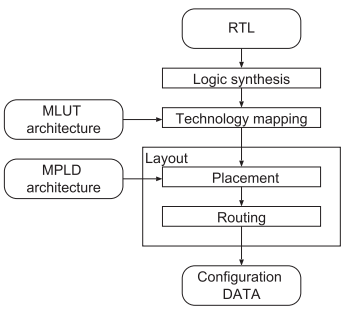


Fig. 7 Entire flow of our EDA tool.

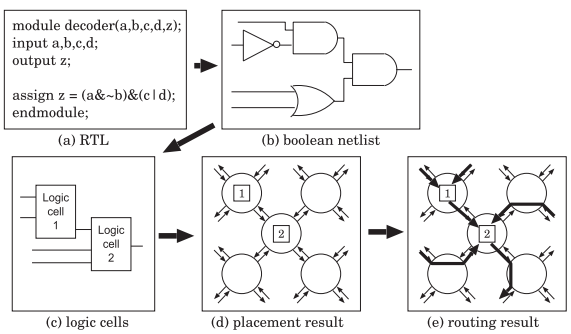


Fig. 8 Data conversion.

sign of the target circuit written in HDL (Fig. 8 (a)). This is done using a logic synthesizer such as Design Compiler [9]. Next, in technology mapping, the gate-level netlist is converted to a cell-level netlist (Fig. 8 (c)), making cells by clustering gates. Each cell is represented by a LUT, and must be smaller than an MLUT in terms of the number of inputs and outputs. In the current version of our EDA tool, this is done using an open source technology mapping tool [10]. After technology mapping, the cells in the cell-level netlist are placed in the target MPLD (Fig. 8 (d)). In other words, each cell is assigned to an MLUT, which can be shared by cells as long as the total size of the cells is less than that of the MLUT. After the placement of the cells, the route of each net, which electrically connects cells, is decided in the routing process. Finally, the bit-stream data for MLUTs is generated as the configuration data of the target MPLD.

Because the placement and routing processes depend on the target device, it is difficult to directly apply FPGAs' placement and routing techniques to MPLDs. Unlike FPGAs, MPLDs have no specific resources for routing, and use MLUTs as both logic and routing elements. Therefore, if these features are not considered in the placement and routing processes, MPLDs cannot function efficiently and effectively. Therefore, in this paper, we propose a placement and routing method for MPLDs and demonstrate the ability of the MPLD architecture.

4. Placement Algorithm

Placement for MPLDs is a process that involves the allocation of cells to MLUTs. Our proposed placement algorithm is based on a meta-heuristic algorithm called SA, which is often used for circuit placement. SA is also used by VPR, which is a well-known placement tool for FPGAs [5], [6]. In this section, we explain how to fit SA to the placement process for MPLDs.

4.1 SA

SA [7] is a combinatorial optimization technique. The flow of SA is shown in Fig. 9. SA generates a neighbor solution from the current solution and compares them using a cost function before deciding whether to move it to the neighbor solution. SA searches for an optimal solution by repeating this process. A feature of this method is the use of the temperature parameter T. When deciding whether to move a solution, if the cost of the neighbor solution is higher than that of the current solution, the move is accepted by the probability of $e^{-\frac{\Delta cost}{T}}$. The higher the value of T is, the higher will be the acceptance ratio. As shown in Fig. 9, T decreases due to the cooling process after sufficient generation of neighbor solutions in the temperature. At the beginning of SA, the solution space is widely searched. As the temperature T decreases, the searched area in the solution space is converged. Then, at the end of SA, the area near the optimal solution is extensively searched. This method reduces the possibility of being caught near a local optimum by stochastically accepting worse solutions. SA is terminated when T attains the termination temperature T_{end} where (in VPR and our placement method) $T_{end} = \varepsilon \frac{\text{cost}}{N_{net}}$, N_{net} is the number of the nets of the target circuit, and ε is a user-defined constant and is set to 0.005.

```
s = s0;                                  // s: current sol., s0: initial sol.
sb = s;                                  // sb: best sol.
c = cost(s);                             // c: current cost., cost(s): cost of s
T = T0;                                  // T: current temp., T0: initial temp.
M = M0;                                  // M: number of loops, M0: initial M

while(T>Tend){                           // repeat until T reaches the terminal temp.
    loop_cnt = 0;
    while(loop_cnt <M){                  // repeat neighbor generation
        sn = neighbor(s);                // make a neighbor solution
        cn = cost(sn);                   // calculate the cost of the neighbor

        if (rand() <e^{(c-cn)/T}){       // accept check (comparing the costs)
            s = sn;                      // update the current solution
            c = cn;
            update_best_sol(sb,s); // update the best solution if necessary
        }
        loop_cnt = loop_cnt + 1;
    }
    T = α*T;                             // cooling
    M = β*M;
}

return sb;
```

Fig. 9 Pseudo code of SA.

The number of loops M refers to the number of times the neighbor solution is generated at the same temperature. In our implementation, by binary search, the initial temperature T_0 is set to the temperature at which the acceptance ratio is about 90%. In our proposed placement method, the number of loops at the initial temperature is defined as $M_0 = 10 \times {N_{net}}^{1.33}$, as defined in VPR. After a neighbor solution is generated M times, T cools down.

In cooling, T and M are updated as follows: $T_{k+1} = \alpha \times T_k$, and $M_{k+1} = \beta \times M_k$. In our method, α and β are set to 0.9 and 1, respectively.

4.2 Neighbor Function

When using SA, we need to define a way to generate neighbor solutions from the current solution and compare them. Our placement method adopts two ways to generate a neighbor solution. One is *the migration of a logic cell* and the other is *the exchange of logic cells*.

In the migration of a logic cell, a logic cell and an MLUT are selected at random, and the cell is replaced to the MLUT. With this move, the maximum range of moves needed to limit the migration distance is given. The migration distance of a cell is defined by the length of the shortest path without considering distant lines from the source to the destination MLUTs of the cell. The length of a path indicates the number of MLUTs on the path. The maximum range of moves m is initially defined by the length of the longer (horizontal or vertical) edge of the target MPLD. Then, it is gradually decreased by the function m_{k+1} =

$\max(4, 0.9m_k)$ as T cools down. By considering this range, a wider area of the solution space is searched at the beginning of SA, and an intensive search is conducted at the end for optimization. In the exchange of cells, two cells are selected at random, and their positions (*i.e.*, MLUTs to which they belong) are exchanged.

Note that the total number of input (output) signals of the cells in an MLUT must be less than the number of input (output) terminals of the MLUT. Infeasible neighbor solutions are canceled, and other solutions are generated until a feasible one is found.

4.3 Cost Function

One of the most important factors of SA is its cost function. Because SA is an algorithm that searches for the optimal solution by comparing the costs of solutions evaluated by its cost function, a function that adequately evaluates a desirable solution at a low (*i.e.*, good) value is necessary. In our cost function, in addition to the estimated total wire length, which is also used in VPR, estimated wire congestion and *cell nearness penalty* that reserves a routing area are considered. When calculating these factors, the routing directions of signals are considered.

The first factor of our cost function is the *estimated total wire length*, and is defined as the summation of the shortest path lengths of all nets. The longer the total wiring length, the higher the cost, and it has the effect of shortening the total wiring length. It suppresses the amount of used routing resources, shortens the signal delays, and causes cells with strong connections (connected with many nets) to become mutually close.

The second factor is the *wire congestion* that judges the degree of the concentration of the routes of the nets. In other words, placements with congested areas are evaluated at high value by this factor. As a result, the interference between the routes of nets is suppressed.

The third factor is the *cell nearness penalty*. This checks the length between cells and works as a repulsive force to prevent the cells from being too close to each other. This factor reserves MLUTs for routing around cells.

On the basis of these three factors, our proposed cost function for MPLDs is defined as Expression (1):

$$\text{Cost} = p \times \text{length} + q \times \text{congestion} + r \times \text{nearness}. \quad (1)$$

In this cost function, the above mentioned three factors are represented as *length*, *congestion*, and *nearness* with coefficients p, q, and r, respectively, which are user defined coefficients to control the balance of the factors. To ease the calculation, distant lines are not considered in the cost function.

4.3.1 Length

Length in Expression (1) represents the estimated total wire length and its role is to reduce the wire length. It is formulated as shown in Expression (2):

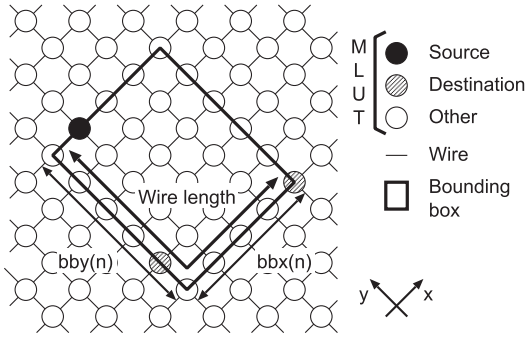


Fig. 10 Bounding box and wire length.

$$\text{length} = \sum_{n \in E_c} q(n)\{bb_x(n) + bb_y(n)\}, \quad (2)$$

where E_c is the set of all nets of the target cell-level netlist, $q(n)$ is the weight for the net n and depends on the number of cells to which the net connects, and $bb_x(n) + bb_y(n)$ is the half perimeter of the bounding box of all cells (MLUTs) to which the net connects. According to the best value of $q(n)$ discussed in [11], we approximate $q(n)$ by $0.615 \min\{s(n)^{0.381}, 50\}$, where $s(n)$ is the number of cells to which the net n connects. The bounding box of cells (MLUTs) refers to the minimum rectangle covering all the cells (MLUTs). The bounding box of cells to which a net n connects is called the bounding box of the net n. $bb_x(n)$ ($bb_y(n)$) is the length of the bounding box along the x-axis (y-axis). Note that because our target MPLD can be considered as a diagonal grid of MLUTs, we define bounding boxes and x- and y-axes, diagonally, as shown in Fig. 10. MLUT(k, l) refers to the MLUT at point (k, l) in the diagonal Cartesian coordinate system.

4.3.2 Congestion

Congestion in Expression (1) represents the estimated congestion of the routes of the nets. When the estimated routes of the nets are congested in a specific area, the factor becomes high. It reduces the overcrowding of cells, which interferes with routing. In the definition of (wire) congestion, the bounding box used in the definition of length is also used. In the estimation, we assume that each net is routed in the area of its bounding box without any detour, thereby allowing overlapping of routes. The bounding boxes of the nets can overlap with each other, which means that there is a possibility of the interference of nets, and if many of the nets' bounding boxes overlap in a specific area, this possibility increases. Therefore, the area on which nets concentrate is roughly estimated from the number of overlapping bounding boxes of nets.

In addition, we consider the direction of the routes, because only the nets whose routes have the same direction can interfere with each other in the MPLD architecture. Thus, for each wire of an AD pair, its congestion level is separately calculated, as shown in Fig. 11. The congestion of a placement solution is defined as Expression (3):

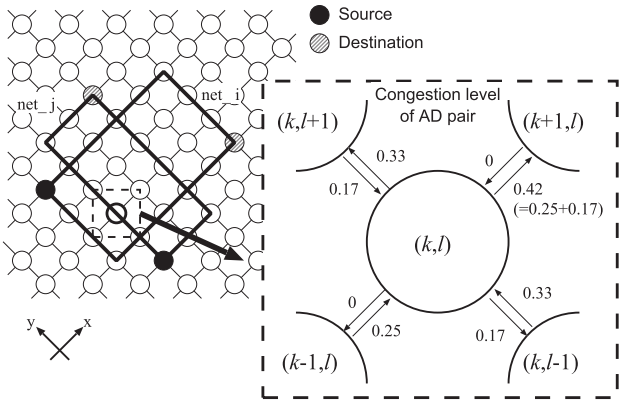


Fig. 11 Example of calculation of congestion.

$$\text{congestion} = \sum_{\text{MLUT}(k,l)\in V} \Big\{ g_x(k,l)^2 + g_{-x}(k,l)^2 + g_y(k,l)^2 + g_{-y}(k,l)^2 \Big\}, \quad (3)$$

where V is the set of all MLUTs in the target MPLD and $g_x(k,l)$ is the congestion level of the wire from the MLUT(k,l) in the x direction (*i.e.*, the wire from MLUT(k,l) to MLUT$(k+1,l)$). Similarly, $g_{-x}(k,l)$, $g_y(k,l)$, and $g_{-y}(k,l)$ are the congestion levels of the wires from the MLUT(k,l) in the $-x$, y, and $-y$ directions, respectively. The congestion level of a wire of an AD pair is the summation of all the congestion levels given by the bounding boxes covering that wire. A congestion level given by a bounding box has its direction. Suppose MLUT(u,v) is the signal source of a net n. The congestion level for a wire from the MLUT(k,l) in the x direction ($-x$ direction) is added the congestion level of the bounding box if $k \geq u$ ($k \leq u$). The congestion levels in the y and $-y$ directions are added in the same way. $c_x(n)$ and $c_y(n)$, which are the congestion levels in the $\pm x$ and $\pm y$ directions of the bounding box, respectively, are calculated as follows:

$$c_x(n) = \frac{1}{bb_y(n)+1}, \quad c_y(n) = \frac{1}{bb_x(n)+1}.$$

The congestion levels of the congested areas are emphasized by the squared parameters g_x, g_{-x}, g_y, and g_{-y} in Expression (3).

For example, in the case where the two bounding boxes of nets i and j overlap with each other as shown in Fig. 11, the congestion levels added by the bounding boxes of nets i and j are as follows:

$$c_x(i) = 0.25, \ \ c_y(i) = 0.17, \ \ c_x(j) = 0.17, \ \ c_y(j) = 0.33.$$

Now, we focus on the congestion levels for wires of AD pairs around the MLUT (k,l) in Fig. 11. For example, both the bounding boxes of nets i and j have congestion levels in the x direction (*i.e.*, $c_x(i) = 0.25$ and $c_x(j) = 0.17$). Thus, $g_x(k,l) = 0.42$ $(= 0.25 + 0.17)$ is provided as the congestion level of the wire in the x direction. In another example, the bounding box of net i has a congestion level in the y direction (*i.e.*, $c_y(i) = 0.17$), and the bounding box of net j has a congestion level in the $-y$ direction (*i.e.*, $c_y(j) = 0.33$). In this case, the congestion levels in the y and $-y$ directions are respectively provided as $g_y(k,l) = 0.17$ and $g_{-y}(k,l) = 0.33$, without any interference. Congestion of a placement solution can be estimated in this way.

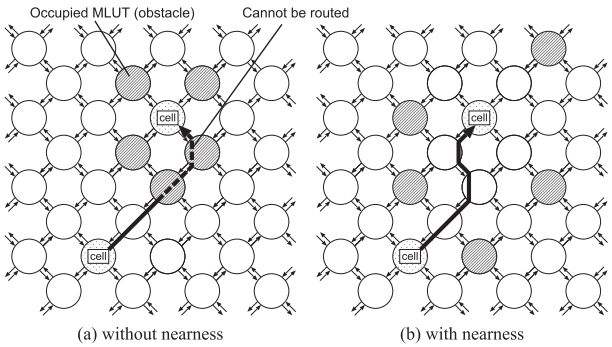


Fig. 12 Effect of nearness.

4.3.3 Nearness

Unlike FPGAs, an MPLD uses the MLUTs as both logic and routing elements. Thus, MLUTs used as routing elements are necessary between MLUTs that contain logic cells. If logic cells are overcrowded in a specific area, it becomes impossible to ensure that there are sufficient routing elements for connections. To resolve this problem, our cost function incorporates the nearness between logic cells as part of the cost.

The nearness in Expression (1) represents the closeness of logic cells to each other. The cost of the nearness of a pair of logic cells a and b is defined as

$$\text{p\_nearness}(a,b) = \begin{cases} \sigma - d(a,b) & (0 < d(a,b) \leq \sigma) \\ 0 & (\text{otherwise}) \end{cases},$$

where $d(a,b)$ is the distance (*i.e.*, the shortest path length in the MPLD) between logic cells a and b and σ is a user defined constant that is empirically set to 4 in our experiments. The nearness of the placement solution is defined as

$$\text{nearness} = \sum_{a,b\in V_c} \text{p\_nearness}(a,b),$$

where V_c is a set of the logic cells of the cell-level netlist, and works as a repulsive force between cells to ensure MLUTs for routing (Fig. 12). Thus, σ can be considered to correspond to the number of tracks (between adjacent LUTs) in FPGAs. Note that p_nearness$(a,b) = 0$ if the distance between cells a and b is 0 (*i.e.*, a and b are packed in the same MLUT). This ensures the efficiency of the placement.

5. Routing Algorithm

The routing process decides the paths of the nets between placed logic cells. Our routing algorithm is based on Dijkstra's shortest path algorithm. For rip-up and re-routing,

we adopt a subset of Shirota's algorithm [12]. In the routing process, we use both adjacent and distant lines. We consider wire congestion through the routing process. The details are described as follows.

5.1 Flow of Routing Process

Our routing method consists of three steps: *pre-routing*, *actual routing*, and *rip-up and re-routing*. In pre-routing, each net is routed ignoring the previously routed nets. It is used to estimate wire congestion, and the estimated congestion is used in actual routing. The nets that are routed by pre-routing are removed before actual routing. In actual routing, each net is routed considering the estimated congestion and the previously routed nets. If it fails to route all the nets, rip-up and re-routing is conducted. In this step, nets passing through a congested area are removed and re-routed.

5.2 Pre-Routing

In pre-routing, nets are routed one at a time, ignoring the previously routed nets. That is, nets can overlap with each other. A multi-terminal net is decomposed into two-terminal nets when the net is routed. This is done by using the maze routing algorithm [13]. Note that an MPLD has distant lines that connect distant MLUTs. They are considered in maze routing.

5.3 Actual Routing

An MPLD is modeled as a directed graph $G = (V, E)$, where V is a set of the vertices corresponding to MLUTs and E is a set of the edges corresponding to wires of AD pairs. On the basis of the result of pre-routing, the estimated congestion for each edge is defined by the number of nets passing through the edge in pre-routing. The initial length of each edge is defined by the estimated congestion of the edge. In actual routing, nets are routed one at a time. Thus, the length is updated to infinity when the edge is used. For each net, the shortest path is found as its route using Dijkstra's shortest path algorithm. For a multi-terminal net, a Steiner tree is found by adding a terminal to the previously found partial Steiner tree one at a time in decreasing order of the distance from the source terminal. During actual routing, the estimated congestion of each edge is updated once for every 5% of routing that is completed in terms of the number of routed nets, and is reflected by the length of the edge.

5.4 Rip-Up and Re-Routing

If all the nets cannot be routed in actual routing, rip-up and re-routing is conducted using a subset of the algorithm proposed in [12]. It consists of three phases: (a) *routing with violations*, (b) *local rip-up and re-routing*, and (c) *global rip-up and re-routing*.

(a) Routing with violations is almost the same as actual routing. However, when the edge is used, the length of each edge is not updated to infinity but to a large value.

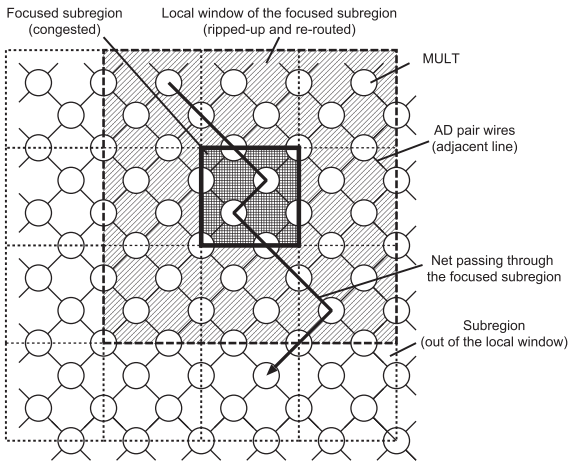


Fig. 13 Subregion and local window.

(b) In local rip-up and re-routing, the routing area is divided into subregions (rectangular tiles), and up to 50 subregions are processed in decreasing order of the number of violations present. (Let $N_{net}(w)$ be the number of nets passing through the wire w. The number of violations on an AD pair wire w is denoted by $v(w)$, and is defined as $v(w) = \max\{N_{net}(w) - 1, 0\}$. The number of violations in a subregion t is defined as $\sum_{\text{wire } w \text{ in } t} v(w)$.) A subregion and its eight neighbor subregions form a *local window* (Fig. 13). For each subregion, the highest-cost net passing through the subregion is selected, and the segment of the net in the local window is removed and re-routed. The cost of a net is defined by the sum of the number of violations on the AD pair wires (in the subregion) through which the net passes. For each subregion t, rip-up and re-routing of a net is performed up to $N_{net}(t)$ times, where $N_{net}(t)$ is the number of nets (partially or entirely) included in the subregion.

(c) In global rip-up and re-routing, a net passing through the congested region is selected, and re-routing is performed by Dijkstra's shortest path algorithm, as in the case of actual routing. Global rip-up and re-routing is performed up to 50 times.

In our routing algorithm, we omit local rip-up and re-routing after global rip-up and re-routing in [12]. In addition, Dijkstra's shortest path algorithm is used instead of the fast coarse maze routing algorithm used in [12], for simplicity.

6. Experiments

We conducted experiments to evaluate our proposed method and the capability of MPLDs. In the experiments, circuits from ISCAS'85 and ISCAS'89 benchmark suites [8], [14], [15] were placed and routed on MPLDs with 15×30, 33×36, 63×60 and 93×90 MLUTs.[†] Each of the coefficients p, q, and r of the cost function of placement was set to 0, 1, 5, 10,

[†]An $H \times W$ MPLD is decomposed into W columns each of which has H MLUTs.

Table 1 Placement and routing result of ISCAS'89.

			our method					VPR	
circuit	gates	MLUTs	p	q	r	Rtg R.	MLUT U.	Rtg R.	MLUT U.
s27	10	15*30	1	5	0	100.0%	1.06%	100.0%	1.07%
s208	96	15*30	5	1	0	100.0%	4.14%	100.0%	1.06%
s298	119	15*30	5	5	0	100.0%	8.43%	100.0%	8.93%
s344	160	15*30	5	5	0	100.0%	11.40%	100.0%	11.55%
s349	161	15*30	1	5	0	100.0%	11.47%	100.0%	11.73%
s382	158	15*30	10	15	0	100.0%	11.80%	100.0%	11.95%
s386	159	15*30	5	1	1	100.0%	17.16%	100.0%	17.38%
s400	162	15*30	15	0	1	100.0%	11.77%	100.0%	12.13%
s420	196	15*30	1	1	5	100.0%	13.65%	100.0%	14.02%
s444	181	15*30	5	1	0	100.0%	11.25%	100.0%	12.87%
s510	211	15*30	10	10	20	100.0%	41.89%	93.2%	N/A
s526	193	15*30	1	5	10	100.0%	13.20%	100.0%	14.19%
s526n	194	15*30	1	1	10	100.0%	13.80%	100.0%	14.48%
s641	379	33*36	5	5	1	100.0%	12.84%	98.3%	N/A
s731	393	15*30	5	5	15	100.0%	31.71%	98.3%	N/A
s820	289	33*36	15	10	15	100.0%	20.14%	94.2%	N/A
s832	287	33*36	10	5	10	100.0%	21.48%	92.3%	N/A
s838	390	33*36	1	1	0	100.0%	10.66%	100.0%	11.11%
s953	395	63*60	20	1	15	100.0%	12.05%	95.6%	N/A
s1196	529	63*60	1	5	5	100.0%	20.51%	91.3%	N/A
s1238	508	63*60	15	5	15	100.0%	14.62%	92.9%	N/A
s1423	657	63*60	5	0	0	100.0%	8.16%	100.0%	8.16%
s1488	653	63*60	1	5	1	100.0%	17.19%	92.2%	N/A
s1494	647	63*60	15	10	10	100.0%	16.33%	87.4%	N/A
s5378	2779	93*90	1	1	1	100.0%	17.97%	81.9%	N/A
s9234	5597	93*90	1	1	1	100.0%	11.86%	91.3%	N/A
s13207	7951	93*90	1	15	10	100.0%	27.48%	99.8%	N/A
s15850	9772	93*90	1	1	5	98.4%	N/A	77.0%	N/A
s35932	16095	93*90	1	10	5	96.2%	N/A	92.5%	N/A
s38417	22179	93*90	1	5	5	97.2%	N/A	88.3%	N/A
s38584	19253	93*90	1	5	10	96.4%	N/A	86.3%	N/A
P&R success R.			87.1%					45.2%	

15, and 20. Note that our cost function when $q = 0$ and $r = 0$ is equivalent to that of VPR [5]. For each combination of the coefficient values, placement and routing were performed 10 times, and the best solution was chosen. The size of a subregion in the routing process was set to 3×3 MLUTs. σ in p_nearness was set to 4.

Table 1 shows the experimental results of ISCAS'89 benchmark circuits using both our method and VPR (emulated by our method when $q = 0$ and $r = 0$). In the table, Rtg R. represents the ratio (expressed as percentage) of the number of nets that were successfully routed to the total number of nets. MLUT U. (usage) represents the ratio of the number of used MLUTs. Note that MLUTs are also used as routing elements. In addition, p, q, and r are the coefficients of the cost function when the best result (with the highest Rtg R., and then the lowest MLUT usage) is achieved. P&R success R. (ratio) represents the ratio of the number of circuits successfully routed to the total number of circuits. According to Table 1, when a circuit is small compared to the target MPLD, the best p tends to be large when compared to the corresponding q and r, and vice versa. This is because there is sufficient room for routing when the circuit is small, but there is not enough room when the circuit is large. This implies that congestion and nearness work well to improve the placement when there are only a small number of MLUTs that can be used for routing. Moreover, in the experiment, only 14 circuits out of the 31 circuits were successfully routed on MPLDs using VPR, while 27 circuits were successfully routed using our method when the coefficients fit the circuits and MPLDs. This shows the effectiveness of the congestion and nearness factors in placement for MPLDs.

Next, to compare MPLDs with FPGAs, we made MPLDs imitate FPGAs by fixing logic and routing elements in three patterns, FPGA patterns 1, 2 and 3 (Fig. 14), and mapped the benchmark circuits using VPR. Note that the FPGA pattern 1 contains more logic elements than the pattern 3, and the pattern 3 contains more routing elements than the pattern 1. The FPGA pattern 2 is an intermediate pattern between the FPGA patterns 1 and 3.

Table 2 shows the experimental results for the FPGA patterns 1, 2 and 3 obtained using VPR. By comparing Tables 1 and 2, we observed that with our method, MPLDs were the best in terms of both Rtg R. and MLUT U., and thus P&R success R. This shows the effectiveness of MPLDs that results from the MLUT's flexibility and our method. In addition, we found that the pattern 2 was much better than the pattern 1 in terms of Rtg R. This confirms the importance of the amount of routing resources in Rtg R. On the other hand, the pattern 2 resulted in the worse MLUT U. This is because

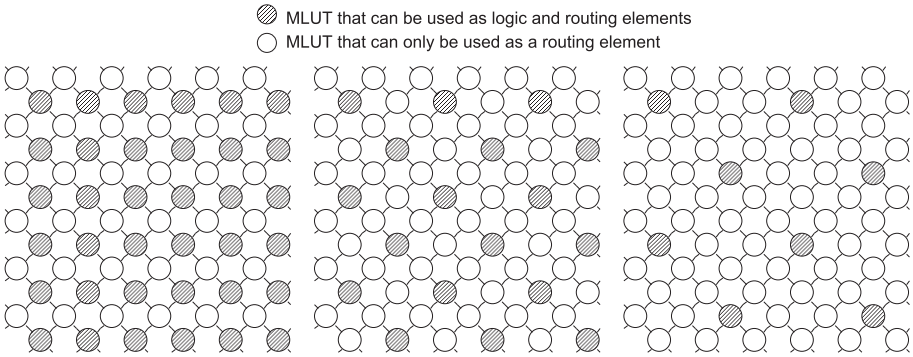


Fig. 14 MLUT patterns to imitate FPGAs.

Table 2 Placement and routing result of each FPGA pattern.

	pattern1		pattern2		pattern3	
circuit	Rtg R.	MLUT U.	Rtg R.	MLUT U.	Rtg R.	MLUT U.
s27	100.0%	1.39%	100.0%	1.46%	100.0%	2.23%
s208	100.0%	4.98%	100.0%	6.34%	100.0%	7.62%
s298	100.0%	10.82%	100.0%	11.47%	100.0%	15.40%
s344	100.0%	15.06%	100.0%	14.74%	100.0%	18.00%
s349	100.0%	15.03%	100.0%	15.32%	100.0%	18.44%
s382	100.0%	13.64%	100.0%	15.48%	97.2%	N/A
s386	100.0%	20.96%	100.0%	24.78%	100.0%	22.86%
s400	100.0%	13.97%	100.0%	16.20%	95.7%	N/A
s420	100.0%	15.51%	100.0%	14.62%	98.7%	N/A
s444	100.0%	15.43%	100.0%	15.51%	97.8%	N/A
s510	98.4%	N/A	93.7%	N/A	N/A	N/A
s526	100.0%	19.36%	100.0%	18.32%	N/A	N/A
s526n	100.0%	15.76%	100.0%	18.40%	N/A	N/A
s641	100.0%	13.80%	100.0%	14.47%	100.0%	15.88%
s731	99.6%	N/A	100.0%	35.52%	N/A	N/A
s820	99.8%	N/A	100.0%	23.08%	86.6%	N/A
s832	99.6%	N/A	100.0%	23.79%	89.7%	N/A
s838	100.0%	12.77%	100.0%	14.65%	98.2%	N/A
s953	96.1%	N/A	100.0%	15.61%	100.0%	14.73%
s1196	92.8%	N/A	99.9%	N/A	100.0%	19.62%
s1238	94.7%	N/A	100.0%	16.10%	100.0%	19.14%
s1423	100.0%	9.41%	100.0%	9.11%	100.0%	10.67%
s1488	94.3%	N/A	96.4%	N/A	99.6%	N/A
s1494	95.1%	N/A	99.4%	N/A	99.8%	N/A
s5378	91.5%	N/A	100.0%	19.95%	100.0%	20.27%
s9234	96.8%	N/A	100.0%	16.13%	100.0%	13.13%
s13207	92.4%	N/A	100.0%	32.10%	N/A	N/A
s15850	94.4%	N/A	97.2%	N/A	N/A	N/A
s35932	91.8%	N/A	95.4%	N/A	N/A	N/A
s38417	92.1%	N/A	96.4%	N/A	N/A	N/A
s38584	92.7%	N/A	94.2%	N/A	N/A	N/A
P&R success R.	48.4%		74.2%		41.9%	

the total wire length was extended by its sparse logic elements. MPLDs obtained with our method can shorten the total wire length by increasing the density of logic elements and reserving routing elements in the process.

Table 3 shows the placement and routing results for c1908 from ISCAS'85 benchmark circuits for various patterns of p, q, and r. c1908 was mapped (*i.e.*, placed and routed) on an MPLD with 33×36 MLUTs. Success R. refers to the success rate of mapping. Note that mapping was conducted 10 times for each coefficient pattern. With respect to the weight for wire length, $p = 1$ was the best. This is because a large p leads to overcrowding of logic cells, thereby shortening the total wire length. Next, with respect to the weight for wire congestion, $q = 1$ was the best, although there was no sufficient effect, except when $q = 0$. With respect to the weight for nearness, a large r drastically improved the success rate of mapping, although it slightly worsened the total wire length and subsequently the MLUT usage. This again shows the high effectiveness of the nearness factor in placement for MPLDs.

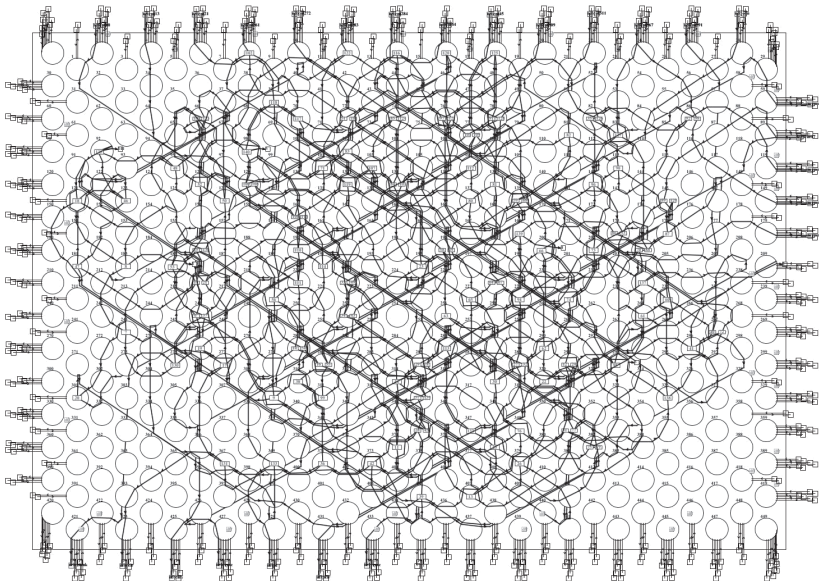


Fig. 15 Placement and routing result for s510 on a 15×30 MPLD.

Table 3 Result for c1908 for each cost.

p	q	r	Rtg R.	MLUT U.	success R.
1	1	1	100.0%	28.91%	40%
0	1	1	95.0%	N/A	0%
5	1	1	100.0%	28.12%	10%
10	1	1	95.0%	N/A	0%
15	1	1	92.9%	N/A	0%
20	1	1	92.9%	N/A	0%
1	0	1	95.3%	N/A	0%
1	5	1	100.0%	28.95%	60%
1	10	1	100.0%	30.03%	60%
1	15	1	100.0%	31.90%	70%
1	20	1	100.0%	32.84%	50%
1	1	0	90.3%	N/A	00%
1	1	5	100.0%	30.94%	90%
1	1	10	100.0%	30.94%	100%
1	1	15	100.0%	35.89%	100%
1	1	20	100.0%	35.89%	70%

In the experiments, some of the large circuits could not be mapped to MPLDs. Although the circuits may be too large for their target MPLDs, we propose two ways to improve our placement algorithm: (1) feedback of congestion information from the routing process to the placement process and (2) adaptive setting of parameters in nearness penalty. (1) The estimation of congestion in the placement process is based on the manner in which the bounding boxes of nets overlap with each other. This is a rough estimation and includes many errors. Because detailed congestion data is obtained after the routing process, it is expected that this problem will be eased by feeding back the detailed congestion data from the routing process to the placement process. (2) The nearness coefficient r and the parameter σ in p_nearness are shared by all the pairs of logic cells. However, the amount of routing resources required around a logic cell depends on the logic cell and its connections. It is therefore possible to improve routing results by analyzing the target netlist and providing different nearness parameters to pairs of logic cells.

Finally, Fig. 15 depicts a placement and routing result for s510 obtained from ISCAS89 benchmark circuits by using our method.

7. Conclusions

In this paper, we proposed a placement and routing method for a new reconfigurable device (MPLD) considering the overcrowding of logic cells and the interference of nets, and we evaluated its effectiveness and the MPLD's ability to realize a circuit. The experimental results showed that our proposed method reduces the overcrowding in logic cells and the subsequent interference of nets. As a result, circuits were successfully implemented on MPLDs.

Attempts are being made to put the device into commercial use in near future. The prototype MPLD chips have been fabricated and we have confirmed that our placement and routing method can drive these chips. Our future work includes clustering to handle large circuits, and feedback of routing results for effective placement.

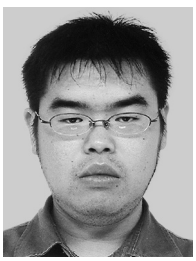
Acknowledgments

We would like to thank Mr. Ken Taomoto and Dr. Hideyuki Kawabata, Hiroshima City University, for providing the placement and routing viewer for MPLDs.

References

[1] S. Brown, R. Francis, J. Rose, and Z. Vranesic, Field-Programmable Gate Arrays, Kluwer Academic Publishers, 1992.
[2] Z. Marrakchi, H. Mrabet, U. Farooq, and H. Mehrez, "FPGA Interconnect topologies exploration," Int. J. Reconfigurable Computing, vol.2009, pp.1–13, 2009.
[3] N. Hirakawa, M. Yoshihara, K. Tanigawa, T. Hironaka, and M.

Sato, "A PLD Architecture for high performance computing," Proc. 2008 Int. Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, pp.35–42, 2008.
[4] K. Tanigawa, M. Asaeda, A. Yamada, M. Sato, and T. Ishiguro, "Memory array based PLD architecture for high-density logic mapping – Implementation of first demo chip," Proc. COOL chips XIV, poster 1, 2011.
[5] V. Bets and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," Proc. IEEE FPL 1997, pp.213–222, 1997.
[6] V. Bets, J. Rose, and A. Marquardt, Architecture and CAD dor Deep-submicron FPGAs, Kluwer Academic Publishers, 1999.
[7] P.J.M. van Laarhoven and E.H.L. Aarts, Simulated Annealing: Theory and Applications, Springer, 1987.
[8] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," Proc. 1989 Int. Symposium on Circuit and Systems, pp.1929–1934, May 1989.
[9] Synopsys, Inc., "Synopsys," http://www.synopsys.com/
[10] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification," http://www.eecs.berkeley.edu/˜alanmi/abc, Release 70930.
[11] C.E. Cheng, "Accurate and efficient placement routability modeling," Proc. IEEE/ACM ICCAD, pp.690–695, Nov. 1994.
[12] H. Shirota, S. Shibatani, and M. Terai, "A new rip-up and reroute algorithm for very large scale gate arrays," IEICE Trans. Fundamentals, vol.E80-A, no.3, pp.506–513, March 1997.
[13] S.M. Sait and H. Youssef, VLSI physical design automation: Theory and practice, pp.239–241, World Scientific Pub, 1999.
[14] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits," Proc. IEEE Int. Symposium on Circuit and Systems, pp.695–698, 1985.
[15] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," Proc. IEEE Design and Test of Computers, pp.72–80, 1999.



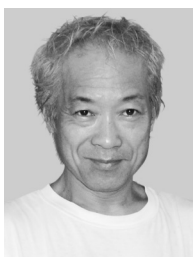
Masatoshi Nakamura received his B.S. degree in Computer Engineering from Hiroshima City University. He is currently a student at the Graduate School of Information Sciences, Hiroshima City University.

Masato Inagi received his B.E. and M.E. degrees in Computer Science and his Ph.D. degree in Engineering from Tokyo Institute of Technology in 2000, 2002, and 2008, respectively. He was a researcher at The University of Kitakyushu from 2005 to 2008. He has been a Research Associate at the Logic Circuit System lab., Hiroshima City University, since 2008. His research interest includes combinatorial algorithms for VLSI design automation.

Kazuya Tanigawa received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Hiroshima City University in 1999, 2001, and 2004, respectively. He has been a Research Associate at the Computer Architecture lab., Hiroshima City University, since 2004. His research interests are in the areas of reconfigurable architecture and parallel algorithms.

Tetsuo Hironaka received his B.E. degree from Yamaguchi University in 1988 and received his M.S. and Ph.D. degrees in Engineering from Kyushu University in 1990 and 1993, respectively. From 1993 to 1994, he served as a Research Associate in Kyushu University. From 1994 to 2006, he was an Associate Professor at Hiroshima City University. Since 2006, He has been a Professor at the Computer Architecture lab., Hiroshima City University. His research interests are in the areas of reconfigurable architecture, system software, and on-chip multiprocessing.

Masayuki Sato received his associate degree in Engineering from Ibaraki National College of Technology in 1971, and his Ph.D. degree from Tokyo Metropolitan University in 2009. He is currently with Taiyo Yuden Co., Ltd.



Takashi Ishiguro received his B.E. and M.E. degrees in Materials Engineering and his Ph.D. degree in Engineering from Tokyo Institute of Technology in 1977, 1979, and 1982, respectively. Since 1982, he has been with Taiyo Yuden Co., Ltd. In 2000, he received National Invention Award from Japan Science and Technology Agency for the invention of CD-R.