International Journal of Innovative
Computing, Information and Control
Volume 11, Number 6, December 2015
 ISSN 1349-4198
pp. 2001–2012

FPGA PLACEMENT BASED ON SELF-ORGANIZING MAPS

Motoki Amagasaki, Masahiro Iida, Morihiro Kuga
and Toshinori Sueyoshi

Graduate School of Science and Technology
Kumamoto University
2-39-1 Kurokami, Chuo-ku, Kumamoto 860-8555, Japan
{ amagasaki; iida; kuga; sueyoshi }@cs.kumamoto-u.ac.jp

Received May 2015; revised September 2015

Abstract. *A wide variety of field-programmable gate array (FPGA) routing structures have been proposed, as exemplified by three-dimensional stack and hierarchical-type FPGAs. However, since traditional FPGA tools are limited to island-style routing architectures, great effort is needed to construct various routing structures. To overcome this problem, we develop an FPGA design framework that is focused on improving exploration efficiency for various FPGA routing architectures. In the paper, we propose an FPGA placement method based on Kohonen self-organizing maps (SOMs). SOMs are one type of unsupervised learning artificial neural network. Because a lattice structure is typically used for the output layer of an SOM, the routing structure (routing topology) of an FPGA can be directly represented by the output layer. This is known as topological mapping, and it allows for various FPGA routing structures to be handled flexibly. We report the result of experimentally evaluating two types of FPGA structures: hierarchical fault-tolerant FPGAs and three dimensional FPGAs.*
Keywords: FPGA, Placement, SOM

1. **Introduction.** A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a user after manufacturing. In recent years, a wide variety of routing structures have been proposed, as exemplified by three-dimensional stack [1, 2] and hierarchical-type FPGAs [3]. Thus, it is necessary to develop electronic design automation (EDA) tools that can flexibly support different routing structures. Furthermore, increasing integration densities due to process scaling have made it necessary to obtain high-quality solutions as quickly as possible. FPGA placement has a large effect on execution time and circuit performance. Simulated annealing (SA) is often used as a solver for FPGA placement. Although SA-based placement methods [4] offer high-quality solutions, they require extensive computation, which is expected to become a problem for handling large-scale circuits in the future. Furthermore, for the majority of placement tools, adding support for routing structures that are not island-style structures requires major modifications to the program code.

To overcome this difficulty in creating various routing structures and handling the resulting computational complexity, we therefore propose an FPGA placement method based on Kohonen self-organizing maps (SOMs) [5]. SOMs are one type of unsupervised learning artificial neural network. The proposed placement algorithm projects the set of input patterns (the input vectors) onto the output layer according to degree of similarity. Because a lattice structure is typically used for the output layer of SOMs, the routing structure (routing topology) of an FPGA created in this way can be directly represented by the output layer. This is known as topological mapping, and it allows for various FPGA routing structures to be handled flexibly. Furthermore, in order to increase

computational efficiency, we employ a batch-learning-type SOM algorithm, which allows high parallelization and good convergence. Although some SOM-based FPGA placement algorithms [6, 7] have been studied, most tools are used under limited conditions. In this study, the proposed SOM-based placement tool is compared with traditional FPGA placement tools under various routing topologies.

Note that problems with limits on allowed topologies occur in the routing process, too. Our research group has developed EasyRouter [8], which represents various wiring topologies by using routing-resource graphs and performs routing on the obtained graphs, to address this. This paper is concerned with the placement problem. The structure of the rest of this paper is as follows. We briefly discuss related work in Section 2. We describe an FPGA placement problem in Section 3. We describe the proposed method in Section 4 and discuss modeling of the FPGA routing structure in Section 5. We then present the results of performance evaluation in Section 6 and, finally, summarize the paper in Section 7.

2. **Related Work.** The self-organizing principle has been previously applied to the VLSI placement problem [9, 10, 11, 12, 13]. For example, [9] presented the SOAP algorithm for circuit placement in arbitrarily shaped regions, including two-dimensional rectilinear regions and non-planar surfaces of three-dimensional objects. In [10], a three-layer force-directed SOM designed to resolve the circuit placement problem with arbitrarily shaped rectilinear modules is proposed. These studies treated VLSI placement only. Several publications have presented FPGA placement algorithms aimed at achieving short execution time. Although some SOM-based FPGA placement algorithms [6, 7] have been studied, most tools are used under limited conditions. In contrast, there are few studies about using various routing structures. Our proposed placement algorithm offers both the flexibility to use various structures and a reduction in execution time in comparison with conventional tools.

3. **FPGA Placement Problem.** This section describes the basic structure of FPGAs and defines the FPGA placement problem, which is treated as a combinatorial optimization problem.

3.1. **Structure of FPGAs.** Figure 1 shows the structure of traditional island-style FPGAs [14, 15, 16]. An FPGA consists of a logic unit of logic blocks (LBs) arranged into a two-dimensional array, an input-output block (IOB), and a routing unit that connects the LBs. Figure 2 shows the structure of an LB, which consists of basic logic elements (BLEs) and local routing for connecting these elements within the LB. A BLE consists of a lookup table (LUT) for implementing arbitrary combination logics, flip-flops (FFs) for constructing a sequential circuit, and a selector. An LB that is equipped with multiple BLEs is called a cluster-based LB. Connections between LBs are implemented by switch blocks (SBs) and connection blocks (CBs). The routing architectures include different lengths of wires. Figure 3 shows an example of the segment length structure. Single lines indicate connections between neighboring SBs, double lines those between SBs that are two LBs apart, and quad lines those between SBs that are four LBs apart. In this way, high-speed signal propagation is made possible by directly connecting distant SBs.

3.2. **Definition of the FPGA placement problem.** In this section, we provide an overview of FPGA placement as a preliminary step to define the placement problem. Figure 4 shows an overview of the placement process. The netlist contains information about the logic elements and the connection relations between them, as shown in Figure 4(a). The placement shown in Figure 4(b) is set by a one-to-one mapping between the

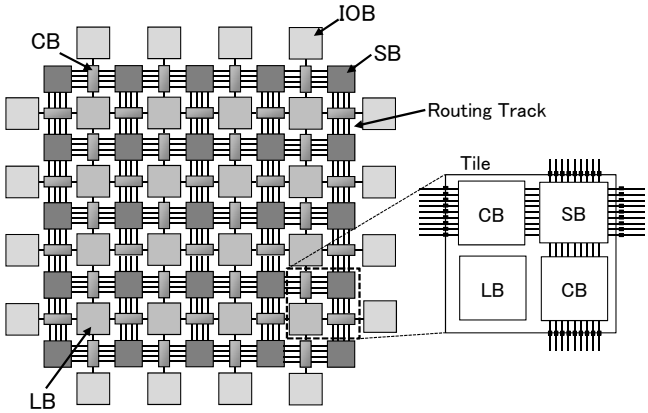

FIGURE 1. Island-style FPGA

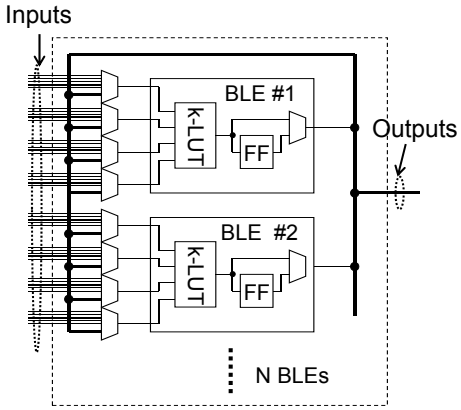

FIGURE 2. Logic cluster structure

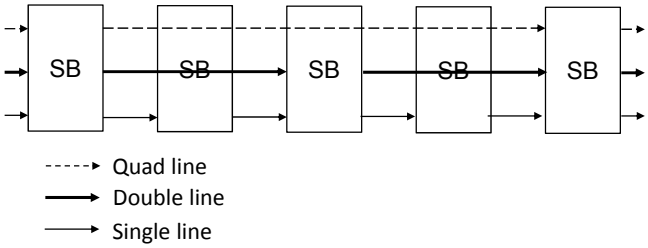

FIGURE 3. Example of wiring segmentation distribution

netlist and the logic elements in the FPGA. Next, we define the placement problem. Let the set of logic elements in the FPGA be $(c_0, c_1, \ldots, c_{l-1}) \in C$, where l is the total number of logic elements. The circuit to be implemented in the FPGA is given by the netlist $G(M, E)$. Here, $(m_0, m_1, \ldots, m_{n-1}) \in M$ are the n logic modules contained in the netlist, where each module is a circuit that can be implemented as a single logic cluster or IOB. The set $(e_0, e_1, \ldots, e_{u-1}) \in E$ contains connections representing the u signal wires that exist in the netlist.

The FPGA placement problem is to find a mapping of the logic modules M onto the logic elements C in a way that satisfies the constraints imposed by the given netlist $G(M, E)$ and minimizes the score on some objective function. As a result, valid mappings

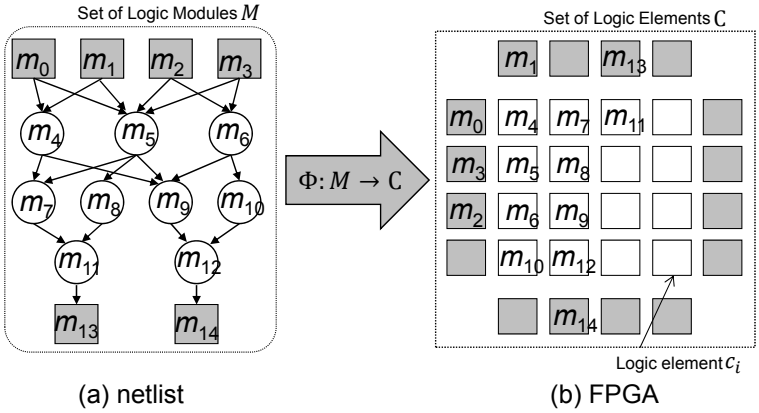

FIGURE 4. FPGA placement problem: (a) graph representation of netlist, (b) result of FPGA placement

satisfy the following conditions.

$$\phi : M \to C, \quad m \to c \tag{1}$$

Placement is performed with the primary goal of minimizing the wiring length. However, it is difficult to calculate a realistic value of the total wiring length for each trial. Therefore, a cost function that estimates the virtual wiring length, such as by using a bounding box [4], is typically chosen for the objective function.

4. Proposed Placement Algorithm.

This section describes the proposed SOM-based FPGA placement algorithm.

4.1. Overview of proposed placement algorithm.

Our novel SOM-based placement algorithm consists of an input layer and an output layer, as in traditional SOMs [5]. The input layer consists of input vectors, which represent the connection relations in the netlist. In the output layer, correlations are made with the nodes ('neurons' in the model) that reference logic elements in the FPGA. The SOM-based placement method consists of a competitive process that determines the winning neuron (designated as the best match unit, or BMU) and a cooperative process that uses the winner for learning. In the cooperative process, learning is initially performed over a wide range, with the range becoming gradually narrower as learning progresses. Since this method gradually changes from global optimization to local optimization, it has the advantage of being resistant to becoming trapped in only locally optimal solutions. In addition, this method performs batch learning, which considers that the input set is determined in advance and assumes convergence of the solution.

4.2. Input vectors.

To improve the placement quality, it is necessary to accurately apply the features of the netlist to the input vectors. Furthermore, it is preferable for the number of dimensions of the input vectors to be as small as possible to ease the computational load. In related research [6, 7], the input has been represented as an adjacency matrix. However, as the size of the netlist becomes large, the input vectors form a huge but sparse binary matrix. This must be avoided to conserve memory and reduce computational load.

Therefore, we use the Shimbel index [17] for representing features of the input vectors. The Shimbel index is an index of adjacency in a graph. In the field of network analysis, it is used to determine the graph distance from a specially designated node, known as the observation point, to a target node. First, we define the set of input vectors that is generated by the netlist. The netlist can be treated as a directed graph with edge

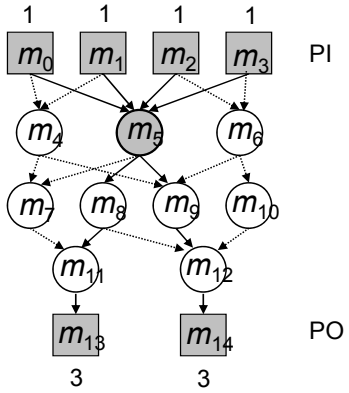

FIGURE 5. Example of input vector on module m_5

directions determined by connection and each of the modules represented as a vertex. For modules $m_i, m_j \in G$ in a directed graph G, the shortest graph distance from m_i to m_j is denoted by $d(m_i, m_j)$. Here, we define the set of inputs and outputs in the netlist as P, which is given by $(p_0, p_1, \ldots, p_{k-1}) \in P$ when the total number of primary inputs (PIs) and primary outputs (POs) is k. The distance of a logic module m_i from p_k can be expressed as $d(p_k, m_i)$, with d agreeing with the Shimbel index everywhere. Thus, each logic module m_i has k distance relations in the set of inputs and outputs. In other words, the input vectors that correspond to a logic module m_i are defined as follows, where $\vec{m_i}$ is the k-dimensional input vector of m_i and n is the total number of logic modules.

$$\vec{m_i} = [d(p_0, m_i), \ldots, d(p_{k-1}, m_i)] \quad (0 \leq i \leq n-1) \tag{2}$$

When the distance is found by using the Shimbel index, we trace along the fan-out direction (when a PI is taken as the observation point) or along the fan-in direction (when a PO is taken as the observation point).

Creation of the input vectors is described with reference to Figure 5. At the logic module m_5, there are four input directions, PIs (m_0, m_1, m_2, m_3), and two output directions, POs(m_{13}, m_{14}). Applying the formula for Shimbel index, the distance from each of m_0, m_1, m_2, and m_3 is 1; the distance from each of m_{13} and m_{14} is 3; the m_5 input vector is expressed as follows.

$$\begin{aligned} \vec{m_5} &= [d(m_0, m_5), d(m_1, m_5), d(m_2, m_5), d(m_3, m_5), d(m_{13}, m_5), d(m_{14}, m_5)] \\ &= [1, 1, 1, 1, 3, 3] \end{aligned}$$

Although this example was calculated by taking the Shimbel index between nodes to be 1, it could also be calculated with weights applied to the nodes. In the above method, because the circuit inputs and outputs are taken as the observation starting points, the input vector is k-dimensional and does not depend on the number of modules. Therefore, it is anticipated that the computational load and memory usage will be less than when using an adjacency matrix.

4.3. **Competition and cooperation.** The SOM output layer consists of layers of reference nodes, and each reference node has the same number of dimensions as the input vector. Taking the total number of PIs and POs as k, a vector of reference nodes, $\vec{c_i}(t)$, can be expressed as follows.

$$\vec{c_i}(t) = [c_{i0}, c_{i1}, \ldots, c_{ik-1}] \quad (0 \leq i \leq l-1) \tag{3}$$

Here, t is a discrete time coordinate. In an SOM, the reference vectors start in a randomly initialized state, and learning progresses by iteratively having vectors compete.

The competition stage determines the winner and the cooperation stage updates the neurons adjacent to the winner. During each competition stage, all neurons accept all input vectors $\vec{m}_j$ simultaneously. At this point, the reference node that is the most similar to the input vector for the reference vector c_i is designated as the BMU for that stage and is represented by the following formula.

$$c_{BMU_j}(t) = \underset{i}{\operatorname{argmin}} \mid\mid \vec{m}_j - \vec{c}_i(t) \mid\mid \tag{4}$$

where c_{BMU_j} is the BMU that corresponds to $\vec{m}_j$.

In the competition stage, the distances of each input vector from all the reference nodes are calculated, and the reference node corresponding to the shortest distance is taken as the BMU. The reference node that is selected as the BMU is placed at the logic module having the corresponding input vector. Minkowski distance is used for calculating the distance during learning. The Minkowski distance $dist$ between an input vector m_j and a reference vector c_i is calculated as

$$dist = \sum^{k} \mid \vec{m}_j - \vec{c}_i \mid^5 \tag{5}$$

When the distances to all the input vectors are calculated and the BMU node is selected, the SOM learning process transits to the cooperation stage. In the cooperation stage, the reference vector is selected as the BMU, $c_{BMU_j}(t)$, and its neighboring neurons are updated according to the following equation to move closer to the input data.

$$\vec{c}_i(t+1) = \frac{\sum_{j=0}^{n-1} h_{j,i}(t) \vec{c_{BMU_j}}}{\sum_{j=0}^{n-1} h_{j,i}(t)} \quad (0 \leq i \leq l-1) \tag{6}$$

$$h_{j,i}(t) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(t)}\right) \tag{7}$$

Here, $h_{j,i}$, the proximity function, consists of an update range $\sigma(t)$ and a Gaussian function; $d_{j,i}$ is the distance between the jth BMU node, $\vec{c_{BMU_j}}$, and the reference node c_i. When a simple 4-neighbor network topology is used for the output layer, the Manhattan distance can be used. The value of σ is large when learning starts but decreases monotonically as learning progresses. It is defined by the following formula.

$$\sigma(t) = \sigma_0\left(1 - \frac{t}{T}\right) \tag{8}$$

Here, T is the total number of learning steps and σ_0 represents the initial value. Note that the connections and weights between reference nodes can be freely defined.

Figure 6 shows pseudocode for the proposed SOM-based placement method. Taking the number of logic elements as n, the computational complexity is $O(n)$. Since the computational complexity of the VPR algorithm is $O(n^{4/3})$ [18], it is clear that the computational complexity of the proposed method is an improvement.

5. **Modeling of the FPGA Routing Structure.** This section describes how the routing structure in the FPGA is modeled as the output layer. We take the LBs and IOBs in the FPGA as the reference nodes in the output layer and the connection relations between them as the edges between the reference nodes. Further, the edges between the reference nodes are weighted according to the wiring delays between logic elements. Values normalized by the delay of a single segment are used as the weights of the edges. Taking the reference vectors corresponding to c_i and c_j, there are some candidates for connection.

```
/*program SOM placement algorithm*/

for (y ← 0; y < array_size+2; y++){
    for (x ← 0; x < array_size+2; x++){
        for (i ← 0; i < n; i++)
            M [y][x][i] = RANDOM VALUE; /* Set the initial value in output node*/
    }
}

for (t ← 0; t < t max; t++){ /*Set the number of leaning*/
    for (u ← 0; u < n; u++){
        D* = MAXINT; /*Initialize similarity*/
        for (y ← MIN_Y; y < MAX_Y; y++) {
            for (x ← MIN_X; x < MAX_X; x++) {
                for (i ← 0; i < DIM; i++ ) { /*Calculate BMU node*/
                    D+ = abs(X[u][i] − M[y][x][i])^5; /*Calculate similarity*/
                    if (D* >= D) {
                        D* = D;
                        x* = x;
                        y* = y;
                    }
                }
            }
        }
    }
    for (y ← MIN_Y; y < MAX_Y; y++) {/*±nσ update*/
        for (x ← MIN_X; x < MAX_X; x++) {
            for (i ← 0; i < n; i++ ) {
                h = exp(−1 ∗ (abs(x* − x) + abs(y* − y))^2/(2 ∗ σ^2));
                for (j ← 0; j < DIM; j++)
                    M[y][x][j] = h_dummy ∗ X_dummy[j];
                for (j ← 0; j < DIM; j++)
                    M[y][x][j] += h*X[i][j];
            }
        }
    }
}

placement(M);/* Assign the logic element on LBs*/
```

FIGURE 6. Pseudo-code in SOM placement

From the above, the shortest distance $delay_{i,j}$ between c_i and c_j can be formulated as follows.

$$delay_{i,j} = \min_{0 \le m \le N-1} \left(d^0_{(i,j)}, d^1_{(i,j)}, \ldots, d^{N-1}_{(i,j)} \right) \tag{9}$$

If there are N types of delay paths in the target routing structure, then the minimum distance along a path that includes each of the delays is $delay_{i,j}$. This is because the path that minimizes the delay should be selected for FPGA placement. Two types of architectures are discussed here as illustrative examples.

5.1. **Hierarchical routing structure.** Figure 7(a) shows a fault-tolerant FPGA, which we proposed in a previous paper [3]. This architecture has a hierarchical structure in

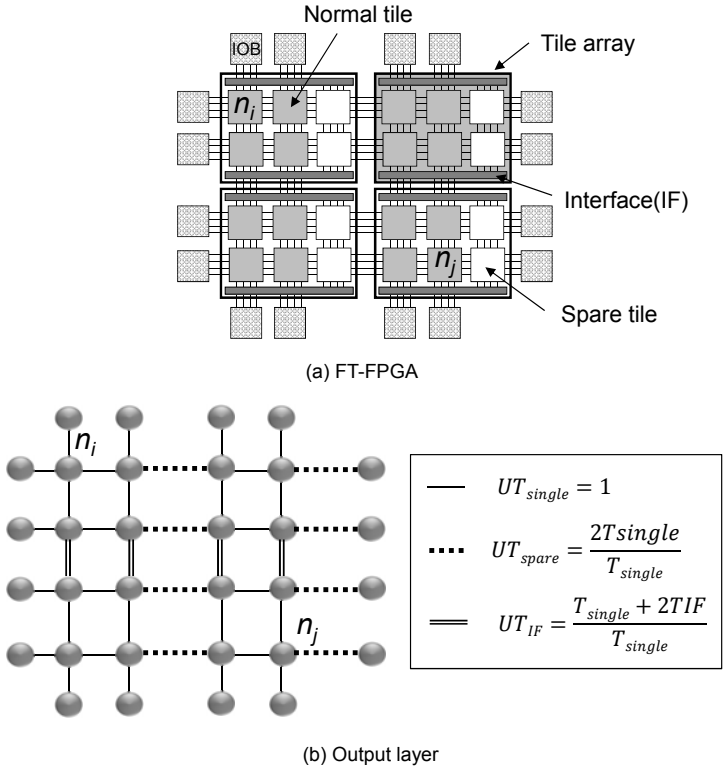

FIGURE 7. FT-FPGA: (a) FT-FPGA structure, (b) output node of FT-FPGA

which a spare tile for recovering faults and an interface for shifting the circuit structure to a spare tile (by column) are added to each region, or tile array. During placement, it is necessary to consider a network that includes spare tiles and interface delays. Figure 7(b) shows the SOM output layer that corresponds to the setup shown in Figure 7(a). Stepping between tile arrays in the vertical direction involves passing through two interfaces; thus, two interfaces worth of delay are added to edges that correspond to boundary areas. We represent the wiring delay of a single segment by T_{single} and the delay of passing through an interface by T_{IF}. For stepping between tile arrays in the horizontal direction and passing through the spare tile, the common weight of such edges, UT_{spare}, is defined as follows.

$$UT_{spare} = \frac{2T_{single}}{T_{single}} = 2 \tag{10}$$

Similarly, the common edge weight for stepping between tile arrays in the vertical direction via interfaces, UT_{IF}, is defined as follows.

$$UT_{IF} = \frac{T_{single} + 2T_{IF}}{T_{single}} \tag{11}$$

The distance $delay_{i,j}$ between neurons c_i and c_j in the diagram is thus given by the following equation.

$$delay_{i,j} = 4 + UT_{spare} + UT_{IF} \tag{12}$$

5.2. **Three-dimensional stacked structure.** In this section, we take the example of a three-dimensional stacked structure, with the structure as shown in Figure 8(a). In this structure, two layers of regular FPGA are stacked on top of each other, with a three-dimensional SB used to add vertical connections in order to allow connections between layers, and through-silicon vias (TSVs) used for the wiring in the vertical direction [2].

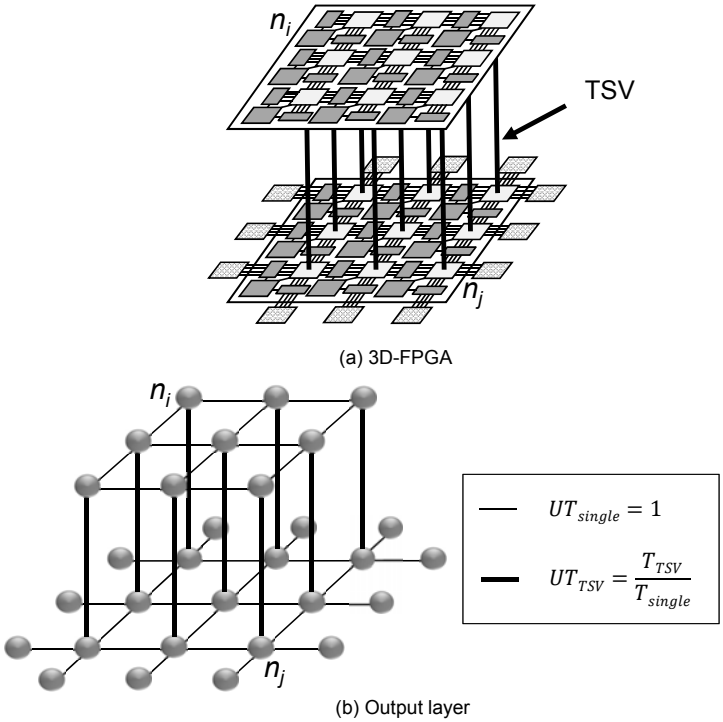

FIGURE 8. 3D-FPGA: (a) 3D-FPGA structure, (b) output node of 3D-FPGA

Figure 8(b) shows the SOM output layer that corresponds to Figure 8(a). First, the array of output nodes is extended in the vertical direction to form three-dimensional stacked layers. Next, a network corresponding to the TSVs is added, and weights are added to account for the TSV delays. The delay of a single line is represented by T_{single}, and the delay from one TSV is represented by T_{TSV}. The value of the TSV delay is normalized by the delay value of a single line to UT_{TSV}, which can be calculated as follows.

$$UT_{TSV} = \frac{T_{TSV}}{T_{single}} \tag{13}$$

The distance $d_{i,j}$ between output nodes n_i and n_j in the diagram is given by the following equation.

$$delay_{i,j} = 4 + UT_{TSV} \tag{14}$$

6. **Evaluation.**

6.1. **Evaluation target.** The two different architectures described in Section 5 are used for the evaluation. The values after layout are used as the delay values for determining the edge weights. The number of LUT inputs is set to six, and the number of logical clusters is set to four. The SB wiring topology is set to Wilton type [19], and the flexibility is set to $F_s = 0.3$. The fault-tolerant FPGA shown in Figure 7 is set to have a tile array size of 2×2.

6.2. **Evaluation procedure and evaluation environment.** In this section, we report the result of experimentally evaluating two types of FPGA structure: hierarchical fault-tolerant FPGA and three dimensional FPGA. Figure 9 shows the design flow of the evaluation in the proposed tool, SOM-Placer. Topology mapping is performed on the MCNC benchmark circuit [20] by using ABC [21], and clustering is performed using T-VPack [22]. Placement is performed on the obtained netlist by using SOM-Placer, VPlace (a VPlace [4] placement tool developed by Toronto University), and TPR [23] (a placement

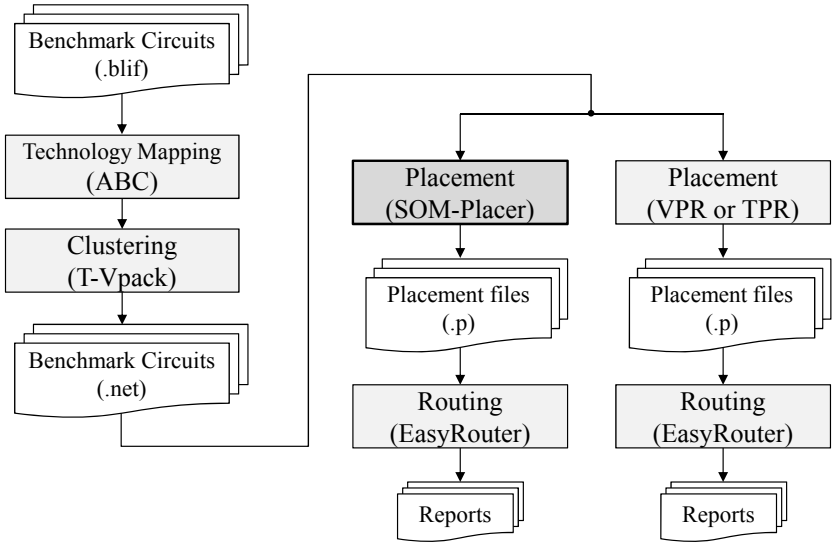

FIGURE 9. Evaluation flow

tool for three-dimensional FPGAs). VPlace uses an SA-based placement method, and TPR uses a partitioning-based placement method that relies on hMetis [24]. We believe that these benchmarks and placement tools are practical in FPGA academic region. Note that VPlace is used with modifications for hierarchical FPGAs. The placement files were routed by EasyRouter [8], which was previously developed by our research group, and the results were compared. EasyRouter is a routing tool that allows for architectures to be defined by writing C# scripts, which makes it easy to support various wiring structures. Note that SOM-Placer is coded in C++, whereas VPlace and TPR are coded in C. To account for the effect of the initial placement, the evaluation used the average values obtained from starting with 5 different seed values. The parameter σ_0 was initially set to 70% of the output layer size.

6.3. **Evaluation results and discussion.** Table 1 summarizes the evaluation results for the fault-tolerant FPGA. Compared with the SA-based VPlace, SOM-Placer increased the total wiring length by 14% and the maximum delay by 19% but decreased the execution time by 96%.

The wiring segments in the target three-dimensional stacked FPGA are treated as single lines, and the TSV delay value and wiring length are assumed to be the same as those for single lines. Furthermore, the number of TSVs is assumed to be the same as the number of horizontal channels. The evaluation results are summarized in Table 2. The existing TPR method is a partitioning-based placement tool that uses hMetis and is based on an algorithm that seeks to optimize wiring length. Compared with TPR, the proposed method increased the total wiring length by 2% but reduced the maximum delay by 12%. Furthermore, the execution time was reduced by 78%. Although the performance of SOM-Placer is low compared with SA-based VPR, execution time is greatly improved. Both the performance and the execution time of SOM-Placer are improved relative to partitioning-based TPR. In conclusion, our algorithm is efficient for exploring FPGA architectures by using relative evaluation.

7. **Conclusions.** In this paper, we proposed an SOM-based FPGA placement algorithm, which we have implemented as SOM-Placer. SOM-Placer facilitates easy modeling of new FPGA architectures without any limitations, which can significantly shorten the development cycle. In SOM-Placer, we consider the FPGA routing structure as the output layer

TABLE 1. Evaluation results of FT-FPGA

Circuit	Placement	Total number of wires	Delay (ns)	Execution time (s)
alu4	SOM-Placer	4,794	63.70	0.08
	VPR	4,575	53.16	3.34
apex2	SOM-Placer	11,329	69.29	0.18
	VPR	9,890	56.06	6.28
apex4	SOM-Placer	9,118	88.44	0.13
	VPR	7,237	75.68	5.49
ex5p	SOM-Placer	5,129	68.05	0.29
	VPR	4,887	53.42	5.45
ex1010	SOM-Placer	10,861	77.51	0.08
	VPR	9,343	74.11	6.04
misex3	SOM-Placer	5,584	63.64	0.08
	VPR	4,811	57.64	3.50
seq	SOM-Placer	10,645	82.69	0.59
	VPR	10,006	71.51	9.57
spla	SOM-Placer	35,983	142.44	0.92
	VPR	29,689	108.59	22.40

TABLE 2. Evaluation results of 3D-FPGA

Circuit	Placement	Total number of wires	Delay (ns)	Execution time (s)
alu4	SOM-Placer	4,708	9.46	0.04
	TPR	4,232	11.09	0.57
apex2	SOM-Placer	8,586	11.96	0.07
	TPR	8,262	12.29	0.92
apex4	SOM-Placer	7,338	11.24	0.14
	TPR	7,128	11.68	0.79
ex5p	SOM-Placer	4,178	10.06	0.21
	TPR	6,543	15.55	0.43
ex1010	SOM-Placer	8,461	11.19	0.12
	TPR	8,211	14.66	0.87
misex3	SOM-Placer	4,953	8.75	0.12
	TPR	4,533	9.73	0.55
seq	SOM-Placer	9,128	11.96	0.35
	TPR	10,547	15.61	0.99
spla	SOM-Placer	28,855	22.98	0.51
	TPR	25,685	22.81	2.42

of SOM. We evaluated the proposed FPGA design flow by testing two different devices to show its performance and extensibility. In this evaluation, two types of FPGA structures, a hierarchical fault-tolerant structure and a three-dimensional FPGA structure, were treated. Although the critical path delay of our method is 19% longer than that of VPlace, its execution time is 96% faster on average. In contrast, the critical path delay of our method is 12% shorter than that of VTR, and its execution time is 78% faster on average. In conclusion, our algorithm is efficient for exploring FPGA architectures by using relative evaluation.

REFERENCES

[1] M. J. Alexander, J. P. Cohoon, J. L. Colflesh, J. Karro, G. Robins and C. Science, Three-dimensional field-programmable gate arrays, *Proc. of the 8th Annual IEEE International ASIC Conference and Exhibit*, pp.253-256, 1995.

[2] A. Gayasen, V. Narayanan, M. Kandemir and A. Rahman, Designing a 3-D FPGA: Switch box architecture and thermal issues, *IEEE Trans. VLSI Systems*, vol.16, no.7, pp.882-893, 2008.

[3] M. Amagasaki, Q. Zhao, M. Iida, M. Kuga and T. SUeyoshi, Fault-tolerant FPGA: Architectures and design for programmable logic intellectual property core in SoC, *IEICE Trans. Information and Systems*, vol.E98-D, no.2, 2015.

[4] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, M. Fang and J. Rose, VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling, *Proc. of ACM Symposium on FPGAs*, pp.133-142, 2009.

[5] T. Kohonen, The self-organizing map, *Proc. of IEEE*, vol.78, pp.1464-1480, 1990.

[6] M. Maniatakos, S. Xu and W. L. Miranker, Constraint-based placement and routing for FPGAs using self-organizing maps, *Proc. of IEEE International Conference on Tools with Artificial Intelligence*, pp.465-469, 2008.

[7] M. S. Zamani and G. R. Hellestrand, A neural network approach to the placement problem, *Proc. of ASP-DAC*, pp.413-416, 1995.

[8] Q. Zhao, K. Inoue, M. Amagasaki, M. Iida and T. Sueyoshi, FPGA design framework combined with commercial VLSI CAD, *IEICE Trans. Information and Systems*, vol.E96-D, no.8, pp.1602-1612, 2013.

[9] S. S. Kim and C. M. Kyung, Circuit placement on arbitrarily shaped regions using the self-organization principle, *IEEE Trans. CAD*, vol.11, no.7, 1992.

[10] R. I. Chang and P. Y. Hsiao, VLSI circuit placement with rectilinear modules using three-layer force-directed self-organizing maps, *IEEE Trans. Neural Networks*, vol.8, no.5, 1997.

[11] C. X. Zhang and D. A. Mlynski, Mapping and hierarchical self-organizing neural networks for VLSI placement, *IEEE Trans. Neural Networks*, vol.8, no.2, 1997.

[12] M. S. Zamani and F. Mehdipur, Using Kohonen map for the placement of regular VLSI designs, *Proc. of International Conference on Computational Intelligence and Multimedia Applications*, pp.65-69, 1999.

[13] C. Aykanat, T. Bultan and I. Haritaoglu, A fast neural-network algorithm for VLSI cell placement, *Neural Netw.*, pp.1671-1684, 1998.

[14] Altera Corporation, *StraixIII Device Handbook*, 2007.

[15] V. Betz, J. Rose and A. Marquardt, *Architecture and CAD for Deep Submicron FPGAs*, Kluwer, New York, NY, 1999.

[16] Xilinx, *VIrtex-5 User Guide*, 2008.

[17] A. Shimbel, Structural parameters of communication networks, *Bulletin of Mathematical Biology*, vol.15, no.4, pp.501-507, 1953.

[18] A. Marquardt, V. Betz and J. Rose, Timing-driven placement for FPGAs, *Proc. of FPGA*, pp.203-213, 2000.

[19] G. G. Lemieux and D. M. Lewis, Analytical framework for switch block design, *Proc. of FPL*, pp.122-131, 2002.

[20] K. McElvain, IWLS'93 benchmark set: Version 4.0, *The MCNC International Workshop on Logic Synthesis*, 1993.

[21] A. Mishchenko et al., *ABC: A System for Sequential Synthesis and Verification*, http://www.eecs.berkeley.edu/ alanmi/abc/, 2009.

[22] A. Marquardt, V. Bets and J. Rose, Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density, *Proc. of FPGAs*, pp.37-46, 1999.

[23] C. Ababei, H. Mogal and K. Bazargan, Three-dimensional place and route for FPGAs, *IEEE Trans. CAD of Integrated Circuits and Systems*, vol.25, no.6, pp.1132-1140, 2006.

[24] N. Selvakkumaran and G. Karypis, Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization, *IEEE Trans. CAD of Integrated Circuits and Systems*, vol.25, no.3, pp.504-517, 2006.