2008 20th IEEE International Conference on Tools with Artificial Intelligence

Constraint-Based Placement and Routing for FPGAs using Self-Organizing Maps

Michail Maniatakos[1], Songhua Xu[2], and Willard L. Miranker[2]

1: Electrical Engineering Department, Yale University, New Haven, CT 06520 USA
2: Department of Computer Science, Yale University, New Haven, CT 06520, USA

{michail.maniatakos, songhua.xu}@yale.edu, miranker@cs.yale.edu

Abstract

Field-programmable gate arrays (FPGAs) are becoming increasingly popular due to low design times, easy testing and implementation procedures and low costs. FPGAs placement and routing are NP-complete problems dealt well with modern tools using heuristic algorithms. As modern FPGAs increase in size and also new capabilities, such as Run-Time Reconfiguration (RTR), are introduced, the complexity of these problems is greatly increased. In this paper we approach both problems using a modified version of Kohonen Self-Organizing map. The algorithm, consisting of four phases, takes into consideration constraints that may apply to the FPGA design (such as I/O pins, resource constraints like global clock etc). The modified algorithm yields a good topological map of the design to be placed, minimizing the average distance between connecting logic blocks.

Index Terms—FPGA, self-organizing feature map, placement, routing, constraints

1. Introduction

Field-programmable gate arrays (FPGAs) are semiconductor devices, which consist of programmable components called "logic blocks". These blocks can be programmed to perform different functions (such as AND, OR) or to store data. Logic blocks connect through wires running all over the FPGA board. Many connected logic blocks create an FPGA design that performs a specified operation. An FPGA board can be reprogrammed, while its main counterpart, Application Specific Integrated Circuits (ASICs) are manufactured for a specific application and their operation cannot change. The main disadvantage of using FPGAs compared to ASICs is that FPGAs are pre-manufactured so their cost increases linearly for every board, while ASICs have a huge initial cost but production cost for larger quantities increases slowly. Also, FPGAs are slower and more power consuming. On the other hand, an FPGA has no initial manufacturing cost, it has low recurring engineering costs and is significantly cheaper than ASICs for small quantities.

A user programs the board using High-level description languages (HDLs); the output code is converted by tools to logic blocks. The exact place that a logic block will be stored is defined through the "placement" procedure. Similarly, the wire tracks that will be used to connect logic blocks are defined through the "routing" process. Placement and routing are often interactive because good routing is highly dependent on good placement.

In this paper we approach placement and routing processes using a modified version of Kohonen's Self-Organizing Map algorithm defined in [1]. Specifically, we modify the notion of a winning neuron and which neurons are updated.

In Section 2 the Self-Organizing map algorithm is given, while in Section 3 the placement and routing process of an FPGA is described. In Section 4 we introduce our approach on FPGA placement and routing using SOMs. Section 5 presents a case study where our algorithm is used to place a complex design on an FPGA board. Finally, in Section 6 performance figures are presented along with a discussion of the results of the algorithm.

2. Self-Organizing Map Algorithm

Self-organizing maps (SOMs) are a special class of artificial neural networks, based on both competitive and cooperative learning. The main purpose of the SOM is to transform an incoming signal pattern of arbitrary dimension into a one or two dimensional discrete map. Each neuron in the map is fully connected to all source nodes in the input layer [2]. SOM training is based on two basic principles:

Competition. The prototype vector that is most similar to an input data vector (where similarity can be defined in terms of Euclidean distance) "wins" the competition and is then transformed in order to be more "similar" to the input vector. By means of this process the algorithm learns the position of input data.

1082-3409/08 $25.00 © 2008 IEEE
DOI 10.1109/ICTAI.2008.55

Cooperation: Besides the winning neuron, all its neighbors (where neighborhood radius must be defined by some parameter) are moved towards the input data vector as well. With cooperation, the map self-organizes.

A detailed description of the SOM algorithm is presented in [2].

3. FPGA placement and routing

An FPGA board consists of several logic blocks. In a simple FPGA design, a logic block contains a Look-Up Table (LUT) and a flip-flop, so it could either perform a specific function defined in the LUT or store a single bit. A LUT contains the truth table of the function implemented. The output of a logic block goes to the input of another logic block through wiring tracks, unless Input/Output connections are specified (usually at the edges of the board). Placement and Routing are essential parts of a typical application synthesis flow.

During *Placement*, all packed blocks of logic have to be assigned to specific block locations in the prefabricated two-dimensional array of the FPGA board. Ideally, perfecting localized routability in each subsection of the board would yield the best placement, but given the distributed nature of interconnect and dependencies caused by segmentation this approach becomes infeasible. So a metric to evaluate an algorithm performance is the wiring length of the placement. Placement is an NP-complete problem.

After placing the logic blocks in specific places, these blocks must be connected using routing segments and switches to create a connecting path through FPGA's routing tracks. This is called *Routing* and is likewise an NP-complete problem, because an FPGA has a limited number of wiring tracks running around the board. The most commonly used algorithms for routing are the maze-routing algorithms [3],[4]. These algorithms are based on Dijkstra's shortest path algorithm. The placement and routing processes must not be separated; a specific placement may not be routable at all, while a slightly different one may yield a good quality routing.

During placement and routing procedures, FPGAs constraints have to be considered. A straightforward example of a constraint is a logic block that receives its input from outside the board, so it has to be placed on the I/O blocks (usually at the edges of the board). So the route and place algorithms must take into consideration the special nature of this logic block. Another example of a constraint comes from switch pins that exist in specific places on the board where the corresponding logic block must be placed appropriately to receive the switch state.

Besides such constraints, there are some constraints that affect design's performance and not design correctness. For example, in an FPGA design all clocked elements (such as Flip-Flops) share the same global clock; so the clock signal must arrive at clocked elements simultaneously and as fast as possible (for better performance). Thus, these elements must be placed near the clock buffers that produce the clock signal.

4. Modified SOM algorithm

We devise an algorithm to achieve good placement and routing on the FPGA board. Good placement yields good routing and vice versa.

We modified part of the Kohonen SOM algorithm in order to handle possible constraints of the FPGA design. The lattice we use for the SOM algorithm corresponds to the real layout of the design we want to place and route. So, a neuron in the lattice represents a logic block that has to be placed in a logic block of the FPGA, while synapses represent the connections between logic blocks (a logic block can have up to four connections). For example, for the simple design shown in Fig. 1, the lattice that would be produced is show in Fig. 2.

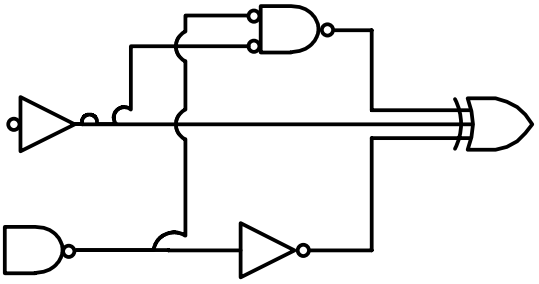

Figure 1. Simple design to be placed

Constraints that can be applied in the algorithm fall in two different categories:

Strict constraints: Constraints in this category are defined as conditions that *must* hold in order for the design to work properly. For example, a specific logic block must be placed on a specific block (e.g. an I/O pin).

Relaxed constraints: Violation of constraints in this category will not affect the correctness of the design but its performance. The modified algorithm may violate some relaxed constrains in order to achieve a better placement and routing. For example a logic block should be placed as near as possible to an FPGA resource. The final distance between the logic block and the resource affects the performance of the design. Generally, we can regard these types of constraint as *resource race constraints*, because multiple elements have to be placed close to a specific resource.

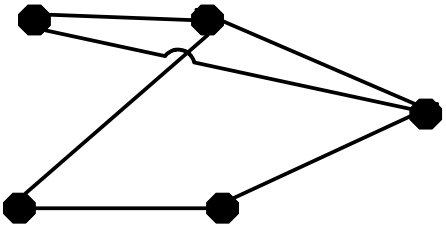

Figure 2. Lattice for sample circuit of Fig. 1

A constraint has an *importance* property, graded from 1-10, where 10 implies great importance. Strict constrains are automatically assigned a value of 10. This property helps the algorithm evaluate constraint importance and drives the algorithm to prioritize the constraints to be considered, optimizing the quality of the final placement.

The ultimate goal of this algorithm is to minimize distances between interconnecting logic blocks, so as to maximize design performance. The algorithm consists of four discrete phases and an initialization phase.

Algorithm Initialization: We first define the lattice of the circuit to be placed (as described in Fig. 3). A logic block (LB) is represented by a neuron and synapses between neurons represent the connections between the logic blocks. So if the design has M logic blocks to be placed we have the following set of neurons in the lattice labeled arbitrarily 1, …, M.

These M neurons connect through the MxM connection matrix C where:

$$C_{ij} = \begin{cases} 1, if\ LBs\ i\ and\ j\ are\ connected \\ 0, otherwise \end{cases} \quad (5)$$

A strict constraint is defined by three values [s_1, s_2, s_3], where s_1 and s_2 define the coordinates where the logic block must be placed, and s_3 is the index of the logic block that has to be placed in the position [s_1, s_2] on the FPGA board. For example, the set [1, 1, 5] specifies that the logic block 5 has to be placed in the [1, 1] position of the FPGA board.

Similarly, a relaxed constraint is defined by two values [c_1, c_2], where c_1 is the index of the logic block and c_2 is the index of the resource. For example, the set [2, 3] implies that block 2 must be placed close to resource 3.

A resource is defined by three values [r_1, r_2, r_3], where r_1 and r_2 are the X-Y coordinates of the resource and r_3 specifies its importance property.

Finally, N denotes the dimension of the squared FPGA board where the logic blocks must be placed (for example for N=20 we have a board with 20x20 = 400 positions for logic blocks).

Initialization of the algorithm consists of placing the lattice randomly on the FPGA board, employing a uniform distribution.

Phase 1 - Constraints set: During this phase we place the strict constrained logic blocks in the exact coordinates defined by design constraints. During following phases, these neurons are not involved in the competitive or cooperative process of the self-organizing map (they remain fixed throughout the whole process of ordering and convergence).In case of a conflict, we move the logic block to the nearest unoccupied block.

Phase 2 - Resource Competition: In the second phase the algorithm considers the relaxed constrains set by the FPGA design. During this phase, the algorithm's effort is to move the logic blocks (neurons) close to the resources in order to optimize placement quality.

An SOM self-organizes based on input vectors, so we have to generate these vectors. For each resource, one input vector is generated. For example, if we have a resource in [2, 2] position on the board, then a two-dimensional input vector is generated in the same position ([2, 2]). Fig. 3 presents the feature map using a resource placed at the bottom left of the FPGA board.

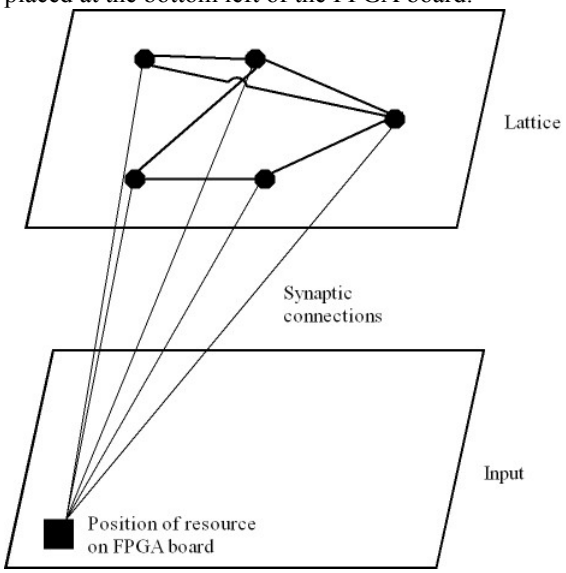

Figure 3. Self-organized feature space

Using this method, we drive winning neurons closer to resources. Only constrained neurons are allowed to win, so they will gradually move closer to the resources. The probability that an input vector will be selected during sampling is directly proportional to the importance property defined for each resource. So the modified algorithm for this phase is the following:

1. *Initialization*. The values of the initial weight vectors are copied from the weight vectors of phase 1 (e.g. we continue using the same feature map)
2. *Sampling*. We choose a sample x from the input space, with probability directly proportional to the

importance of this vector. Thus, more important resources will be sampled more often than others.

3. *Similarity matching*. We find the best-matching neuron i(x), where i is a neuron that competes for the resource. Therefore, only racing neurons are allowed to win. The best matching criterion is Euclidean distance:

$$i(x) = \arg\min_k \|x(n) - w_k\|, i, k \in R_L \quad (6)$$

where R_L is the subset on neurons that compete for Resource L.

4. *Updating*. We adjust the synaptic weight vectors of all neurons using the following formula:

$$w_j(n+1) = w_j(n) + \eta_1(n) h_1(x(n) - w_j(n)) \quad (7)$$

where learning rate parameter $\eta_1(n)$ gradually decreases over time, and h_1 is the neighborhood constant; so neighborhood radius remains constant throughout the algorithm.

5. *Continuation*. These steps are repeated until no noticeable changes in the map are observed.

At the end of this phase the constrained neurons will be closer to the desired resources, while the rest of the map will still be unordered. The synaptic weights of these neurons, like strict constrained neurons presented in Phase 1, won't be updated during subsequent phases. Phase 2 is repeated for every resource defined in the FPGA design.

Phase 3 - Ordering and Convergence: During this phase, placement is finalized using the Kohonen SOM algorithm presented in Section II. Again, the input data vectors must be defined. Similarly to Phase 2, one input vector is created for every resource and for every strict constrained neuron (placed during Phase 1). So the final input vector set consists of two dimensional vectors that represent the position of the resources and strict constrained neurons.

When this phase completes, a good geometric approximation of the design to be placed will be produced. The result is not guaranteed to be optimal; placement and routing are NP-complete problems.

Also, due to the mathematical nature of the SOM algorithm, logic blocks' coordinates will have real values; this is not allowed, because logic blocks should be placed on distinct logic block places. This misalignment defect is targeted in the next and final phase.

Phase 4 - Quantization: During this final phase logic blocks are moved to the nearest logic block location. If during this quantization phase more than one logic block is to be placed in a single location, only the nearest logic block is allowed to move there; in case of a tie, the first in the list is moved there. Then the rest of the blocks are placed in the nearest unoccupied block using Euclidean distance.

After this final phase we get a design that fulfills the constraints of the FPGA design while achieving good placement and routing. In most of the cases the modified SOM algorithm manages to reduce the distances of the initial random placement up to 10 times. A specific example of using the algorithm follows.

5. Case study

In this section the modified SOM algorithm is demonstrated. Assuming we have an FPGA board of 10x10 logic block locations, we will attempt to place and route 30 randomly connected logic blocks (keeping the limit of up to 4 connections though).

We add three strict constraints in our case study:

```
[ 1  4  3]
[10  2 26]
[ 5  5 15]
```

The first two values are the X-Y coordinates of position that the logic block must be placed in, while the third value specifies which logic block will be placed there.

We also add one resource located in four different places with different importance properties:

```
[ 2  2 10]
[ 2  7 10]
[ 7  2  5]
[ 7  7  5]
```

In the above matrix the first two values in a row specify the X and Y coordinate of the resource while the third specifies its importance.

The logic blocks that will compete for this resource are defined in the following matrix:

```
[10 22 8 20]
```

The above matrix specifies that the logic blocks with index 10, 22, 8 and 20 will compete for the resource.

During initialization of the algorithm, we randomly place the logic blocks on the board. The initial summed distance between all logic blocks is 1345 units.

In phase 1 the strict constrained logic blocks are moved to their specified positions. In this case study the summed distance is slightly increased, but it could also be slightly decreased or remain the same.

Phase 2 is the resource competition phase. Logic blocks are placed close to the resources and they are evenly spaced among them. The summed distance in this stage was decreased to 273.44 units, which is very good considering the fact that the algorithm actually hasn't started final ordering and convergence.

In phase 3 the final ordering and convergence of the algorithm is performed. After 600 iterations the map

finally converged. The distance is much smaller (126.04) compared to the initial one (1345.00), which ensures us that the algorithm has greatly improved the placement and routing of the design.

During the 4th and final phase, we quantize the final positions of the logic blocks. This quantization yields increased distances (170 units) This is expected because close to resources there are more logic blocks, so conflicts will occur because of limited logic block locations. While conflicting logic blocks move to nearby places, summed distance increases.

6. Results and discussion

6.1 Performance Evaluation

In this section we will evaluate algorithm performance. Generally, the best measurement for placement's quality is the summed distance of logic blocks. We could also use other measurements, such as the mean quantization and the topological error.

We first focus on the performance of the algorithm for different iterations, using summed distance as our quality measurement. The summed distance of logic blocks in each phase will be calculated. By performing 20 different runs of the algorithm we get the distances presented in Fig. 4.

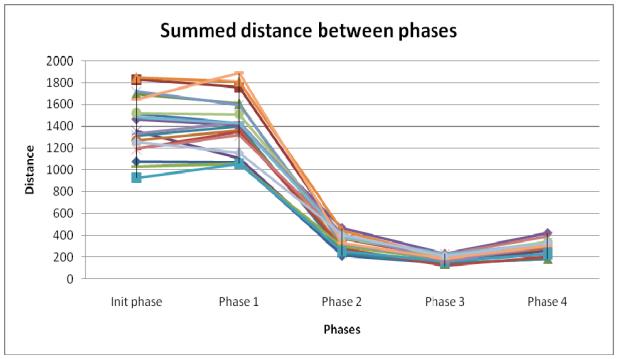

Figure 4. Summed distance between all phases of the algorithm for 20 runs

The first conclusion from this figure is that the algorithm exhibits similar performance for each run: great improvement during Phase 2, further improvement during final ordering and convergence during Phase 3, while during the Quantization Phase 4 summed distance are slightly increased. Also, Phase 3 has the least deviation compared to all other phases; so algorithm performance seems to be deterministic.

Another important conclusion is that the final placement is independent of the initial random placement; so no matter how good or bad the initial placement is, algorithm performance isn't affected.

Next consider evaluation of the average performance of the algorithm for many runs. We perform 100 iterations of the algorithm and calculate the average value of summed distance between phases. The results shown in Fig. 5 are clear; all iterations exhibit the same behavior as described previously.

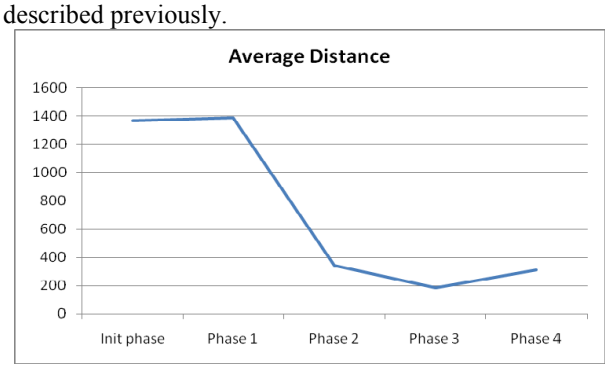

Figure 5. Average distance between phases for 100 runs

6.2 Discussion

The proposed methodology is an approach to placement and routing problems using a self-organizing neural network. An FPGA design can be viewed as a lattice that has to be organized optimally.

The modified algorithm produces a good topological approach of the design to be placed; it decreases distances between logic blocks and avoids intersecting wires by decreasing the topology error. It is possible that this layout may be fine-tuned locally (for example by using single logic block swaps and recalculating distances) but the overall design will adhere to the same topology.

We should also mention that the algorithm is slow, and for large designs it would probably require a large number of iterations to converge. Also the resource race phase results are not deterministic; so we can't predict with certainty the specifications of the placed layout. However, exploring all possible parameters for each training phase of the algorithm could yield greater certainty, trading-off an optimized placement.

10. References

[1] Kohonen T., The Self Organizing Map, Proceedings of the IEEE Vol. 78 No. 9 (1990) pp. 1464-1480

[2] S. Haykin, Neural Networks: A comprehensive foundation (Upper Saddle River, NJ: Prentice Hall, 1999)

[3] Tessier R., Fast Place and Route Approaches for FPGAs, PhD thesis, MIT, 1999

[4] C. Lee, An algorithm for path Connections and its Applications, IRE Transactions on Electronic Computers, Sept 1961