A GENETIC ALGORITHM FOR MIXED MACRO AND STANDARD CELL PLACEMENT

Theodore W. Manikas
Department of Electrical Engineering
The University of Tulsa
Tulsa, OK 74104

Marlin H. Mickle
Department of Electrical Engineering
The University of Pittsburgh
Pittsburgh, PA 15261

ABSTRACT

The objective of mixed macro and standard cell placement is to arrange components on a chip such that the resultant layout area and interconnection wire lengths are minimal. A common approach is to divide the problem into separate macro cell and standard cell placement problems. However, this approach ignores the relationships between the macro and standard cells, which can affect the quality of the final solution. This paper describes a genetic algorithm that uses the relationship information to determine a more efficient placement solution.

1. INTRODUCTION

An important part of VLSI circuit design is the placement process. Many contemporary systems use a combination of macro cells and standard cells in order to implement all the desired functions on a chip: this is the *mixed macro and standard cell* design style. In a typical design, there are usually many more standard cells than macro cells. Thus, before the placement procedure is performed, the standard cells are partitioned into groups, or *domains*, to reduce the total number of components to be initially placed. Contemporary mixed macro and standard cell placement tools use a top-down, multiple-stage approach to determine the arrangement of components on a chip [1-3]. During the first stage, *block placement*, the arrangement of the macro cells and domains is determined. The dimensions of each domain have not yet been defined, so they must be estimated before block placement can be performed. After the block placement stage has been completed, a *cell placement* stage is applied to each domain to determine the arrangement of its standard cells.

Unfortunately, there are often significant differences between the estimated domain dimensions and the actual dimensions after cell placement. When domain estimation errors occur, it becomes necessary to adjust the domain dimensions and repeat the placement process. For a large circuit with many domains, there may be many repetitions of this process before the estimation errors are within acceptable levels of tolerance. One way to address this limitation is to incorporate the relationship between the block and cell levels into the placement process. This paper describes the development of a *genetic algorithm* placement tool that uses this relationship to determine an optimal mixed macro and standard cell layout.

2. GENETIC ALGORITHM FOR PLACEMENT

Genetic algorithms are heuristic optimization methods whose mechanisms are analogous to biological evolution [4]. A genetic algorithm starts with a set of *genotypes*, which represent possible solutions for a given problem. Each genotype is assigned a *fitness value*, based on how well the genotype meets the problem objectives. Using these fitness values, genotypes are randomly selected to be *parents*. These parents form new genotypes, or *offspring*, via *reproduction*. Parent selection and reproduction operations continue for several iterations until the algorithm converges to an optimal solution.

For the placement problem, previous applications of genetic algorithms include Esbensen's method for macro cell placement [5] and GASP for standard cell placement [6]. Each method uses a particular genotype structure to represent the layout for a given design style. We developed GAP (Genetic Algorithm for Placement) for mixed macro and standard cell netlists. Our work builds upon previous efforts by expanding the genotype structures to handle both macro cell and standard cell layout.

First, GAP uses the hMetis program [7] to partition a netlist into a specified number of domains. Then, GAP uses a genetic algorithm to produce a valid mixed macro and standard cell layout. Figure 1 shows the structure of this genetic algorithm.

2.1.1 Genotype Structure

The partitioning process forms a *cell membership tree*: each leaf corresponds to a cell in the netlist, while the internal nodes represent domains. GAP constructs a *genotype template* by proceeding top-down on the cell membership tree. Each level of the tree has its own encoding to represent a valid layout for the particular level. An example of a GAP genotype is shown in Figure 2, while Figure 3 shows its corresponding layout. This genotype is for a small netlist with two macro cells (*A* and *B*), and ten standard cells (1 – 10) that are grouped into one domain (*C*). Thus, the genotype template tree has a system node (block level) with one domain node *C* (cell level).

First, an encoding is randomly generated for the system node. For the *block level* layout representation, GAP uses Esbensen's genotype structure: a *Polish expression* [8] is randomly generated to represent the block *topology* (relative locations), while a *bitstring* is randomly generated to represent the block *orientations*. In Figure 2, the Polish expression is *AB*+*C**, while the bitstring is 001 000 111. This means that macro cell *A* is below macro cell *B*, while domain *C* is to the right of *A* and *B*. Macro cell *A* has orientation 1 (rotated 90 degrees), macro cell *B* has orientation 0 (no rotation), and domain *C* has orientation 7 (rotated 270 degrees and reflected across the y-axis).

The *cell level* consists of standard cells within a domain. GAP uses the genotype structure of GASP [6]: an *ordered string* is randomly generated to represent the standard cell topology within the domain. In Figure 2, the ordered string for domain *C* is {1 2 3 4 5 | 6 7 8 9 10}. This means that domain *C* has two rows of standard cells: row 1 contains cells 1 – 5, while row 2 contains cells 6 – 10.

2.1.2 Fitness Evaluation and Parent Selection

Each genotype is evaluated according to a *fitness function*. This function determines the quality of the solutions that are represented by the genotypes, with respect to the problem objectives. For mixed macro and standard cell layout, the objectives are to minimize the layout area and interconnection wire lengths. Thus, each genotype must be decoded to yield an area cost and a wire length cost for the fitness function. The *area cost* is computed as the product of the layout height and width, while the *wire length cost* is computed as the sum of all wire lengths for the nets in the layout. Since nets will not be routed until after the placement process has been completed, the wire lengths are estimated using the half-perimeter wire length metric [9].

GAP uses the approach of Esbensen for fitness evaluation and parent selection. Genotypes are ranked based on Pareto dominance using the method of [10]; Whitley's rank-based selection method [11] is then used for parent selection based on these dominance-assigned ranks.

2.1.3 Reproduction

Reproduction consists of crossover and mutation. The crossover operator creates a new genotype, based on the structures of two parent genotypes. Since each parent genotype tree follows the structure of the genotype template, the offspring genotype tree must also follow this structure. The main difference between the offspring and its parents will be the encoding for each node on the genotype tree.

Figure 4 shows two parent genotypes (a) *P1* and (b) *P2*, and the resultant offspring (c) *Q*. The root of each genotype tree is the system node, and the node encoding is a combination of a Polish expression and a bitstring. For offspring Q, the corresponding root node is created by applying Cohoon's crossover operator [12] to the Polish expression and uniform crossover [13] to the bitstring.

The next node on the genotype trees is domain C; this has an ordered string encoding. Therefore, the corresponding node on the offspring genotype *Q* is created by applying cycle crossover [14]. An additional operation is to determine the number of rows *r* for the offspring *Q*. The number *r* is randomly generated: $1 \leq r \leq$ #cells. For this example, $r = 5$.

An offspring may also undergo *mutation*, which alters its structure. The mutation operator developed for GAP follows the same general principle as the crossover operator: traverse the offspring tree in a top-down manner and apply the appropriate mutation operation to each node encoding. For the system node, Wong's mutation operator [8] is applied to the Polish expression and bitwise mutation [5] is applied to the bitstring. At each domain node, pairwise mutation is applied to the ordered string [6].

3. EXPERIMENTAL RESULTS

GAP was compared against the previous top-down, multiple-stage placement approach. In order to provide a fair comparison, Esbensen's placement tool was used for the block placement stage of the top-down approach, while GASP was used for the cell placement stage. The methods were tested on the MCNC mixed macro and standard cell benchmark netlists *a3, g2*, and *t1* [15]. Each netlist was partitioned into ten domains, and ten trials were run for each method on each data set. Figure 5 shows the mean area response for each netlist, while Figure 6 shows the mean wire length response. Compared to the top-down, multiple-stage approach, GAP yielded an average of 27% improvement in layout area and an average of 10% improvement in layout wire length.

The reason for these improvements is most likely due to the elimination of the need for domain size estimation in GAP. Domain size estimation methods tend to overestimate dimensions in order to allow space to complete standard cell placement within the domain. Large domain dimensions result in large layout areas and increased interconnection wire lengths. In contrast, GAP is able to obtain the actual domain size since this information is encoded in its genotype structure. This eliminates the overestimation of domain sizes, and results in reduced layout areas and wire lengths.

4. ACKNOWLEDGEMENT

This work was partially supported by the University of Tulsa Faculty Development Summer Fellowship Program.

5. REFERENCES

[1] D. P. Mehta and N. Sherwani, "On the Use of Flexible, Rectilinear Blocks to Obtain Minimum-Area Floorplans in Mixed Block and Cell Designs," *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, pp. 82-97, 2000.

[2] A. Shanbhag, S. Danda, and N. Sherwani, "Floorplanning for Mixed Macro Block and Standard Cell Designs," in *4th Great Lakes Symposium on VLSI*, 1994, pp. 26-29.

[3] M. Upton, K. Samii, and S. Sugiyama, "Integrated Placement for Mixed Macro Cell and Standard Cell Designs," in *Proc. of the 27th ACM/IEEE Design Automation Conf.*, 1990, pp. 32-35.

[4] M. Mitchell, *An Introduction to Genetic Algorithms*: MIT Press, 1996.

[5] H. Esbensen and E. S. Kuh, "A Performance-Driven IC/MCM Placement Algorithm Featuring Explicit Design Space Exploration," *ACM Trans. on Design Automation of Electronic Systems*, vol. 2, pp. 62-80, 1997.

[6] K. Shahookar and P. Mazumder, "A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, vol. 9, pp. 500-511, 1990.

[7] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Domain," in *Proc. 34th ACM/IEEE Design Automation Conf.*, 1997, pp. 530-533.

[8] D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Design," in *Proc. of the 23rd ACM/IEEE Design Automation Conf.*, 1986, pp. 101-107.

[9] D. G. Schweikert, "A 2-dimensional Placement Algorithm for the Layout of Electrical Circuits," in *Proc. of the 13th ACM/IEEE Design Automation Conf.*, 1976, pp. 408-416.

[10] C. M. Fonseca and P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," in *Proc. 5th Inter. Conf. on Genetic Algorithms*, 1993, pp. 416-423.

[11] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," in *Proc. 3rd Inter. Conf. on Genetic Algorithms*, 1989, pp. 116-121.

[12] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Richards, "Distributed Genetic Algorithms for the Floorplan Design Problem," *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, vol. 10, pp. 483-492, 1991.

[13] G. Syswerda, "Uniform Crossover in Genetic Algorithms," in *Proc. 3rd Inter. Conf. on Genetic Algorithms*, 1989, pp. 2-9.

[14] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A Study of Permutation Crossover Operators on the Traveling Salesman Problem," in *Proc. International Conf. on Genetic Algorithms and their Applications*, 1987, pp. 224-230.

[15] CBL, MCNC benchmark suite, Collaborative Benchmarking Laboratory, North Carolina State University, http://www.cbl.ncsu.edu, Directory: /pub/Benchmark_dirs/LayoutSynth90/bench/mixed.

```
construct population of 200 genotypes
evaluate population
for g = 1 to 100 do
    for i = 1 to 50 do
        select parents
        create offspring
        with probability P{0.015} do
            mutate offspring
        add offspring to population
    end for
    evaluate offspring
    eliminate the 50 lowest-ranked
        genotypes from population
end for
final solution = top-ranked genotype
```

Figure 1. GAP genetic algorithm.

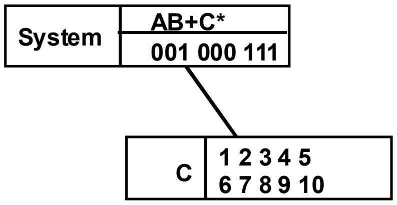

Figure 2. Genotype encoding example.

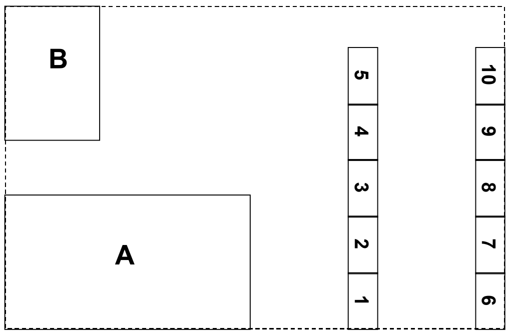

Figure 3. Layout represented by genotype example.

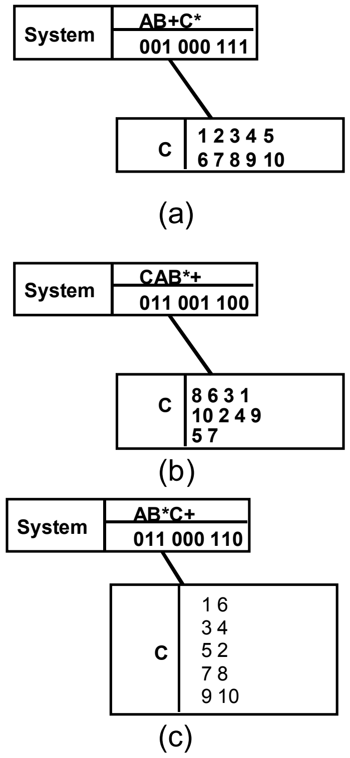

Figure 4. Crossover on (a) and (b) to form (c).

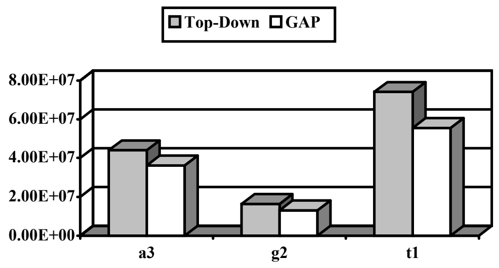

Figure 5. Mean area response.

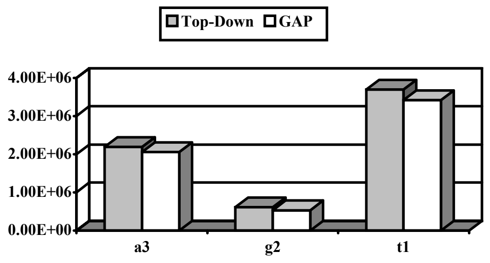

Figure 6. Mean wire length response.