Soft Macro Cell Generation by Two Dimensional Folding

by

Lukas P.P.P. van Ginneken, Jos T.J. van Eijndhoven,
Paul R.M. van Teeffelen, Theo J. Deckers

Automatic System Design Group, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tel: 31-40-473710, fax: 31-40-455925, telex: 51163, eleslukas@heithe5.bitnet

ABSTRACT - **We present a macro cell generator for flexible multilevel nMOS nor arrays. New is a two dimensional column and row folding algorithm that optimizes area usage and cell shape. The accurate control of the shape is important in combination with top down floor planning. Compared to PLA's the new cell generator allows for multilevel logic decomposition, and generates denser and more flexible layouts. The folding algorithm uses an elegant hierarchical divide and conquer technique, and produces results that are 59% smaller than a simulated annealing approach.**

1. INTRODUCTION

The building block placement problem has been shown expensive to solve. The problem of packing many blocks of different shapes into a minimal rectangular area is very difficult for heuristics. To overcome these difficulties top down floorplanning [7] ignores the rigid shape constraints and assumes that any shape can be made. The shapes of the blocks can be optimized after the floor plan topology has been determined [9]. To make floor planning feasable, it is necessary that cell shapes are controlled. So far few cell generators have this capability and they mainly rely on placement and routing.

Our cell generator uses different amounts of folding in both dimensions to to obtain the desired aspect ratio. The two dimensional folding algorithm uses a hierarchical divide and conquer approach to gradually refine the folding. The circuit is composed of horizontal and vertical strips, with a connection pattern that is determined by its function. When multiple strips are assigned to the same column or row, we say that they have been folded. Folding strips into the same column may make other foldings in both dimensions impossible. Therefore the orientation of folding can be used to steer the process towards the desired aspect ratio. For larger arrays two dimensional folding also increases the functional density.

Folding techniques have been developed for PLA's [2] where the objective was to minimize the area of the PLA. These techniques can handle certain constraints for the pins, but the possibilities of adapting the shape and pin positions are limited. Another limitation of PLA folding is that only two strips can be folded into one column. Also channel routing can be formulated as a one dimensional folding problem (track assignment). The unconstrained one dimensional case can easily be solved by the left edge algorithm. For the constrained case heuristics have been given by [8] and others.

Unconstrained folding is used in Weinberger arrays [6] where the nets can be assigned using the left edge algorithm. The Weinberger array can be viewed as a PLA in which the and and or planes have been merged into one plane. This makes multi level logic possible. We use a variation of the Weinberger array that allows for two dimensional folding. Not only can several nets be assigned to the same row, but also multiple gates can be assigned to the same column. Signal terminal locations can be specified at any side of the array.

2. DOUBLE FOLDED LOGIC ARRAYS

The double folded logic array consists of vertical and horizontal strips. All strips, except some strips used for I/O terminals, are connected pairwise forming a cross. Each cross realizes a nor gate. In fig.1 the schematic diagram of such a nor gate is given.

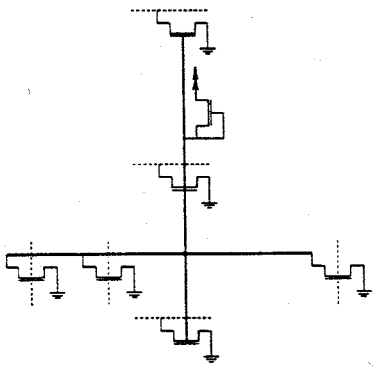

Figure 1. Cross of strips

Since the vertical strips connect the gates of transistors they will be refered to as *gates*, even though no transistor may be present. The horizontal strips will be called *nets*.

The nets are connected to the drains of the transistors; the sources are always connected to ground. This way the transistors function as the pull down devices of the nor gates. The output is formed by a via, which connects the horizontal net to the vertical gate thus forming a conducting cross carrying the output signal of the nor. A depletion transitor is connected to the gate as a pull-up. The pull-ups of several nors are placed in rows to allow them to be connected by a power net.

Several gates may be assigned to one *column*, and several nets may be assigned to one *row*. This multiple assignment is known as *folding*. Obviously, strips in the same column are not allowed to overlap. The amount of folding is therefore limited by the connections that have to be made. To make two dimensional folding possible multiple rows of pull up devices are introduced.

This layout style has a number of features, which make efficient folding possible. The most important one is that the sequence of the transistors and the output is irrelevant. This way the folding algorithm is completely free to determine the most optimal sequence. Secondly, the devices and vias that are located at the intersections of the gates and nets, have sizes that are

This work was supported by the EEC under Esprit project nr. 991, and by the Foundation FOM under project nr. EEL.31.0417

comparable to the width of the wires. As those sizes diverge more, seperating the devices and the interconnections into disjunct areas becomes advantageous. An important property of nors is that all pull down transistors can have the same size, when the pull ups have the same size. The size of the pull downs does not need to depend on the number of inputs.

In fig.2 the schematic of a small example circuit is given. The circuit is a full adder which implements the following boolean equations:

$$D=AB+AC+BC$$
$$E=ABC+\bar{D}(A+B+C)$$

This 5 level implementation is not necessarily the best, but it illustrates the capabilities of the technique. Since a PLA realization needs only 3 levels, this is always a possible solution. The multilevel implementation however needs 7 transistors less then a 3 level implementation.

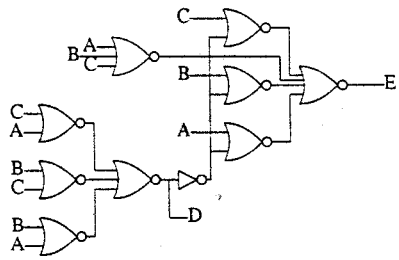

Figure 2. Full adder schematic circuit.

The layout of the above circuit can be seen in fig.3. The implementation presented here is suitable for a simple nMOS process with one layer of metal. The gates are realized in polysilicon and the nets in metal. For the ground extra vertical difusion strips are introduced everywhere. They do not take part in the folding process. Notice that two rows of pull ups have been placed, which allows several gates to share the same column.

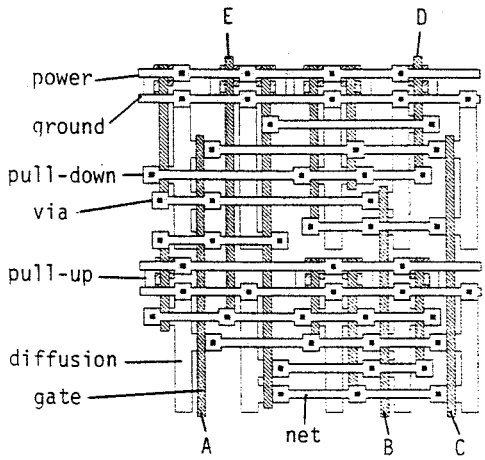

Figure 3. Mask layout of a full adder

3. PROGRAM FLOW

The program begins by reading a set of general boolean expressions, as generated by the logic decomposition algorithms. These boolean expressions are mapped into nors, adding some inverters where necessary. This is done by local transformations. It is guaranteed that no two inverters in series will be present.

After the mapping to nors is done, the net/gate incidence structure is composed. For the input and output terminals four pseudo strips are introduced, one for each side of the array. Each pseudo strip will be assigned to a predetermined side of the array by the folding. This way the terminals are forced to the side that was specified.

In the next stage the folding algorithm assigns nets and gates to rows and columns. In this stage the power and ground connections are ignored. The folding algorithm uses a hierarchical top down approach similar to the approach of [1] to gate array routing. The array is repeatedly subdivided by orthogonal cutting lines. Each time the 2xN folding problem is solved. This way the assignment is gradually refined.

The power and ground nets are inserted in the array after the folding. Subsequently the pull-ups are assigned to valid positions on the supply nets. If a single net were used for the ground and power, folding would become impossible. Therefore we do not consider the power and ground during the folding.

The power and ground line are placed in pairs, and always occupy a full row in the array. The pull up devices are placed on the intersection of its gate and a power line. The number of power and ground lines is therefore determined by the legal positions of the pull ups. Because pull-up are wider than pull downs and vias, two pull-ups are not allowed to be adjacent between two ground diffusion strips. The minimum number of power and ground lines can be determined by a linear scan over the array. A row of pull ups is placed whenever the placement of a pull up of some gate can no longer be delayed. The first row of pull ups must always be to the top of the array to connect the ground diffusions (See fig.3).

A simple grid based compaction algorithm is applied before the masks are generated. The design rules are specified on an element to element basis. Different rules apply in different directions and when elements are connected. Vias and transistors are required to be on the center of the grid lines. When the arrays become larger, however, this compaction becomes less effective. The probability that somewhere on a grid line the maximum design rule must be applied increases with the size of the array.

4. THE FOLDING ALGORITHM

The folding problem can be formally stated as follows: The circuit is specified as a bipartite graph $B(G,N,E)$, with the nodes G representing the gates and N representing the nets. The edges $E \subset G \times N$ represent the gate / net incidences. The circuit is to be realized on a grid of *rows* and *columns*. The set of grid points is represented by $Z \times Z$. The layout of a circuit is determined by a *gate assignment function* $\phi: G \rightarrow Z$ which assigns gates to columns and a *net assignment function* $\psi: N \rightarrow Z$ which assigns nets to rows. Let $\nu(n)$ denote the set of neighbors of n: $\nu(n) = \{g \in G \mid (g,n) \in E\}$. The *span* σ of a net $n \in N$ is an interval of columns defined as $\sigma(n) = [\min_{g \in \nu(n)} \phi(g), \max_{g \in \nu(n)} \phi(g)]$. The spans of gates that are assigned to the same column are not allowed to overlap:

$$\forall_{g_i,g_j \in G} [\phi(g_i)=\phi(g_j) \Rightarrow \sigma(g_i) \cap \sigma(g_j) = \emptyset]$$

Since the problem is symmetric the same goes, mutatis mutandis, for the nets. The objective of the folding algorithm is to find a valid ϕ and ψ subject to some cost function, for instance area.

The heuristic that we propose uses a divide and conquer approach similar to the approach of [1]. The design is repeatedly subdivided by straight orthogonal cutting lines. After each vertical division it is decided which gates are placed to the right of the cutting line and which are placed to the left. After each horizontal division the nets are divided into two groups. After the kth horizontal cut the nets are divided into k+1 sets $N_0..N_k$.

$$N = \bigcup_{i=0}^{k} N_i \qquad \forall_{N_i,N_j} [N_i \cap N_j = \emptyset]$$

Similarly the gates are partitioned into $\ell+1$ sets $G_0,..,G_\ell$ after ℓ vertical cuts. The sets are ordered physically, that is, they imply a constraint on the functions ϕ and ψ.

$$\forall_{g\in G,h\in G}\ [i<j \Rightarrow \phi(g)<\phi(h)]$$

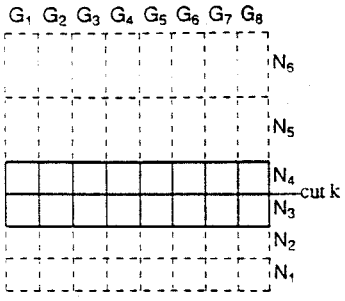

Figure 4. A 2xN subproblem

When a subset G_i is partitioned into two subsets G_i and G_{i+1} this implies a restriction of ϕ (although a solution remains always possible). The index of remaining sets G_j with $j>i$ increases by one. As the exact assignment has not yet been determined the span of a gate will be defined as

$$\bar{\sigma}(g) = [\ \min\{i \mid N_i \cap v(g) <> \emptyset\},\ \max\{i \mid N_i \cap v(g) <> \emptyset\}\ ]$$

To make an estimate of the resulting size of the array, and in order to evaluate the consequences of cutting line decisions, we can use a few numbers that predict the number of rows needed for a set of nets, or the number of columns needed for a set of gates. The maximum number of columns needed for a set of gates is given by

$$\mu(G_i) = \max_j \#\{g\in G_i \mid j\in\bar{\sigma}(g)\}$$

Notice that this is the exact number of columns if $\sigma=\bar{\sigma}$. For the minimum two lower bounds can be given:

$$\delta(G_i) = \max_j \#\{g\in G_i \mid j\in\bar{\sigma}(g)\wedge j+1\in\bar{\sigma}(g)\} \quad \text{and} \quad \gamma(G_i) = \max_{n\in N}\#(v(n)\cap G_i)$$

These upper and lower bounds can be used to estimate the final size of the array.

When a cut is made the nets or gates are divided into two groups. The algorithm chooses an element which results in the largest gain in the object function. A possible objective function, which optimizes area is

$$\left(\sum_{i=0}^{\ell}\frac{\max(\delta(G_i),\gamma(G_i))+\mu(G_i)}{2}\right)\times\left(\sum_{i=0}^{k}\frac{\max(\delta(N_i),\gamma(N_i))+\mu(N_i)}{2}\right)$$

In this objective function the width and height of the array are estimated as the mean of the upper and lower bounds.

To control the shape one could change the object function. However, experiments show that the algorithm is rather insensitive to small changes in the objective function. The shape is therefore controlled by selecting the orientation of the next cutting line. A vertical cutting line tends to make the array wider and lower, a horizontal line makes the array higher and narrower. The orientation is chosen such that the estimated array size is corrected to match the desired aspect ratio. This is a very effective control mechanism, which is also very accurate because the repeated cuts allow for corrections that gradually become smaller.

To partition a set of strips we use a variation on the mincut algorithm [4, 5]. The algorithm starts with an arbitrary partitioning into two sets. It transfers strips one by one to the other set in an attempt to find a better solution. An element which has been transfered becomes *locked* and cannot be moved any more. When all strips have become locked we have completed a *pass*. The algorithm performs several passes until no improvement can be found. Each time the initial state of the next pass is the best state encountered so far. A pass is continued even when the current state becomes worse. This helps to escape from local minima.

To prevent all elements from moving to one set, elements are not allowed to be selected from a set which contains less than ⅓ of the total number of elements. The algorithm continues to perform moves, until no valid move is possible (all elements are locked or all unlocked elements are in a set smaller than ⅓). In practice only a few passes are necessary to arive in a minimum. This is reached when the initial state of the pass is the best state encountered.

PARTITIONING ALGORITHM:

1. Begin with an arbitrary partition. Save this initial state.
2. Begin of a pass: unlock all strips.
3. Is one set smaller than ⅓? If yes then limit the following search to members of the larger set.
4. Transfer all strips back and forth once. Remember which strip has the lowest increase or largest decrease of the object function.
5. Transfer that strip and lock it.
6. Is the present state better than the initial state? If yes then save the present state.
7. Are there still unlocked strips? And are they in a set larger than ⅓? If yes then go to 2.
8. The best state becomes the initial state of the next pass. Did we encounter a state that was better than the initial state? If yes then go to 1.

5. EXPERIMENTAL RESULTS

In [3] the same optimization problem was formulated, and a solution using simulated annealing was proposed. From this paper the following example circuit was used as a bench mark. The circuit contains 36 gates and 28 nets. The terminals were required to be at the same side as in the given example. The result from [3] is given in Fig.5 and uses 21 rows and 21 columns. The result of the new algorithm (Fig.6) uses only 18 rows and only 10 columns, an improvement of 59%.

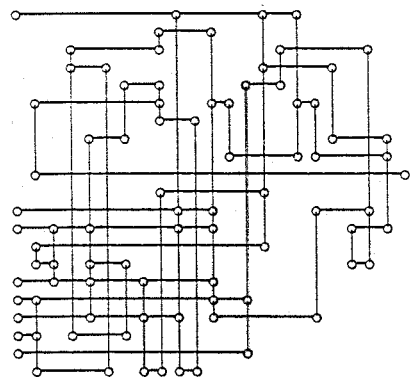

Figure 5. Example circuit from [3]

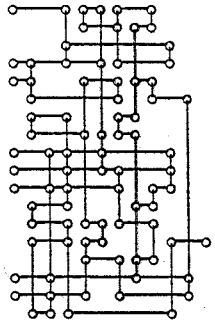

Figure 6. Result of the new algorithm

An example circuit has been layed out with 10 different aspect ratios. In fig.8 five of the results are shown. The numbers indicate the desired aspect ratio. The arrays tend to be approximately 20% too high, which is due to the assignment of power and ground after the folding. Since this factor is quite constant it is easily compensated for.

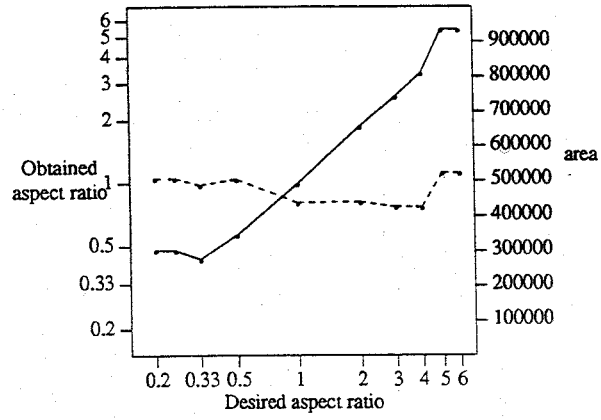

Figure 7. Control over the aspect ratio

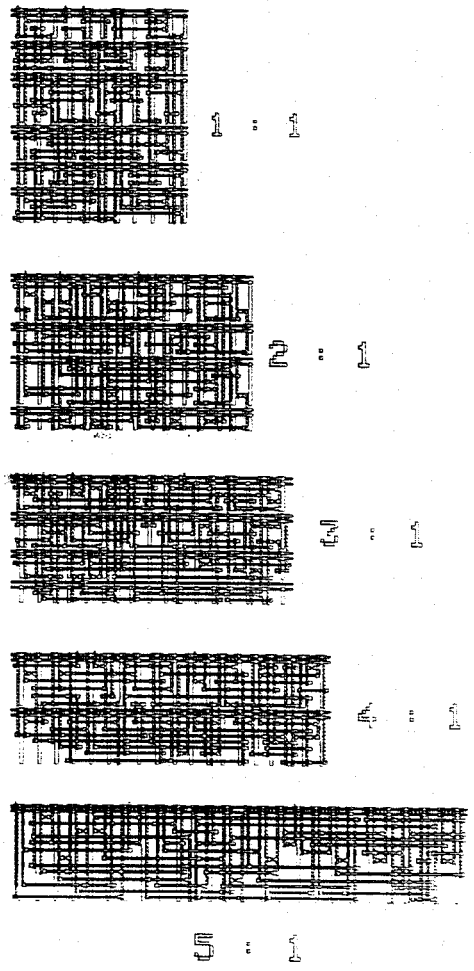

Figure 8. The same circuit with different aspect ratios

In fig.7 the result of this experiment is plotted in a graph. We can see that there is an almost linear releation between the desired aspect ratio and the actually obtained aspect ratio. The aspect ratio can be well controlled upto a maximum or minimum ratio. At the same time the area (dashed) is almost independent of the shape.

6. CONCLUSIONS

A generator for flexibly shaped cells was presented. It uses an elegant hierarchical divide and conquer algorithm. The shape and the pin positions can be controlled accurately, while the area remains constant. This shape optimization is very useful in floorplan optimization in the sense of [9].

The two dimensional folding technique results in very dense layouts. Compared with a simulated annealing algorithm the results were 59% better. A quick comparison with an automatic standard cell program showed a 10% area reduction. It turns out that the object function does not control shape efficiently. Choosing the cut line orientation helps more. Even though all strips are required to be perfectly straight, the amount of freedom in folding is very large. The algorithm has no difficulty to fill any shape of rectangle densely.

This layout style allows for larger regular layout structures than conventional PLA's or Weinberger arrays. An advantage is that also multilevel logic can be implemented. Modern logic optimization tools such as logic decomposition can be applied. By introducing feedback it is also possible to realize sequential circuits such as finite state machines.

The proposed algorithm could also be applied to other logic families, CMOS, and to gate matrix layout. To do this a net with transistors will have to be replaced by a more complex piece of layout.

The two dimensional folding problem has not yet been studied extensively, and other algorithms must be investigated. Folding can be seen as a simultaneous placement and routing technique. Extending this idea it might also be possible to use the algorithm of [1] for the first number of cuts. The wires would not have to be straight, and the mutual influence of the different parts of the layout would be smaller.

REFERENCES

[1] M.Burstein, S.J.Hong, R.Pelavin: "Hierarchical VLSI Layout: Simultaneous Placement and Wiring of Gate Arrays", *Proc.IFIP Int.Conf.on Very Large Scale Integration*, Trondheim, 16-19 Aug.1983, F.Anceau and E.J.Aas (eds.), pp.45-60.

[2] G.De Micheli: "Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications", *IEEE Trans. on Computer Aided Design*, Vol CAD-2, No.3, July 1983, pp.151-167, errata in Vol.CAD-3, No.3, July 1984, pp.256.

[3] S.Devadas and A.R. Newton: "Genie: A Generalized Array Optimizer for VLSI Synthesis", *Proc. 23rd Design Automation Conf.*, Las Vegas, 29 June-2 July 1986, pp.631-637.

[4] C.M. Fiduccia and R.M. Mattheyses: "A Linear-Time Heuristic for Improving Network Partitions", *Proc.19th Design Automation Conf.*, Las Vegas, 1982, pp.175-181.

[5] B.W. Kernighan and S.Lin: "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell System Technical Journal*, Feb. 1970, pp.291-307.

[6] A.Weinberger: "Large Scale Integration of MOS Complex Logic: A Layout Method", *IEEE Journal of Solid State Circuits*, vol.SC 2, No.4, Dec.1967, pp.182-190.

[7] L.P.P.P. van Ginneken and R.H.J.M. Otten: "Stepwise Layout Refinement", *Proc. Int. Conf. on Computer Design*, Port Chester NY, Oct. 8-11 1984, pp30-36.

[8] T.Yoshimura and E.S. Kuh: "Efficient Algorithms for Channel Routing", *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol CAD-1, No.1, Jan 1982, pp.25-35.

[9] R.H.J.M. Otten: "Efficient Floorplan Optimization", *Proc. Int. Conf. Computer Design*, Port Chester NY, Oct. 1983.