AN ENVIRONMENT TO DESIGN DIGITAL CIRCUITS BASED ON THE BRAZILIAN GATE-ARRAY

CARRO, L., MORAES, F., JOHANN, M., KINDEL, M., BENONI, P., MIGLIORIN, G., REIS. R. & SUZIM, A.

Departamento de Engenharia Elétrica
Instituto de Informática
Av. Oswaldo Aranha 103 - Porto Alegre - CEP 90035-190
carro@iee.ufrgs.br

Abstract

This paper describes AGÁ (H), a low cost custom environment to design digital circuits based on the Brazilian Gate Array. The matrix and its metal programming are provided by Centro Tecnológico para a Informática, CTI. All aspects regarding the current status of the CAD system are presented, as well as future developments and distribution policy. Although all the system runs in IBM-PC and Sun workstations, the project is targeted to IBM-PC platforms.

1) Introduction and motivation

The CTI will soon release access to a pre-diffused one metal programmable matrix, in 1.5 technology. This first matrix is aimed at digital circuits development, with an area proportional to 2500 gates (/CHA 94/). To support a design of a circuit, using this matrix, a robust and efficient CAD system is needed, covering design aspects like simulation, placement and routing.

The goal of the AGÁ (H) environment is to provide an easy access to semi-custom design to traditional electronic system designers, in a fast, reliable and low-cost way. Figure 1 shows the design flow of the approach. The designer has access to the system through the digital simulator. After simulating the desired circuit, the user must define the input and output pin position. Placement and routing activities are automatically executed, with no user interference. After this, the system produces a set of masks to program the final layout to be processed by the CTI. Overall design time will be reduced in the future with the use of advanced tools like logic synthesis and more refined routing algorithms.

The paper is organized as follows: section 2 presents the front-end environment, where the simulator and its input structural language are described. Section 3 presents the placement algorithm, while section 4 shows the routing strategy. Finally, section 5 presents our conclusions.

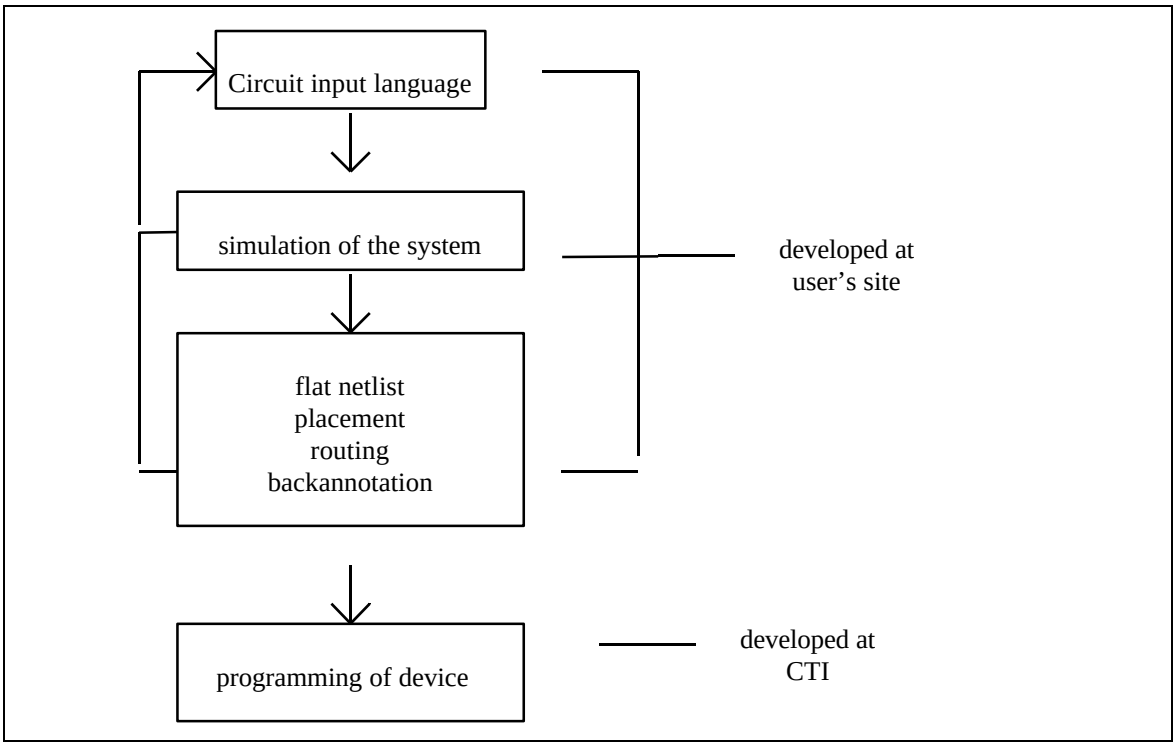

Figure 1 - Design flow

2) The front-end environment

To have access to the system, a designer must have his/her circuit described in the SLX hierarchical logic-level language, as depicted in figure 2. The language has been used in the SLX simulator, developed at UFRGS (/CAR 93/). All primitives described in the library are accepted in the simulator as well.

```
/* Simulation of  one bit counter          */
/*time step and final time definition      */
.tran 1n 200n
/* creates subcircuit  TR with AND and XOR */
.SUBCT TR (s1,e3,e2,e1)
.GATE AND(1n,1n): I1(s1;n1,e3)
.GATE XOR(2n,2n): I2(n1;e2,e1)
.ENDS
/* Instances the block and a D type FF */
.HGATE TR: I1(D,R,T,Q)
.GATE FFD(2n,2n): I2(Q;CK,D)
/* generates some input signals */
.sinal clock1(CK): (10n,10n)
.sinal clock2(R): (20n,200n)
.sinal clock3(T): (40n,100n)
/* print nodes; notice hierarchy           */
.print CK R T Q D I1.n1
```

Figure 2 - Example of input language: one bit counter

The simulator is based on a time wheel, and the inertial delay model is used. This means that if a pulse has a shorter duration than the gate delay, the pulse will not be propagated to the gate output. Each individual cell can have a different delay. This is because the load capacitance in MOS circuits plays a main role in the total gate delay. In this way, fanout connections and routing capacitances influence gate delays in different ways, depending on the position of a particular cell in the circuit.

After the circuit is routed, a backannotation program automatically takes care of load capacitances due to fanout and metal connectionsn, and based on previously studied derating formulas, updates individual cell delays. This gives the designer a fairly good estimate on the total delay of his/her circuit.

After a hierarchical logic description of the circuit is obtained, the designer can simulate the circuit by calling the WSLX interface under Windows 3.1. By opening the design file the simulation is automatically started, and results are exhibited in a graphical screen like the one shown in figure 3.

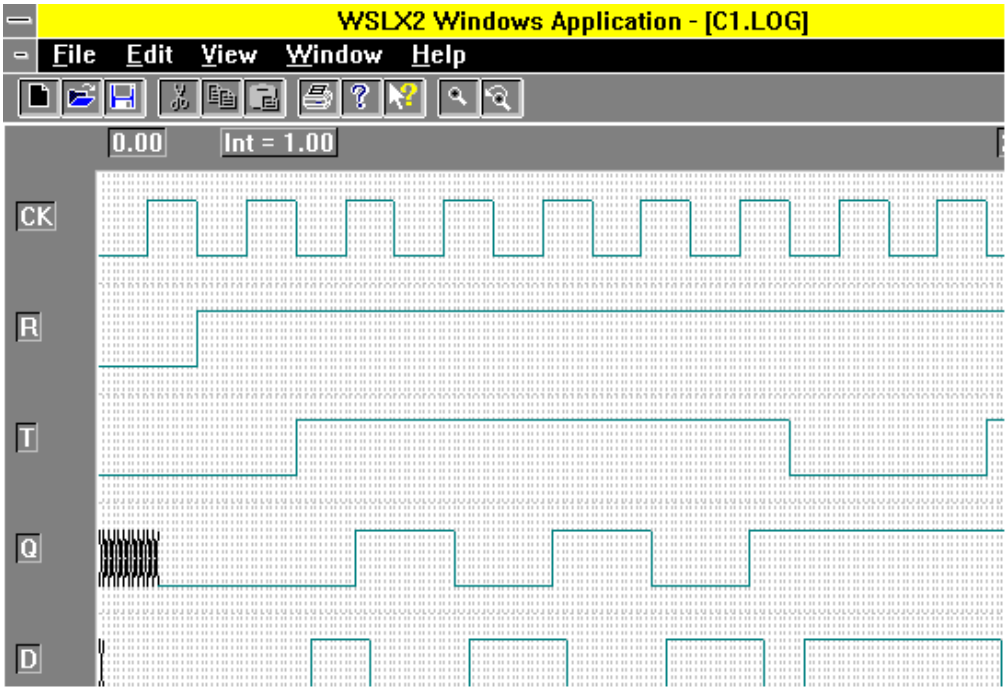

Figure 3 - Output screen of counter simulation

WSLX has 3 logic levels, as described in table 1. A logic signal can assume logical values 0, 1 or X. The later refers to an unknown state of a node, caused by a short between two outputs or a non initialized flip-flop. All three values can have weak strength. This situation arises when a CMOS transmission gate or pass transistor stops driving a node, or no one drives the bus like in 3-state inverters. Since the value is retained in the node capacitance for some time, it can be logically valid, but is overdriven by any other signal that manages to drive the node some time after.

logic value	representation	caused by
0	line	normal gate behavior
1	line	normal gate behavior
X	crossed lines	shorted outputs with different values
0 - weak	dotted line	bus not connected
1 - weak	dotted line	bus not connected
X - weak	dotted line	bus not connected

Table 1- Logic levels

3) Partitioning the design

Once the designer is satisfied with the logical behavior of his/her design, a flat representation of the circuit must be produced by the *planif* program. This description with a file defining the position of input/output pads will be the input to the placement tool, in charge of distributing all cells within the available space in the matrix.

Algorithms for cell placement on pre-characterized circuits must take into account the restricted space to implement connections between cells. In other layout styles, like standard-cells, one can add extra spaces to complete all connections. For pre-characterized approaches, silicon area devoted to routing purposes is fixed. In this way, the placement algorithm must distribute the cells homogeneously in the space, in order to have the best equilibrium between horizontal and vertical routing, avoiding congestion areas in one direction.

Among the known algorithms for cell placement (cluster-grow, partition based, simulated annealing, simulate evolution), the most suited for the present problem, resulting in an equilibrium between horizontal and vertical routing without congestion areas, is a partition based algorithm, called quadrature placement.

When a circuit is divided into 2 blocks, the number of common signals between them is called cutsize. The basic algorithm for the quadrature placement is the min-cut (minimum cutsize), presented in /FID 82/. The min-cut procedure divides a circuit into two blocks, having as cost function the reduction of the cutsize, that is, the number of lines crossing partitions.

The quadrature placement alternatively divides the circuit in horizontal and vertical directions, minimizing the cutsize in each direction. Figure 4 illustrates the method. The circuit is initially partitioned in the vertical direction, using the min-cut algorithm, into two blocks with the same area. This first partition reduces the routing density in the middle of the channels, avoiding congestion in these areas. Next, one has two horizontal partitions, where the area in each one is proportional to the number of rows. For example, for 5 rows there might be an area ratio of 3:2, for 4 rows 2:2 an so on. The partitioning procedure stops when no more horizontal partitions are possible (area ratio1:0).

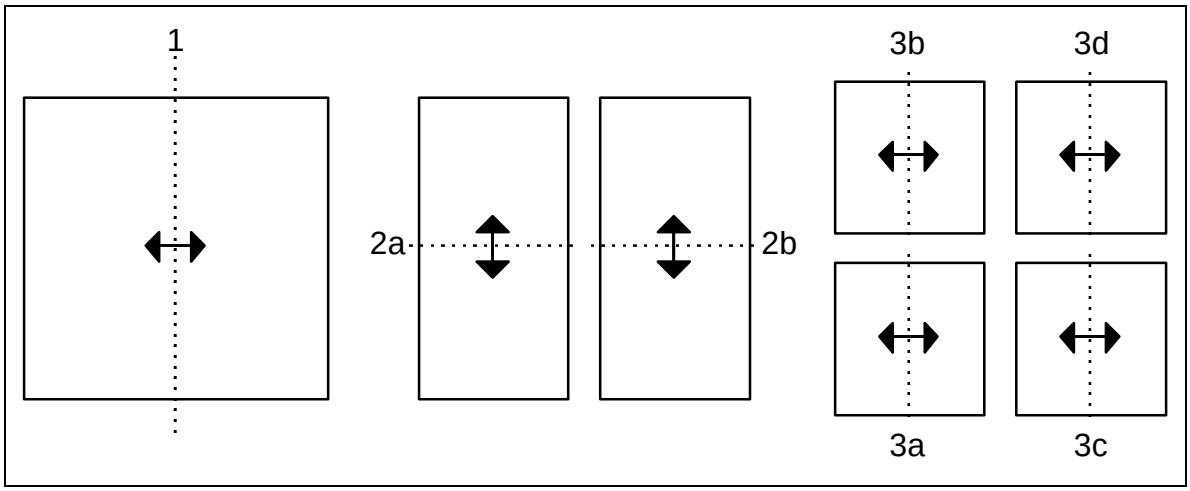

Figure 4 - Quadrature placement

This process results in a set of quadrants, with few cells (typically 2-8) in each one. The cells in each quadrant can be placed using a simple algorithm. As an example, cell order can be obtained directly from the connectivity between cells.

To improve the efficiency of the quadrature placement, the pin propagation algorithm (/DUN 85/) must be implemented. Pin propagation tries to place cells with common signals in adjacent quadrants, reducing routing length, and so improving the electrical performance of the circuit. The pin propagation algorithm is our main contribution, since references give no information on how to solve this problem.

The main idea is the following: to make a partition within a quadrant, each already partitioned quadrant is also taken into account. For the horizontal partition, the quadrants are processed line by line, from left to right. For the vertical partition, the quadrants are processed column by column, from bottom to top.

3.1) Horizontal pin propagation

Supposing the program is doing the horizontal partition of the quadrant number 11 (figure 5-b). The signals considered as interface for this quadrant are:

- South pins :
  - south input/output signals in quadrant 11,
  - common signals between quadrant 11 and quadrants 1 to 8,
  - common signals between the bottom side of quadrants 9 and 10 (already partitioned) and quadrant 11.

- North pins:
  - north input/output signals in quadrant 11,
  - common signals between quadrant 11 and quadrants 17 to 32,
  - common signals between the top side of quadrants 9 and 10 (already partitioned) and quadrant 11.

The vertical pin propagation (figure 5-a) is similar to the horizontal one. Instead of south/north interface pins, the east/west external pins are used.

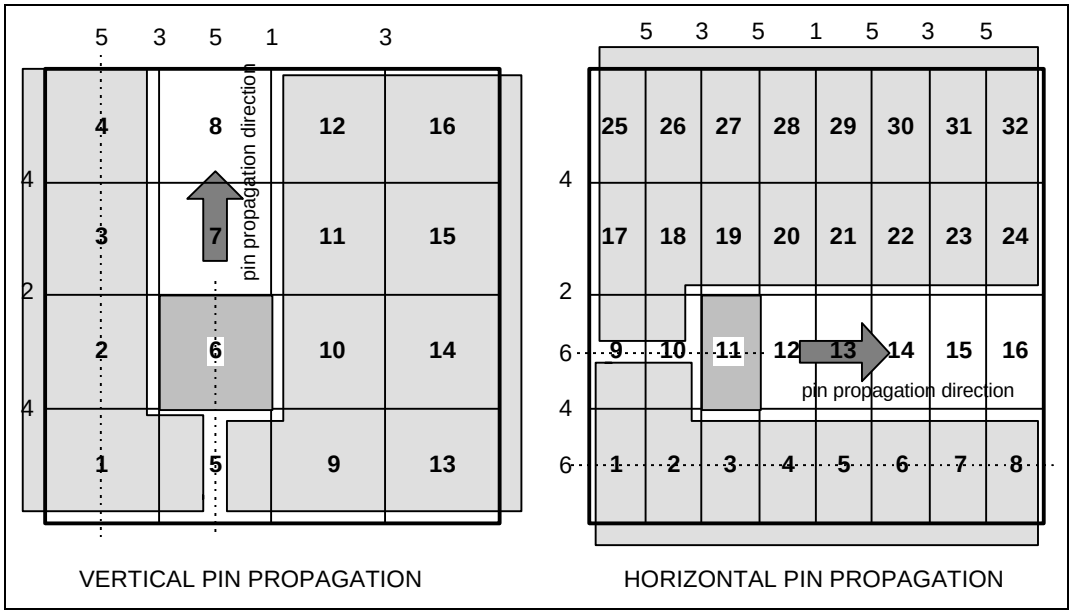

Figure 5 - Pin Propagation

This placement algorithm was compared to another one (cluster-grow with linear placement). The automatic layout synthesis tool TROPIC /MOR 93/ was modified in order to accept the original and the new placement algorithms, generating circuits with the same routing procedure. Preliminary results indicate a great reduction on the number of routing tracks, since the new method distributes homogeneously the connections in both directions, while the old one minimizes the cut between rows, congesting the horizontal routing. Table 2 shows the number of routing tracks required to implement a set of benchmark circuits.

circuit	**number of transistors**	**number of tracks - original placement**	**number of tracks - new placement**
adder	40	6	6
shift8	192	23	22
ls161	222	41	32
cont8	238	42	35
alugate	424	52	39
cla	528	73	57
c880	1570	168	109
c1908	3146	274	151

Table 2 - Routing tracks required using two placement algorithms

Another advantage of this placement method is the reduced CPU time required to place the cells. Table 3 gives the CPU time, in seconds, to place some circuits in a Sun Sparc1 workstation. Table 4 presents the same examples running in the target platform, the IBM -PC family. As one can see, design partitioning is automatically done within a short time even for a PC computer.

circuit	gates	signals	first cutsize	CPU time (sec)
adder	10	15	4	0.1
ls181	63	79	13	1.6
c880	383	445	35	11.6
c1908	880	915	39	32.1
c3540	1669	1721	95	117.9
s5378	2958	2996	113	371.2

Table 3 - CPU time in seconds to realize the cell placement (Sun Sparc1 workstation)

circuit	486DX40	486DX-4/100
adder	0.3	0.2
ls181	2.1	0.9
c880	15.9	6.9
c1908	46.0	20.5
c3540	144.6	68.6
s5378	441.0	224.7

Table 4 - CPU time in seconds to cell placement using the PC family.

4) Routing algorithm

Interconnections of the GA2500 matrix must be implemented using metal 2 layer, connecting to pre-defined metal 1 structures, the underpasses. These are positioned in the routing channels, column-wise, having access points at each two horizontal tracks.

The routing system receives the relative positioning of the circuit after placement is completed. Each cell has already been allocated to a row, and all are ordered. Feedthroughs, whenever needed, are also positioned by the placement program as separated cells. The first routing step generates an absolute placement of cells, consisting of a fixed *X* position for each cell, including feedthroughs.

The second routing step is global routing. A connection between 2 cells of the same row can be implemented in the lower or in the upper channel. For an arbitrary set of connections, cells and rows, some connections must be in a defined channel, others have many degrees of freedom, and these are the most difficult ones to tackle. The global routing step is responsible for deciding which segments of each net will be assigned to which channels. This is done by assigning a status of "*on*" or "*off*" to each cell terminal in the adjacent channels.

To generate the channel routing problem, each "*on*" pin of a cell must receive a *X* coordinate that defines the I/O position. This is needed because many cell terminals have access points in the middle of the layout, and can be accessed in several points of the cell boundary. All possible access points of a terminal, described in the cell library, are evaluated to chose one of them. This task is called Over-the-Cell assignment.

At this moment one has the list of upper and lower nets, characterizing a normal routing channel /SHE 93/. However, there is still a space of two tracks available between the reference boundary of the cells and the channel itself, where the topology allows a river routing. The step of Over-the-Power Assignment is responsible to explore these tracks. Each connected pin is than assigned to one underpass.

If all previous steps were successfully completed, this indicates that the generated absolute placement and global routing guarantees the detailed routing of the entire circuit. Then, the Left-Edge algorithm can be used to generate the optimal channel routing, since there are no vertical constraints /SHE 93/. After the core is completely routed, connections to the circuit PADs are provided.

5) Conclusions

This paper has shown the H (AGÁ) design environment, a CAD system to the development of digital circuits in a Brazilian Gate Array. Although in its initial phase, preliminary results show the feasibility of the approach.

Distribution of this CAD package is in charge of UFRGS and CTI. Future expansions include a high level input language, allowing synthesis and a fast design cycle. Placement and routing programs will also suffer some changing, to increase gate usage within the matrix.

6) References

/CAR 93/ CARRO, L. Técnicas de simulação de sistemas eletrônicos. Exame de profundidade. CPGCC-UFRGS.

/CHA 94/ CHAVEZ, F., BEHRENS, F., RAZERA, L.A., RIBAS, R. & FINCO, S. Biblioteca de Células da Gate Array GA 2500. Documento Interno 010/93. LPCI-IM-CTI, Centro Tecnlógico para Informática, Campinas, 12 Jul. 1994

/DUN 85/ DUNLOP, A. E. & KERNIGHAN, B. A procedure for placement of standard-cell VLSI circuits. IEEE Transactions on CAD, v.CAD-4, n.1, Jan. 1985, pp.92-98.

/FID 82/ FIDUCCIA, C.M. & MATTHEYSES, R.M. A linear time heuristic for improving network partitions. In: ACM/IEEE Design Automation Conference, **proceedings**. Las Vegas, 1982. pp. 175-181.

/MOR 93/ MORAES, F., AZERMARD, N. ROBERT, M. & AUVERGNE, D. Flexible Macrocell Layout Generator. In: 4th ACM/SIGDA Physical Design Workshop, **proceedings**, Los Angeles, 1993. pp. 105-116.

/SHE 93/ SHERWANI, N.. Algorithms for VLSI Physical Design Automation. Massachusetts, Kluwer Academic Publishers, 1993. 488p.