Published in 1st Annual Mentor Graphics Alpine Local Users' Group Conference' (ALPSLUG), 1, 1-4, 1996
which should be used for any reference to this work

A standard-cell library for a Sea-of-Gates: an experience with “ICStation“ and “ICBlocks“

Alexis Boegli*, François Corthay**, Hans Peter Amann*, Fausto Pellandini*

*IMT, Institute of Microtechnology
University of Neuchâtel
Rue A.-L. Breguet 2
CH 2000 Neuchâtel, Switzerland

**EIV, School of Engineering Valais
Route du Rawyl 47
CH 1950 Sion, Switzerland

The aim of the present work was to port a standard-cell library for a Sea-of-Gates (a “SOG” is a pre-diffused circuit where just the “metal“ layers are missing) from “CIF” files existing in a competitors format to Mentor Graphics library format. Obviously, neither “ICStation“ nor “ICBlocks“ have been designed to handle standard-cells on a “SOG“. Therefore, many practical problems arose. In particular, we noted that “ICBlocks“ when routing did not always respect the grid parameters we set, resulting in “DRC” errors of “poly-via” type between elements of the “SOG“ and the routed interconnects.

INTRODUCTION.

The pre-diffused circuits called “Sea-Of-Gate” (SOG) are circuits where the “metal“ layers are missing. The pre-diffused layers of the “SOG” constitute electronic elements (transistors, capacitors, resistors, etc...) that we could connect with the “metal“ layers to form the desired circuits. This kind of circuits is particular designed for little series because of a low manufacturing cost. For example, we could use a “SOG” circuit to test an architecture before a traditional integration.

The aim of the present work was to port a standard-cell library for “SOG” circuits (library “v84xx” from “EM Marin”, technology “cmn20ee”), to test the “Mentor Graphics” capability (in particular “ICStation” and “ICBlocks”) to place and route standard cells on “SOG” circuits with respect the topology of the “SOG”. The work realized at IMT was the following: First, the purely geometrical imported “CIF” cells had to be completed with I/O ports, power supply ports, “vias“, and blockages such that they became usable within “ICStation”. The cells had then to be prepared such that “ICBlocks” fits them correctly to the given “SOG” floorplan (in our case, this is the “v8412”). This operation consist in defining the origin and the boundaries of the cells so that the blockage transistors of the different cells could be automatically superposed with “ICBlocks“. Finally, the placed cells had to be routed taking into account the pre-diffused layers. The goal is to avoid making “poly-vias“ errors between the pre-diffused "poly" and the “vias“ of the “metal“ layers. We note that the operation of placing and routing the cells on a “SOG” is topologically restricting. Obviously, neither “ICStation” nor “ICBlocks” have been designed to handle standard-cells on a “SOG”. Therefore, many practical problems arose. In particular, we noted that “ICBlocks” when routing did not always respect the grid parameters we set, resulting in “DRC” errors of “poly-vias“ type between pre-diffused elements of the “SOG” and the “ICBlocks” set down “vias“ of the “metal“ layers. In order to facilitate the work, “AMPLE" scripts have been written. This scripts are accessible in “ICStation” with a pull down menu we build. The functionality proposed in the menu are: (i) treatment of the imported cells, (ii) floorplaning of a “v8412 SOG”, (iii) place and route the cells on the floorplan, (iv) detection and correction of the “ICBlocks” generated “poly-vias“ errors. The standard-cell library has been completed with libraries developed at “EIV”: (i) a library for logic synthesis with “Autologic II” and (ii) a symbol library for schematic entry with “Design Architect”. The resulting libraries are now ready for test and use in educational projects at “EIV”.

TREATMENT OF THE “V84XX” LIBRARY.

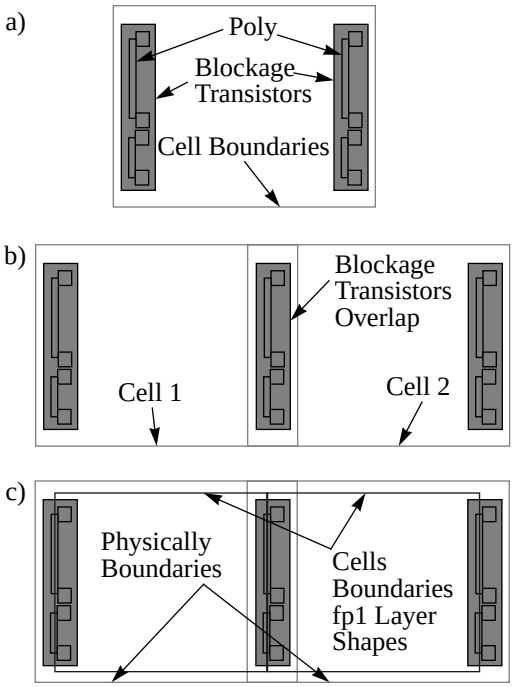
Introduction.

At the beginning, we just have the layout of the cells we imported in “ICStation” with the “CIF” format. In each cell, on the shapes corresponding to the ports, we found the names of the ports. The treatment of the cells consist in defining the port shapes, the blockages, the boundaries and the origin so that they could be insert in a “ICStation” standard-cells library. After treatment, “ICStation” should be able to place and route automatically the cells.

Boundaries and blockage transistors.

On the left and the right of the cells, there are blockage transistors (fig.1a). In order to avoid a wastage of transistors and therefore of place, it is necessary to define the boundaries of the cell (with a fp1 layer shape) so that the blockage transistors can overlap after automatic placing (fig.1b). At the top and the bottom of the cells, we define the boundaries in such a manner that the surface become minimum (fig.1c). In this way, we obtain a largest routing surface outside the cells. On the left and the right, the boundaries of the cell pass in the middle of the blockage transistors "poly" (fig.1c), it is in this way that the blockage transistors could automatically overlap after “ICBlocks“ placing.

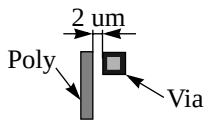
FIGURE 1:



Routing blockages definition.

With the “cmn20ee“ technology and in the case of a “v84xx SOG“, in order to route the cells, we have the use of two “metal“ layers (“metal” and “metal2”). When a metal transition occur (“metal” <=> “metal2”), it is necessary to put “vias“. According to the layout rules, it is necessary that the “poly-vias“ distances are more or equal to 2um (fig.2). Into the cell, if we add blockages around every layers (which means that “metal.block” around “metal”, “metal2.block” around “metal2” and “metal.block”+ “metal2.block” with the property “block_dir= z“ around every “poly”), we note that “ICBlocks“ doesn't succeed in finding a path! There are too much information. In order to obtain a good routing, it is necessary to simplify at the very maximum the blockages. We chose to forbid the “metal” routing into the cells. Therefore, we added “metal.block” type blockages corresponding to the boundaries of the cells. Since we authorize “metal2” routing over the cells, it is necessary to add “metal2.block” type blockages around each “metal2 shapes” of the cells. In order to avoid any problem, we added “poly.block” type blockages corresponding to the limits of the cells.

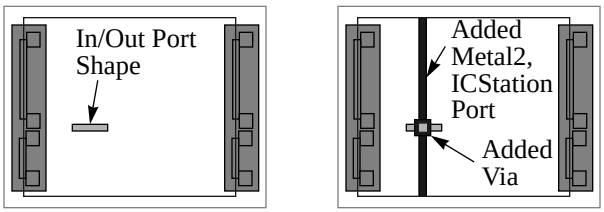
FIGURE 2:



In/out ports definition.

In observing the layout of the cells, we note that the ports are always “metal” made. But we forbade “metal” routing into the cells. For this reason, in order to define the in/out ports, it is inevitable adding “vias“ and “metal2”. To facilitate the “ICBlocks“ routing job, we choose to add “metal2“ port shapes crossing vertically the cells (fig.3.). These Shapes will be then definite as ports.

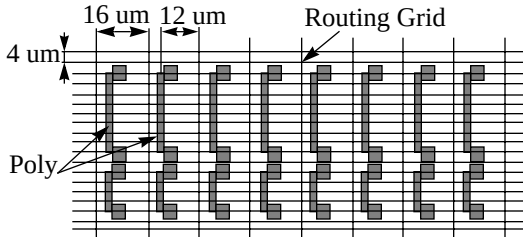
FIGURE 3:



Routing and “poly-vias“ errors.

Prohibiting the “metal“ routing into the cells eliminates the “poly-vias“ type errors into the cells, but not outside the cells. In order to eliminate the “poly-vias“ errors outside the cells, we tested different approaches. The one that consists in adding blockages around the pre-diffused “poly” failed. “ICBlocks“ doesn't tolerate shapes next to cells. We tried also to define some blocks containing the blockages. Then, we placed these blocks between the cells. There also, “ICBlocks“ doesn't support blocks surroundings the standard cells. The solution we adopted is the following. We demand to “ICBlocks“ of respecting a grid. Indeed, for a “v8412 SOG”, if the routing are on a asymmetrical grid of X pitch 16um and Y pitch 4um (fig.4), then one avoids all “poly-vias“ errors. But, in order “ICBlocks“ could respect the grid, it is necessary that ports are on the grid. More precisely, it is necessary that the gravity centers of the port shapes are on the grid.

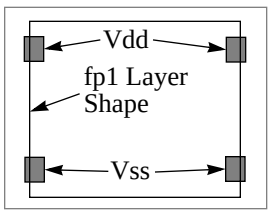
FIGURE 4:



Vdd/Vss power supply ports definitions.

For the power supplies, we add “metal“ shapes that we define like power ports (fig.5).

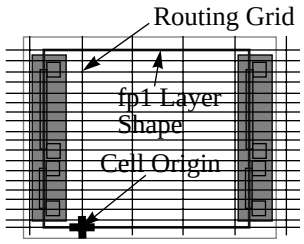
FIGURE 5:



Cells origin.

In order to have the same cells origin, we chose to place it on the routing grid, on the bottom left boundaries of the cells (fig.6).

FIGURE 6:



Automatic treatment of the imported cells

The library we imported contain 68 standard cells. In order to automate the treatment of the cells, we wrote an “AMPLE" script authorizing a semi-automatic treatment of the cells. At first, this program test: (i) if the port names are on the grid. (ii) if there is “metal“ or “metal2“ beneath port names. If these two tests are passed, then the program is in charge of defining the boundaries of the cell, the blockages and the ports. If tests are not satisfied, the program stop and the user must modifying the cell consequently. After cell treatment, the program offers the possibility of deleting the internal aspect of the cell. It allows to create a library of phatom cells, thus keep the confidentiality of the cell layout. When every cells were treated, we could introduce them in a “ICStation“ library of standard cells.

PLACE AND ROUTE OPERATIONS ON A “V8412 SOG”.

Introduction.

Obviously, neither “ICStation“ nor “ICBlocks“ have been designed to handle standard-cells on a “SOG“. The biggest problem that met us, it is the “poly-vias“ errors outside the standard cells. After we tested different methods in order to eliminate the “poly-vias“ errors, we adopt the method generating the less errors. Then, we wrote an “AMPLE" script allowing to detect and mark the errors, and then correcting them.

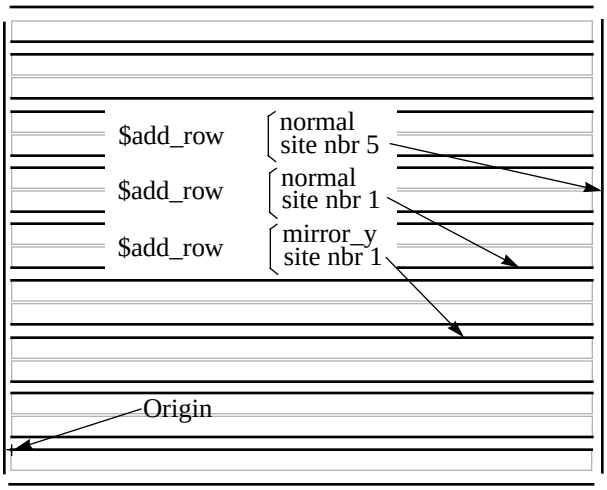
“v8412 SOG”, overview.

The “v8412 SOG“ we used contain: 46 pads, 1 voltage reference, 3 long-channel PMOS transistors, 3 long-channel NMOS transistors, 12 lateral PNP transistors, 6 capacitors (4.6 pF), 72 resistors (20KOhm), 1 crystal oscillator, 16 lines of 135 complementary MOS transistors pairs (Sea-Of-Gate core). The 16 lines of MOS transistors pairs form the core of the Sea-Of-Gate. It is on this 16 lines that we can place then route the “v84xx” standard-cells. We could note that a line on two contains symmetrically placed transistor pairs with regard to the X axis. It means that the standard cells we wants to place on these lines must respect this symmetry. Also, we must work with this structure in order to obtain the desired circuit. This is the principle of the “SOG“. Therefore, the work consists in creating the last 4 masks: “contact“, “via”, “metal”, “metal2”.

Floorplan for “v8412 SOG”.

To be able to place and route the standard-cells on the “SOG“, it is necessary to create a floorplan (fig.7). So that “ICBlocks“ places the cells correctly, we must add two types of rows (Normal & Mirror_Y). And to place the ports we add 4 rows around the core of the “SOG“.

FIGURE 7:



Operation of placing the cells.

When we created the floorplan, we could place the cells with a automatic way or by hand. In the two case it is necessary to take care of placing the cells correctly, with respect for the topology of the “SOG“. In this way, to place correctly the cells, it is necessary that: (i) the origin of the cell is placed on the left of a standard-cells Row (fig.7). (ii) the pitch of the user grid defined in the Process is 16um. If these two points are respected, then we could use the function: `$autoplace_standard_cells` to place the standard-cells. In order to manually place the cells, we could use the hierarchy window of “ICStation“.

Operation of routing the placed cells.

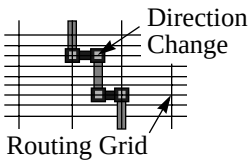
If the library was properly made, after the cells placing operation on the floorplan, all the cell ports must be on the routing grid.In this way, if “ICBlocks“ respects the grid, then we won't obtain “poly-vias“ errors.Since the X and Y pitches of the routing grid are respectively 16um and 4 um, then we must change the user grid pitch to 4um. After that, we could called the `$autoroute_all` function with the following options:

TABLE I:

```
$autoroute_all
(@global_and_detail, @fixed, @all_channels,
 4, @false, 1, 1, @true, 0, @false, @false,
 @no, @no, @no, 0, 0, @no, @horizontal,
 @yes, 1, 100, 0, [3, 2], 1, @gridded, 1,
 4, @edge_mode, 3, 20, 20, [100, 10], [],
 [], @keep, @false, @false, @nocdl, @no);
```

Unfortunately, we note that “ICBlocks“ doesn't always respect the routing grid. Especially during direction changes (fig.8). We were therefore forced to detect, then to correct the “poly-vias“ errors after routing the cells.

FIGURE 8:



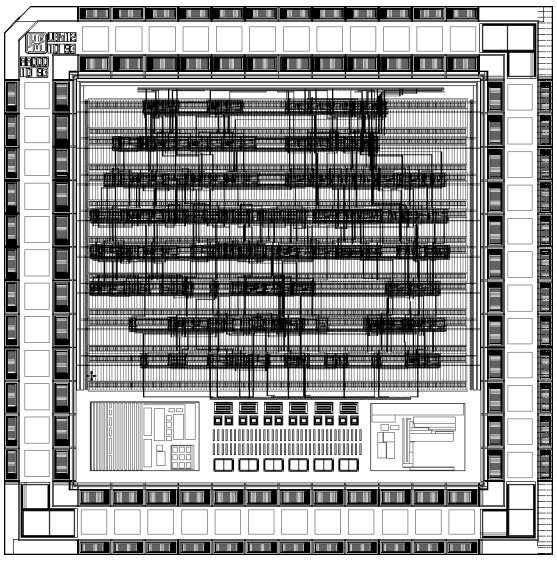
Operations automation.

In order to automate the floorplaning, the place and route operations, the correction of the “poly-vias“ errors, we wrote “AMPLE" scripts. These Scripts could be called with the menu we add to “ICStation“. The automation of the operations of floorplaning, place and route are relatively simple to do, because we use “ICStation“ and “ICBlocks“ functions as they are. On the other hand, for the detection and the correction of the “poly-vias“ errors, we proceed in the following manner: (i) we add unused layer shapes above the pre-diffused "poly" of the “SOG“. These shapes must be bigger from 2um in all directions (fig.2). (ii) Afterwards, we execute a “AND” operation between the shapes corresponding to the pre-diffused "poly" bigger from 2um and the “vias“ add by “ICBlocks“ during the routing. (iii) The result corresponds to every “poly-vias“ errors, that we memorize in a group. (iv) He will be enough of traveling through the found errors and of correcting them in erasing the via, then in using the function `$route_point_to_point`.

APPLICATION EXAMPLE.

The application example we chose, is a 8 bits bidirectional serial parallel interface achieved at the EIV. On one hand the circuit is connected to a I2C bus, and on the other hand to a 8 bits parallel bus. The schematics was achieved with Autologic II and Design Architect, and the logic of system consisting of 12 schematics organized into a hierarchy containing 115 elements of the standard-cells library. The result of placing and routing the logic on a “v8412 SOG“ is shown in figure 9. We note that the integration density is relatively low. In fact, we established that the performances of “ICBlocks“ are greatly degraded when we doesn't enable “ICBlocks“ moving the cells during routing the cells.

FIGURE 9:



CONCLUSION.

The goal of this work was to port a “v84xx” standard-cells library for Sea-of-Gate in “ICStation“ of manner to be able to place and route the standard-cells on a “v8412 SOG“. In this paper, we showed how treat the cells so that they could be used in “ICStation“. We also showed how to proceed with “ICBlocks“ to place and route standard-cells on a “v8412 SOG“. We noted that, for some incomprehensible reasons, “ICBlocks“ when routing did not always respect the grid parameters we set, resulting in DRC errors of “poly-via” type between elements of the “SOG“ and the routed interconnects. To automatically detect and correct these errors, we wrote “AMPLE" scripts. The “AMPLE" scripts we wrote add a menu to “ICStation“. Thanks to this menu, the treatment of the imported cells, the “v8412 SOG“ floorplaning, and the operations of place and route the cells are easier. We did this work for an alone standard-cells library and for an only “SOG“. We are now ready to extend this work to other libraries, other “SOG“ and other technologies. Unfortunately, we established that the performances of “ICBlocks“ are greatly degraded when we doesn't enable “ICBlocks“ moving the cells during routing the cells. As a result a mediocre integration density. The force of “ICStation“ and more generally from the tools of Mentor Graphics lies in the easiness of programming new functions with “AMPLE" scipts.