A Physical Synthesis Design Flow based on Virtual Components

Fernando Gehm Moraes, Michel Robert, Daniel Auvergne, Nadine Azemard

A Physical Synthesis Design Flow Based on Virtual Components

Fernando Moraes[1], Michel Robert[2], Daniel Auvergne[2], Nadine Azemard[2]

[1]FACIN-PUCRS - Av. Ipiranga, 6681 – Prédio 16
90619-900, Porto Alegre, Brazil
moraes@inf.pucrs.br

[2]LIRMM, UMR 5506 CNRS/Uni. Montpellier 2
161 rue ADA
34392 – Montpellier – Cedex 5 - France
{robert,auvergne,azemard}@lirmm.fr

Abstract

This paper presents a virtual library design flow for automatic layout synthesis tools. The motivation to develop such design flow is to enable the use of static CMOS complex gates (*SCCGs*), to optimize area and delay at the logic and physical abstraction levels. The use of *SCCGs* increases the design space (number of different primitive cells) when compared to the state-of-the-art cell based approaches. Therefore, a new design flow is defined, which replaces cell libraries by virtual components, creating in this way a *virtual library*. We present the procedure to create such virtual library in the Synopsys environment, and to integrate logic synthesis tools in automatic layout synthesis.

1 INTRODUCTION

Previous publications ([1], [2], [3]) have presented the advantages/disadvantages of using traditional cell based design or tools for automatic layout synthesis.

Briefly, cell based design has a well established design flow (from behavioral VHDL), cells are pre-characterized and area/delay figures can be computed at the gate level. However, it is not possible to size the transistor widths (the solution in libraries is to provide several templates for each cells, increasing the library complexity/ management) and the library is designed for one given fabrication process. Considering deep-submicron processes, where wire delay must be considered, it should be possible to size each gate according to the load to be driven.

Following [1,2], cell synthesis from transistor netlist level is a flexible way to support open library architecture and satisfy timing closure on cell based design.

In [1] we have: "*Cell synthesis is concerned with creating cell layouts starting with only a transistor level netlist for each cell. It is completely flexible in terms of a target library architecture and does not require any pre-existing cell-specific layout information*." In [2] we have: "*...it clearly appears that on timing-critical cell-based designs it is nowadays a severe penalty not to use a synthesizer*."

Then, one alternative to the cell-based approaches is to automatically generate the layout of each gate, without using cell libraries. Using such approach, each gate can be individually sized, and synthesis will be independent of process fabrication.

Automatic layout synthesis tools to be effectively used by industry or IP providers must solve the following problems: (*i*) how to define a virtual library; (*ii*) how integrate automatic layout synthesis with a logic synthesis tool, using a non limited virtual library; (*iii*) how to characterize the virtual library to allow delay prediction at the gate level.

This paper presents a solution for the first two questions: definition of the virtual library and the integration of layout and logic synthesis tools. Solution for library characterization has been presented in [9] and will be used as necessary.

Section 2 presents the conceptual design flow using a virtual library methodology. Section 3 presents the implementation of such methodology using Synopsy [4] as the logic synthesis environment. Section 4 presents the physical synthesis [5], as well as the optimization step of the design flow.

2 LIBRARY FREE TECHNOLOGY MAPPING

The concept of a virtual library is based on using cells available through a layout generator, instead of using a set of pre-characterized cells.

Due to design and maintenance cost, the number of cells available into a library is quite small when compared to the possibilities given by using complex gates (or *SCCGs* - Static CMOS Complex Gates). The number of different functions that can be implement with SCCGs having 4 or less transistors connected in series is 3503 ([6]) and this number grows to 425803 for arrays of 5 transistors. So, the use of SCCGs available through layout generation increases the design space (number of different primitive cells) when compared to state-of-the-art cell based approaches.

The set of available SCCGs is defined by user constrains, e.g., number of transistors in series, maximum fanout and output isolation. These topological restrictions define the virtual library to be used in library free design approaches.

Therefore, virtual library approach means the possibility to use any kind of *SCCG* for a given circuit, with individual transistor sizing.

2.1 Design Flow

Figure 1 shows the design flow for the virtual library approach. The circuit is specified using a behavioral/structural VHDL description. A logic synthesis tool synthesizes the circuit (Synopsys in the figure), generating as result a structural VHDL. A virtual library containing a rich set of *SCCGs* is used during the technology mapping. User constraints, like critical path delay, output loads, fanout restrictions are specified in the synthesis script.

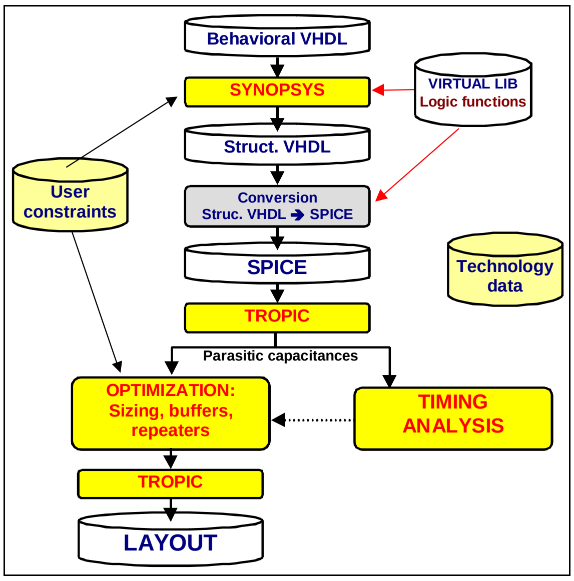

Figure 1 - Design flow in the virtual library approach

This structural description is automatically converted to a SPICE format, since for the automatic layout synthesis a transistor level description is required. The transistors are initially sized with a constant width. The width is a multiple of the minimum transistor width for the employed technology.

A first layout synthesis is performed (Tropic in the figure), aiming parasitic capacitance evaluation. The transistor level description and the parasitic capacitances are used for timing analysis and electrical optimization. If timing constraints are not met, some optimization strategies can be employed: transistor sizing and/or buffer/repeater insertion.

After electrical optimization, if needed, a second layout synthesis is made, resulting in the final layout of the circuit, with each transistors individually sized. It is important to keep the same placement between both layout syntheses, in order to get the same routing and consequently the same parasitic capacitances.

3 VIRTUAL LIBRARY APPROACH IN THE SYNOPSYS ENVIRONMENT

We present in this section a practical implementation of a virtual library approach.

Figure 2 illustrates the first part of the flow, which consists to generate a structural VHDL form a behavioral description, using a virtual library during the technology mapping.

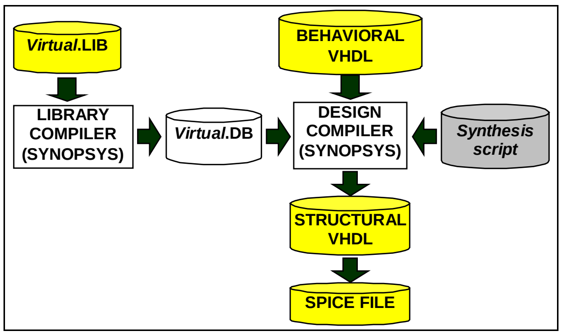

Figure 2 - Virtual library approach in the Synopsys environment

The first action is to describe this virtual library. For example, Figure 3 shows a *SCCG* description in the Synopsys format. Each *SCCG* have: (1) its name, which is used in the final structural VHDL; (2) the area, which is proportional to the number of inputs (this is true, since a *linear matrix* layout style is used by layout synthesis tools [5] [7]); (3) the cell function, defining the *SCCG*, used during technology mapping and spice netlist generation; (4) *SCCG* timing model.

```
cell(AOI1125) {
    area : 18;
    pin(z) {
     direction : output;
     function : "!(A*(B+(C*(D+E*(F+G*(H+I))))))";
     max_capacitance : 0.250000;
     timing() {
        intrinsic_rise : 0.25 ;
        intrinsic_fall : 0.30 ;
        rise_resistance : 0.15 ;
        fall_resistance : 0.10 ;
        related_pin : "A B C D E F G H I" ;
       }
    }
    pin(A B C D E F G H I) {
     direction : input;
     capacitance : 0.250000;
     }
  }
```

Figure 3 - *SCCG* described in the Synopsys format (simplified timing)

In this example a very simple time model, the *generic CMOS* [8], is used. Work under development consists in defining a complete *table lookup model*. Each *SCCG* will be characterized from a set of delay equations for sub-micron technology [9]. In this way, the delay prediction at the logic level will have the same accuracy of the cell-based approaches.

A virtual library is automatically generated from the technology data (electrical model) and topological constraints. No electrical simulations or layout generations are required to perform this task. *There is no restriction on the number of cells into the virtual library, all CMOS static SCCG can be described*.

The sequential elements, flip-flops, latches and tri-states are also inserted in the virtual library file. These sequential elements are not generated automatically, due to its complex description. A template file is modified, according to the technology data. The sequential library has 9 components: 4 FFDs (only D, set, reset, set/rest), 4 LATCHs (only D, set, reset, set/rest) and 1 buffer tri-state.

Once the virtual library file generated, it is compiled using the "*library compiler*" [8] tool (Synopsys), resulting in a library file accepted by the logic synthesis tool, "*design compiler*" (Synopsys).

The input VHDL description can be **behavioral**, **structural** or a **mixed** of them. If a behavioral VHDL (Figure 4-a) is used as input, the designer describes his circuit without references to the cells inside the virtual library.

```
…
input_adder2<=(not busB)  when uins.ula=AsubB else
       (others=>'0') when uins.ula=negA else
       (others=>'1') when uins.ula=decA else
       busB;
…
process(ck,reset,uins)
   begin
    if (reset = '1') then
          n <= '0';   z <= '0';
     elsif ck'event and ck = '0' then
       if uins.nz ='1' then
           n <= out_ula(15);
          if  out_ula="0000000000000000"
             then z <= '1';
             else z<='0';
          end if;
      end if;
    end if;
end process;
```

(a) Partial behavioral VHDL description

```
entity my_circuit is
    port( pg: out reg2; PGij, PGjk, PGkl : in reg2 );
end my_circuit;

architecture A1 of my_circuit is
 signal s1, s2 : std_logic;
  begin
   U0: nor3  port map(PGij(0),PGjk(0),PGkl(0),s1);
   U1: inv   port map( s1, pg(0));
   U3: AOI_BK3 port map (PGij(1), PGij(0),
                PGjk(1), PGjk(0), PGkl(1), s2);
   U4 : inv port map ( s2, pg(1));
end A1;
```

(b) Example of a structural VHDL

Figure 4 - Behavioral and structural VHDL descriptions

If a structural (Figure 4-b) or mixed description is used, the designer must known in advance the cell names, since the logic synthesis will make the mapping using the cell names of the description. In this example, the cells *nor3, inv* and *AOI_BK3* must be in the virtual library, since they are instanced (command *port map*).

A *synthesis script*, containing the area and timing constraints, guides the logic synthesis. Figure 5 shows a synthesis script, where area (*set_max_area*), fanout (*set_max_fanout*) and delay constraints (*set_max_delay*) are imposed. It is important to generate a flat output description (commands *uniquify* and *ungroup*). This is a constraint imposed by the layout synthesis tool.

```
link_library = {sccg33.db libdff.db}
target_library = {sccg33.db libdff.db}

cell_list = { sklansky }

foreach (cell, cell_list) {

    read -f vhdl  cell + ".vhdl"
    current_design cell

    set_input_delay 0.0 all_inputs()
    set_max_delay 1.0 -to all_outputs()
    out_load = 0.25
    set_load out_load all_outputs()
    set_max_area 300
    set_max_fanout 6 cell

    uniquify –force
    ungroup –flatten -all
    compile -map_effort medium

    vhdlout_write_components = TRUE
    vhdlout_single_bit = "TRUE"
    vhdlout_dont_create_dummy_nets  = "TRUE"

    write -format vhdl -hierarchy -output cell +
                                          "_S.vhdl"
   }
quit
```

Figure 5 - Synthesis script, with delay and area constraints

The structural VHDL, synthesized by Synopsys and mapped over our virtual library, is automatically converted to a SPICE format. The final SPICE netlist has three components: sub-circuits, reference to the sub-circuits and output/input pins.

Each cell (*component*, in VHDL) used in the structural VHDL is translated into a spice sub-circuit, using the command *function* (see Figure 3). Figure 6 shows a cell equation, and the respective spice sub-circuit, automatically generated. The translation of an equation to a sub-circuit is performed by a recursive function, which considers the operation '*' (and) as transistors connected in series and the operation '+' as transistors connected in parallel, for the N plan (the P plan is dual to the N plan). Observe that all functions in the virtual library are negative. The tool responsible to translate the netlist spice from structural VHDL reads the same file used by the library compiler tool. The sequential elements are also created from templates (nine previously defined spice subcircuits are used).

All references to cells (*port map*, in VHDL) used in the structural VHDL is translated sub-circuit calls. Finally, the input/output pins are obtained from the *entity* declaration. The resulting SPICE file is used as input description for our layout synthesis tool, TROPIC3.

Example of an initial VHDL and final SPICE files are shown in Figure 7.

!((A*B*C)+(D*E*F)+(G*H*I))

```
SUBCKT AOI1 out  A B C D E F G H I vcc
MN1    gnd   I   6 NMOS w=2.0um l=0.25um
MN2      6   H   3 NMOS w=2.0um l=0.25um
MN3      3   G out NMOS w=2.0um l=0.25um
MN4    gnd   F   5 NMOS w=2.0um l=0.25um
MN5      5   E   2 NMOS w=2.0um l=0.25um
MN6      2   D out NMOS w=2.0um l=0.25um
MN7    gnd   C   4 NMOS w=2.0um l=0.25um
MN8      4   B   1 NMOS w=2.0um l=0.25um
MN9      1   A out NMOS w=2.0um l=0.25um
MP10   vcc   I  16 PMOS w=2.0um l=0.25um
MP11   vcc   H  16 PMOS w=2.0um l=0.25um
MP12   vcc   G  16 PMOS w=2.0um l=0.25um
MP13    16   F  15 PMOS w=2.0um l=0.25um
MP14    16   E  15 PMOS w=2.0um l=0.25um
MP15    16   D  15 PMOS w=2.0um l=0.25um
MP16    15   C out PMOS w=2.0um l=0.25um
MP17    15   B out PMOS w=2.0um l=0.25um
MP18    15   A out PMOS w=2.0um l=0.25um
.ends AOI1
```

Figure 6 – *SCCG* equation and its spice description

```
Entity DIV2 is
      Port(clk,rst: in std_logic; d2_clk: out std_logic );
end DIV2;
Architecture arch of DIV2 is
  signal temp : std_logic;
begin
  d2_clk <= temp;
  process (clk,rst)
   begin
    if rst='0' then   temp<='0';
      elsif clk'event and clk='1' then  temp<=not temp;
    end if;
  end process;
end;

Entity div4 is
   Port(clk,rst:in std_logic; div,div4,div8: out std_logic );
end div4;
Architecture arch of div4 is
   signal carry : std_logic_vector(2 downto 0);
begin
  st0: DIV2 Port Map(clk=>clk,     rst=>rst, d2_clk=>carry(0));
  st1: DIV2 Port Map(clk=>carry(0),rst=>rst, d2_clk=>carry(1));
  st2: DIV2 Port Map(clk=>carry(1),rst=>rst, d2_clk=>carry(2));
  div  <= carry(0);
  div4 <= carry(1);
  div8 <= carry(2);
end;
```

(a) Input description: VHDL

```
**   SUBCIRCUITS  (generated from the lib file)
.SUBCKT DFFR D rst clk QN Q vcc
MN1      H clk gnd gnd NMOS  l=0.25u  w=2u
MP2      H clk vcc vcc PMOS  l=0.25u  w=2u
MN3     NH   H gnd gnd NMOS  l=0.25u  w=2u
MP4     NH   H vcc vcc PMOS  l=0.25u  w=2u
…
.ends DFFR

.SUBCKT GAT1_1   out  A vcc
MN1    gnd   A out gnd NMOS  l=0.25u  w=2u
MP2    vcc   A out vcc PMOS  l=0.25u  w=2u
.ends GAT1_1

**   COMPONENTS  (generated from the VHDL file)
X1  net21 n3 clk net21 div vcc DFFR
X2  net20 n3 n1 net20 div8 vcc DFFR
X3  net19 n3 n2 net19 div4 vcc DFFR
X4  n3 rst vcc GAT1_1
X5  n2 net21 vcc GAT1_1
X6  n1 net19 vcc GAT1_1

**  INPUTS AND OUTPUTS (generated from ENTITY in the VHDL file)
*interface:  div8       orient n  output
*interface:  div4       orient n  output
*interface:  div        orient n  output
*interface:  rst        orient s  input
*interface:  clk        orient s  input
```

(b) Spice file automatically obtained from the input VHDL

Figure 7 - Original VHDL description and resulting SPICE file

4 PHYSICAL SYNTHESIS

This Section presents briefly the layout synthesis features and the work under development in timing analysis/optimization.

The layout synthesis tool, TROPIC [5], is a *macro-cell generator*. The initial SPICE description, obtained from a behavioral VHDL, is decomposed into leaf cells that will be assembled together by dedicated place and route tools, without constitute a separate library. Two instances of the same logic function can have different layouts, according to its environment.

The main features of the layout style:

- *Linear-matrix* layout style, each cell row is composed of two horizontal diffusion strips;
- Routing is implemented with 3 metal layers and stacked contacts, reducing the routing area;
- No layout compaction. This is the main feature of TROPIC, since it allows a very fast layout synthesis. Tools, like LAS [7], create an intermediate symbolic layout description, requiring layout compaction, consuming a lot of CPU time.
- Complete parasitic capacitance/resistance evaluation.
- Simple technology file to describe design rules (28 rules) and parasitic capacitances/resitances (26 rules).

At the cell level, we can enumerate the following features (Figure 8):

- connection between N and P plan directly in metal 1;
- minimum separation between N and P;
- gnd/vcc wires between transistors, in metal2;
- over-the-cell routing (OTC), connecting internal nodes of *SCCGs* and nets belonging to only one channel;
- jogs are automatically inserted in the polysilicon wires, to reduce diffusion area (capacitance reduction);
- polysilicon gate is only aligned to a virtual grid if it is connect to the routing region of the circuit.

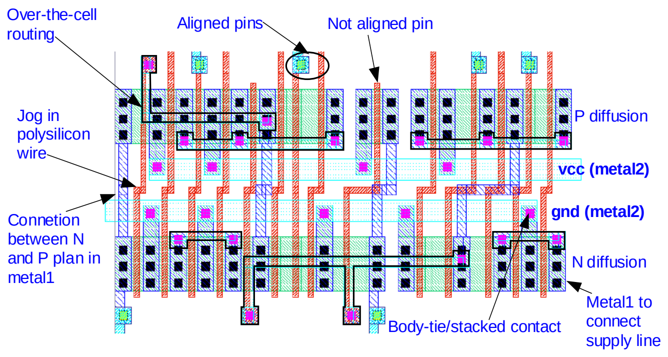

Figure 8 - Layout example (row level)

The layout generator has only 2 input files: the SPICE netlist and the design rules file. Its outputs are the layout (CIF format) and the parasitic capacitance/resistance estimation (flat spice netlist). Figure 9 shows some transistor densities for TROPIC for 2 different processes. For a 0,25 µm process, an average transistor density of 60000 transistors/mm^2 was obtained. Refinements in the routing algorithm can improve this density up to 20% [5]. The CPU time, to generate the layout and estimate parasitic capacitances, for the largest example (15000 transistors), was 227,67 seconds (in an Ultra-10 Sparc).

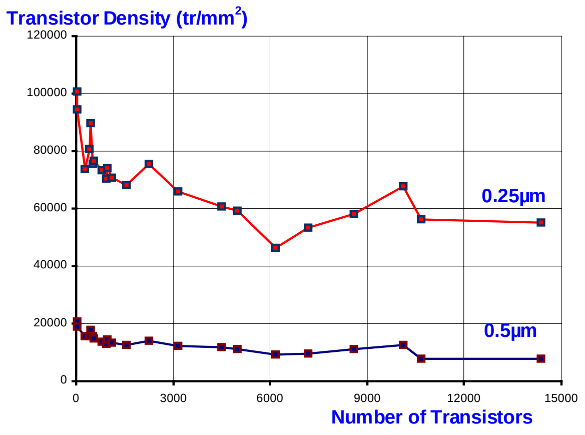

Figure 9 - Transistor densities for TROPIC

As CPU time for layout generation is no more a bottleneck, iterations can be made to obtain an optimized circuit. The logic synthesis tool can execute initial iterations to get accurate information on routing length (parasitic capacitances) and area. This can guide the technology mapping and also indicate where buffers must be inserted, since the real load in each node is calculated during the layout synthesis.

There are three capacitance components: line-to-ground, line-to-line and crossover capacitances. The line-to-ground and crossover capacitances are calculated by the traditional formulation based in the area and perimeter of the interconnections. We used a simple and accurate 2½D methodology described in [11]. For each connection in the layer *i*, we analyze the immediate neighbor in the same layer, all crossunders in the layer i-1 and all crossovers in the layer i+1, treating the layers i±2 like ground planes. However, the line-to-line capacitance is a function of the distance between two connections and the thickness of the layer. We use the empiric formulation described in [10].

The components of the diffusion capacitance, the area and perimeter of drain/source areas also computed by the layout generator.

The circuit delay obtained through electrical simulation using the evaluated capacitances was compared to extracted capacitances. Preliminary results show an average difference of only 5% between estimation and extraction.

The resistances are calculated using the number of squares in each wire, plus the number of contacts and vias. This simple model is been improved. If resistances are also evaluated, the user can choose between 3 representation models (with one state): L, T and Pi.

4.1 Electrical Optimization

After layout generation and capacitance evaluation, the last step in the design flow is the timing analysis and electrical optimization.

At the physical step of the design, the delay and power performances of MOS integrated circuits can be controlled by the appropriate selection of transistor sizes. This sizing problem is generally addressed as a non-linear optimization problem with a number of parameters equal to the number of transistors to be considered. Alternatives can be obtained by regularly varying the transistor drive capabilities until the longest and shortest paths fulfill the delay-power constraint [12]. The consideration on the sensitive paths of the drive capability of gates may reduce the global optimization problem to local optimizations allowing effective management of the circuit delay and power.

The satisfaction of imposed delay constraints implies a full path enumeration with a complete account of the real (post layout) evaluation of the performance parameters of the different switching blocks. For that, it is necessary to get available fast and accurate path identification techniques. This has been obtained using graph exploring techniques such as BFS or DFS algorithms. However in these enumeration techniques all the circuit paths are considered and stored at the expense of prohibitive CPU time and memory allocation when considering large circuits. As a result no complete path enumeration and classification, allowing speed-power trade-off on the different branches can be performed.

The problem of circuit performance optimization must be addressed on a complete path enumeration. We considered an incremental technique, which supplies the enumeration of the longest (shortest) paths in a decreasing (increasing) delay order.

One advantage of this technique is to work on a user specified limited number of paths [13], allowing the easy application of different path optimization criteria. For that we developed a *sizing algorithm* to satisfy delay (power) constraints. This algorithm size the gates belonging to the paths classified by the incremental technique. The selected criterion for gate resizing is defined through the gate load to drive ratio, evaluated on each node and which has been shown to constitute a robust metric for the gate strength and the gate loading evaluation [13]. This parameter associated to the physical performance description of the gate load (considering the estimated parasitic capacitances), gives direct indication of its loading level. Results of such work can be found in [12].

5 DISCUSSION

We have presented in this paper a complete design flow for automatic layout synthesis tools, coupled with logic synthesis and electrical optimization. Our main contributions are (*i*) the definition of the virtual library using SCCGs and (*ii*) the integration of the logic synthesis to the physical synthesis. The characterization of the virtual library is a work under development.

Another alternative to the virtual library was investigated. A simple library (containing basic gates: and, or, inverters) is used during the logic synthesis, resulting in an intermediate description. This description is used by a specific tool for technology mapping, like SIS, generating the SPICE netlist with complex gates. This approach was rejected, since there is no way to control design constraints at the logic level because only simple gates are used.

The advantages of virtual libraries over cell libraries are:

- great flexibility during the technology mapping;
- realistic parasitic evaluation without electrical extraction;
- an incremental path enumeration allowing transistor sizing according to the user constraints;
- easy technology migration.

Design flow based on virtual libraries can replace cell-based approaches in deep sub-micron technologies. This approach allows a fast technology independent IP prototyping for the design of systems on silicon.

References

[1] M.LEFEBVRE, D.MARPLE, C.SECHEN. *The Future of Custom Cell Generation in Physical Synthesis*. DAC'97.

[2] P.IENNE, A.GRIEßING. *Practical Experiences with Standard-Cell Based Datapath Design Tools - Do We Really Need Regular Layouts?* DAC'98.

[3] J.L.BURNS; J.A.FELDMAN. *C5M – A Control-Logic Layout Synthesis System for High-Performance Microprocessors*. IEEE Transactions on CAD, Vol. 17, no. 1, January 1998, pp. 14-23.

[4] http://www.synopsys.com.

[5] F.MORAES, M.ROBERT, D.AUVERGNE. *A Virtual CMOS Library Approach for Fast Layout Synthesis*. VLSI, 1999, pp 415-426. Lisbon - Portugal.

[6] E.DETJENS; G.GANNOT; R.RUDELL; S.VIN-CENTELLI; A.WANG. *Technology mapping in MIS*. ICCAD, Santa Clara, 1987, pp. 116-119.

[7] CADENCE. *Virtuoso layout synthesizer - LAS - user guide*. CADENCE™ Version 4.2, October 1991

[8] SYNOPSYS. *Library Compiler User Guide, Volume 1*. Synopsys Documentation, v1998.08.

[9] J.DAGA, D.AUVERGNE. *A Comprehensive Delay Macro-Model of Submicrometer CMOS Logics.* IEEE Journal of Solid States Circuits, vol 34, n°1, pp.42-55, January 1999.

[10] J.CHERN, J.HUANG, L.ARLEDGE, P.LI, P.YANG. *Multilevel Metal Capacitances Models for CAD Design Synthesis Systems.* IEEE Electron Devices Letters, v.13, n.1, p.32-34, Feb. 1992.

[11] J.CONG, A.KAHNG, D.NOICE; N.SHIRALI, S.YEN. *Analysis and Justification of a Simple, Practical 2 1/2D Capacitance Extraction Methodology*. UCLA Computer Science Technical Report 970013, 1996.

[12] S.CREMOUX; N.AZEMARD; D.AUVERGNE. *Path resizing based on incremental technique*. ISCAS98, Monterey, USA, 1998.

[13] S.YEN, D.DU, S.GHANTA. *Efficient Algorithms for Extracting the k Most Critical Paths in Timing Analysis*. DAC'89, pp. 649-654, June 1989.

Acknowledgements

Fernando Moraes gratefully acknowledges the support of the CNPq through grant no. 522939/96-1.