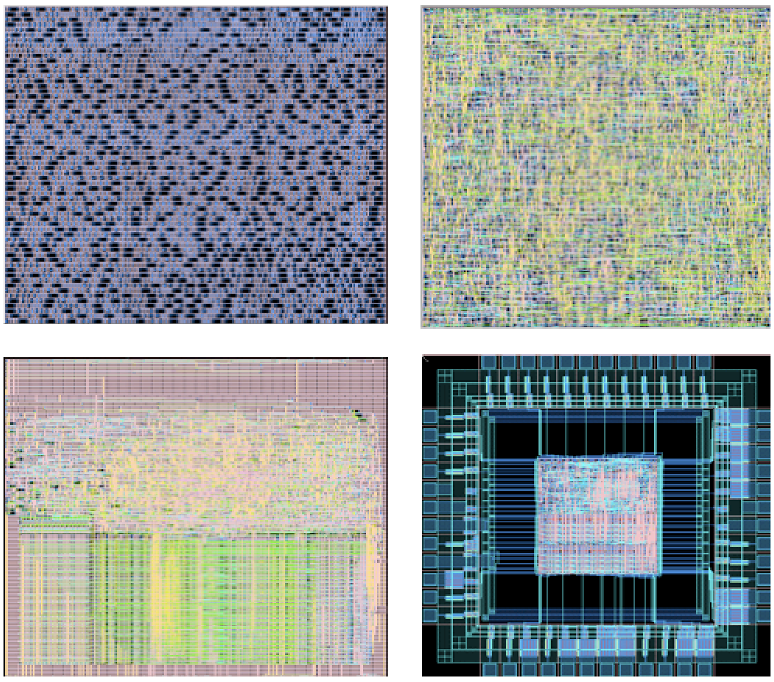
Introduction to VLSI CMOS Circuits Design [1]



Carlos Silva Cardenas
Catholic University of Perú

Takeo Yoshida
University of the Ryukyus

Alberto Palacios Pawlovsky
Toin University of Yokohama

August 18, 2006

[1]Work supported by a grant of the Ministry of Education and Science of Japan and the Toin University of Yokohama.

Contents

List of Figures

List of Tables

Preface

Technology has advanced up to a point where almost anyone with the right tools and knowledge could do what few years ago was the task of a group of specialists armed with powerful workstations and very high expensive software tools. Nowadays we have powerful and inexpensive computers and very elaborate open source software that open us many ways to do different things. This is true not only in the field of web page design, movie production, music composition, and many other fields, but also in digital circuit design.

Today we could do some basic research work or test innovative circuit designs with free design software tools. However since many of the available tools have been developed by volunteers and/or researchers and educators as a mean for teaching and/or research, they almost always lack of adecuate documentation for those beginning to look into the field of digital circuit design and who are in need of detailed information to use those tools in education, basic design and/or test of circuits.

In this book we target the Alliance tools developed at LIP6 of the Pierre and Marie Curie University of Paris since it is a complete set of tools covering many steps of the design process of a VLSI circuit. The authors of this book want to contribute, with its grain of salt, by putting together some of the information that is dispersed in a way that even beginners could benefit from it.

This book is free and is made available to any person that want to use it in a course or for personal education. The print and sold of this book for any lucrative purpose is prohibited. However, you can freely print or reproduce the book or part of it for educative and/or non lucrative purposes. You do not need to previously contact us in this case, provided you include the front page and this preface with the corresponding copies.

The content of the book is very ambitious and we hope others will feel motivated to contribute to its enhancement. You can add, modify and enhance it at your will provided you reach all the modifications to the authors to make them publicly available (In the same spirit of the GPL license for software). We have exhaustively reviewed the content of the book, but we know that there must be some corrections left. If you find any error and/or omission, or have any suggestion or idea, please feel free to contact any of us. We will try to include it (them) in new editions of the book.

Carlos Silva Cardenas (csilva@pucp.edu.pe), Takeo Yoshida (yoshida@ie.u-ryukyu.ac.jp), Alberto Palacios Pawlovsky (pawlovsky@cc.toin.ac.jp).

Chapter 1

Introduction to the Alliance Tools

The set of tools provided by Alliance lets us design and test a circuit from its specification to its layout form and many of its intermediate formats. Alliance provides a symbolic cell library that makes the design of circuits independent of the technology used in their fabrication step. The cells libraries include a standard cell library and several specific purpose cells for memory and data path logic. Many of the tools in Alliance can be used independently as command line tools. Others have a graphic user interface that requires Motif and the X11 library used in many variants of UNIX and Linux. If we provide an adequate technology file the design obtained with Alliance can be converted to CIF or GDSII format for silicon fabrication.

In the following sections we introduce briefly each tool with a very simple example to illustrate their use and options.

1.1 ASIMUT

ASIMUT (A SIMUlation Tool) is a command-line tool with many capabilities. It can be used to check the correctness of the description of a circuit in its VHDL behavioral or structural form. It can also be used for simulating and generating the outputs of the circuit for a given set of inputs. One runs asimut by typing:

```
%  asimut [options] [target_file] [input_pattern_file] [output_pattern_file]
```

Lets try ASIMUT with a small example. For this we will use the behavioral description of the 2-input multiplexer of Figure 1.1.

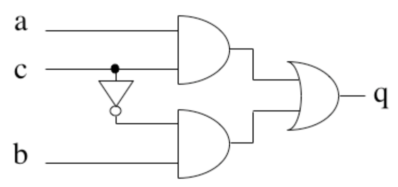


Figure 1.1: A 2-input multiplexer.

In this circuit if the control input **c** is 1 then the output **q** will take the value present in input **a**, but if **c** is 0 then the output of the circuit will take the value of input **b**.

The description of this circuit in its behavioral form is:

```
-- mux.vbe : a discrete 2-input multiplexer circuit
-- VHDL behavioral description
ENTITY mux IS
PORT(
  a   : IN BIT;
  b   : IN BIT;
  c   : IN BIT;
  q   : OUT BIT;
  vdd : IN BIT;
  vss : IN BIT );
END mux;

ARCHITECTURE vbe OF mux IS
 BEGIN
  q <=((a AND c) OR (NOT(c) AND b));
 END vbe;
```

Let's check this circuit description with ASIMUT. But before that, let's check that the environment variables that are referenced by ASIMUT are correctly set. We suppose that you are working with the default setting after installing Alliance. You can check the corresponding environment variables as follows.

```
% env | grep MBK
MBK_IN_LO=vst
MBK_OUT_LO=vst
MBK_IN_PH=ap
MBK_OUT_PH=ap
MBK_WORK_LIB=.
MBK_CATAL_NAME=CATAL
MBK_SCALE_X=100
MBK_CATA_LIB=.:/usr/local/alliance/cells/sxlib:/usr/local/alliance/cells/dp_sxlib:
/usr/local/alliance/cells/rflib:/usr/local/alliance/cells/romlib:
/usr/local/alliance/cells/ramlib:/usr/local/alliance/cells/padlib
MBK_TARGET_LIB=/usr/local/alliance/cells/sxlib
MBK_C4_LIB=./cellsC4
MBK_VDD=vdd
MBK_VSS=vss
```

The most important one for ASIMUT is MBK_IN_LO that tells ASIMUT the extension of the (default) type of the input file. The default type in ASIMUT is structural. A behavioral file must be specified using the -b option. The extension (suffix) of the behavioral file is specified by the VH_BEHSFX environment variable. To just check the syntax correctness of the design file of our multiplexer we use the following command.

```
% asimut -b -c mux
```

Note that we only indicate the target file name since its type (.vbe extension) is explicitly stated by one of the options (**-b**). The output will be the following:

```
% asimut -b -c mux

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @   @@   @
            @@@     @@     @     @                           @    @@    @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@       @@
           @  @@     @@@@     @@@@    @@@ @@  @@   @@    @@       @@
           @  @@       @@@@     @@    @@  @@  @@   @@    @@       @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@       @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@       @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMUlation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching 'mux' ...
BEH : Compiling 'mux.vbe' (Behaviour) ...
making GEX ...
```

Let's modify the above description changing output **q** to **x** in the line describing the behavior of the circuit (**x** is an unknown port name since it is not declared in the entity part of the behavioral description). In this case

ASIMUT will detect the error and will give the following output:

```
% asimut -b -c mux

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @   @@   @
            @@@     @@     @     @                           @    @@    @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@       @@
           @  @@     @@@@     @@@@    @@@ @@  @@   @@    @@       @@
           @  @@       @@@@     @@    @@  @@  @@   @@    @@       @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@       @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@       @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMUlation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching ‘mux‘ ...
BEH : Compiling ‘mux.vbe‘ (Behaviour) ...
‘mux.vbe‘ Error 17 line 17 :‘x‘ unknown port or signal

        cannot continue further more.

                have a nice day...
```

In this case ASIMUT points us to the second line from the bottom, where **x** appears and also displays a message telling us that it is an unknown (undefined) port or signal. The design flow up to this point using ASIMUT is shown in Figure 1.2.

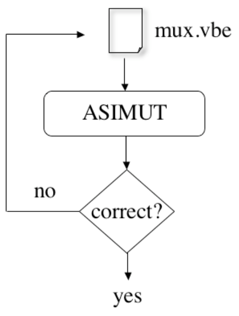


Figure 1.2: Checking the correctness of the design file using ASIMUT.

When running ASIMUT, as shown above, we used two options. The first one **-b** tells ASIMUT that the input file is of behavioral type. The second one **-c** tells ASIMUT to compile the target file. This option makes ASIMUT check the target file for syntax errors before compiling it. After compilation ASIMUT sees if an input file has been given. If the input pattern file has been given it is compiled and linked to the HDL description of the target design. Then, ASIMUT simulates the description and uses the input pattern file to generate an output pattern file for the given circuit. Both the input and output pattern files are written in the **pat** format used in Alliance. Further details about this format will be given in the section covering *genpat*. Here we will explain an abbreviated way to generate the input pattern file.

Every input pattern file has the following general format:

```
-- description of the input and output ports : terminal_type terminal_name logic_type ;
-- use B for binary values, O for octal ones and X for hexadecimal values.
-- source and ground terminals
 in  vdd   B;
 in  vss   B;
-- other input and output ports of the circuit

-- the input values and expected outputs by time interval
-- <time> [description] : input_values_ in_ the_order_given_above expected_value(s) ;
-- output values are preceded by a ?
-- we write the expected values after it ex. ?1 or ?0
-- if one wants the simulator to calculate an output value we indicate it with ?*
 begin
 <0  ns>  initial_value   : 10 00 ?*;
 <+1 ns> next_interval  : 10 00 ?*;
 end;
```

Here lines starting with −− indicate a comment line. Following the above guidelines we can write the input pattern file *mux_in.pat* of the multiplexer circuit as follows:

```
 -- terminals of the mux circuit
 in  vdd   B;
 in  vss   B;
 in  c     B;
 in  a     B;
 in  b     B;
 out q   B;
  -- time interval description of the input values and expected (undetermined) output
 begin
 <0  ns> mux_test           : 10 000 ?*;
 <+1 ns> notc_nota_notb     : 10 000 ?*;
 <+1 ns> notc_nota_b        : 10 001 ?*;
 <+1 ns>                    : 10 001 ?*;
 <+1 ns> notc_a_notb        : 10 010 ?*;
 <+1 ns>                    : 10 010 ?*;
 <+1 ns> notc_a_b           : 10 011 ?*;
 <+1 ns>                    : 10 011 ?*;
 <+1 ns> c_nota_notb        : 10 100 ?*;
 <+1 ns>                    : 10 100 ?*;
 <+1 ns> c_nota_b           : 10 101 ?*;
 <+1 ns>                    : 10 101 ?*;
 <+1 ns> c_a_notb           : 10 110 ?*;
 <+1 ns>                    : 10 110 ?*;
 <+1 ns> c_a_b              : 10 111 ?*;
 <+1 ns>                    : 10 111 ?*;
 end;
```

As shown inn this file, we has written an input pattern file for all the possible combinations of the input values. We start with 00 at time 0 and then at 1ns intervals we modify the input values to test the circuit for all its possible input combinations. We can visualize these patterns using *xpat* (see section 1.30) giving the following command line and opening *mux_in.pat* from the file menu of this graphic tool.

```
% xpat &
```

The opened file will display the input signal and undetermined output signal as shown in Figure 1.3. With this input pattern file we can now run ASIMUT and make it compile the circuit and its input pattern file and

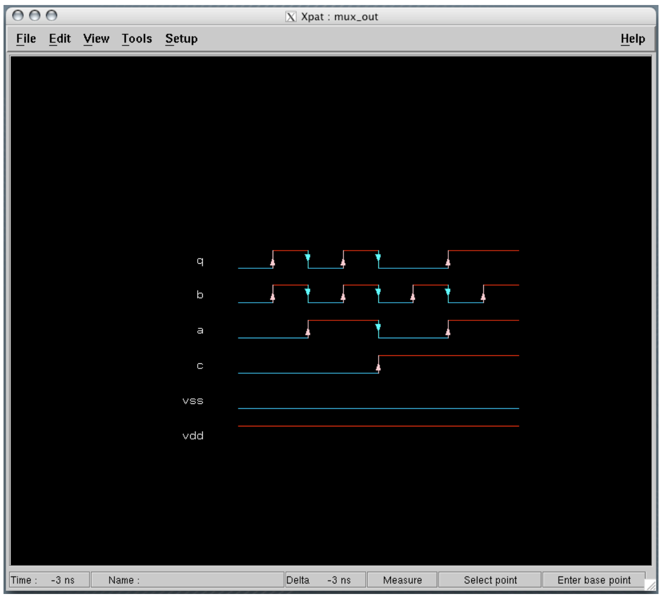


Figure 1.3: The mux_in.pat shown by xpat.

generate the corresponding output pattern file. We run this time ASIMUT as follows.

```
% asimut -b mux mux_in mux_out

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @    @@  @
            @@@     @@     @     @                           @     @@   @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@        @@
           @  @@     @@@@     @@@@    @@@ @@  @@   @@    @@        @@
           @  @@       @@@@     @@    @@  @@  @@   @@    @@        @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@        @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@        @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@        @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@        @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMULation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching ‘mux‘ ...
BEH : Compiling ‘mux.vbe‘ (Behaviour) ...
making GEX ...
searching pattern file : ‘mux_in‘ ...
restoring ...
linking ...
executing ...
###----- processing pattern 0 : 0 ps -----###
###----- processing pattern 1 : 1000 ps -----###
###----- processing pattern 2 : 2000 ps -----###
###----- processing pattern 3 : 3000 ps -----###
###----- processing pattern 4 : 4000 ps -----###
###----- processing pattern 5 : 5000 ps -----###
###----- processing pattern 6 : 6000 ps -----###
###----- processing pattern 7 : 7000 ps -----###
###----- processing pattern 8 : 8000 ps -----###
###----- processing pattern 9 : 9000 ps -----###
###----- processing pattern 10 : 10000 ps -----###
###----- processing pattern 11 : 11000 ps -----###
###----- processing pattern 12 : 12000 ps -----###
###----- processing pattern 13 : 13000 ps -----###
###----- processing pattern 14 : 14000 ps -----###
###----- processing pattern 15 : 15000 ps -----###
```

Again we tell ASIMUT with option **-b** that the input file is of behavioral type, that the input pattern file is named *mux_in* and that we want the output pattern file generated by the simulation to be named *mux_out* (Note that no extension is used with either of the file names used in the command line tools of Alliance). As noted in this output the behavioral file is compiled, ASIMUT searches for the input pattern file, links the files and executes them at the intervals indicated in the input pattern file. The design flow up to this point using ASIMUT is shown in Figure 1.4.

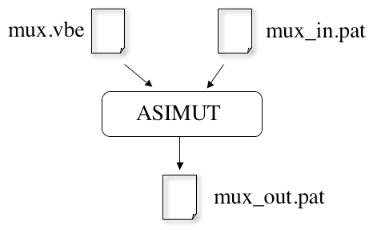


Figure 1.4: Simulating the behavioral design file using ASIMUT.

We can again open and display the output file *mux_out.pat* with *xpat*. This time the output signal **q** also displays as shown in Figure 1.5. This signal has been derived from the behavioral description of the circuit and the values assigned to **c**, **a** and **b**. We can check that the multiplexer circuit behaves as expected passing **b** to output **q** when **c** is 0 and **a** when **c** is 1. It is possible also to indicate in the input pattern file if the last values

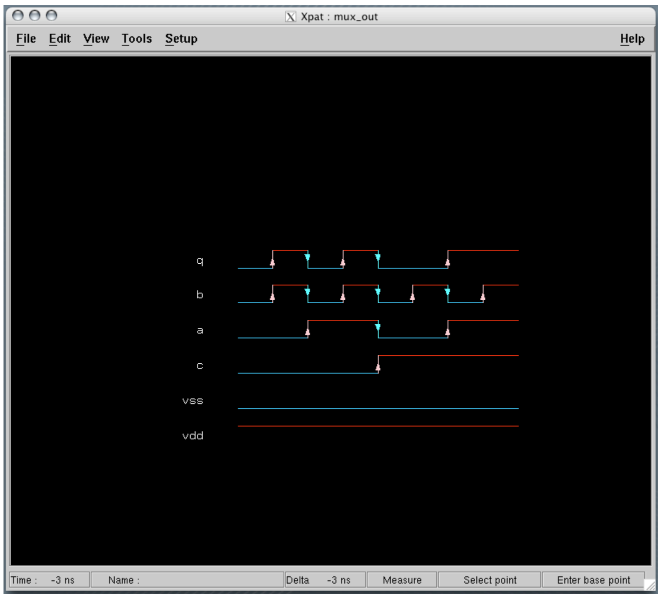


Figure 1.5: The mux_out.pat shown by xpat.

in the simulation must be saved to use them as initial values of the circuit in subsequent simulations. In this case we just add one line with *save*; before *end*; of the input pattern file. Let's try this using a modified copy of our previous input pattern file (only the last three lines of muxs_in.pat are shown).

```
<+1 ns>                    : 10 110 ?*;
<+1 ns> c_a_b              : 10 111 ?*;
<+1 ns>                    : 10 111 ?*;
save;
end;
```

When using this input pattern file the output of the simulation will be the same, but this time an additional file named with the same input pattern file name and extension **.sav** will be generated. Using this **.sav** file we can set the initial conditions of a circuit before running the simulation with any other input pattern file.

The options in ASIMUT are shown in Table 1.1. The **.ext** extension is the extension specified by the environment variable VH_DLYSFX. We can check its value as follows.

```
% env | grep VH
VH_MAXERR=10
VH_BEHSFX=vbe
VH_PATSFX=pat
VH_DLYSFX=dly
```

Table 1.1: Options in ASIMUT.

Option	Description
-b	The RTL circuit description is a behavioral one
-backdelay [min, max, typ] delayfile	Use delayfile (**.ext**) file of backannotated delays.
-bdd	Use BDDs (Binary Decision Diagram) to represent expressions.
-c	Run only the compilation stage.
-core corefile	At error dump the state of the circuit in both an ascii (**.cor**) file and a binary (**.sav**) file.
-dbg[sbpldc]	Call the debugger (developer use only).
-defaultdelay (-dd)	Use only null delays (no after clause in the VHDL file).
-fixeddelay value (-fd value)	Fix all the delays in the description to value.
-h	Display the ASIMUT help file.
-i value	Initialize all signals of the description to value. This can be 0 or 1.
-i savefile	Read a **.sav** savefile and use it to initialize the description.
-inspect instancename	Produce a pattern file for the interface of the instance identified by instancename.
-l n	Print at most n characters for pattern labels. Default n is 15.
-nores	Do not generate the output pattern file
-p n	Load at most n patterns from the input pattern file each time. A value of 0 (default) for n loads the whole input pattern file.
-t	Trace signals when making BDDs (developer use).
-transport	Use transport delay model (default is inertial).
-zerodelay (-zd)	Make all the delays of the VHDL description null delays.

This output tells us that the maximum allowed number of errors is 10. And, that the suffix (extension) of the behavioral, pattern and delay files are **.vbe**, **.pat** and **.dly**, respectively. We must be careful when using any option (check the available ones with asimut -h). For example, the $-bdd$ option makes the simulation faster but would increase memory requirements. We must be careful also when naming files. If we intend to use the $-i$ option, the initialization file must not be named 1 or 0 or it will be considered as the initialization value. On the otherhand, options like $-p$ let us reduce memory allocation when a great number of patterns is simulated (after every n patterns, the simulation result is printed in the output pattern file).

Not all the VHDL syntax is supported in Alliance. We can use only a subset of VHDL to describe our designs. So, descriptions out of the supported subset also cause errors in ASIMUT at compilation time. We review briefly in the following sections the VHDL subset supported by the Alliance tools.

1.1.1 VHDL Subset Supported in Alliance

The Alliance VHDL subset is IEEE Std 1076-1987 compliant. This makes the descriptions written in this subset usable with any commercial/free VHDL tool. The Alliance subset let us describe only digital synchronous circuits. A circuit described with this subset is a design entity that can be a submodule of other entities or the top level module of a design. Each design entity in VHDL has a definition of the ports that let the design communicate with the outside world. The ports could be of type *in*, *out*, and *inout*. The *in* type ports handle input of values to the circuit. The *out* type the output values of the circuit. If both input and output is allowed through a port it must declared of type *inout*.

The predefined set of types handled by the ports is shown in Table 1.2. Remember that user defined types are not supported in the VHDL subset used in Alliance. In VHDL we can specify an entity both structurally and behaviorally in one file. In other words we can mix descriptions in one design file. However in the subset supported by Alliance this is not allowed. We must use one **.vbe** file for the behavioral description of the entity and a **.vst** file for its structural description. In a hierarchical design only the leaf entities are required to have a corresponding behavioral description. In the following subsections we briefly describe the details of the syntax supported in the structural and behavioral subsets used in Alliance.

1.1.1.1 Structural VHDL Subset

In the subset supported in Alliance the structural description of a design can include signal declarations and component declarations. The internal signals can be any of the types of Table 1.2. The structural description itself is a set of component instantations. Component ports must be declared with the same name, type and kind and in the same order in which they appear in the corresponding entity declaration. Component interconnection

Table 1.2: Set of types used in the VHDL subset of Alliance.

Type	Description
bit	The standard bit type: '0' or '1'
bit_vector	Array of bits
mux_bit	Resolved subtype of bit using the mux resolution function. This function checks that only one driver is actually connected to a signal. The effective value of the signal is the value of the active driver. If all drivers are disconnected, the value of the signal is '1' (pullup). A signal of type mux_bit must be declared with the kind bus.
mux_vector	Array of mux_bit
wor_bit	Resolved subtype of bit using the wor resolution function. This function allows a signal be driven by more than one driver. All active drivers have to drive the same value. The effective value of the signal is the value of active drivers. If all drivers are disconnected, the value of the signal is '1' (pull up). A signal of type wor_bit must be declared with the kind bus.
wor_vector	Array of wor_bit
reg_bit	Resolved subtype of bit using the reg resolution function. This function checks that only one driver is actually connected to a signal. The effective value of the signal is the value of the active driver. A signal of type reg_bit must be declared with the kind register (which makes the signal keep its previous value when all drivers are disconnected).
reg_vector	Array of reg_bit

is described using port mapping. We can do explicit and implicit port mapping. However unconnected ports are not allowed. Moreover, only the concatenation operator & can be used in port mapping. Let us see an example using the multiplexer circuit of the previous section. For this we will use the following leaf *AND* gate.

```
-- AND gate entity andg is port(a  : in  bit;
     b  : in  bit;
     q  : out bit;
   vdd  : in  bit;
   vss  : in  bit); end andg;

architecture vbe of andg is begin
  q <= (a AND b); end vbe;
```

And, the following leaf *OR* gate design.

```
-- OR gate entity org is port(a: in  bit;
     b: in  bit;
     q: out bit;
   vdd: in  bit;
   vss: in  bit); end org;

architecture vbe of org is begin
  q <= (a OR b); end vbe;
```

And the following leaf *NOT* gate design.

```
-- inverter (NOT) gate entity invg is port(a : in bit;
     x : out bit;
   vdd : in bit;
   vss : in bit); end invg;

architecture vbe of invg is begin
  x <= NOT(a); end vbe;
```

We will combine these gates as shown in Figure 1.6. The corresponding structural VHDL file is as follows.

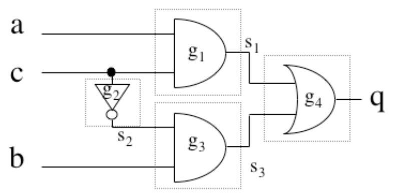


Figure 1.6: One possible structural configuration of our multiplexer circuit.

```
entity mux is
   port (
      a   : in   bit;
      b   : in   bit;
      c   : in   bit;
      q   : out  bit;
      vdd : in   bit;
      vss : in   bit
 ); end mux;

architecture structural of mux is

signal s1 : bit; signal s2 : bit; signal s3 : bit;

Component andg
   port (a   : in   bit;
         b   : in   bit;
         q   : out  bit;
        vdd : in   bit;
        vss : in   bit); end component;

Component org
   port (a   : in   bit;
         b   : in   bit;
         q   : out  bit;
        vdd : in   bit;
        vss : in   bit); end component;

Component invg
   port (a   : in   bit;
         q   : out  bit;
        vdd : in   bit;
        vss : in   bit); end component;

begin

g1 : andg port map (a  => a,
          b  => c,
          q  => s1,
          vdd => vdd,
          vss => vss);

g2 : invg port map (a  => c,
          q  => s2,
          vdd => vdd,
          vss => vss);

g3 : andg port map (a  => s2,
          b  => b,
          q  => s3,
          vdd => vdd,
          vss => vss);

g4 : org port map (a  => s1,
          b  => s3,
          q  => q,
          vdd => vdd,
          vss => vss);

end structural;
```

In this way we can build any hierarchical design. But remember that if the corresponding circuit is at the bottom of the hierarchy (it is a leaf design) we must also provide its behavioral description.

Previously, we were able in Alliance to generate the structural description of a design using a tool called SCMAP that mapped a behavioral description (**.vbe**) into a structural one using the standard cells provided in Alliance. This tool is no longer available, but we can generate these files using two other tools, BOOM and BOOG. We will show one trivial example of their use applying them to generate the structural files of the leaf designs of our multiplexer.The first one is a tool that performs a boolean minimization of the target behavioral description (in our design the leaf designs are basic gates, so nothing worthy will be done). To use it we give the following command.

```
% boom -l 3 -d 50 invg
```

That tells boom to minimize the design described in **invg.vbe**. The option **-l** gives the level of optimization (a value between 0 and 3). The option **-d** indicates what we want to optimize in percentage. A value of 0(%) indicates that we want to optimize delay. A value of 100(%) will optimize area. In our example we indicate that both, delay and area must be equally optimized. The corresponding output will be as follows.

```
% boom -l 3 -d 50 invg

                   @@@@@@@                         @@@     @@@
                     @@   @@                        @@      @@
                     @@    @@                       @@@   @@@
                     @@    @@     @@@       @@@     @@@   @@@
                     @@   @@    @@   @@   @@   @@   @ @@ @ @@
                     @@@@@@    @@     @@ @@     @@  @ @@ @ @@
                     @@    @@  @@     @@ @@     @@  @ @@@  @@
                     @@     @@ @@     @@ @@     @@  @  @@  @@
                     @@     @@ @@     @@ @@     @@  @  @   @@
                     @@    @@   @@   @@   @@   @@   @      @@
                   @@@@@@@@       @@@       @@@    @@@    @@@@

                              BOOlean Minimization

                   Alliance CAD System 5.0 20040928,  boom 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr

        --> Parse BEH file invg.vbe
        --> Drive BEH file invg_o
```

Since we have not indicated an output file the original file name followed by **_o** will be generated (In the output it is the file name drived by BOOM). We give the same command for the other two gates **andg**, and **org**. With the corresponding optimized behavioral designs we then use BOOG to generate the corresponding structural descriptions. BOOG will map the behavioral description and generate and optimized structural description based on the standard cell library (sxlib) provided by Alliance. To use it we give the following command.

```
% boog invg_o invg -x 1 -m 2
```

That tells boog to use **invg_o.vbe**, and generate an optimized **invg.vst** structural file. The option **-x** indicates that we want to generate a **invg.xsc** output file. This file will display the cells used and the delay paths. If we indicates a value of 0 for this option only the critical path will be colored. If we indicate a value of 1 all the paths will be colored. The option **-m** indicates what we want to optimize (we can use values between 0 and 4). A value of 0 indicates that we want to optimize area. A value of 4 will optimize delay. In our example we indicate that both, delay and area must be optimized. The corresponding output will be as follows.

```
% boog invg_o invg -x 1 -m 2

                  @@@@@@@                             @@@@ @
                    @@   @@                         @@    @@
                    @@    @@                       @@      @
                    @@    @@     @@@       @@@    @@
                    @@   @@    @@   @@   @@   @@  @@
                    @@@@@@    @@     @@ @@     @@ @@     @@@@@
                    @@    @@  @@     @@ @@     @@ @@     @ @@
                    @@     @@ @@     @@ @@     @@ @@    @  @@
                    @@     @@ @@     @@ @@     @@  @@      @@
                    @@    @@   @@   @@   @@   @@    @@    @@
                  @@@@@@@@       @@@       @@@        @@@@

                         Binding and Optimizing On Gates

             Alliance CAD System 5.0 20040928, boog 5.0 [2003/01/09]
             Copyright (c) 2000-2005,                 ASIM/LIP6/UPMC
             Author(s):                              Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr

        MBK_VDD        : vdd
        MBK_VSS        : vss
        MBK_IN_LO      : vst
        MBK_OUT_LO     : vst
        MBK_WORK_LIB   : .
        MBK_TARGET_LIB : /usr/local/alliance/cells/sxlib

Reading default parameter...
50% area - 50% delay optimization
Reading file 'invg_o.vbe'...
Controlling file 'invg_o.vbe'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Mapping Warning: Cell 'halfadder_x4' isn't supported
Mapping Warning: Cell 'halfadder_x2' isn't supported
Mapping Warning: Cell 'fulladder_x4' isn't supported
Mapping Warning: Cell 'fulladder_x2' isn't supported
Controlling lib '/usr/local/alliance/cells/sxlib'...
Preparing file 'invg_o.vbe'...
Capacitances on file 'invg_o.vbe'...
Unflattening file 'invg_o.vbe'...
Mapping file 'invg_o.vbe'...
Saving file 'invg.vst'...
Quick estimated critical path (no warranty)...116 ps from 'a' to 'q'
Quick estimated area (with over-cell routing)...750 lambda?
Details...
        inv_x2: 1
        Total: 1
Saving delay gradient in xsch color file 'invg.xsc'...
End of boog...
```

We give the same command for the other two gates **andg_o**, and **org_o**. These will give us with the **.vst** files corresponding to our leaf designs. Check it as follows.

```
% ls *.vst
andg.vst        inv.vst         invg.vst        mux.vst         org.vst
```

Once we have all these files we can now simulate our structural description of the mux using ASIMUT as follows.

```
% asimut mux mux_in muxst_out
```

We are using here the input patterns we used for testing our behavioral description, and we are now saving the generated patterns files in a **muxst_out.pat** file. We can use this file to compare the outputs obtained with those of the behavioral description. This time we get the following output.

```
% asimut mux mux_in muxst_out

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @   @@   @
            @@@     @@     @     @                           @    @@    @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@       @@
           @  @@     @@@@     @@@@    @@@ @@  @@   @@    @@       @@
           @  @@       @@@@     @@    @@  @@  @@   @@    @@       @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@       @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@       @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMUlation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching 'mux' ...
compiling 'mux' (Structural) ...

flattening the root figure ...

searching 'o2_x2' ...
BEH : Compiling 'o2_x2.vbe' (Behaviour) ...
making GEX ...

searching 'inv_x2' ...
BEH : Compiling 'inv_x2.vbe' (Behaviour) ...
making GEX ...

searching 'a2_x2' ...
BEH : Compiling 'a2_x2.vbe' (Behaviour) ...
making GEX ...

searching pattern file : 'mux_in' ...
restoring ...

linking ...
executing ...
###----- processing pattern 0 : 0 ps -----###
###----- processing pattern 1 : 1000 ps -----###
###----- processing pattern 2 : 2000 ps -----###
###----- processing pattern 3 : 3000 ps -----###
###----- processing pattern 4 : 4000 ps -----###
###----- processing pattern 5 : 5000 ps -----###
###----- processing pattern 6 : 6000 ps -----###
###----- processing pattern 7 : 7000 ps -----###
###----- processing pattern 8 : 8000 ps -----###
###----- processing pattern 9 : 9000 ps -----###
###----- processing pattern 10 : 10000 ps -----###
###----- processing pattern 11 : 11000 ps -----###
###----- processing pattern 12 : 12000 ps -----###
###----- processing pattern 13 : 13000 ps -----###
###----- processing pattern 14 : 14000 ps -----###
###----- processing pattern 15 : 15000 ps -----###
```

You would have notice that this time we do not indicate the type of the input file (we did it before using the $-b$ option). This is possible at this point since the corresponding environmental variables are correctly set. When in doubt use the following command.

```
% env | grep MBK
```

This will give the following output (this one corresponds to our environment at to this point in the design flow).

```
% env | grep MBK
MBK_IN_LO=vst
MBK_OUT_LO=vst
MBK_IN_PH=ap
MBK_OUT_PH=ap
MBK_WORK_LIB=.
MBK_CATAL_NAME=CATAL
MBK_SCALE_X=100
MBK_CATA_LIB=.:/usr/local/alliance/cells/sxlib:/usr/local/alliance/cells/dp_sxlib:
/usr/local/alliance/cells/rflib:/usr/local/alliance/cells/romlib:/usr/local/alliance/cells/ramlib:
/usr/local/alliance/cells/padlib
MBK_TARGET_LIB=/usr/local/alliance/cells/sxlib
MBK_C4_LIB=./cellsC4
MBK_VDD=vdd
MBK_VSS=vss
```

We will not detail them here, but notice that MBK_IN_LO is set to **vst**, in this case the default format expected by ASIMUT. The design flow using ASIMUT up to this point is shown in Figure 1.7. Details of the behavioral

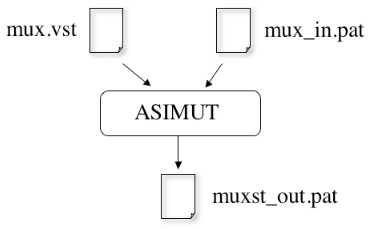


Figure 1.7: Basic structural simulation using ASIMUT.

subset supported by the Alliance tools are given in the following subsection.

1.1.1.2 Behavioral VHDL Subset

In many cases we do not need to specify the structure of the design but its function. In this case we make no reference to the internal structure of the design entity. Such a description of the design is called functional or behavioral. We have already seen the behavioral description of three basic gates (the *AND*, *OR* and *NOT* gates of the previous subsection) and of a multiplexer. In those small designs it is possible to describe the behavior of the circuits as functions of their inputs. However more complex circuits require other types of descriptions. VHDL provide many means for describing complex behaviors, but Alliance supports only part of them. In the subset supported by Alliance only concurrent statements are allowed in the behavioral description. Also, we can not use sequential statements (loops, etc.) or processes. Timing information is allowed in the behavioral description using **after** clauses. However, these delay values are used only in the simulation of the design and not in the synthesis or formal proof stages. The allowed concurrent statements in the subset of Alliance are shown in Table 1.3.

Table 1.3: Concurrent Statements in the VHDL Subset of Alliance.

Type	Example
simple signal assignment	q <= a **and** b;
conditional signal assignment	q <= D **when** clock = '1' **and** E = '0';
selected signal assignment	**with** expression **select** target <= value1 **when** choice1, value2 **when** choice2;
concurrent assert statement	**assert** Reset = '1';
block statement	clump **: block begin** A <= B **or** C; D <= B **and not** C; **end block** clump **;**

When using concurrent statements, we must assure that an ordinary signal is assigned only once. Also, the signal value must be explicitly defined by the signal assignment. When using selected signal assignment we must ensure that the target value is defined for every value that the select expression can take. The restrictions that could arise from this limited set of statements is relaxed through the use of resolved signals (see the VHDL section). We can also use guarded signals. The following example, a bus with two drivers (taken from the man pages), shows the use of resolved and guarded signals.

```
begin first_driver_of_mux : block (Sel1 = '1') begin
 Distributed_Mux <= guarded Data1; end block;

second_driver_of_mux : block (Sel2 = '1') begin
 Distributed_Mux <= guarded Data2; end block;
```

We can use sequential statements but their sequential elements must be explicitly declared using the type reg_bit or reg_vector and be of type register. Also, a sequential element must be assigned inside a block statement by a guarded assignment. An example showing this type of statements is shown in the following edge-triggered flip-flop description.

```
 begin
  Reg <= guarded Din;
 end block;
```

One more example (also taken from the man pages) is the following level sentitive latch description.

```
 begin
  Lat <= guarded Din;
 end block;
```

If we intend to use the behavioral description during the logic synthesis process, we must assure that the guarded signal depends on only one signal (as in the above examples). In the subset supported by Alliance only the logical operators (*not*, *and*, *or*, *xor*, *nor*, and *nand*), concatenation (&), equality (=), and nonequality (/ =) are supported. Arithmetic (+ and −), comparison (<, >) and all other operators (*mod*, <= ∗∗, etc.) are not supported in Alliance. As indicated before, timing information is supported using **after** clauses. But they can not be used in sequential statements (those handling signals of type register). However they can be used with signals of kind **bus**. They can also be used in selected signal assigments provided they have the same **after** clause (same delay). The transport option is not allowed in the subset handled by Alliance. In the assert statement only two levels of severity are supported. One is **warning** that prints a message if the assert condition is not satisfied. The other is **error** that prints an error message an stops the simulation. In Alliance is also possible to use D as don't care value for synthesis purposes but we must be careful since it is not a VHDL standard supported value. During simulation D is replaced by 0 by ASIMUT. The only array types supported in Alliance are those listed in Table 1.2.

1.2 B2F

The B2F (Behavior to Finite state machine format abstractor) tool is a translator from a behavior description to a FSM (Finite State Machine) format. It uses as input a RTL (Register Transfer Level) VHDL description (in the behavioral subset used in Alliance) in **.vbe** format. This tool uses a symbolic simulation algorithm to build a graph equivalent to the target finite state machine that is given as output. We drive the tool giving the following command.

```
% b2f [options] input_file output_flle
```

The options of B2F are shown in Table 1.4. Table 1.4. Let's see one example of use of this tool. For this, we

Table 1.4: Options Available for the B2F Tool.

Option	Description
-V	Sets Verbose mode on
-I bit_string	Initial value of the state register
-O o0,...,on	Initial value (one index list)
-Z z0,...,zn	Initial value (zero index list)
-R reset_cond	Reset condition

will use the following VHDL description of a simple controller.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity RWFSM is
   port(
      CLK : in  std_logic;
      START : in  std_logic;
      RW    : in  std_logic;
      NE    : in  std_logic;
      BUSY  : out std_logic);
end RWFSM;

architecture RTL of RWFSM is
   type STATE_TYPE is (IDLE, READING, WRITING, WAITING);
   signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin
   comb : process(CURRENT_STATE, START, RW, NE)
   begin
      BUSY  <= '0';
      case CURRENT_STATE is
         when IDLE =>
            if START = '0' then
               NEXT_STATE <= IDLE;
            elsif RW = '1' then
               NEXT_STATE <= READING;
            else
               NEXT_STATE <= WRITING;
            end if;
         when READING =>
            BUSY      <= '1';
            if NE = '0' then
               NEXT_STATE <= READING;
            else
               NEXT_STATE <= WAITING;
            end if;
         when WRITING =>
            BUSY      <= '1';
            if NE = '0' then
               NEXT_STATE <= WRITING;
            else
               NEXT_STATE <= WAITING;
            end if;
         when WAITING =>
            BUSY       <= '1';
            NEXT_STATE <= IDLE;
      end case;
   end process;

   seq : process
   begin
      wait until CLK'event and CLK = '1';
      CURRENT_STATE <= NEXT_STATE;
   end process;
end RTL;
```

we will convert this file to an Alliance behavioral description (.vbe file). We do this with the VASY tool of alliance using the following command.

```
% vasy -Vaop -I vhd rwfsm
```

This will produce the rwfsm.vbe file. Having this file we can then use B2F for converting it to a graph format. For this, we use the following command.

```
% b2f -V rwfsm rwgraph
```

This will produce a rwgraph.fsm file that in turn we can be visualize with the XFSM of Alliance. Be careful, we must give the corresponding command in an environment where X11 is available (XFSM is X11-based tool).

```
% xfsm
```

This will start the visualization tool XFSM. From its File menu we can choose the rwgraph.fsm file that will shows as shown in Figure 1.8. As could be noted from this output XFSM is not an elaborated tool. It does not show properly self-loops and does not have editing capabilities either.

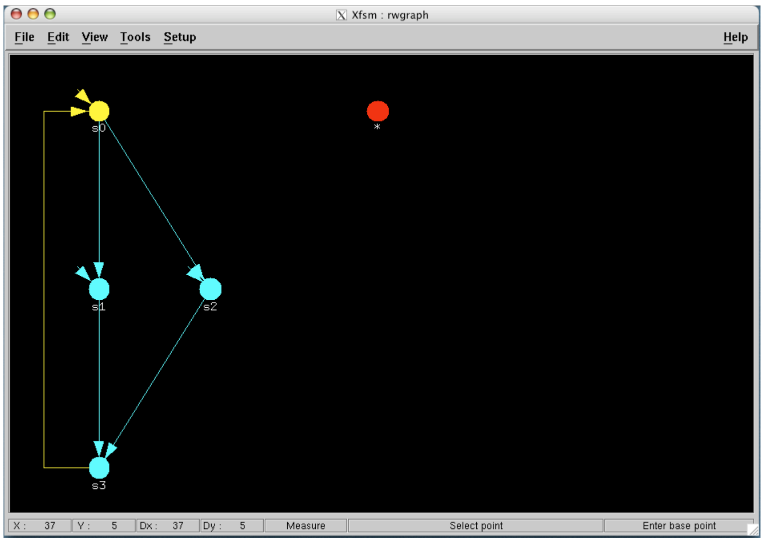


Figure 1.8: The rwgraph.fsm shown by xfsm.

1.3 BOOG

The BOOG (Binding and Optimizing On Gates) tool maps a behavioral description onto a predefined standard cell library. As we have shown in one example of the use of ASIMUT this tool is usually used in a second step of the synthesis process in Alliance. It is used after BOOM. The description optimized by BOOM is the input of this tool. BOOG builds a boolean network equivalent to the description obtained with BOOM. Then for each boolean function of each node of the network it tries to find a cell or a set of cells that implements that function. The result will be a structural description based on the cells of the sxlib library of Alliance. We use this tool given a command of the following form.

```
% boog  [-hmxold] input_file output_file  [lax_file]
```

The options available in BOOG are shown in Table 1.5. When using BOOG it is important to check that

Table 1.5: Options Available for the BOOG Tool.

Option	Description
-h	Help mode. Displays possible uses of boog.
-m optim_mode	Optimization mode. Can be defined in lax file, it's only a shortcut to define it on command line. This mode number has an array defined between 0 and 4. It indicates the way of optimization the user wants. If 0 is chosen, the circuit area will be improved. On the other hand, 4 will improve circuit delays. 2 is a medium value for optimization.
-x xsch_mode	Generate a '.xsc' file. It is a color map for each signals contained in output_file network. This file is used by xsch to view the netlist. By choosing level 0 or 1 for xsch_mode, you can color respectively the critical path or all signals with delay graduation.
-o output_file	Just another way to show explicitely the VST output file name.
-l lax_file	Just another way to show explicitely the LAX parameter file name.
-d debug_file	Generates a VBEdebug file. It comes from internal result algorithm. Users aren't concerned.

the corresponding environment variables are set correctly. The most important ones are **MBK_CATALIB**, **MBK_TARGETLIB**, and **MBK_OUTLO**. **MBK_CATALIB** gives the paths of auxiliary input files (the beahvioral ones). **MBK_TARGETLIB** determines the path of the directory of the selected standard cell library. Finally, **MBK_OUTLO** gives the output format of the structural description.

1.4 BOOM

The BOOM (BOOlean Minimization) tool is used in the first step in the synthesis process of Alliance. It optimizes a behavioral description using an RBDD (Reduced Ordered Binary Decision Diagram) representation of its logic function. This tool gives good results for small random logic but is not usable in datapath optimization. We use this tool given the following command.

```
% boom [-VTOAP] [-l num] [-d num] [-i num] [-a num] [-sjbgpwtmorn] filename [outname]
```

The options available in BOOM are shown in Table 1.6.

Table 1.6: Options Available for the BOOM Tool.

Option	Description
-V	Verbose mode on. Each step of the optimization is displayed on the standard output.
-T	Trace mode on. Some debug informations are displayed on the standard output.
-O	Reverses initial Bdd variables order.
-A	BOOM performs a local optimization and keeps the architecture of the initial description by saving most of the intermediate signals. This mode is well-suited for big or regular circuits such as multipliers, adders. By default BOOM performs a global optimization and removes most of the intermediate signals so that the outputs are expressed in terms of the inputs or the internal registers. This mode is well-suited for random circuits such as FSMs.
-P	Uses a parameter file input_name.boom describing optimization directives and constraints. (see below for the exact syntax)
-l num	Specifies the optimization level [0-3] (default is 0, low level).
-d num	Specifies the delay optimization percent (default is 0%delay, 100% surface).
-i num	Specifies the number of iterations for the choosen optimization algorithm (for experts only).
-a num	Specifies the amplitude during bdd reordering (for experts only).
-sjbgpwtmorn	Specifies which algorithm has to be used for the boolean optimization.

We have already seen an example of the application of BOOM. Let's now use it and BOOG to optimize our mux circuit. First we optimize its behavioral description with BOOM. As we do previously we will use the maximum level of optimization and optimize delay and area. In this case BOOM will give the following output.

```
% boom -l 3 -d 50 mux

                   @@@@@@@                         @@@     @@@
                     @@   @@                        @@     @@
                     @@    @@                       @@@   @@@
                     @@    @@     @@@       @@@     @@@   @@@
                     @@   @@    @@   @@   @@   @@   @ @@ @ @@
                     @@@@@@    @@     @@ @@     @@  @ @@ @ @@
                     @@    @@  @@     @@ @@     @@  @ @@@  @@
                     @@     @@ @@     @@ @@     @@  @  @@  @@
                     @@     @@ @@     @@ @@     @@  @  @   @@
                     @@    @@   @@   @@   @@   @@   @      @@
                   @@@@@@@@       @@@       @@@    @@@    @@@@

                              BOOlean Minimization

                   Alliance CAD System 5.0 20040928,  boom 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s):                 Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr

        --> Parse BEH file mux.vbe
        --> Drive BEH file mux_o
```

BOOM generated a minimized version of mux called mux_o, that we will use as input to BOOG. As n the ASIMUT example we run BOOG indicating that we want to optimize delay and area and that we want an output file to visualize all the delay paths of it. Running BOOG will give the following output.

```
% boog mux_o mux_oo -x 1 -m 2

                  @@@@@@@                             @@@@ @
                    @@   @@                         @@    @@
                    @@    @@                       @@      @
                    @@    @@     @@@       @@@    @@
                    @@   @@    @@   @@   @@   @@  @@
                    @@@@@@    @@     @@ @@     @@ @@     @@@@@
                    @@    @@  @@     @@ @@     @@ @@     @ @@
                    @@     @@ @@     @@ @@     @@ @@    @  @@
                    @@     @@ @@     @@ @@     @@  @@      @@
                    @@    @@   @@   @@   @@   @@    @@    @@
                  @@@@@@@@       @@@       @@@        @@@@

                         Binding and Optimizing On Gates

             Alliance CAD System 5.0 20040928, boog 5.0 [2003/01/09]
             Copyright (c) 2000-2005,                 ASIM/LIP6/UPMC
             Author(s):                              Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr

        MBK_VDD        : vdd
        MBK_VSS        : vss
        MBK_IN_LO      : vst
        MBK_OUT_LO     : vst
        MBK_WORK_LIB   : .
        MBK_TARGET_LIB : /usr/local/alliance/cells/sxlib

Reading default parameter...
50% area - 50% delay optimization
Reading file 'mux_o.vbe'...
Controlling file 'mux_o.vbe'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Mapping Warning: Cell 'halfadder_x4' isn't supported
Mapping Warning: Cell 'halfadder_x2' isn't supported
Mapping Warning: Cell 'fulladder_x4' isn't supported
Mapping Warning: Cell 'fulladder_x2' isn't supported
Controlling lib '/usr/local/alliance/cells/sxlib'...
Preparing file 'mux_o.vbe'...
Capacitances on file 'mux_o.vbe'...
Unflattening file 'mux_o.vbe'...
Mapping file 'mux_o.vbe'...
Saving file 'mux_oo.vst'...
Quick estimated critical path (no warranty)...657 ps from 'c' to 'q'
Quick estimated area (with over-cell routing)...3000 lambda?
Details...
        ao2o22_x2: 1
        inv_x2: 1
        Total: 2
Saving delay gradient in xsch color file 'mux_oo.xsc'...
End of boog...
```

The synthesis process with BOOM and BOOG is shown in Figure 1.9. To see the schematics of the mux design

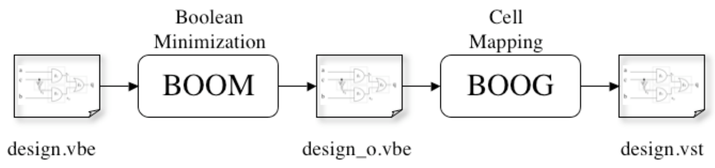


Figure 1.9: The synthesis process using BOOM and BOOG

we use the graphical tool XSCH. We call it with the following command.

```
% xsch
```

Then from its File menu we can choose any .vst file in our working directory. Let's see first our original mux.vst structural design. This is shown in Figure 1.10. Let's see now the optimized and synthesized mux circuit (mux_oo.vst). This is shown in Figure 1.11. As could be seen from these figures the original mux circuit is formed by discrete components while the optimized and mapped design is made of only two cells, one inverter

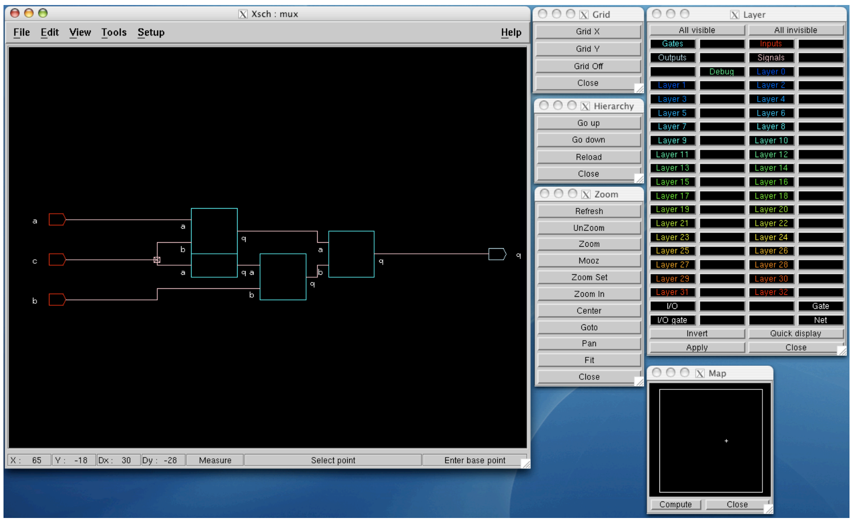


Figure 1.10: The mux.vst shown by xsch.

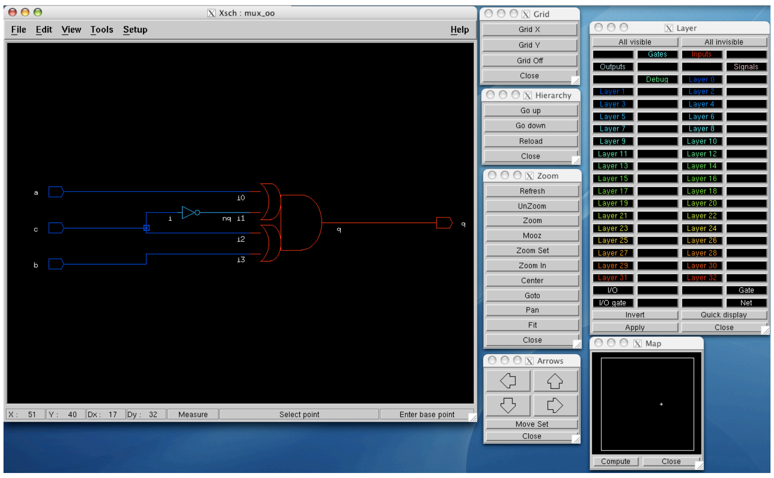


Figure 1.11: The mux_oo.vst shown by xsch.

and a complex and-or gate (both have been chosen by BOOG from the sxlib cell library of Alliance). When using XSCH you have to play with the layers and elements to display to obtain before obtaining the best schema that fits your needs. Use these schematics to see what effect different settings have in the display of your designs. After you find the best window positioning and setting save it using the **Save Config** command from the **Setup** menu of XCSH. The only environment variable we must be aware when running BOOM is MBK_WORL_LIB. This must be set to our working directory.

1.5 COUGAR

This tool is the netlist extractor used in Alliance and formerly was known as Lynx (It changed its name to avoid problems in systems that also use the popular text web browser Lynx). COUGAR is a hierarchical extractor that is applied to a symbolic layout, but it can be applied also to a real one if the corresponding technological file (RDS file) is provided. COUGAR can only extracts CMOS transistors (no other devices are supported). It also extracts parasitic capacitance and resistance but with very low precision. COUGAR is usually run with the following command.

```
COUGAR  [ -v ] [-c ] [ -f ] [ -t ] [-ar ] [ -ac ] input_name [output_name ]
```

The options available in COUGAR are shown in Table 1.7. When no options are specified, COUGAR will extract the current hierarchical level. The resulting net list will be the list of interconnections at the current layout hierarchy level. COUGAR is used to verify designs. We will see an small example of its use applying it to our mux circuit. To do that we need to place and route it first. For this, we will use other tools provided by Alliance. First we place the circuit (the optimized by BOOG and BOOM) using OCP. The corresponding command follows (take into account that the MBK_IN_LO MBK_OUT_PH variables determine the type of input

Table 1.7: Options Available for the COUGAR Tool.

Option	Description
-t	Notifies a transistor level extraction, the symbolic layout cell is flattened to transistor layout before extraction.
-f	The symbolic layout cell is flattened to the catalog level before extraction. Use ”man catal” for detail on the catalog file. If the catalog is empty, or doesn't exist, the netlist is an interconection of transistors, if it isn't, the netlist is an interconection of gates or blocks whose names are defined in the catalog.
-v	Verbose mode on. Each step of the extraction is displayed on the standard output, along with some statistics.
-c	Generates a core file representing the conflictuel net, when COUGAR detects two external connectors with different names on the same signal, or when it finds two external connectors having the same name but not internally connected to the same net, or when it cannot correctly extract an L shaped transistor.
-ac	Extract capacitance to ground on losig.
-ar	Extract interconnect resistance and capacitance to ground. Value of resistance foreach layer can be changed in the RDS file.

file and output files to and from OCP).

```
% ocp mux_oo muxoop

                            @@@         @@@@ @ @@@@@@@
                          @@   @@     @@    @@   @@   @@
                         @@      @@   @@      @   @@    @@
                        @@       @@ @@       @   @@    @@
                        @@       @@ @@           @@   @@
                        @@       @@ @@           @@@@@
                        @@       @@ @@           @@
                        @@       @@ @@           @@
                         @@      @@   @@      @   @@
                          @@   @@     @@    @@   @@
                            @@@         @@@@   @@@@@@

                           Placer for Standards Cells

                   Alliance CAD System 5.0 20040928,   ocp 5.0
                   Copyright (c) 2001-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr
 o Special Net detected : vss
 o Special Net detected : vdd
 o No More Mouvement Possible .....

Ocp : placement finished
```

This gives us a muxoop.ap file. We can visualize this file using the Alliance GRAAL tool. We call it from the command line as **graal** and from its File menu we choose the muxoop.ap file. This is shown in Figure 1.12. We

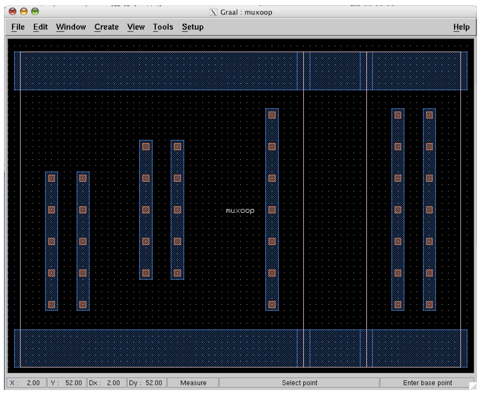


Figure 1.12: The muxoop.ap (the mux design placed by OCP) shown by graal.

can now route the placed mux design with the Alliance NERO tool as follows.

```
% nero -V -p muxoop mux_oo muxoor

                   @@@    @@@              @@@@@@@
                    @@     @              @@   @@
                    @@@    @              @@    @@
                    @ @@   @    @@@@@     @@    @@      @@@
                    @ @@   @   @     @    @@   @@    @@   @@
                    @  @@  @  @@     @@   @@@@@     @@      @@
                    @   @@ @  @@@@@@@@@   @@  @@    @@      @@
                    @   @@ @  @@           @@   @@   @@      @@
                    @    @@@  @@      @    @@   @@   @@      @@
                    @     @@   @@    @@   @@    @@   @@   @@
                   @@@    @@     @@@@   @@@@@   @@@    @@@

                               Negotiating Router

                   Alliance CAD System 5.0 20040928,  nero 5.0
                   Copyright (c) 2002-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

                                S/N 20021117.1


  o  MBK environment :

     MBK_IN_LO       := vst
     MBK_OUT_LO      := vst
     MBK_IN_PH       := ap
     MBK_OUT_PH      := ap
     MBK_WORK_LIB    := .
     MBK_CATA_LIB    := .
                        /usr/local/alliance/cells/sxlib
                        /usr/local/alliance/cells/dp_sxlib
                        /usr/local/alliance/cells/rflib
                        /usr/local/alliance/cells/romlib
                        /usr/local/alliance/cells/ramlib
                        /usr/local/alliance/cells/padlib
     MBK_CATAL_NAME  := CATAL
     MBK_VDD         := vdd
     MBK_VSS         := vss
     MBK_SEPAR       := .

  o  Loading netlist "mux_oo"...
  o  Loading layout "muxoop"...
  o  Flattening layout...
  o  Flattening netlist...
  o  Building netlist dual representation (lofigchain)...
  o  Binding logical & physical views...

  o  Loading design into grid...
     o  Small design, global routing disabled.
     o  Allocating grid size [15,11,3].
     o  Loading external terminals.
     o  Finding obstacles.
     o  Loading nets into grid.
     o  Allocating the net scheduler.
     o  Reading power grid.

  o  Local routing stage.
     - [   4] (hp :=     4) "a"
     - [   3] (hp :=     4) "b"
     - [   2] (hp :=     6) "q"
     - [   1] (hp :=    14) "c"
     - [   0] (hp :=    17) "inv_x2_sig"

  o  Routing stats :
     - routing iterations    := 169
     - re-routing iterations := 0
     - ratio                 := 0%.

  o  Dumping routing grid.
  o  Saving MBK figure "muxoor".
  o  Saving layout as "muxoor"...
```

We told NERO to run in verbose mode, to use the muxoop placement file of the mux_oo netlist and generate an output file called muxoor (all this can be easily confirmed from the above output). NERO gives us a muxoor.ap file that can also be visualized with GRAAL.

```
% graal
```

This time we choose the muxoor.ap file. This is shown in Figure 1.13. The layout processing with OCP and

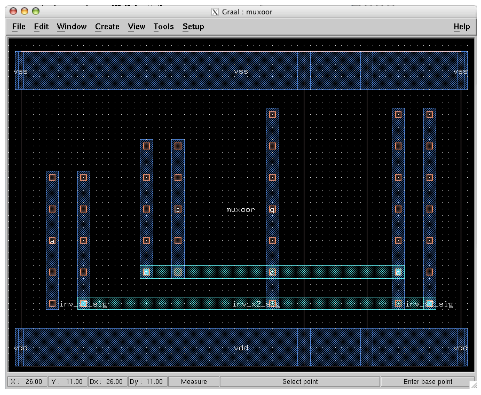


Figure 1.13: The muxoor.ap (mux file routed by NERO) shown by graal.

NERO is shown in Figure 1.14. Once the design is routed, we can now use COUGAR to extract from it the

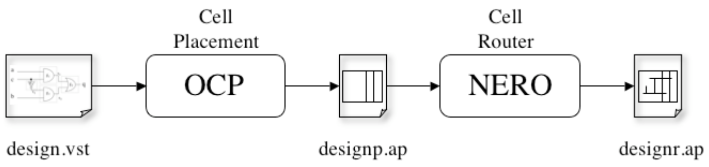


Figure 1.14: The process of a design file with OCP and NERO.

corresponding net list. Before using it we need to set the MBK_OUT_LO environment variable to **al**. We do this with the following command (under a C shell like tcsh or csh)

```
% setenv MBK_OUT_LO al
```

In a bash (or sh) shell it will be,

```
% export MBK_OUT_LO = al
```

And, in a standard Bourne shell,

```
% MBK_OUT_LO = al
% export MBK_OUT_LO
```

We can verify if the setting gone well, by providing the following command.

```
% echo $MBK_OUT_LO
al
```

That will return the environment variable value. Once set this variable we use COUGAR as follows.

```
% COUGAR muxoor muxooc

               @@@@ @
             @@    @@
            @@      @
           @@       @    @@@    @@@  @@@@    @@@@@@   @@@@    @@@ @@@
           @@           @@   @@   @@    @@   @@  @@   @@   @    @@@  @@
           @@          @@     @@  @@    @@   @   @@   @@   @@   @@   @@
           @@          @@     @@  @@    @@   @   @      @@@@@   @@
           @@          @@     @@  @@    @@    @@@     @@   @@   @@
            @@      @ @@      @@  @@    @@   @@      @@    @@   @@
             @@    @@  @@   @@    @@   @@@   @@@@@@  @@   @@@   @@
               @@@@      @@@      @@@@  @@ @@   @@@  @@@@  @@ @@@@
                                             @     @
                                              @@@@@

                       Netlist extractor ... formerly Lynx

                  Alliance CAD System 5.0 20040928, COUGAR 1.21
                  Copyright (c) 1998-2005,       ASIM/LIP6/UPMC
                  Author(s):  Ludovic Jacomme and Gregoire Avot
                  Contributor(s):             Picault Stephane
                  E-mail        :   alliance-users@asim.lip6.fr


        ---> Extract symbolic figure muxoor
        <--- done !

        ---> Total extracted capacitance
        <--- 0.0pF
```

COUGAR takes the routed symbolic layout input file muxoor.ap and generates the extracted netlist muxooc.al. Now we can use another Alliance tool, LVX, to compare the structural file mux_oo.vst and the netlist extracted by COUGAR. We do that as follows.

```
% lvx vst al mux_oo muxooc

                       @@@@@@      @@@@    @@@ @@@@   @@@@
                         @@         @@      @   @@     @
                         @@         @@      @    @@   @
                         @@          @@    @      @@ @
                         @@          @@    @       @@
                         @@           @@  @        @@
                         @@           @@  @        @@@
                         @@            @@@        @  @@
                         @@      @     @@@       @    @@
                         @@     @       @       @      @@
                       @@@@@@@@@@       @      @@@    @@@@

                             Gate Netlist Comparator

                   Alliance CAD System 5.0 20040928,   lvx 1.4
                   Copyright (c) 1992-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

***** Loading mux_oo (vst)...

***** Loading muxooc (al)...

***** Compare Terminals ...........
***** O.K.      (0 sec)

***** Compare Instances ..........
***** O.K.      (0 sec)

***** Compare Connections ..........
***** O.K.      (0 sec)

===== Terminals .......... 6
===== Instances .......... 2
===== Connectors ......... 17

***** Netlists are Identical. *****     (0 sec)
```

The last line tells us that the netlists are identical, in other words that the mux design has been correctly routed. The design verification that includes COUGAR can be visualized as shown in Figure 1.15. COUGAR

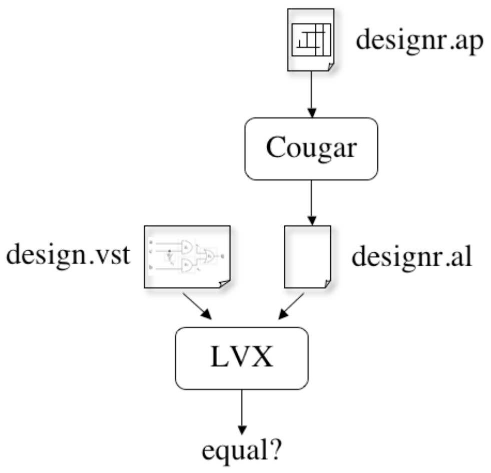


Figure 1.15: The verification flow including COUGAR.

can also extract a SPICE netlist from the design in **ap** format. In this case we need to set the environment variable MBK_OUT_LO to **spi** and MBK_SPI_MODEL (we only show the setting for a C shell).

```
% setenv MBK_OUT_LO spi
% setenv MBK_SPI_MODEL $ALLIANCE_TOP/etc/spimodel.cfg
```

Then, we can extract the SPICE netlist as follows.

```
% COUGAR -t muxoor mux_oo

               @@@@ @
             @@    @@
            @@      @
           @@       @    @@@    @@@  @@@@    @@@@@@   @@@@    @@@ @@@
           @@          @@   @@   @@    @@   @@  @@   @@   @    @@@  @@
           @@         @@     @@  @@    @@   @   @@   @@   @@   @@   @@
           @@         @@     @@  @@    @@   @   @      @@@@@   @@
           @@         @@     @@  @@    @@    @@@     @@   @@   @@
            @@      @ @@     @@  @@    @@   @@      @@    @@   @@
             @@    @@  @@   @@   @@   @@@   @@@@@@  @@   @@@   @@
               @@@@      @@@      @@@@  @@ @@   @@@  @@@@  @@ @@@@
                                            @     @
                                             @@@@@

                       Netlist extractor ... formerly Lynx

                  Alliance CAD System 5.0 20040928, COUGAR 1.21
                  Copyright (c) 1998-2005,       ASIM/LIP6/UPMC
                  Author(s):  Ludovic Jacomme and Gregoire Avot
                  Contributor(s):              Picault Stephane
                  E-mail        :   alliance-users@asim.lip6.fr


        ---> Extract symbolic figure muxoor
        <--- done !

        ---> Total extracted capacitance
        <--- 0.0pF
```

This kind of netlist can then simulated with a SPICE simulator if we provide an adequate transistor model card.

1.6 DREAL

This tool is a hierarchical real layout editor similar to graal (the tool we used to visualize the placed and routed designs), but it has no design rule checker neither an extractor as graal does. We use DREAL providing the following command.

```
% dreal [-l file_name] [-xor] [-debug] [-install] [-force]
```

The options available in DREAL are shown in Table 1.8. This tool is usually used on the output of the Alliance

Table 1.8: Options Available for the DREAL Tool.

Option	Description
-l filename	Load the filename (with or without extension)
-xor	Two graphic cursor methods can be used, invert (default) or xor.
-debug	
-install	Switch to a private color map.
-force	This option force all graphical objects to be displayed.

tool S2R. S2R converts the symbolic layout obtained with NERO into a real technology layout in **cif** or **gds** format. The output format is given by the RDS_OUTPUT environment variable. We can check its value using.

```
% env | grep RDS
RDS_IN=cif
RDS_OUT=cif
RDS_TECHNO_NAME=/usr/local/alliance/etc/cmos.rds
```

We will run S2R on the routed symbolic layout of our mux.

```
% s2r -v muxoor muxcore
                                      @@@@
                                     @   @@
                                    @@    @@
                            @@@@@@  @@@   @@ @@@ @@@
                           @@    @   @   @@   @@@  @@
                           @@@           @    @@   @@
                            @@@@        @     @@
                              @@@@     @      @@
                           @    @@@   @    @  @@
                           @@    @@  @@@@@@   @@
                           @ @@@@@  @@@@@@@  @@@@

                        Symbolic to Real layout converter

                   Alliance CAD System 5.0 20040928,   s2r 5.0
                   Copyright (c) 2002-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

        o loading technology file : /usr/local/alliance/etc/cmos.rds
        o loading all level of symbolic layout : muxoor
        o removing symbolic data structure
        o layout post-treating
                with top connectors,
                with sub connectors,
                with signal names,
                without scotch.
        --> post-treating model ao2o22_x2
            rectangle merging :
        --> post-treating model inv_x2
            rectangle merging :
        --> post-treating model tie_x0
            rectangle merging :
        --> post-treating model muxoor
            ring flattenning  :
            rectangle merging :
        o saving muxcore.cif
        o memory allocation informations
        --> required rectangles = 0  really allocated = 0
        --> Number of allocated bytes: 55448
```

We then use DREAL to see the generated muxcore real layout file (the technology provided by Alliance is not a real 0.35 um technology, but it is useful for the example we are handling). Remember that DREAL requires a X11 environment, before trying to use the following command to run DREAL.

```
% dreal -l muxcore
```

The corresponding real layout (after flattening) is shown in Figure 1.16.

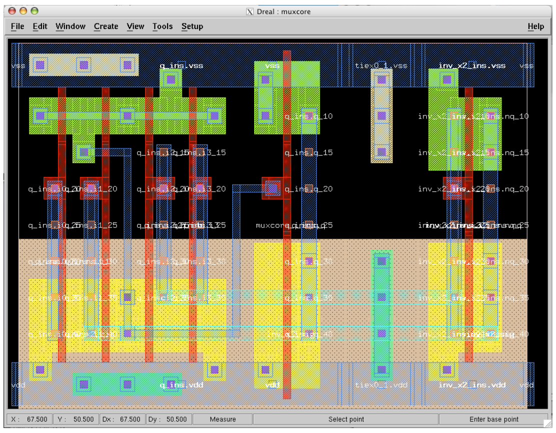


Figure 1.16: The real layout of our mux shown by DREAL.

1.7 DRUC

DRUC (Design RUle Checker) is the Alliance parametrized VLSI design rule checker. The rules it uses are defined by the RDS_TECHNO_NAME environment variable. DRUC flattens all the hierarchy of the design to check if there is any violation to the specified design rules, so when applying it is necessary to assure that root and instantiated cells are in the current directory. Let's apply DRUC to our muxoor.ap design file to check it.

```
% druc -v muxoor

                  @@@@@@@     @@@@@@@                    @@@@ @
                    @@   @@     @@   @@                @@    @@
                    @@    @@    @@    @@              @@      @
                    @@     @@   @@    @@  @@@  @@@@  @@       @
                    @@     @@   @@   @@    @@    @@  @@
                    @@     @@   @@@@@      @@    @@  @@
                    @@     @@   @@  @@     @@    @@  @@
                    @@     @@   @@   @@    @@    @@  @@
                    @@    @@    @@   @@    @@    @@   @@      @
                    @@   @@     @@    @@   @@   @@@    @@    @@
                  @@@@@@@     @@@@@   @@@   @@@@  @@     @@@@

                               Design Rule Checker

                   Alliance CAD System 5.0 20040928,  druc 5.0
                   Copyright (c) 1993-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

Flatten DRC on: muxoor
Delete MBK figure : muxoor
Load Flatten Rules : /usr/local/alliance/etc/cmos.rds

layer RDS_NWELL 4.;
layer RDS_NTIE 2.;

....intermediate output has been deleted for space reasons.

fin regles
Unify : muxoor

Create Ring : muxoor_rng
Merge Errorfiles:

Merge Error Instances:
instructionCourante :  56
End DRC on: muxoor
Saving the Error file figure
Done
  0

File: muxoor.drc is empty: no errors detected.
```

As could be notice from the output, no errors has been detected. And the layout generated by OCP and NERO does not contain violations to the design rules specified in the cmos.rds file.

The options available in DRUC are shown in Table 1.9. The option **-h** generates many intermediate files, but there is no documentation about its use.

Table 1.9: Options Available for the DRUC Tool.

Option	Description
-v	Verbose mode on. Each step of the DRC is output on the standard output
-h	Hierarchical design rule checking. Generates lots of files locally, to be used by future invocations of druc.

1.8 FlatBeh

The FlatTBeh (**Flat**tens to **Beh**avioral) tool generates a behavioral description from a structural one. It could be used to check if the description obtained in the synthesis process of a design is correct. It flattens the structural description of the target design down to its leaf behavioral components. Then, it uses those descriptions to create a behavioral description of the target design. This tool can be started with the following command.

```
flatbeh root_structural_file  [ output_file ]
```

This tool has no options. Its parameters are only the root structural design and the ouput file name. We must check that MBK_CATA_LIB is set to the path where the descriptions of the design are given. MBK_IN_LO must be set to vst. Also, if needed, we can set MBK_CATAL_NAME to the file (normally CATAL) that lists the files names of the behavioral description where FlatBeh must stop the flattening of the structural root design. Let' apply it to our mux_oo.vst design generated by BOOG.

```
% flatbeh mux_oo mux_oo

      @@@@@@@@@  @@@@                      @@@@@@@               @@@
        @@    @    @@                @       @@   @@              @@
        @@     @   @@               @@       @@    @@             @@
        @@         @@     @@@@      @@       @@    @@    @@@@@    @@ @@@
        @@   @     @@    @@   @   @@@@@@@@   @@   @@    @     @   @@@  @@
        @@@@@@     @@    @@   @@    @@       @@@@@@    @@     @@  @@    @@
        @@   @     @@      @@@@@    @@       @@    @@  @@@@@@@@@  @@    @@
        @@         @@    @@   @@    @@       @@     @@ @@         @@    @@
        @@         @@   @@    @@    @@       @@     @@ @@      @  @@    @@
        @@         @@   @@   @@@    @@   @   @@    @@   @@    @@  @@    @@
      @@@@@@     @@@@@@  @@@@  @@    @@@@  @@@@@@@@       @@@@   @@@@  @@@@

                              a netlist abstractor

           Alliance CAD System 5.0 20040928, flatbeh 5.0 [2000/11/01]
           Copyright (c) 1993-2005,                    ASIM/LIP6/UPMC
           Author(s):                Fran?ois DONNET, Huu Nghia VUONG
           E-mail        :                alliance-users@asim.lip6.fr

     =========================  Environnement  =========================
     MBK_WORK_LIB        = .
     MBK_CATA_LIB        = .:/usr/local/alliance/cells/sxlib:/usr/local/alliance/cells/dp_sxlib
    :/usr/local/alliance/cells/rflib:/usr/local/alliance/cells/romlib:/usr/local/alliance/cells/ramlib
    :/usr/local/alliance/cells/padlib
     MBK_CATAL_NAME      = CATAL
     ============================== Files ==============================
     Netlist file        = mux_oo.vst
     Output  file        = mux_oo.vbe
     ===================================================================

Loading './mux_oo.vst'
flattening figure mux_oo
 loading inv_x2
 loading ao2o22_x2
Restoring array's orders
BEH : Saving 'mux_oo' in a vhdl file (vbe)
```

We can test this circuit with ASIMUT using a modified version of mux_in.pat (muxoo_in.pat).

```
 -- terminals of the mux circuit
 in  vdd   B;
 in  vss   B;
 in  c     B;
 in  a     B;
 in  b     B;
 out q   B;
  -- time interval description of the input values and expected output
 begin
 <0  ns> mux_test           : 10 000 ?0;
 <+1 ns> notc_nota_notb     : 10 000 ?0;
 <+1 ns> notc_nota_b        : 10 001 ?1;
 <+1 ns>                    : 10 001 ?1;
 <+1 ns> notc_a_notb        : 10 010 ?0;
 <+1 ns>                    : 10 010 ?0;
 <+1 ns> notc_a_b           : 10 011 ?1;
 <+1 ns>                    : 10 011 ?1;
 <+1 ns> c_nota_notb        : 10 100 ?0;
 <+1 ns>                    : 10 100 ?0;
 <+1 ns> c_nota_b           : 10 101 ?0;
 <+1 ns>                    : 10 101 ?0;
 <+1 ns> c_a_notb           : 10 110 ?1;
 <+1 ns>                    : 10 110 ?1;
 <+1 ns> c_a_b              : 10 111 ?1;
 <+1 ns>                    : 10 111 ?1;
 end;
```

We use this file as input for ASIMUT.

```
% asimut -b mux_oo muxoo_in muxoo_out

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @   @@   @
            @@@     @@     @     @                           @    @@    @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@       @@
           @  @@     @@@@    @@@@    @@@ @@  @@   @@    @@        @@
           @  @@       @@@@    @@    @@  @@  @@   @@    @@        @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@        @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@        @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@        @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@        @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMUlation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching 'mux_oo' ...
BEH : Compiling 'mux_oo.vbe' (Behaviour) ...
making GEX ...
searching pattern file : 'muxoo_in' ...
restoring ...
linking ...
executing ...
###----- processing pattern 0 : 0 ps -----###
###----- processing pattern 1 : 1000 ps -----###
###----- processing pattern 2 : 2000 ps -----###
###----- processing pattern 3 : 3000 ps -----###
###----- processing pattern 4 : 4000 ps -----###
###----- processing pattern 5 : 5000 ps -----###
###----- processing pattern 6 : 6000 ps -----###
###----- processing pattern 7 : 7000 ps -----###
###----- processing pattern 8 : 8000 ps -----###
###----- processing pattern 9 : 9000 ps -----###
###----- processing pattern 10 : 10000 ps -----###
###----- processing pattern 11 : 11000 ps -----###
###----- processing pattern 12 : 12000 ps -----###
###----- processing pattern 13 : 13000 ps -----###
###----- processing pattern 14 : 14000 ps -----###
###----- processing pattern 15 : 15000 ps -----###
```

Checking in this way that the generated behavioral description is also correct.

1.9 FLATLO

The FLATLO (FLATtens LOgical) tool inputs a hierarchical netlist and flattens it to a given level in either a **vst** or Alliance logical **al** format. The general format for using this tools is,

```
% flatlo [option] logical_figure [instance] output_name
```

The options available in FLATLO are shown in Table 1.10. If the **instance** is indicated it is flattened in the logical description.

Table 1.10: Options Available for the FLATLO Tool.

Option	Description
-r	Flattens the target file to the catalog.
-t	Flattens the target file to the transistor level.

1.10 FLATPH

The FLATPH (FLATtens PHysical) tool flattens a hierarchical symbolic layout to a given level. The tool is used with the following commnad line.

```
% flatph [option] logical_figure [instance] output_name
```

The options available in are shown in Table 1.11. If the **instance** is indicated it is flattened in the physical description.

Table 1.11: Options Available for the FLATPH Tool.

Option	Description
-r	Flattens the target file to the catalog.
-t	Flattens the target file to the transistor level.

1.11 FMI

The FMI (Finite state machine MInimizer) tool takes a FSM description (.fsm file) and minimizes it into an equivalent reduced (in the number of states) FSM. We use this tool using the following command.

```
fmi [-V] input_file  output_file
```

The options available in FMI are shown in Table 1.12. Let's apply this tool to the FSM we generated in the

Table 1.12: Options Available for the FMI Tool.

Option	Description
-V	Verbose mode. Each step of the minimization is displayed.

example used with the B2F tool.

```
% fmi -V rwgraph rwmin
                          @@@@@@@@@  @@@     @@@ @@@@@@
                            @@    @   @@     @@    @@
                            @@     @  @@@   @@@    @@
                            @@        @@@   @@@    @@
                            @@   @    @ @@ @ @@    @@
                            @@@@@@    @ @@ @ @@    @@
                            @@   @    @ @@@  @@    @@
                            @@        @  @@  @@    @@
                            @@        @  @   @@    @@
                            @@        @      @@    @@
                          @@@@@@     @@@    @@@@ @@@@@@

                                FSM Minimization

                   Alliance CAD System 5.0 20040928,   fmi 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s): Fr?d?ric P?trot, Ludovic Jacomme
                   Contributor(s):        Jean-Marie Alexandre
                   E-mail        : alliance-users@asim.lip6.fr

        --> Run FSM Compiler
        --> Compile file rwgraph
        --> Build Binay Decision Diagrams
        --> Identify equivalent states
        --> Initial number states    : 4
        --> Number of deleted states : 1
        --> Save file rwmin
```

As could be noticed from the above output one state has been deleted, now having the following FSM.

```
-- DATE : Thu Jan 27 23:56:30 2005
ENTITY rwmin IS
PORT
(vss : IN BIT;
  vdd : IN BIT;
  busy : OUT BIT;
  ne : IN BIT;
  rw : IN BIT;
  start : IN BIT;
  clk : IN BIT);
END rwmin;

ARCHITECTURE FSM OF rwmin IS
TYPE STATE_TYPE IS ( s0,s3,s2 );
SIGNAL CURRENT_STATE, NEXT_STATE: STATE_TYPE;
--PRAGMA FIRST_STATE s0
--PRAGMA CLOCK clk
BEGIN
  PROCESS( CURRENT_STATE, vss, vdd, ne, rw, start)
  BEGIN
  CASE CURRENT_STATE IS
    WHEN s3 =>
      IF (( '1') = '1') THEN NEXT_STATE <= s0; END IF;
      busy <= '1';
    WHEN s2 =>
      IF (( ne) = '1') THEN NEXT_STATE <= s3;
      ELSE NEXT_STATE <= s2; END IF;
      busy <= '1';
    WHEN s0 =>
      IF (( ((start AND NOT(rw)) OR (NOT(start AND NOT(rw)) AND (start AND rw)))) = '1')
      THEN NEXT_STATE <= s2;
      ELSE NEXT_STATE <= s0; END IF;
      busy <= '0';
  END CASE;
  END PROCESS;

  PROCESS( clk )
  BEGIN
    IF ( (NOT((clk'STABLE)) AND clk) )
    THEN CURRENT_STATE <= NEXT_STATE;
    END IF;
  END PROCESS;
END;
```

1.12 FSP

The FSP (Finite State machine Proof) tool checks the equivalence of two FSM descriptions. It does it formally using a product based algorithm. We use this tool with the following command.

```
% fsp [-V] format1 format2 file1  file2
```

The options available in FSP are shown in Table 1.13. Let's apply it to the fsm descriptions we got in the examples using B2F and FMI.

Table 1.13: Options Available for the FSP Tool.

Option	Description
-V	Verbose mode. Each step of the formal proof is displayed.

```
% fsp -V fsm fsm rwgraph rwmin

                         @@@@@@@@@    @@@@ @  @@@@@@@
                           @@    @   @    @@    @@   @@
                           @@     @ @@     @    @@    @@
                           @@       @@@         @@    @@
                           @@   @    @@@@       @@   @@
                           @@@@@@      @@@@     @@@@@
                           @@   @        @@@    @@
                           @@       @      @@   @@
                           @@       @@     @@   @@
                           @@       @@@    @    @@
                         @@@@@@     @  @@@@   @@@@@@

                                FSM formal Proof

                   Alliance CAD System 5.0 20040928,   fsp 5.0
                   Copyright (c) 1999-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr

        --> Run FSM Compiler
        --> Compile file rwgraph
        --> Run FSM Compiler
        --> Compile file rwmin
        --> Formal proof between "rwgraph.fsm" and "rwmin.fsm"

        ==> "rwgraph.fsm" and "rwmin.fsm" are identicals
```

As could be noticed from the above output, the original FSM description obtained with B2F and the one obtained with FMI are identical.

1.13 GENPAT

GENPAT (GENerator of PATterns) is a tool that is used to create a file of input patterns for ASIMUT. It is a C interface (set of functions) that ease the creation of the input patterns. This tool inputs a C file describing the input patterns and generates as output a **.pat** file. As in other tools the extension is not given at the command input. As we explained in the section describing ASIMUT a **.pat** file can be easily edited for small circuits, but it becomes complicated for complex design. GENPAT is intended to automate this editing work. We use GENPAT with the following command.

```
% genpat [-v] [-k] [file]
```

The options available in GENPAT are shown in Table 1.14. We will try genpat generating the input patterns

Table 1.14: Options Available for the GENPAT Tool.

Option	Description
-v	Verbose mode.
-k	Keeps the executable and the compilation Makefile after completion.

for our multiplexer circuit using the following muxgp.c file.

```
#include <stdio.h>
#include <genpat.h>

/*Define the number of test patterns */
#define MAXCYCLE 8

/* Function to allocate a memory buffer, convert the first parameter into a string
and return the pointer to it.*/

char *inttostr (int integer, int len)
 {
  char *str;
  str = (char *)mbkalloc(len*sizeof(char)+1);
  sprintf(str, "%.32d", integer);
  return(&str[32-len]);
 }

/* Apply power to the circuit */
void power (void)
 {
  AFFECT (inttostr (0, 32), "Vdd", "0b1");
  AFFECT (inttostr (0, 32), "Vss", "0b0");
 }

/*Generate the patterns for the mux */
void patterns (void)
 {
  int i,a,b,c,q;
  for (i=0; i<MAXCYCLE; i++)
   {
    LABEL ( "pat" );
    if (i%8 < 4) c = 0; else c = 1;
    AFFECT(inttostr(i, 32), "c", inttostr(c,32));
    if (i%4 < 2) a = 0; else a = 1;
    AFFECT(inttostr(i, 32), "a", inttostr(a,32));
    if(i%2 == 0) b = 0; else b = 1;
    AFFECT(inttostr(i, 32), "b", inttostr(b, 32));
    if(c == 1) q = a; else q = b;
    AFFECT(inttostr(i, 32), "q", inttostr(q, 32));
   }
 }

int main(void)
 {
  /* Declares the name of the pattern file */
  DEF_GENPAT ("muxgp_in");

  /* Declare the name, format and type of each of the inputs/outputs */
  DECLAR ( "Vdd", ":0", "B", IN, "", "");
  DECLAR ( "Vss", ":5", "B", IN, "", "");
  DECLAR ( "a", ":2", "B", IN, "", "");
  DECLAR ( "b", ":2", "B", IN, "", "");
  DECLAR ( "c", ":2", "B", IN, "", "");
  DECLAR ( "q", ":2", "B", OUT, "", "");

  /* Generate the patterns */
  power();
  patterns();

  /* Save the pattern file */
  SAV_GENPAT();
  /* Return success code */
  return(0);
 }
```

We can check if the input pattern file is correct viewing it with xpat. You must see the output shown in Figure 1.17. There are several functions that can be used in the input files of GENPAT, we expect to describe

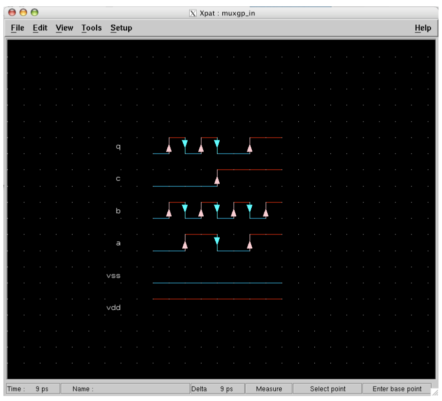


Figure 1.17: The muxgp_inpat file shown by XPAT.

them in one of the annexes of the book.

1.14 GRAAL

GRAAL is the hierarchical symbolic layout editor of Alliance. It is a graphical tool that requires an X11 and Motif installation. GRAAL functionalities can be accessed through its seven menus. GRAAL could be called with a command as the one that follows.

```
% graal [-l file_name] [-scale n] [-debug] [-xor] [-install] [-force]
```

The options available in are shown in Table 1.15. We have already seen an application of GRAAL in the

Table 1.15: Options Available for the GRAAL Tool.

Option	Description
-l filename	Load the filename (with or without extension)
-scale n	Scale the filename (range of n is not explained in the graal man pages)
-debug	
-xor	Two graphic cursor methods can be used, invert (default) or xor.
-install	Switch to a private color map.
-force	This option force all graphical objects to be displayed.

placement and routing of our multiplexer, so we will skip it here.

1.15 K2F

K2F is a command line tool of Alliance that translates a FSM description in Alliance format (.fsm) to Berkeley (.kiss2) format or viceversa. It could be called with the following command.

```
k2f In_format Out_format Input_name [ Output_name ]
```

If the Output_name is not given the Input_name will be used. There are no options available in K2F. The KISS (Keep It Simple Stupid) format can be used to simplify an FSM description with the SIS or VIS tools developed by the University of California at Berkeley. Let's try this tool on the example used in the explanation of the

B2F tool.

```
% k2f fsm kiss2 rwgraph

                         @@@@   @@@@   @@@@   @@@@@@@@@
                          @@     @    @   @@    @@    @
                          @@    @    @@    @@   @@     @
                          @@   @     @@@   @@   @@
                          @@  @@      @   @@    @@   @
                          @@ @ @@          @     @@@@@@
                          @@@  @@         @      @@   @
                          @@    @@      @       @@
                          @@    @@     @    @   @@
                          @@     @@   @@@@@@    @@
                         @@@@   @@@@ @@@@@@@  @@@@@@

                         Translator Kiss <-> FSM Format

                   Alliance CAD System 5.0 20040928,   k2f 5.0
                   Copyright (c) 1996-2005,     ASIM/LIP6/UPMC
                   Author(s):                 Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr



        --> Translate figure rwgraph
        <-- done
```

That will give us the following rwgraph.kiss2 file.

```
.i 5
.o 1
.s 4
.r s0
.p 8

10--- s0 s2 0
11--- s0 s1 0
0---- s0 s0 0
--1-- s1 s3 1
--0-- s1 s1 1
--1-- s2 s3 1
--0-- s2 s2 1
----- s3 s0 1
.e
```

You can also try to simplify this description with any of the indicated Berkeley tools (if you have installed them in your system).

1.16 L2P

The command line tool L2P creates a PostScript (.ps) file from a symbolic or real layout file input. The options available in L2P are shown in Table 1.16. A PostScript dictionary is a set of macros used during the interpretation of a PostScript file. By modifying the macros in the internal PostScript dictionaries of L2P you can control which layer to output, how to plot (fill) the rectangles(empty, hashed, filled), specify the colors of the rectangles , and many other things. The generated postscript file is mainly formed of a dictionary and orders of drawing rectangles. If no options are indicated L2P will generate an standard output file. This is generally enough for leaf cell designs, but maybe not for large circuits. Let's generate the PostScript file of our

Table 1.16: Options Available for the L2P Tool.

Option	Description
-color	Generates a color PostScript file. The default value gives a black and white PostScript file.
-drawingsize=$w \times h$	Specifies the drawing area in cents of a inch. By default, wide =725 and height = 1068 for french A4 paper. If the drawing size is bigger than the paper area, then the drawing will be splitted on several pages.
-fA3	The drawing is done on A3 format paper.
-fLETTER	The drawing is done on LETTER format paper.
-fLEGAL	The drawing is done on LEGAL format paper.
-givebwdict	Give the Black & White internal PostScript dictionary. See below.
-givecolordict	This option must be unique on the command line. When used as in 'l2p -givebwdict', l2p gives on the standard output its Black & White internal Postscript dictionary. You get the standard dictionary by a line of the form 'l2p -givecolordict > dict.ps'. You can then edit it, in order to reuse it with l2p, see the '-usedict' option below.
-help	Gives you this man page that explains how to use l2p.
-noheader	Prevents the border and various info, as the cellname and the position of the page in the drawing, from being printed.
-papersize=$x \times y \times w \times h$	Specifies the paper area in cents of a inch. By default, 50x50x726x1069 is used for a4 paper (x and y low left points, and width and height values, respectively).
-pages=$x\_pages \times y\_pages$	Specifies the drawing area in pages. It can be useful, instead of having to calculate the size in cenths of inch of the drawing, to give it in numbers of pages. It takes care of the resizing of the paper and whether there is a header.
-tsize=s	Available sizes:6,8,10,12,14.The default value is 8.
-nrname	No name at all will be displayed.
-ncname	The external connector's names won't be displayed.
-nrfname	The references's names won't be displayed.
-niname	The instances's names won't be displayed.
-nsname	The segments's names won't be displayed.
-mfeed	Manual feed. If set, it informs the printer that it will be fed by the user himself, for each printing.
-real	Uses real file (cif, gds). By default, uses symbolic layout file (ap, cp).
-resol=x	Sets the resolution of the file in dots per inch (dpi) and therefore limitate the size of the PostScript. Each rectangle whose width and height are smaller than the resolution will not be printed. The default value is 72dpi. It should only be changed in one wants to produce a plot of several meters. Usually, you must provide a PostScript file sized for A4 paper with a much better resolution than 72dpi. You can then increase that value to up to 1000dpi, but be aware that the size of the file will probably be bigger than a 44Mb SyQuest cartridge that is used in PAO for exchanging data files.
-rflattencatal	Flattens the cell to the catalog level. Be careful, this option requires a lot of memory.
-rflattentrans	Flattens the cell to the transistor level before printing. Be careful, this option requires a lot more memory.
-rotate	Rotate the cell 90 degrees. This is useful if you have a wide cell, and you want it printed in landscape mode.
-scale=f.f	Forces the cell to be printed with a certain scale (a floating-point number). This is very useful, when you are printing a whole library of cells, and you want all cells to be printed to the same scale. You can find at which scale a cell was printed by looking at the beginning of the file : 'head n1_y-1x1.ps' will show you a PostScript comment beginning by '%SCALE=3.78435' for example.
-usedict=filename	The output Postcript file contains a Postcript dictionary of macros. There are two standard dictionnaries used by l2p. One for black and white and another for color prints. This allows you to use a PostScript dictionary different from the two internally encoded into l2p.

symbolic routed multiplexer design.

```
% l2p muxoor

                         @@@@@@        @@@@   @@@@@@@
                           @@         @   @@    @@   @@
                           @@        @@    @@   @@    @@
                           @@        @@@   @@   @@    @@
                           @@         @   @@    @@   @@
                           @@             @     @@@@@
                           @@            @      @@
                           @@           @       @@
                           @@      @   @    @   @@
                           @@     @   @@@@@@    @@
                         @@@@@@@@@@  @@@@@@@  @@@@@@

               (L)ayout to(2) (P)aper: A tool for PostScript plots

                   Alliance CAD System 5.0 20040928,   l2p 5.0
                   Copyright (c) 1994-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr


- Loading technology file : /usr/local/alliance/etc/cmos.rds
- Loading mbk figure: muxoor.ap
- Saving PostScript figure: muxoor
        81 masks, BBox=(x=-4,y=0,w=288,h=200)
        Paper=(0.500,0.250) 7.400 inch(es) wide, 11.190 inch(es) tall
        Drawing= 7.389 inch(es) wide, 10.931 inch(es) tall
        width first, SCALE=1.847222, resol=100dpi  Centering:(0,113)
        One page   PS(36,36,532,787)   RDS(-4,0,288,426)
- All done.
```

This will produce the black and white (not flattened) view of the routed design of the multiplexer as shown in Figure 1.18.

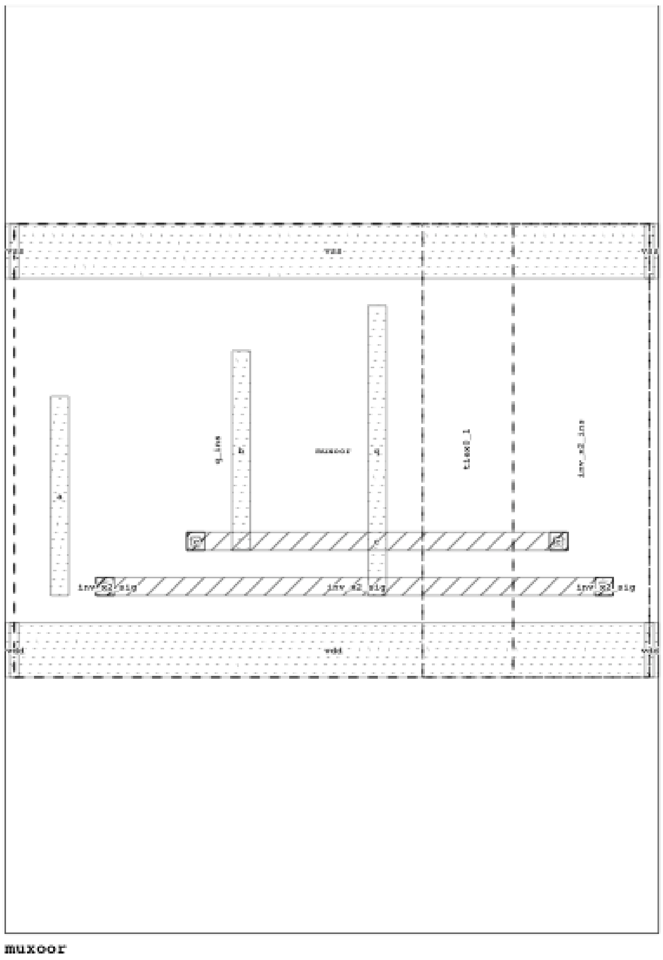


Figure 1.18: Snapshot of the Postcript file generated by L2P (default setting).

1.17 LOON

LOON (Light Optimizing On Nets) is a command line tool of Alliance that is used in the synthesis process after applying BOOM and BOOG to a design. Its input netlist could be hierarchical (it will be flattened if necesary). LOON is used with the following command line.

```
% loon  loon <input_file> <output_file> [-l <lax_file>] [-x <xsch_mode>] [-m <optim_mode>]
```

The options available in LOON are shown in Table 1.17. Let's apply LOON to the multiplexer design we

Table 1.17: Options Available for the LOON Tool.

Option	Description
-h	Help mode. Displays possible uses of loon.
-o outfile	Overwrites the source file if no output_file is given. This can be useful if you don't want several netlist files.
-m optmode	Optimization mode. Can be defined in a lax file, it's a shortcut to define it on the command line. The mode number takes a value between 0 and 4. It indicates the type of optimization the user wants. If 0 is given, the circuit area will be improved. On the other hand, 4 will improve circuit delays. 2 is a medium value for optimization.
-x xschmode	Generate a '.xsc' file. Which is a color map for the signals contained in the output_file. This file is used by xsch when viewing the netlist. By choosing level 0 or 1 for xsch_mode, you can color respectively the critical path or all signals with delay graduation.
-l laxfile	Just another way to indicate explicitely the LAX parameter file name.

optimized with BOOM and BOOG.

```
% loon mux_oo mux_ool -x 0 -m 0
                   @@@@@@                          @@@    @@@
                     @@                             @@     @
                     @@                             @@@    @
                     @@           @@@       @@@     @ @@   @
                     @@         @@   @@   @@   @@   @ @@   @
                     @@        @@     @@ @@     @@  @  @@  @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@      @ @@     @@ @@     @@  @    @@@
                     @@     @   @@   @@   @@   @@   @     @@
                   @@@@@@@@@@     @@@       @@@    @@@    @@
                           Local optimization on Nets
             Alliance CAD System 5.0 20040928, loon 5.0 [2003/12/07]
             Copyright (c) 2000-2005,                 ASIM/LIP6/UPMC
             Author(s):                              Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr
        MBK_IN_LO       : vst
        MBK_OUT_LO      : vst
        MBK_TARGET_LIB  : /usr/local/alliance/cells/sxlib

Reading default parameter...
100% area optimization
Reading file 'mux_oo.vst'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Capacitances on file 'mux_oo.vst'...
Delays on file 'mux_oo.vst'...657 ps
Area on file 'mux_oo.vst'...3000 lamda? (with over-cell routing)
Details...        ao2o22_x2: 1 (75%)
                  inv_x2: 1 (25%)
                  Total: 2
Worst RC on file 'mux_oo.vst'...16 ps
Improving RC on critical path for file 'mux_ool.vst'...643 ps
Improving all RC for file 'mux_ool.vst'...
Worst RC on file 'mux_ool.vst'...16 ps
Area on file 'mux_ool.vst'...3000 lamda? (with over-cell routing)
Details...        ao2o22_x2: 1 (75%)
                  inv_x2: 1 (25%)
                  Total: 2
Critical path (no warranty)...643 ps from 'c' to 'q'
Saving file 'mux_ool.vst'...
Saving critical path in xsch color file 'mux_ool.xsc'...
End of loon...
```

Of course, in the case of this very simple design, there will be no changes in the description already gotten with BOOG, since the cells of it did not change.

1.18 LVX

LVX (Logical Versus eXtracted) tool of Alliance let us compare two gate-level or block-level netlists. This tool is used to compare a specification (logical netlist) to the netlist extracted from a physical netlist. Using this tool we can verify if the placement and routing of a logical design has been done correctly. Since this is a one-level hierarchical tool we must use COUGAR to extract a netlist for comparison. We use LVX with the following command.

```
lvx format1 format2 filename1 filename2 [ -a ] [ -o ] [ -f ]
```

The options available in LVX are shown in Table 1.18. If the option -f is used the two netlists are flattened

Table 1.18: Options Available for the LVX Tool.

Option	Description
-a	Some routers generate layout with several physical connectors for power and ground (VDD or VSS). If those connectors are not internally connected, they will have different indexed names (VDD1 , VDD2 etc...) in the extracted netlist. It is possible to perform reduction on those power and ground connectors before comparison, using the -a option. After reduction, each instance contains only one VDD connector and one VSS connector, as the main figure.
-o	In this case, lvx produces a modified netlist (saved with the name filename2), which is a copy of netlist 2 with ordered connectors. Terminals and instance connectors are relisted in the order of the models in net-list 1. The saved netlist is done with the MBK_OUT_LO(1) format, so user has to set this variable before running lvx . If he does not, default value is used, and netlist 1 could be lost if filename are identical and input format same as output format. If -a option is used, then the saved net-list is the reduced net-list with only one VDD and one VSS .
-f	The two netlists are flattened to the leaf cells contained in the catalog file. Usually the extracted netlist is a flatten netlist, while the logical one can be a hierarchical net-list.

using the values of the environment variables MBK_CATA_LIB and MBK_WORK_LIB. MBK_CATA_LIB indicates the path to the cell library catalog file, and MBK_WORK_LIB the path where the file indicated by MBK_CATAL_NAME indicates the user blocks that must not be flattened.

LVX compares external connectors names, instances names, names of connections and unconnected signals, in this order. If connector or instance names do not match LVX stops with an error. It will not compare signals if terminal or instance names differ.

We have already seen an example of application of LVX in the section of COUGAR, so will skip it here.

1.19 MOKA

The MOKA (MOdel checKer Ancestor) tool of Alliance let's check an FSM or RTL description using a list of formulae described in a CTL (Control Temporal Logic) file. MOKA has some restrictions. It needs that all the register in the behavioral description have the same clock condition and doesn allows the use of tristate or multiplexed buses (The mux_bit and wor_bit are not supported). We use MOKA with the following command.

```
% moka [-VDB] fsmfile ctlfile
```

The options available in MOKA are shown in Table 1.19. Let's use MOKA on the example provided for this

Table 1.19: Options Available for the MOKA Tool.

Option	Description
-V	Sets verbose mode on.
-D	Sets debug mode on. Each step of the model checking is detailed on the standard output. In particular all states set are displayed for each CTL sub-expression.
-B	The input file is a VHDL description using the Alliance VHDL subset

tool in its manual pages. We have called both files fsm.vhd and fsm.ctl respectively. We first convert the VHDL file in a behavioral description using the following command

```
% vasy -Vao -I vhd fsm
```

After the conversion has been done. We run MOKA as follows.

```
% moka -VB fsm fsm

                 @@@     @@@           @@@@   @@@@       @
                  @@     @@             @@     @         @
                  @@@   @@@             @@    @         @@@
                  @@@   @@@     @@@     @@   @          @@@
                  @ @@ @ @@   @@   @@   @@  @@         @  @@
                  @ @@ @ @@  @@     @@  @@ @ @@        @  @@
                  @ @@@  @@  @@     @@  @@@  @@       @    @@
                  @  @@  @@  @@     @@  @@    @@      @@@@@@@
                  @  @   @@  @@     @@  @@    @@     @      @@
                  @      @@   @@   @@   @@     @@    @      @@
                 @@@    @@@@    @@@    @@@@   @@@@ @@@@    @@@@

                            MOdel checKer Ancestor

                   Alliance CAD System 5.0 20040928,  moka 5.0
                   Copyright (c) 2002-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr

        --> Running BEH Compiler
            Compiling file fsm
        --> Verifying BEH figure
        --> Running CTL Compiler
            Compiling file fsm
        --> Building BDDs
        --> Model checking of "fsm"
            Checking formula prop1
            EX((ack = '1'))
            > states set:
               (NOT(a_cs) AND b_cs)
               (NOT(a_cs) AND NOT(b_cs)) (first state)
            Checking formula prop2
            AG((req -> AF(ack)))
            > all reachable states
            Checking formula prop4
            ((req = '1') AU (ack = '1'))
            > states set:
               (a_cs AND b_cs)
               (NOT(a_cs) AND b_cs)
               (NOT(a_cs) AND NOT(b_cs)) (first state)
        <-- done
```

1.20 NERO

NERO (NEgotiating ROuter) is a simple over the cell router provided in Alliance that can process designs of up to 4K gates in size. It takes as inputs a netlist and a placement of it. By default it assumes that the netlist and placement file have the same name. The user can provide with an optional different name for the placement file.

If a design has a half perimeter greater than 800 lambdas, NERO will use "global routing" to route it. When NERO uses "global routing", the longest nets will be routed first using layers 3 and 4. Then, the remaining nets will be routed using all other available layers. When global routing is used at least four layers are used for routing. The nets in this case are routed from the shortest to the longest one with the same routing algorithm. NERO provides good results with small netlists (less than 500 gates) and could not converge when handling complex netlists. We use NERO with the following command.

```
% nero [options] netlist layout
```

The netlist and layout file names must be different or the first one will be overwritten. The options available in NERO are shown in Table 1.20. We already shown an application of NERO to our mux example in the section explaining , so we will not repeat it here. Other examples will be given in the chapters showing more complex design examples.

Table 1.20: Options Available for the NERO Tool.

Option	Description
-h, –help	Prints help.
-v, –verbose	Verbose mode.
-V, –very-verbose	Unbearably verbose.
-c, –core-dump	Allows core dump in case of a crash.
-2, -3, -4, -5, -6	Sets the numbers of layers used for routing. The default for small designs is 2 and 4 for big ones.
-L, –local	Turns off the global routing stage, whatever the size of the design is. Be warned that this will enormously slow down the routing process.
-G, –global	Turns on the global routing regardless of the size of the design.
-p file, –place file	Specifies a name for the placement file different from the netlist name.

1.21 OCP

The OCP tool is used for placement in Alliance. It uses as input a netlist of standard cells. The netlist format can be structural VHDL, EDIF or Alliance internal format. This format is given by the MBK_IN_LO variable. Optionally, we can also give a file indicating the placement of connectors (ioc type file). If nonstandard blocks are used in the design, one has to provide the placement for those blocks. The output of the placer will be a physical layout file with placed cells and connectors. The format of the output is defined by the MBK_OUT_PH environment variable . We use OCP with the following command.

```
% ocp [options] netlist outputname
```

The options available in OCP are shown in Table 1.21. Let's use OCP to place the LOON optimized multiplexer

Table 1.21: Options Available for the OCP Tool.

Option	Description
-partial NAME	Defines the user defined placement file. It must have an extension equal to that defined by MBK_IN_PH.
-c	Will make OCP place the connectors randomly on each side of the abutment box.
-ring	Makes OCP to place the connectors for the ring pad placement tool. The placement is random but only on the north and south side unless the -ioc option is given, in which case the constraints given in the NAME.ioc file will be respected, but with layers suitable for ring. This option is temporary and is provided for the ring pad placement tool.
-ioc NAME	The placement of connectors will be done accordingly to what is specified in the NAME.ioc file.
-margin m	Sets the amount of free area added in percentage of the cells area. The resulting area will be equal to CELL_AREA * (1 + m). By default, the margin value is 0.2 (20%)
-eqmargin	Will distribute equitably the margin between cells. By default the largest number of 2-pitch tie cells is inserted to have the best well and bulk polarity.
-rows NR	Forces the design to be placed in NR rows. This option will not be used if a defined placement file is used.
-v	Verbose mode.
-gnuplot	Generates gnuplot files for editing statistics.

design (mux_ool.vst), but this time will make OCP also place the connectors of the design. Placing the connectors will prepare the design for the ring pad placement. In this case we use the following command.

```
% ocp -ring mux_ool muxoolpring

                            @@@         @@@@ @ @@@@@@@
                          @@   @@     @@    @@   @@   @@
                         @@     @@   @@      @   @@    @@
                        @@       @@ @@       @   @@    @@
                        @@       @@ @@           @@   @@
                        @@       @@ @@           @@@@@
                        @@       @@ @@           @@
                        @@       @@ @@           @@
                         @@     @@   @@      @   @@
                          @@   @@     @@    @@   @@
                            @@@         @@@@   @@@@@@

                           Placer for Standards Cells

                   Alliance CAD System 5.0 20040928,   ocp 5.0
                   Copyright (c) 2001-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

 o Special Net detected : vss
 o Special Net detected : vdd
 o No More Mouvement Possible .....

Ocp : placement finished
```

This will give us the placement shown in Figure 1.19. We can also force a vertical placement of the design

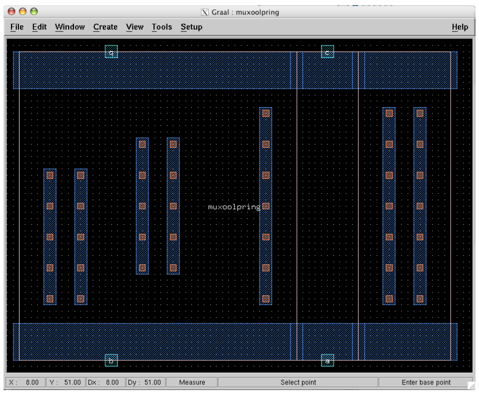


Figure 1.19: Pad oriented placement of the multiplexer circuit using OCP.

indicating the number of rows we want to use, but we must carefully choose the number of rows to avoid the use of an excesive number of tie cells (blank cells).

1.22 Proof

Proof is an Alliance tool intended to compare an extracted behavioral description from a layout to the corresponding behavioral description of the design. To extract a behavioral description from a layout we need one tool called yagle that is no longer available in Alliance. It is now available from Avertec (http://www.avertec.com/). They provide the binaries of this tool for non-commercial and/or university research. Proof is used with the following command.

```
% proof [-a] [-d]  file1  file2
```

The options available in Proof are shown in Table 1.22. As an example of the use of Proof we will compare the

Table 1.22: Options Available for the proof Tool.

Option	Description
-a	This option asks proof to keep the common auxiliary signals. Proof keeps all intermediate signals that have the same name in both descriptions (A common signal is considered as an input and an output of each description). This option can be useful for descriptions containing large equations. It may be used when proof has failed or if you want to debug in step by step mode the two different descriptions.
-d	The program displays errors when the behavioral descriptions are different. Equations are displayed when it's possible.

original behavioral description of our multiplexer design with that obtained with BOOM.

```
% proof -a -d mux mux_o

                @@@@@@@                                     @@@
                  @@   @@                                  @  @@
                  @@    @@                                @@  @@
                  @@    @@ @@@ @@@     @@@       @@@      @@
                  @@   @@   @@@  @@  @@   @@   @@   @@  @@@@@@@@@
                  @@@@@     @@   @@ @@     @@ @@     @@   @@
                  @@        @@      @@     @@ @@     @@   @@
                  @@        @@      @@     @@ @@     @@   @@
                  @@        @@      @@     @@ @@     @@   @@
                  @@        @@       @@   @@   @@   @@    @@
                @@@@@@     @@@@        @@@       @@@    @@@@@@

                                  Formal Proof

                   Alliance CAD System 5.0 20040928, proof 5.0
                   Copyright (c) 1990-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

================================  Environment  ================================
MBK_WORK_LIB            = .
MBK_CATA_LIB            =.:/usr/local/alliance/cells/sxlib:/usr/local/alliance/cells/dp_sxlib:
/usr/local/alliance/cells/rflib:/usr/local/alliance/cells/romlib:/usr/local/alliance/cells/ramlib:
/usr/local/alliance/cells/padlib
=======================  Files, Options and Parameters  =======================
First VHDL file         = mux.vbe
Second VHDL file        = mux_o.vbe
The common auxiliary signals are kept
Errors are displayed
===============================================================================

Compiling ’mux’ ...
Compiling ’mux_o’ ...
Looking for the common auxiliary signals :


---> final number of nodes = 9(3)

Running Abl2Bdd on ‘mux_o‘
--------------------------------------------------------------------------------
             Formal proof with Ordered Binary Decision Diagrams between

             ’./mux’  and  ’./mux_o’
--------------------------------------------------------------------------------
==============================  PRIMARY OUTPUT  ===============================
=============================  AUXILIARY SIGNAL  ==============================
==============================  REGISTER SIGNAL  ==============================
===============================  EXTERNAL BUS =================================
================================ INTERNAL BUS =================================

                            Formal Proof : OK

pppppppppppppppppppppppprrrrrrrrrrrooooooooooooooooooooooooooooooofffffffffffffff
--------------------------------------------------------------------------------
```

As could be seen from this output proof check the equivalence of primary outputs, signals and buses in that order. Internally proof uses a ROBDD (Reduced Ordered Binary Decision Diagram) to prove the equivalence of two behavioral descriptions.

1.23 Ring

Ring is the pad ring router of Alliance. It is used with the following command (it has no options).

```
% ring source result [stat]
```

We will apply ring to our multiplexer design, but first we will edit a .rin file describing the orientation of its inputs and outputs. This file is as follows.

```
north(q,c)
south(b,a)
east(p_vssb)
west(p_vddb)
width(vss 50 vdd 80)
```

The above file is not enough to use ring. We must also provide a structural file of our design that includes the pads.

1.24 S2R

S2R (Symbolic to Layout) is the tool in Alliance that let us translate our symbolic design to a real one. For this, S2R uses the technology file defined by the RDS_TECHNO_NAME environment variable. If we provide an appropiate technology file the output of S2R will be a layout file that can be given to a foundry for the fabrication of the design. During the design with Alliance we can also place and route the pads of the core using Ring. However, real pads are specific to a foundry and must be designed to fit to those of Alliance. One thing that must be taken into account is that the pads in Alliance have their abutment box at the lower left coordinate. We must also check that the foundry cells have an abutment box size multiple of the symbolic grid step used in Alliance. We use S2R with the following command.

```
s2r [-tc1rv] source [result]
```

The options available in S2R are shown in Table 1.23. We determine the type of the output file of S2R setting

Table 1.23: Options Available for the S2R Tool.

Option	Description
-t	Suppress the denotching phase. s2r performs gap filling, denotching in order to avoid DRC errors. This operation is time consuming. It is mandatory for the foundry but not really useful for capacitance evaluation.
-c	Deletes connectors and node names at all hierarchy level. Theses objects link the physical view and the logical view of a chip. If simulation is to be done after physical mapping, connectors must appear, so that the extractors and simulators can use them. This must not be used when preparing a final layout for the foundry. For the factory, the top level connectors are forbidden, since they do not represent any physical reality.
-1	Creates the top level cell with the instances as black boxes. This may be useful for hierarchical extractors, since the file size may be greatly reduced.
-r	Does not replace black boxes. Cells flagged with the G attribute in the catal(5) file will not be replaced by their equivalent layout loaded from disk.
-v	Verbose mode on.

the appropiate file type through the RDS_OUT environment variable. The default type is cif, but we can also set it to gds. The catalog file name used in pad substitution is set by the MBK_CATAL_NAME variable. The variable RDS_IN is used to define the format of the cells used in the substitution process. The format of the symbolic layout is determined by the MBK_IN_PH variable.

1.25 SCAPIN

The SCAPIN (SCAn Path INsertion) tool is used in Alliance to insert a scan path in the netlist of a design.

```
scapin [-VRB] [-P file] infile pathfile outfile
```

The inserted scan path will contain all the registers specified in the file specified by the pathfile (.path) file. SCAPIN will also add three new connectors in the netlist: scan_in, scan_out, and scan_test. In some cases also an output buffer is added to the scan_out connector. The options available in SCAPIN are shown in Table 1.24. The input and output file formats are determined by the MBK_IN_LO and MBK_OUT_LO variables, respectively. We must also check the SCAN_PARAM_NAME that determines the location of the .scapin parameter file (It contains the properties (ports names, models names, etc.) of the cells needed in the scan path insertion).

Table 1.24: Options Available for the SCAPIN Tool.

Option	Description
-V	Verbose mode on.
-R	All registers of the scan path are replaced by an equivalent cannable register cell (called reg-mux). (With the default option a simple multiplexor is added just before all registers of the scan path).
-B	Adds an output buffer before the output connector scan_out.
-P file	Specifies a parameter file (with extention .scapin) containing the properties of all cells needed for the scan path insertion.

1.26 SYF

The SYF (SYnthesizer of Finite state machines) tool of Alliance is used to generate VHDL data flow descriptions from a VHDL FSM description. This description can use an internal stack. We can generate with this tool Mealy and Moore style FSM with output regsiters if desired. In the Moore style we can generate a timing-optimized implementation that emulates a ROM with a microsequencer. We can also implement the scan-path of the corresponding registers.

```
syf -a|j|m|u|o|r [-CDEOPRSTV] input [output]
```

The options available in SYF are shown in Table 1.25. Let's use SYF on the mipsR3000 example provided in

Table 1.25: Options Available for the SYF Tool.

Option	Description
-a	Uses "Asp" as encoding algorithm.
-j	Uses "Jedi" as encoding algorithm.
-m	Uses "Mustang" as encoding algorithm.
-u	Uses an encoding given by user through input.enc file. In this file, a line started by a # character is a comment. A valid line contains one state name followed by its hexadecimal code.
-o	Uses the one hot encoding algorithm.
-r	Uses distinct random numbers for state encoding.
-C	Checks the transition's consistency.
-D	With this option syf doesn't optimize unused, i.e Don't Care, codes.
-E	Saves the encoding result in the output.enc. This file has the same syntax as input.enc file which is used by -u option.
-O	With this option syf places registers on the outputs.
-P	Implements a scan-path for the state registers, stack registers and possibly output registers. Scan-path mechanism is directely included in states decoder. Users should use scapin for a correct insertion of a scan-path in a netlist.
-I	Doesn't inverse outputs polarity.
-R	This option is only available for MOORE FSM. With this option, syf emulates a ROM with micro-sequencer implementation : there is no combinatorial logic between the state registers and the FSM outputs. This can be mandatory for external timing constraints.
-S	With this option syf doesn't take into account the cost of the transitions to compute an encoding.
-V	Verbose mode on.
-F	Sets synopsis output format.

Alliance. We run SYF on the mips_seq.fsm file already created in that tutorial. We will run SYF as follows.

```
% syf -o -V mips_seq mips_seqs

                            @@@@ @  @@@@   @@@@ @@@@@@@@@
                           @    @@   @@     @     @@    @
                          @@     @    @@   @      @@     @
                          @@@          @@ @       @@
                           @@@@         @@        @@   @
                             @@@@       @@        @@@@@@
                               @@@      @@        @@   @
                          @      @@     @@        @@
                          @@     @@     @@        @@
                          @@@    @      @@        @@
                          @  @@@@     @@@@@@    @@@@@@

                                 FSM Synthesizer

                  Alliance CAD System 5.0 20040928,     syf 5.0
                  Copyright (c) 1995-2005,       ASIM/LIP6/UPMC
                  Author(s): Ludovic Jacomme and Chaker Sarwary
                  E-mail        :   alliance-users@asim.lip6.fr

                --> Compile FSM file mips_seq
                --> Check FSM figure outputs
                --> Simplify FSM figure
                --> Verify FSM figure
                --> Identify reset conditions

                --> Name    : mips_seq
                --> States  : 99
                --> Inputs  : 32
                --> Outputs : 47
                --> Edges   : 353
                --> Stacks  : 0

                --> Encode FSM figure
                --> One hot encoding
                --> Translate FSM to BEH
                --> Simplify BEH figure

                --> Literals : 5019

                --> Save BEH file mips_seqs
```

This generate using a one-hot encoding the file mips_seqs.vbe that can be used to synthesize the corresponding design in Alliance.

1.27 VASY

VASY (VHDL Analyzer for sYnthesis) is a tool that converts a VHDL description to a synthesizable equivalent description usable by Alliance or other synthesis tools. VASY could handle a VHDL subset larger than the one supported in Alliance. VASY can generate an output in VHDL or verilog format. We use VASY with the following command.

```
vasy [-VpavsoipSHL] [-C num] [-E num] [-I format]  [-P  file]  filename  [outname]
```

The options available in VASY are shown in Table 1.26.

1.28 X2Y

X2Y is a netlist format converter. It is a tool used with the following command.

```
% x2y informat outformat infile outfile
```

There are no options in the X2Yy tool. The formats handled in X2Y are shown in Table 1.27.

Table 1.26: Options Available for the VASY Tool.

Option	Description
-D	Sets Debug mode on. As option [Level] and [File] could be given.
-V	Verbose mode on. Each step of the analysis is displayed onn the standard output.
-I	Specifies the VHDL input format such as Alliance VHDL format vbe, vst or industrial VHDL format vhd or vhdl.
-H	In a structural description, all model of instances are recursively analyzed (By default VASY analyzes only models with generic parameters). The leaves cells are defined by a file called CATAL.
-P	Specifies a 'file.pkg' containing a list of logical and physical package name.
-v	Drives an equivalent description in Verilog format.
-a	Drives an equivalent description in Alliance VHDL format vbe and/or vst. We can note that with this option, all arithmetic operators are expanded in an equivalent set of boolean expressions, because these operators don't belong to the Alliance VHDL subset.
-s	Drives an equivalent VHDL description (with the extension .vhd) accepted by most of industrial synthesis tools.
-r	
-o	Authorizes to overwrite existing files.
-i	Drives initial signal values (taken into account only with option -s).
-p	Adds power supply connectors (vdd and vss). Usefull option to enter in Alliance.
-S	Uses Std_logic instead of Bit (taken into account only with option -s).
-C	When the size of the adder is greater or equal to num a Carry Look Ahead adder is generated, instead of a Ripple Carry adder (taken into account only with option -a).
-E	Comparators are expanded in an equivalent set of boolean expressions, when their size is greater than num (taken into account only with option -a).
-L	A file .lax (see lax(5) for details) is generated. This file contains the list of all signals that must be kept during the synthesis step, using boom. (taken into account only with option -a).
-B	Drives a .boom file (with -a only).

Table 1.27: Formats handled by the X2Y Tool.

Option	Description
al	ALLIANCE netlist
ap	ALLIANCE layout
cct	HILO netlist
cp	VTI layout
edi	EDIF netlist or layout
fne	VTI extracted netlist
hns	VTI netlist
spi, sp, cir	SPICE netlist
vlg	VERILOG netlist
vst	VHDL netlist

1.29 XFSM

This is a X-window graphical tool to visualize FSMs (Finite State Machines). It has no manual pages and no options.

1.30 XPAT

The XPAT (X window PAT file visualizer) tool is just a visualizer that let's see the .pat files used by ASIMUT or generated by it. We use it with the following command.

```
% xpat [-l file_name] [-xor] [-install] [-force]
```

The options available in XPAT are shown in Table 1.28.

Table 1.28: Options Available for the XPAT Tool.

Option	Description
-l filename	Load the filename (with or without extension)
-xor	Two graphic cursor methods can be used, invert (default) or xor.
-install	Switch to a private color map.
-force	This option force all graphical objects to be displayed.

1.31 XSCH

This is a X-window graphical tool to visualize schematics. All its functionalities are accessed through its menus.

```
%   xsch  [-l  file_name]  [-xor]  [-install]  [-force]  [-I  input_format]   [-slide file_name ...]
```

The options that can be used with XSCH are shown in Table 1.29.

Table 1.29: Options Available for the XSCH Tool.

Option	Description
-l file_name	Load the file file_name (with or without extension)
-xor	Two graphic cursor methods can be used, invert or xor.
-force	This option force all graphical objects to be displayed.
-install	Switch to a private color map.
-I input_format	Specifies the input format (vst(5), al(5), vbe(5))
-slide file_name	Enables slide mode, all specifies files will be displayed one by one using keys +/- in the graphic window.

1.32 XVPN

This is a X-window graphical tool to visualize Petri nets. It has no manual pages neither a help option.

Chapter 2

Combinational Design Examples

In chapter 1 we have briefly seen how to use most of the tools in Alliance. In this chapter we will explain the design of combinational circuits. We will provide some examples and explain them. We will start with a bottom-up built circuit. This style could be useful when designing more complex circuits.

2.1 The design of a 4-bit multiplier using adder trees

Here we will explain the design of a basic 4-bit multiplier circuit. A multiplier can be built using the three blocks shown in Figure 2.1. The first block producing the partial products can be implemented with very simple

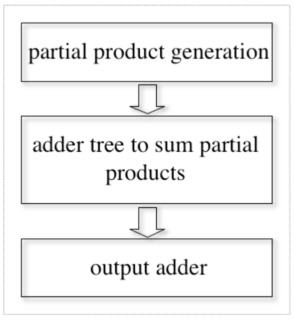

partial product generation
adder tree to sum partial products
output adder

Figure 2.1: The basic blocks to built.

circuitry as the one shown in Figure 2.2. For the adder trees we can use many schemes. In Figure 2.3 we shown

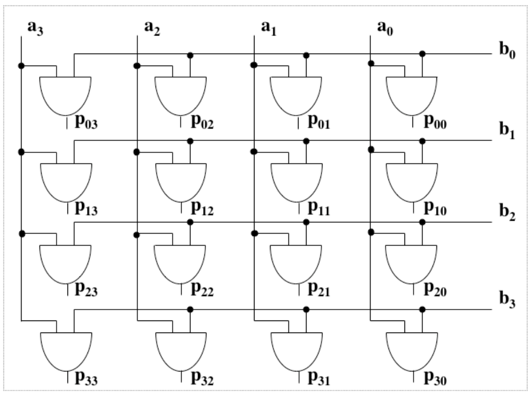


Figure 2.2: 4-bit partial product generator.

three schemes. The first one (most left) is the one of Wallace[1]. The one in the middle is due to Dadda[2], and the one on the right is of reference[3]. We will design the multiplier of this section using these three schemes and compare the areas of the circuits obtained with them. Wallace is the method more frequently used in research and commercial designs. Dadda's tree is an optional method that produces smaller designs. The third one is another way of structuring the adder in the tree. With this method we reduce the number of half adders in the tree and the number of bits to the final adder. The rectangular blocks in the trees represent the half adder and

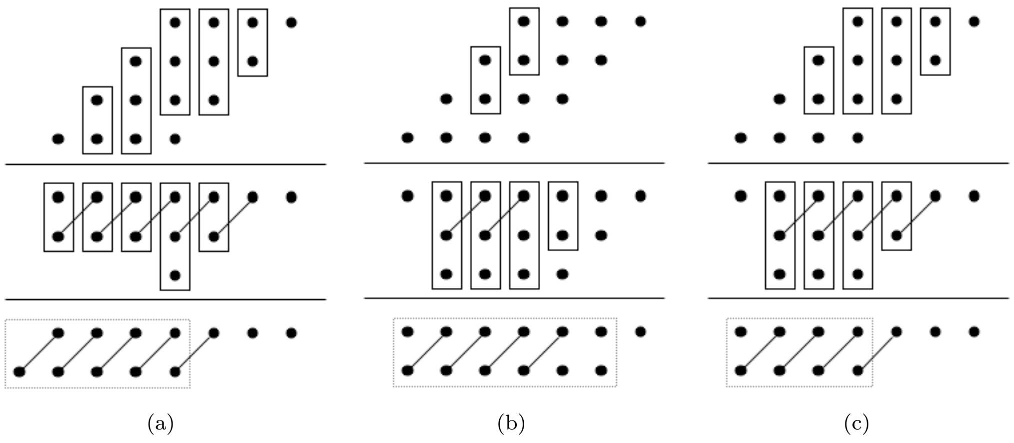


Figure 2.3: Three adder-tree schemes. (a) Wallace's (b) Dadda's (c) Palacios's

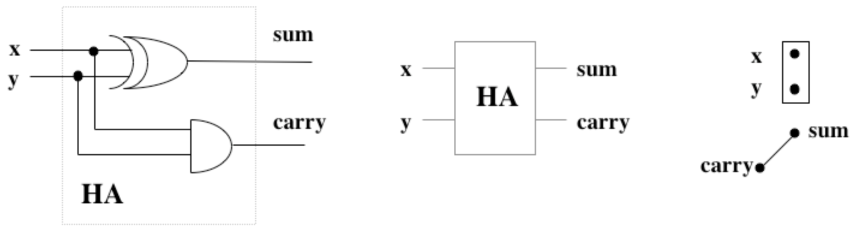


Figure 2.4: Half adder (HA) circuit details.

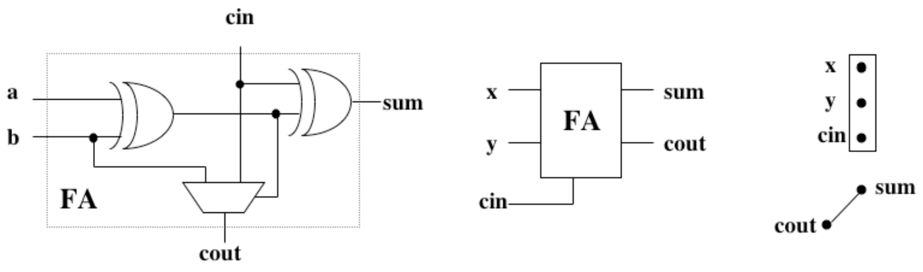


Figure 2.5: Full adder (FA) circuit details.

full adder circuits usen in them. Details of these circuits are given in Figure 2.4 and Figure 2.5, respectively. In this example we will implement the last stage of the multiplier with a simple ripple carry adder as the one shown in Figure 2.6. The hierarchical structure of our multiplier is the one shown in Figure 2.7.

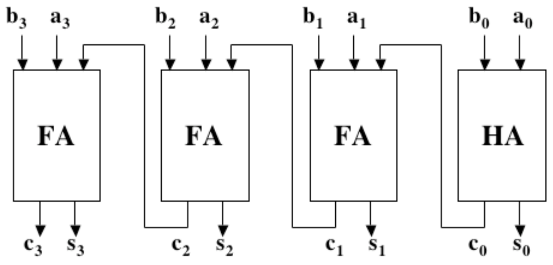


Figure 2.6: 4-bit ripple carry adder.

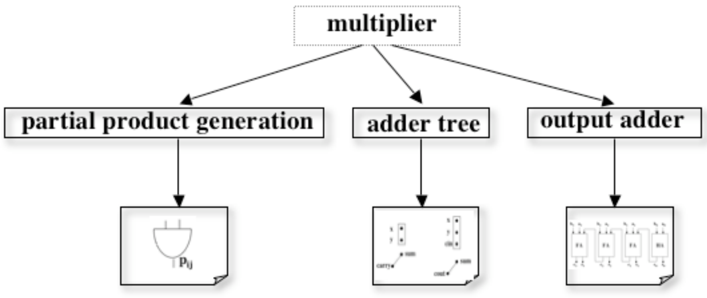


Figure 2.7: Design hierarchy of the multiplier.

2.1.1 Design of the partial product generator

We will design the partial product generator starting with the following behavioral description.

```
-- 4-bit partial product generator
entity myppg4 is
port(a0,a1,a2,a3,b0,b1,b2,b3,vdd,vss : in bit;
     p00,p01,p02,p03,p10,p11,p12,p13 : out bit;
     p20,p21,p22,p23,p30,p31,p32,p33 : out bit);
end ppg4;

--behavior
architecture vbe of myppg4 is
begin
p00 <= b0 AND a0;
p01 <= b0 AND a1;
p02 <= b0 AND a2;
p03 <= b0 AND a3;
p10 <= b1 AND a0;
p11 <= b1 AND a1;
p12 <= b1 AND a2;
p13 <= b1 AND a3;
p20 <= b2 AND a0;
p21 <= b2 AND a1;
p22 <= b2 AND a2;
p23 <= b2 AND a3;
p30 <= b3 AND a0;
p31 <= b3 AND a1;
p32 <= b3 AND a2;
p33 <= b3 AND a3;
end vbe;
```

We will optimize and synthesize it using BOOM, BOOG and LOON. We will use BOOM with the options to optimize speed.

```
% boom -l 3 -d 0 myppg4 ppg4

                   @@@@@@@                         @@@     @@@
                     @@   @@                        @@     @@
                     @@    @@                       @@@   @@@
                     @@    @@     @@@       @@@     @@@   @@@
                     @@   @@    @@   @@   @@   @@   @ @@ @ @@
                     @@@@@@    @@     @@ @@     @@  @ @@ @ @@
                     @@    @@  @@     @@ @@     @@  @ @@@  @@
                     @@     @@ @@     @@ @@     @@  @  @@  @@
                     @@     @@ @@     @@ @@     @@  @  @   @@
                     @@    @@   @@   @@   @@   @@   @      @@
                   @@@@@@@@       @@@       @@@    @@@    @@@@

                              BOOlean Minimization

                   Alliance CAD System 5.0 20040928,  boom 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr

        --> Parse BEH file myppg4.vbe
        --> Drive BEH file ppg4
```

If you check the output file obtained with BOOM you will notice that it does not improve the already given description. Next, we use BOOG also optimizing delay (the output has been shortened to make it fit).

```
% boog ppg4 myppg4 -x 0 -m 4
                  @@@@@@@                             @@@@ @
                    @@   @@                         @@    @@
                    @@    @@                       @@      @
                    @@    @@     @@@       @@@    @@
                    @@   @@    @@   @@   @@   @@  @@
                    @@@@@@    @@     @@ @@     @@ @@     @@@@@
                    @@    @@  @@     @@ @@     @@ @@     @ @@
                    @@     @@ @@     @@ @@     @@ @@    @  @@
                    @@     @@ @@     @@ @@     @@  @@      @@
                    @@    @@   @@   @@   @@   @@    @@    @@
                  @@@@@@@@       @@@       @@@        @@@@

                         Binding and Optimizing On Gates

             Alliance CAD System 5.0 20040928, boog 5.0 [2003/01/09]
             Copyright (c) 2000-2005,                ASIM/LIP6/UPMC
             Author(s):                             Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr
Reading default parameter...
100% delay optimization
Reading file 'ppg4.vbe'...
Controlling file 'ppg4.vbe'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Controlling lib '/usr/local/alliance/cells/sxlib'...
Preparing file 'ppg4.vbe'...
Capacitances on file 'ppg4.vbe'...
Unflattening file 'ppg4.vbe'...
Mapping file 'ppg4.vbe'...
Saving file 'myppg4.vst'...
Quick estimated critical path (no warranty)...324 ps from 'a3' to 'p33'
Quick estimated area (with over-cell routing)...20000 lambda?
Details...
        a2_x2: 16
        Total: 16
Saving critical path in xsch color file 'myppg4.xsc'...
End of boog...
```

We then apply LOON to the .vst file we have obtained (the output has also been shortened to make it fit).

```
% loon -x 0 -m 4 myppg4 ppg4
                   @@@@@@                          @@@    @@@
                     @@                             @@     @
                     @@                             @@@    @
                     @@           @@@       @@@     @ @@   @
                     @@         @@   @@   @@   @@   @ @@   @
                     @@        @@     @@ @@     @@  @  @@  @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@      @ @@     @@ @@     @@  @    @@@
                     @@     @   @@   @@   @@   @@   @     @@
                   @@@@@@@@@@     @@@       @@@    @@@    @@

                           Local optimization on Nets

             Alliance CAD System 5.0 20040928, loon 5.0 [2003/12/07]
             Copyright (c) 2000-2005,                ASIM/LIP6/UPMC
             Author(s):                             Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr
Reading default parameter...
100% delay optimization
Reading file 'myppg4.vst'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Capacitances on file 'myppg4.vst'...
Delays on file 'myppg4.vst'...324 ps
Area on file 'myppg4.vst'...20000 lamda? (with over-cell routing)
Details...
        a2_x2: 16 (100%)
        Total: 16
Critical path (no warranty)...324 ps from 'a3' to 'p33'
Saving file 'ppg4.vst'...
Saving critical path in xsch color file 'ppg4.xsc'...
End of loon...
```

With this we have a synthesized structural description of our original partial product generator. For the other two blocks we need as basic (leaf) block half and full adder circuits. The design and synthesis of them is shown in the following section.

2.1.2 Half and Full Adder design

In the design of the adder tree and output adder of the multiplier we will use the half adder and full adder of Figure 2.4 and Figure 2.5, respectively. The half adder is given by the following behavioral description (remember that we must provide both the behavioral and structural description of our leaf designs).

```
-- half adder
entity ha is
port(a,b,vdd,vss : in bit;
     sum,carry   : out bit);
end ha;

--behavior
architecture vbe of ha is
signal a_bar, b_bar : bit;
begin
a_bar <= NOT a;
b_bar <= NOT b;
sum <= (a_bar AND b) OR (a AND b_bar);
carry <= a AND b;
end vbe;
```

As could you notice this does not correspond to the circuit of Figure 2.4. We will obtain that description using BOOM optimizing for delay.

```
% boom -l 3 -d 0 myha ha
```

We obtain the following behavioral file that is indeed the flow description of the circuit of Figure 2.4.

```
-- VHDL data flow description generated from ‘ha‘
--              date : Mon Feb 14 11:05:26 2005


-- Entity Declaration

ENTITY ha IS
  PORT (
  a : in BIT;   -- a
  b : in BIT;   -- b
  vdd : in BIT; -- vdd
  vss : in BIT; -- vss
  sum : out BIT;        -- sum
  carry : out BIT       -- carry
  );
END ha;


-- Architecture Declaration

ARCHITECTURE behaviour_data_flow OF ha IS

BEGIN

carry <= (b AND a);

sum <= (b XOR a);
END;
```

We also run BOOG and LOON on this circuit as follows.

```
% boog ha myha -x 0 -m 4
```

```
% loon -x 0 -m 4 myha ha
```

In the case of the full adder we will use the following behavioral description that matches the circuit of Figure 2.5.

```
-- full adder
entity fa is
port(a,b,cin,vdd,vss : in bit;
     sum,cout        : out bit);
end fa;

--behavior
architecture vbe of fa is
signal xor1 : bit;

begin
xor1 <= a XOR b;
cout <= b when xor1 = '0' else cin;
sum <= cin XOR xor1;
end vbe;
```

As for the half adder we run BOOM, BOOG and LOON optimizing speed (delay).

```
% boom -l 3 -d 0 myfa fa
```

```
% boog fa myfa -x 0 -m 4
```

```
% loon -x 0 -m 4 myfa fa
```

You can check that the multiplexer of the original circuit has been changed for the oa2ao222_x2 cell during the synthesis process. We are now ready to design the adder trees and the output adder of the multiplier. We will show in the following section the design of the adder tree.

2.1.3 Adder Tree

As we have shown in Figure 2.3, there are many ways of implementing the adder tree used in a multiplier. We will explain here the design of the tree using the approach of Wallace[1]. The behavioral description of it is the following one.

```
 entity wallace4 is
 port(p01,p02,p03 : in bit;
      p10,p11,p12,p13 : in bit;
      p20,p21,p22,p23 : in bit;
      p30,p31,p32,p33 : in bit;
      vdd,vss : in bit;
      p1,p2,p3a,p3b,p4a,p4b,p5a,p5b,p6a,p6b,p7b : out bit);
end wallace4;

architecture structural of wallace4 is
Component ha
 port(a,b : in bit;
      vdd,vss : in bit;
      sum,carry : out bit);
end component;

component fa
 port(a,b,cin : in bit;
      vdd,vss : in bit;
      sum,cout : out bit);
end component;

signal h1c,f1s,f1c,f2s,f2c : bit;
signal f3s,f3c,h2s,h2c : bit;
```

```
begin
-- Wallace tree
-- first level

ha1 : ha
 port map(a => p01, b  => p10,
          vdd => vdd, vss => vss,
          sum => p1, carry => h1c);

fa1 : fa
 port map(a => p02, b => p11, cin => p20,
          vdd => vdd, vss => vss,
          sum => f1s, cout => f1c);

fa2 : fa
 port map(a => p03, b => p12, cin => p21,
          vdd => vdd, vss => vss,
          sum => f2s, cout => f2c);

fa3 : fa
 port map(a => p13, b => p22, cin => p31,
          vdd => vdd, vss => vss,
          sum => f3s, cout => f3c);

ha2 : ha
 port map(a => p23, b  => p32,
          vdd => vdd, vss => vss,
          sum => h2s, carry => h2c);

-- second level
ha3 : ha
 port map(a => f1s, b  => h1c,
          vdd => vdd, vss => vss,
          sum => p2, carry => p3b);

fa4 : fa
 port map(a => f2s, b => f1c, cin => p30,
          vdd => vdd, vss => vss,
          sum => p3a, cout => p4b);

ha4 : ha
 port map(a => f3s, b  => f2c,
          vdd => vdd, vss => vss,
          sum => p4a, carry => p5b);

ha5 : ha
 port map(a => h2s, b  => f3c,
          vdd => vdd, vss => vss,
          sum => p5a, carry => p6b);

ha6 : ha
 port map(a => p33, b  => h2c,
          vdd => vdd, vss => vss,
          sum => p6a, carry => p7b);

end structural;
```

Since we already synthesized its basic blocks we can describe the tree as above. For the synthesis process we have several choices. We will apply here FlatBeh to obtain a behavioral description of the adder tree.

```
% flatbeh mywallace mywallace
```

Then we will apply to it BOOM, BOOG and LOON (of course, we can also process it with only BOOG and LOON).

```
% boom -l 3 -d 0 mywallace wallace
```

```
% boog wallace swallace -x 0 -m 4
```

```
% loon -x 0 -m 4 swallace wallace
```

This completes the synthesis process for the adder tree. In the following section we design the last block required by the multiplier.

2.1.4 Output Adder

The design of the output adder is very simple. Since it it not a leaf design we just need to provide just it structural description.

```
entity add5 is
 port(a3,a4,a5,a6 : in bit;
      b3,b4,b5,b6,b7 : in bit;
      vdd,vss : in bit;
      s3,s4,s5,s6,s7,c7 : out bit);
end add5;

architecture structural of add5 is
Component ha
 port(a,b : in bit;
      vdd,vss : in bit;
      sum,carry : out bit);
end component;

component fa
 port(a,b,cin : in bit;
      vdd,vss : in bit;
      sum,cout : out bit);
end component;

signal c3,c4,c5,c6 : bit;

begin
-- carry ripple adder for the 4-bit multiplier
ha1 : ha
 port map(a => a3, b  => b3,
          vdd => vdd, vss => vss,
          sum => s3, carry => c3);

fa1 : fa
 port map(a => a4, b => b4, cin => c3,
          vdd => vdd, vss => vss,
          sum => s4, cout => c4);

fa2 : fa
 port map(a => a5, b => b5, cin => c4,
          vdd => vdd, vss => vss,
          sum => s5, cout => c5);

fa3 : fa
 port map(a => a6, b => b6, cin => c5,
          vdd => vdd, vss => vss,
          sum => s6, cout => c6);

ha2 : ha
 port map(a => c6, b  => b7,
          vdd => vdd, vss => vss,
          sum => s7, carry => c7);

end structural;
```

Here we also use FlatBeh, BOOM, BOOG and LOON to synthesize the design.

```
% flatbeh myadd5 myadd5
```

```
% boom -l 3 -d 0 myadd5 add5
```

```
% boog add5 sadd5 -x 0 -m 4
```

```
% loon -x 0 -m 4 sadd5 add5
```

We are now ready to proceed with the design of the multiplier. This is described in the following section.

2.1.5 4-bit Multiplier

Since we have all the components of our multiplier we can start with a structural description of it.

```
entity multiplier4 is
 port(a0,a1,a2,a3 : in bit;
      b0,b1,b2,b3 : in bit;
      vdd,vss : in bit;
      p0,p1,p2,p3,p4,p5,p6,p7,p8 : out bit);
end multiplier4;

architecture structural of multiplier4 is
Component ppg4
 port(a0,a1,a2,a3 : in bit;
      b0,b1,b2,b3 : in bit;
      vdd,vss : in bit;
      p00,p01,p02,p03 : out bit;
      p10,p11,p12,p13 : out bit;
      p20,p21,p22,p23 : out bit;
      p30,p31,p32,p33 : out bit);
end component;

Component wallace
 port(p01,p02,p03 : in bit;
      p10,p11,p12,p13 : in bit;
      p20,p21,p22,p23 : in bit;
      p30,p31,p32,p33 : in bit;
      vdd,vss : in bit;
      p1,p2,p3a,p3b,p4a,p4b,p5a,p5b,p6a,p6b,p7b : out bit);
end component;

Component add5
 port(a3,a4,a5,a6 : in bit;
      b3,b4,b5,b6,b7 : in bit;
      vdd,vss : in bit;
      s3,s4,s5,s6,s7,c7 : out bit);
end component;

signal p01,p02,p03 : bit;
signal p10,p11,p12,p13 : bit;
signal p20,p21,p22,p23 : bit;
signal p30,p31,p32,p33 : bit;
signal p3a,p3b,p4a,p4b,p5a,p5b,p6a,p6b,p7b : bit;

begin
-- partial product generation
ppg : ppg4
 port map(a0 => a0, a1 => a1, a2 => a2, a3 => a3,
          b0 => b0, b1 => b1, b2 => b2, b3 => b3,
          vdd => vdd, vss => vss,
          p00 => p0, p01 => p01, p02 => p02, p03 => p03,
          p10 => p10, p11 => p11, p12 => p12, p13 => p13,
          p20 => p20, p21 => p21, p22 => p22, p23 => p23,
          p30 => p30, p31 => p31, p32 => p32, p33 => p33);

-- Wallace tree
w4 : wallace
 port map(p01 => p01, p02 => p02, p03 => p03,
      p10 => p10, p11 => p11, p12 => p12, p13 =>p13,
      p20 => p20, p21  =>p21, p22 => p22, p23 =>p23,
      p30 => p30, p31 => p31, p32 => p32, p33 =>p33,
      vdd => vdd, vss => vss,
      p1 => p1 , p2 => p2, p3a => p3a, p3b => p3b, p4a => p4a,
      p4b => p4b, p5a => p5a, p5b => p5b, p6a => p6a, p6b => p6b,
      p7b => p7b);

-- output adder

add : add5
 port map(a3 => p3a, a4 => p4a, a5 => p5a, a6 => p6a,
          b3 => p3b, b4 => p4b, b5 => p5b, b6 => p6b, b7 => p7b,
          vdd => vdd, vss => vss,
          s3 => p3, s4 => p4, s5 => p5, s6 => p6, s7 => p7, c7 => p8);

end structural;
```

As we have already donw before, we use first FlatBeh to obtain a behavioral representation of our multiplier.

```
% flatbeh mymultiplier4 mymultiplier4
      @@@@@@@@@  @@@@                      @@@@@@@               @@@
        @@    @    @@                @       @@   @@              @@
        @@     @   @@               @@       @@    @@             @@
        @@         @@     @@@@      @@       @@    @@    @@@@@    @@ @@@
        @@   @     @@    @@   @   @@@@@@@@   @@   @@    @     @   @@@  @@
        @@@@@@     @@    @@   @@    @@       @@@@@@    @@     @@  @@    @@
        @@   @     @@      @@@@@    @@       @@    @@  @@@@@@@@@  @@    @@
        @@         @@    @@   @@    @@       @@     @@ @@         @@    @@
        @@         @@   @@    @@    @@       @@     @@ @@      @  @@    @@
        @@         @@   @@   @@@    @@   @   @@    @@   @@    @@  @@    @@
      @@@@@@     @@@@@@  @@@@  @@    @@@@  @@@@@@@@       @@@@   @@@@  @@@@

                              a netlist abstractor
           Alliance CAD System 5.0 20040928, flatbeh 5.0 [2000/11/01]
           Copyright (c) 1993-2005,                     ASIM/LIP6/UPMC
           Author(s):                Fran?ois DONNET, Huu Nghia VUONG
           E-mail        :                alliance-users@asim.lip6.fr
     =========================  Environnement  =========================
     MBK_WORK_LIB        = .
     MBK_CATA_LIB        = .:/usr/local/alliance/cells/sxlib:/usr/local/alliance/cells/dp_sxlib:
     /usr/local/alliance/cells/rflib:/usr/local/alliance/cells/romlib:
     /usr/local/alliance/cells/ramlib:/usr/local/alliance/cells/padlib
     MBK_CATAL_NAME      = CATAL
     ============================== Files ==============================
     Netlist file        = mymultiplier4.vst
     Output  file        = mymultiplier4.vbe
     ===================================================================
Loading './mymultiplier4.vst'
flattening figure mymultiplier4
 loading buf_x2
 loading oa2a22_x2
 loading oa2ao222_x2
 loading o3_x2
 loading inv_x2
 loading noa2ao222_x1
 loading o2_x2
 loading mx2_x2
 loading nao22_x1
 loading xr2_x1
 loading noa22_x1
 loading nxr2_x1
 loading no2_x1
 loading no3_x1
 loading na2_x1
 loading an12_x1
 loading a2_x2
Restoring array's orders
BEH : Saving 'mymultiplier4' in a vhdl file (vbe)
```

With the behavioral description we can then optimize it first with BOOM.

```
% boom -l 3 -d 0 mymultiplier4 multiplier4
                   @@@@@@@                         @@@     @@@
                     @@   @@                        @@     @@
                     @@    @@                       @@@   @@@
                     @@    @@     @@@       @@@     @@@   @@@
                     @@   @@    @@   @@   @@   @@   @ @@ @ @@
                     @@@@@@    @@     @@ @@     @@  @ @@ @ @@
                     @@    @@  @@     @@ @@     @@  @ @@@  @@
                     @@     @@ @@     @@ @@     @@  @  @@  @@
                     @@     @@ @@     @@ @@     @@  @  @   @@
                     @@    @@   @@   @@   @@   @@   @      @@
                   @@@@@@@@       @@@       @@@    @@@    @@@@

                              BOOlean Minimization
                   Alliance CAD System 5.0 20040928,  boom 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr
        --> Parse BEH file mymultiplier4.vbe
        --> Drive BEH file multiplier4
```

Then we use BOOG to map the description on the cell library provided by Alliance.

```
% boog multiplier4 smultiplier4 -x 0 -m 4

                  @@@@@@@                             @@@@ @
                    @@   @@                         @@    @@
                    @@    @@                       @@      @
                    @@    @@     @@@       @@@    @@
                    @@   @@    @@   @@   @@   @@  @@
                    @@@@@@    @@     @@ @@     @@ @@     @@@@@
                    @@    @@  @@     @@ @@     @@ @@     @ @@
                    @@     @@ @@     @@ @@     @@ @@    @  @@
                    @@     @@ @@     @@ @@     @@  @@      @@
                    @@    @@   @@   @@   @@   @@    @@    @@
                  @@@@@@@@       @@@       @@@        @@@@

                         Binding and Optimizing On Gates

             Alliance CAD System 5.0 20040928, boog 5.0 [2003/01/09]
             Copyright (c) 2000-2005,                 ASIM/LIP6/UPMC
             Author(s):                              Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr

        MBK_VDD        : vdd
        MBK_VSS        : vss
        MBK_IN_LO      : vst
        MBK_OUT_LO     : vst
        MBK_WORK_LIB   : .
        MBK_TARGET_LIB : /usr/local/alliance/cells/sxlib

Reading default parameter...
100% delay optimization
Reading file 'multiplier4.vbe'...
Controlling file 'multiplier4.vbe'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Mapping Warning: Cell 'halfadder_x4' isn't supported
Mapping Warning: Cell 'halfadder_x2' isn't supported
Mapping Warning: Cell 'fulladder_x4' isn't supported
Mapping Warning: Cell 'fulladder_x2' isn't supported
Controlling lib '/usr/local/alliance/cells/sxlib'...
Preparing file 'multiplier4.vbe'...
Capacitances on file 'multiplier4.vbe'...
Unflattening file 'multiplier4.vbe'...
Mapping file 'multiplier4.vbe'...
Saving file 'smultiplier4.vst'...
Quick estimated critical path (no warranty)...2990 ps from 'b2' to 'p4'
Quick estimated area (with over-cell routing)...291000 lambda?
Details...
        na2_x1: 39
        no2_x1: 32
        a2_x2: 31
        inv_x2: 12
        oa22_x2: 11
        na3_x1: 11
        o2_x2: 11
        noa22_x1: 10
        nao22_x1: 7
        on12_x1: 6
        xr2_x1: 6
        a3_x2: 5
        ao22_x2: 5
        nxr2_x1: 5
        no3_x1: 4
        mx3_x2: 4
        oa2ao222_x2: 4
        nao2o22_x1: 4
        o3_x2: 3
        na4_x1: 2
        noa2ao222_x1: 2
        zero_x0: 1
        ao2o22_x2: 1
        a4_x2: 1
        mx2_x2: 1
        no4_x1: 1
        buf_x2: 1
        Total: 220
Saving critical path in xsch color file 'smultiplier4.xsc'...
End of boog...
```

Finally with LOON we reduce delays (which will insert buffers where appropiate).

```
% loon -x 0 -m 4 smultiplier4 multiplier4

                   @@@@@@                          @@@    @@@
                     @@                             @@     @
                     @@                             @@@    @
                     @@           @@@       @@@     @ @@   @
                     @@         @@   @@   @@   @@   @ @@   @
                     @@        @@     @@ @@     @@  @  @@  @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@      @ @@     @@ @@     @@  @    @@@
                     @@     @   @@   @@   @@   @@   @     @@
                   @@@@@@@@@@     @@@       @@@    @@@    @@

                           Local optimization on Nets

             Alliance CAD System 5.0 20040928, loon 5.0 [2003/12/07]
             Copyright (c) 2000-2005,                ASIM/LIP6/UPMC
             Author(s):                             Fran?ois Donnet
             E-mail        :            alliance-users@asim.lip6.fr

        MBK_IN_LO       : vst
        MBK_OUT_LO      : vst
        MBK_TARGET_LIB  : /usr/local/alliance/cells/sxlib

Reading default parameter...
100% delay optimization
Reading file 'smultiplier4.vst'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Capacitances on file 'smultiplier4.vst'...
Delays on file 'smultiplier4.vst'...3304 ps
Area on file 'smultiplier4.vst'...291000 lamda? (with over-cell routing)
Worst RC on file 'smultiplier4.vst'...634 ps
Inserting buffers on critical path for file 'multiplier4.vst'...1 buffer inserted -> 3263 ps
Improving RC on critical path for file 'multiplier4.vst'...3246 ps
Improving all RC for file 'multiplier4.vst'...
Worst RC on file 'multiplier4.vst'...373 ps
Area on file 'multiplier4.vst'...293000 lamda? (with over-cell routing)
Details...
        na2_x1: 39 (13%)
        no2_x1: 32 (10%)
        a2_x2: 31 (13%)
        oa22_x2: 11 (5%)
        na3_x1: 11 (4%)
        o2_x2: 11 (4%)
        inv_x2: 11 (2%)
        noa22_x1: 10 (5%)
        nao22_x1: 7 (3%)
        on12_x1: 6 (2%)
        xr2_x1: 6 (4%)
        a3_x2: 5 (2%)
        ao22_x2: 5 (2%)
        nxr2_x1: 5 (3%)
        no3_x1: 4 (1%)
        mx3_x2: 4 (4%)
        oa2ao222_x2: 4 (3%)
        nao2o22_x1: 4 (2%)
        o3_x2: 3 (1%)
        buf_x2: 2 (0%)
        na4_x1: 2 (1%)
        noa2ao222_x1: 2 (1%)
        zero_x0: 1 (0%)
        ao2o22_x2: 1 (0%)
        a4_x2: 1 (0%)
        mx2_x2: 1 (0%)
        no4_x1: 1 (0%)
        inv_x8: 1 (0%)
        Total: 221
Critical path (no warranty)...3246 ps from 'b2' to 'p4'
Saving file 'multiplier4.vst'...
Saving critical path in xsch color file 'multiplier4.xsc'...
End of loon...
```

We are now ready to take our design into the layout phase in the design process. We use first OCP to place it.

```
% ocp -ring multiplier4 mymultiplier4

                            @@@         @@@@ @ @@@@@@@
                          @@   @@     @@    @@   @@   @@
                         @@     @@   @@      @   @@    @@
                        @@       @@ @@       @   @@    @@
                        @@       @@ @@           @@   @@
                        @@       @@ @@           @@@@@
                        @@       @@ @@           @@
                        @@       @@ @@           @@
                         @@     @@   @@      @   @@
                          @@   @@     @@    @@   @@
                            @@@         @@@@   @@@@@@

                           Placer for Standards Cells

                   Alliance CAD System 5.0 20040928,   ocp 5.0
                   Copyright (c) 2001-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

 o Special Net detected : vdd
 o Special Net detected : vss
...............................................................
Ocp : placement finished
```

With the circuit in place now we route it with NERO.

```
% nero -V -2 -p mymultiplier4 multiplier4 multiplier4
```

At this point it is useful to check the design flow we have followed. This is displayed in Figure 2.8. As could

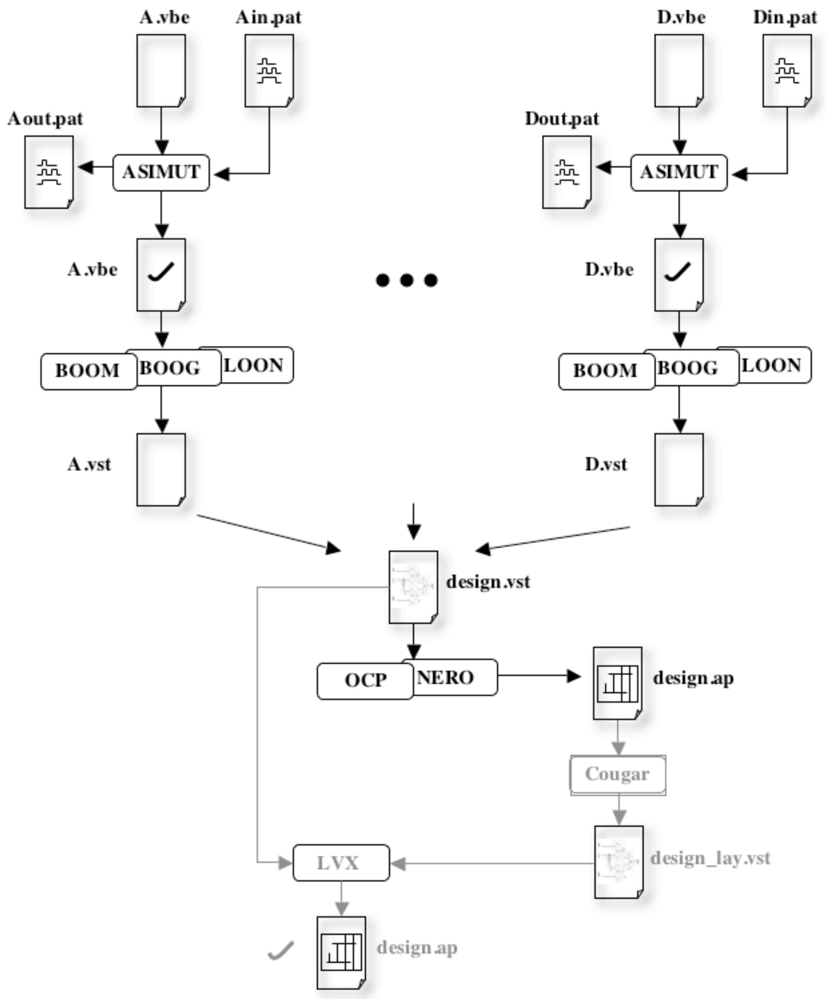


Figure 2.8: Design flow in a hierarchical design up to the placement step.

be noticed from this figure we could extract with Cougar a structural description from the routed design file, and then compare it with the synthesized one by LOON. Let's do it.

```
% cougar -f -v multiplier4 multiplier4_lay
```

This will give us a structural description of our design. We use LVX to compare it with the one obatained with LOON.

```
% lvx vst vst multiplier4 multiplier4_lay
                       @@@@@@      @@@@    @@@ @@@@   @@@@
                         @@         @@      @   @@     @
                         @@         @@      @    @@   @
                         @@          @@    @      @@ @
                         @@          @@    @       @@
                         @@           @@  @        @@
                         @@           @@  @        @@@
                         @@            @@@        @  @@
                         @@      @     @@@       @    @@
                         @@     @       @       @      @@
                       @@@@@@@@@@       @      @@@    @@@@

                             Gate Netlist Comparator

                   Alliance CAD System 5.0 20040928,   lvx 1.4
                   Copyright (c) 1992-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

***** Loading multiplier4 (vst)...
Warning 2 : consistency checks will be disabled
***** Loading multiplier4_lay (vst)...
***** Compare Terminals .............
***** O.K.      (0 sec)
***** Compare Instances ............
***** O.K.      (0 sec)
***** Compare Connections ...........................
***** O.K.      (0 sec)
===== Terminals .......... 19
===== Instances .......... 221
===== Connectors ......... 1213
***** Netlists are Identical. *****     (0 sec)
```

That tell us that the two netlists are identical. Let's check now the design rules of the layout using DRUC.

```
% druc multiplier4
                  @@@@@@@     @@@@@@@                    @@@@ @
                    @@   @@     @@   @@                @@    @@
                    @@    @@    @@    @@              @@      @
                    @@     @@   @@    @@  @@@  @@@@  @@       @
                    @@     @@   @@   @@    @@    @@  @@
                    @@     @@   @@@@@      @@    @@  @@
                    @@     @@   @@  @@     @@    @@  @@
                    @@     @@   @@   @@    @@    @@  @@
                    @@    @@    @@   @@    @@    @@   @@      @
                    @@   @@     @@    @@   @@   @@@    @@    @@
                  @@@@@@@     @@@@@   @@@   @@@@  @@     @@@@

                               Design Rule Checker

                   Alliance CAD System 5.0 20040928,  druc 5.0
                   Copyright (c) 1993-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

Flatten DRC on: multiplier4
Delete MBK figure : multiplier4
Load Flatten Rules : /usr/local/alliance/etc/cmos.rds
Unify : multiplier4
Create Ring : multiplier4_rng
Merge Errorfiles:
Merge Error Instances:
instructionCourante :  56
End DRC on: multiplier4
Saving the Error file figure
Done
  0
File: multiplier4.drc is empty: no errors detected.
```

We are now ready to convert our symbolic design to a real one (using the technology provided by Alliance). For this, we use S2R.

```
% s2r multiplier4

                                      @@@@
                                     @   @@
                                    @@    @@
                            @@@@@@  @@@   @@ @@@ @@@
                           @@    @   @   @@   @@@  @@
                           @@@           @    @@   @@
                            @@@@        @     @@
                              @@@@     @      @@
                           @    @@@   @    @  @@
                           @@    @@  @@@@@@   @@
                           @ @@@@@  @@@@@@@  @@@@

                        Symbolic to Real layout converter

                   Alliance CAD System 5.0 20040928,   s2r 5.0
                   Copyright (c) 2002-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr


        o loading technology file : /usr/local/alliance/etc/cmos.rds
        o loading all level of symbolic layout : multiplier4
        o removing symbolic data structure
        o layout post-treating
                with top connectors,
                with sub connectors,
                with signal names,
                without scotch.
        o saving multiplier4.cif
```

We can visualize the layout obtained using DREAL.

```
% dreal
```

The layout obtained with S2R is shown in Figure 2.9. The other two trees shown in Figure 2.3 can be

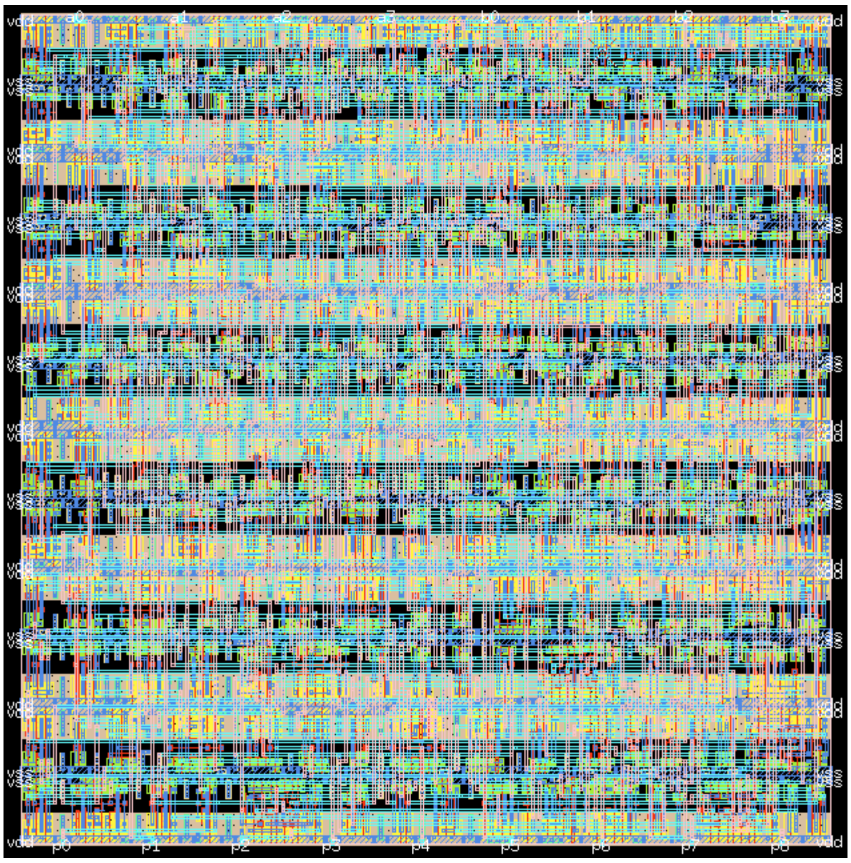


Figure 2.9: Layout shown by DREAL.

implemented using the leaf designs we provide with the sources of the book.

2.2 The use of Makefiles to automate the design flow

Here we will use the 4-bit multiplier design of the alliance-example directory to explain in some detail the use of Makefiles to automate the design process and make it easier modifying files in your designs. The example we will show here already has its own make file. We will explain here how to build your own one. At the end you will have a Makefile almost identical to the one provided with the example. Since we worked all the text in a tcsh shell environment, we will follow an approach similar to that used in the tutorials of Alliance. We will set environment variables in the command line and leave the running of tools to the Makefile. First, we will obtain a behavioral (Alliance) description from the VHDL file provided in the example. For this we use VASY. We will set the environment variables as indicated in the original Makefile.

```
setenv MBK_WORK_LIB .; setenv MBK_CATAL_NAME NO_CATAL
```

And, we will edit the following Makefile.

```
ALLIANCE_BIN=$(ALLIANCE_TOP)/bin

VASY   = $(ALLIANCE_BIN)/vasy

multi4.vbe : multi4.vhdl
             $(VASY)  -a -B -o -p -I vhdl multi4
```

This is a simplified version of the original Makefile, having only what is needed to run VASY. Let's explain the above Makefile. At the top we used what is called macros. Macros are similar to variables. They let us store name of files, directory paths, etc. In our make file we use one macro to define the path to the bin directory in Alliance. The second macro uses the first one to point to the VASY tool. In the second macro we expand the first one by typing $\$(macro)$

The second half of our make file defines a dependency. The $multi4.vbe4$ is the target and the argument after the : is the source file needed to get the target. In other words, it has the following format $target : sourcefile(s)$. In the lines following this dependency definition we define the commands to generate the target using the source file(s). In this case we need just one command to run VASY on the source file with the corresponding options to obtain the behavioral Alliance description. Remember that the command lines must be preceded by a tab. With this make file we can now run VASY as follows.

```
% make multi4.vbe
/usr/local/alliance/bin/vasy -a -B -o -p -I vhdl multi4

                 @@@@    @@@       @        @@@@ @  @@@@   @@@@
                  @@      @        @       @    @@   @@     @
                  @@      @       @@@     @@     @    @@   @
                   @@    @        @@@     @@@          @@ @
                   @@    @       @  @@     @@@@         @@
                    @@  @        @  @@       @@@@       @@
                    @@  @       @    @@        @@@      @@
                     @@@        @@@@@@@   @      @@     @@
                     @@@       @      @@  @@     @@     @@
                      @        @      @@  @@@    @      @@
                      @      @@@@    @@@@ @  @@@@     @@@@@@

                           VHDL Analyzer for SYnthesis

                   Alliance CAD System 5.0 20040928,  vasy 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   Contributor(s):             Frederic Petrot
                   E-mail        : alliance-users@asim.lip6.fr

        --> Run VHDL Compiler
        --> Compile file multi4
        --> Drive Alliance file multi4
```

We now simulate the behavioral description with the pat file provided in the example. Before that we need to change some environment variables.

```
% setenv MBK_CATAL_NAME CATAL_ASIMUT_VASY
% setenv MBK_IN_LO vst
% setenv MBK_OUT_LO vst
```

We don't change MBK_WORK_LIB since it is the same used in deriving the behavioral description. We have to modify our Makefile as follows.

```
ALLIANCE_BIN=$(ALLIANCE_TOP)/bin

VASY   = $(ALLIANCE_BIN)/vasy
ASIMUT = $(ALLIANCE_BIN)/asimut

multi4.vbe : multi4.vhdl
      $(VASY) -a -B -o -p -I vhdl multi4

res_vasy_1.pat : multi4.vbe
      $(ASIMUT) -b multi4 multi4 res_vasy_1
```

We have added one macro to use ASIMUT and one more dependency relation. The target in this case depends on the behavioral file we have already obtained. If it were not available the first dependency relation will be executed before executing the second one.

Makefiles not only check dependencies but also for modification times of the files. So, in case the VHDL file is newer than the behavioral file when trying to run the simulation on it, the behavioral will be updated running VASY on the modified VHDL file. Let's run the Makefile on the second target to simulate the behavioral file.

```
% make res_vasy_1.pat
/usr/local/alliance/bin/asimut -b multi4 multi4 res_vasy_1

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @   @@   @
            @@@     @@     @     @                           @    @@    @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@       @@
           @  @@     @@@@     @@@@    @@@ @@  @@   @@    @@       @@
           @  @@       @@@@     @@    @@  @@  @@   @@    @@       @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@       @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@       @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMUlation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching 'multi4' ...
BEH : Compiling 'multi4.vbe' (Behaviour) ...
making GEX ...

searching pattern file : 'multi4' ...
restoring ...

linking ...
executing ...
###----- processing pattern 0 : 10000 ps -----###
###----- processing pattern 1 : 20000 ps -----###
...
###----- processing pattern 453 : 4540000 ps -----###
###----- processing pattern 454 : 4550000 ps -----###
###----- processing pattern 455 : 4560000 ps -----###
```

We are now ready to start the synthesis process with BOOM, BOOG and LOON. First, we modify our make file to make it run BOOM. In it we add the lines defining TOUCH and also add the corresponding dependency relation.

```
ALLIANCE_BIN=$(ALLIANCE_TOP)/bin

VASY   = $(ALLIANCE_BIN)/vasy
ASIMUT = $(ALLIANCE_BIN)/asimut
BOOM   = $(ALLIANCE_BIN)/boom
TOUCH  = touch

multi4.vbe : multi4.vhdl
      $(VASY) -a -B -o -p -I vhdl multi4

res_vasy_1.pat : multi4.vbe
      $(ASIMUT) -b multi4 multi4 res_vasy_1

boom.done : multi4_o.vbe
      @$(TOUCH) boom.done

multi4_o.vbe : multi4.vbe multi4.boom res_vasy_1.pat
      $(BOOM) -VP multi4 multi4_
```

The first dependency, requesting the generation of the boom.done file, will let us check if boom has been runned, and also will run indirectly BOOM. The @ before the touch command will avoid the echoing of it (every command line that starts with a @ won't display). But first, we modify one environment variable.

```
% setenv MBK_CATAL_NAME CATAL
```

Then, we call BOOM with the following make call (The output has been shortened to make it fit).

```
% make boom.done
/usr/local/alliance/bin/boom -VP multi4 multi4_o
                   @@@@@@@                         @@@     @@@
                     @@   @@                        @@      @@
                     @@    @@                       @@@   @@@
                     @@    @@     @@@       @@@     @@@   @@@
                     @@   @@    @@   @@   @@   @@   @ @@ @ @@
                     @@@@@@    @@     @@ @@     @@  @ @@ @ @@
                     @@    @@  @@     @@ @@     @@  @ @@@  @@
                     @@     @@ @@     @@ @@     @@  @  @@  @@
                     @@     @@ @@     @@ @@     @@  @  @   @@
                     @@    @@   @@   @@   @@   @@   @      @@
                   @@@@@@@@       @@@       @@@    @@@    @@@@

                              BOOlean Minimization

                   Alliance CAD System 5.0 20040928,  boom 5.0
                   Copyright (c) 2000-2005,     ASIM/LIP6/UPMC
                   Author(s):                  Ludovic Jacomme
                   E-mail        : alliance-users@asim.lip6.fr

        --> Parse BEH file multi4.vbe
        --> Check figure multi4
        --> Parse parameter file multi4.boom
            keep signal 'rtlcarry_0 7'
...
            keep signal 'rtlcarry_4 1'
        --> Optimization parameters
            Algorithm : simulated annealing
            Keep aux  : no
            Area      : 100 %
            Delay     :   0 %
            Level     :   0
        --> Initial cost
            Surface  : 216250
            Depth    : 25
            Literals : 194
        --> Translate Abl to Bdd
            Total Bdd nodes 518
        --> Optimization % 100
        --> Final cost
            Surface  : 295000
            Depth    : 18
            Literals : 251
        --> Post treat figure multi4
        --> Drive BEH file multi4_o
```

This will generate an empty file with the name boom.done that will signal that BOOM has been runned, and the corresponding minimized behavioral description has been generated. We now modify the Makefile to run BOOG. We add the following lines to our Makefile.

```
BOOG   = $(ALLIANCE_BIN)/boog

TARGET_LIB      = $(ALLIANCE_TOP)/cells/sxlib

boog.done : multi4_o.vst
      @$(TOUCH) boog.done

multi4_o.vst : multi4_o.vbe
      $(BOOG) multi4_o
```

And, before running make we set the target library (accordingly to the original Makefile environment variables setting).

```
% setenv TARGET_LIB $ALLIANCE_TOP/cells/sxlib
```

At this point only this variable has not been set yet. After this, we are ready to run BOOG as we have done with BOOM (the output has been shortened to make it fit).

```
% make boog.done
/usr/local/alliance/bin/boog multi4_o

                  @@@@@@@                             @@@@ @
                    @@   @@                         @@    @@
                    @@    @@                       @@      @
                    @@    @@     @@@       @@@    @@
                    @@   @@    @@   @@   @@   @@  @@
                    @@@@@@    @@     @@ @@     @@ @@     @@@@@
                    @@    @@  @@     @@ @@     @@ @@     @ @@
                    @@     @@ @@     @@ @@     @@ @@    @  @@
                    @@     @@ @@     @@ @@     @@  @@      @@
                    @@    @@   @@   @@   @@   @@    @@    @@
                  @@@@@@@@       @@@       @@@        @@@@

                         Binding and Optimizing On Gates

             Alliance CAD System 5.0 20040928, boog 5.0 [2003/01/09]
             Copyright (c) 2000-2005,                ASIM/LIP6/UPMC
             Author(s):                             Fran?ois Donnet
             E-mail        :            alliance-users@asim.lip6.fr

        MBK_VDD        : vdd
        MBK_VSS        : vss
        MBK_IN_LO      : vst
        MBK_OUT_LO     : vst
        MBK_WORK_LIB   : .
        MBK_TARGET_LIB : /usr/local/alliance/cells/sxlib

Reading default parameter...
50% area - 50% delay optimization
Reading file 'multi4_o.vbe'...
Controlling file 'multi4_o.vbe'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Mapping Warning: Cell 'halfadder_x4' isn't supported
...
Controlling lib '/usr/local/alliance/cells/sxlib'...
Preparing file 'multi4_o.vbe'...
Capacitances on file 'multi4_o.vbe'...
Unflattening file 'multi4_o.vbe'...
Mapping file 'multi4_o.vbe'...
Saving file 'multi4_o.vst'...
Quick estimated critical path (no warranty)...3362 ps from 'x 0' to 'r 3'
Quick estimated area (with over-cell routing)...218250 lambda?
Details...
        xr2_x1: 26
...
        ao2o22_x2: 1
        Total: 145
Saving critical path in xsch color file 'multi4_o.xsc'...
End of boog...
```

We now set our Makefile to run LOON. Since the environment is the same to the one used by BOOG we don't need to modify it. We add to our Makefile the following lines.

```
LOON   = $(ALLIANCE_BIN)/loon

loon.done : multi4.vst
      @$(TOUCH) loon.done

multi4.vst : multi4_o.vst
      $(LOON) multi4_o multi4
```

And, run make as follows (the output has been cropped to make it fit).

```
% make loon.done
/usr/local/alliance/bin/loon multi4_o multi4

                   @@@@@@                          @@@    @@@
                     @@                             @@     @
                     @@                             @@@    @
                     @@           @@@       @@@     @ @@   @
                     @@         @@   @@   @@   @@   @ @@   @
                     @@        @@     @@ @@     @@  @  @@  @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@        @@     @@ @@     @@  @   @@ @
                     @@      @ @@     @@ @@     @@  @    @@@
                     @@     @   @@   @@   @@   @@   @     @@
                   @@@@@@@@@@     @@@       @@@    @@@    @@

                           Local optimization on Nets

             Alliance CAD System 5.0 20040928, loon 5.0 [2003/12/07]
             Copyright (c) 2000-2005,                ASIM/LIP6/UPMC
             Author(s):                             Fran?ois Donnet
             E-mail        :             alliance-users@asim.lip6.fr

        MBK_IN_LO       : vst
        MBK_OUT_LO      : vst
        MBK_TARGET_LIB  : /usr/local/alliance/cells/sxlib

Reading default parameter...
50% area - 50% delay optimization
Reading file 'multi4_o.vst'...
Reading lib '/usr/local/alliance/cells/sxlib'...
Capacitances on file 'multi4_o.vst'...
Delays on file 'multi4_o.vst'...5635 ps
Area on file 'multi4_o.vst'...218250 lamda? (with over-cell routing)
Details...
        xr2_x1: 26 (26%)
...
        ao2o22_x2: 1 (1%)
        Total: 145
Worst RC on file 'multi4_o.vst'...321 ps
Inserting buffers on critical path for file 'multi4.vst'...6 buffers inserted -> 5409 ps
Improving RC on critical path for file 'multi4.vst'...5265 ps
Improving all RC for file 'multi4.vst'...
Worst RC on file 'multi4.vst'...321 ps
Area on file 'multi4.vst'...224500 lamda? (with over-cell routing)
Details...
        xr2_x1: 26 (26%)
...
        ao2o22_x2: 1 (1%)
        Total: 151
Critical path (no warranty)...5265 ps from 'y 2' to 'r 7'
Saving file 'multi4.vst'...
Saving critical path in xsch color file 'multi4.xsc'...
End of loon...
```

Before proceeding to place our design, we test it with ASIMUT using the pattern file we used to check the behavioral description produced by VASY. For this we add the following lines to our Makefile.

```
res_synth_1.pat : multi4.vst
      $(ASIMUT) multi4 multi4 res_synth_1
```

Before running the simulation we need to set the MBK_CATAL_LIB variable (all others do not need to be changed).

```
% setenv MBK_CATAL_LIB $TARGET_LIB
```

And, run ASIMUT using the following make command (the output has been cropped).

```
% make res_synth_1.pat
/usr/local/alliance/bin/asimut multi4 multi4 res_synth_1

             @        @@@@ @     @                            @@@@@@@@@@
             @       @    @@    @@@                           @   @@   @
            @@@     @@     @     @                           @    @@    @
            @@@     @@@              @@@ @@ @@@   @@@  @@@@       @@
           @  @@     @@@@     @@@@    @@@ @@  @@   @@    @@       @@
           @  @@       @@@@     @@    @@  @@  @@   @@    @@       @@
          @    @@        @@@    @@    @@  @@  @@   @@    @@       @@
          @@@@@@@   @      @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@     @@   @@    @@  @@  @@   @@    @@       @@
         @      @@  @@@    @    @@    @@  @@  @@   @@   @@@       @@
       @@@@    @@@@ @  @@@@   @@@@@@ @@@@ @@@ @@@   @@@@  @@    @@@@@@

                                A SIMUlation Tool

                 Alliance CAD System 5.0 20040928, asimut v3.02
                 Copyright (c) 1991...1999-2005, ASIM/LIP6/UPMC
                 E-mail        :    alliance-users@asim.lip6.fr

        Paris, France, Europe, Earth, Solar system, Milky Way, ...
initializing ...
searching ‘multi4‘ ...
compiling ‘multi4‘ (Structural) ...

flattening the root figure ...

searching ‘ao2o22_x2‘ ...
BEH : Compiling ‘ao2o22_x2.vbe‘ (Behaviour) ...
making GEX ...
...
searching ‘buf_x2‘ ...
BEH : Compiling ‘buf_x2.vbe‘ (Behaviour) ...
making GEX ...

searching pattern file : ‘multi4‘ ...
restoring ...
linking ...
executing ...
###----- processing pattern 0 : 10000 ps -----###
...
###----- processing pattern 455 : 4560000 ps -----###
```

We can open the .pat file generated by ASIMUT and check that after each signal change the correct outputs are obtained within the 10ns cycle used in the input pattern file. To check that the output obtained this time is identical to the one when ASIMUT was applied to the behavioral description you can use the **diff** command to see if they are identical. But take into account that in the case of the behavioral file the delay used by ASIMUT is zero, and the ones used above are those of the corresponding cells used in implementing the design. They sum up to an approximately 5.3ns (see output of LOON and the delay of the critical path), so at the time of changing of the inputs, the outputs of the design in the output pat files will differ. We are now ready to place our design using OCP. We need to add the following lines to our Makefile. And in this case no special variable setting is needed (all them are set correctly with all the changes done in previous steps).

```
OCP    = $(ALLIANCE_BIN)/ocp

multi4_p.ap : res_synth_1.pat
      $(OCP) -v -gnuplot -ioc multi4  multi4 multi4_p
```

We place the design using the following command (the output has been shortened to make it fit).

```
% make multi4_p.ap
/usr/local/alliance/bin/ocp -v -gnuplot -ioc multi4  multi4 multi4_p

                            @@@         @@@@ @ @@@@@@@
                           @@  @@     @@    @@   @@   @@
                         @@     @@   @@      @   @@    @@
                        @@       @@ @@       @   @@    @@
                        @@       @@ @@           @@   @@
                        @@       @@ @@           @@@@@
                        @@       @@ @@           @@
                        @@       @@ @@           @@
                         @@     @@   @@      @   @@
                          @@   @@     @@    @@   @@
                            @@@         @@@@   @@@@@@

                           Placer for Standards Cells

                   Alliance CAD System 5.0 20040928,   ocp 5.0
                   Copyright (c) 2001-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

o ALLIANCE environment:
  o ALLIANCE_TOP   : /usr/local/alliance
o MBK environment:
  o MBK_IN_LO      : vst
...
  o MBK_CATA_LIB   : .
                     /usr/local/alliance/cells/sxlib
...
 o Special Net detected : vss
 o Special Net detected : vdd
 o Number total of instances is ....  151
 o Number of instances to place is ....  151
 o Number of instances already placed is ....  0
 o Number of nets is .... 159
 o Computing Initial Placement ...
 o User Margin : 0.2
 o Number of Rows : 10
 o Width of the abutment box : 108
 o Height of the abutment box : 100
 o conspace : 13.5 1st connector : 6.75
 o conspace : 13.5 1st connector : 6.75
 o Initial Placement Computing ... done.
 o Beginning global placement ....
 o Initial RowCost = 73.6
 o Initial BinCost = 88
 o Initial NetCost = 14396
 o Initial Cost = 1
 o Computing Initial Temperature ...
Loop = 1, Temperature = 0.351534, Cost = 1.0917
  RowCost = 65.2, BinCost = 156, NetCost = 14861
  Success Ratio = 99.4941%, Dist = 1, Delta = 0.5
...
Loop = 49, Temperature = 5.62198e-05, Cost = 0.552694
  RowCost = 34, BinCost = 60, NetCost = 7887.5
  Success Ratio = 1.872%, Dist = 0.1, Delta = 0.590525
 o Global Placement finished .....
 o Gain for RowCost      = 53.8043%
 o Gain for BinCost      = 31.8182%
 o Gain for NetCost      = 45.2105%
 o NetCost Estimated = 7887.5
 o Final Optimization in process ...
 o Net Cost before Final Optimization... 8405.5
 o Final Optimization succeeded ...
 o Final Net Cost ..... 6358.5
 o Final Net Cost Optimization ..... 24.3531%
 o Total Net Optimization .... 55.8315%

Ocp : placement finished
gnuplot files created - in order to use them, please type :
..
gnuplot view.gpl
NO PREPLACEMENT GIVEN
 o Destruction of DATABASE ....
```

We now route the design using NERO. For this, we modify our Makefile adding the following lines.

```
NERO   = $(ALLIANCE_BIN)/nero

multi4.ap : multi4_p.ap multi4.vst
      $(NERO) -V -$(METAL_LEVEL) -p multi4_p multi4 multi4
```

The environment setting needed by NERO is almost the same of the one used by OCP so we modify nothing this time. However, we define a new variable to set the number of layers used in the routing of the design.

```
% setenv METAL_LEVEL 2
```

We then run NERO with the following command (the output has been shortened to make it fit).

```
% make multi4.ap
/usr/local/alliance/bin/nero -V -2 -p multi4_p multi4 multi4

                   @@@    @@@           @@@@@@@
                    @@     @              @@   @@
                    @@@    @              @@    @@
                    @ @@   @    @@@@@     @@    @@      @@@
                    @ @@   @   @     @    @@   @@     @@   @@
                    @  @@  @  @@      @@   @@@@@     @@      @@
                    @   @@ @  @@@@@@@@@   @@  @@     @@      @@
                    @   @@ @  @@          @@   @@   @@      @@
                    @    @@@  @@      @   @@   @@   @@      @@
                    @     @@   @@    @@   @@    @@   @@   @@
                   @@@    @@     @@@@   @@@@@   @@@    @@@

                               Negotiating Router

                   Alliance CAD System 5.0 20040928,  nero 5.0
                   Copyright (c) 2002-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr
                                S/N 20021117.1

  o  MBK environment :
     MBK_IN_LO       := vst
...
     MBK_CATA_LIB    := .
                        /usr/local/alliance/cells/sxlib
...
                        /usr/local/alliance/cells/padlib
     MBK_CATAL_NAME  := CATAL
     MBK_VDD         := vdd
     MBK_VSS         := vss
     MBK_SEPAR       := .
  o  Loading netlist "multi4"...
  o  Loading layout "multi4_p"...
  o  Flattening layout...
  o  Flattening netlist...
  o  Building netlist dual representation (lofigchain)...
  o  Binding logical & physical views...
  o  Loading design into grid...
  o  Local routing stage.
     - [ 158] (hp :=     9) "r 0"
...
     - [   0] (hp :=   192) "x 1"

  o  Routing stats :
     - routing iterations    := 314327
     - re-routing iterations := 22122
     - ratio                 := 6.57514%.

  o  Dumping routing grid.
  o  Saving MBK figure "multi4".
  o  Saving layout as "multi4"...
```

We can check the correctness of the design using Cougar and LVX. We must set some variables to run the first one.

```
% setenv RDS_TECHNO ../etc/techno-035.rds
% setenv RDS_TECHNO_NAME $RDS_TECHNO
% setenv MBK_IN_LO al
% setenv MBK_OUT_LO al
% setenv MBK_CATAL_LIB $TARGET_LIB
```

We modify our Makefile accordingly and add the corresponding dependency relation and command.

```
COUGAR = $(ALLIANCE_BIN)/cougar

RDS_TECHNO      = ../etc/techno-035.rds

multi4_e.al : multi4.ap
      $(COUGAR) -v -ac multi4 multi4_e
```

We then use the following make command.

```
% make multi4_e.al
/usr/local/alliance/bin/cougar -v -ac multi4 multi4_e

               @@@@ @
             @@    @@
            @@      @
           @@       @    @@@    @@@  @@@@    @@@@@@   @@@@    @@@ @@@
           @@          @@   @@   @@    @@   @@  @@   @@   @    @@@  @@
           @@         @@     @@  @@    @@   @   @@   @@   @@   @@   @@
           @@         @@     @@  @@    @@   @   @      @@@@@   @@
           @@         @@     @@  @@    @@    @@@     @@   @@   @@
            @@      @ @@     @@  @@    @@   @@      @@    @@   @@
             @@    @@  @@   @@   @@   @@@   @@@@@@  @@   @@@   @@
               @@@@      @@@      @@@@  @@ @@   @@@  @@@@  @@ @@@@
                                            @     @
                                             @@@@@

                       Netlist extractor ... formerly Lynx

                  Alliance CAD System 5.0 20040928, cougar 1.21
                  Copyright (c) 1998-2005,       ASIM/LIP6/UPMC
                  Author(s):  Ludovic Jacomme and Gregoire Avot
                  Contributor(s):              Picault Stephane
                  E-mail        :   alliance-users@asim.lip6.fr

        ---> Parse technological file ../etc/techno-035.rds
                  RDS_LAMBDA        = 24
                  RDS_UNIT          = 80
                  RDS_PHYSICAL_GRID = 2
                  MBK_SCALE_X       = 100
        ---> Extract symbolic figure multi4
                ---> Translate Mbk -> Rds
                ---> Build windows
                <--- 288
                ---> Rectangles    : 4895
                ---> Figure size   : (   -100,   -116 )
                                     (  54100,  50116 )
                ---> Cut transistors
                <--- 0
                ---> Build equis
                <--- 170
                ---> Delete windows
                ---> Build signals
                <--- 170
                ---> Build instances
                <--- 246
                ---> Build transistors
                <--- 0
                ---> Save netlist
        <--- done !
        ---> Total extracted capacitance
        <--- 4.5pF
```

We can also use Cougar to extract a SPICE file from the routed design. We must change the following variables.

```
% setenv MBK_IN_LO spi
% setenv MBK_OUT_LO spi
```

And set a few new ones.

```
% setenv SPI_MODEL $ALLIANCE_TOP/etc/spimodel.cfg
% setenv MBK_SPI_MODEL $SPI_MODEL
% setenv MBK_SPI_ONE_NODE_NORC "true"
% setenv MBK_SPI_NAMEDNODES "true"
```

and add the following lines to our Makefile.

```
SPI_MODEL       = $(ALLIANCE_TOP)/etc/spimodel.cfg

multi4_e.spi : multi4.ap
      $(COUGAR) -v -ac multi4 multi4_e
```

Now we can run Cougar again using the following command.

```
% make multi4_e.spi
/usr/local/alliance/bin/cougar -v -ac multi4 multi4_e

               @@@@ @
             @@    @@
            @@      @
           @@       @    @@@    @@@  @@@@    @@@@@@   @@@@    @@@ @@@
           @@          @@   @@   @@    @@   @@  @@   @@   @    @@@  @@
           @@         @@     @@  @@    @@   @   @@   @@   @@   @@   @@
           @@         @@     @@  @@    @@   @   @      @@@@@   @@
           @@         @@     @@  @@    @@    @@@     @@   @@   @@
            @@      @ @@     @@  @@    @@   @@      @@    @@   @@
             @@    @@  @@   @@   @@   @@@   @@@@@@  @@   @@@   @@
               @@@@      @@@      @@@@  @@ @@   @@@  @@@@  @@ @@@@
                                            @     @
                                             @@@@@

                       Netlist extractor ... formerly Lynx

                  Alliance CAD System 5.0 20040928, cougar 1.21
                  Copyright (c) 1998-2005,       ASIM/LIP6/UPMC
                  Author(s):  Ludovic Jacomme and Gregoire Avot
                  Contributor(s):              Picault Stephane
                  E-mail        :   alliance-users@asim.lip6.fr

        ---> Parse technological file ../etc/techno-035.rds

                  RDS_LAMBDA        = 24
                  RDS_UNIT          = 80
                  RDS_PHYSICAL_GRID = 2
                  MBK_SCALE_X       = 100

        ---> Extract symbolic figure multi4

                ---> Translate Mbk -> Rds

                ---> Build windows
                <--- 288

                ---> Rectangles    : 4895
                ---> Figure size   : (   -100,   -116 )
                                     (  54100,  50116 )

                ---> Cut transistors
                <--- 0
                ---> Build equis
                <--- 170
                ---> Delete windows
                ---> Build signals
                <--- 170
                ---> Build instances
                <--- 246
                ---> Build transistors
                <--- 0
                ---> Save netlist

        <--- done !

        ---> Total extracted capacitance
        <--- 4.5pF
```

We now use LVX to compare the structural description of our design with the first one extracted by Cougar. We must change some environment variables as follows.

```
% setenv MBK_IN_LO vst
% setenv MBK_OUT_LO vst
```

All other setting are the same, so we add the follwing lines to our Makefile.

```
LVX     = $(ALLIANCE_BIN)/lvx

lvx.done : multi4.vst multi4_e.al multi4_e.spi
      $(LVX) vst al multi4 multi4_e -f
      $(TOUCH) lvx.done
```

And run the comparison as follows.

```
% make lvx.done
/usr/local/alliance/bin/lvx vst al multi4 multi4_e -f

                       @@@@@@      @@@@    @@@ @@@@   @@@@
                         @@         @@      @   @@     @
                         @@         @@      @    @@   @
                         @@          @@    @      @@ @
                         @@          @@    @       @@
                         @@           @@  @        @@
                         @@           @@  @        @@@
                         @@            @@@        @  @@
                         @@      @     @@@       @    @@
                         @@     @       @       @      @@
                       @@@@@@@@@@       @      @@@    @@@@

                             Gate Netlist Comparator

                   Alliance CAD System 5.0 20040928,   lvx 1.4
                   Copyright (c) 1992-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr



***** Loading and flattening multi4 (vst)...

***** Loading and flattening multi4_e (al)...


***** Compare Terminals ..............
***** O.K.      (0 sec)

***** Compare Instances ............
***** O.K.      (0 sec)

***** Compare Connections ......................
***** O.K.      (0 sec)

===== Terminals .......... 18
===== Instances .......... 151
===== Connectors ......... 812


***** Netlists are Identical. *****     (0 sec)

touch lvx.done
```

That tells us that the two descriptions are identical. We cann now use DRUC to check the symbolic design rules used. To do this, we set a new variable a change one.

```
% setenv RDS_TECHNO_SYMB ../etc/techno-symb.rds
% setenv RDS_TECHNO_NAME $RDS_TECHNO_SYMB
```

And add the related variable, the corresponding definition, and the corresponding dependency relation to our

Makefile.

```
DRUC   = $(ALLIANCE_BIN)/druc

RDS_TECHNO_SYMB = ../etc/techno-symb.rds

druc.done : lvx.done multi4.ap
      $(DRUC) multi4
      $(TOUCH) druc.done
```

Then, we can run DRUC with the following command.

```
% make druc.done
/usr/local/alliance/bin/druc multi4

                  @@@@@@@     @@@@@@@                    @@@@ @
                    @@   @@     @@   @@                @@    @@
                    @@    @@    @@    @@              @@      @
                    @@     @@   @@    @@  @@@  @@@@  @@       @
                    @@     @@   @@   @@    @@    @@  @@
                    @@     @@   @@@@@      @@    @@  @@
                    @@     @@   @@  @@     @@    @@  @@
                    @@     @@   @@   @@    @@    @@  @@
                    @@    @@    @@   @@    @@    @@   @@      @
                    @@   @@     @@    @@   @@   @@@    @@    @@
                  @@@@@@@     @@@@@   @@@   @@@@  @@     @@@@

                               Design Rule Checker

                   Alliance CAD System 5.0 20040928,  druc 5.0
                   Copyright (c) 1993-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr

Flatten DRC on: multi4
Delete MBK figure : multi4
Load Flatten Rules : ../etc/techno-symb.rds

Unify : multi4

Create Ring : multi4_rng
Merge Errorfiles:

Merge Error Instances:
instructionCourante :  56
End DRC on: multi4
Saving the Error file figure
Done
  0

File: multi4.drc is empty: no errors detected.
touch druc.done
```

That tells us that there are not rule violations in our design. We are now ready to take our symbolic design to a real one. We use for this S2R. We first set one variable as follows.

```
% setenv RDS_TECHNO_NAME $RDS_TECHNO
```

All other variables are correctly set at this point so we add the following lines to our Makefile.

```
S2R    = $(ALLIANCE_BIN)/s2r

multi4.cif : druc.done
      $(S2R) -v multi4
```

Now we can run S2R as follows (the output has been shortened to make it fit).

```
% make multi4.cif
/usr/local/alliance/bin/s2r -v multi4

                                      @@@@
                                     @   @@
                                    @@    @@
                            @@@@@@  @@@   @@ @@@ @@@
                           @@    @   @   @@   @@@  @@
                           @@@           @    @@   @@
                            @@@@        @     @@
                              @@@@     @      @@
                           @    @@@   @    @  @@
                           @@    @@  @@@@@@   @@
                           @ @@@@@  @@@@@@@  @@@@

                        Symbolic to Real layout converter

                   Alliance CAD System 5.0 20040928,   s2r 5.0
                   Copyright (c) 2002-2005,     ASIM/LIP6/UPMC
                   E-mail        : alliance-users@asim.lip6.fr


        o loading technology file : ../etc/techno-035.rds
        o loading all level of symbolic layout : multi4
        o removing symbolic data structure
        o layout post-treating
                with top connectors,
                with sub connectors,
                with signal names,
                without scotch.
        --> post-treating model no2_x1
            rectangle merging :
            . RDS_NWELL .................................
            . RDS_PWELL .................................
            . RDS_NIMP .................................
            . RDS_PIMP .................................
            . RDS_ACTIV .................................
            . RDS_POLY .................................
            . RDS_ALU1 .................................
...
        --> post-treating model tie_x0
            rectangle merging :
            . RDS_NWELL .................................
            . RDS_PWELL .................................
            . RDS_NIMP .................................
            . RDS_PIMP .................................
            . RDS_ACTIV .................................
            . RDS_ALU1 .................................
        --> post-treating model rowend_x0
            rectangle merging :
            . RDS_NWELL .................................
            . RDS_ALU1 .................................
        --> post-treating model multi4
            ring flattenning  :
            . RDS_NWELL .................................
            . RDS_NIMP .................................
            . RDS_PIMP .................................
            . RDS_ACTIV .................................
            . RDS_POLY .................................
            rectangle merging :
            . RDS_NWELL .................................
            . RDS_NIMP .................................
            . RDS_PIMP .................................
            . RDS_ACTIV .................................
            . RDS_POLY .................................
            . RDS_ALU1 .................................
            . RDS_ALU2 .................................
            . RDS_ALU3 .................................
        o saving multi4.cif
        o memory allocation informations
        --> required rectangles = 4913  really allocated = 7
        --> Number of allocated bytes: 298508
```

That gives us a cif file of our design. This can then be visualized using dreal.

Bibliography

[1] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, Vol. EC-13, No.1, pp. 14-17, Feb., 1964.

[2] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza* Vol. 34, pp. 349-356, December, 1965.

[3] Alberto Palacios Pawlovsky, "A new algorithm for the configuration of fast adder trees,"*IEICE Trans. Fundamentals* Vol. E83-A, No.12, pp. 2426-2430, December, 2000.