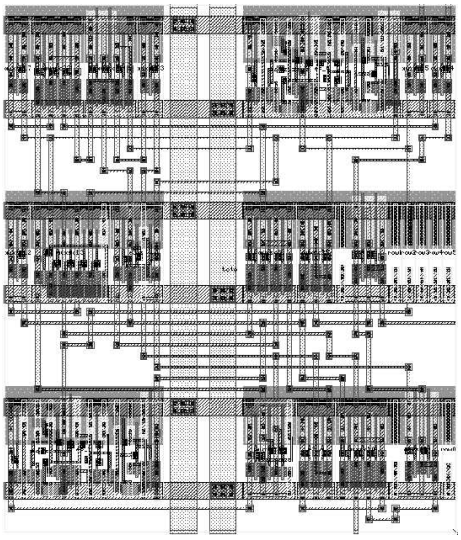
ALLIANCE TUTORIAL

Pierre & Marie Curie University
2001 - 2004

PART 1
Simulation

Frederic AK Kai-shing LAM
Modified by LJ



The purpose of this tutorial is to provide a quick turn of some **ALLIANCE** tools, developed at the LIP6 laboratory of Pierre and Marie Curie University.

The tutorial is composed of 3 main parts independent from each other:

- VHDL modeling and simulation
- Logical synthesis
- Place and route

Before going further you must ensure that all the environment variables are properly set (source alcenv.sh or alcenv.csh file) and that the Alliance tools are available when invoking them at the shell prompt.

All the tools used in this tutorial are documented at least with a manual page.

Contents

PART 1 : VHDL modeling and simulation

All the files used in this part are located in the `/tutorial/simulation/src` directory.
This directory contains two subdirectories and one Makefile :

- The Makefile allows you to validate automatically the entire simulation part
- **addaccu_beh** = the behavioral description (Register Transfert Level)
  - Makefile to validate automatically the entire behavioral description
  - addaccu.vbe is the behavioral description of addaccu
  - patterns.pat is the simulation patterns for addaccu
  - addaccu_dly.vbe is the behavioral description of addaccu with delay
  - patterns_dly.pat is the simulation patterns for addaccu with delay
  - addaccu4.vhdl is the behavioral description of addaccu using standard VHDL subset
- **addaccu_struct** = the structural view
  - Makefile to validate automatically the entire structural view
  - pat_new.c is the vectors generation file
  - addaccu.vbe is the behavioral description of addaccu
  - mux.vbe is the behavioral description of multiplexer
  - accu.vbe is the behavioral description of accumulator
  - alu.vbe is the behavioral description of adder
  - addaccu.vst is the structural view of addaccu
  - mux.vst is the structural view of multiplexer
  - accu.vst is the structural view of accumulator
  - alu.vst is the structural view of adder

The **ALLIANCE** tools used are :

- **vasy** : **VHDL** analyzer and convertor.
- **asimut** : **VHDL** Compiler and Simulator.
- **genpat** : Procedural generator of stimuli.

You can obtain the detailed informations on an any **ALLIANCE** tool by typing the command :

```
> man <tool name>
```

To validate the behavioral and the structural description you can :

- run the **UNIX** commands in the order indicated by this tutorial.

- validate automatically the entire behavioral (or structural) description using the command :

```
> make
```

If you want to start again this validation from the beginning, you just have to type :

```
> make clean
> make
```

1 Behavioral VHDL

1.1 Introduction

The goal of this part is to write then to simulate the behavior of a very small circuit : An accumulating adder which we will call addaccu.

The description of the behavior of addaccu will be made in **Behavioral VHDL (DATAFLOW)**.

1.2 Behavioral Description

The behavioral description of a circuit consists on a set of boolean functions calculating the outputs according to the inputs with the use of possible internal signals ; in our case, a signal which connects the output of the accumulator to the entry of the multiplexer (reg_out), another which connects the output of the multiplexer to the entry of the adder (mux_out) and finally a signal for carry (carry).

At first, you must write the file of behavioral description of addaccu. This description must be of type : without delay (without After clause).

This file will have the extension ".vbe" which is the usual extension to indicate a **VHDL** behavioral file (Vhdl BEhaviour description). This description will have three distinct parts:

- **Block 1** : The 4 bits adder.
- **Block 2** : The 4 bits multiplexer.
- **Block 3** : The 4 bits accumulator.

The circuit has the following interface:

- a 4 bits input bus a.
- a 4 bits input bus b.
- a 4 bits output bus S.
- a clock input signal ck.
- a control input signal sel.
- two alimentation inputs signals VDD and VSS.

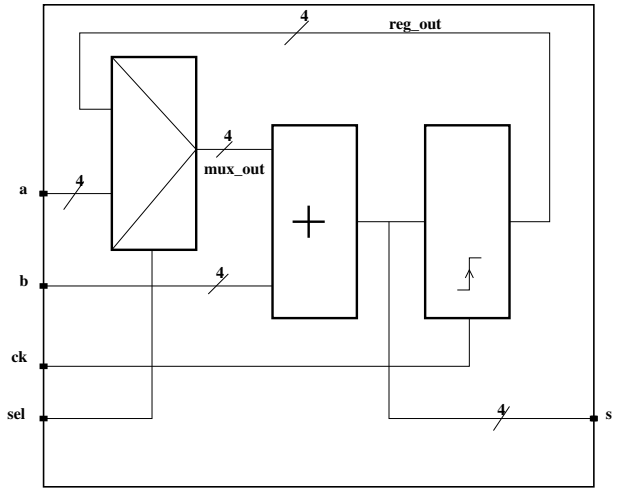


Figure 1: **accumulating adder**

1. **mux** is a 4 bits multiplexer 1 among 2
   mux truth table :
   sel = 0 => mux_out = a
   sel = 1 => mux_out = reg_out

2. **alu** is a 4 bits adder
   s = b + mux_out

3. **accu** is a register (flip-flop)
   ck = 0 => reg_out = reg_out
   ck = 1 => reg_out = reg_out
   ck : 0->1 => reg_out = s

Then you must validate your description while compiling with **ASIMUT**.

```
> asimut -b -c <file name>
```

- **file name** is the file name of your behavioral description without extension (**addaccu**).
- **-b** option to indicate that the description is purely behavioral.
- **-c** option to compile without simulating.

If you do not wish to use the environment variables positioned by default, other environment variables can be used by **ASIMUT**.

```
> MBK_WORK_LIB = .
> MBK_CATA_LIB = .
> MBK_CATAL_NAME = CATAL
> MBK_IN_LO = VST
```

under Bash :

```
> export var = value
```

under standard Bourne Shell :

```
> var = value
> export var
```

under C Shell :

```
> setenv var value
```

The meaning of these variables is to be discovered in the **man** of **ASIMUT** tool.

1.3 Description with Standard VHDL subset

Alliance tools use a very particular and restricted **VHDL** subset (vbe and vst file format).

If you want to describe the behavior of your circuit (at Register Transfert Level) with a more common **VHDL** subset you can use **VASY** to automatically convert your **VHDL** descriptions in Alliance subset.

The file addaccu4.vhdl is a description of the addaccu circuit, using classical **VHDL** subset (with process statements, IEEE 1164 VHDL types, aritmetic operators etc ...)

You can convert this description to the **.vbe** file format using **VASY** :

```
> vasy -Vao addaccu4.vhdl
```

You can then compile and simulate the generated file addaccu4.vbe using **asimut** exactly as it has been done with the addaccu.vbe file.

1.4 Stimuli of test

Once the behavioral description compiled successfully (without any error), to validate your description you must write a file of nonexhaustive but intelligent vectors of test.

Therefore you must write a file **patterns.pat** which contains a dozen vectors of test. These vectors of test make it possible to check that the adder makes the additions well with or without carry propagation , that the multiplexer gives the good operand to the input of the adder following the value of **sel** signal and finally, that the accumulator correctly memorizes the output value of the adder.

In order not to have signals overlapping temporally (phenomenon of " glitch "), you will use a clock with very high period (tck = 100ns) compared to the propagation times. The clock must respect the following rate: 1 low state of 50 ns, then 1 high state of 50 ns, etc...

If the **PAT** syntax does not appear to you obvious, have a look to the man giving the patterns files format : PAT format.

```
> man 5 pat
```

The **5** refers here to the class of handbooks for files formats.

- **man 1** : User Commands.
- **man 2,3** : Libraries.
- **man 5** : Files format.
- **man 7** : Environment variables.

1.5 Simulation

Now you only have to simulate your addaccu with your vectors of tests, without any delay in order to check very quickly that the results on the outputs are well those which you wait.

```
> asimut -b addaccu patterns result_vbe
```

- **addaccu** : file name of the behavioral description (**addaccu.vbe**).
- **pattern** : file name of the vectors (**pattern.pat**).
- **result_vbe** : file name of the patterns result (**result_vbe.pat**).
- **-b** : option to indicate a purely behavioral description.

The file of resulting vectors must be seriously analyzed to check the results of simulation. It is possible to use the graphical pattern viewer **xpat** to analyze the results of the simulation.

1.6 Delays

The behavioral description written previously includes only zero-delay concurrent assignements. It is however possible to specify propagation times by using AFTER clauses, because the operations in a real circuit are not done instantaneously. For more details, do refer to the man for VBE files format.

You must modify your behavioral description to add delays :

- For the adder : **4 ns**.
- For the multiplexer : **2 ns**.
- For the accumulator : **3 ns**.

The installation of the delay for the accumulator requires an intermediate signal **reg** because you cannot put delay on a signal of **register** type. In the test vectors file, it is necessary to put the option **spy** on the signals with delays so that we can see these delays. In the contrary case, these signals are sampled only at the times of the clock-edge.

Then you must validate this modified behavioral description while simulating with **asimut** .

```
> asimut -b addaccu_dly patterns_dly result_dly
```

The results obtained (result_dly.pat) must be different from those obtained without AFTER clauses (result_vbe.pat). To understand why, it is necessary to deeply analyze the temporal behavior of your circuit. The step of 50 ns used for the test vectors does not really make possible to observe the true temporal behavior of your circuit. You can spy on all the transitions from an internal signal or an output by specifying this characteristic while declaring in the file of test vectors (option **spy** , for more details, consult the man for patterns files format).

2 Structural VHDL

2.1 Introduction

The goal of this part is to write then to simulate in a hierarchical way the structural view of the circuit presented in first part of this Tutorial. The circuit will be describe in two levels of hierarchy :

- The first level will write the circuit like the instanciation of three blocks.
- The second level will write each of the three blocks in term of elementary gates of the standard library.

Structural description of addaccu will be made in **STRUCTURAL VHDL** .
This part contains five distinct steps:

- **step 1** : Generation of the complete set of vectors and validation of the addaccu.
- **step 2** : **VHDL** structural description of the addaccu.
- **step 3** : Simulation and validation of the structural addaccu on a hierarchical level.
- **step 4** : structural description and validation of each block.
- **step 5** : Simulation and validation of the structural addaccu on 2 hierarchical levels.

2.2 Stimuli Generation

Normally, the behavioral description has been successfully compiled, and validated with some hand made vectors. Now you must create a file of test vectors more consequent (a hundred clock-edges).

However, the writing of the stimuli file directly is a tiresome work. The tool **genpat** enables you to undertake this work in a procedural way. The language **genpat** is a subset of " C " functions. For more informations on genpat and the functions of the associated library do not hesitate to use the command:

```
> man genpat
```

Moreover, each basic function from **genpat** has its man, the functions are in capital letters, as by example:

```
> man AFFECT
```

Here are some suggestions for your file of vectors generation :

- Write a function independent of the management of the clock. This clock will be synchronized on 2 times: a low state of 50 ns followed by a high state of 50 ns.
- All the inputs of the circuit must be positioned in the first vector.
- Initialize the accumulating register with the function **INIT**.

Once your file **pat_new.c** is written you must compile it. The following commands make it possible to compile the file of procedural description and to generate the file of vectors pat_new.pat.

```
> genpat pat_new
```

If no error has occurred, the file **pat_new.pat** is now created. You only have to simulate your behavioral addaccu with this new set of vectors

```
> asimut -b -zerodelay addaccu pat_new res_new
```

The -zerodelay option states here that you wish a purely boolean simulation (without considering the propagation times). You obtain then a file of vectors (res_new.pat) result.

This file will be useful to you for the validation of the next stages

2.3 Structural View

The objective here is to realize a hierarchy on one level by making so that the structural view of the accumulating adder addaccu.vst instancies the behavioral description of the 3 basic components, the adder alu.vbe, the multiplexer mux.vbe and the accumulator accu.vbe.
Initially you must write the structural description file of addaccu. This file will have the extension " vst " which is the usual extension to indicate a **VHDL** structural file (Vhdl Structural view). This view will contain the instanciation of three independent blocks:

Block 1 : The 4 bits adder.
Block 2 : The 4 bits Multiplexer.
Block 3 : The 4 bits accumulator.

You must create a **CATAL** file containing the identifier of each block followed by the attribute 'C' indicating that it is a basic element of the hierarchy. This shows you the importance of the **CATAL** file which forces the simulator **asimut** to use the behavioral sight of the components which are listed. You have to set the environment variable MBK_IN_LO:

```
> MBK_IN_LO = vst
> export MBK_IN_LO
```

The meaning of all the usable variables is to be discovered in the man of **asimut** tool.

Lastly, validate your structural description while compiling with
asimut .

```
> asimut -c addaccu
```

Then simulate your circuit with the vectors file obtained previously (the res_new.pat file obtained by simulation zero-delay of the behavioral description).

```
> asimut -zerodelay -nores addaccu res_new
```

The -nores option states here that you do not wait a result file. When you do not have any more error of simulation you will have to create the structural view of each of the 3 blocks.

2.4 Structural view and validation of each block

Now you have to pass to a hierarchy on 2 levels. So it is necessary to write a structural view .vst for each basic component of the accumulating adder and to test one by one replacing the behavioral description of the basic components of the accumulating adder by their structural views by modifying the **CATAL** file (by removing the component name).

Each block (alu, accu, mux) must now be described like an interconnection of elementary gates. The gates which are to instanciate will be chosen among those available in the library of standard cells **SXLIB** . For the functionality of the various cells and their interface, the sxlib man is available. The behavioral description of each cell is present in
/alliance/cells/sxlib .

You must set the environment variable **MBK_CATA_LIB** to be able to reach these cells.

```
> MBK_CATA_LIB=/alliance/cells/sxlib
> export MBK_CATA_LIB
```

For each block adopt following methodology to replace the behavioral description of the block by its structural view:

- Write the structural view of the block **(vst)** .
- Compile this block (asimut -c <block_name>) to validate its syntax
- Remove its identifier from the **CATAL** file.
- Simulate circuit addaccu again:

```
> asimut -zerodelay -nores addaccu res_new
```

2.5 Simulation and validation of the addaccu on 2 hierarchical levels

Now you only have to simulate your addaccu described in a hierarchical way (in which the basic elements are the library cells).

- Erase the CATAL file, which is not necessary any more, the library of cells standards having its own catalogue.
- Simulate again the addaccu circuit

```
> asimut addaccu pat_new res_dly
```

Thus you will have replaced the behavioral description of the three blocks by their structural view.

- You can again simulate the addaccu circuit in order to observe its temporal behavior precisely (each cell of the standard library has a given propagation time). You will use the **spy** option for the internal signals and the outputs.

```
> asimut addaccu pat_new res_dly
```