

High speed introduction to some key aspects of Very high speed ic Hardware Description Language and the Alliance¹ tools.

Part Two:

Nigel Gunton 09/02, 09/08

Department of Design and Engineering

Grouping	individual
Corequisites	The emacs VHDL templates worksheet and a copy of the Alliance tools overview.
Prerequisites	Completion of Part One
Courses	CSI, EE, any?
Requirements	GNU/Linux or Unix system with Alliance tools, Stamina
Summary	Provides an introduction to the language and the tools used for UFEEHH-30-2
Duration	Up to 3 hrs

1. Conventions

In common with many other texts, this worksheet endeavours to follow certain typographic conventions :

Constant width

is used for directory names, filenames, commands, keywords, functions. All terms shown in constant width should be typed literally. It is also used to show the output from commands.

Constant width italic

is used in syntax and command examples where you should replace it with your own values.

Italic

is also used for general examples where you should substitute your own value(s). URLs are shown in italic.

Bold

is used within the body of the text for emphasis of a point.

2. Implementing more complex models

Having created several simple models and verified them, you will now explore two different approaches to developing a model. The first takes a high level, behavioural approach, where we describe the behaviour of the design and then let the tools implement this in terms of components from a library. The second approach is a hierarchical, structural model where you will specify the components to be used and how they are interconnected. This is sometimes called a netlist and is, in effect, a written description of a schematic.

The structural models, those with a `.vst` suffix can be viewed with the tool `xsch`. This tool works in a similar fashion to the pattern viewer introduced in the previous worksheet. You will need to use it to compare the synthesis results of the two models in this exercise.

¹This software belongs to the ALLIANCE CAD system from the CAO-VLSI team at ASIM/LIP6/UPMC laboratory. LIP6/ASIM, University P. et M. Curie, PARIS, FRANCE. There is a link to their website on <http://www.cems.uwe.ac.uk/~ngunton>

2.1. Behavioural Model

Implement the following function, it is an arbitrary piece of logic and the truth table is given below.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

By using a Karnaugh map you can derive a minimal form of the function.

$$Z = \bar{A}.B + A.C$$

BC	00	01	11	10
A				
0	0	0	1	1
1	0	1	1	0

Having derived this fairly simple expression we can describe it as a behavioural model in vhdl. Note that we say nothing about how it might be implemented. In fact the implementation will depend entirely on the target hardware. Also note the use of parentheses in the logic expression. The precedence rules in vhdl are unusual.

Create a new folder and then use emacs to enter the following code. Convert it to a dataflow description by using `vasy`.

```
entity logic_function is
    port (
        a, b, c : in bit;
        z       : out bit);
end logic_function;

architecture behavioural of logic_function is
begin
    z <= (not a and b) or (a and c);
end behavioural;
```

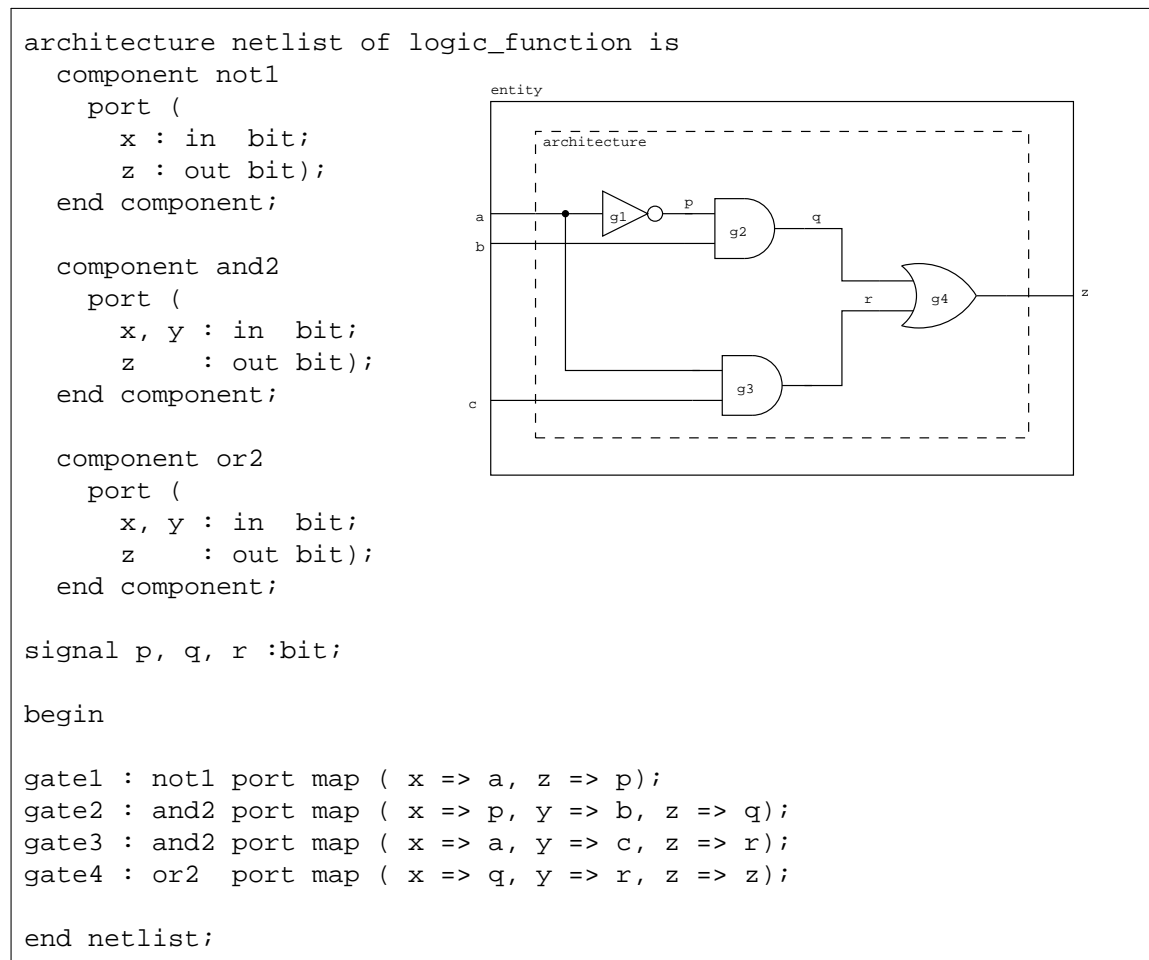
- Write a pattern file and simulate this model using `asimut`. View the results with `xpat` and confirm that they match the truth table.
- Create the structural model with `boog` and view the result with `xsch`. Is it what you expected? How many gates were used? What was the surface area and the propagation delay? Remember that this information is in the output of `boog`.
- Run a simulation using the structural model and view the results in `xpat`. What differences are there between this simulation run and the results from the earlier simulation?

2.2. Netlists

2.2.1. Preparation

Create a new folder and copy into it your *and*, *or* and *not* gates. This new folder will be treated as the 'work' folder by the tools and will expect these gates to be in it. Copy your *logic_function* into this folder. Open this file in *emacs* and delete the existing architecture. Replace the architecture with the one given below.

In this architecture we have specified the components that are needed for the design, the signals that will provide the connections between the instances of the components and the mapping of the component ports (formal ports) to the actual ports and signals of our design. The code is a written description of the diagram. Study them both and ensure that you understand the relationship between the two. Read the 'Gotchas' below²



'Gotcha's

- The component and signal declarations come before the keyword *begin*. The architecture template places the cursor after *begin*.
- The file name **must** match the entity name
- If you get in a tangle with a template then type *C-g* a couple of times. This cancels/abandons any mini-buffer command.
- The *port map* template is triggered by typing *my_instance_name : my_component_name*. You will then be prompted for the formal port name (of the component) and then the actual port or signal to which it is to be attached.

² 'gotchas' == Got yer (you) slang for 'caught you out', also for common mistakes.

2.2.2. Action

Ensure that you have `.vst` versions of all your gates in your current folder along with a copy of the pattern file. There is no need to do any pre-processing as you have created a hierarchical netlist, you just need to run the simulator. The simulator will read the component files at the start of the run. You can view your structural model with `xsch`. It should look similar to the diagram. Use `xpat` to view the results of the simulation and ensure that it matches the truth table.

3. Optional Extras

3.1. Packages

Components, amongst other vhd constructions, can be kept in a separate file. This means that you can have different models of components in different packages. You can then specify the package required in the top level model in a similar way to specifying a library. This is shown below as an optional exercise. This may be useful when developing the components for your CPU in the assignment.

It is best to start by creating a new folder for this exercise and to copy across your components and your top-level hierarchical model which will need to be renamed with a `.vhd` extension for this exercise.

Enter the following and save it as `test_package.vhd`.

```
package test is

  component not1
    port (
      x : in bit;
      z : out bit);
  end component;

  component and2
    port (
      x, y : in bit;
      z : out bit);
  end component;

  component or2
    port (
      x, y : in bit;
      z : out bit);
  end component;

end test;
```

Side-bar: Copy'n'paste, cut'n'paste You can copy'n'paste from the netlist to save time. C-x 2 to create a new window, C-x C-f `test_package.vhd` to create the new file. Make sure that you have the netlist in one window, C-x o to swap the cursor round the windows. Move the cursor to the start of the first component and type C-space, cursor to the end of the components and type M-w to copy the block into the kill ring. C-x o to move the cursor to the other window and finally C-y to yank it back. Use C-w to cut rather than copy.

Now create a new file called `my_package.pkg`, this is to inform the tool chain of the name of our package and is specific to the tools that we are using. It should contain only the line

```
work.test.all : test_package
```

Finally, almost, modify your top level file so that it is as shown below. It will now use our library of simple logic gates.

```
library work;
use work.test.all;

entity logic_function is

    port (
        a, b, c : in  bit;
        z       : out bit);

end logic_function;

architecture direct of logic_function is

    signal p, q, r : bit;

begin -- netlist

-- This is an alternative way of showing the port mapping.
-- The formal ports are inferred, only the actual ports/signals
-- are shown and are mapped to the formal ports in the order shown

    g1: Not1 port map (a, p);
    g2: And2 port map (p, b, q);
    g4: Or2  port map (q, r, z);
    g3: And2 port map (a, c, r);

end direct;
```

Convert it to a form that can be synthesized with the following command

```
vasy -VaopH -I vhd -P my_package logic_function
```

Write a test pattern for it and then simulate it. If you've got this far then congratulations. It may throw some interesting errors as it looks for structural models of the individual gates. Details from the lab tutor.