

High speed intro to some key aspects of Very high speed ic Hardware Description Language

Nigel Gunton 09/02 - 01/10/03¹

School of Computer Systems Engineering

Grouping	individual
Prerequisites	Knowledge of *nix and emacs
Courses	CRTS, CSE, any?
Requirements	GNU/Linux or Unix system with Alliance tools, Stamina
Summary	Provides an introduction to the language and the tools used for UQC149S2
Duration	3 hrs

Background

This work sheet is intended to give a guided tour to some of the key features of the language and of the Alliance toolkit. As such it is something of a 'painting by numbers' worksheet.

1. Setting the Scene

The first part of this worksheet is about setting up your environment in order that the tools work correctly. It is included here so that you gain a little experience in setting up an environment and also so that you will know how to set things up at home if you wish to use the tools at home.

1.1. Your Environment

This worksheet assumes that you are using the `bash` shell on a GNU/Linux system. The following typographic conventions are used :

Typewriter font is used for literal commands eg `ypcat passwd` , type what you see.

Italic font means replace with actual names eg `cat mypatfile`, substitute an appropriate filename. Sometimes used to represent system responses to commands.

Side-bar: How can you tell which shell you are using?

```
ypcat passwd |grep `whoami`
```

Your default shell is at the end of the line. Note that those are back-ticks, top left on my keyboard.

- 1) Type `asimut` on the command line. If you get a usage message then your environment is set up. If you get *command not found* then you need to add the following to your start-up file, `.bashrc`. This is the resource configuration file for your default shell and can be edited to customize your default command line environment. When next logged onto milly, have a look at your `.cshrc` file (more `.cshrc`)

```
. /usr/local/alliance/etc/alc_env.sh
```

Note that the dot is important. This will then set up the correct values and paths for the Alliance tools. On the command line enter `. .bashrc`

- 2) Start emacs and load (or create) the file `.emacs` in your home directory. To ensure that you are in your home directory type `cd` at the command prompt. Add the following to this file and

¹ The exercises presented here are derived from the course book, *Digital System Design with VHDL*, Zwoliński, and modified for the Alliance toolkit.

save the file.

```
(autoload 'vhdl-mode "vhdl-mode" "VHDL Editing Mode" t)
(setq auto-mode-alist
  (append
    '(("\\.vhd$" . vhdl-mode)
      ("\\.vbe$" . vhdl-mode)
      ("\\.vst$" . vhdl-mode)
      ("\\.fsm$" . vhdl-mode)
    ) auto-mode-alist))
```

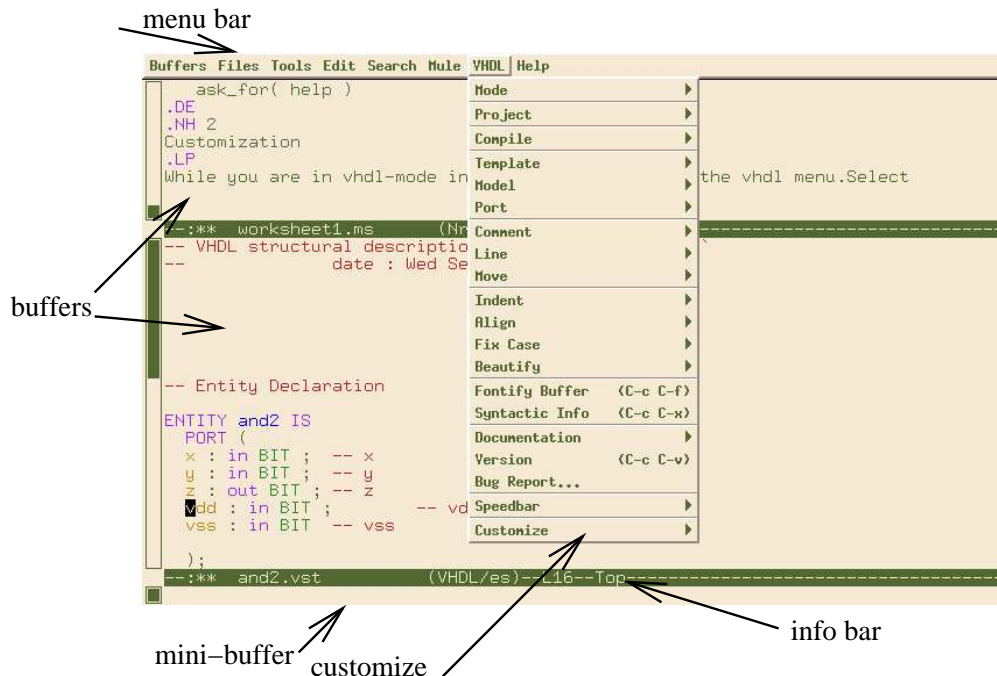
Understanding this file is not required, it's in a language called lisp. Save it and exit emacs.

Exit emacs and restart it so that it registers the changes in the start-up file and create a new file called *whatever.vbe*. The info bar should contain the words (VHDL/es) and the menu bar should now have a VHDL menu option. If not then check for error messages on opening emacs and check that your .emacs file was entered correctly.

```
If ( still stuck) then
  ask_for( help )
```

1.2. Customization

In the interests of professionalism you should ensure that all your code files have proper comment headers etc². Emacs vhdl-mode provides a great deal of support for this and allows customization of the defaults. So while you are in vhdl-mode in emacs, left-click on the vhdl menu. Select Customize→Browse VHDL Group



You will be presented with a new window within emacs listing all the configurable options for the VHDL group. Middle-click on the + sign adjoining Vhdl Electric. See below

```
[ - ]- [Group] Vhdl
  [ + ]-- [Group] Vhdl Mode
  [ + ]-- [Group] Vhdl Project
  [ + ]-- [Group] Vhdl Compile
  [ + ]-- [Group] Vhdl Style
→ [ + ]-- [Group] Vhdl Electric
  [ + ]-- [Group] Vhdl Model
```

and again on [Group] Vhdl Header and yet again on [Option] Vhdl Platform Spec. Edit the field to identify your O/S and your tool-chain, eg

Alliance 5.0 on Intel86 GNU/Linux.

NB: Press `q` to quit the customization buffer.

When you have entered an appropriate string, click on `Set for current session` and again on `Set for future sessions`. You can also set a default value for `Company Name` while you are at it. These changes will be written to your `.emacs` file. To test these changes, click on the `VHDL` menu and then on `Template→Insert header`³. A header template will be inserted into the file, the mini-buffer will become active and will be waiting for you to enter the `Title` of this file. When you have entered that and typed return the cursor will jump to the `Description` entry in the header. Enter an extremely long, longer than one line, description without worrying about reaching the end of the line and watch the behaviour of emacs. When you have entered the description you can use the arrow keys to move to the end of the header. Note that there is a `Project` field in the header as well. This field is also customizable and is left as an exercise for the adventurous. When you are finished you can kill the `.vbe` file with the emacs command `C-c k` as we will not be using it again.

Side-bar: The Customize Browser

The middle mouse button allows you to navigate the layers of the `Customize Browser`. In short if it is highlighted when you move the mouse over it then it is an active area. Alternatively `C-x o`⁴ to put the cursor in the customize browser and use 'p' and 'n' for the previous and next options, return to select/open/close the hierarchy.

2. VHDL at last

2.0.1. Templates in emacs

You are strongly advised to read/do the emacs templates worksheet concurrently with this section :)

2.1. Entities & Architectures

Create a new directory to hold the files that will be developed while working through the following examples. Return to emacs and type `C-x C-f` to open a new file. With the cursor in the mini-buffer enter `path_to_your_new_directory/and2.vhd` and press return. You will be presented with an empty buffer. Start with a header template, provide a title, eg *And2 example* and a description of *Data-flow model of a 2 input and gate* as given below. Then enter `C-x C-s` to save it.

```
entity And2 is
    port (
        x, y : in bit;
        z    : out bit);
end And2;
architecture ex1 of And2 is
begin -- ex1
    z <= x and y;
end ex1;
```

This code describes a 2 input and gate. The **entity** describes the interface between the gate and the rest of the world. It is a black box description. The **architecture** describes the behaviour or

² It is highly recommended that you get to grips with RCS, the revision control system. A worksheet is available, emacs supports it and makes it easy, try the tools menu.

³ Alternatively type `C-c C-t C-h`

⁴ `C-x o` means hold down Control and tap x, release control and press o. If confused then read the emacs crib sheet, or do the emacs tutorial `C-h t` :)

the structure of the device. It is possible to have several different **architectures** for one **entity**. Detailed discussion of the keywords will take place in the lectures but it should be fairly clear which are the language keywords.

- The **entity** description starts with the keyword followed by the name of the entity and the reserved word **is**. It finishes with the reserved word **end** followed by the name of the entity. Note that the language is **NOT** case sensitive, unlike C.
- **Ports** have direction, **in**, **out**, **inout**, plus some modifiers, and a type such as **bit**.
- The **architecture** in this first example describes the behaviour of the logic gate. The **architecture** starts with the keyword followed by a descriptive word for the architecture, keyword **of**, entity name, keyword **is**. The body of the architecture is bracketed with the **begin** and **end** keywords. The body describes how the output z is assigned (\leftarrow) the logical **and** of the two inputs x and y . This is describing the behaviour of the system. Shortly we will look at an example that describes the structure of a design.

2.1.1. More Gates

Create two more vhdl models of gates based on the example above. One should implement a 2 input **or** gate and the other a **not** gate. Name them with the `.vhd` extension. The following assumes that you have named them `or2.vhd` and `not1.vhd`. The tools to be used in the next few steps will be discussed in greater detail in the next worksheet. For the moment just follow the steps.

Side-bar: The Alliance⁵ simulation & synthesis tools

The Alliance tools are a free to use design package for the development of custom ASICs. They focus on certain aspects of the language which they divide into subsets. The subset in use is identified by the file extension. `.vhd` is used for the industry standard vhdl, `.fsm`, `.vbe` and `.vst` refer to state machine, data-flow and structural models respectively. Different tools are used to convert between these subsets.

2.2. Behavioural => Synthesizable

The VHDL language was intended for formally describing and modelling hardware systems in a consistent way. (U.S. Gov. contracts were the driving force. see also ADA). It also now gets (mis)used to create real hardware through a process known as synthesis. Because not all constructs within the language can be synthesised we need to convert behavioural models into implementable ones. We can simulate the models at each stage and ensure that the behaviour of the model still conforms as expected. The final step is to 'fit' the design to a particular technology which might be custom ASIC using a standard cell library, a CPLD or an FPGA.

3. The Alliance tools

The first step is to convert the behavioural model into a data-flow model. This is done with `vasy`, VHDL Analyzer for Synthesis⁶. This step will also add power rails to the model. The example below looks in the current directory for `and2.vhd`, adds power inputs, converts to alliance, outputs a file called `and2.vbe`, overwriting any existing `and2.vbe`, whilst being verbose, pause for breath. In an xterm enter ...

```
vasy -Vaop -I vhd and2
```

Your model of an and gate needs to be tested to ensure that it correctly performs the logical and operation on the inputs. A set of test vectors needs to be generated and applied to your model. Some VHDL tool-chains use a 'Test-bench', a wrapper program written in VHDL which drives the model under test. We will be driving our models directly through the use of a file, containing test vectors, known as a pattern file (`man 5 pat`). Open a new buffer in emacs with the file

⁵This 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>

and.in.pat and enter the following (continues on the next page)

```

in vdd B;
in vss B;
in x B;
in y B;
out z B;

begin
< 0 ns> and_test : 10 00 ?*;
< +1 ns> notxy   : 10 00 ?*;
< +1 ns> notx_y  : 10 01 ?*;
< +1 ns>         : 10 01 ?*;
< +1 ns> x_noty  : 10 10 ?*;
< +1 ns>         : 10 10 ?*;
< +1 ns> x_y     : 10 11 ?*;
< +1 ns>         : 10 11 ?*;
end;

```

The structure of a pattern file is

```

# a comment that will appear in the output file
-- a comment that won't.

-- The declarative part containing
-- a list of inputs and outputs to match the entity in the form
-- direction name format eg

-- an input called x of format bit looks like
in x B;

-- this is followed by the description block

-- keyword

begin

-- then a list of pattern description statements in the form

-- [date] [label] : list_of_values;

<0 ns> init: 11001FF ?***;
<+1 ns> run : 11000AB ?***;

-- keyword

end;

```

- The list of values is in the same order as the order of declaration, one value for each input.
- Output values can be specified explicitly, if a predicted output value does not match the actual output value during simulation then the simulator will halt with an error. Output values can also be represented by a wildcard '*' in which case the simulator will report the actual value.
- Valid formats for the values are **B**inary, **O**ctal and **heX**adecimal.
- labels should be meaningful.

3.1. Simulation

The simulator is called asimut (a simulation tool for hardware descriptions) and has many options, however the basic format will be one of the following

asimut -c filename

for testing the integrity of your **structural** vhdl file (.vst), no simulation

asimut -b filename in_pat_file out_pat_file

This will simulate a **dataflow** (.vbe) model using the named input pattern file and writing the results to the named output file.

asimut filename in_pat_file, out_pat_file

This will simulate a **structural** (.vst) model etc.

In this instance we require the second format as we are simulating a dataflow model. Run the simulation and view the results. Repeat the process with the other two gates that you have created.

Side-bar: The Alliance pattern files

Having saved the pattern file you can view the file as a waveform with the waveform viewer ... type xpat & on a command line.

Interlude

Another way to describe the logical operator **and**. Only the architecture is provided. The entity is the same as for the earlier and gate. The result after synthesis should be the same as the earlier architecture.

```

Architecture ex2 of and2 is

    signal xy : bit_vector(0 to 1);

begin
    xy <= x&y;
    with xy select
        z <= '1' when "11",
           '0' when others;
end ex2;

```

Create a new version of the and gate (.vhd) using this architecture and convert it to a data-flow model. Compare it with the earlier model, simulate it using the same input test pattern and a new output pattern. Are there any differences?

Coffee Break

4. Implementing more complex models

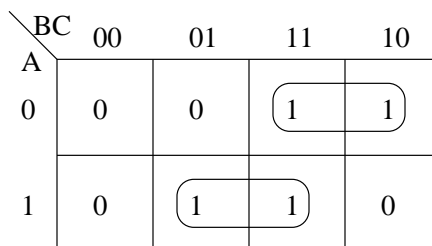
4.1. Design phase

Design a model to implement the following function, the truth table for which is

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.

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



A behavioural model of this function could be

entity comb_function is

```

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

```

end comb_function;

architecture behavioral of comb_function is
begin

```

  z <= (not a and b) or (a and c);
end behavioural;
```

4.2. Netlists

For this to be actually implemented in hardware it would either have to be a custom design or be reduced to a combination of standard logic gates. There are tools that will do it for us but we will start by doing it manually using the gates that we have already designed. We can keep the existing entity and create a new architecture.

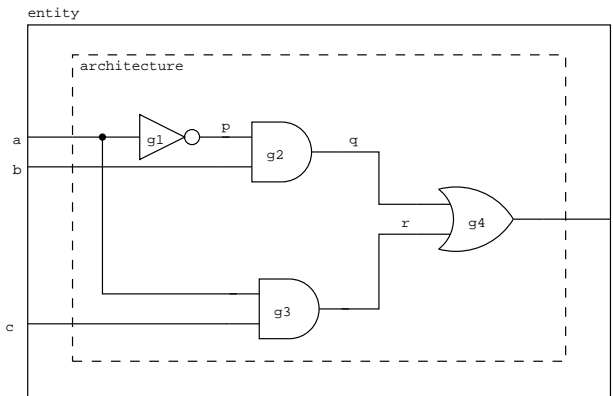
architecture netlist of comb_function is

```

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;
```



```

signal p, q, r :bit;
```

begin

```

g1 : not1 port map ( x => a, z => p);
g2 : and2 port map ( x => p, y => b, z => q);
g3 : and2 port map ( x => a, y => c, z => r);
g4 : or2  port map ( x => q, y => r, z => z);
```

end netlist;

We have specified the components that are needed for the design, the signals that will provide the connections between the components and the mapping of the component ports to the actual ports and signals of our design. If we wish, the components can be hidden in a **package**. There are many ways that a package can be used in VHDL, this exercise demonstrates just one use.

4.3. Packages 'mark 1'

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 our package. It should contain only the line

```
work.test.all : test_package7
```

This file is a tool specific file and not part of the VHDL standard. 'work' is a defined word and refers, by default to the current directory. The keyword 'all' refers to everything in the named library, very similar to java.

Finally, almost, write a new architecture for our model which will use our library of simple logic gates.

```
library work;    -- use current directory
use work.test.all; -- use the package test from current directory
```

```
entity comb_function is

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

end comb_function;

architecture direct of comb_function is

  signal p, q, r : bit;

begin -- netlist

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

⁷ The spaces either side of the ':' are required or you will get an obscure compile error :)

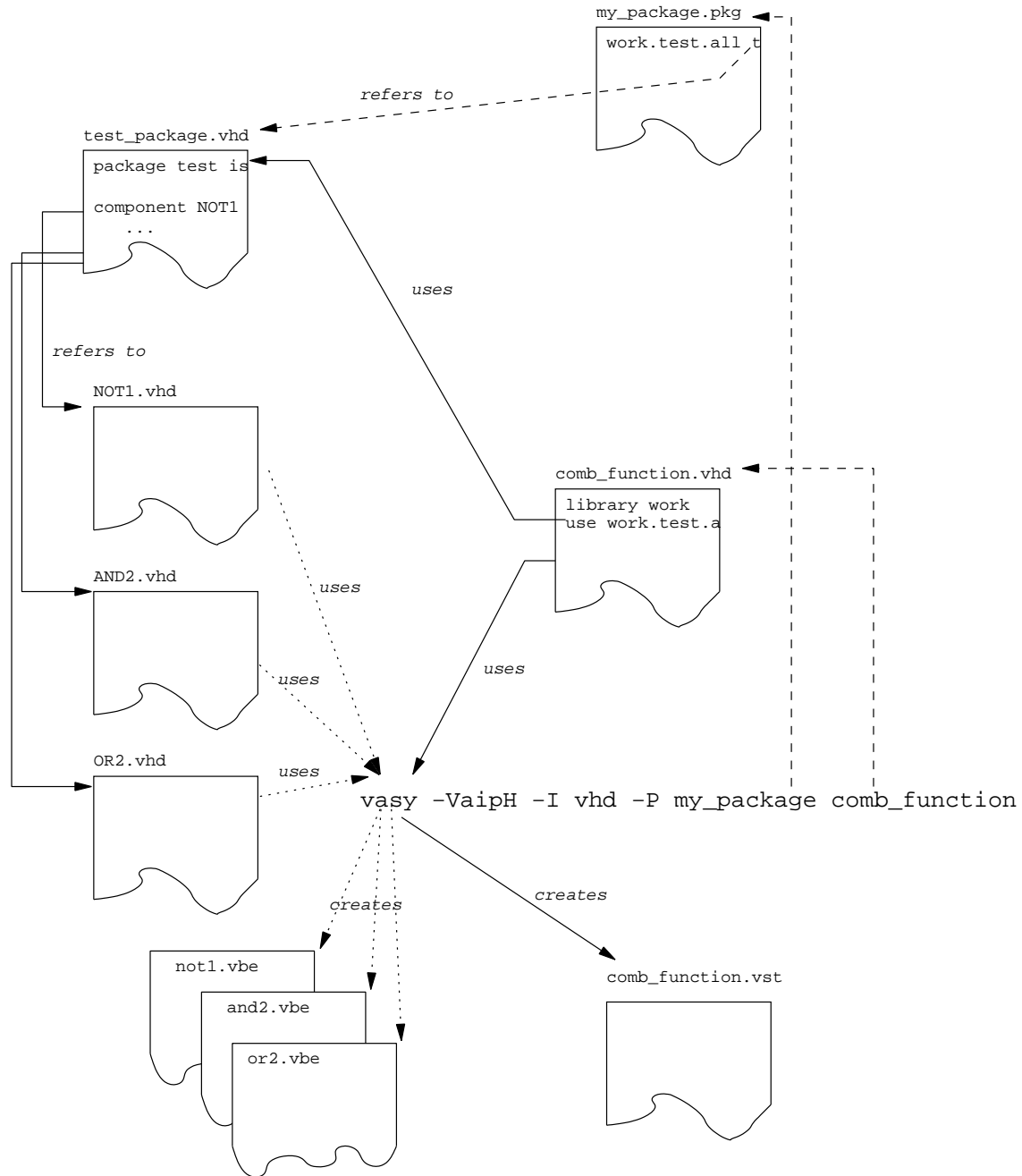
```
g3: And2 port map (a, c, r);
```

```
end direct;
```

Convert it to a form that can be synthesized using the command line

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

Write a test pattern for it and then simulate it. If you've got this far then congratulations.



Visual representation of the various files used and created when using the above command line