Structural VHDL

• Although we still work with schematic designs, the input to the synthesis tool must be a VHDL description of the structure of the design (i.e. what blocks are present and how they are interconnected)
  – This is termed a netlist
  – Will see an example of this in the lab

• VHDL permits this through a subset of the language commonly referred to as ‘structural VHDL’

• NB: it’s uncommon to have to write structural VHDL code, but a knowledge is essential!
Structured VHDL - II

- Example
  - The ALU schematic used in lab session 1
  - See ‘ALUSCH.VHD’
So what does the VHDL code look like?

```vhdl
architecture SCHEMATIC of DESIGN is

  Component Declarations

  Internal Signal Declarations

  Component Specification

  Component Instantiation

end SCHEMATIC;
```
Structural VHDL- IV

• Component declarations
  – Used to declare components within the structure
  – Must be a declaration for every component in the structure
  – Looks very similar to the entity definition

• E.g. for the stim_alu block

```vhdl
COMPONENT stim_alu
  PORT(
    stim_a : OUT std_logic_vector(3 DOWNTO 0);
    stim_b : OUT std_logic_vector(3 DOWNTO 0);
    stim_clock : OUT std_logic;
    stim_inst : OUT std_logic_vector(1 DOWNTO 0);
    stim_reset : OUT std_logic
  );
END COMPONENT;
```

Compare with entity
Structural VHDL - V

• Internal signal declarations
  – Use to declare internal signals within the architecture (i.e. the *nets*)

• E.g.

```vhdl
SIGNAL reset : std_logic;
SIGNAL clock : std_logic;
SIGNAL inst : std_logic_vector(1 DOWNTO 0);
SIGNAL a : std_logic_vector(3 DOWNTO 0);
```

These appear as labels on schematic

@a<3:0>
Structural VHDL- VI

• Component specification
  – Use to specify where the component architectures are coming from
  – i.e. the source library

  FOR ALL: alu USE ENTITY dd4lab1.alu(behavior);
  FOR ALL: stim_alu USE ENTITY dd4lab1.stim_alu(behavior);

  Specifies that all ‘stim_alu’ architectures comes from dd4lab1 library
Structural VHDL - VI

- Component instantiation
  - ‘Creates’ instances of the components within the design and how they are connected to other parts of the design

```
I1 : stim_alu
    PORT MAP (  
        stim_a(3 DOWNTO 0) => a(3 DOWNTO 0),  
        stim_b(3 DOWNTO 0) => b(3 DOWNTO 0),  
        stim_clock => clock,  
        stim_inst(1 DOWNTO 0) => inst(1 DOWNTO 0),  
        stim_reset => reset  
    );
```

Instance Label

\{ Specifies Mappings \}
Finally, once VHDL has been written, how is it verified?
- We create another block to test it using a testbench.

**Simulation & Verification**

- Stimulus Block
- Block under Test
Simulation & Verification- II

• Block Under Test
  – This is the code (or complete system) that we have produced that we wish to test

• Stimulus Block
  – This block generates test signals (or stimuli) to ‘excite’ the block under test
  – Written also in VHDL (can also use language features such as files etc. for complex test data generation)

• Testbench
  – Simply a top level schematic etc. which contains the block under test and the stimulus block
Simulation & Verification- III

• What exactly to we put in the stimulus block?
• We wish to test the functionality of the block that has been written to check that it matches the specification for the block
• Testing conducted in a ‘scientific’ manner
  – Not 95% writing code and 5% simulation!
  – A good approach to testing means that you can be 95% confident of the design working in Silicon
  – Consider all possible test cases
VHDL- Concluding Remarks

• Have introduced all the main points of writing VHDL code for synthesis

• Only a limited range of the language features are suitable for synthesis
  – Standards are currently being developed to define the actual range!

• The final output is only as good as the designer who wrote the code!
  – Just as a programmer can write inefficient ‘C’ code, a VHDL programmer can write bad VHDL code which can have drastic effects (e.g. 1000% more Flip-Flops than are really necessary)