COEN-2710 Microprocessors - Lecture 5

VHDL Refresher (Project 2)

Cristinel Ababei
Marquette University
Department of Electrical and Computer Engineering

1

Outline

- VHDL Overview
- Basic VHDL Modelling
  - Entity declaration
  - Architecture declaration
- Structural vs. Behavioural Description
- Combinational, Sequential
- Testbenches
- Resources

2

VHDL overview

- What does VHDL stand for?
  - Very High Speed Integrated Circuit (VHSIC) Hardware Description Language
- VHDL is a formal language for specifying the behavior and structure of a digital circuit
  - Concurrent and sequential statements
  - Machine-readable specification
  - Man- and machine-readable documentation
- Initially developed under DOD auspices, later standardized as IEEE standards 1076-1987, 1076-1993, & 1076-1164 (standard logic data type)
- A concurrent language, initially aimed at simulation, later at synthesis
- Syntax similar to ADA and Pascal
- Verilog is another, equally popular, hardware description language (HDL)

3

Basic VHDL Modeling: Design Unit

- A VHDL Design Entity or Unit always consists of:
  1) **Entity declaration:** Names entity and defines interfaces between entity and its environment.
  2) **Architecture:** Establishes relationship between inputs and outputs of design

```
ENTITY entity_name IS
        PORT ( name_list : mode type);
END entity_name;
```

```
ARCHITECTURE body_name OF entity_name IS
        -- declarative_statements
BEGIN
        -- activity_statements
END body_name;
```

4

1) Entity Declaration

- Names entity and defines interfaces between entity and its environment.

```
entity entity-name is port (
    port-name-A:  mode  type;
    port-name-B:  mode  type;
    port-name-C:  mode  type;
    …
    );
end [entity][entity-name];
```

5

Port

- Each I/O signal in the entity statement is referred to as a **port**.
- A port is analogous to a pin on a schematic.
- A port is a data object.
- Can be assigned values.
- Can be used in expressions.

6

Mode

- The **mode** describes the direction in which data is transferred through a port.
- There are 4 different modes:

Mode	Description
in	Data only flows into the entity (input)
out	Data only flows out of the entity (output)
inout	Data flows into or out of the entity (bidirectional)
buffer	Used for internal feedback

7

Type

- VHDL is a strongly **typed language**
  - Data objects of different types cannot be assigned to one another without the use of a type-conversion function.
- There are two broad categories of data types:
  - Scalar - stores a single value
  - Composite - stores multiple values
- VHDL data types include:

scalar	bit
	boolean
	integer
	character
	std_ulogic
	std_logic
composite	bit_vector
	string
	std_ulogic_vecto
	std_logic_vector

8

2) Architecture Declaration

- Establishes relationship between inputs and outputs of design.

```
architecture architecture-name of entity-name is
    [declarations]
begin
    architecture body
end [architecture][architecture-name];
```

9

Architecture body

- Several different models or styles may be used in the architecture body including:
  - Behavioral/Functional
    - Dataflow
    - Algorithmic
  - Structural
- These models allow to describe the design at different levels of abstraction.

10

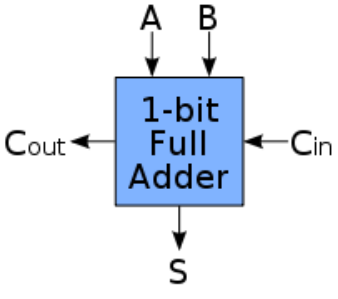
Architecture statement

- One or more architecture statements may be associated with an entity statement.
  - Only one may be referenced at a time.
- Declarations
  - Signals and components.
- Architecture body
  - Statements that describe the functionality of the design (i.e., the circuit).

11

Example 1: Entity Declaration

```
entity FULL_ADDER is
   port (
         A, B, Cin: in std_logic;
         S:         out std_logic;
         Cout:      out std_logic);
end FULL_ADDER;
```

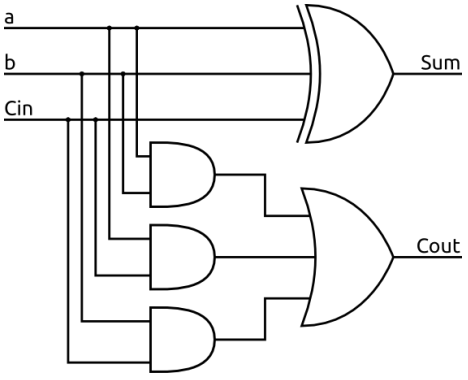


12

Example 1: Architecture Declaration

```
architecture My_Structural of FULL_ADDER is

begin
    S <= A xor B xor Cin;
    Cout <= (A and B) or (A and Cin) or (B and Cin);
end My_Structural;
```

C_{in}	A	B	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



13

Outline

- VHDL Overview
- Basic VHDL Modelling
  - Entity declaration
  - Architecture declaration
- Structural vs. Behavioural Description
- Combinational, Sequential
- Testbenches
- Resources

14

1) Structural Description in VHDL

- *Structural model*: describe how it is composed of subsystems, based on a precise knowledge of the internal structure.
  - Component declaration and instantiation
- A *structural architecture* describes the schematic by defining the interconnection of *components*
- Simplest components: associated with design entities describing AND, OR, etc. switching algebra operations; logic gates basically
- Use component statement in structural descriptions

15

Component Declaration Format

The following is the FORMAT for declaring components:

```
COMPONENT component_name
    PORT ( clause ) ;
END COMPONENT;
```

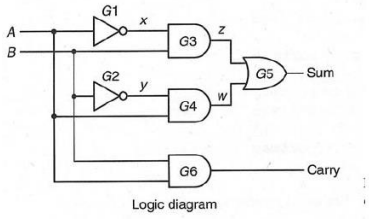
Note the similarity between component declaration statement and entity declaration statement. Both have a header, port clause, and end statement.

This similarity is not coincidental. Components are virtual design entities.

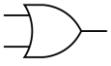
16

Example 2: Design entities used as components in top-level entity

```
library IEEE ;
use IEEE.std_logic_1164.all;

entity HALFADDER is
        port ( A, B : in std_logic;
                SUM, CARRY : out std_logic);
end HALFADDER;

architecture my_arch of HALFADDER is
begin
        SUM <= (not A and B) or (A and not B);
        CARRY <= A and B;
end my_arch;

library IEEE ;
use IEEE.std_logic_1164.all;

entity ORGATE is
        port (A, B : in std_logic;
                RES : out std_logic);
end component;

architecture behavioral of ORGATE is
begin
        RES <= A or B;
end behavioral;
```



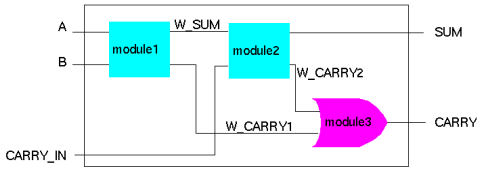
Logic diagram



17

Example 2: Top-level 1-bit FULLADDER - Component Declarations

```
entity FULLADDER is
   port (A,B, CARRY_IN: in std_logic;
          SUM, CARRY:  out std_logic);
end FULLADDER;

architecture STRUCT of FULLADDER is

 -- component declarations go here!
 component  HALFADDER
   port (A, B : in std_logic;
         SUM, CARRY : out std_logic);
 end component;

 component  ORGATE
    port (A, B : in std_logic;
          RES : out std_logic);
 end component;

 signal W_SUM, W_CARRY1, W_CARRY2 : std_logic;

begin
 -- component instantiations go here!
end STRUCT;
```

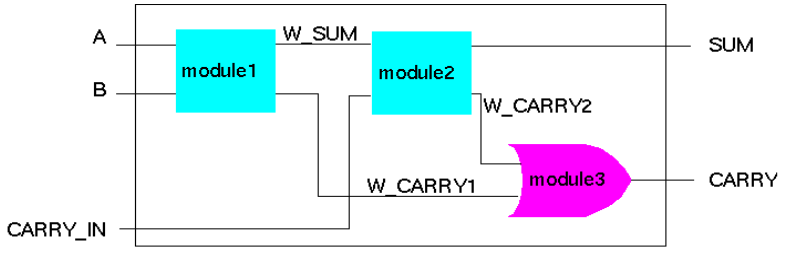


Full adder: 2 halfadders + 1 OR-gate

- In a component declaration, all module types, which will be used in the architecture, are declared.
- Their declaration must occur before the **begin** keyword of the architecture statement.
- The port list elements of the component are called **local elements,** they are **not signals**

18

Component Instantiation → Hierarchy!



Full adder: 2 halfadders + 1 OR-gate

- A **module** can be assembled out of several submodules → hierarchical model description
- A purely **structural architecture** does not describe any functionality and contains just a list of components, their instantiation and their interconnections

19

Example 2: Component Instantiation

```
architecture STRUCT of FULLADDER is

component HALFADDER
   port (A, B : in std_logic;
         SUM, CARRY : out std_logic);
end component;

component ORGATE
   port (A, B : in std_logic;
         RES : out std_logic);
end component;

signal W_SUM, W_CARRY1, W_CARRY2: std_logic;

begin

  MODULE1:  HALFADDER  port map (A, B, W_SUM, W_CARRY1);
  MODULE2:  HALFADDER  port map (W_SUM, CARRY_IN, SUM, W_CARRY2);
  MODULE3:  ORGATE     port map (W_CARRY2, W_CARRY1, CARRY);

end STRUCT;
```

- **Component instantiations** occur in the statements part of an architecture (after the keyword "begin").
- The choice of components is restricted to those that are already declared, either in the declarative part of the architecture or in a **package**.
- The connection of signals to the entity port: positional association, the first signal of the port map is connected to the first port from the component declaration.

20

Component Instantiation: Named Signal Association

```
entity FULLADDER is
  port (A,B, CARRY_IN: in bit;
        SUM, CARRY: out bit);
end FULLADDER;

architecture STRUCT of FULLADDER is

  component HALFADDER
     port (A, B : in bit;
           SUM, CARRY : out bit);
  end component;
  ...
  signal W_SUM, W_CARRY1, W_CARRY2 : bit;

begin

   MODULE1: HALFADDER
        port map ( A     => A,
                   SUM   => W_SUM,
                   B     => B,
                   CARRY => W_CARRY1 );
   ...
end STRUCT;
```

- Named association:
  - left side: "formals" (port names from component declaration)
  - right side: "actuals" (architecture signals)
  - Independent of order in component declaration

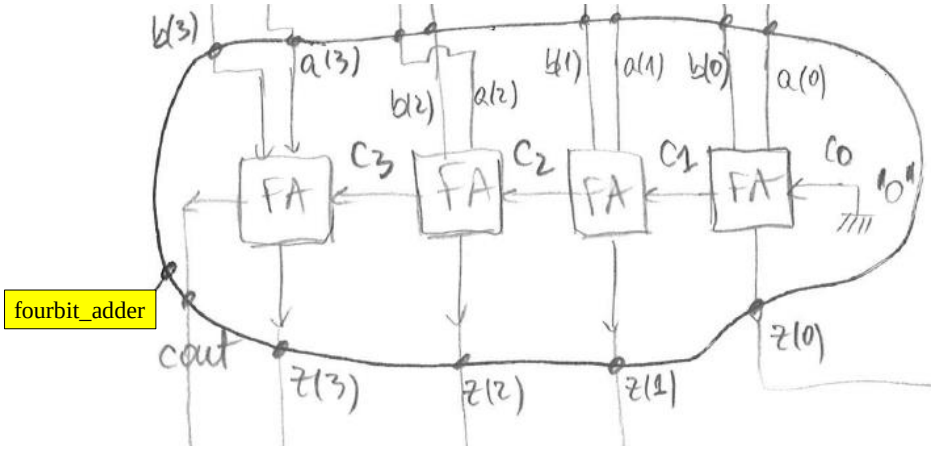
21

Example 3: Structural fourbit_adder design entity

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fourbit_adder is
       port( a, b: in STD_LOGIC_VECTOR(3 downto 0);
             z: out STD_LOGIC_VECTOR(3 downto 0);
             cout: out STD_LOGIC);
end fourbit_adder;

architecture MY_STRUCTURE of fourbit_adder is

    component FULL_ADDER
       port( a, b, carry_in: in  STD_LOGIC;
             sum, carry: out STD_LOGIC );
    end component;

    signal c0, c1, c2, c3 : STD_LOGIC;
begin

    c0 <= '0';
    b_adder0: FULL_ADDER port map (a(0), b(0), c0, z(0), c1);
    b_adder1: FULL_ADDER port map (a(1), b(1), c1, z(1), c2);
    b_adder2: FULL_ADDER port map (a(2), b(2), c2, z(2), c3);
    b_adder3: FULL_ADDER port map (a(3), b(3), c3, z(3), cout);

end MY_STRUCTURE;
```

22

Example 3: fourbit_adder



23

2) Behavioral Description in VHDL

- Specify a set of statements to model the function, or behavior, of the design.
- Dataflow: uses concurrent statements
  - Concurrent statements:
    - Are executed at the same time; they mimic the actual hardware parallelism (processes, signal assignment)
    - Order is unimportant
- Algorithmic: uses sequential statements
  - Sequential statements:
    - Are executed in sequence (if, case, loops – while, for – assertion)
    - Order is very important

24

Behavioral synthesis

- Advantages
  - Easy to write HDL code; fewer lines of VHDL code
  - Useful especially for automatic generation of state machines
  - Faster simulation than RTL
- Disadvantages
  - May not be synthesizable

25

Example 4: Behavioral fourbit_adder

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity add is
  generic (
    width : positive := 4);
  port (
    in1   : in  std_logic_vector(width-1 downto 0);
    in2   : in  std_logic_vector(width-1 downto 0);
    sum   : out std_logic_vector(width-1 downto 0);
    carry : out std_logic);
end add;

architecture BEHAVIORAL of add is
  signal temp : unsigned(width downto 0);

begin
  temp  <= resize(unsigned(in1), width+1)+resize(unsigned(in2),
width+1);
  sum   <= std_logic_vector(temp(width-1 downto 0));
  carry <= temp(width);
end BEHAVIORAL;
```

26

Outline

- VHDL Overview
- Basic VHDL Modelling
  - Entity declaration
  - Architecture declaration
- Structural vs. Behavioural Description
- Combinational, Sequential
- Testbenches
- Resources

27

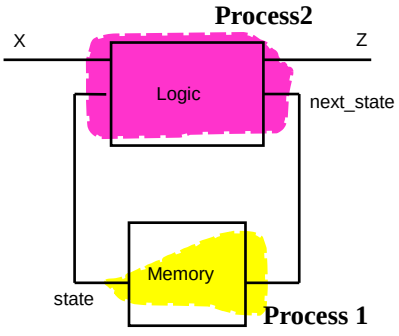
Combinational vs. Sequential Circuits

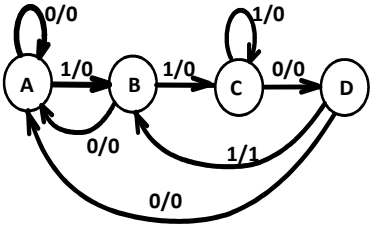
- Combinational circuits
  - See `fourbit_adder` example on previous slides
  - Browse resources provided on the last slide of this presentation
- Sequential circuits
  - Adopt the "Two process VHDL coding style" described here:
    - http://dejazzer.com/eece4740/lectures/lec02_edge_detector_mealy_moore.pdf
    - http://dejazzer.com/eece4740/lectures/lec03_d_sequential_2.pdf
  - Browse resources provided on the last slide of this presentation

28

Example 5: VHDL coding style
Use “two-process” approach for FSMs

Sequential Recognizer Circuit
What “pattern” does it recognize?





29

Example 5: VHDL coding style
Use “two-process” approach for FSMs

```
library ieee;
use ieee.std_logic_1164.all;

entity seq_rec_MEALY is
 port (CLK, RESET, X: in std_logic;
       Z: out std_logic);
end seq_rec;

architecture process_2 of seq_rec_MEALY is
  type state_type is (A, B, C, D);
  signal state, next_state: state_type;

begin
-- continue on next slide…
```

30

```
-- process 1: implements positive edge-triggered
-- flip-flop with asynchronous reset

state_register: process (CLK, RESET)
begin
 if (RESET = '1') then
  state <= A;
 elsif (CLK'event and CLK = '1') then
  state <= next_state;
 end if;
end process;

-- continue on next slide…
```

31

```
-- process 2: Z and next_state implemented
-- as functions of input X and state
X_and_next_state_functions: process (X, state)
begin
 case state is
   when A =>
     Z <= '0';
     if X = '1' then next_state <= B;
     else next_state <= A;
     end if;
   when B =>
     Z <= '0';
     if X = '1' then next_state <= C;
     else next_state <= A;
     end if;
   when C =>
     Z <= '0';
     if X = '1' then next_state <= C;
     else next_state <= D;
     end if;
   when D =>
     if X = '1' then Z <= '1'; next_state <= B;
     else Z <= '0'; next_state <= A;
     end if;
 end case;
end process;
end architecture;
```

32

Outline

- VHDL Overview
- Basic VHDL Modelling
  - Entity declaration
  - Architecture declaration
- Structural vs. Behavioural Description
- Combinational, Sequential
- Testbenches
- Resources

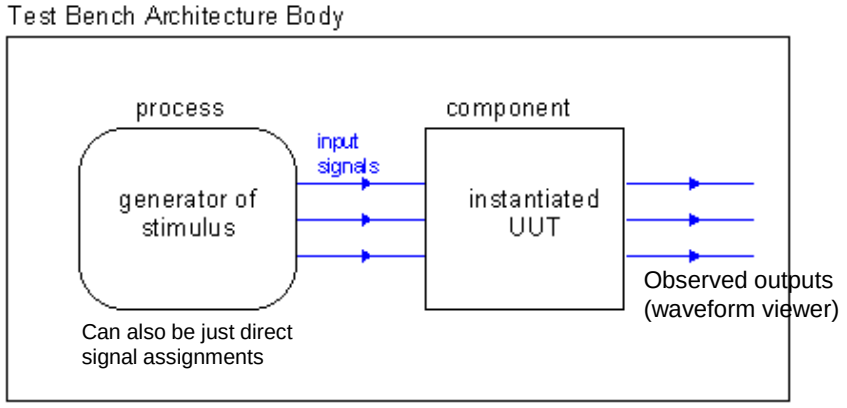
33

Testbenches

- Used to verify the specified functionality of a design
  - Provides the stimuli (test vectors) for the Unit Under Test (UUT) or Design Under Test (DUT), analyzes the UUT's response or stores the values in a file.
  - Simulation tools visualize signals by means of a waveform which the designer compares with the expected response. Debug if does not match.
- Does not need to be synthesizable
- No ports to the outside, self-contained

34

Testbench Concept



Test Bench design entity has no I/O ports!

35

Testbenches

- Simple testbench responses can be analyzed by waveform inspection
- Sophisticated testbenches may require more complicated verification techniques
  - Can take >50% of project resources
  - Do not underestimate the value/importance of testbenches!

36

Structure of a VHDL Testbench

```
entity TB_TEST is
end TB_TEST;

architecture BEH of TB_TEST is
  -- component declaration of UUT
  -- internal signal definition
begin
  -- component instantiation of UUT

  -- clock and stimuli generation
  wait for 100 ns;
  A <= 0;
  CLK <= 1;
  …
end BEH;

configuration CFG1 of TB_TEST is
   for BEH;
       -- customized configuration
   end for;
end CFG_TB_TEST;
```

- Declaration of the **Unit Under Test (UUT)**
- Connection of the UUT with testbench signals
- Stimuli and clock generation (behavioral modeling)
- Response analysis
- A **configuration** may be used to pick the desired components for simulation
  - May be a customized configuration for testbench simulation

37

Example 6: Simple Testbench

```
library ieee;
use ieee.std_logic_1164.all;

entity ADDER is
   port (A,B : in bit;
      CARRY,SUM : out bit);
end ADDER;


architecture RTL of ADDER is
begin
  ADD: process (A,B)
  begin
     SUM <= A xor B;
     CARRY <= A and B;
  end process ADD;
end RTL;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity TB_ADDER IS -- empty entity is defined
end TB_ADDER;      -- no need for interface

architecture TEST of TB_ADDER is
   component ADDER
     port (A, B: in bit;
           CARRY, SUM: out bit);
   end component;
   signal A_I, B_I, CARRY_I, SUM_I : bit;
begin
  UUT: ADDER port map(A_I, B_I, CARRY_I, SUM_I);

  STIMULUS: process
    begin
       A_I <= '0'; B_I <= '0'; wait for 10 ns;
       A_I <= '1'; B_I <= '1'; wait for 10 ns;
       A_I <= '1'; B_I <= '0'; wait for 10 ns;
       A_I <= '1'; B_I <= '1'; wait for 10 ns;
       wait;
      -- and so on…
  end process STIMULUS;
end TEST;
```

38

Resources

- Read description of each part of Project #2, provided on D2L
- Read materials suggested in the steps for Project #2 on the website of this class:
  - http://dejazzer.com/coen2710/index.html
- (Optional) Read slides and lecture notes of this class (focuses on VHDL design and FPGAs):
  - http://dejazzer.com/eece4740/index.html

39