ECE 3401 Lecture 5

Basic VHDL Modeling

VHDL Structural Elements

- **Entity**: description of *interface* consisting of the port list.
  - The primary hardware abstraction in VHDL, analogous to a symbol in a block diagram.
- **Architecture**: description of the *function* of the corresponding module.
- **Process**: allows for a *sequential* execution of the assignments
- **Configuration**: used for simulation purposes.
- **Package**: holds the definition of commonly used data types, constants and subprograms.
- **Library**: the logical name of a collection of compiled VHDL units (object code).
  - Mapped by the simulation or synthesis tools.
Data Objects

- Data objects hold a value of specified type. They belong to one of three classes:
  - Constants
  - Signals
  - Variables

- **Constants** and **variables** are typically used to *model the behavior of the circuit*.

- **Signals** are typically used to *model wires and flip-flops*

- Must be declared before they are used

Constants

- A constant holds a value that cannot be changed within the design description.
- **Constant** must be declared in *Entity, Architecture, Process, Package*.
  - A constant defined in a *package* can be referenced by any entity or architecture for which the package is used.
  - **Local Property**: A constant declared in an *entity/architecture/process* is visible only within the local environment

- **Example**:
  - `constant RISE_TIME: TIME := 10 ns;`  -- declares a constant RISE_TIME of type TIME, with a value of 10 ns
  - `constant BUS_WIDTH: INTEGER := 8;`  -- declares a constant BUS_WIDTH of type INTEGER with a value of 8.
Constants

ENTITY example IS

    CONSTANT width : integer := 8;

    PORT ( input : IN bit_vector (width-1 DOWNTO 0);
           output: OUT bit_vector (width-1 DOWNTO 0);

END example;

- The above constant represents the width of a register.
- The identifier \texttt{width} is used at several points in the code. To change the width requires only that the constant declaration be changed and the code recompiled.

Signals

- \textbf{Signal} represents the logic signals or wires in a circuit. Signals can also represent the state of a memory

- There are three places in which signals can be declared in a VHDL code
  - Entity declaration
  - Declarative part of an architecture
  - Declarative part of a package.
Signals

- A signal has to be declared with an associated type:
  - SIGNAL signal_name : type_name;
- The signal’s type_name determines the legal values that the signal can have and its legal use in VHDL code.

**Signal types:**
- (1) bit  
- (2) bit_vector  
- (3) std_logic  
- (4) std_logic_vector  
- (5) std_ulogic  
- (6) signed  
- (7) unsigned  
- (8) integer  
- (9) enumeration  
- (10) boolean

---

**Signal – Example (1)**

- SIGNAL  Ain :  BIT_VECTOR (1 TO 4);
- Note:
  - The syntax “lowest_index TO highest_index” is useful for a multi-bit signal that is simply an array of bits.
  - In the signal Ain, the most-significant (left-most) bit is referenced using lowest_index, and the least-significant (right-most) bit referenced using highest_index.

- Example:
  - The signal "Ain" comprises 4 bit objects.
  - The assignment statement  Ain <= "1010"
  - Results in  Ain(1) = 1, Ain(2) = 0, Ain(3) = 1, Ain(4) = 0
Signals – Example (2)

- SIGNAL Byte: BIT_VECTOR (7 downto 0);
- Note:
  - The signal "Byte" comprises eight bit objects.
  - The assignment statement:
    Byte <= "10011000";
    Result in Byte(7)=1, Byte(6)=0, Byte(5)=0,
    Byte(4)=1, Byte(3)=1, Byte(2)=0, Byte(1)=0,
    Byte(0)=0

Signals

- STD_LOGIC & STD_LOGIC_VECTOR types
  - STD_LOGIC type provides more flexibility than
    the BIT type.
  - To use this type, we must include these
    statements in VHDL files.
    - LIBRARY IEEE;
    - USE IEEE.STD_LOGIC_1164.ALL;
  - Note: These statements provide the access to
    the std_logic_1164 package, which defines the
    STD_LOGIC type.
IEEE Standard Logic Type

```plaintext
type STD_ULOGIC is (
  'U', -- uninitialized
  'X', -- strong 0 or 1 (= unknown)
  '0', -- strong 0
  '1', -- strong 1
  'Z', -- high impedance
  'W', -- weak 0 or 1 (= unknown)
  'L', -- weak 0
  'H', -- weak 1
  '-', -- don’t care);
```

- 9 different signal states
- The new type is implemented as enumerated type by extending the existing '0' and '1' symbols with additional ASCII characters.
- Superior simulation results
- Bus modelling
- The 'u' symbol is the leftmost symbol of the declaration, i.e. it will be used as initial value during simulation.

- Defined in package 'IEEE.std_logic_1164'
- Similar data type 'std_logic' with the same values
- Array types available: 'std_(u)logic_vector', similar to 'bit_vector'

Variables

- A **variable**, unlike a SIGNAL, does not necessarily represent a wire in a circuit.

- Variables can be **used in sequential areas only**
  - The scope of a variable is the process or the subprogram.
  - A variable in a subprogram does not retain its value between calls.

- **Variable assignment is immediate, not scheduled.**
Logic Operators

- and, or, xor, xnor, nand, nor, not

- Example:
  - \( Z \leq (\text{not } A \text{ and } B) \text{ or } (A \text{ and } \text{not } B); \)

Concurrency

- VHDL concurrent statements execute in a *concurrent* fashion, i.e., statements execute only when associated signals change value.
- There is no master, procedural flow of control; each concurrent statement execute in a *nonprocedural stimulus/response*.

```vhdl
ENTITY example1 IS
    PORT (x1, x2, x3 : IN BIT;
          f : OUT BIT);
END example1;

ARCHITECTURE logicFunc OF example1 IS
    SIGNAL a1, b1: BIT;
BEGIN
    -- Concurrent signal assignment statements
    a1 <= x1 AND x2;
    b1 <= NOT x2 AND x3;
    f <= a1 NOR b1;
END logicFunc;
```
VHDL Model

- **Structural model**: describe how it is composed of subsystems
  - Component declaration and instantiation

- Behavioral model: describe the function of entity in an abstract way

Structural Modeling

- The **STRUCTURE** architecture describes the schematic for the majority logic by defining the interconnection of components

- Components are associated with design entities describing the AND and OR switching algebra operations.

- Use **component statement** in structural statement description.
Hierarchical Model Layout

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

Component Declaration

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

architecture STRUCT of FULLADDER is
signal W_SUM, W_CARRY1, W_CARRY2 : bit;

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

component ORGATE
  port (A, B: in bit;
       RES: out bit);
end component;
begin
  . . .
```

- In a component declaration, all module types, which will be used in the architecture, are declared.
- This declaration has to occur before the 'begin' keyword of the architecture statement.
- The port list elements of the component are called local elements, they are not signals.
Component Declaration Format

-- The following is the FORMAT for declaring components.

COMPONENT component_name
    PORT ( clause ) ;
END COMPONENT;

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. Component to entity association is provided by a configuration.

Component Instantiation

```
architecture STRUCT of FULLADDER is
    component HALFADDER
        port (A, B : in bit;
            SUM, CARRY : out bit);
    end component;
    component ORGATE
        port (A, B : in bit;
            RES : out bit);
    end component;
    signal W_SUM, W_CARRY1, W_CARRY2: bit;
begin -- statement part
    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 statement 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:
  - Default: positional association, the first signal of the port map is connected to the first port from the component declaration.
Component Instantiation: Named Signal Association

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

```vhdl
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;
```

Syntax of the Component Instantiation Statement

- **Label**: component_name
  - [GENERIC MAP (association_list)]
  - [PORT MAP (association_list)]

The label and component name are the only required elements of the component instantiation statement.

- **GENERIC MAP** is optional if there are no generics declared within the entity declaration of the instantiated component or there is no need to override the declared generic.

- **PORT MAP** describes how each component instance is connected to the rest of the system.
Example 1:

Write a VHDL description for the circuit in below figure. Use component instantiation statement and an internal signal $x$.

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration begins
ENTITY example2_6 IS
  PORT (a, b, c : IN STD_LOGIC;
         d : OUT STD_LOGIC);
END example2_6;

ARCHITECTURE arch2_6 OF example2_6 IS
  COMPONENT and1
    PORT (p, q : IN STD_LOGIC;
       r : OUT STD_LOGIC);
  END COMPONENT;

  COMPONENT or1
    PORT (p, q : IN STD_LOGIC;
       r : OUT STD_LOGIC);
  END COMPONENT;

  -- Declare signals to interconnect logic operators
  SIGNAL x : BIT;

BEGIN
  U1A : and1 PORT MAP (a, b, x);
  U2A : or1 PORT MAP (x, c, d);
END arch2_6;
```

Entity and1 IS

```vhdl
PORT (p, q : IN STD_LOGIC;
       r : OUT STD_LOGIC);
END and1;
```

ARCHITECTURE arch OF and1 IS

```vhdl
BEGIN
  r <= p AND q;
END arch;
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration begins
ENTITY example2_6 IS
  PORT (a, b, c : IN STD_LOGIC;
        d : OUT STD_LOGIC);
END example2_6;

-- entity declaration ends. The PORT clause declares signals (a, b, c, d) that interfaces the module to the outside world.

ARCHITECTURE arch2_6 OF example2_6 IS

  -- component declaration portion of architecture.
  -- Before a component is instantiated in a logic circuit, it must first be declared.
  -- The two components to be declared are an AND and OR gate given the names "and1" and "or1" respectively.

  COMPONENT and1
    PORT (p, q : IN STD_LOGIC;
          r : OUT STD_LOGIC);
  END COMPONENT;

  COMPONENT or1
    PORT (p, q : IN STD_LOGIC;
          r : OUT STD_LOGIC);
  END COMPONENT;
-- signal declaration portion of architecture.
-- Declare signals to interconnect logic operators
-- The circuit has an internal signal with the name \( x \) which is used by both components, this signal should also be declared prior to its usage in the architecture body.

\[
\text{SIGNAL } x : \text{BIT};
\]

-- Component Instantiation portion of architecture
-- The component instantiation statement connect logic operators to describe schematic.

\[
\text{BEGIN}
\begin{align*}
\text{U1A : and1 PORT MAP ( a, b, x );}
\text{U2A : or1 PORT MAP ( x, c, d );}
\end{align*}
\text{END arch2_6;}
\]

Example 2:
Write a VHDL code for the circuit shown below. The inputs to the circuit is \( x_1, x_2, x_3 \), and the output is \( f \).
Example 2: VHDL code - Behavior

```vhdl
entity example1 is
  port (x1, x2, x3: in bit;
        f: out bit);
end example1;

architecture behavior of example1 is
begin    -- Architecture statement region
  f <= (x1 and x2) nor (not x2 and x3);
end behavior;
```

Example 2: VHDL code - Structure

```vhdl
entity example1 is
  port (x1, x2, x3: in bit;
        f: out bit);
end example1;

architecture structure of example1 is
component and is
  port (a, b: in bit; f: out bit);
end component and;

component nor is
  port (a,b: in bit; f: out bit);
end component nor;

component inv is
  port (a: in bit; f: out bit);
end component inv;

signal x2bar,u1aout,u5aout: bit;
begin
  u1a: component and port map(x1,x2,u1aout)
  u2a: component inv port map(x2,x2bar)
  u3a: component nor port map(u1aout,u5aout,f)
  u5a: component and port map(x2bar,x3,u5aout);
end structure;
```
Example 3: Multiply-Accumulator

A two-input multiply accumulator device: multiply two 16-byte data.
Internal signals are: reg_in1, reg_in2, mul, reg_mul, adder, accum. Two serial-to-parallel registers u1 and u2 convert the input bit stream of the data into a 16-wide data. The multiplier u3 sends its output to the adder u5 via register u4.

These components are found in the package **ECE_252_package** which is found in the library **WORK**.

The **USE** statement makes all the components in this package visible to our design.

```vhdl
LIBRARY WORK;
USE WORK.ECE_252_PACKAGE.ALL;
ENTITY mac IS
    GENERIC (tco : time := 10 ns);
    PORT (in1, in2 : IN BIT_VECTOR (15 DOWNTO 0);
          clk    : IN BIT;
          output : OUT BIT_VECTOR(31 DOWNTO 0);
    END mac;
```
ARCHITECTURE structure OF mac IS
   -- component declaration part, components include a register called reg,
   -- an adder called adder, a multiplier called multiply, a buffer called buf
   COMPONENT reg
      GENERIC (width : integer := 16);
      PORT (d : IN BIT_VECTOR (width-1 DOWNTO 0);
            clk : IN BIT;
            q : OUT BIT_VECTOR (width-1 DOWNTO 0));
   END COMPONENT;

   COMPONENT adder
      PORT (port1, port2 : IN BIT_VECTOR (31 DOWNTO 0);
            output : OUT BIT_VECTOR (31 DOWNTO 0));
   END COMPONENT;

   COMPONENT multiply
      PORT (port1, port2 : IN BIT_VECTOR (15 DOWNTO 0);
            output : OUT BIT_VECTOR (31 DOWNTO 0));
   END COMPONENT;

   COMPONENT buf
      PORT (input : IN BIT_VECTOR (31 DOWNTO 0);
            output : OUT BIT_VECTOR (31 DOWNTO 0));
   END COMPONENT;

   -- signal declaration portion of architecture
   SIGNAL reg_in1, reg_in2 : BIT_VECTOR (15 DOWNTO 0);
   SIGNAL mul, reg_mul, adder, accum : BIT_VECTOR (31 DOWNTO 0);

   -- Component instantiation and logic interconnection
   BEGIN
      u1: reg GENERIC MAP(16) PORT MAP (in1, clk, reg_in1);
      u2: reg GENERIC MAP(16) PORT MAP (in2, clk, reg_in2);
      u3: multiply PORT MAP (reg_in1, reg_in2, mul);
      u4: reg GENERIC MAP(32) PORT MAP (mul, clk, reg_mul);
      u5: adder PORT MAP (reg_mul, accum, adder);
      u6: reg GENERIC MAP(32) PORT MAP (adder, clk, accum);
      u7: buf PORT MAP (accum, output)
   END structure;