1- Introduction

2- Basic Structures of VHDL

3- Combinational Circuits

4- Sequential Circuits

5- Memory

6- Writing Testbenches

7- Synthesis Issues

8- RTL Cores

2- Basic Structures of VHDL

- **VHDL as an HDL**
- **A VHDL Description**
- **A VHDL Description – Enumeration Type**
- **Signals and Variables**

VHDL as an HDL - What is VHDL?

- VHDL is a hardware description language for simulation, for modeling, automatic hardware generation and testing
- VHDL became *IEEE standard 1076* in December 1987
- VHDL is a hardware description language
  - You don’t program in VHDL, you describe hardware
- VHDL is not a software language

- C++ is a software language
- You program in software languages
- You don’t describe hardware

VHDL as an HDL - VHDL vs. C++

VHDL

1. Concurrency
2. Timing
   - Wall Clock
   - Real Time
   - Delta Time

C++

1. Timing
   - Wall Clock

- Hardware components are *concurrent*
- *Wall Clock* is the actual time taken from the start of a simulation to the end
- Activities are scheduled in terms of *real time*
- *Delta Time* is VHDL's way of fooling you into thinking that it is performing its processes concurrently

VHDL as an HDL - Execution

VHDL

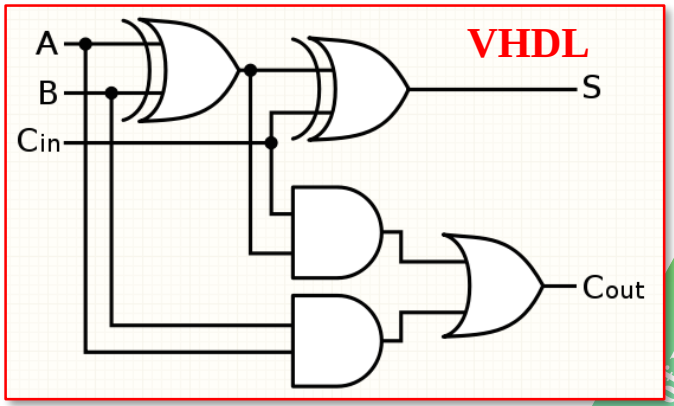
- In hardware, process execution depends on events on right hand side
- An event causes all processes to begin (Running Race)

C++

- In software, expressions are executed in the same order in which they appear in a program (Relay Race)

VHDL as an HDL - Concurrency

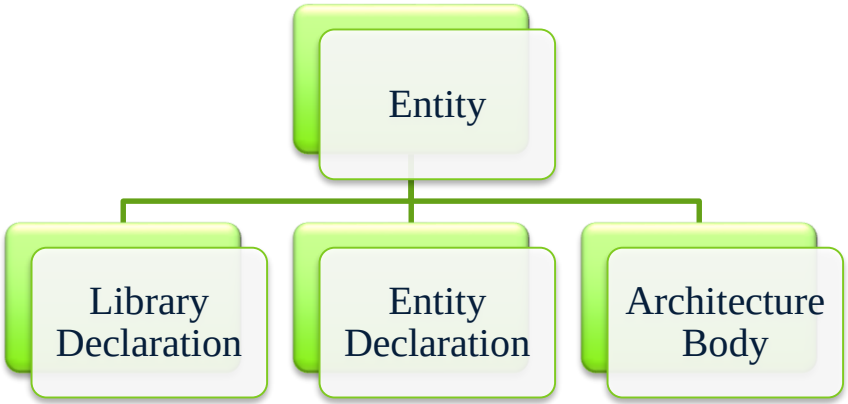
- A description has **signals** as ports
- A description is composed of concurrent ***processes***
- ***Processes*** communicate via **signals**
- New **signals** declared as needed
- Timing (Real-time, Delta-time) is associated with signals
- ***Processes*** wake up when an event occurs on any of their input signals, i.e., **sensitivity list**
- The **order** in which ***processes*** appear in a description is **not important**
- A ***process*** can be an **instance** of another component, a **signal assignment**, a **process statement**, . . .
- A ***process*** if it is a **process statement**, it can have local variables
- No timing associated with variables



A VHDL Description, a Hardware Entity

A VHDL component description:

- Entity declaration for interface declaration
- Architecture for functionality specification



A VHDL Description, a Hardware Entity

VHDL Description:

Entity

- Library Declaration
- Entity Declaration
- Architecture Body1
- Architecture Body2

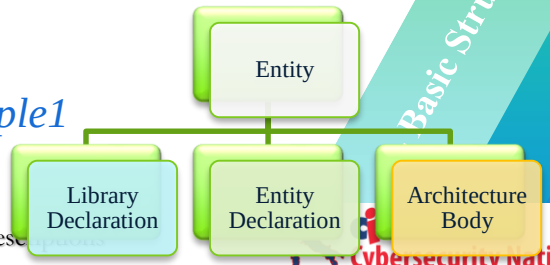
Commonly Used Packages

Define input/output

The Processing

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  ENTITY entity1 IS PORT (i1, i2 : IN BIT; w1 : OUT BIT);
6  END ENTITY entity1;
7
8  ARCHITECTURE simple1 OF entity1 IS
9     SIGNAL s1 : BIT;
10 BEGIN
11    statement1;
12    statement2;
13    statement3;
14 END ARCHITECTURE simple1;
```

- Using standard libraries
- A simple entity, *entity1*
- A simple architecture, *simple1* belongs to *entity1*



Basic Structures

A VHDL Description, a Hardware Entity

Library Declaration

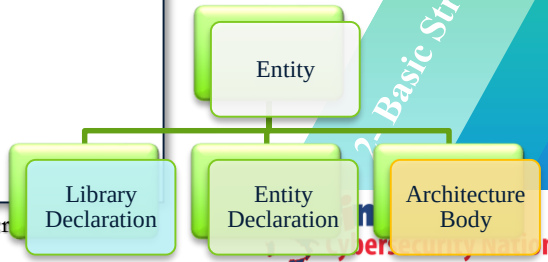
Entity Declaration

Architecture Body

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.std_logic_unsigned.ALL;
4
5  ENTITY UP_COUNTER IS
6      PORT (
7          clk: IN std_logic; -- clock input
8          reset: IN std_logic; -- reset input
9          counter: OUT std_logic_vector(3 DOWNTO 0)-- counter output
10         );
11 END UP_COUNTER;
12
13 ARCHITECTURE Behavioral OF UP_COUNTER IS
14     SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
15 BEGIN
16     PROCESS (clk, reset)
17     BEGIN
18         IF (clk'EVENT AND clk='1') THEN
19             IF (reset='1') THEN
20                 counter_up <= "0000";
21             ELSE
22                 counter_up <= counter_up + "0001";
23             END IF;
24         END IF;
25     END PROCESS;
26     counter <= counter_up;
27 END Behavioral;
```

Concurrent Statements

- An entity contains input, output ports and their types
- Architecture body contains several concurrent statements



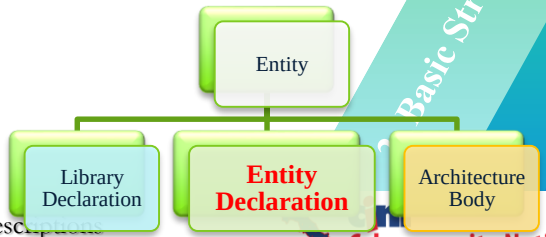
A VHDL Description, a Hardware Entity

Entity Name

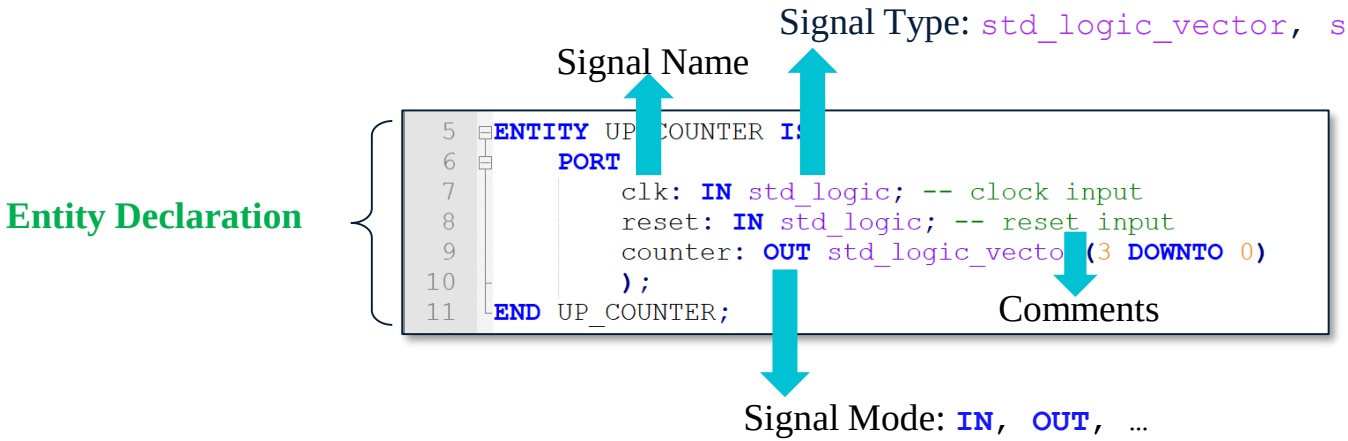
Entity Declaration

```
5  ENTITY UP_COUNTER IS
6      PORT (
7          clk: IN std_logic; -- clock input
8          reset: IN std_logic; -- reset input
9          counter: OUT std_logic_vector(3 DOWNTO 0)
10         );
11 END UP_COUNTER;
```

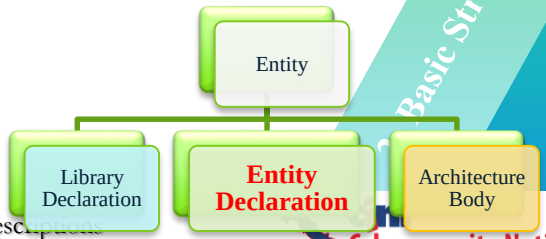
- Entity name, ports, types and modes



A VHDL Description, a Hardware Entity



- Entity name, ports, types and modes



A VHDL Description, a Hardware Entity

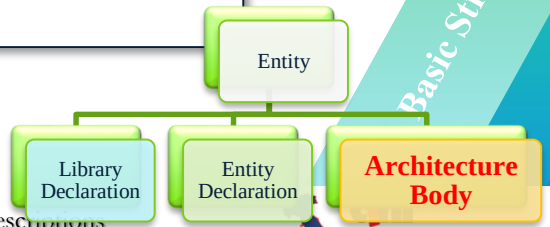
Architecture Name

Entity Name

Architecture Body

```
13  ARCHITECTURE Behavioral OF UP_COUNTER IS
14      SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
15  BEGIN
16      PROCESS (clk, reset)
17      BEGIN
18          IF (clk'EVENT AND clk='1') THEN
19              IF (reset='1') THEN
20                  counter_up <= "0000";
21              ELSE
22                  counter_up <= counter_up + "0001";
23              END IF;
24          END IF;
25      END PROCESS;
26      counter <= counter_up;
27  END Behavioral;
```

- In addition to signals in the entity declaration, can declare new signals in the architecture declarative part



A VHDL Description, a Hardware Entity

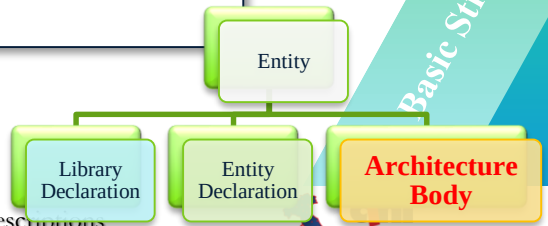
Signal Declaration

Architecture Body

```
13  ARCHITECTURE Behavioral OF UP_COUNTER IS
14      SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
15  BEGIN
16      PROCESS (clk, reset)
17      BEGIN
18          IF (clk'EVENT AND clk='1') THEN
19
20
21
22              counter_up <= counter_up + "0001";
23          END IF;
24      END IF;
25      END PROCESS;
26      counter <= counter_up;
27  END Behavioral;
```

Signal Name Signal Type

- In addition to signals in the entity declaration, can declare new signals in the architecture declarative part



A VHDL Description, a Hardware Entity - Enumeration Types

- Basic logic types are of the enumeration type
- Logic values are the enumeration elements

- **`bit:`** can take **`'0'`** or **`'1'`**
- **`bit_vector:`** is a vector of **`bit`** values
- **`std_logic:`** can take the value **`'X'`**, **`'0'`**, **`'1'`**, **`'Z'`** and five others
- **`std_logic_vector:`** is a vector of **`std_logic`** values

Signals and Variables - Declaration

Signal Declaration

```
1  ARCHITECTURE two_processes OF aCircuit IS
       SIGNAL d : BIT;
       SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
4  BEGIN
5      p1: PROCESS (a, b, cv)
6          VARIABLE e : BIT;
7          VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
8      BEGIN
9          -- Can see all of aCircuit, plus d, dv, e, and ev.
10         . . .
11     END PROCESS;
12
13     p2: PROCESS (av, bv, c)
14         VARIABLE f : BIT;
15         VARIABLE fv : BIT_VECTOR (7 DOWNTO 0);
16     BEGIN
17         -- Can see all of aCircuit, plus d, dv, f, and fv.
18         . . .
19     END PROCESS;
20 END ARCHITECTURE two_processes;
```

- Declare signals, scalars and vectors in the architecture declarative part, no variables
- Temporary variables local to a process only within process declaration part

Signals and Variables - Declaration

Signal Declaration

Variable Declaration

Variable Declaration

```
1  ARCHITECTURE two_processes OF aCircuit IS
       SIGNAL d : BIT;
       SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
4  BEGIN
5      p1: PROCESS (a, b, cv)
           VARIABLE e : BIT;
           VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
8      BEGIN
9          -- Can see all of aCircuit, plus d, dv, e, and ev.
10         . . .
11     END PROCESS;
12
13     p2: PROCESS (av, bv, c)
           VARIABLE f : BIT;
           VARIABLE fv : BIT_VECTOR (7 DOWNTO 0);
16     BEGIN
17         -- Can see all of aCircuit, plus d, dv, f, and fv.
18         . . .
19     END PROCESS;
20 END ARCHITECTURE two_processes;
```

- Declare signals, scalars and vectors in the architecture declarative part, no variables
- Temporary variables local to a process only within process declaration part

2- Basic Structures of VHDL

Signals and Variables - Assignment

- Variables can only be read or written in the processes they are declared
- Signals can be read or written in all architecture processes

Variable Assignment

```
1  ARCHITECTURE mixed_processes_assignments OF aCircuit IS
2      SIGNAL d : BIT;
3      SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
4  BEGIN
5      p1: PROCESS (a, b, cv)
6          VARIABLE e : BIT;
7          VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
8      BEGIN
9          IF (a = b) THEN ev := av; ELSE ev := bv;
10
11         IF (a = '1') THEN wv <= av; ELSE wv <= "1000111";
12         d <= e;
13     END PROCESS;
14
15     dv <= av XOR bv;
16     w <= d AND a;
17 END ARCHITECTURE mixed_processes_assignments;
```

Signals and Variables - Assignment

- Variables can only be read or written in the processes they are declared
- Signals can be read or written in all architecture processes

```
1  ARCHITECTURE mixed_processes_assignments OF aCircuit IS
2      SIGNAL d : BIT;
3      SIGNAL dv : BIT_VECTOR (7 DOWNTO 0);
4  BEGIN
5      p1: PROCESS (a, b, cv)
6          VARIABLE e : BIT;
7          VARIABLE ev : BIT_VECTOR (7 DOWNTO 0);
8      BEGIN
           IF (a = b) THEN ev := av; ELSE ev := bv;
           IF (a = '1') THEN wv <= av; ELSE wv <= "1000111";
           d <= e;
13     END PROCESS;
14
       dv <= av XOR bv;
       w <= d AND a;
17 END ARCHITECTURE mixed_processes_assignments;
```

Variable Assignment

Signal Assignment

Signal Assignment