

EE 1193

INTRODUCTION TO HDL

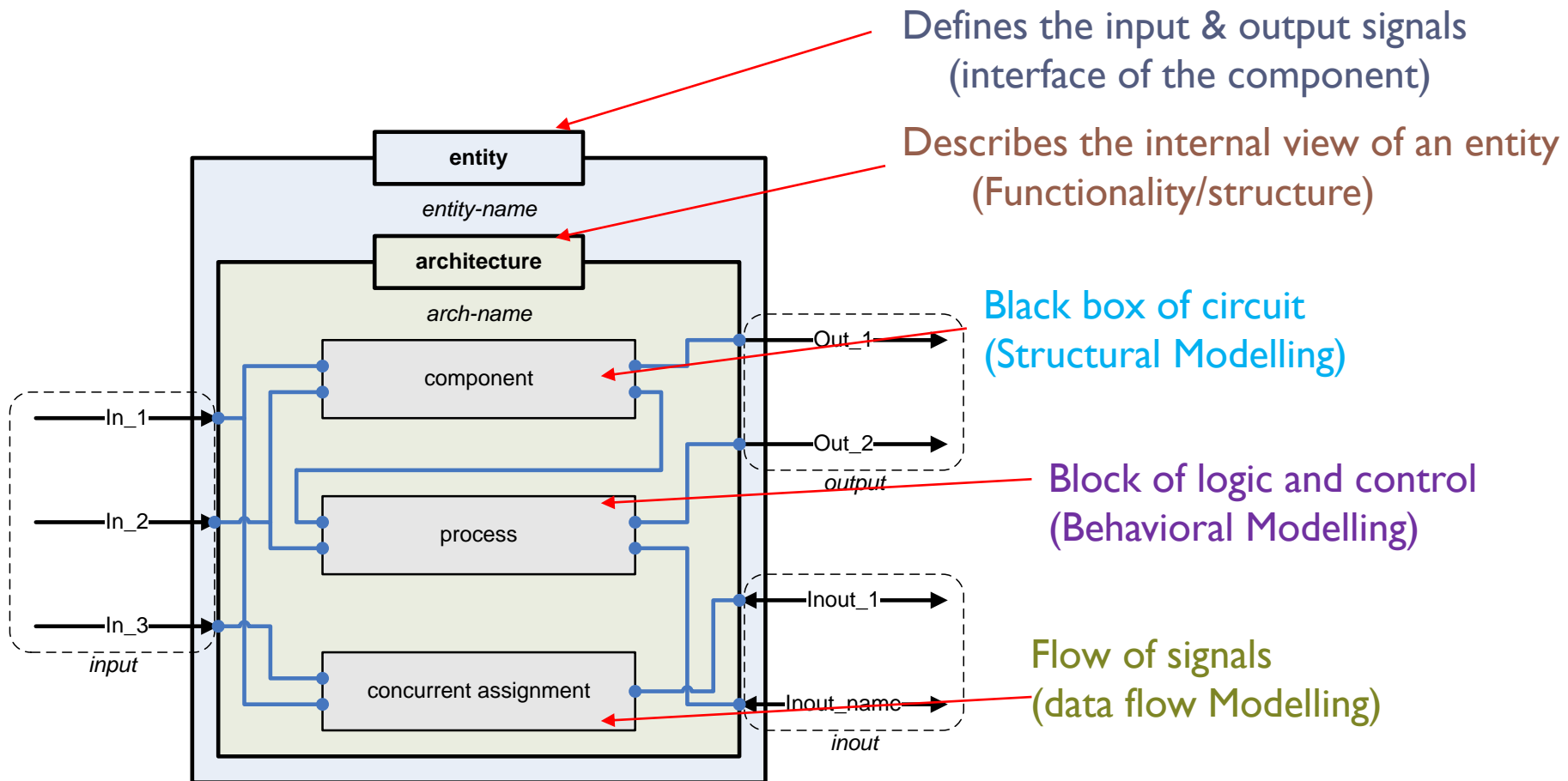
MODULE: HDL basics – preliminaries for Behavioral HDL

Dr. Amit Acharyya, IITH

Basic terminology

- Library: Library is a collection of pre-defined keywords and components defined in it and contains all characteristics of all components.
- Entity: A hardware abstraction of digital systems is called entity. All designs are expressed in entities.
- Architecture: All entities that can be simulated have architecture description. The architecture describes the behavior of entity.
- Process: A process is a basic unit of execution in VHDL source code. All operations that are performed in a simulation of a VHDL description are broken into single or multiple processes
- Bus: A group of signals at one position.

A Generic VHDL Model



Entity Declaration

```
Entity entity_name is  
    Generic (generic list);  
    Port (port list);  
End [entity_name];
```

E.g.: entity nor_gate is
 port(a,b: in bit; c: out bit);
end nor_gate;

Architecture Body Syntax

architecture *architecture-name* of *entity-name* is
[architecture-item-declaration;]
begin
Concurrent-statements;
Process-statements;
concurrent-assertion-statement
concurrent-signal-assignment-statement
component-instantiation-statement
end [*architecture-name*] ;

Generic VHDL Structure

```
library lib_name;  
use lib_name.package_name.item_name;  
entity entity-name is  
    [port ( ln_1, ln_2, ln_3 : in bit;  
           out_1, out_2 : out bit;  
           inout_1, inout_2 : inout bit);]  
end [entity] [entity-name];  
  
architecture arch-name of entity-name is  
    [declaration] ---Signals, constants, component.. etc.  
begin  
    architecture body -----Structural&/Behavioral&/Data flow  
end [architecture] [arch-name];
```

Library declaration and accessing

Architecture body declaration

- ▶ This is called behavioral description/Register Transfer Level (RTL) description

Data objects

- Constant
- Variable
- Signal

Declaration:

Constant `t:time:=10 ns;`

Variable `bus: bit_vector(3 downto 0);`

Variable `sum: integer range 0 to 100:=10;`

Signal `clock: bit;` `--std_logic`

Signal `data_bus: bit_vector (0 to 7); --std_logic_vector`

Structural Modelling

Example: Full adder

```
Library ieee;
Use ieee.std_logic_1164.all;
entity system_st is
    port ( A, B, Cin: in bit;
          Sum, Carry : out bit);
end system_st;
architecture structural of system_st is
    component gate1
        port ( IN_A, IN_B, IN_C : in bit;
              OUT_Z : out bit );
    end component;
    component gate2
        port ( IN_A, IN_B, IN_C : in bit;
              OUT_Z : out bit );
    end component;
begin
    G1 : gate1
        port map (A, B, Cin, Sum);
    G2 : gate2
        port map (A, B, Cin, Carry);
end structural;
```

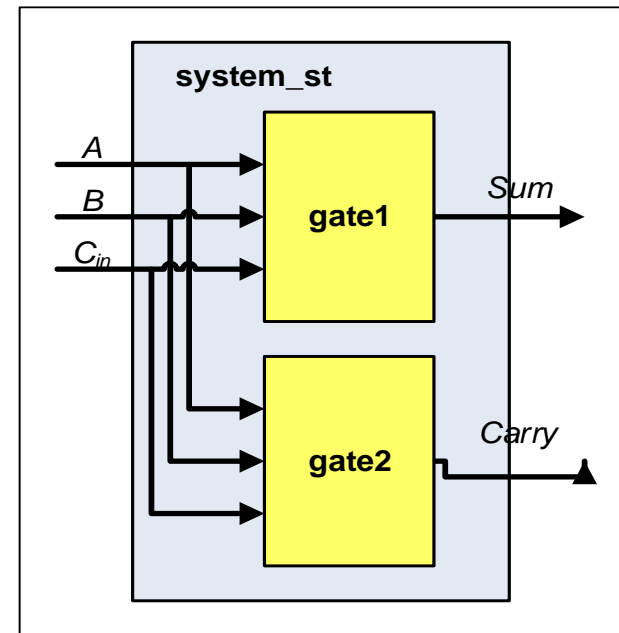


Fig: Schematic of full adder

Basic elements

In Structural Modelling an entity is modeled as a set of components connected by signals(wires) → Netlist

- Component
- Signal

Component :An instance of already designed circuit/system/gate.

- It is simply a black box, whose functionality is not explicitly apparent from its model
- A component should be declared before its appearance in the design

Component declaration

- It declares the name and the interface of a component.
- Interface specifies the mode and the type of the ports
- Components are declared in the declaration part of the architecture body

Syntax:

```
component component-name  
    port ( list-of-interface-ports ) ;  
end component;
```

component-name: may or may not refer to the name of an already existing entity in a library. If it does not, it must be explicitly bound to an entity; otherwise, the model cannot be simulated

Cont....

list-of-interface-ports: name (may be different), mode and type for each port of the component in a manner similar to that specified in an entity declaration.

Examples:

```
component NAND2
```

```
    port (A, B: in MVL; Z: out MVL);
```

```
end component;
```

```
component MP
```

```
    port (CK, RESET, RON, WRN: in BIT;
```

```
          DATA_BUS: inout INTEGER range 0 to 255;
```

```
          ADDR_BUS: in BIT_VECTOR(15 downto 0));
```

```
end component;
```

Component Instantiation

- Defines a subcomponent of the entity in which it appears
- It connects the signals in the entity with the ports of the subcomponent

Syntax:

Component-label: component-name port map (association-list)

Can be any legal identifier (name of the instance)

Must be the name of a component declared earlier

Associates the signals in the entity (*actuals*), with the ports of a component (*locals*)

Actual must be an object of class signal;
Expressions or objects of class variable or constant are not allowed;
Can be a keyword *open* to indicate a port that is not connected

Association

Two ways to associate locals with the actuals.

1. Positional association
2. Named association

Positional association:

Depending on the order of the ports they appear in the component declaration, the locals will be mapped to the actuals.

Association-list format: $actual_1, actual_2, actual_3, \dots, actual_n$

i.e. the first port in the component instantiation corresponds to the first actual in the component instantiation, the second with the second,.....

Cont.

Example: Instance of NAND2 component

-- Component declaration:

```
component NAND2  
port (A, B: in BIT; Z: out BIT);  
end component;
```


-- Component instantiation:

```
NI: NAND2 port map (S1, S2, S3);
```

Component label



Actual S1,S2,S3 are associated
with ports A,B & Z(locals) of the
NAND2



Named association

➤ Association-list format:

$\text{local}_1 \Rightarrow \text{actual}_1, \text{local}_2 \Rightarrow \text{actual}_2, \dots, \text{local}_n \Rightarrow \text{actual}_n$

The ordering of the associations is not important since the mapping between the actuals and locals are explicitly specified.

Scope of the locals is restricted to be within the port map part of the instantiation for a particular component

The type of the local and the actual being associated must be the same

The modes of the ports must conform i.e. if the local is readable(writable) then the actual should be readable(writable)

$\text{in} \Rightarrow \text{in}, \text{out} \Rightarrow \text{out} \ \& \ \text{in/out/inout} \Rightarrow \text{inout}$


Cont.

-- Component declaration:

```
component NAND2  
port (A, B: in BIT; Z: out BIT);  
end component;
```

-- Component instantiation:

```
NI: NAND2 port map (A=>S1, Z=> S2,  
B=>S3);
```



Order is not important ;
A, B, Z are ports (locals) of the component
declared and S₁, S₂, S₃ are ports (actuals) of the
entity declared

cont.

- Structural models can be simulated only after the entities that the components represent are modeled and placed in a design library.
- The lowest level entities must be behavioral models.
- All components that are instantiated will be executed concurrently, so we no need to bother about the order of the component instantiation.

Signal declaration:

- Signals should be declared before their use in the architecture body
- No need to specify the mode

signal *list-of-signal-names* : **type-name** [*:= initial-value*] ;

Ex: **signal** *sig_0 sig_1* : **bit** ;

Examples

- **Assumption:** The behavioral models of the components being instantiated are available

Ex I: 4-bit adder

- Requires four full adders

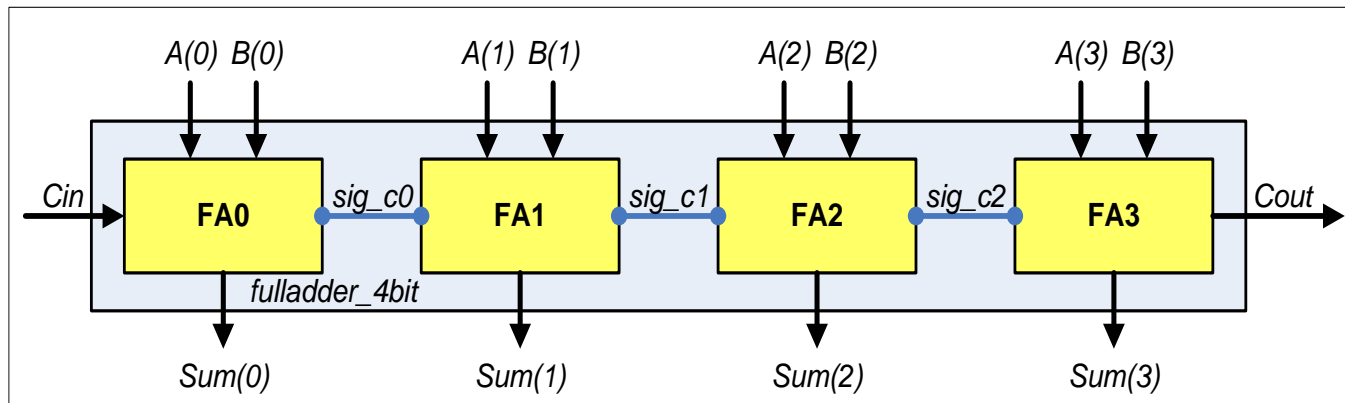


Fig. Block diagram of 4-bit adder

Structural description of the 4-bit adder

```
library ieee;
use ieee.std_logic_1164.all
entity fulladder_4bit is
    port (A, B : in bit_vetcor (3 downto 0);
          Cin : in bit;
          Sum : out bit_vetcor (3 downto 0);
          Cout : out bit);
end fulladder_4bit ;
architecture hirarchical of fulladder_4bit is
    component fulladder_st
        port (A, B, Cin : in bit;
              Sum, Cout : out bit);
    end component;
    signal sig_c0, sig_c1, sig_c2 : bit;
begin
    FA0 : fulladder_st port map (A(0), B(0), Cin, Sum(0), sig_c0);
    FA1 : fulladder_st port map (A(1), B(1), sig_c0, Sum(1), sig_c1);
    FA2 : fulladder_st port map (A(2), B(2), sig_c1, Sum(2), sig_c2);
    FA3 : fulladder_st port map (A(3), B(3), sig_c2, Sum(3), Cout);
end hirarchical;
```

Ex2: 4-bit Register

--library declaration

entity reg4

port (d: in bit_vector(3 downto 0); en,clk:in bit;
q:out bit_vector(3 downto 0));

end reg4;

architecture *struct* of reg4 is

component *d_latch*

port (d, clk : in bit;
q : out bit);

end component;

component *and2_op*

port (x, y : in bit;
z : out bit);

end component;

signal *int_clk* : bit;

begin

DFF3 : *d_latch* **port map**(d(3), *int_clk*, q(3));

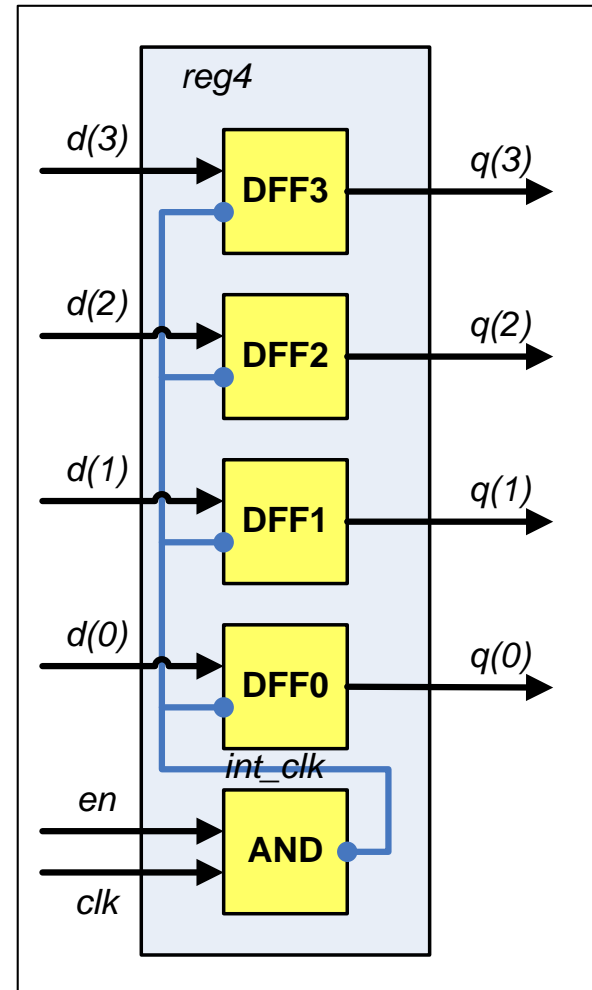
DFF2 : *d_latch* **port map**(d(2), *int_clk*, q(2));

DFF1 : *d_latch* **port map**(d(1), *int_clk*, q(1));

DFF0 : *d_latch* **port map**(d(0), *int_clk*, q(0));

AND : *and2_op* **port map**(en, clk, *int_clk*);

end *struct*;



Exercise: Full adder: structural description

3-input XOR gate:

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

entity xor_gate is
  port(
    in1: in std_logic;
    in2: in std_logic;
    in3: in std_logic;
    out1: out std_logic);
end entity xor_gate;

architecture flow of xor_gate is
  begin
    out1 <= (in1 xor in2) xor in3;
  end architecture flow;
```

In1*in2+n2*in3+n1*in3 :AO3_gate

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

entity AO3_gate is
port(
    in1: in std_logic;
    in2: in std_logic;
    in3: in std_logic;
    out1: out std_logic);
end entity AO3_gate;

architecture flow of AO3_gate is
begin
    out1 <= (in1 and in2) or (in2 and in3) or (in1 and in3);

end architecture flow;
```

Full adder structural description

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

```
entity Full_adder is  
  port(  
    in1: in std_logic;  
    in2: in std_logic;  
    in3: in std_logic;  
    sum: out std_logic;  
    carry: out std_logic);  
end entity Full_adder;
```

architecture flow of Full_adder is

```
  component xor_gate  
    port(  
      in1: in std_logic;  
      in2: in std_logic;  
      in3: in std_logic;  
      out1: out std_logic);  
  end component;
```


Cont..

```
component AO3_gate
  port(
    in1: in std_logic;
    in2: in std_logic;
in3: in std_logic;
    out1: out std_logic);
  end component;

begin
  summ: xor_gate port map(in1=>in1,in2=>in2,in3=>in3,out1=>sum);
  carry1: AO3_gate port
map(in1=>in1,in2=>in2,in3=>in3,out1=>carry);

end architecture flow;
```

Test bench

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

entity tb_fulladder_gate is end tb_fulladder_gate;
architecture test of tb_fulladder_gate is
    component full_adder
        port(
            in1: in std_logic;
            in2: in std_logic;
in3: in std_logic;
            sum: out std_logic;
            carry: out std_logic);
    end component;

    signal in1 : std_logic:= '0';
    signal in3 : std_logic:= '0';
    signal in2 : std_logic:= '0';
    signal sum: std_logic;
    signal carry: std_logic;
```

Cont..

begin

 dut:full_adder port map(in1=>in1, in2=>in2, in3=>in3,
sum=>sum, carry=>carry);

 process begin

 wait for 20 ns; in1<= '0'; in2<= '0'; in3<='0';

wait for 20 ns; in1<= '0'; in2<= '0' ;in3<='1';

wait for 20 ns; in1<= '0'; in2<= '1' ;in3<='0';

wait for 20 ns; in1<= '0'; in2<= '1' ;in3<='1';

wait for 20 ns; in1<= '1'; in2<= '0' ;in3<='0';

wait for 20 ns; in1<= '1'; in2<= '0' ;in3<='1';

wait for 20 ns; in1<= '1'; in2<= '1' ;in3<='0';

wait for 20 ns; in1<= '1'; in2<= '1' ;in3<='1';

wait;

end process;

end architecture test;

Behavioral Modelling



Behavioral description

Behavior of the entity is expressed using sequentially executed procedural code and semantics that of a high-level programming languages (Ex: C/C++,..etc.) when you capture the underlying behavior of the circuit instead of the exact circuit structure.

Ex: Behavioral description of full adder

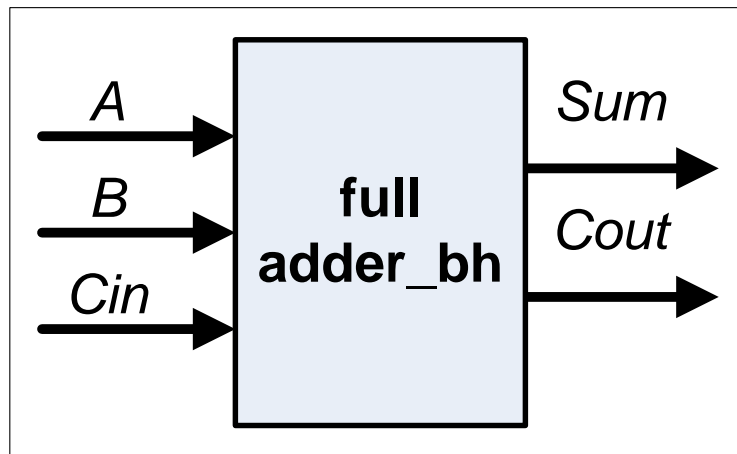


Fig: entity of full adder

Cont..

```
entity fulladder_bh is  
  port ( A, B, Cin : in bit;  
         Sum, Cout : out bit);  
end fulladder_bh;
```

```
architecture behavioral of fulladder_bh is  
begin  
  process (A , B, Cin)  
  begin  
    if ( A = '0' and B = '0' and Cin = '0') then  
      Sum <= '0';  
      Cout <= '0';  
    elsif ( A = '0' and B = '0' and Cin = '1') then  
      Sum <= '1';  
      Cout <= '0';  
    elsif ( A = '0' and B = '1' and Cin = '0') then  
      Sum <= '1';  
      Cout <= '0';  
    elsif ( A = '0' and B = '1' and Cin = '1') then  
      Sum <= '0';  
      Cout <= '1';  
    elsif ( A = '1' and B = '0' and Cin = '0') then  
      Sum <= '1';  
      Cout <= '0';
```

```
    elsif ( A = '1' and B = '0' and Cin = '1') then  
      Sum <= '0';  
      Cout <= '1';  
    elsif ( A = '1' and B = '1' and Cin = '0') then  
      Sum <= '0';  
      Cout <= '1';  
    else  
      Sum <= '1';  
      Cout <= '1';  
    end if;  
  end process;  
end behavioral;
```

Process Statement

process statement is the set of sequential statements, which describes the functionality of a portion of an entity.

An architecture can have multiple processes to describe the functionality of the entity.

Syntax of the Process statement

[*process-label*:] **process** [(*sensitivity-list*)]
[*process-item-declarations*] -----scope is local

begin

–all are sequential-statements

variable-assignment-statement

signal-assignment-statement

wait-statement

if-statement

case-statement

loop-statement

null-statement

exit-statement

next-statement

assertion-statement

end process [*process-label*];

Cont....

Sensitivity list: A set of signals that the process is sensitive. i.e. whenever an event occurs on any of the signals in the sensitivity list, the sequential statements within the process are executed in a sequential order, in the order they appear (Like High level languages).

- Process **suspends** after executing the last sequential statement and waits for another event to occur on a signal in the sensitivity list. (never be exited)

Process is always in active state or suspended state throughout the simulation time.

- Variables should be declared in the item declaration part of the process.

Wait statement

Usually the process suspends after executing the last sequential statement. The alternative way to suspend the process is using the wait statement.

Basic forms of “Wait” statement:

wait on sensitivity-list;

ex: wait on a, b; --suspends until an event occurs on any one of the sensitivity-list

wait until Boolean-expression;

ex: wait until (x>y);

wait for time-expression;

ex: wait for 3ns;

Cont..

- A process without sensitivity list and wait statements will never get suspended and would remain in an infinite loop during the initialization phase of simulation.
- A process without sensitivity list should have at least one wait statement

Ex: **process** -- No sensitivity list.

variable TEMP1 ,TEMP2: BIT;

begin

 TEMP1 :=A **and** B:

 TEMP2 := C **and** D;

wait on A, B, C, D; -- Replaces the sensitivity list.

- Having the sensitivity list and the wait statements in the process is an error.

IF Statement

- Selects a sequence of statements for execution based on the value of a condition.

Syntax:

```
if boolean-expression then  
    sequential-statements  
    [ elsif boolean-expression then  
        sequential-statements ]  
    [ else  
        sequential-statements ]  
End if;
```

It can have zero or more *elsif* clauses and an optional *else* clause.

Cont..

```
ex: if (a>b) then  
    c=a+b;  
    elsif (a<b)  
    c=a-b;  
    else  
    c=a*b  
    end if;
```

Case Statement

- Selects one of the branches for execution based on the value of expression (one dimensional array)
- Choices: Single value or range of values(using '|' or by using 'others' clause)

Syntax:

case *expression* **is**

when *choices* **=>** *sequential-statements* -- branch #1

when *choices* **=>** *sequential-statements* -- branch #2

-- Can have any number of branches.

[**when** **others** **=>** *sequential-statements*] -- last branch

end case;

- All possible values of the expression must be covered in the case statement.

Cont..

Example: 4*I multiplexer

entity MUX is

port (A, B, C, D: **in** BIT; CTRL: **in** BIT_VECTOR(0 to 1);

 Z: **out** BIT);

end MUX;

architecture MUX_BEHAVIOR **of** MUX is

constant MUX_DELAY: TIME := 10 ns;

begin

 PMUX: **process** (A, B, C, D, CTRL)

variable TEMP: BIT;

begin

case CTRL is

when "00" => TEMP := A;

when "01" => TEMP := B;

when "10" => TEMP := C;

when "11" => TEMP := D;

end case;

 Z <= TEMP **after** MUX_DELAY;

end process PMUX;

end MUX_BEHAVIOR

Null Statement

Sequential statement that doesn't cause any action and execution continues with the next statement

Syntax: null;

ex: case S is

 when "00" => d<=a+b;

 when "01" => null;

 when others => d<=a*b;

Loop Statement

To execute the same set of sequential statements iteratively.

Syntax: [*loop-label* :] *iteration-scheme* **loop**
 sequential-statements
 end loop [*loop-label*] ;

There three *iteration-schemes*

I.For:

Syntax: **for** identifier *in* range

Example:

FACTORIAL := 1;

for NUMBER in 2 to N loop

FACTORIAL := FACTORIAL * NUMBER;

end loop;

implicitly integer no need to declare it.
No declaration for the Loop identifier

Cont..

2. While

Syntax: **While** Boolean-expression

Example:

```
J:=0;SUM:=10;  
WH-LOOP: while J < 20 loop  
SUM := SUM * 2;  
    J:=J+3;  
end loop;
```

3. Using exit or next statement

Example:

```
SUM:=1;j:=0;  
L2: loop      -- This loop also has a label.  
j:=j+21;  
SUM := SUM* 10;  
exit when SUM > 100;  
end loop L2;
```

Exercise

Ex: Behavioral description of full adder

```
library ieee;
use ieee.std_logic_1164.all;
entity fulladder_bh is
    port ( A, B, Cin : in std_logic;
           Sum, Cout : out std_logic);
end fulladder_bh;
architecture behavioral of fulladder_bh is
begin
    process (A , B, Cin)
    begin
        if ( A = '0' and B = '0' and Cin = '0') then
            Sum <= '0';
            Cout <= '0';
        elsif ( A = '0' and B = '0' and Cin = '1') then
            Sum <= '1';
            Cout <= '0';
```

Cont..

```
elsif ( A = '0' and B = '1' and Cin = '0') then
    Sum <= '1';
    Cout <= '0';
elsif ( A = '0' and B = '1' and Cin = '1') then
    Sum <= '0';
    Cout <= '1';
elsif ( A = '1' and B = '0' and Cin = '0') then
    Sum <= '1';
    Cout <= '0';
elsif ( A = '1' and B = '0' and Cin = '1') then
    Sum <= '0';
    Cout <= '1';
elsif ( A = '1' and B = '1' and Cin = '0') then
    Sum <= '0';
    Cout <= '1';
else
    Sum <= '1';
    Cout <= '1';
end if;
end process;
end behavioral;
```

Test bench

```
library ieee;
use ieee.std_logic_1164.all;
entity tb_fulladder_bh is
end tb_fulladder_bh;
architecture tester of tb_fulladder_bh is
    component fulladder_bh
        port(      A: in std_logic;      B: in std_logic;      Cin: in std_logic;
              sum:out std_logic;      Cout:out std_logic);
    end component;
    signal A : std_logic:='0';
    signal B : std_logic:='0';
    signal Cin : std_logic:='0';
    signal sum:std_logic;
    signal Cout:std_logic;
begin
```

Cont..

```
dut:fulladder_bh port map(A=>A, B=>B, Cin=>Cin, sum=>sum, Cout=>Cout
);
```

```
    process
```

```
begin
```

```
    wait for 20 ns; A<= '0'; B<= '0'; Cin<='0';
```

```
    wait for 20 ns; A<= '0'; B<= '0' ;Cin<='1';
```

```
    wait for 20 ns; A<= '0'; B<= '1' ;Cin<='0';
```

```
    wait for 20 ns; A<= '0'; B<= '1' ;Cin<='1';
```

```
    wait for 20 ns; A<= '1'; B<= '0' ;Cin<='0';
```

```
    wait for 20 ns; A<= '1'; B<= '0' ;Cin<='1';
```

```
    wait for 20 ns; A<= '1'; B<= '1' ;Cin<='0';
```

```
    wait for 20 ns; A<= '1'; B<= '1' ;Cin<='1';
```

```
    wait;
```

```
end process;
```

```
end architecture tester;
```

Practice ex3

Counter:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
entity counter8 is

    --generic ( n : natural := 4);
    port(
        clk : in std_logic;
        nreset : in std_logic;
        nready : in std_logic;
        count : out std_logic_vector (2 downto 0)
    );

end counter8;
architecture counter8_arch of counter8 is
    signal temp : unsigned (2 downto 0);
    constant MAXCOUNT : unsigned (2 downto 0) := "111";
begin
```

Cont..

```
p0: process (nreset, clk) is
    --signal temp : unsigned (3 downto 0);
begin
    if (nreset = '0') then
        temp <= (others => '0');
        --count <= (others => '0');
    elsif (rising_edge(clk)) then
        if (nready = '1') then
            temp <= (others => '0');
        elsif (temp = MAXCOUNT) then
            temp <= (others => '0');
        else
            temp <= temp + 1;
        end if;
    end if;

    --count <= std_logic_vector(temp);

end process p0;
count <= std_logic_vector(temp);
end counter8_arch;
```


Enable generator

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
-----
entity en_gen is
    port(    clk : in std_logic;
            nreset: in std_logic;
            count_in: in std_logic_vector(2 downto 0);
            enable : out std_logic
            );
end entity en_gen;
-----
```

```
architecture en_gen_arch of en_gen is
begin
    process (nreset, clk) is
    begin
        if (nreset = '0') then
            enable <= '0';
        elsif (rising_edge(clk)) then
            if (count_in = "110") then
                enable <= '1';
            else
                enable <= '0';
            end if;
        end if;
    end process;
end en_gen_arch;
```

Flip-flop

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
-----
entity flop7 is

    port(
        clk: in std_logic;
        nreset: in std_logic;
        enable: in std_logic;
        data_in : in std_logic_vector (15 downto 0);
        data_out: out std_logic_vector(15 downto 0)
    );

end entity flop7;
-----
```

architecture flop7_arch of flop7 is

```
begin
    process (nreset, clk) is
        begin
            if (nreset = '0') then
                data_out <= (others=>'0');
            elsif (rising_edge(clk)) then
                if (enable = '1') then
                    data_out <= data_in;
                end if;
            end if;
        end process;
    end flop7_arch;
```

top

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
-----
entity top is
    port (
        clk : in std_logic;
        nreset: in std_logic;
        nready : in std_logic;
        data_in: in std_logic_vector(15 downto 0);
        data_out: out std_logic_vector(15 downto 0)
    );

end entity top;
-----

architecture top_arch of top is
    -----

    signal counter : std_logic_vector(2 downto 0);
```

```
signal enable_out : std_logic;
```

```
-----  
component counter8 is
```

```
    port(
```

```
        clk : in std_logic;
```

```
        nreset : in std_logic;
```

```
        nready : in std_logic;
```

```
        count : out std_logic_vector (2 downto 0)
```

```
    );
```

```
end component;
```

```
component en_gen is
```

```
    port(
```

```
        clk : in std_logic;
```

```
        nreset: in std_logic;
```

```
        count_in: in std_logic_vector(2 downto 0);
```

```
        enable : out std_logic
```

```
    );
```

```
end component;
```

```
component flop7 is
```

```
port(  
    clk: in std_logic;  
    nreset: in std_logic;  
    enable: in std_logic;  
    data_in : in std_logic_vector (15 downto 0);  
    data_out: out std_logic_vector(15 downto 0)  
);  
  
end component;  
-----  
begin  
  
unit_counter: counter8  
    port map(  
        clk    => clk,  
        nreset => nreset,  
        nready => nready,  
        count  => counter  
    );
```

```
unit_enable: en_gen
    port map(
        clk          => clk,
        nreset       => nreset,
        count_in     => counter,
        enable       => enable_out
    );

unit_flop: flop7
    port map(
        clk          => clk,
        nreset       => nreset,
        enable       => enable_out,
        data_in      => data_in,
        data_out     => data_out
    );

end top_arch;
```


TEST BENCH

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

entity tb_top is
end entity tb_top;

architecture tb_top_arch of tb_top is

component top is
    port (
        clk : in std_logic;
        nreset: in std_logic;
        nready : in std_logic;
        data_in: in std_logic_vector(15 downto 0);
        data_out: out std_logic_vector(15 downto 0)
    );

end component;

-----
```

```
--component ports--
signal clk : std_logic := '1';
signal nreset : std_logic := '1';
signal nready : std_logic;
signal data_in : std_logic_vector(15 downto 0) := (others => '0');
signal data_out : std_logic_vector(15 downto 0);
begin
---component instantiation----
dut: top
    port map(
        clk => clk,
        nreset => nreset,
        nready => nready,
        data_in => data_in,
        data_out => data_out
    );
--clock generation--
```

```
clock_gen: process(clk) -- clock generator and one shot clear signal
begin
    clk <= not clk after 500 ns; -- 1000 ns period
end process clock_gen;
wavegen_proc: process
begin
    nreset <= '0';
    nready <= '1';
    wait for 1000 ns;
    nreset <= '1';
    wait for 1000 ns;
    nready <= '0';
wait for 1000 ns;
    --count_in <= "000";
    data_in <= "0111001100110011";
    wait for 1000 ns;
    --count_in <= "001";
    data_in <= "0001001100110011";
    wait for 1000 ns;
    --count_in <= "010";
    data_in <= "0010001100110011";
```

```
wait for 1000 ns;
--count_in <= "011";
data_in <= "0011101100110011";
wait for 1000 ns;
--count_in <= "100";
data_in <= "0011010100110011";
wait for 1000 ns;
--count_in <= "101";
data_in <= "0011010000110011";
wait for 1000 ns;
--count_in <= "110";
data_in <= "0011001111000011";
wait for 1000 ns;
--count_in <= "111";
data_in <= "0011001100101111";
wait for 1000 ns;
--count_in <= "000";
data_in <= "0011001100111111";
wait for 1000 ns;
--count_in <= "001";
data_in <= "0011101101110011";
wait for 1000 ns;
--count_in <= "010";
data_in <= "0111001100110011";
```

```
wait for 1000 ns;
--count_in <= "011";
data_in <= "0011011100110011";
wait for 1000 ns;
--count_in <= "100";
data_in <= "0011101100110011";
wait for 1000 ns;
--count_in <= "101";
data_in <= "0011101100110011";
wait for 1000 ns;
--count_in <= "110";
data_in <= "0011111110110011";
wait for 1000 ns;
--count_in <= "111";
data_in <= "0011001110110011";
wait for 1000 ns;
--count_in <= "000";
data_in <= "0011001100110011";
wait for 1000 ns;
--count_in <= "001";
data_in <= "0011001110110011";
wait for 1000 ns;
```

```
--count_in <= "010";  
data_in <= "0011001101110011";  
wait for 1000 ns;  
--count_in <= "011";  
data_in <= "0011001100111011";  
wait for 1000 ns;  
--count_in <= "100";  
data_in <= "0011001100110111";  
wait for 1000 ns;  
--count_in <= "101";  
data_in <= "0011101110111011";  
wait for 1000 ns;  
--count_in <= "110";  
data_in <= "0111011100110011";  
wait for 1000 ns;  
--count_in <= "111";  
data_in <= "0011001101110011";  
wait for 2000 ns;  
  
end process wavegen_proc;  
end tb_top_arch;
```