

CO102

Computer Hardware

Experiment

Lecture 04: VHDL – Sequential Statements

Liang Yanyan

澳門科技大學
Macau of University of Science and Technology

Sequential and Combinational Code

- VHDL code is inherently concurrent.
 - E.g. combinational statements.
- Codes inside PROCESS statement are executed sequentially.
- Sequential code is also called behavioral code.
- PROCESS is often used to implement synchronous circuit (mostly logic operations depend on clock signals).

```
[label:] PROCESS (sensitivity list)
[VARIABLE name type [range]; ]
BEGIN
    (Sequential code);
END PROCESS;
```


Example

```
test_proc: process  
begin
```

```
    S <= "100";  
    A <= "1010";  
    B <= "0110";  
    wait for 10ns;
```

```
    S <= "101";  
    A <= "1000";  
    B <= "1110";  
    wait for 10ns;
```

```
    S <= "110";  
    A <= "1011";  
    B <= "0111";  
    wait for 10ns;
```



Sequential
exucution

Concurrent

- The entire PROCESS is treated as a block, and it is still concurrent with other statements.
- E.g. all the following statements are executed concurrently.

Architecture beta of mydesign is

Begin

A <= B and C;

p1: process ... end process;

D <= E or F;

p2: process ... end process;

p3: process ... end process;

end beta

PROCESS

- A PROCESS is a sequential section of VHDL code.

```
[label:] PROCESS (sensitivity list)
    [VARIABLE name type [range]; ]
    BEGIN
        (Sequential code);
    END PROCESS;
```

- Label: name of this process, e.g. myproc1, this is optional.
- Sensitivity list: a list of signal names, the process is executed every time a signal in the list changes.
 - No sensitivity list if “wait for” is used, but only for simulation.
- Variable: define the variables used in the process.
 - Initial value is not synthesizable, only used in simulation.

Simple example

```
library ieee;
use ieee.std_logic_1164.all;
-----
entity ABC is
    port(
        A0,A1,A2 : in std_logic;
        F0      : in  std_logic;
        F1      : out std_logic;
        F2      : out std_logic );
end ABC;
```

```
-----
architecture ABC_arch of ABC is
    signal tmp1: std_logic;
    signal tmp2: std_logic;
begin
    tmp1 <= A1;
    process(A0)
    begin
        tmp2 := A2;
        if rising_edge(A0) then
            F1 <= not F0;
            F2 <= tmp1 and tmp2;
        end if;
    end process;
end ABC_arch;
```

IF Statement

- “IF” statement is intended for sequential code.
 - It is used inside PROCESS statement.

```
IF condition THEN
    statements;
ELSIF conditions THEN
    statements;
ELSE
    statements;
END IF;
```

Case/When Statement

- Select particular statements to execute depends on conditions.
- Similar to “When/else” statement (Combinational).
“Case/When” statement is intended for sequential code.
 - It is used inside PROCESS statement.

```
CASE selection IS  
  WHEN value1 =>  
    statement1;  
  WHEN value2 =>  
    statement2;  
  WHEN others =>  
    statement3;  
END CASE;
```

```
CASE selection IS  
  WHEN value1 =>  
    statement1;  
  WHEN value2 =>  
    statement2;  
  WHEN others =>  
    NULL;      --No action here  
END CASE;
```


Case/When example 1

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity IfComb is
  port
    (aSel  : in std_logic_vector(3 downto 0);
     aDin  : in std_logic_vector(3 downto 0);
     aDout : out std_logic);
end IfComb;
architecture rtl of IfComb is
begin
  process(aSel, aDin)
  begin
    if aSel(3)='1' then
      aDout <= aDin(3);
    elsif aSel(2)='1' then
      aDout <= aDin(2);
    elsif aSel(1)='1' then
      aDout <= aDin(1);
    elsif aSel(0)='1' then
      aDout <= aDin(0);
    --else
    -- aDout <= '0';
    end if;
  end process;
end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity CaseComb is
  port
    (aSel  : in std_logic_vector(3 downto 0);
     aDin  : in std_logic_vector(3 downto 0);
     aDout : out std_logic);
end CaseComb;
architecture rtl of CaseComb is
begin
  process(aSel, aDin)
  begin
    case aSel is
      when "1000" =>
        aDout <= aDin(3);
      when "0100" =>
        aDout <= aDin(2);
      when "0010" =>
        aDout <= aDin(1);
      when "0001" =>
        aDout <= aDin(0);
      when others =>
        --aDout <= '0';
    end case;
  end process;
end rtl;
```

Case/When example 2

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity dff is
port (d, clk, rst : in std_logic;
      q : out std_logic);
end dff;
architecture rtl of dff is
begin
    process(clk, rst)
    begin
        case rst is
            when '1' => q <= '0';
            when '1' =>
                if (clk'event and clk = '1') then
                    q <= '0';
                end if;
            when others => NULL;
        end case;
    end process;
end dff;
```

Implement combinational circuit

```
architecture my_arch of ABC is  
begin  
    mp1: process(d0,d1)  
    begin  
        out1 <= d0 and d1;  
    end process;  
end my_arch;
```

Compare the functional difference
of these two VHDL programs.

```
architecture my_arch of ABC is  
begin  
    out1 <= d0 and d1;  
end my_arch;
```

Synchronous and Asynchronous “reset”

```
process (clk, reset)
begin
    if (clk = '1' and clk'event) then
        if (reset = '1') then
            out1 <= "0000";
        end if;
    else
        ...
        out1 <= ...
        ...
    end if;
end process;
```

Synchronous

Asynchronous

```
process (clk, reset)
begin
    if (reset = '1') then
        out1 <= "0000";
    elsif (clk = '1' and clk'event) then
        ...
        out1 <= ...
        ...
    end if;
end process;
```