CPE 462
VHDL: Simulation and Synthesis

Topic #06 - a) PROCESS, If and Variables

Nuno Alves (nalves@wne.edu), College of Engineering

Mini-Review

- Last week we discussed concurrent-code.
- Concurrent-code is also known as *dataflow*-code.
- What is concurrent code?
- We will now talk about sequential code.
- Sequential-code is also know as *behavioral* code.

```
entity main_block is
   port (clk, a, b : in bit);
end main_block;

architecture myarch of main_block is
signal x,y,z : bit;

begin
   x <= a AND b;
   y <= a XOR b;

   process (clk)
      begin
         y <= a OR b;
      end process

end architecture;
```

Sequential Code

- VHDL code is inherently concurrent.
- **PROCESSES**, FUNCTIONS, and PROCEDURES are the only sections of code that are executed sequentially.
- However, as a whole, any of these blocks is still concurrent with any other statements placed outside it.
- With sequential code we can build sequential circuits as well as combinational circuits.

```
entity main_block is
   port (clk, a, b : in bit);
end main_block;

architecture myarch of main_block is
signal x,y,z : bit;

begin
   x <= a AND b;
   y <= a XOR b;

   process (clk)
      begin
         y <= a OR b;
      end process

end architecture;
```

- The following statements are only valid inside sequential code (such as processes): IF, WAIT, CASE, and LOOP.
- **Warning:** The following statements are only valid inside concurrent code: WHEN, GENERATE, BLOCK.
- VARIABLES are also restricted to be used in sequential code only (for example, inside a PROCESS).
- Contrary to a SIGNAL, a VARIABLE can never be global, so its value can not be passed out directly.
- We will talk about VARIABLES today.

PROCESS

- A PROCESS is a sequential section of VHDL code.
- It is characterized by the presence of IF, WAIT, CASE, or LOOP, and by a sensitivity list (except when WAIT is used).

What is a sensitivity list?

- Its a list of signals in parenthesis, which tells when the process should be re-evaluated to update the outputs.
- The signal sensitivity list is used to specify which signals should cause the process to be re-evaluated.

```
entity main_block is
   port (clk, a, b : in bit);
end main_block;

architecture myarch of main_block is
signal x,y,z : bit;

begin
   x <= a AND b;
   y <= a XOR b;

   process (clk)
      begin
         y <= a OR b;
      end process

end architecture;
```

PROCESS

- A PROCESS must be installed in the main code, and is executed every time a signal in the sensitivity list changes (or the condition related to WAIT is fulfilled).
- Its syntax is shown below:

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

```
entity main_block is
   port (clk, a, b : in bit);
end main_block;

architecture myarch of main_block is
signal x,y,z : bit;

begin
   x <= a AND b;
   y <= a XOR b;

   process (clk)
      begin
         y <= a OR b;
      end process

end architecture;
```

PROCESS

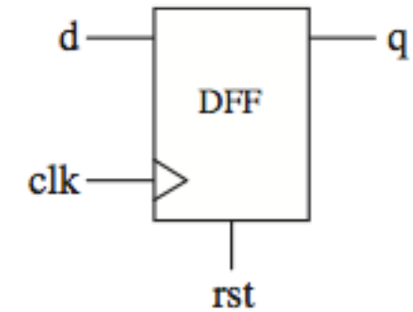
```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

- VARIABLES are optional.
- If variables are used, they must be declared in the declarative part of the PROCESS (before the word BEGIN).
- **The initial value is not synthesizable**, being only taken into consideration in simulations.
- The use of a label is also optional. Its purpose is to improve code readability.

```
entity main_block is
   port (clk, a, b : in bit);
end main_block;

architecture myarch of main_block is
signal x,y,z : bit;

begin
   x <= a AND b;
   y <= a XOR b;

   process (clk)
      begin
         y <= a OR b;
      end process

end architecture;
```

Constructing a synchronous circuit

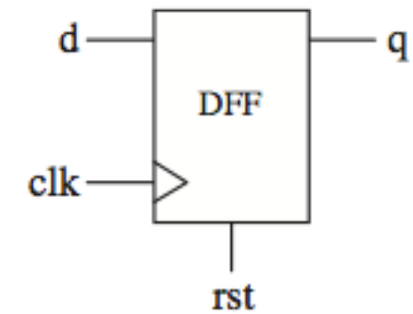
- To construct a synchronous circuit, monitoring a signal (clock, for example) is necessary.
- A common way of detecting a signal change is by means of the EVENT data-attribute.
- For instance, if clk is a signal to be monitored, then clk'EVENT returns TRUE when a change on clk occurs (rising or falling edge).




```
1  ------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------
5  ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END dff;
9  -------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12    PROCESS (clk, rst)
13    BEGIN
14       IF (rst='1') THEN
15          q <= '0';
16       ELSIF (clk'EVENT AND clk='1') THEN
17          q <= d;
18    END IF;
19    END PROCESS;
20 END behavior;
21 -------------------------------------
```

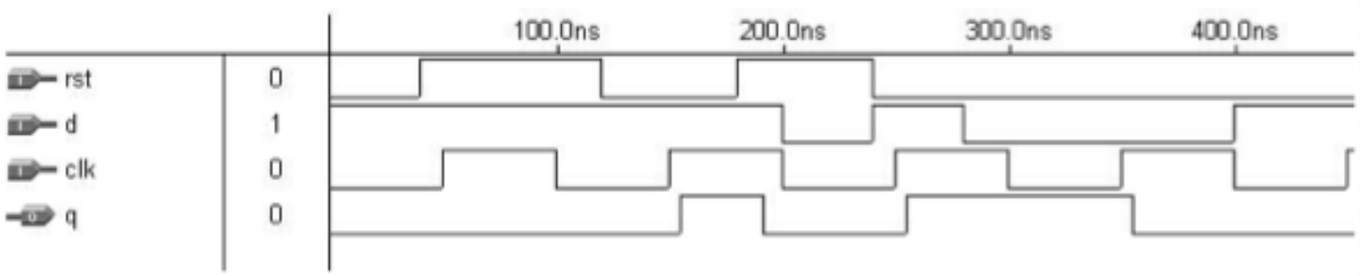
Constructing a synchronous circuit

- A D-type flip-flop (DFF) is the most basic building block in sequential logic circuits. In it, the output must copy the input at either the positive or negative transition of the clock signal (rising or falling edge).
- Here we use the IF statement (to be discussed later) to design a DFF with asynchronous reset. If rst='1', then the output must be q='0' (lines 14–15), regardless of the status of clk.




```
1  ------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------
5  ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END dff;
9  -------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12    PROCESS (clk, rst)
13    BEGIN
14       IF (rst='1') THEN
15          q <= '0';
16       ELSIF (clk'EVENT AND clk='1') THEN
17          q <= d;
18    END IF;
19    END PROCESS;
20 END behavior;
21 -------------------------------------
```

Constructing a synchronous circuit




```
1  ---------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END dff;
9  ---------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12    PROCESS (clk, rst)
13    BEGIN
14       IF (rst='1') THEN
15          q <= '0';
16       ELSIF (clk'EVENT AND clk='1') THEN
17          q <= d;
18    END IF;
19    END PROCESS;
20 END behavior;
21 ---------------------------------------
```

Introduction to *signals* and *variables*

- VHDL has two ways of passing non-static values around: SIGNAL or by a VARIABLE.
- A SIGNAL can be declared in the declarative part of the architecture (or inside a block), while a VARIABLE can only be declared inside a piece of sequential code. Therefore, a signal can be be global, a variable is always local.
- What is the scope of a, b and c?

```
entity test is
   port (clk : in bit);
end entity;

architecture myarch of test is
signal b : bit;
begin
  b1: block
  signal a : bit;
  begin
      a<='0';
  end block;

  process (clk)
  variable c : bit;
  begin
      b<='0';
  end process;

end architecture;
```

Declaring *signals* inside processes is not OK

- VHDL has two ways of passing non-static values around: SIGNAL or by a VARIABLE.

- A SIGNAL can be declared in the declarative part of the architecture (or inside a block), while a VARIABLE can only be declared inside a piece of sequential code. Therefore, a signal can be be global, a variable is always local.

- What is the scope of a, b and c?

```
1   entity test is
2       port (clk : in bit);
3   end entity;
4
5   architecture myarch of test is
6   signal b : bit;
7   begin
8       b1: block
9       signal a : bit;
10      begin
11          a<='0';
12      end block;
13
14      process (clk)
15      signal c : bit;
16      begin
17          b<='0';
18      end process;
19
20  end architecture;
21
```

Declaring *variables* outside processes is not OK

- VHDL has two ways of passing non-static values around: SIGNAL or by a VARIABLE.

- A SIGNAL can be declared in the declarative part of the architecture (or inside a block), while a VARIABLE can only be declared inside a piece of sequential code. Therefore, a signal can be be global, a variable is always local.

- What is the scope of a, b and c?

```
1   entity test is
2       port (clk : in bit);
3   end entity;
4
5   architecture myarch of test is
6   variable b : bit;
7   begin
8       b1: block
9       signal a : bit;
10      begin
11          a<='0';
12      end block;
13
14      process (clk)
15      variable c : bit;
16      begin
17          b<='0';
18      end process;
19
20  end architecture;
```

More on *variables*

- The value of a VARIABLE can never be passed out of the PROCESS directly; if necessary, then it must be assigned to a SIGNAL.
- On the other hand, the update of a VARIABLE is immediate, that is, we can promptly count on its new value in the next line of code.
- That is not the case with a SIGNAL (when used in a PROCESS), for its new value is generally only guaranteed to be available after the conclusion of the present run of the PROCESS.

Signal update inside a PROCESS

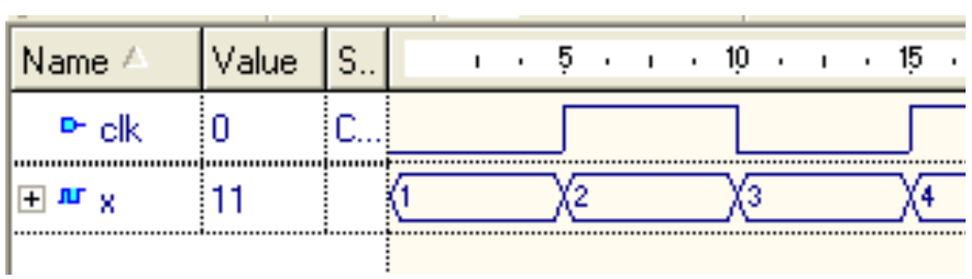
- A SIGNAL (when used in a PROCESS), is generally only guaranteed to be available after the conclusion of the present run of the PROCESS.
- Seriously? So what should x wave-form be?

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

entity test is
  port (clk : in bit);
end entity;

architecture myarch of test is
signal x : std_logic_vector(3 downto 0):="0000";
begin

  process (clk)
  begin
      x <= x+1;
      x <= x+1;
  end process;

end architecture;
```

Signal update inside a PROCESS

- A SIGNAL (when used in a PROCESS), is generally only guaranteed to be available after the conclusion of the present run of the PROCESS.

- Seriously? So what should x wave-form be?




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

entity test is
  port (clk : in bit);
end entity;

architecture myarch of test is
signal x : std_logic_vector(3 downto 0):="0000";
begin

  process (clk)
  begin
      x <= x+1;
      x <= x+1;
  end process;

end architecture;
```

Assignment operator

- The assignment operator for a SIGNAL is “<=”
- The assignment operator for a VARIABLE it is “:=”

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   entity test is
6       port (clk : in bit);
7   end entity;
8
9   architecture myarch of test is
10  signal x : std_logic_vector(3 downto 0):="0000";
11  begin
12
13      process (clk)
14      variable c : std_logic_vector(3 downto 0);
15      begin
16          x <= x+1;
17          c := c+1;
18      end process;
19
20  end architecture;
21
```

IF statement

- As mentioned earlier, IF, WAIT, CASE, and LOOP are the statements intended for sequential code.
- Therefore, they can only be used inside a PROCESS, FUNCTION, or PROCEDURE.

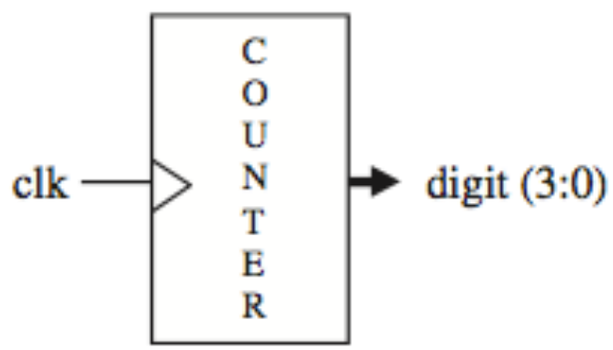
```
IF conditions THEN assignments;
ELSIF conditions THEN assignments;
...
ELSE assignments;
END IF;
```

Example:

```
IF (x<y) THEN temp:="11111111";
ELSIF (x=y AND w='0') THEN temp:="11110000";
ELSE temp:=(OTHERS =>'0');
```

Example: One-digit Counter

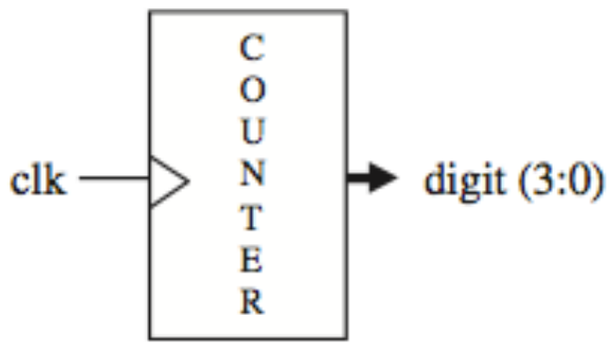
This code implements a progressive 1-digit decimal counter (0 to 9 to 0).

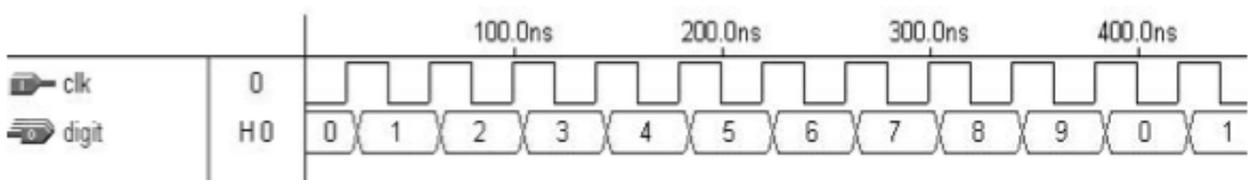



```
1  ---------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY counter IS
6     PORT (clk : IN STD_LOGIC;
7           digit : OUT INTEGER RANGE 0 TO 9);
8  END counter;
9  ---------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    count: PROCESS(clk)
13       VARIABLE temp : INTEGER RANGE 0 TO 10;
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          temp := temp + 1;
17          IF (temp=10) THEN temp := 0;
18          END IF;
19       END IF;
20       digit <= temp;
21    END PROCESS count;
22 END counter;
23 ---------------------------------------------
```

It contains a single input (clk) and a 4-bit output (digit). The IF statement is used in this example. A variable, temp, was employed to create the four flip-flops necessary to store the 4-bit output signal.

Example: One-digit Counter






```
1  ---------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY counter IS
6     PORT (clk : IN STD_LOGIC;
7           digit : OUT INTEGER RANGE 0 TO 9);
8  END counter;
9  ---------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    count: PROCESS(clk)
13       VARIABLE temp : INTEGER RANGE 0 TO 10;
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          temp := temp + 1;
17          IF (temp=10) THEN temp := 0;
18          END IF;
19       END IF;
20       digit <= temp;
21    END PROCESS count;
22 END counter;
23 ---------------------------------------------
```

Minor problems with this code:

- What is the initial value of temp?
- How can I initialize this circuit to start counting at zero?