Budapest University of Technology and Economics
Department of Electron Devices

Lab. 3
Sequential statements, Processes and Variables

Osama Ali

osamaalisalman.khafajy@edu.bme.hu

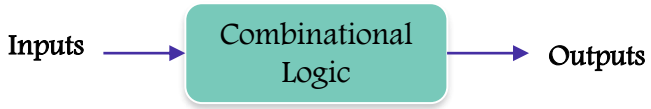
Ahmad Halal

ahmadhalalftesah@edu.bme.hu

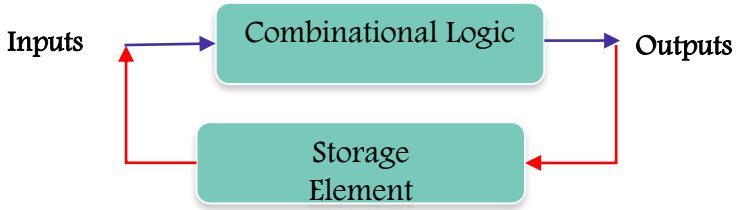
20 of May 2022

Sequential Statements

Can be used for building both combinational and sequential logic circuits.

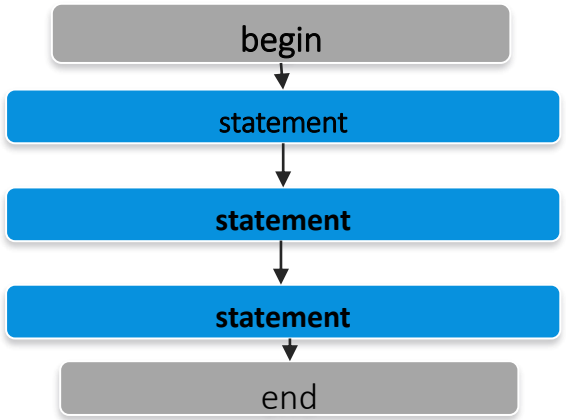


- The output of the circuit depends solely on the current inputs.
- The system requires no memory and can be implemented using conventional logic gates.



- The output does depend on previous inputs.
- A storage elements are required, which are connected to the combinational logic block through a feedback loop.

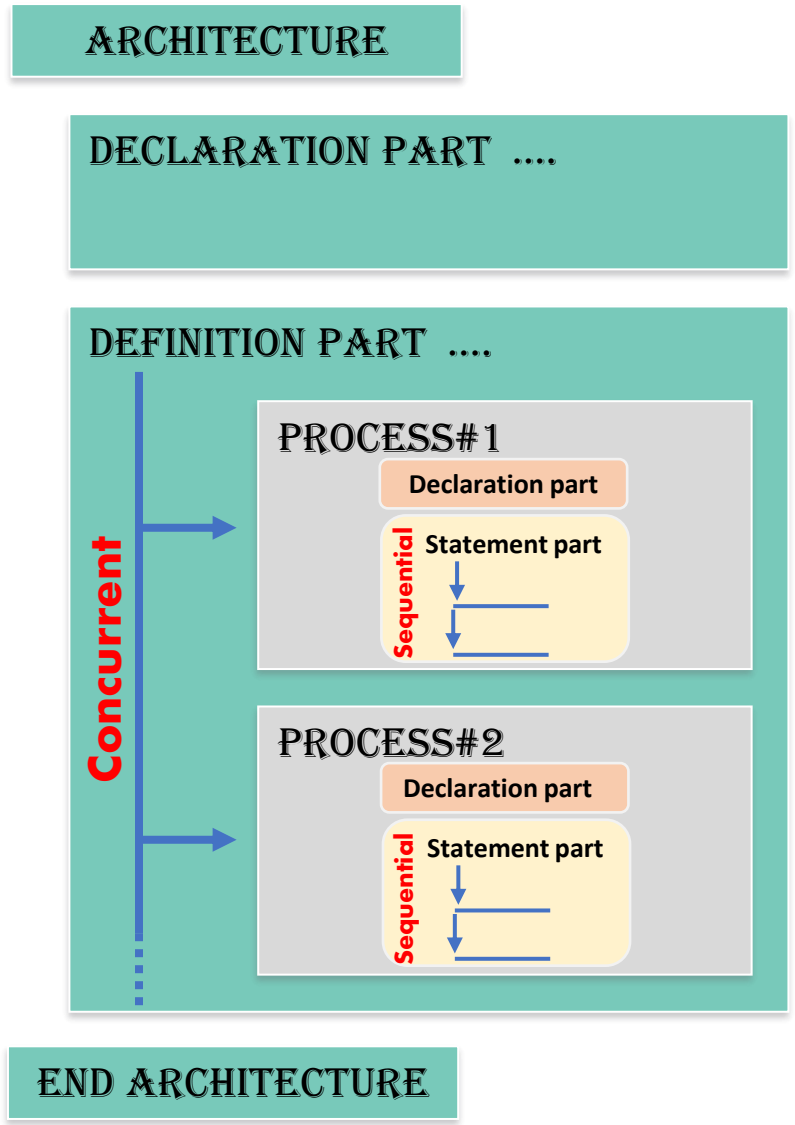
Unlike the concurrent statements, the sequential statements are executed line by line.



This is done by enclosing the sequential statements inside a VHDL construct known as a "**process**".
A "process" can appear anywhere after the "begin" statement of the "architecture".

Properties of process Statement

- Process statement contains only sequential statements.
- The process statement is itself a concurrent statement.
- A process statement has a declaration section and a statement part.
- In the declaration section, types, variables, constants, subprograms, and so on can be declared.
- The statement part contains only sequential statements.
- All processes in an architecture behave concurrently.



Sensitivity List

- The process statement can have an explicit sensitivity list that causes the statements inside the process statement to execute whenever one or more elements of the list change value.
- The process must have an explicit sensitivity list or, a WAIT statement. **OR…?**
- The sensitivity list **should** contain all input signals used in that process.

```
process (A,B)
begin
    if (A='1' or B='1') then
        Z <= '1';
    else
        Z <= '0';
    end if;
end process;
```

```
process
begin
    if (A='1' or B='1') then
        Z <= '1';
    else
        Z <= '0';
    end if;
    wait on A,B;
end process;
```

Signal Assignment **Versus Variable** Assignment

Signal

```
Architecture beh_1 of test is
    signal A,B,C: integer;
    signal Y, Z : integer;
    signal M, N : integer;
begin
    process (A,B,C,M,N)
    begin
        M <= A;
        N <= B;
        Z <= M + N;
        M <= C;
        Y <= M + N;
    end process;
end Architecture;
```

Variable

```
Architecture beh_1 of test is
    signal A,B,C: integer;
    signal Y, Z : integer;
begin
    process (A,B,C)
        variable M, N: integer;
    begin
        M := A;
        N := B;
        Z <= M + N;
        M := C;
        Y <= M + N;
    end process;
end Architecture;
```

If Statement

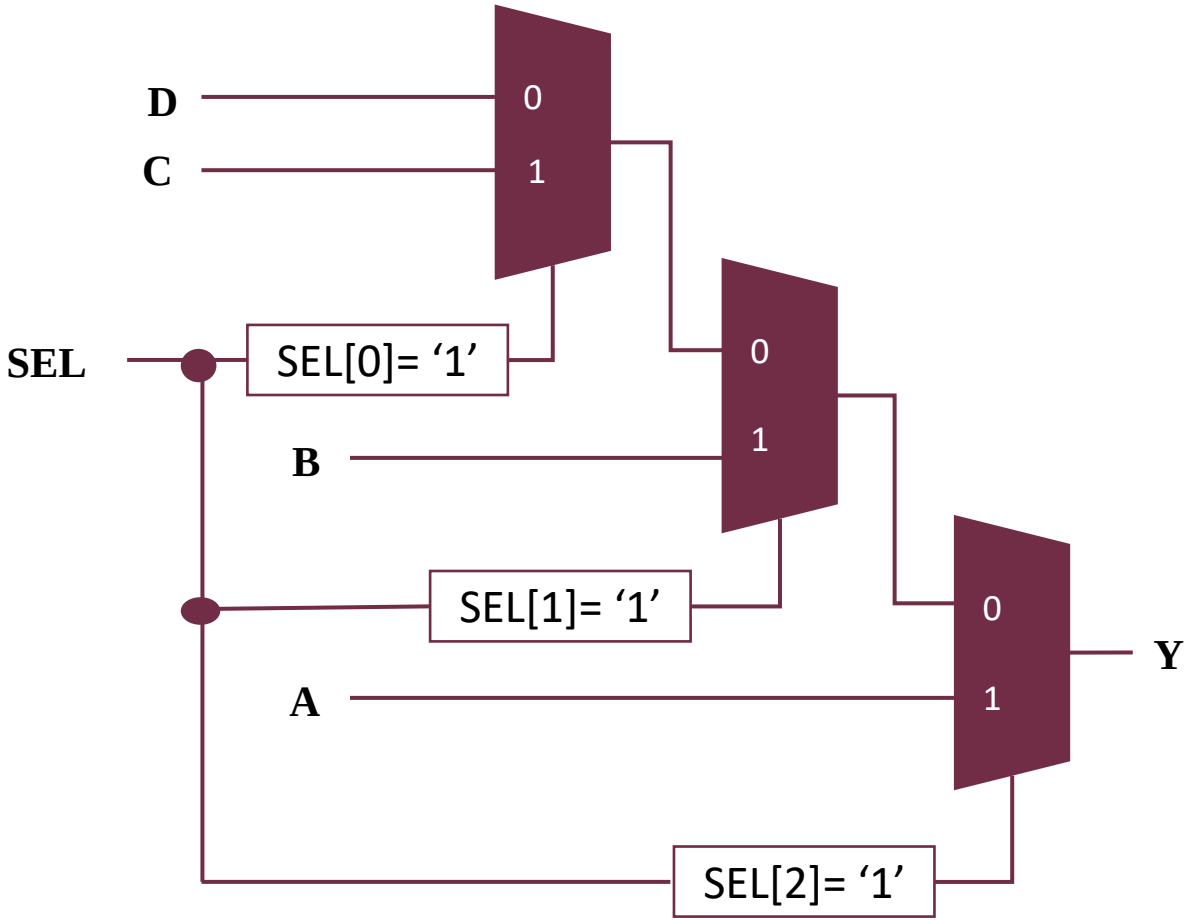
- Syntax:

```
if (condition 1) then

elsif (condition 2) then

elsif (condition 3) then


else

end if;
```

- "IF" statement evaluates each condition in order
- Statement can be nested
- **Spaghetti code (**Avoid using more than three levels of if…else statements)
- When defining the condition, use parentheses to differentiate levels of operations on the condition

If Statement

```
process (sel, a, b, c, d)
begin
if sel(2) = '1' then
    y <= a;
elsif sel(1) = '1' then
    y <= b;
elsif sel(0) = '1' then
    y <= c;
else
    y <= d;
end if;
end process
```



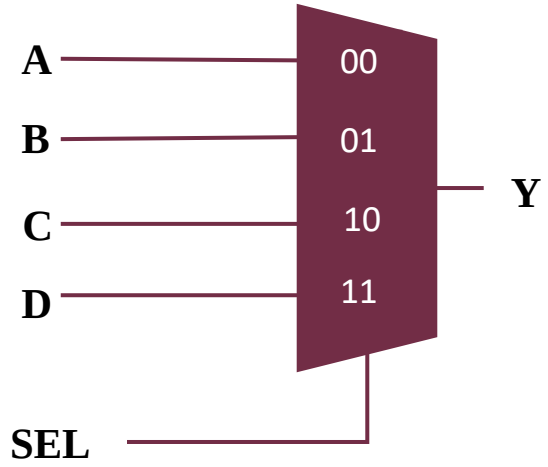
Generates a priority structure.
Corresponds to “when-else” command in the concurrent part.

Case Statement

- Syntax:

```
case expression is
        when choice1 => {statements}
        when choice2 => {statements}
    when others  =>  {statements}
End case;
```

```
process (sel, a, b, c, d)
begin
case sel is
    when "00" =>   Y <=a;
    when "01" =>   Y <=b;
    when "10" =>   Y <=c;
  when others =>   Y <=d;
end case;
end process;
```



- "Case" statement is a series of parallel checks to check a condition.
- Satements following each "when" clause is evaluated only if the choice value matches the expression value.
- Correspononds to "with…select" in concurrent statements

Process statement

- Two types of processes
  - Combinatorial (asynchronous)
  - Clocked (synchronous)

Combinatorial Process (asynchronous)

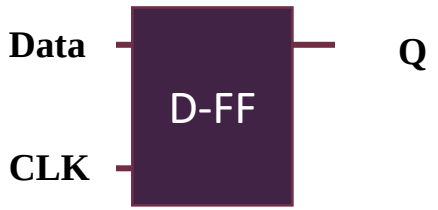
- Generates combinational logic.
- All inputs must be present in the sensitivity list.

```
process (a, b, c)
begin
     x <= (a and b) or c;

end process;
```

Clocked Process (synchronous)

Any signal assigned under a clk' event generates a flip-flop

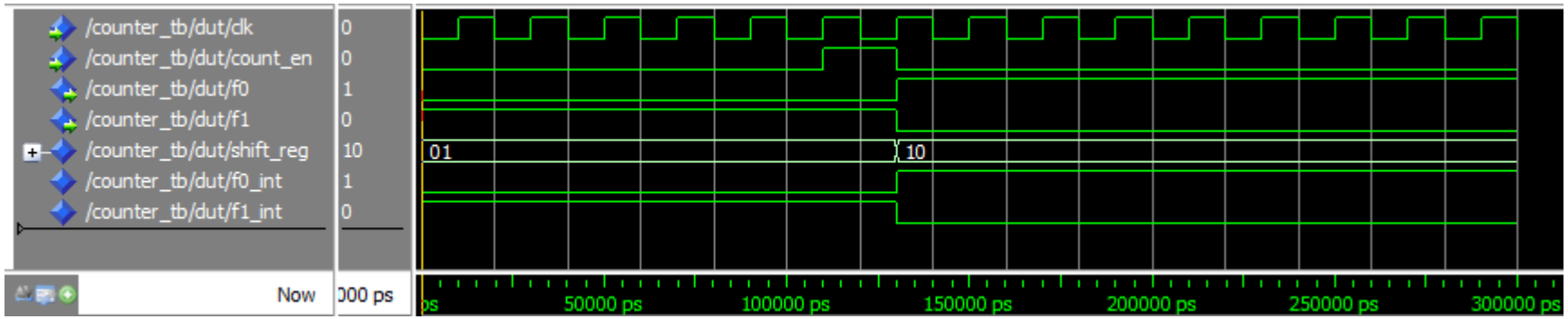
```
process (clk)
begin
    if (clk' event and clk = '1') then
        Q <= Data;
    end if;
end process;
```



- Clocked processes having an “else” clause will generate wrong hardware.

Bad Hardware Description

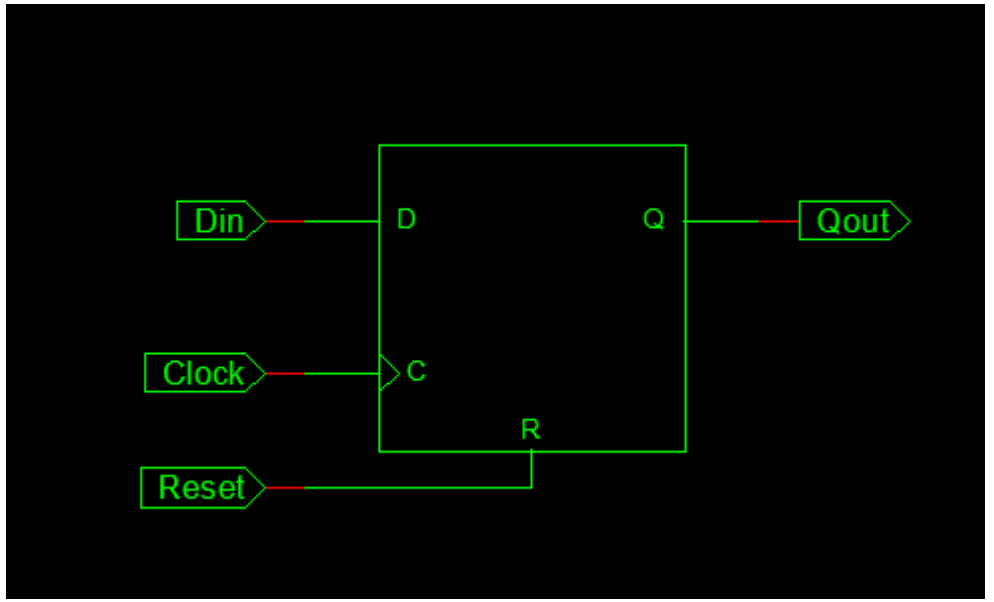
```
process (clk)
begin
    if (clk' event and clk = '1') then
        out1 <= a and b;
    else
        out1 <= c;
    end if;
end process;
```



Synchronous

- Synchronous Reset: Flip-flops are reset on the active edge of the clock when reset is held active.

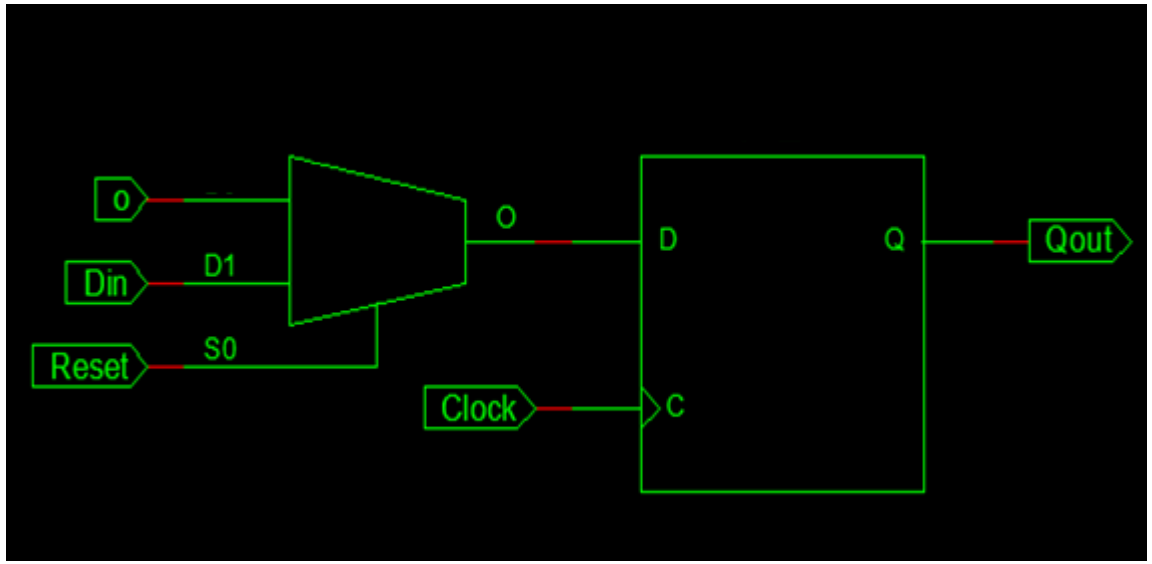
```
process (clk)
begin
    if (clk' event and clk = '1') then
        if (Reset = '1') then
            Q <= '0';
        else
            Q <= D;
        end if;
    end if;
end process;
```



Asynchronous

- Asynchronous Reset: Flip-flops are cleared as soon as reset is asserted.

```
PROCESS (clk, rst)
BEGIN
    IF (clk'EVENT AND clk='1') THEN
      IF (rst='1') THEN
        q1 <= '0';
      else  q1 <= d;

    END IF;
    END IF;
END PROCESS;
```

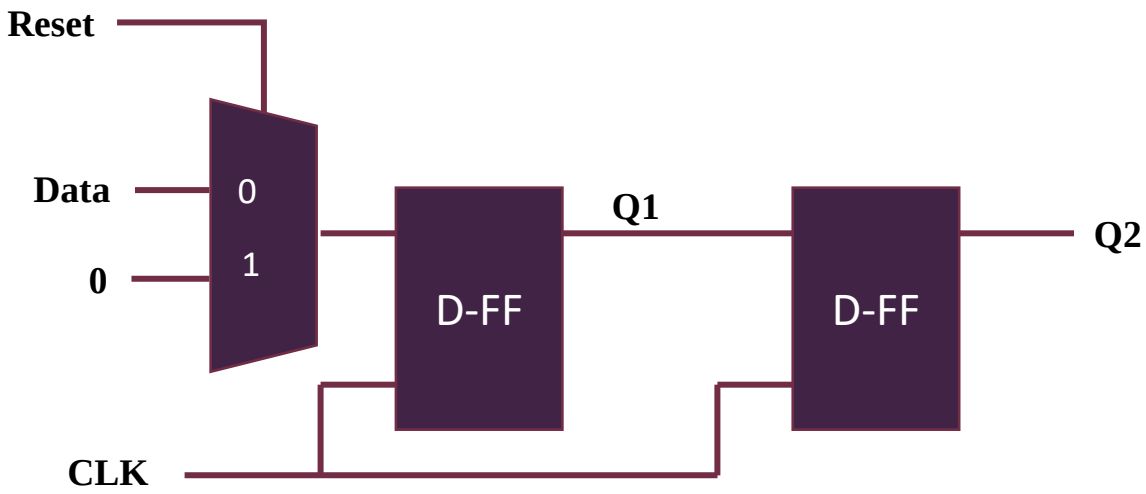


Clock generation in the testbench

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

entity Lab_4 is
end Lab_4;

architecture TEST_behavior of Lab_4 is

-- Insert component declaration for device under test (DUT) here.


-- Insert  signal declarations here.

SIGNAL CLKK : STD_LOGIC := '0';
SIGNAL RST : STD_LOGIC := '0';
SIGNAL D : STD_LOGIC := '0';


begin
-- Clock Generation
CLKK <= NOT CLKK AFTER 10 NS;
-- Insert component instantiation (i.e. port map statement) here.


   stimulus: process
   begin

   -- Insert your testbench code here.

   end process;

end TEST behavior;
```

Write a VHDL code that can model the below Circuit of D flip-flop using behavioral modeling.

- The circuit consists of two D_flip-flops one with synchronous reset and one without.
- Create a new project and label it as “Lab_5_1”
- Download the VHDL testbench file (TestBench_Lab5.vhd.). Develop it to test and analyze the design behavior.



Write a VHDL code that can model the below Circuit of D flip-flop using behavioral modeling.

- The circuit consists of two D_Flip-flops one with Asynchronous reset and one without.
- Create a new project and label it as “Lab_5_2”
- Download the VHDL testbench file (TestBench_Lab5.vhd.). Develop it to test and analyze the design behavior.