UNIVERSITÀ DEGLI STUDI DI TRIESTE

Dipartimento di
Fisica
Dipartimento d'Eccellenza 2023-2027

VHDL
Part 2

Laboratorio di acquisizione e controllo dati

UniTs a.a. 2025-2026

Models in VHDL architecture

- The behavior of the digital circuit is modelled in the VHDL architecture through concurrent statements

- There are three main approaches to model a circuit in VHDL architecture
  - **Data-flow** style
  - **Behavioral** style
  - **Structural** style (not for this course)

Data flow style

- Describes the circuit as a **representation of how the data flows in it**
- Statements describe the relationship between **input** and **output** signals with VHDL **operators** (e.g. AND, OR, NOT)
- Uses **concurrent**, **conditional**, **selected** signal assignment statements
- All the exercises seen up to now are data flow style architectures
- Works well for **small circuits** where you have an **idea of the underlying hardware**

Behavioral style

- Higher circuit abstraction level
- Models how the **circuit should behave**
- Leaves the details of the **implementation** to the **synthesis tool**
- Used for **complex circuits**
- Uses **process** statement

Process statement

```
-- PROCESS STATEMENT
    -- Within a process statement, all statements are executed sequentially
    -- The process itself is a concurrent statement and will be executed concurrently with other concurrent statements in the architecture
    -- The process statement is used when some commands need to be executed in a sequential manner

    -- Process has an identifier, a sensitivity list, a declaration part and an implementation part containing sequential statements
        -- Syntax for process statement
    <identifier> : process (sensitivity_list) is
        <item_declaration>
     begin
        <sequential_statements>
     end process <identifier>;
```

Data flow vs behavioral

```
-- EXAMPLE: OR GATE DATA FLOW MODEL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity or_gate_df is
Port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Y : out STD_LOGIC
);
end or_gate_df;

architecture dataflow of or_gate_df is

begin
Y <= A or B;
    -- concurrent signal assignment statement
    -- the statement in the data flow style architecture is re-evaluated any time there is a change in signal A or signal B

end dataflow;


-- EXAMPLE: OR GATE BEHAVIORAL MODEL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity or_gate_b is
Port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Y : out STD_LOGIC
);
end or_gate_b;

architecture behavioral of or_gate_b is

begin

    or_proc: process (A,B) is
    begin
        Y <= A or B;
        -- sequential signal assignment statement
            -- same syntax as of the concurrent signal assignment statement
            -- it is sequential because it appears inside a process statement
        -- execution of the statement in the behavioral style architecture is controlled by which signals appear in the process sensitivity list
    end process or_proc;

end behavioral;
```

Sequential statements

- There are three types of sequential statements
  - **Sequential** signal assignment statement (see previous slide)
  - **if** statement
  - **case** statement

if statement

```
-- IF STATEMENT
    -- Sequential equivalent of the conditional signal assignment statement

    -- A sequence of statements is executed if an associated condition is true
        -- Depending on the conditions, the statements associated with one or none of the branches is executed

    -- Syntax for if statement
        if <condition> then
            <statements>
        elsif <condition> then
            <statements>
        else
            <statements>
            -- the final "else" clause does not have an associated "then" keyword
            -- the final "else" clause is an optional catch-all
                -- if none of the previous conditions is evaluated as true, then this statement is executed
                -- unless you want to implement a memory function, you must use the else clause!!!
                -- we will discuss this more in the D-FF exercise
end if;
```

case statement

```
-- CASE STATEMENT
    -- sequential equivalent of the selected signal assignment statement

    -- a sequence of statements is executed if an associated expression is true
    -- the assignment is made depending upon the value of the single control expression
    -- only one set of sequential statements is executed for each execution of the case statement
    -- the statements executed are determined by the first when branch to evaluate as true

case <expression> is
    when <choices> =>
        <sequential_statements>
    when <choices> =>
        <sequential_statements>
    when others =>
        <sequential_statements>
        -- "when others" is an optional catch-all, but it is good practice to include it
end case;
```

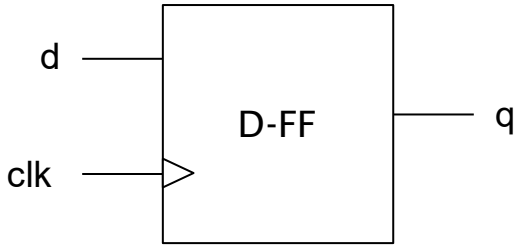
VHDL for sequential circuits

- All examples seen so far are **combinational** circuits
  - They produce outputs based **only on the current inputs**

- A **sequential** circuit is a type of digital logic circuit whose output depends not only on the **current inputs** but also on the **previous states** of the circuit
  - Sequential circuits have **memory elements** (e.g. flip-flop) that store information about previous states
  - They are often controlled by a **clock signal** that synchronizes state changes

- In VHDL sequential circuits are typically modelled with **behavioral style** architecture
  - In general, a certain logic circuit can be modelled in different ways, that will result in the same hardware
  - But for synchronous memory elements, behavioral style is the optimal method

D-FF




```vhdl
-- EXAMPLE: D-FF

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_flip_flop is
    Port (
        clk : in  STD_LOGIC;
        d   : in  STD_LOGIC;
        q   : out STD_LOGIC
    );
end d_flip_flop;

architecture Behavioral of d_flip_flop is

begin
    process(clk)
        -- the statements within process are executed every time there is a change in the logic level of clk
    begin
        if rising_edge(clk) then
            -- "rising_edge() is a VHDL function
            -- changes in the circuit happen only on the rising egde of the clk input

            -- THIS WAY OF WRITING THE CODE IS WHAT MAKES THE D-FF OUTPUT SYNCHRONOUS TO THE CLOCK INPUT!!! ---

            q <= d;
        end if;
        -- a condition that indicates what should happen if the listed if condition is not met, is not provided
        -- if the if condition is not met, the D-FF does not chnage the value of q
        -- the D-FF must remember the current value
        -- THIS IS THE WAY TO INDUCE MEMORY IN VHDL!!!
    end process;

end Behavioral;

```

Test bench of D-FF

```vhdl
-- EXAMPLE: TEST BENCH OF D-FF
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_flip_flop_tb is
end d_flip_flop_tb;

architecture Behavioral of d_flip_flop_tb is

    -- Component declaration
    component d_flip_flop
        Port (
            clk : in  STD_LOGIC;
            d   : in  STD_LOGIC;
            q   : out STD_LOGIC
        );
    end component;

    -- Testbench signals
    signal clk_tb        : STD_LOGIC := '0';
    signal d_tb          : STD_LOGIC := '0';
    signal q_tb          : STD_LOGIC;
    signal stop_clock_tb : BOOLEAN := false; -- note BOOLEAN type, not seen before

    -- Clock period constant
    constant CLK_PERIOD : time := 10 ns;
    -- synthax for constant declaration
            -- constant <name> : type := <initial value>;
        -- "time" is a data type used in simulations
            -- FPGA does not have the concept of time
        -- the constant is given a value at declaration that cannot be changed afterwards

```

Test bench of D-FF

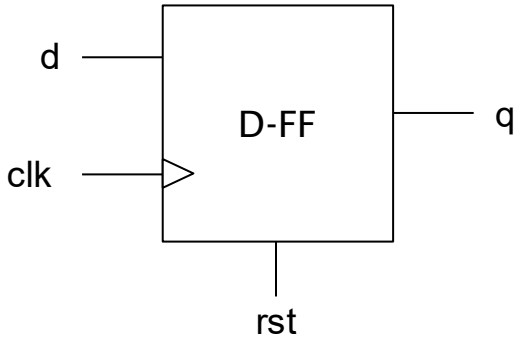
```
33 begin
34
35     -- DUT instantiation
36     uut: d_flip_flop
37         port map (
38             clk => clk_tb,
39             d   => d_tb,
40             q   => q_tb
41         );
42
43     -- Clock generation
44     clk_process : process
45     begin
46         while not stop_clock_tb loop     -- we do not discuss the while loop construct in this course
47                                          -- refer to the suggested book
48             clk_tb <= '0';
49             wait for CLK_PERIOD/2;
50             clk_tb <= '1';
51             wait for CLK_PERIOD/2;
52         end loop;
53         wait;
54     end process;
55
56     -- Stimulus process
57     stim_proc : process
58     begin
59
60         wait for 12 ns;
61         d_tb <= '1';
62
63         wait for 10 ns;
64         d_tb <= '0';
65
66         wait for 10 ns;
67         d_tb <= '1';
68
69         wait for 10 ns;
70         d_tb <= '0';
71
72         wait for 20 ns;
73         stop_clock_tb <= true;  -- stop the clock
74
75         wait;
76     end process;
77
78 end Behavioral;
```

Exercise 6

- Simulate the D-FF in EDA playground

Exercise 7

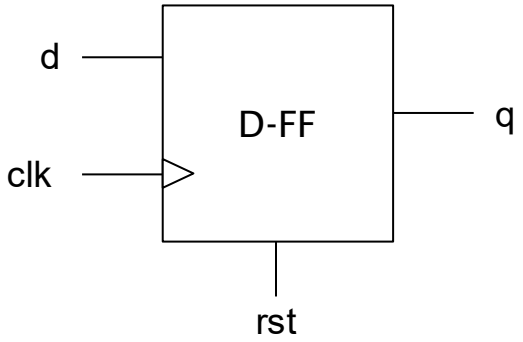
- Write the code for a **D-FF** with **asynchrounous reset** and simulate it in EDA playground



- Instructions
  - Use behavioral style architecture with process statement
  - Use the if statement
  - **For sequential logic, a process can have at most two signals in the sensitivity list**
    - **Clock and reset, if the reset is asynchrououns**
    - **Clock, if there is no reset or the reset is synchronous**

Exercise 8

- Write the code for a **D-FF** with **synchrounous reset** and simulate it in EDA playground



- Instructions
  - Use behavioral style architecture with process statement
  - Use the if statement
    - Tip: if statements can be nested

Component declaration

```
-- EXAMPLE: TEST BENCH OF D-FF
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_flip_flop_tb is
end d_flip_flop_tb;

architecture Behavioral of d_flip_flop_tb is

    -- Component declaration
    component d_flip_flop
        Port (
            clk : in  STD_LOGIC;
            d   : in  STD_LOGIC;
            q   : out STD_LOGIC
        );
    end component;

    -- Testbench signals
    signal clk_tb         : STD_LOGIC := '0';
    signal d_tb           : STD_LOGIC := '0';
    signal q_tb           : STD_LOGIC;
    signal stop_clock_tb : BOOLEAN := false; -- note BOOLEAN type, not seen before

    -- Clock period constant
    constant CLK_PERIOD : time := 10 ns;
    -- synthax for constant declaration
            -- constant <name> : type := <initial value>;
        -- "time" is a data type used in simulations
            -- FPGA does not have the concept of time
        -- the constant is given a value at declaration that cannot be changed afterwards

begin

    -- DUT instantiation
    uut: d_flip_flop
        port map (
            clk => clk_tb,
            d   => d_tb,
            q   => q_tb
        );

    -- Clock generation
    clk_process : process
    begin
        while not stop_clock_tb loop    -- we do not discuss the while loop construct in this course
                                        -- refer to the suggested book
            clk_tb <= '0';
            wait for CLK_PERIOD/2;
            clk_tb <= '1';
            wait for CLK_PERIOD/2;
        end loop;
        wait;
    end process;

    -- Stimulus process
    stim_proc : process
    begin

        wait for 12 ns;
        d_tb <= '1';

        wait for 10 ns;
        d_tb <= '0';

        wait for 10 ns;
        d_tb <= '1';

        wait for 10 ns;
        d_tb <= '0';

        wait for 20 ns;
        stop_clock_tb <= true;  -- stop the clock

        wait;
    end process;

end Behavioral;
```

- Here we have a module called “d_flip_flop_tb”
- Within this module we use another module “d_flip_flop”
- To do this we used this syntax

```
-- Component declaration in the declarative region of your architecture
component <component_name>
port (
    <port_0_name> : <mode> <type>;
    ...
    <port_n_name> : <mode> <type>
);
end component;
```

```
-- Component instantiation in the implementation region of the architecture
<instantiation_name> : <component_name>
port map (
<component_port_0_name> = > <parent_module_signal>,
...
<component_port_n_name> = > <parent_module_signal>,
);
```

Entity declaration

- We could have done the same (use a module within a module) with this syntax

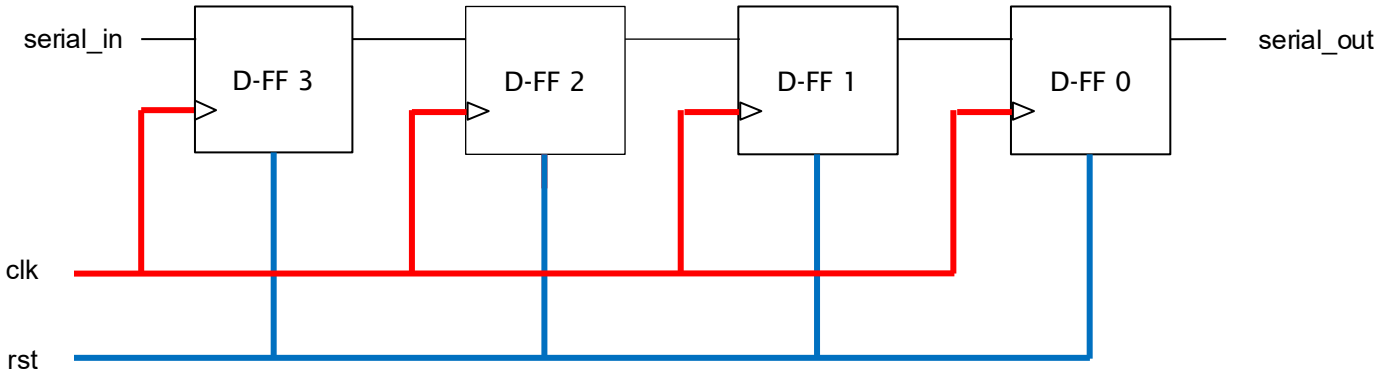
```
-- Entity instantiation in the implementation region of the architecture
<instantiation_name>: entity work.<entity_name> -- work is a special keyword in VHDL
                                                -- it always refers to the same library as the main module
                                                -- main module and instantiated module shoudl be in the same library

port map (
<entity_port_0_name> = > <parent_module_signal>,
...
<entity_port_n_name> = > <parent_module_signal>,
);
```

```
1  -- EXAMPLE: TEST BENCH OF D-FF
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity d_flip_flop_tb is
6  end d_flip_flop_tb;
7
8  architecture Behavioral of d_flip_flop_tb is
9
10     -- Testbench signals
11     signal clk_tb        : STD_LOGIC := '0';
12     signal d_tb          : STD_LOGIC := '0';
13     signal q_tb          : STD_LOGIC;
14     signal stop_clock_tb : BOOLEAN := false; -- note BOOLEAN type, not seen before
15
16     -- Clock period constant
17     constant CLK_PERIOD : time := 10 ns;
18     -- synthax for constant declaration
19          -- constant <name> : type := <initial value>;
20         -- "time" is a data type used in simulations
21          -- FPGA does not have the concept of time
22         -- the constant is given a value at declaration that cannot be changed afterwards
23
24 begin
25
26     -- DUT instantiation
27     uut: entity work.d_flip_flop -- work is a special keyword in VHDL
28                                  -- it always refers to the same library as the main module
29                                  -- main module and instantiated module shoudl be in the same library
30         port map (
31             clk => clk_tb,
32             d   => d_tb,
33             q   => q_tb
34         );
35
36     -- Clock generation
37     clk_process : process
38     begin
39         while not stop_clock_tb loop     -- we do not discuss the while loop construct in this course
40                                          -- refer to the suggested book
41             clk_tb <= '0';
42             wait for CLK_PERIOD/2;
43             clk_tb <= '1';
44             wait for CLK_PERIOD/2;
45         end loop;
46         wait;
47     end process;
48
49     -- Stimulus process
50     stim_proc : process
```

Exercise 9

- Write the code for a **4-bit SISO shift register in VHDL** using D flip-flops, a clocked process, and asynchronous reset



- Instruction
  - Syntax to access individual bit of a bus
    - one_bit_signal <= my_bus(n)
    - one_bit_signal gets the value of the n^{th} bit in the bus called my_bus

Summary part 2

- VHDL modeling styles
  - **Data-flow** style
    - Uses concurrent, conditional, sequential signal assignment statements
  - **Behavioral** style
    - Uses process statements

- Process statements
  - Process is a concurrent statement
  - Statements appearing within process are sequential statements

- Sequential statements
  - **Sequential** signal assignment statement
  - **if** statement
  - **case** statement

Summary part 2

- Sequential circuits are modelled with behavioral style architecture

- Memory is induced in VHDL by not specifying output conditions for every possible input condition
  - So, unless you want to model a memory, always specify all conditions
  - Unintended generation of memory is flagged by the synthesis tool
  - Memory adds complexity to the synthesized circuit

- Constants
  - Declared in **declarative** part of architecture
  - Constants are assigned a value at declaration that cannot be changed afterwards
  - This is done using the **assignment operator :=**

Summary part 2

- Exercises
  - 6: D-FF
  - 7: D-FF with asynchronous reset
  - 8: D-FF with synchronous reset
  - 9: 4-bit SISO shift reset with asynchronous reset