Introduzione alle FPGA

Lab 2

Prof. Laura Gonella

Laboratorio di Acquisizione e Controllo Dati
a.a. 2024-25

Examples and exercises

- Example: OR gate
- Exercise 1: AND gate
- Exercise 2: 2to1 MUX

- Example: OR gate with “process”
- Example: D-FF
- Exercise 3: 2to1 MUX with “process” and “if” statement

- Exercise 4: D-FF with synchronous reset
- Exercise 5: Shift register

process statement

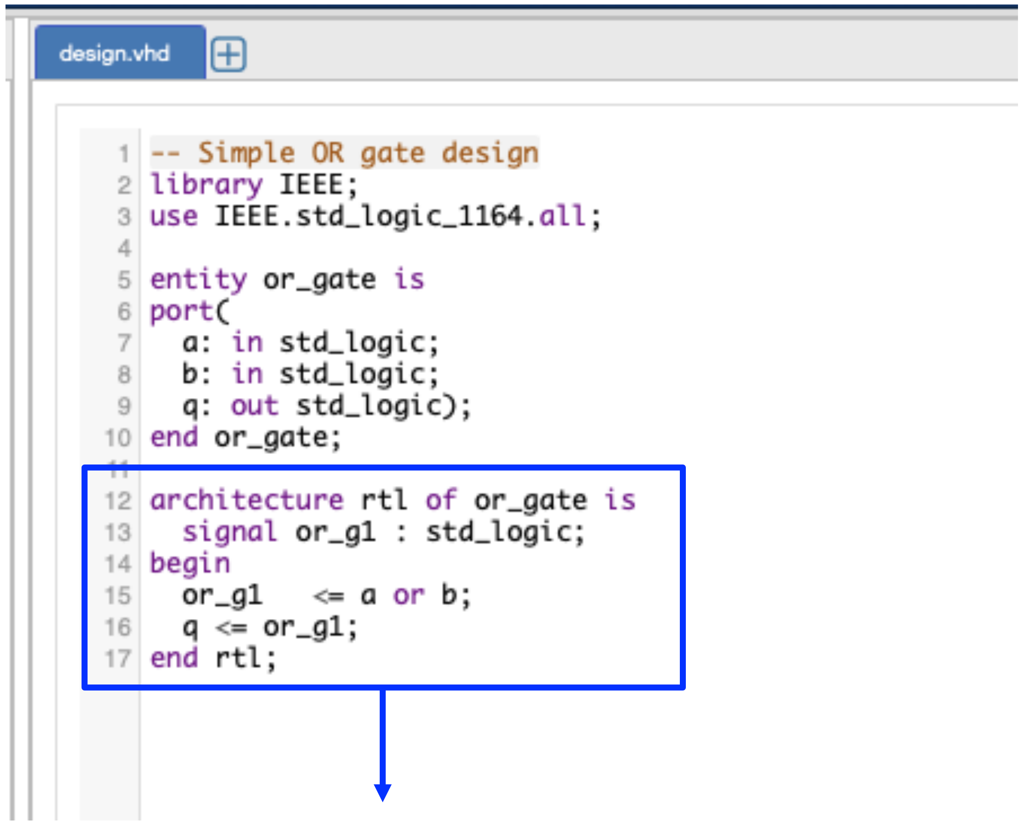
- The **process** statement is used in VHDL to define blocks to be evaluated sequentially
  - Statements inside a process are evaluated sequentially (like most programming languages)
  - Multiple process blocks are evaluated concurrently

- A process can have a sensitivity list
  - List of signal to which the process is sensitive (for example a clock)
  - The process is executed only when there is a change to a signal in the sensitivity list

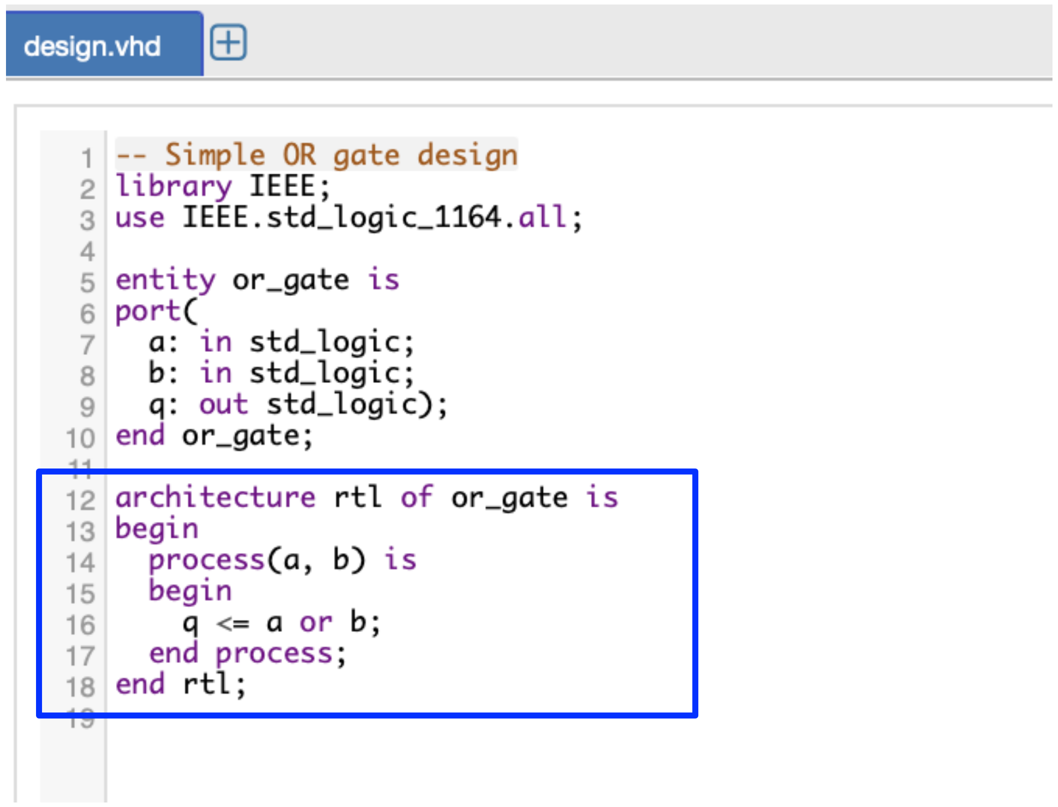
- Processes are used to define a block of combinational logic or a block of sequential logic – the latter is far more used

```
<process_name> : process(signalA, signalB)
begin
  statement 1;
  statement 2;
  ...
  statement N;
end process <process_name>;
```

Example: OR gate with process statement



```vhdl
-- Simple OR gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end or_gate;

architecture rtl of or_gate is
  signal or_g1 : std_logic;
begin
  or_g1    <= a or b;
  q <= or_g1;
end rtl;
```



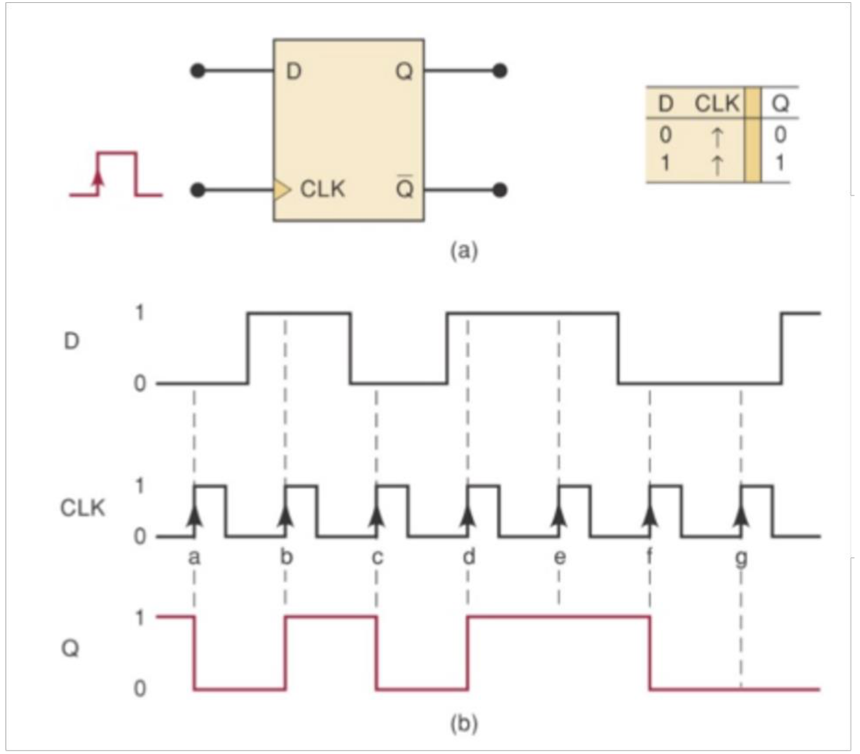
```vhdl
-- Simple OR gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end or_gate;

architecture rtl of or_gate is
begin
  process(a, b) is
  begin
    q <= a or b;
  end process;
end rtl;
```

Could have been written as:

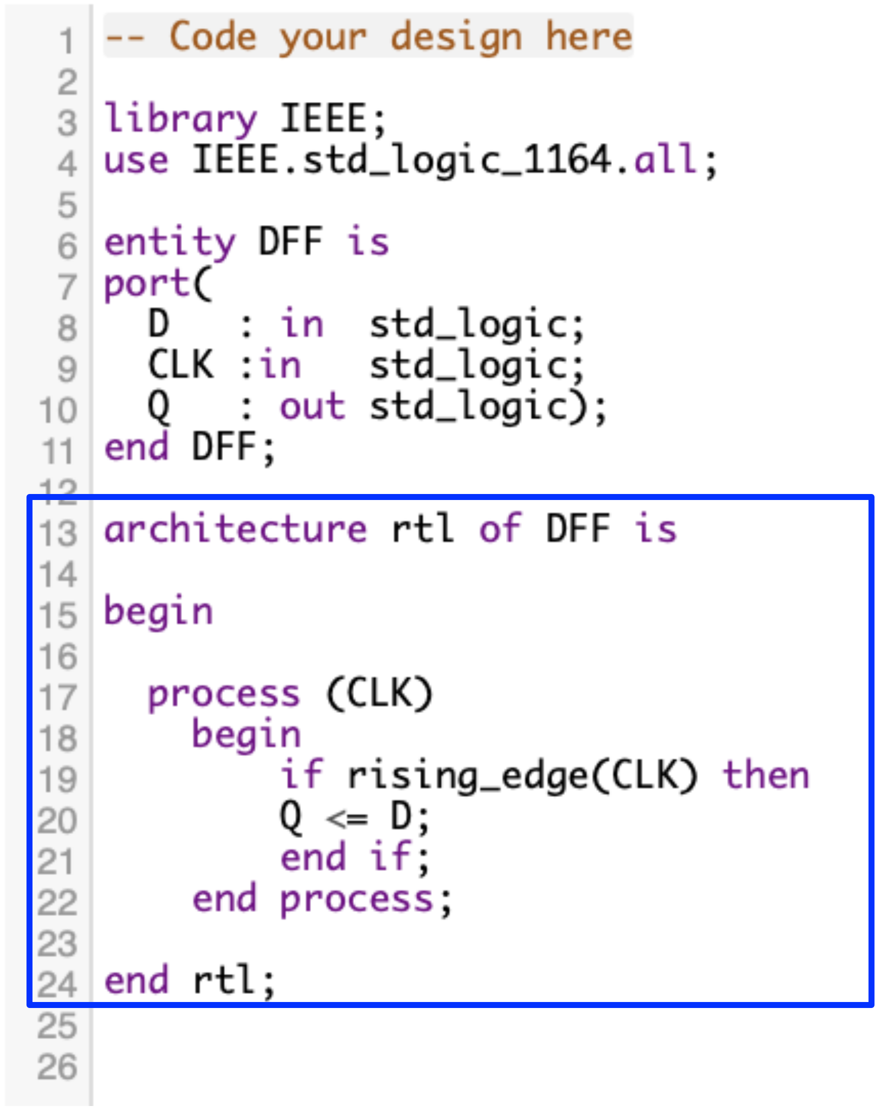
```vhdl
architecture rtl of or_gate is
  begin
    q <= a or b;
  end rtl
```

Example of sequential logic with process: D-FF design

```vhdl
-- Code your design here

library IEEE;
use IEEE.std_logic_1164.all;

entity DFF is
port(
  D   : in  std_logic;
  CLK :in   std_logic;
  Q   : out std_logic);
end DFF;

architecture rtl of DFF is

begin

  process (CLK)
    begin
        if rising_edge(CLK) then
        Q <= D;
        end if;
    end process;

end rtl;
```



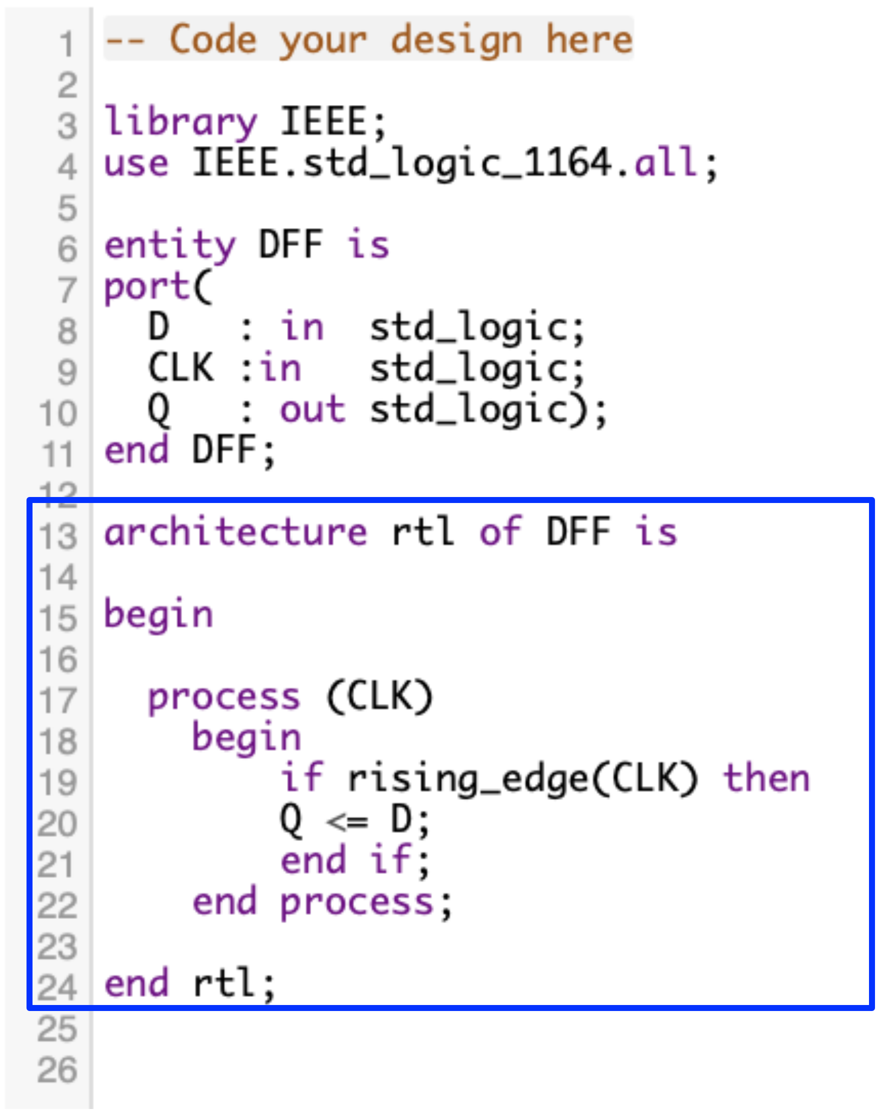
Edge detection



```vhdl
-- Code your design here

library IEEE;
use IEEE.std_logic_1164.all;

entity DFF is
port(
  D   : in  std_logic;
  CLK :in   std_logic;
  Q   : out std_logic);
end DFF;

architecture rtl of DFF is

begin

  process (CLK)
    begin
        if rising_edge(CLK) then
        Q <= D;
        end if;
    end process;

end rtl;
```

- **process** block
  - In this example, the sensitivity list contains the clock signal, CLK
    - The process is executed only when there is a change to the clock
- **rising_edge(CLK)**
  - functions to detect signal and clock edges in the **ieee.std_logic_1164.all** package
  - **rising_edge(s)** returns true, if there is a rising edge on the signal **s**
  - **falling_edge(s)** returns true, if there is a falling edge on the signal **s**

Conditional statement: if



```
-- Code your design here

library IEEE;
use IEEE.std_logic_1164.all;

entity DFF is
port(
  D   : in  std_logic;
  CLK :in   std_logic;
  Q   : out std_logic);
end DFF;

architecture rtl of DFF is

begin

  process (CLK)
    begin
      if rising_edge(CLK) then
        Q <= D;
      end if;
    end process;

end rtl;
```

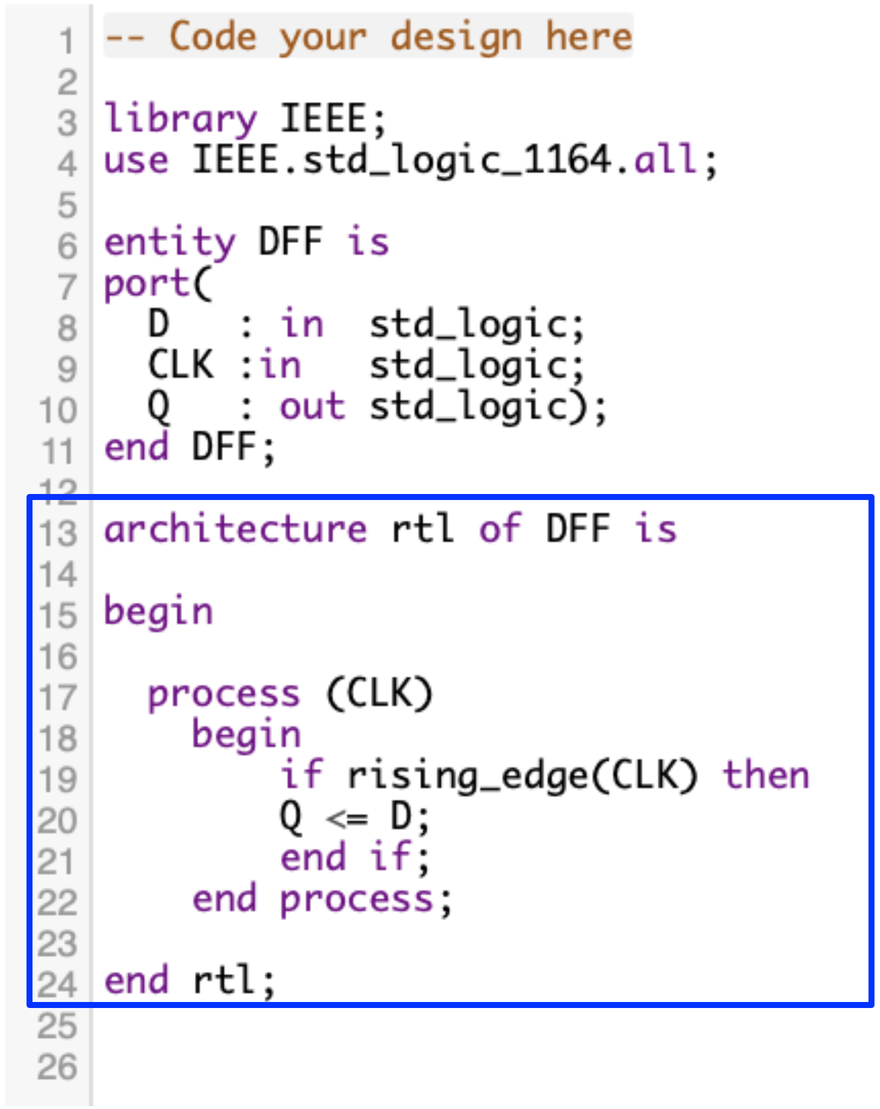
- Conditional statement: **if**
  - Conditional execution inside a process
  - Condition should return a Boolean (true/false)
  - Keywords: **if, elsif, and else**

```
if <condition1> then
  <vhdl statement>;
end if;
```

```
if <condition1> then
  <vhdl statement 1>;
else
  <vhdl statement 2>;
end if;
```

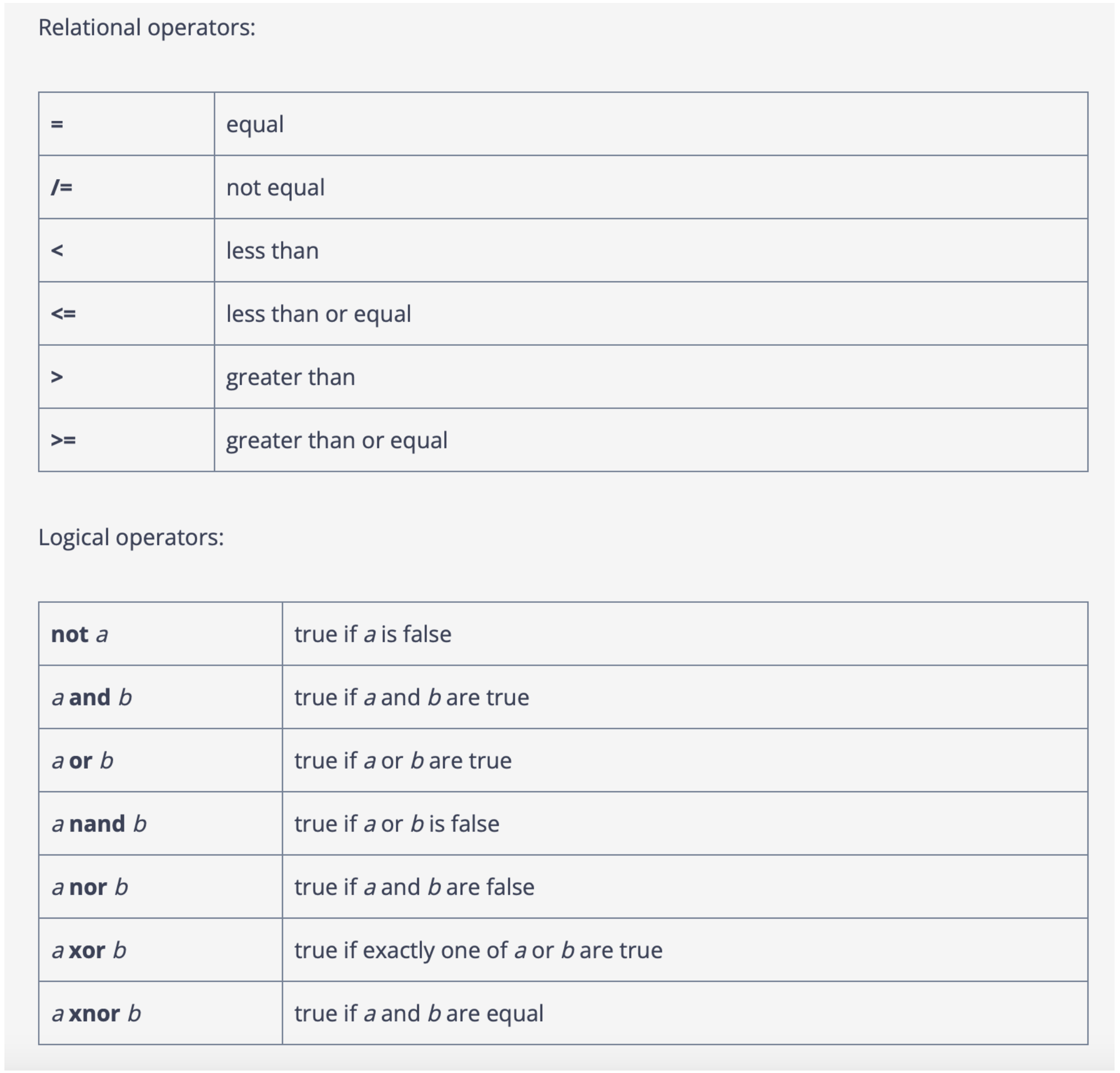
```
if <condition1> then
  <vhdl statement 1>;
elsif <condition2> then
  <vhdl statement 2>;
else
  <vhdl statement 3>;
end if;
```

Conditional statement: if



```
-- Code your design here

library IEEE;
use IEEE.std_logic_1164.all;

entity DFF is
port(
  D   : in  std_logic;
  CLK :in   std_logic;
  Q   : out std_logic);
end DFF;

architecture rtl of DFF is

begin

  process (CLK)
    begin
        if rising_edge(CLK) then
        Q <= D;
        end if;
    end process;

end rtl;
```

- Conditional statement: **if**



Relational operators:

=	equal
/=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

Logical operators:

not *a*	true if *a* is false
a **and** *b*	true if *a* and *b* are true
a **or** *b*	true if *a* or *b* are true
a **nand** *b*	true if *a* or *b* is false
a **nor** *b*	true if *a* and *b* are false
a **xor** *b*	true if exactly one of *a* or *b* are true
a **xnor** *b*	true if *a* and *b* are equal

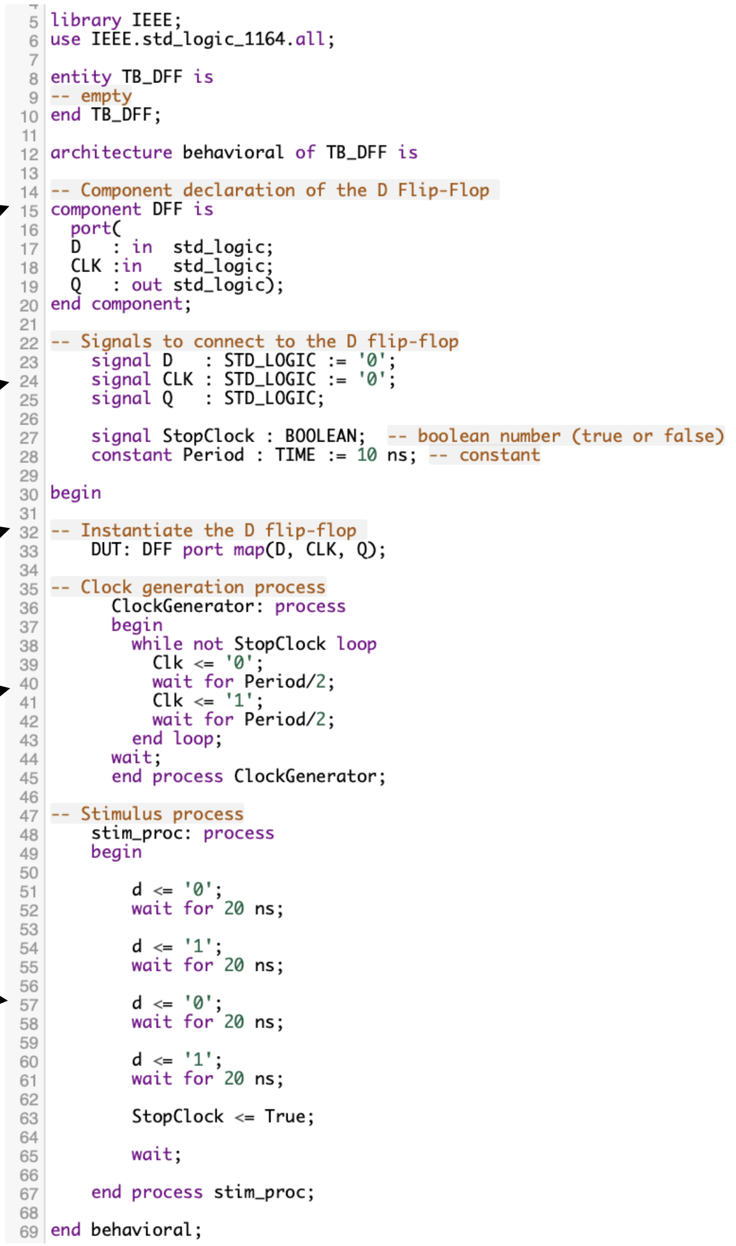
Example of sequential logic with process: D-FF testbench

- **Library**

- **Entity** (empty)

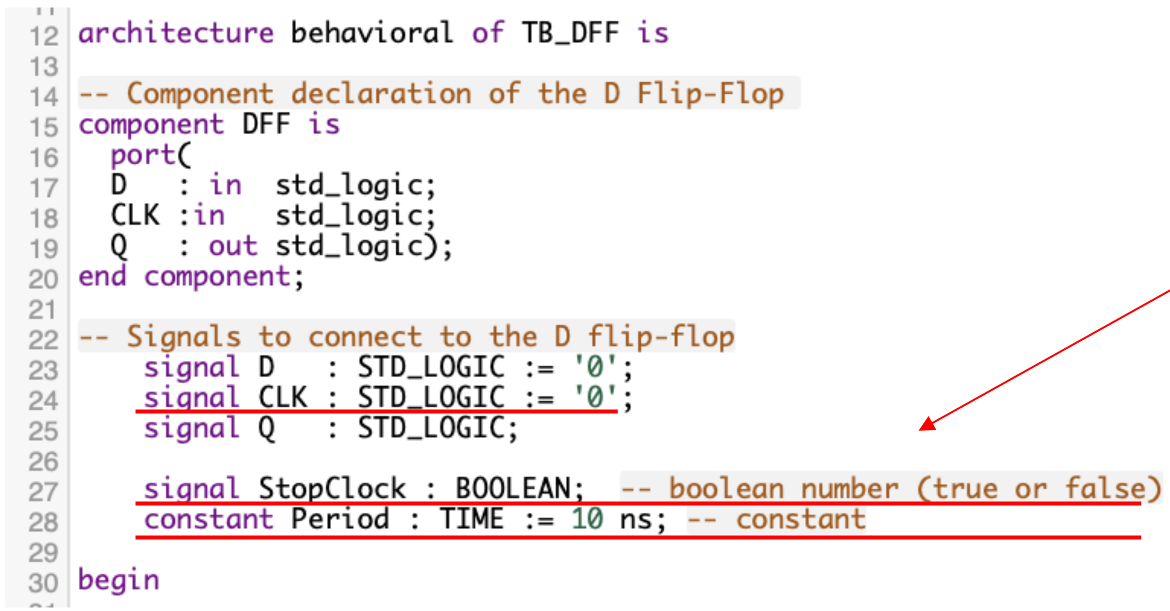
- **Architecture**
  - Declaritive part
    - Component declaration
    - Signal declaration

  - Implementation part
    - Component instantiation
    - Clock generation process
    - Stimulus process



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity TB_DFF is
-- empty
end TB_DFF;

architecture behavioral of TB_DFF is

-- Component declaration of the D Flip-Flop
component DFF is
  port(
  D   : in  std_logic;
  CLK :in   std_logic;
  Q   : out std_logic);
end component;

-- Signals to connect to the D flip-flop
    signal D   : STD_LOGIC := '0';
    signal CLK : STD_LOGIC := '0';
    signal Q   : STD_LOGIC;

    signal StopClock : BOOLEAN;  -- boolean number (true or false)
    constant Period : TIME := 10 ns; -- constant

begin

-- Instantiate the D flip-flop
    DUT: DFF port map(D, CLK, Q);

-- Clock generation process
      ClockGenerator: process
      begin
        while not StopClock loop
          Clk <= '0';
          wait for Period/2;
          Clk <= '1';
          wait for Period/2;
        end loop;
      wait;
      end process ClockGenerator;

-- Stimulus process
    stim_proc: process
    begin

        d <= '0';
        wait for 20 ns;

        d <= '1';
        wait for 20 ns;

        d <= '0';
        wait for 20 ns;

        d <= '1';
        wait for 20 ns;

        StopClock <= True;

        wait;

    end process stim_proc;

end behavioral;
```

Constants in VHDL



```vhdl
architecture behavioral of TB_DFF is

-- Component declaration of the D Flip-Flop
component DFF is
  port(
  D   : in  std_logic;
  CLK :in   std_logic;
  Q   : out std_logic);
end component;

-- Signals to connect to the D flip-flop
    signal D   : STD_LOGIC := '0';
    signal CLK : STD_LOGIC := '0';
    signal Q   : STD_LOGIC;

    signal StopClock : BOOLEAN;  -- boolean number (true or false)
    constant Period : TIME := 10 ns; -- constant

begin
```

Architecture – Declaritive part

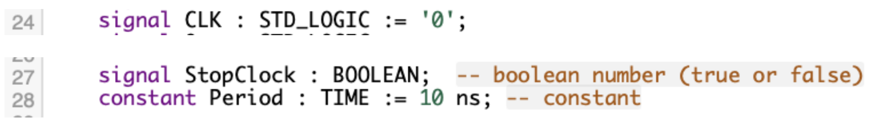
- Component declaration
- Signals declaration
  - Note the signals and constant to be used for the clock generation process

constant

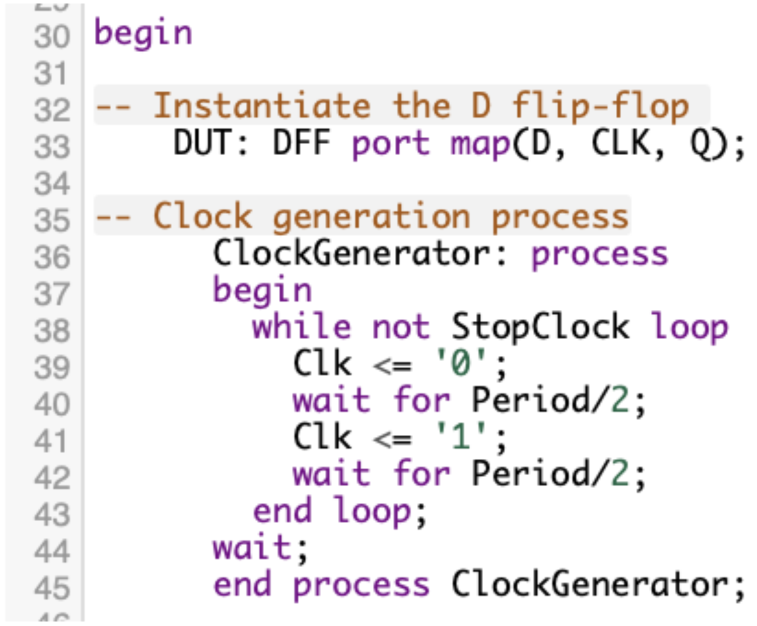
- Can be defined in **process**, **architecture** or **package** blocks
- **:=** assignment operator for constants

Clock simulation

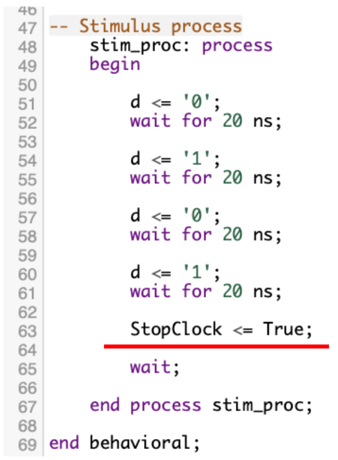
Signals and constant for clock generation process defined in the declaritive part of the architecture



```
signal CLK : STD_LOGIC := '0';

signal StopClock : BOOLEAN;  -- boolean number (true or false)
constant Period : TIME := 10 ns; -- constant
```



```
begin

-- Instantiate the D flip-flop
    DUT: DFF port map(D, CLK, Q);

-- Clock generation process
      ClockGenerator: process
      begin
        while not StopClock loop
          Clk <= '0';
          wait for Period/2;
          Clk <= '1';
          wait for Period/2;
        end loop;
      wait;
      end process ClockGenerator;
```



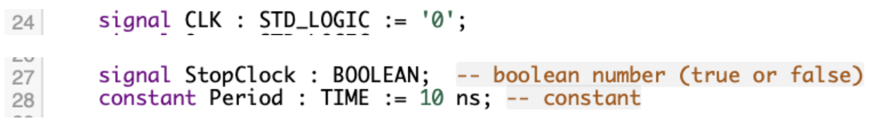
```
-- Stimulus process
    stim_proc: process
    begin

        d <= '0';
        wait for 20 ns;

        d <= '1';
        wait for 20 ns;

        d <= '0';
        wait for 20 ns;

        d <= '1';
        wait for 20 ns;

        StopClock <= True;

        wait;

    end process stim_proc;

end behavioral;
```

<u>Architecture – Implemenation part</u>

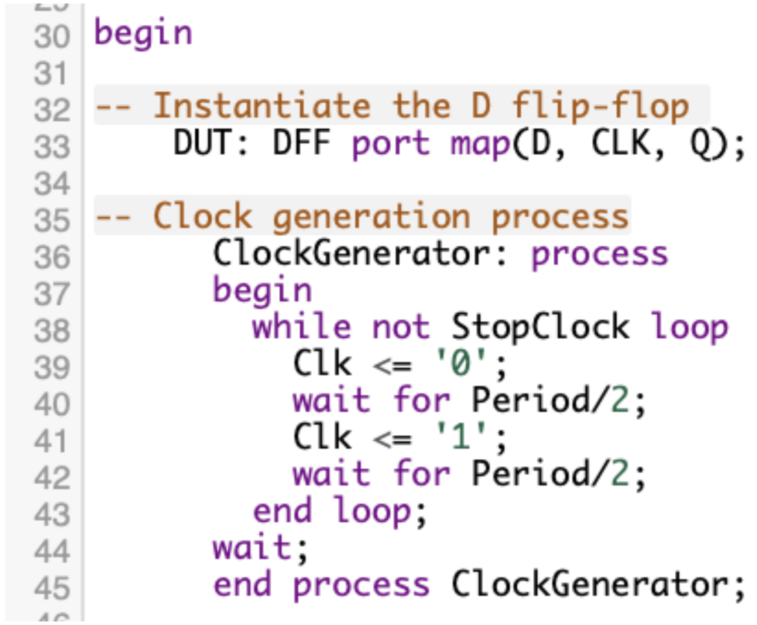
- Component instantiation
- Clock generation process
    - Simple to define in testbenches
    - Can be running indefinitely or for a finite number of clock cycles
    - In this example, we use the signal **StopClock** to stop the clock

Clock simulation

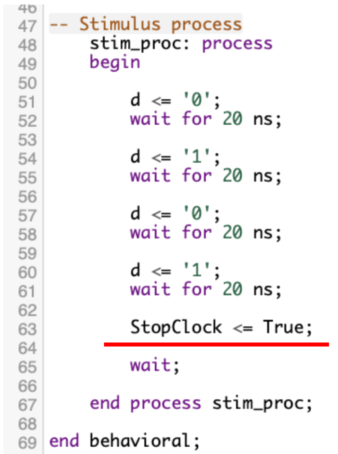
Signals and constant for clock generation process defined in the declaritive part of the architecture



```
    signal CLK : STD_LOGIC := '0';

    signal StopClock : BOOLEAN;  -- boolean number (true or false)
    constant Period : TIME := 10 ns; -- constant
```



```
begin

-- Instantiate the D flip-flop
    DUT: DFF port map(D, CLK, Q);

-- Clock generation process
      ClockGenerator: process
      begin
        while not StopClock loop
          Clk <= '0';
          wait for Period/2;
          Clk <= '1';
          wait for Period/2;
        end loop;
      wait;
      end process ClockGenerator;
```



```
-- Stimulus process
   stim_proc: process
   begin

      d <= '0';
      wait for 20 ns;

      d <= '1';
      wait for 20 ns;

      d <= '0';
      wait for 20 ns;

      d <= '1';
      wait for 20 ns;

      StopClock <= True;

      wait;

   end process stim_proc;

end behavioral;
```

<u>Architecture – Implemenation part</u>

- Component instantiation
- Clock generation process
  - Examples of clock generation syntax

```
architecture behavior of testbench is
  signal clk : std_logic := '0';
begin
  clk <= not clk after 5 ns;
end behavior;
```

```
architecture behavior of testbench is
  signal clk : std_logic := '0';
  constant CLK_PERIOD : time := 10 ns;
begin
  clk <= not clk after CLK_PERIOD/2;
end behavior;
```

```
architecture behavior of testbench is
signal clk : std_logic := '0';
constant NCYCLES : integer := 100;
begin
  clk_proc : process
  begin
    for I in 0 to NCYCLES-1 loop
      clk <= not clk;
      wait for 5 ns;
      clk <= not clk;
      wait for 5 ns;
    end loop;
    wait;
  end process;
end behavior;
```

while and for statements

while statement

- Repeats a block of code as long as the condition is true.
- The condition is evaluated before each iteration.
- Used for loops where the number of iterations is not known beforehand.

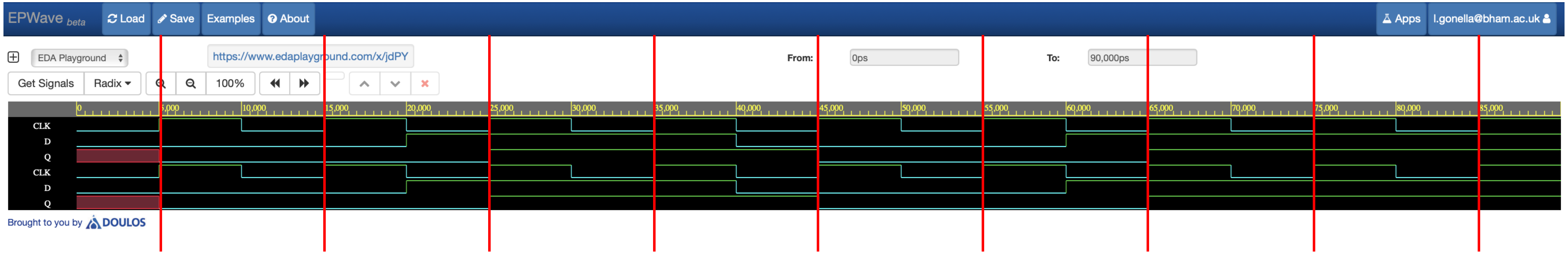
```
while <condition> loop
    <sequential statements>
end loop;
```

for statement

- Repeats a block of code a fixed number of times.
- The loop variable automatically iterates over the specified range.
- Used when the number of iterations is known beforehand.

```
for <loop_variable> in <range> loop
    <sequential statements>
end loop;
```

Example of sequential logic with process: D-FF simulation waveforms



if vs when/else

- VHDL is a concurrent language, it provides two different solutions to implement a conditional statement:
  - Sequential conditional statement: **if**
  - Concurrent conditional statement: **when/else**

- The sequential conditional statement can be used in process
- The concurrent conditional statement can be used in the architecture concurrent section, i.e. between the **begin/end** section of the VHDL architecture definition

- Total equivalence between **if** sequential statement and **when/else** statement; either can be used

- When using a conditional statement, one must pay attention to the final hardware implementation
  - A conditional statement can be translated into a MUX or a comparator or a huge amount of combinatorial logic

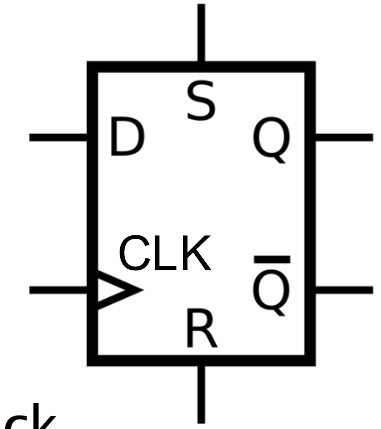
Exercise 3: MUX 2to1 code

- Rewrite the MUX example using **process** and the **if** statement

End of part 2

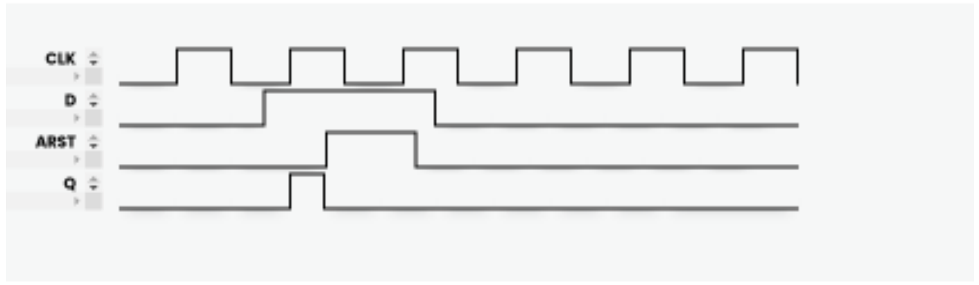
- process statement
- Edge detection
- Conditional statement: if
- Constants in VHDL
- Clock simulation
- while and for statements
- Example: OR gate with “process”
- Example: D-FF
- Exercise 3: 2to1 MUX with “process” and “if” statement

Reset signal

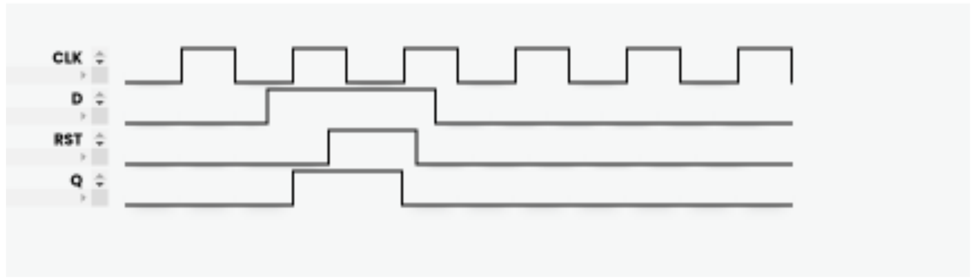


- Typically the D-FF have an input for the reset signal
- Upon application of the reset $Q = 0, \bar{Q} = 1$
- The reset signal can be synchronous or asynchronous to the process clock

```
process(CLK, RESET)
begin
    if RESET = '1' then
        Q      <= '0';           -- Asynchronous reset: Reset Q to '0'
    elsif rising_edge(CLK) then
        Q      <= D;             -- On rising clock edge, transfer D to Q
    end if;
end process;
```



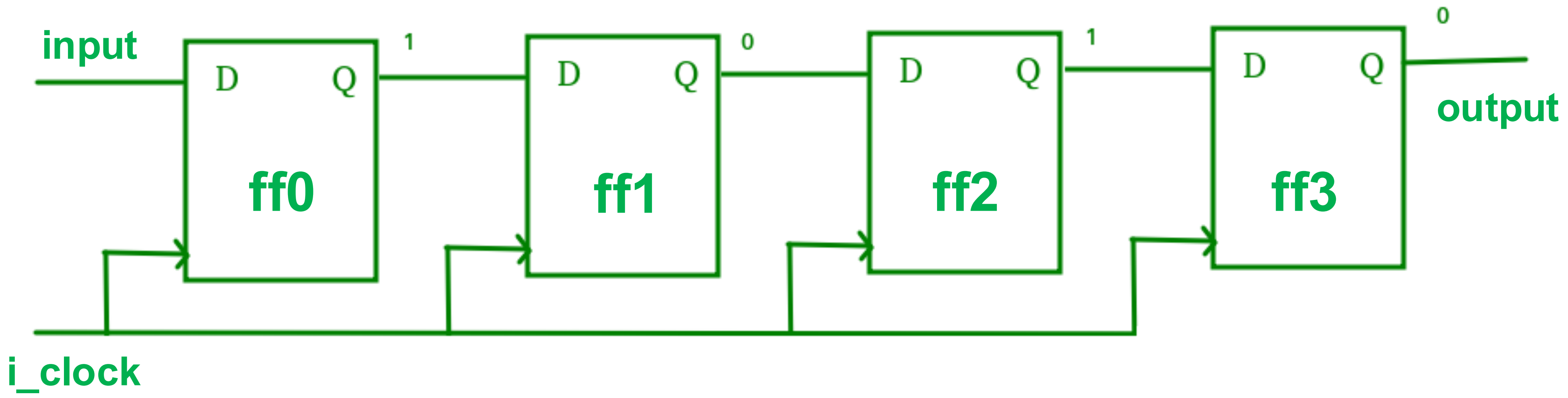
```
process(CLK, RESET)
begin
    if rising_edge(CLK) then
       if RESET = '1' then
        Q      <= '0';           -- Synchronous reset: Reset Q to '0'
       else
        Q      <= D;             -- On rising clock edge, transfer D to Q
       end if;
    end if;
end process;
```



Exercise 4: Design and simulate a D-FF with synchronous reset

Exercise 5: Design and simulate a shift register

- Design and simulate a shift register made of 4 D-FF with serial input, serial output



Assigning vectors

```
signal vec : std_logic_vector(7 downto 0) := (others => '0'); -- vec is 00000000
vec <= "10100000";                                             -- vec is now 10100000
vec(0) <= '1';                                                 -- vec is now 10100001
vec(2 downto 1) <= "01";                                       -- vec is now 10100011
vec(7 downto 5) <= vec(2 downto 0);                            -- vec is now 01100011
```

End of part 3

- Reset signal (synch, asynch)
- Assigning vectors

- Exercise 4: D-FF with synchronous reset
- Exercise 5: Shift register

Resources

- NANDLAND
- fpga4fun
- Free range VHLD book