Lab02 – Conditional assignments, Registers & Counters

1 Conditional assignments

1.1 Simple multiplexer circuit

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

entity Multiplexer is port(
    D : in std_logic_vector(3 downto 0);
    SEL : in std_logic_vector(1 downto 0);
    Q : out std_logic
);end Multiplexer;

architecture arch of Multiplexer is begin
    with SEL select Q <=
        D(0) when "00",
        D(1) when "01",
        D(2) when "10",
        D(3) when "11",
        '0' when others;
end arch;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity Multiplexer_tb is
end entity;

architecture ben of Multiplexer_tb is
component Multiplexer is port(
    D : in std_logic_vector(3 downto 0);
    SEL : in std_logic_vector(1 downto 0);
    Q : out std_logic);
end component;


signal D : std_logic_vector(3 downto 0);
signal SEL : std_logic_vector(1 downto 0);
signal Q : std_logic;

begin
    mult : Multiplexer port map(D => D, SEL => SEL, Q => Q);
    stim : process
    variable Dinputs : std_logic_vector(3 downto 0);
    variable SELinputs : std_logic_vector(1 downto 0);
    begin
      for i in 0 to 2**Dinputs'length - 1 loop
            Dinputs := std_logic_vector(to_unsigned(i, Dinputs'length));
            D <= Dinputs;
            for j in 0 to 2**SELinputs'length - 1 loop
                SELinputs := std_logic_vector(to_unsigned(j, SELinputs'length));
                SEL <= SELinputs;
                wait for 10 ns;
          end loop;
      end loop;
    end process;
end ben;
```

1.2 Single D-FF

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

entity D_FF is port(
    D : in std_logic;
    Clk : in std_logic;
    Q : out std_logic
);end D_FF;

architecture arch of D_FF is begin
    proc : process(Clk) begin
        if (rising_edge(Clk)) then
            Q <= D;
        end if;
    end process;
end arch;
```

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


entity D_FF_tb is
end entity;

architecture ben of D_FF_tb is
component D_FF is port(
    D : in std_logic;
    Clk : in std_logic;
    Q : out std_logic);
end component;

signal D : std_logic := '0';
signal Clk : std_logic := '0';
signal Q : std_logic;


begin
    dff : D_FF port map(D => D, Clk => Clk, Q => Q);
    clock_process : process
    begin
        Clk <= not Clk;
        wait for 7 ns;
    end process;
    D_process : process
    begin
        D <= not D;
        wait for 11 ns;
    end process;
end ben;
```

2 Registers

2.1 Serial-in / Serial-out (shift) registers

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

entity SISO_shiftRegister is
    generic (N: integer );
    port(
        Clk: in std_logic;
        RST: in std_logic;
        EN: in std_logic;
        D: in std_logic;
        Q: out std_logic
    );
end entity SISO_shiftRegister;

architecture arch of SISO_shiftRegister is

    signal registers: std_logic_vector(N-1 downto 0);

    begin

        Q <= registers(N-1);

         process(Clk, RST)
            begin
                if (RST = '1') then

                   -- Put the output to 0
                   registers(N-1) <= '0';

                    -- To put all the content of registers to zero
                    -- registers <= (others => '0');

                elsif (rising_edge(Clk)) then
                    if (EN = '1') then
                        registers(0) <= D;
                        for i in 1 to N-1 loop
                            registers(i) <= registers(i-1);
                        end loop;

                    end if;
                end if;
        end process;

end architecture arch;
```

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

entity SISO_shiftRegister_tb is
end entity;

architecture ben of SISO_shiftRegister_tb is

    component SISO_shiftRegister is
        generic (N: integer
        );
        port(
        Clk: in std_logic;
        RST: in std_logic;
        EN: in std_logic;
        D: in std_logic;
        Q: out std_logic
        );
    end component;

    signal Clk : std_logic := '0';
    signal RST:  std_logic := '0';
    signal EN: std_logic := '1';
    signal D : std_logic;
    signal Q : std_logic;

    begin
        siso_reg : SISO_shiftRegister generic map (N => 8) port map(Clk => Clk, RST => RST,
    EN => EN, D => D, Q => Q);


        clock_process : process
        begin
            Clk <= not Clk;
            wait for 5 ns;
        end process;


        D_process : process
        begin
            D <= '1';
            wait for 14  ns;
            D <= '0';
            wait for 7 ns;
        end process;

        EN_RST_process : process
        begin
            -- Enable is initialised to 1
            wait for 210 ns;
            EN <= '0';
            -- RST is initialised to 0
            wait for 20 ns;
            RST <= '1';
        end process;
end ben;
```

Note that in the SISO shift register module given above, we have used **behavioural style** of modelling. It is also possible to write the same module using **structural style** of modelling, in which an entity is described as a set of interconnected components. In that case, a SISO architecture body could be written as:

```
  flip1 : D_FF port map(Clk => Clk, D => D, Q => s(0));
    Dloop: for i in 1 to N-1 generate
        flip : D_FF port map(Clk => Clk, D => s(i-1), Q => s(i));
    end generate;
    Q <= s(N-1) and not RST;
```

2.2 Parallel-in / Parallel-out registers

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

entity PIPO_Register is
    generic (N: integer);
    port(
        Clk: in std_logic;
        RST: in std_logic;
        LOAD: in std_logic;
        D: in std_logic_vector(N-1 downto 0);
        Q: out std_logic_vector(N-1 downto 0)
    );
end entity PIPO_Register;

architecture arch of PIPO_Register is

    begin

        process(Clk, RST)
            begin
                if (RST = '1') then
                    Q <= (others => '0');

                elsif (rising_edge(Clk)) then
                    if (LOAD = '1') then
                        Q <= D;
                    end if;
                end if;
        end process ;

end architecture arch;
```

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

entity PIPO_Register_tb is
end entity;

architecture ben of PIPO_Register_tb is
    component PIPO_Register is
        generic (N: integer
        );
        port(
        Clk: in std_logic;
        RST: in std_logic;
        LOAD: in std_logic;
        D: in std_logic_vector(N-1 downto 0);
        Q: out std_logic_vector(N-1 downto 0)
        );
    end component;

    signal Clk : std_logic := '1';
    signal RST:  std_logic := '1';
    signal LOAD: std_logic := '1';
    signal D : std_logic_vector(7 downto 0);
    signal Q : std_logic_vector(7 downto 0);

begin
    pipo_reg : PIPO_Register generic map (N => 8)
                                  port map(Clk => Clk, RST => RST, LOAD => LOAD, D => D, Q =>
     Q);


    clock_process : process
    begin
        Clk <= not Clk;
        wait for 5 ns;
    end process;


    D_process : process
    begin
        D <= "00100100";
        wait for 12 ns;
        D <= "10000000";
        wait for 12 ns;
        D <= "11111111";
        wait for 12 ns;
        D <= "01010110";
        wait for 12 ns;
    end process;

    EN_process : process
    begin
        -- Enable is initialised to 1
        wait for 50 ns;
        LOAD <= '0';
    end process;

    RST_process: process
        begin
        wait for 5 ns;
        RST <= '0';
        wait for 70 ns;
        RST <= '1';
    end process;

end ben;
```

2.3 Parallel or Serial-in / Parallel-out registers

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

entity PSIPO_Register is
    generic (N: integer := 4);
    port(
        Clk: in std_logic;
        RST: in std_logic;
        LOAD: in std_logic;
        SERIAL: in std_logic;
        D: in std_logic_vector(N-1 downto 0);
        Q: out std_logic_vector(N-1 downto 0)
    );
end entity PSIPO_Register;

architecture arch of PSIPO_Register is

    signal registers: std_logic_vector(N-1 downto 0);

    begin
        Q <= registers; --parallel output

        proc: process(Clk, RST)
            begin
                if (RST = '1') then
                    registers <= (others => '0');

                elsif (rising_edge(Clk)) then

                    if (LOAD = '1') then
                        if (SERIAL = '0') then
                            registers <= D; --parallel input

                        elsif (SERIAL = '1') then

                            for i in 1 to N-1 loop
                              registers(i) <= registers(i-1);
                            end loop;

                             registers(0) <= D(0); --serial input


                        end if;
                    end if;
                end if;
        end process proc;

end architecture arch;
```

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

entity PSIPO_Register_tb is
end entity;

architecture ben of PSIPO_Register_tb is
    component PSIPO_Register is
        generic (N: integer
        );
        port(
        Clk: in std_logic;
        RST: in std_logic;
        LOAD: in std_logic;
        SERIAL: in std_logic;
        D: in std_logic_vector(N-1 downto 0);
        Q: out std_logic_vector(N-1 downto 0)
        );
    end component;

    signal Clk : std_logic := '1';
    signal RST:  std_logic := '1';
    signal LOAD: std_logic := '1';
    signal SERIAL: std_logic;
    signal D : std_logic_vector(7 downto 0);
    signal Q : std_logic_vector(7 downto 0);

begin
    psipo_reg : PSIPO_Register generic map (N => 8)
                                  port map(Clk => Clk, RST => RST, SERIAL => SERIAL, LOAD =>
    LOAD, D => D, Q => Q);


    clock_process : process
    begin
        Clk <= not Clk;
        wait for 5 ns;
    end process;

    D_process : process
    begin
        D <= "00100100";
        wait for 12 ns;
        D <= "10000000";
        wait for 12 ns;
        D <= "11111111";
        wait for 12 ns;
        D <= "01010110";
        wait for 12 ns;
    end process;

    EN_process : process
    begin
        -- Enable is initialised to 1
        wait for 100 ns;
        LOAD <= '0';
    end process;

    RST_process: process
    begin
        wait for 5 ns;
        RST <= '0';
        wait for 140 ns;
        RST <= '1';
    end process;

    SERIAL_process: process
    begin
        SERIAL <= '0';
        wait for 50 ns;
        SERIAL <= '1';
        wait for 50 ns;
    end process;
end ben;
```

3 Counters

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

entity Counter is generic(
    N : integer
); port(
    Clk, UP : in std_logic;
    Q : out signed(N-1 downto 0)
);end entity;

architecture arch of Counter is

begin
    proc : process(Clk)
    variable i : signed(N-1 downto 0) := (others => '0');
    begin
        if (rising_edge(Clk)) then
            if (UP = '1') then
                i <= i + 1;
            else
                i <= i - 1;
            end if;
            Q <= i;
        end if;
    end process;
end arch;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.math_real.all;

entity Counter_tb is
end entity;

architecture ben of Counter_tb is
component Counter is generic(
    N : integer
); port(
    Clk, UP : in std_logic;
    Q : out signed(N-1 downto 0)
);end component;


signal Clk, UP: std_logic := '0';
signal Q : signed(7 downto 0);

begin
    cntr : Counter generic map (N => 8) port map(UP => UP, Clk => Clk, Q => Q);
    clock_process : process
    begin
        Clk <= not Clk;
        wait for 1 ns;
    end process;
    UP_process : process
    begin
        UP <= not UP;
        wait for 50 ns;
    end process;
end ben;
```