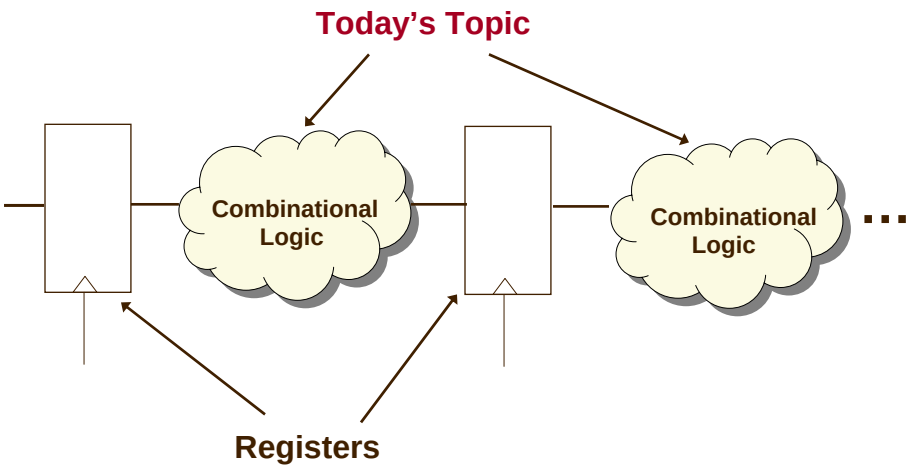
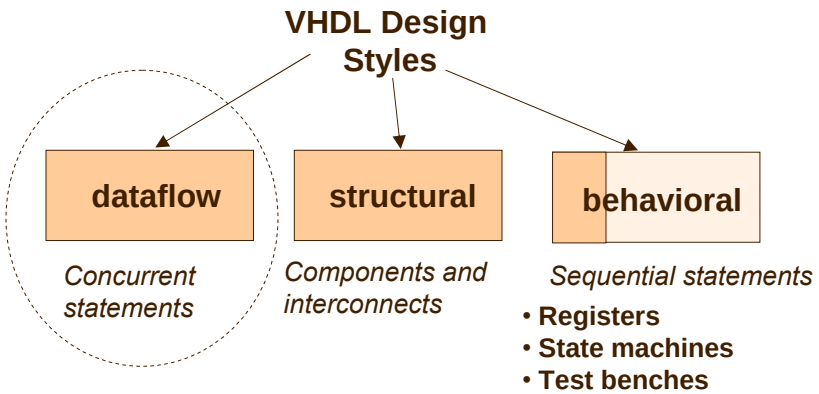
Μοντελοποίηση Dataflow Συνδυαστικής Λογικής σε VHDL

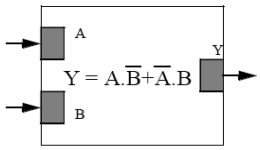
Register Transfer Level (RTL) Design Description

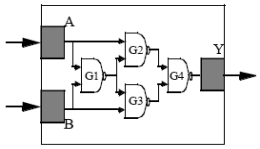


VHDL Design Styles



Behavior vs Structure





An entity can be described by its *behavior* or by its *structure*, or in a mixed fashion.

example: a 2-input XOR gate

- Vhdl model ?

Vhdl code

```
entity XOR is
        port ( A,B : in bit; Y : out bit);
end XOR;

architecture BEHAVIOR of XOR is
begin
        Y <= (A and not B) or (not A and B);
end BEHAVIOR;
```

```
architecture STRUCTURE of XOR is
        component NAND
                port ( A, B : in bit; Y : out bit);
        end component;
        signal C, D, E : bit;
begin
        G1 : NAND port map (A, B, C);
        G2 : NAND port map
                (A => A, B => C, Y => D);
        G3 : NAND port map
                (C, B => B, Y => E);
        G4 : NAND port map (D, E, Y);
end STRUCTURE;
```

```
architecture MIXED of XOR is
        component NAND
                port ( A, B : in bit; Y : out bit);
        end component;
        signal C, D, E : bit;
begin
        D <= A nand C;
        E <= C nand B;
        G1 : NAND port map (A, B, C);
        G4 : NAND port map (D, E, Y);
end MIXED;
```


Signals vs Variables

```
architecture DUMMY_1 of JUNK is
        signal Y : bit := '0';
begin
        process
                variable X : bit := '0';
        begin
                wait for 10 ns;
                X := '1';
                Y <= X;
                wait for 10 ns;
                -- What is Y at this point ? Answer: '1'
                ...
        end process;
end DUMMY_1;
```

- Variables: δήλωση μέσα σε ένα Process

Signals vs Variables

```
architecture DUMMY_2 of JUNK is
        signal X, Y : bit := '0';
begin
        process
        begin
                wait for 10 ns;
                X <= '1';
                Y <= X;
                wait for 10 ns;
                -- What is Y at this point ? Answer: '0'
                ...
        end process;
end DUMMY_2;
```

Signal assignments with 0 delay take effect only after a *delta* delay. *i.e.*, in the next simulation cycle.

Testbenches

Generating selected values of one input

```
SIGNAL test_vector : STD_LOGIC_VECTOR(2 downto 0);

BEGIN
  .......
     testing: PROCESS
     BEGIN
          test_vector <= "000";
          WAIT FOR 10 ns;
          test_vector <= "001";
          WAIT FOR 10 ns;
          test_vector <= "010";
          WAIT FOR 10 ns;
          test_vector <= "011";
          WAIT FOR 10 ns;
          test_vector <= "100";
          WAIT FOR 10 ns;
     END PROCESS;
 ........
END behavioral;
```


Generating all values of one input

```
SIGNAL test_vector : STD_LOGIC_VECTOR(3 downto 0):="0000";

BEGIN
   .......

   testing: PROCESS
   BEGIN
          WAIT FOR 10 ns;
          test_vector <= test_vector + 1;
   end process TESTING;

   ........
END behavioral;
```

Generating all possible values of two inputs

```
SIGNAL test_ab : STD_LOGIC_VECTOR(1 downto 0);
SIGNAL test_sel : STD_LOGIC_VECTOR(1 downto 0);

BEGIN
        .......

                double_loop: PROCESS
                BEGIN
                        test_ab  <="00";
                        test_sel  <="00";
                        for I in 0 to 3 loop
                            for J in 0 to 3 loop
                                    wait for 10 ns;
                                       test_ab <= test_ab + 1;
                            end loop;
                            test_sel <= test_sel + 1;
                        end loop;
                END PROCESS;

        ........
        END behavioral;
```


Generating periodical signals, such as clocks

```
        CONSTANT clk1_period : TIME := 20 ns;
        CONSTANT clk2_period : TIME := 200 ns;
        SIGNAL clk1 : STD_LOGIC;
        SIGNAL clk2 : STD_LOGIC := '0';

      BEGIN
        .......
        clk1_generator: PROCESS
                clk1 <= '0';
                WAIT FOR clk1_period/2;
                clk1 <= '1';
                WAIT FOR clk1_period/2;
        END PROCESS;

        clk2 <= not clk2  after  clk2_period/2;
       .......
      END behavioral;
```

Generating one-time signals, such as resets

```
CONSTANT reset1_width : TIME := 100 ns;
CONSTANT reset2_width : TIME := 150 ns;
 SIGNAL reset1 : STD_LOGIC;
SIGNAL reset2 : STD_LOGIC := '1';

BEGIN
  .......
  reset1_generator: PROCESS
        reset1 <= '1';
        WAIT FOR reset_width;
        reset1 <= '0';
        WAIT;
  END PROCESS;

  reset2_generator: PROCESS
        WAIT FOR reset_width;
        reset2 <= '0';
        WAIT;
  END PROCESS;
  .......
END behavioral;
```


Typical error

```
SIGNAL test_vector : STD_LOGIC_VECTOR(2 downto 0);
SIGNAL reset : STD_LOGIC;

BEGIN
  .......
  generator1: PROCESS
        reset <= '1';
        WAIT FOR 100 ns
        reset <= '0';
        test_vector <="000";
        WAIT;
  END PROCESS;

  generator2: PROCESS
        WAIT FOR 200 ns
        test_vector <="001";
        WAIT FOR 600 ns
        test_vector <="011";
  END PROCESS;
  .......
END behavioral;
```

Wait for vs. *Wait*

Wait for: waveform will keep repeating itself forever

Wait : waveform will keep its state after the last wait instruction.

Report - Examples

```
report "Initialization complete";

report  "Current time = " & time'image(now);

report  "Incorrect branch"  severity error;
```

Generating reports in the message window

```
reports: process(clk_trigger) begin
        if (clk_trigger = '0' and clk_trigger'EVENT) then
                case segments is
                        when seg_0 => report time'image(now) & ": 0 is displayed" ;
                        when seg_1 => report time'image(now) & ": 1 is displayed" ;
                        when seg_2 => report time'image(now) & ": 2 is displayed" ;
                        when seg_3 => report time'image(now) & ": 3 is displayed" ;
                        when seg_4 => report time'image(now) & ": 4 is displayed" ;
                        when seg_5 => report time'image(now) & ": 5 is displayed" ;
                        when seg_6 => report time'image(now) & ": 6 is displayed" ;
                        when seg_7 => report time'image(now) & ": 7 is displayed" ;
                        when seg_8 => report time'image(now) & ": 8 is displayed" ;
                        when seg_9 => report time'image(now) & ": 9 is displayed" ;
                end case;
        end if;
end process;
```


Anatomy of a Process

[label:] process [(*sensitivity list*)]
    [*declaration part*]
begin
    statement part
end process;

Sequential Statements (1)

- If statement

```
if boolean expression then
    statements
elsif boolean expression then
    statements
        ⋮
else boolean expression then
    statements
end if;
```

- else and elsif are optional

Sequential Statements (3)

- Loop Statement

```
for i in range loop
    statements
end loop;
```

- Repeats a section of VHDL code

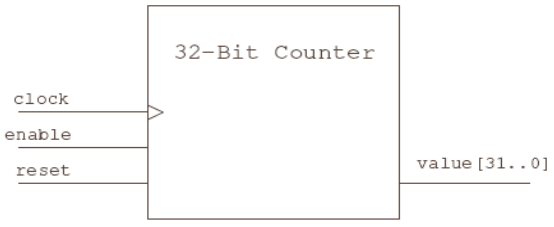
Sequential Statements (2)

- Case statement
- Choices have to cover all possible values of the condition
  - Use others to specify all remaining cases

```
case condition is
    when choice_1 =>
        statements
    when choice_2 =>
        statements
        ⋮

    when others =>
        statements
end case;
```


Παράδειγμα:Σχεδίαση ενός μετρητή

- Σχεδιάστε – υλοποιήστε σε synthesizable VHDL έναν 32-bit μετρητή με είσοδο ενεργοποίησης και ασύγχρονο reset.
- Το block διάγραμμα φαίνεται παρακάτω:



Λύση

- Δήλωση entity:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY counter IS
   PORT(
        reset            : IN     std_logic;
        clock            : IN     std_logic;
        enable           : IN     std_logic;
        value            : OUT    std_logic_vector(31 DOWNTO 0)
   );
END counter;
```

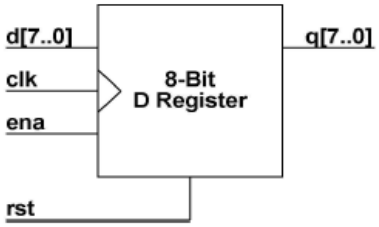

Λύση

- Αρχιτεκτονική του μετρητή:

```
ARCHITECTURE synthesis1 OF counter IS

                                          -- The unsigned type is used
   SIGNAL count  : unsigned(31 DOWNTO 0);  -- so that unsigned arithmetic
                                          -- will be synthesized

BEGIN
   PROCESS (reset, clock)
   BEGIN
        IF (reset = '1') THEN
                count <= X"00000000";
        ELSIF (clock'EVENT) AND (clock = '1') THEN
                IF (enable = '1') THEN
                        count <= count + 1;
                END IF;
        END IF;
   END PROCESS;
                                          -- Here, the count value is
   value <= std_logic_vector(count);      -- converted to std_logic_vector
                                          -- using a conversion function

END synthesis1;
```

Παράδειγμα 2: Καταχωρητής

- Σχεδιάστε έναν 8-bit καταχωρητή σε synthesizable VHDL με είσοδο ενεργοποίησης και ασύγχρονο reset.

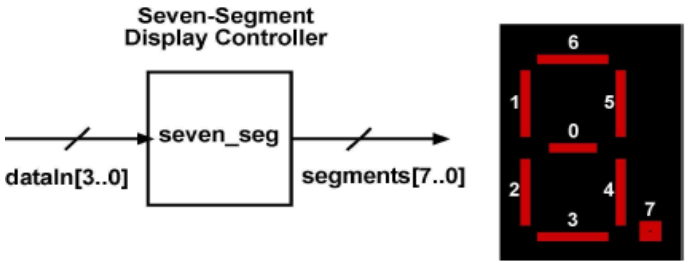


Λύση 2

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dregister IS
   PORT( rst, clk, ena    : IN     std_logic;
         d                : IN     std_logic_vector(7 DOWNTO 0);
         q                : OUT    std_logic_vector(7 DOWNTO 0) );
END dregister;

ARCHITECTURE synthesis1 OF dregister IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
         IF (rst = '1') THEN
                  q <= X"00";
         ELSIF (clk'EVENT) AND (clk = '1') THEN
                  IF (ena = '1') THEN
                           q <= d;
                  END IF;
         END IF;
   END PROCESS;
END synthesis1;
```

Παράδειγμα 3 : Συνδυαστική Λογική

- Σχεδιάστε έναν ελεγκτή 7-segment-display σε synthesizable VHDL



Λύση

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY seven_seg IS
   PORT( dataIn          : IN     std_logic_vector(3 DOWNTO 0);
         segments        : OUT    std_logic_vector(7 DOWNTO 0) );
END seven_seg;

ARCHITECTURE synthesis1 OF seven_seg IS
BEGIN
   WITH dataIn SELECT
         segments <=     "10000001" WHEN "0000",   -- 0
                         "11001111" WHEN "0001",   -- 1
                         "10010010" WHEN "0010",   -- 2
                         "10000110" WHEN "0011",   -- 3
                         "11001100" WHEN "0100",   -- 4
                         "10100100" WHEN "0101",   -- 5
                         "10100000" WHEN "0110",   -- 6
                         "10001111" WHEN "0111",   -- 7
                         "10000000" WHEN "1000",   -- 8
                         "10000100" WHEN "1001",   -- 9
                         "11111111" WHEN OTHERS;
END synthesis1;
```