

Логичко пројектовање

Предавање 5/10

Основе VHDL-а - 2. део

Увод

- У претходној лекцији "Основе VHDL-а", VHDL је представљен као начин да се опише понашање конкуретних система.
- Представљене технике моделирања биле су прикладне за комбинационе мреже јер те врсте мреже имају логичка кола где излази зависе само од вредности на њиховим улазима. То значи да модели ових кола где се континуално доводе сигнали пружа тачне моделе понашања ових кола.
- У претходној лекцији представљени су секвенцијални логички елементи за складиштење где се нису континуално ажурирали њихови излази на основу тренутних вредности улаза. Уместо тога, секвенцијални елементи за складиштење ажурирали су своје излазе само на основу догађаја, нпр. ивицом сигнала clock-а.
- Технике моделирања представљене у лекцији "Основе VHDL-а" нису у стању да прецизно опишу ову врсту понашања. У овој лекцији биће описане VHDL конструкције за моделирање додела сигнала које покрећу догађаје како би се прецизно моделирала секвенцијална логика.
- Ово поглавље такође представља:
 - Тест бенч
 - Најчешће коришћене пакете који повећавају способност и тачност VHDL-а

Процес

- VHDL користи процес за моделовање сигнала који су засновани на догађају.
- Процес је техника за моделовање понашања система, због тога се процес ставља у VHDL архитектуру након наредбе старт.
- Сигнали унутар процеса имају јединствене карактеристике које им омогућавају да прецизно моделују секвенцијалну логику.
- Прво, додељивање сигнала се не одвија док се процес завршава или је суспендован.
- Друго, додељивање сигнала ће бити извршено само једном, сваки пут када се процес покрене.
- Коначно, доделе сигнала ће бити извршене редоследом којим се појављују у процесу. Ово понашање се назива секвенцијално додељивање сигнала. Секвенцијална додела сигнала омогућава процесу да моделује понашање на нивоу преноса регистра где сигнал може бити као операнд доделе и као одредиште другог задатка у оквиру истог процеса.
- VHDL пружа две технике за покретање процеса:
 - Листа осетљивости
 - Wait наредба.

Листа осететљивости

- Листа осетљивости је механизам за контролу када се процес покрене (или стартује).
- Листа осетљивости садржи листу сигнала на које је процес осетљив.
- Ако постоји прелаз код било ког сигнала на листи, процес ће бити активиран и биће извршена додела сигнала у процесу.
- Синтакса за процес који користи листу осетљивости:

```
process_name : process (<signal_name1>,  
  <signal_name2>, ...)  
  
  -- variable declarations  
  
  begin  
    sequential_signal_assignment_1  
    sequential_signal_assignment_2  
    :  
  end process;
```

- Пример:

```
FlipFlop : process (Clock)  
  begin  
    Q <= D;  
  end process;
```

Листа осететљивости

- Листа осетљивости је механизам за контролу када се процес покрене (или стартује).
- Листа осетљивости садржи листу сигнала на које је процес осетљив.
- Ако постоји прелаз код било ког сигнала на листи, процес ће бити активиран и биће извршена додела сигнала у процесу.
- Синтакса за процес који користи листу осетљивости:

```
process_name : process (<signal_name1>,
<signal_name2>, ...)

    -- variable declarations

begin
    sequential_signal_assignment_1
    sequential_signal_assignment_2
    :
end process;
```

- Пример:

```
FlipFlop : process (Clock)
begin
    Q <= D;
end process;
```

У овом примеру, прелаз clock сигнала (LOW на HIGH или HIGH на LOW) ће покренути процес. Додељивање сигнала D на Q ће бити извршено када се процес заврши. Ово понашање је блиско моделу понашања правог D-флип флопа.

Wait наредба

- Wait наредба је механизам за суспендовање (или заустављање) процеса и омогућавање доделе сигнала без потребе да се процес заврши. Када се користи wait наредба, листа осетљивости се не користи.
- Без листе осетљивости, процес ће се одмах покренути. У оквиру процеса се користи Wait наредба да заустави и започне процес.
- Постоје три начина за коришћење Wait наредбе.
- Први је неограничено чекање. У следећем примеру, процес не садржи листу осетљивости, тако да ће се одмах покренути. Кључна реч **wait** се користи за обустављање процеса. Када се дође до ове наредбе, доделе сигнала Y1 и Y2 биће извршене и процес ће бити суспендован на неодређено време.

```
Proc_Ex1 : process
begin
    Y1 <= '0';
    Y2 <= '1';

    wait;
end process;
```



Wait for наредба

- Друга техника употребе wait наредбе за суспендовање процеса је да се она користи у вези са кључном речју **for** и изразом за време.
- У следећем примеру, процес ће се одмах покренути пошто не садржи листу осетљивости.
- Када процес дође до wait наредбе, суспендоваће се и изврши прву доделу сигнала CLK (CLK <= '0'). Након 5 ns, процес ће поново почети. Када стигне до друге wait наредбе, суспендоваће се и изврши другу доделу сигнала CLK (CLK <= "1"). Након још 5 ns, процес ће почети поново и одмах завршити због краја процеса.
- Након што се процес заврши, одмах ће се поново покренути због недостатка листе осетљивости и поновити претходно описано понашање.
- Ово понашање ће се наставити на неодређено време. Овај пример ствара квадратни талас назван CLK (clock) са периодом 10ns.

```
Proc_Ex2 : process
begin
    CLK <= '0'; wait for 5 ns;
    CLK <= '1'; wait for 5 ns;
end process;
```



Wait until наредба

- Трећа техника коришћења wait наредбе за суспендовање процеса је да се користи заједно са кључном речи **until** и логичким условом.
- У следећем примеру, процес ће се поново покренути одмах јер не постоји листа осетљивости.
- Процес ће тада одмах суспендовати и наставити када логички услов буде испуњен (тј. Counter > 15). Када се услов испуни, процес ће почети поново. Када дође до друге wait наредбе, извршиће прво додељивање сигнала (RollOver <= „1“).
- Након 1 ns, процес ће се наставити. Када се процес заврши, он ће извршити другу доделу сигнала (RollOver <= „0“).

```

Proc_Ex3 : process
begin
    wait until (Counter > 15);           --
first wait statement
    RollOver <= '1'; wait for 1 ns;      --
second wait statement
    RollOver <= '0';
end process;

```



Секвенцијална додела сигнала

- Један од збуњујућих концепата процеса је како се секвенцијална додела сигнала понаша.
- Правила додељивања сигнала унутар процеса су као у наставку:
 - Сигнали се не могу декларисати у оквиру процеса.
 - Додељивање сигнала се не одвија док се процес не заврши или обустави.
 - Додељивање сигнала се извршава у редоследу којим се појављују у процесу (када се процес заврши или суспендује).

Example: Behavior of Sequential Signal Assignments within a Process

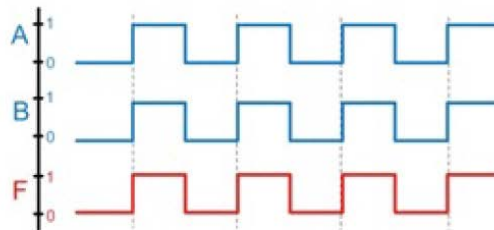
For the following system:



```
entity Ex is
  port (A : in bit;
        F : out bit);
end entity;
```

The output F will match the input A when modeled with the following process:

```
architecture Ex_arch of Ex is
  signal B : bit;
begin
  Proc_Ex : process (A)
  begin
    B <= A;
    F <= not B;
  end process;
end architecture;
```



Секвенцијална додела сигнала 2

- Претходни пример показује понашање секвенцијалног додељивања сигнала када се они извршавају унутар процеса.
- Интуитивно бисмо могли да претпоставимо да ће F бити комплемент од A .
- Међутим, због начина на који се секвенцијално додељивање сигнала обавља унутар процеса, то није случај. Да бисмо разумели ово понашање, погледајмо ситуација у којој A прелази са 0 на 1 где је $B = 0$ и $F = 0$ на почетку.
- Овај прелаз покреће процес пошто је A наведен у листи осетљивости. Када се процес покреће, $A = 1$ пошто се овде налази улаз после покретања транзиције. Прво додељивање сигнала ($B \leq A$) ће изазвати $B = 1$, али ово додељивање се јавља тек након што се процес заврши. То значи да када се други сигнал евалуира ($F \leq \text{not } B$), користи се почетна вредност B од тренутка када је процес покренут ($B = 0$) пошто се B не ажурира на 1 док се процес не заврши.
- Друго додељивање даје $F = 1$. Када се процес заврши, $A = 1$, $B = 1$ и $F = 1$. Понашање овог процеса ће увек резултирати $A = B = F$.
- Ово је контра интуитивно јер нас израз $F \leq \text{not } B$ наводи на то да верујемо да ће F увек бити комплемент од A и B ; међутим, то није случај због начин на који се додељивању сигнали, где се ажурирање у процесу врши само након суспендовања или завршетка процеса.

Example: Behavior of Concurrent Signal Assignments outside a Process

For the following system:



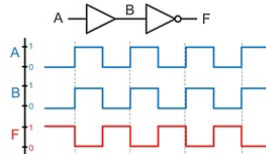
```

entity Ex is
  port (A : in bit;
        F : out bit);
end entity;
  
```

The output F will be the complement of input A when the assignments are executed concurrently.

```

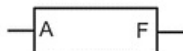
architecture Ex_arch of Ex is
  signal B : bit;
begin
  B <= A;
  F <= not B;
end architecture;
  
```



Секвенцијална додела сигнала 3

Example: Behavior of Concurrent Signal Assignments outside a Process

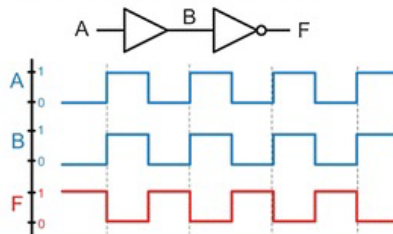
For the following system:



```
entity Ex is  
  port (A : in bit;  
        F : out bit);  
end entity;
```

The output F will be the complement of input A when the assignments are executed concurrently.

```
architecture Ex_arch of Ex is  
  signal B : bit;  
begin  
  B <= A;  
  F <= not B;  
end architecture;
```



Променљиве

- Постоје ситуације унутар процеса у којима је пожељно да се доделе изврше тренутно уместо да се чека да процес буде обустављен.
- За ове ситуације, VHDL обезбеђује концепт променљиве.
- Променљива има следеће карактеристике:
 - Променљиве постоје само унутар процеса.
 - Променљиве су дефинисане у процесу пре почетка наредбе.
 - Када се процес заврши, променљиве се уклањају из система. Ово значи да додељивање променљивих не могу да врше системи изван процеса.
 - Додељивање променљивих врши се помоћу оператора “:=”.
 - Додељивање променљивих врши се тренутно.
 - Променљива се декларише пре наредбе старт у процесу. Синтакса за декларисање променљиве је следеће:

- ```
variable variable_name : <type> :=
<initial_value>;
```

## Променљиве - пример

### Example: Behavior of Variable Assignments within a Process

For the following system:



```

entity Ex is
 port (A : in bit;
 F : out bit);
end entity;

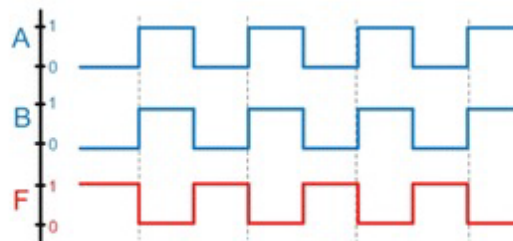
```

```

architecture Ex_arch of Ex is
 signal B : bit;
begin
 Proc_Ex : process (A)
 variable temp : bit := '0';
 begin
 temp := A;
 B <= temp;
 F <= not temp;
 end process;
end architecture;

```

The output F will match the input A when modeled with the following process:



### Наредбе услова

- If/then
- Case
- Infinite Loops
- While Loops
- For Loops

# If/then

## Example: Using If/Then Statements to Model Combinational Logic

Implement the following truth table using an if/then statement within a process.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

```
entity SystemX is
 port (A, B, C : in bit;
 F : out bit);
end entity;
```

Recall that an if/then statement is only legal within a process. In order to create a process that models combinational logic, we need to list each of the inputs to the circuit in the sensitivity list. This will cause the process to trigger and make an assignment to the output whenever there is a change on any of the inputs.

```
architecture SystemX_arch of SystemX is
begin
 SystemX_Proc : process (A, B, C)
 begin
 if (A='0' and B='0' and C='0') then F <= '1';
 elsif (A='0' and B='0' and C='1') then F <= '0';
 elsif (A='0' and B='1' and C='0') then F <= '1';
 elsif (A='0' and B='1' and C='1') then F <= '0';
 elsif (A='1' and B='0' and C='0') then F <= '0';
 elsif (A='1' and B='0' and C='1') then F <= '0';
 elsif (A='1' and B='1' and C='0') then F <= '1';
 elsif (A='1' and B='1' and C='1') then F <= '0';
 end if;
end process;
end architecture;
```

A more compact version of this behavior can be created by taking advantage of the else clause. In this model, only Boolean conditions are listed for outputs corresponding to 1's.

```
architecture SystemX_arch of SystemX is
begin
 SystemX_Proc : process (A, B, C)
 begin
 if (A='0' and B='0' and C='0') then F <= '1';
 elsif (A='0' and B='1' and C='0') then F <= '1';
 elsif (A='1' and B='1' and C='0') then F <= '1';
 else F <= '0';
 end if;
end process;
end architecture;
```

# Case

## Example: Using Case Statements to Model Combinational Logic

Implement the following truth table using a case statement within a process.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

```
entity SystemX is
 port (A, B, C : in bit;
 F : out bit);
end entity;
```

A case statement is only legal within a process. In the following example, the three input scalars (A,B,C) are concatenated into a new variable for use as the input signal to the case statement.

```
architecture SystemX_arch of SystemX is
begin
 SystemX_Proc : process (A, B, C)
 variable ABC : bit_vector (2 downto 0) := "000";
 begin
 ABC := A & B & C;

 case (ABC) is
 when "000" => F <= '1';
 when "001" => F <= '0';
 when "010" => F <= '1';
 when "011" => F <= '0';
 when "100" => F <= '0';
 when "101" => F <= '0';
 when "110" => F <= '1';
 when "111" => F <= '0';
 end case;

 end process;
end architecture;
```

More compact forms of the case statement can be created using the *when others* clause and pipe delimited inputs.

```
case (ABC) is
 when "000" => F <= '1';
 when "010" => F <= '1';
 when "110" => F <= '1';
 when others => F <= '0';
end case;
```

```
case (ABC) is
 when "000" | "010" | "110" => F <= '1';
 when others => F <= '0';
end case;
```



## Infinite Loops

```
loop
exit when boolean_condition; -- optional exit
statement
next when boolean_condition; -- optional next
statement
sequential_statement(s);
end loop;
```

```
Clock_Proc1 : process
begin
 loop
 CLK <= not CLK;
 wait for 50 ns;
 end loop;
end process;
```

```
Clock_Proc2 : process
begin
 loop
 exit when EN='0';
 CLK <= not CLK;
 wait for 50 ns;
 end loop;

 CLK <= '0';
 wait until EN='1';

end process;
```

## While Loops

```
while boolean_condition loop
sequential_statement(s);
end loop;
```

```
Clock_Proc3 : process
begin
while (EN='1') loop
CLK <= not CLK;
wait for 50 ns;
end loop;
```

```
CLK <= '0';
wait until EN='1';
```

```
end process;
```

## For Loops

```
for loop_variable in min to max loop
sequential_statement(s);
end loop;
```

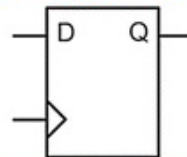
```
for loop_variable in max downto min loop
sequential_statement(s);
end loop;
```

```
Counter_Proc : process
begin
 for i in 0 to 15 loop
 Count_Out <= i;
 wait for 50 ns;
 end loop;
end process;
```

# Атрибути сигнала

| Attribute     | Information returned                                  | Type returned  |
|---------------|-------------------------------------------------------|----------------|
| A'event       | True when signal A changes, false otherwise           | Boolean        |
| A'active      | True when an assignment is made to A, false otherwise | Boolean        |
| A'last_event  | Time when signal A last changed                       | Time           |
| A'last_active | Time when signal A was last assigned to               | Time           |
| A'last_value  | The previous value of A                               | Same type as A |
| B'length      | Size of the vector (e.g., 8)                          | Integer        |
| B'left        | Left bound of the vector (e.g., 7)                    | Integer        |
| B'right       | Right bound of the vector (e.g., 0)                   | Integer        |
| B'range       | Range of the vector "(7 downto 0)"                    | String         |

## Example: Behavioral Modeling of a Rising Edge Triggered D-Flip-Flop Using Attributes



| Clk | D | Q      |                                    |
|-----|---|--------|------------------------------------|
| 0   | X | Last Q | Store<br>Store<br>Update<br>Update |
| 1   | X | Last Q |                                    |
| ┐   | 0 | 0      |                                    |
| ┐   | 1 | 1      |                                    |

```

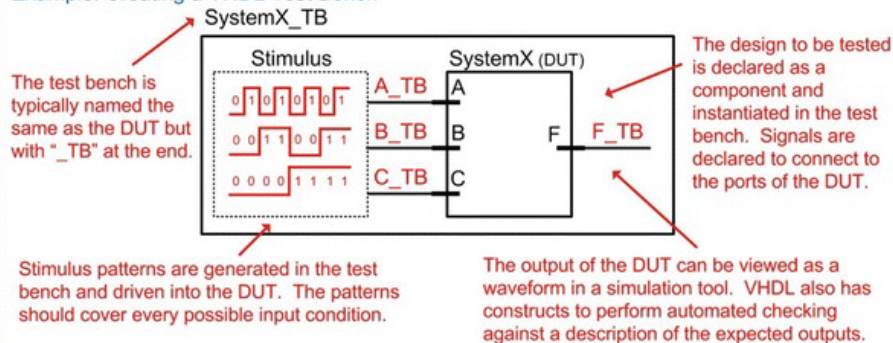
entity Dflipflop is
 port (Clock : in bit;
 D : in bit;
 Q : out bit);
end entity;

architecture Dflipflop_arch of Dflipflop is
begin
 D_FLIP_FLOP : process (Clock)
 _begin
 if (Clock'event and Clock='1') then
 Q <= D;
 end if;
 end process;
end architecture;

```

# Тест бенч

## Example: Creating a VHDL Test Bench



```
entity SystemX_TB is
end entity;

architecture SystemX_TB_arch of SystemX_TB is

 component SystemX
 port (A, B, C : in bit;
 F : out bit);
 end component;

 signal A_TB, B_TB, C_TB : bit;
 signal F_TB : bit;

begin

 DUT1 : SystemX port map (A => A_TB, -- DUT Instantiation
 B => B_TB,
 C => C_TB,
 F => F_TB);

 -- Stimulus Generation
 STIMULUS : process
 begin
 A_TB <= '0'; B_TB <= '0'; C_TB <= '0'; wait for t_wait;
 A_TB <= '0'; B_TB <= '0'; C_TB <= '1'; wait for t_wait;
 A_TB <= '0'; B_TB <= '1'; C_TB <= '0'; wait for t_wait;
 A_TB <= '0'; B_TB <= '1'; C_TB <= '1'; wait for t_wait;
 A_TB <= '1'; B_TB <= '0'; C_TB <= '0'; wait for t_wait;
 A_TB <= '1'; B_TB <= '0'; C_TB <= '1'; wait for t_wait;
 A_TB <= '1'; B_TB <= '1'; C_TB <= '0'; wait for t_wait;
 A_TB <= '1'; B_TB <= '1'; C_TB <= '1'; wait for t_wait;
 end process;

end architecture;
```

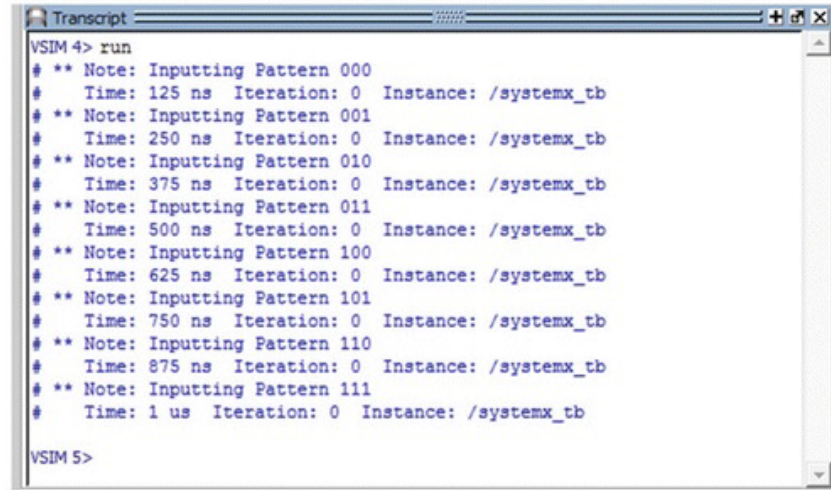
# Report наредба

## Example: Using Report Statements in a VHDL Test Bench

Report statements are inserted in the process to indicate the current stimulus pattern.

```
STIMULUS : process
begin
 A_TB <= '0'; B_TB <= '0'; C_TB <= '0'; wait for 125 ns;
 report "Inputting Pattern 000" severity NOTE;
 A_TB <= '0'; B_TB <= '0'; C_TB <= '1'; wait for 125 ns;
 report "Inputting Pattern 001" severity NOTE;
 A_TB <= '0'; B_TB <= '1'; C_TB <= '0'; wait for 125 ns;
 report "Inputting Pattern 010" severity NOTE;
 A_TB <= '0'; B_TB <= '1'; C_TB <= '1'; wait for 125 ns;
 report "Inputting Pattern 011" severity NOTE;
 A_TB <= '1'; B_TB <= '0'; C_TB <= '0'; wait for 125 ns;
 report "Inputting Pattern 100" severity NOTE;
 A_TB <= '1'; B_TB <= '0'; C_TB <= '1'; wait for 125 ns;
 report "Inputting Pattern 101" severity NOTE;
 A_TB <= '1'; B_TB <= '1'; C_TB <= '0'; wait for 125 ns;
 report "Inputting Pattern 110" severity NOTE;
 A_TB <= '1'; B_TB <= '1'; C_TB <= '1'; wait for 125 ns;
 report "Inputting Pattern 111" severity NOTE;
end process;
```

The following is the transcript showing the results of the report statements.



```
Transcript
VSIM 4> run
** Note: Inputting Pattern 000
Time: 125 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 001
Time: 250 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 010
Time: 375 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 011
Time: 500 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 100
Time: 625 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 101
Time: 750 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 110
Time: 875 ns Iteration: 0 Instance: /systemx_tb
** Note: Inputting Pattern 111
Time: 1 us Iteration: 0 Instance: /systemx_tb
VSIM 5>
```

# Assert наредба

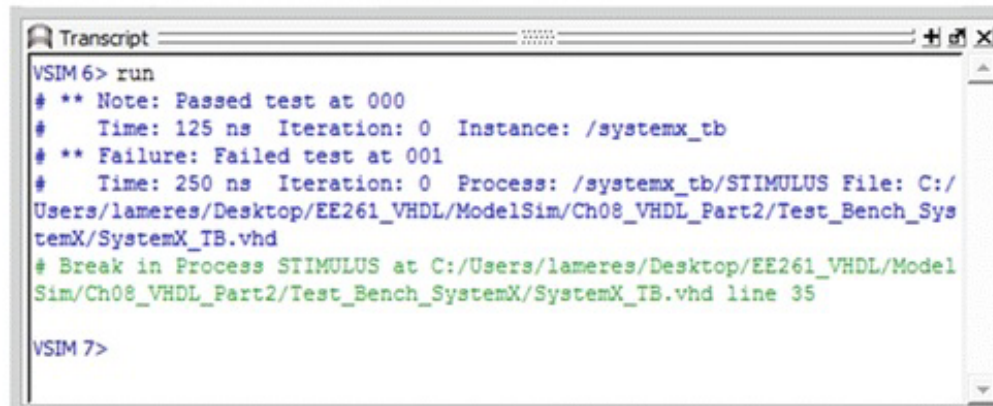
## Example: Using Assert Statements in a VHDL Test Bench

Assert statements are used to check the correctness of the system outputs.

```
STIMULUS : process
begin
 A_TB <= '0'; B_TB <= '0'; C_TB <= '0'; wait for 125 ns;
 assert (F_TB='1') report "Failed test at 000" severity FAILURE;
 assert (F_TB='0') report "Passed test at 000" severity NOTE;

 A_TB <= '0'; B_TB <= '0'; C_TB <= '1'; wait for 125 ns;
 assert (F_TB='1') report "Failed test at 001" severity FAILURE;
 assert (F_TB='0') report "Passed test at 001" severity NOTE;
 :
end process;
```

An intentional failure was introduced at the second input pattern to show how the simulation will end if a report statement is issued with a severity level of FAILURE. The following is the output of the transcript for this case.



```
Transcript
VSIM 6> run
** Note: Passed test at 000
Time: 125 ns Iteration: 0 Instance: /systemx_tb
** Failure: Failed test at 001
Time: 250 ns Iteration: 0 Process: /systemx_tb/STIMULUS File: C:/
Users/lameres/Desktop/EE261_VHDL/ModelSim/Ch08_VHDL_Part2/Test_Bench_Sys
temX/SystemX_TB.vhd
Break in Process STIMULUS at C:/Users/lameres/Desktop/EE261_VHDL/Model
Sim/Ch08_VHDL_Part2/Test_Bench_SystemX/SystemX_TB.vhd line 35
VSIM 7>
```

### Пакети

- Један од недостатака VHDL стандардног пакета је тај што пружа ограничену функционалност у својим типовима података који се могу синтетизовати. Bit и Bit\_vector у току синтезе, немају могућност прецизног моделовања многих топологија имплементираних у савременим дигиталним системима.
- Од примарног интереса су топологије које укључују више кола повезаних на један проводник. Стандардни пакети не дозвољавају ову врсту везе; међутим, ова врста топологије је уобичајна где се врши повезивање више чворова на заједничкој конекцији.
- Осим тога, стандардни пакет не пружа много корисних функција за ове типове, као што је подршка за "don't care", за аритметику помоћу + и - оператора, функције конверзије типова или могућност читања/писања спољних датотека.
- Да би се повећала функционалност VHDL-а, додатни пакети морају да буду укључени
- STD\_LOGIC\_1164
- NUMERIC\_STD
- NUMERIC\_STD\_UNSIGNED
- NUMERIC\_BIT
- NUMERIC\_BIT\_UNSIGNED
- MATH\_REAL
- MATH\_COMPLEX
- TEXTIO and STD\_LOGIC\_TEXTIO
- Legacy Packages (STD\_LOGIC\_ARITH/UNSIGNED/ SIGNED)



## STD\_LOGIC\_1164

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

- std\_ulogic, std\_ulogic\_vector, std\_logic, and std\_logic\_vectorlibrary

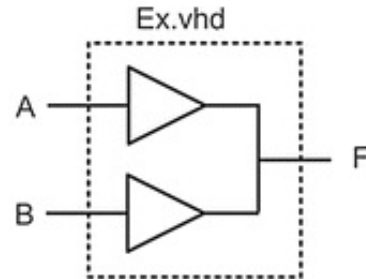
| Value | Description     | Notes                   |
|-------|-----------------|-------------------------|
| U     | Uninitialized   | Default initial value   |
| X     | Forcing unknown |                         |
| 0     | Forcing 0       |                         |
| 1     | Forcing 1       |                         |
| Z     | High impedance  |                         |
| W     | Weak unknown    |                         |
| L     | Weak 0          | Pull-down               |
| H     | Weak 1          | Pull-up                 |
| –     | Don't care      | Used for synthesis only |

```
A <= 'X'; -- assignment to a scalar
(std_ulogic or std_logic)
V <= "01ZH"; -- assignment to a 4-bit vector
(std_ulogic_vector or
 -- std_logic_vector)
```

# STD\_LOGIC\_1164 (2. Aeo)

## STD\_LOGIC\_1164 Unresolved vs. Resolved Conflict Handling

The std\_logic\_1164 package has data types that support this type of topology.



```
architecture Ex_arch of Ex is
begin
 F <= A;
 F <= B;
end architecture;
```

*standard Package  
with types bit*

```
entity Ex is
 port (A,B : in bit;
 F : out bit);
end entity;
```

NOT ALLOWED. This will  
result in compiler error  
"unresolved data type".

*std\_logic\_1164  
package with types  
std\_ulogic*

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

entity Ex is
 port (A,B : in std_ulogic;
 F : out std_ulogic);
end entity;
```

NOT ALLOWED. This will  
result in compiler error  
"unresolved data type".

*std\_logic\_1164  
package with types  
std\_logic*

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

entity Ex is
 port (A,B : in std_logic;
 F : out std_logic);
end entity;
```

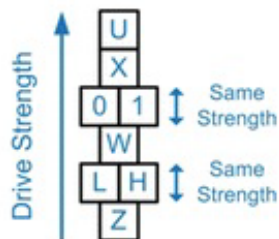
ALLOWED. The output F  
will be determined using a  
*resolution* function.

## STD\_LOGIC Resolution Function

### STD\_LOGIC\_1164 Resolution Function

The std\_logic\_1164 package resolves multiple driver conflict using a resolution function.

#### Relative Drive Strengths



#### Std logic Resolution Function

Driver 2

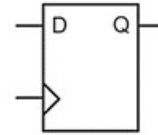
Driver 1

|                 | Uninitialized | Forcing Unknown | Forcing 0 | Forcing 1 | High Impedance | Weak Unknown | Weak 0 | Weak 1 | Don't Care |
|-----------------|---------------|-----------------|-----------|-----------|----------------|--------------|--------|--------|------------|
|                 | U             | X               | 0         | 1         | Z              | W            | L      | H      | -          |
| Uninitialized   | U             | U               | U         | U         | U              | U            | U      | U      | U          |
| Forcing Unknown | X             | U               | X         | X         | X              | X            | X      | X      | X          |
| Forcing 0       | 0             | U               | X         | 0         | X              | 0            | 0      | 0      | X          |
| Forcing 1       | 1             | U               | X         | X         | 1              | 1            | 1      | 1      | X          |
| High Impedance  | Z             | U               | X         | 0         | 1              | Z            | W      | L      | H          |
| Weak Unknown    | W             | U               | X         | 0         | 1              | W            | W      | W      | X          |
| Weak 0          | L             | U               | X         | 0         | 1              | L            | W      | L      | W          |
| Weak 1          | H             | U               | X         | 0         | 1              | H            | W      | W      | H          |
| Don't Care      | -             | U               | X         | X         | X              | X            | X      | X      | X          |

## STD\_LOGIC\_1164

### Logical Operators, Edge Detection Functions

- and, nand, or, nor, xor, xnor, not
- rising\_edge(), falling\_edge()



Example: Behavioral Modeling of a D-Flip-Flop using the rising\_edge() Function

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

entity Dflipflop is
 port (Clock : in std_logic;
 D : in std_logic;
 Q : out std_logic);
end entity;
```

```
architecture Dflipflop_arch of Dflipflop is
begin
 D_FLIP_FLOP : process (Clock)
 _begin
 if (Clock'event and Clock='1') then
 Q <= D;
 end if;
 end process;
end architecture;
```

Equivalent  
Models

```
architecture Dflipflop_arch of Dflipflop is
begin
 D_FLIP_FLOP : process (Clock)
 _begin
 if (rising_edge(Clock)) then
 Q <= D;
 end if;
 end process;
end architecture;
```

## STD\_LOGIC-Type Conversion Functions

| Name                 | Input type        | Return type       |
|----------------------|-------------------|-------------------|
| To_bit()             | std_ulogic        | bit               |
| To_bitvector()       | std_ulogic_vector | bit_vector        |
| To_bitvector()       | std_logic_vector  | bit_vector        |
| To_StdULogic()       | bit               | std_ulogic        |
| To_StdULogicVector() | bit_vector        | std_ulogic_vector |
| To_StdULogicVector() | std_logic_vector  | std_ulogic_vector |
| To_StdLogicVector()  | bit_vector        | std_logic_vector  |
| To_StdLogicVector()  | std_ulogic_vector | std_logic_vector  |

```
A <= To_bit(B);
std_ulogic, A is type bit
V <= To_StdLogicVector(C);
bit_vector, V is std_logic_vector
```

```
-- B is type

-- C is type
```

## NUMERIC\_STD

- Unsigned, signed, +, -, \*, /, mod, rem, and abs
- and, nand, or, nor, xor, xnor, not, shift\_left(), shift\_right(), rotate\_left(), rotate\_right()
- >, <, <=, >=, =, /=,
- rising\_edge(), falling\_edge()

```
library IEEE;
use IEEE.std_logic_1164.all; -- defines types

std_ulogic and std_logic
use IEEE.numeric_std.all; -- defines types
unsigned and signed

 F <= A + B; -- A, B, F are type
unsigned(3 downto 0)
 F <= A - B;

 if (A < B) then -- This condition is TRUE if A
and B are UNSIGNED
 :

 if (A < B) then -- This condition is FALSE if A
and B are SIGNED
```



## NUMERIC\_STD (2. део)

| Name                 | Input type      | Return type                |
|----------------------|-----------------|----------------------------|
| <b>To_integer()</b>  | Unsigned        | Integer                    |
| <b>To_integer()</b>  | Signed          | Integer                    |
| <b>To_unsigned()</b> | integer, <size> | Unsigned (size-1 downto 0) |
| <b>To_signed()</b>   | Integer, <size> | Signed (size-1 downto 0)   |

| Name                      | Input type       | Return type      |
|---------------------------|------------------|------------------|
| <b>std_logic_vector()</b> | Unsigned         | std_logic_vector |
| <b>std_logic_vector()</b> | Signed           | std_logic_vector |
| <b>unsigned()</b>         | std_logic_vector | Unsigned         |
| <b>signed()</b>           | std_logic_vector | Signed           |

```

A <= std_logic_vector(B); -- B is unsigned, A is
std_logic_vector
C <= unsigned(D); -- D is std_logic_vector, C
is unsigned

```

- NUMERIC\_STD\_UNSIGNED

## TEXTIO, STD\_LOGIC\_TEXTIO

## IEEE.textio Package Interpretation of Files

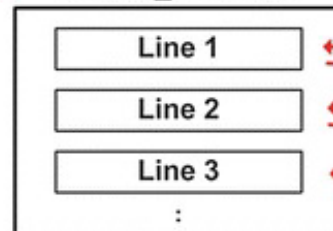
The textio package understands files as a set of *lines*. Each line contains either an integer or a string. Each line is read or written individually in the order they appear in the file.

```
file Fout: TEXT open WRITE_MODE is "output_file.txt";
```

The file handle is "Fout". This handle is used by the writeline() procedure to identify this file.



output\_file.txt



writeline()

VHDL Test Bench  
Line  
Variable

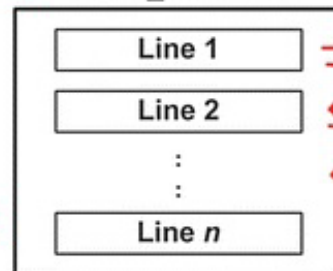
The procedures writeline() transfers information from the VHDL line variable to a new line in the file. Each writeline() call adds a new line to the file.

```
file Fin: TEXT open READ_MODE is "input_file.txt";
```

The file handle is "Fin". This handle is used by the readline() procedure to identify this file.



input\_file.txt



readline()

VHDL Test Bench  
Line  
Variable

The procedures readline() transfers information from a line in the file into the line variable in the VHDL test bench. Each readline() call reads from the next line in the file.

The function endfile() provides a mechanism to determine if the end of the file has been reached during a read.



Ex) while (not endfile(Fin)) loop

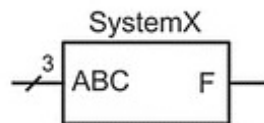


# STD\_LOGIC\_TEXTIO

## Пример

### Example: Writing to an External File from a Test Bench (Part 1)

The following combinational logic circuit is implemented as follows:



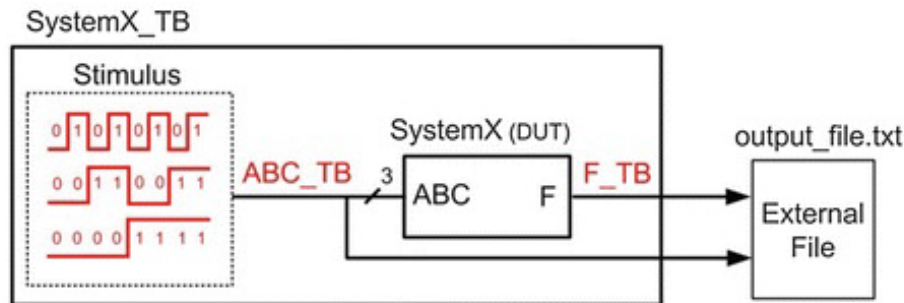
| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

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

entity SystemX is
 port (ABC : in std_logic_vector(2 downto 0);
 F : out std_logic);
end entity;

architecture SystemX_arch of SystemX is
begin
 SystemX_Proc : process (ABC)
 begin
 case (ABC) is
 when "000"|"010"|"110" => F <= '1';
 when others => F <= '0';
 end case;
 end process;
end architecture;
```

A test bench is created to drive in all possible binary codes into the system to test its functionality. The input codes (ABC\_TB) and the DUT output (F\_TB) will be written to an external file called "output\_file.txt".

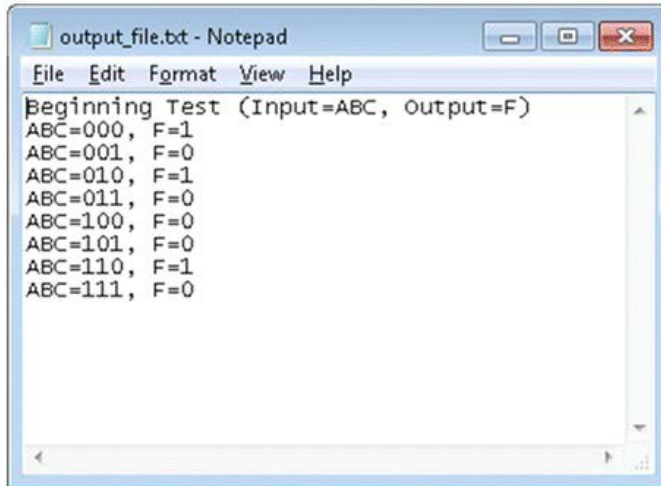


# STD\_LOGIC\_TEXTIO

## Пример (2.део)

### Example: Writing to an External File from a Test Bench (Part 3)

The file "output\_file.txt" is created with the following contents.



```
output_file.txt - Notepad
File Edit Format View Help
Beginning Test (Input=ABC, Output=F)
ABC=000, F=1
ABC=001, F=0
ABC=010, F=1
ABC=011, F=0
ABC=100, F=0
ABC=101, F=0
ABC=110, F=1
ABC=111, F=0
```

### Example: Writing to an External File from a Test Bench (Part 2)

The following test bench is created to perform the testing on SystemX.

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

library STD;
use STD.textio.all;

entity SystemX_TB is
end entity;

architecture SystemX_TB_arch of SystemX_TB is

 component SystemX
 port (ABC : in std_logic_vector(2 downto 0);
 F : out std_logic);
 end component;

 signal ABC_TB : std_logic_vector(2 downto 0);
 signal F_TB : std_logic;

begin
 DUT : SystemX port map (ABC => ABC_TB, F => F_TB);

 STIMULUS : process

 file Fout: TEXT open WRITE_MODE is "output_file.txt";
 variable current_line : line;

 begin

 write(current_line, string'("Beginning Test (Input=ABC, Output=F)"));
 writeline(Fout, current_line);

 ABC_TB <= "000"; wait for 125 ns;

 write(current_line, string'("ABC="));
 write(current_line, ABC_TB);
 write(current_line, string'(", F="));
 write(current_line, F_TB);
 writeline(Fout, current_line);

 ABC_TB <= "001"; wait for t_wait;

 write(current_line, string'("ABC="));
 write(current_line, ABC_TB);
 write(current_line, string'(", F="));
 write(current_line, F_TB);
 writeline(Fout, current_line);

 wait;

 end process;
end architecture;
```

The std\_logic\_textio and textio packages are included to support external I/O access.

Declaration of DUT

Declaration of signals to connect to DUT

Instantiation of DUT

Declare file for writing

Declare line variable

This write() procedure adds text to the line variable.

This writeline() procedure writes the contents of the line variable to the file.

Set first input pattern.

Write input pattern, output value and descriptive text to the line variable and then write the line variable to the file using writeline().

Repeat for next pattern.

Repeat for all other possible inputs (not shown for brevity).