Логичко пројектовање

Предавање 6/10

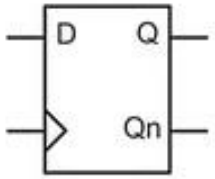
Опис секвенцијалних мрежа у VHDL-у

Увод

- У овој лекцији се врши моделовање секвенцијалних мрежа коришћењем VHDL техника представњених у претходној лекцији.
- VHDL моделовање секвенијалних меморијских елемената.
- VHDL моделовање коначних аутомата.
- VHDL моделовање регистара за трансфер података.
- Разумевање како се VHDL користи за креирање понашања секвенцијалних дигиталних система.

D-флип-флоп

Example: Behavioral Model of a D-Flip-Flop in VHDL

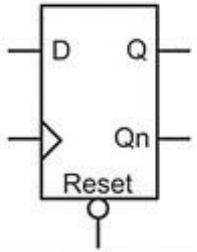


Clk	D	Q	Qn	
0	X	Last Q	Last Qn	Store
1	X	Last Q	Last Qn	Store
↓	0	0	1	Update
↓	1	1	0	Update

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

entity Dflipflop is
    port    (Clock          : in    std_logic;
             D              : in    std_logic;
             Q, Qn          : 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;    Qn <= not D;
          end if;
    end process;

end architecture;
```

D-флип-флоп са асин. ресетом

Example: Behavioral Model of a D-Flip-Flop with Asynchronous Reset in VHDL

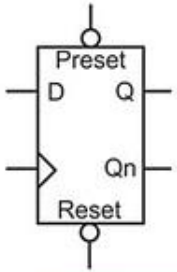


$\overline{R}$	Clk	D	Q	Qn	
0	X	X	0	1	Reset
1	0	X	Last Q	Last Qn	Store
1	1	X	Last Q	Last Qn	Store
1	↓	0	0	1	Update
1	↓	1	1	0	Update

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

entity Dflipflop is
    port    (Clock          : in    std_logic;
             Reset          : in    std_logic;
             D              : in    std_logic;
             Q, Qn          : out   std_logic);
end entity;

architecture Dflipflop_arch of Dflipflop is

  begin

     D_FLIP_FLOP : process (Clock, Reset)
       begin
          if (Reset = '0') then
              Q <= '0'; Qn <= '1';
          elsif (Clock'event and Clock='1') then
              Q <= D;    Qn <= not D;
          end if;
     end process;

end architecture;
```

D-флип-флоп са асин. ресетом и пресетом

Example: Behavioral Model of a D-Flip-Flop with Asynchronous Reset and Preset in VHDL

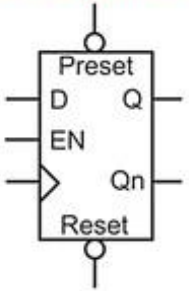


R	P	Clk	D	Q	Qn	
0	X	X	X	0	1	Reset
1	0	X	X	1	0	Preset
1	1	0	X	Last Q	Last Qn	Store
1	1	1	X	Last Q	Last Qn	Store
1	1	↓	0	0	1	Update
1	1	↓	1	1	0	Update

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

entity Dflipflop is
    port   (Clock           : in   std_logic;
            Reset, Preset   : in   std_logic;
            D               : in   std_logic;
            Q, Qn           : out  std_logic);
end entity;

architecture Dflipflop_arch of Dflipflop is

  begin

     D_FLIP_FLOP : process (Clock, Reset, Preset)
       begin
          if (Reset = '0') then
              Q <= '0'; Qn <= '1';
          elsif (Preset = '0') then
              Q <= '1'; Qn <= '0';
          elsif (Clock'event and Clock='1') then
              Q <= D;   Qn <= not D;
          end if;
     end process;

end architecture;
```

D-флип-флоп са синх. Enable

Example: Behavioral Model of a D-Flip-Flop with Synchronous Enable in VHDL



$\overline{R}$	$\overline{P}$	Clk	EN	D	Q	Qn	
0	X	X	X	X	0	1	Reset
1	0	X	X	X	1	0	Preset
1	1	0	X	X	Last Q	Last Qn	Store
1	1	1	X	X	Last Q	Last Qn	Store
1	1	↓	0	X	Last Q	Last Qn	Disabled (ignore clock)
1	1	↓	1	0	0	1	Update
1	1	↓	1	1	1	0	Update

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

entity Dflipflop is
    port    (Clock              : in    std_logic;
             Reset, Preset      : in    std_logic;
             D, EN              : in    std_logic;
             Q, Qn              : out   std_logic);
end entity;

architecture Dflipflop_arch of Dflipflop is

  begin

     D_FLIP_FLOP : process (Clock, Reset, Preset)
       begin
          if (Reset = '0') then
              Q <= '0'; Qn <= '1';
          elsif (Preset = '0') then
              Q <= '1'; Qn <= '0';
          elsif (Clock'event and Clock='1') then
              if (EN = '1') then
                 Q <= D;   Qn <= not D;
              end if;
          end if;
     end process;

end architecture;
```

A nested if/then statement is used to model the synchronous enable.

Моделовање коначних аутомата у VHDL-у

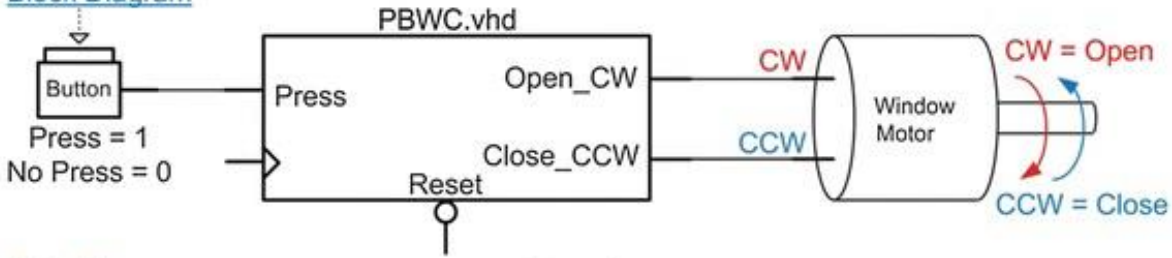
- Коначни аутомати могу се лако моделовати коришћењем конструкција описа понашања из претходне лекције.
- Најчешћа пракса моделирања за FSM је креирање новог кориснички дефинисаног типа који може да преузме описна имена стања из дијаграма стања.
- Затим се креирају два сигнала овог типа, current_state и next_state.
- Једном када су ови сигнали креирани, сви функционални блокови у коначном аутомату могу да користе описна имена стања у њиховим условним додељивањима сигнала.
- Логичка синтеза ће аутоматски доделити кодове стања на основу технологије која се користи (нпр. binary, gray code, one-hot).
- У оквиру VHDL модела коначног аутомата, користе се 3 процеса за описивање сваког од функционалних блокова, меморије стања, логике следећег стања и излазне логике.
- Да би се испитало како да моделујемо аутомат користићемо пример push дугмета (пример контролера прозора из претходне лекције).

Моделовање коначних аутомата у VHDL-у (пример)

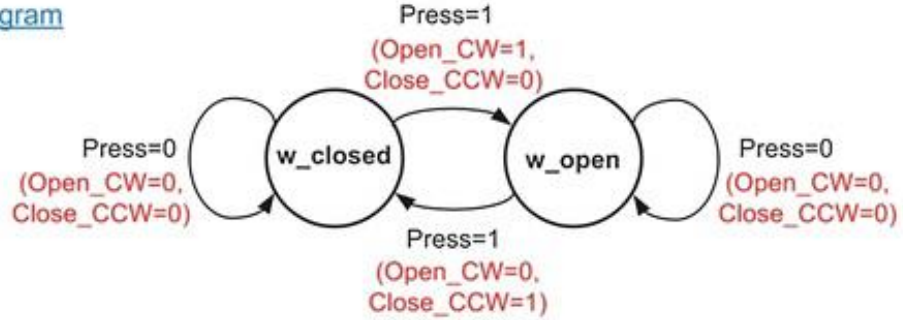
Example: Push-Button Window Controller in VHDL – Design Description

The window controller will send the appropriate control signals to a motor to open or close it whenever a button is pressed. The system must keep track whether the window is open or closed in order to send the correct signal, thus a state machine is needed. The block diagram and state diagram for this system is shown below.

Block Diagram

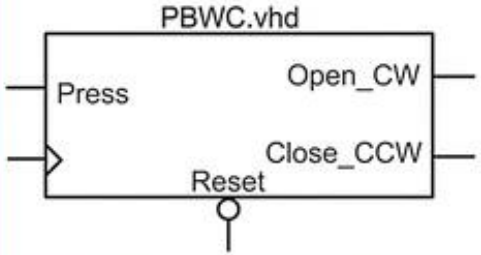


State Diagram



Моделовање коначних аутомата у VHDL-у (пример 2. део)

Example: Push-Button Window Controller in VHDL – Entity Definition




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

entity PBWC is
  port (Clock, Reset        : in  std_logic;
        Press               : in  std_logic;
        Open_CW, Close_CCW  : out std_logic);
end entity;
```

Моделовање стања

- Први корак је креирање новог кориснички дефинисаног, набројивог типа података који може да има вредности које одговарају описним именима стања.
- Имена стања узимао из дијаграма стања (w_closed и w_open).
- Ово је постигнуто декларисањем новог типа пре почетка наредбе у архитектури са кључном речи “type”.
- За овај пример, креираћемо нови тип под називом State_Type и експлицитно набројиве вредности које може да узме овај тип.
- Затим креирамо два нова сигнала названа current_state и next_state. Ова два сигнала ће се користити у целом VHDL моделу.
- Следећа синтакса показује како декларисати нови тип и сигнале current_state и next_state:

```
type State_Type is (w_closed, w_open);
signal current_state, next_state : State_Type;
```

Процес меморије стања

- Овај процес моделира понашање D-флип-флопова у FSM-у који задржавају вредности стања на Q излазима.
- Сваки пут када постоји растућа ивица clock-а, тренутно стање се ажурира са следећом вредношћу стања присутном на D улазима D-флип-флопова.
- Овај процес такође мора да моделира стање ресетовања.
- За пример push дугмета аутомат ће прећи у стање w_closed када се потврди ресетовање.
- У свим другим случајевима, процес ће једноставно ажурирати current_state са next_state на свакоу узлазну ивицу clock-а.
- Модел процеса је веома сличан моделу D-флип-флопа. Ово је очекивано јер ће се овај процес синтетизовати у један или више D-флип-флопа за задржавање тренутног стања.
- Листа осетљивости садржи само Clock и Reset и додељивање се врши само за сигнал current_state.
- Следећи пример показује како се моделује меморија стања овог FSM-а:

```
STATE_MEMORY : process (Clock, Reset)
begin
if (Reset = '0') then
current_state <= w_closed;
elsif (Clock'event and Clock='1') then
current_state <= next_state;
end if;
end process;
```

Процес следећег стања

- Логика следећег стања је комбинациона логика; стога морамо укључити све улазне сигнале који се користе за генерисање следећег стања у листу осетљивости процеса.
- Сигнал current_state ће увек бити укључен у листу осетљивости за процес следећег стања поред свих улаза у систем.
- На пример, систем има још један улаз који се зове Press. Овај процес kreira доделе сигналу next_state.
- Уобичајено је користити case наредбу са доделама за сваки излаз посебно.
- У сваком стању у оквиру case наредбе, користи се if/else наредба за моделовање доделе за различите улазе за Press.
- Следећи пример показује како се моделује логика следећег стања стања за пример FSM-а.

```
NEXT_STATE_LOGIC : process (current_state, Press)
begin
case (current_state) is
when w_closed => if (Press = '1') then
next_state <= w_open;
else
next_state <= w_closed;
end if;
when w_open => if (Press = '1') then
next_state <= w_closed;
else
next_state <= w_open;
end if;
when others => next_state <= w_closed;
end case;
end process;
```

Процес излазне логике

- Излазна логика је комбинациона логика, стога морамо укључити све улазне сигнале који учествују у генерисању излаза.
- Current_state сигнали ће увек бити укључени у листу осетљивости.
- Ако је аутомат Милијевог типа, тада ће и улази бити укључени у листу осетљивости. Ако је аутомат Муровог типа, онда ће само Current_state сигнали бити укључени у листу осетљивости.
- Пример Press дугмета је аутомат Милијевог типа, тако да улаз Press треба да буде укључен у листу осетљивости.
- Овај процес има доделе само за излазе аутомата (Open_CW и Close_CCW).
- Следећи пример показује како се моделује излазна логика за пример FSM-а.

```
OUTPUT_LOGIC : process (current_state, Press)
begin
case (current_state) is
when w_closed => if (Press = '1') then
Open_CW <= '1'; Close_CCW <= '0';
else
Open_CW <= '0'; Close_CCW <= '0';
end if;

when w_open => if (Press = '1') then
Open_CW <= '0'; Close_CCW <= '1';
else
Open_CW <= '0'; Close_CCW <= '0';
end if;

          when others => Open_CW <= '0'; Close_CCW <=
'0';
end case;
end process;
```

Процес излазне логике

- Излазна логика је комбинациона логика, стога морамо укључити све улазне сигнале који учествују у генерисању излаза.
- Current_state сигнали ће увек бити укључени у листу осетљивости.
- Ако је аутомат Милијевог типа, тада ће и улази бити укључени у листу осетљивости. Ако је аутомат Муровог типа, онда ће само Current_state сигнали бити укључени у листу осетљивости.
- Пример Press дугмета је аутомат Милијевог типа, тако да улаз Press треба да буде укључен у листу осетљивости.
- Овај процес има доделе само за излазе аутомата (Open_CW и Close_CCW).
- Следећи пример показује како се моделује излазна логика за пример FSM-а.

```
OUTPUT_LOGIC : process (current_state, Press)
begin
case (current_state) is
when w_closed => if (Press = '1') then
Open_CW <= '1'; Close_CCW <= '0';
else
Open_CW <= '0'; Close_CCW <= '0';
end if;

when w_open => if (Press = '1') then
Open_CW <= '0'; Close_CCW <= '1';
else
Open_CW <= '0'; Close_CCW <= '0';
end if;

          when others => Open_CW <= '0'; Close_CCW <=
'0';
end case;
end process;
```

Моделовање коначних аутомата у VHDL-у (пример 3. део)

Example: Push-Button Window Controller in VHDL – Architecture

```
architecture PBWC_arch of PBWC is

 type   State_Type is (w_closed, w_open);
 signal current_state, next_state : State_Type;

begin
-------------------------------------------------------
  STATE_MEMORY : process (Clock, Reset)
    begin
      if (Reset = '0') then
        current_state <= w_closed;
      elsif (Clock'event and Clock='1') then
        current_state <= next_state;
      end if;
  end process;
-------------------------------------------------------
  NEXT_STATE_LOGIC : process (current_state, Press)
    begin
      case (current_state) is
        when w_closed => if (Press = '1') then
                            next_state <= w_open;
                         else
                            next_state <= w_closed;
                         end if;
        when w_open   => if (Press = '1') then
                            next_state <= w_closed;
                         else
                            next_state <= w_open;
                         end if;
        when others   => next_state <= w_closed;
      end case;
  end process;
-------------------------------------------------------
```

Declaration of user defined type for the signals current_state and next_state.

The first process is used to model the state memory.

The second process is used to model the next state logic.

```
-------------------------------------------------------
  OUTPUT_LOGIC : process (current_state, Press)
    begin
      case (current_state) is
        when w_closed => if (Press = '1') then
                            Open_CW <= '1'; Close_CCW <= '0';
                         else
                            Open_CW <= '0'; Close_CCW <= '0';
                         end if;
        when w_open   => if (Press = '1') then
                            Open_CW <= '0'; Close_CCW <= '1';
                         else
                            Open_CW <= '0'; Close_CCW <= '0';
                         end if;
        when others   => Open_CW <= '0'; Close_CCW <= '0';
      end case;
  end process;

end architecture;
```

The th
used t
output

Експлицитно дефинисање кодирање стања са Subtypes

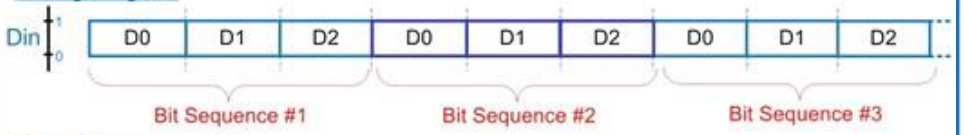
```
subtype State_Type is std_logic;
constant w_open : State_Type := '0';
constant w_closed : State_Type := '1';
signal current_state, next_state : State_Type;
```

Серијски бит детектор секвенце у VHDL-у

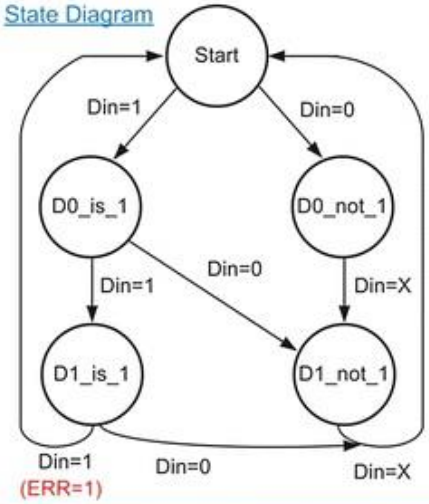
Example: Serial Bit Sequence Detector in VHDL – Design Description and Entity Definition

This circuit will monitor an incoming serial bit stream . The information in the bit stream represents data in groups of 3-bits. The code "111" represents that an error has occurred in the transmitter. The FSM will monitor the incoming bit stream and assert a signal called "ERR" if the sequence "111" is detected. At all other times ERR=0.

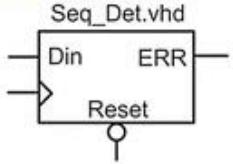
Timing Diagram



State Diagram



Entity Definition




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

entity Seq_Det is
  port (Clock, Reset : in  std_logic;
        Din          : in  std_logic;
        ERR          : out std_logic);
end entity;
```

Example: Serial Bit Sequence Detector in VHDL – Architecture

```
architecture Seq_Det_arch of Seq_Det is

 type   State_Type is (Start, D0_is_1, D1_is_1, D0_not_1, D1_not_1);
 signal current_state, next_state : State_Type;

 begin

-------------------------------------------------------------
  STATE_MEMORY : process (Clock, Reset)
    begin
      if (Reset = '0') then
        current_state <= Start;
      elsif (Clock'event and Clock='1') then
        current_state <= next_state;
      end if;
  end process;
-------------------------------------------------------------
  NEXT_STATE_LOGIC : process (current_state, Din)
    begin
      case (current_state) is
        when Start    => if (Din = '1') then
                           next_state <= D0_is_1;
                         else
                           next_state <= D0_not_1;
                         end if;
        when D0_is_1  => if (Din = '1') then
                           next_state <= D1_is_1;
                         else
                           next_state <= D1_not_1;
                         end if;
        when D1_is_1  => next_state <= Start;
        when D0_not_1 => next_state <= D1_not_1;
        when D1_not_1 => next_state <= Start;
        when others   => next_state <= Start;
      end case;
  end process;
-------------------------------------------------------------
  OUTPUT_LOGIC : process (current_state, Din)
    begin
      case (current_state) is
        when D1_is_1  => if (Din = '1') then
                           ERR <= '1';
                         else
                           ERR <= '0';
                         end if;
        when others   => ERR <= '0';
      end case;
  end process;

end architecture;
```

Declaration of user defined type for the signals current_state and next_state.

Note that in this example there are states decisions that don't require if/then statements.

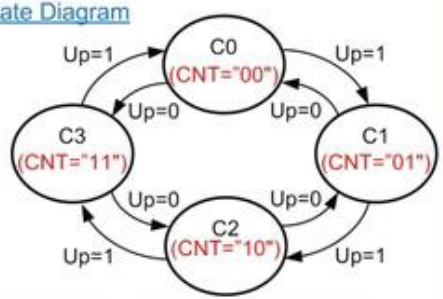
This is a Meally machine so both the current state and the system inputs are present in the sensitivity list.

2-битни бинарни кружни бројач навише/наниже у VHDL-у

Example: 2-Bit Binary Up/Down Counter in VHDL – Design Description and Entity Definition

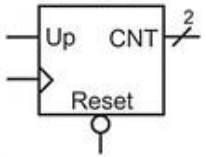
This system will output a synchronous, 2-bit, binary counter. When the system input Up=1, the system will count up. When Up=0, the sytem will count down. The output of the counter is called CNT.

State Diagram



Entity Definition

Counter_2bit_UpDown.vhd




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

entity Counter_2bit_UpDown is
 port (Clock : in  std_logic;
       Reset : in  std_logic;
       Up    : in  std_logic;
       CNT   : out std_logic_vector(1 downto 0));
end entity;
```

Example: 2-Bit Binary Up/Down Counter in VHDL – Architecture (Three Process Model)

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

entity Counter_2bit_UpDown is
 port (Clock, Reset : in  std_logic;
       Up           : in  std_logic;
       CNT          : out std_logic_vector(1 downto 0));
end entity;

architecture Counter_2bit_UpDown_arch of Counter_2bit_UpDown is

 type   State_Type is (C0, C1, C2, C3);
 signal current_state, next_state : State_Type;

 begin
-------------------------------------------------------------
  STATE_MEMORY : process (Clock, Reset)
    begin
      if (Reset = '0') then
        current_state <= C0;
      elsif (Clock'event and Clock='1') then
        current_state <= next_state;
      end if;
  end process;
-------------------------------------------------------------
  NEXT_STATE_LOGIC : process (current_state, Up)
    begin
      case (current_state) is
        when C0     => if (Up = '1') then
                         next_state <= C1;
                       else
                         next_state <= C3;
                       end if;
        when C1     => if (Up = '1') then
                         next_state <= C2;
                       else
                         next_state <= C0;
                       end if;
        when C2     => if (Up = '1') then
                         next_state <= C3;
                       else
                         next_state <= C1;
                       end if;
        when C3     => if (Up = '1') then
                         next_state <= C0;
                       else
                         next_state <= C2;
                       end if;
        when others => next_state <= C0;
      end case;
  end process;
-------------------------------------------------------------
  OUTPUT_LOGIC : process (current_state)
    begin
      case (current_state) is
        when C0     => CNT <= "00";
        when C1     => CNT <= "01";
        when C2     => CNT <= "10";
        when C3     => CNT <= "11";
        when others => CNT <= "00";
      end case;
  end process;

end architecture;
```

A counter is a Moore machine so the output only depends on the current state.

Бројач у VHDL-у - тип UNSIGNED

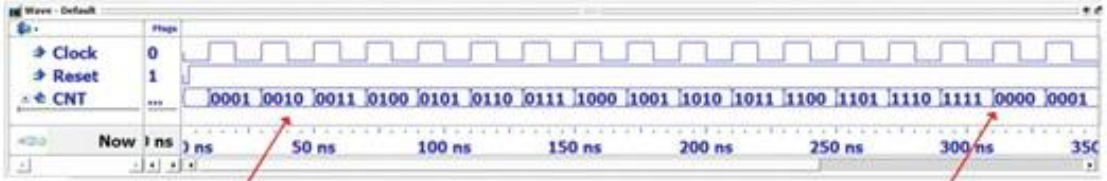
Example: 4-Bit Binary Up Counter in VHDL Using the Type UNSIGNED

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

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out unsigned(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_tmp : unsigned(3 downto 0);

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_tmp <= "0000";
      elsif (Clock'event and Clock='1') then
        CNT_tmp <= CNT_tmp + 1;
      end if;
  end process;

  CNT <= CNT_tmp;

end architecture;
```

The numeric_std package is needed to include the "+" operator. This operator only works on types signed/unsigned, so we will define the output CNT as type unsigned.

An internal signal is needed to support assignments in the form C <= C+1; because a port cannot be used as an argument in a signal assignment.

A concurrent signal assignment is used to continually assign CNT_tmp to CNT.



The counter increments on each rising edge of clock.

When the counter reaches "1111", it rolls over to "0000" and continues.

Бројач у VHDL-у - тип INTEGER

Example: 4-Bit Binary Up Counter in VHDL Using the Type INTEGER

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

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out unsigned(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_int : integer;

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_int <= 0;
      elsif (Clock'event and Clock='1') then

        if (CNT_int = 15) then
          CNT_int <= 0;
        else
          CNT_int <= CNT_int + 1;
        end if;

      end if;
  end process;

  CNT <= to_unsigned(CNT_int, 4);

end architecture;
```

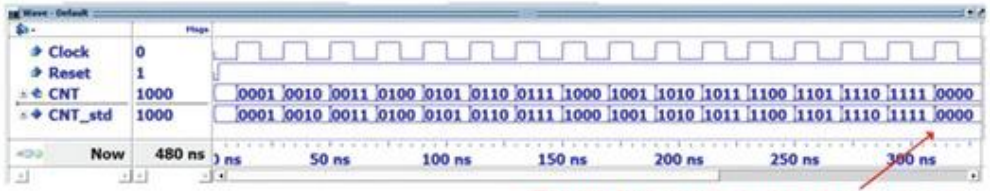
The numeric_std package contains the "+" operator for type integer and a conversion from type integer to type unsigned.

In this example, the output port is defined to be of type unsigned.

An internal signal of type integer is declared to model the counter functionality.

A nested if/then statement checks to see if the integer counter has reached its maximum value.

A concurrent assignment between the internal counter and the output port is made that contains the conversion between type integer and unsigned. The 4 in this function represents the number of unsigned bits to convert the integer into.



The std_logic_vector is treated as unsigned and will roll over once it gets to "1111".

Бројач у VHDL-у - тип STD_LOGIC_VECTOR

Example: 4-Bit Binary Up Counter in VHDL Using the Type STD_LOGIC_VECTOR (1)

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

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out std_logic_vector(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_std : std_logic_vector(3 downto 0);

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_std <= "0000";
      elsif (Clock'event and Clock='1') then
        CNT_std <= CNT_std + 1;
      end if;
  end process;

  CNT <= CNT_std;

end architecture;
```

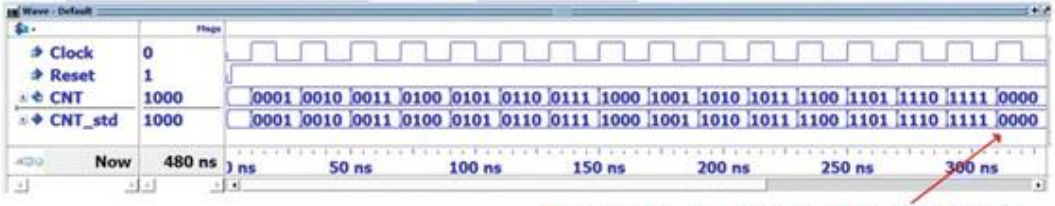
Including this package will treat all std_logic_vector types as unsigned numbers.

The output port is defined to be of type std_logic_vector.

The internal signal to model the counter behavior is declared as type std_logic_vector.

No boundary checking is needed since the 4-bit std_logic_vector will simply roll over.

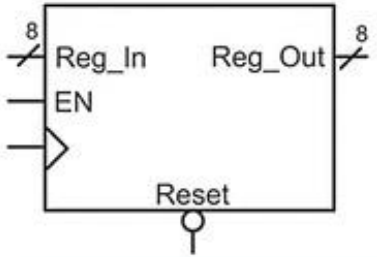
No type conversion is needed since the internal signal and output port are of type std_logic_vector.



The std_logic_vector is treated as unsigned and will roll over once it gets to "1111".

Регистри у VHDL-у

Example: RTL Model of an 8-Bit Register in VHDL

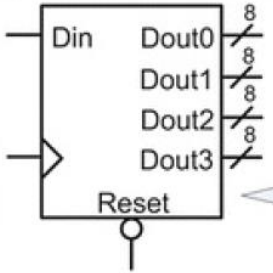


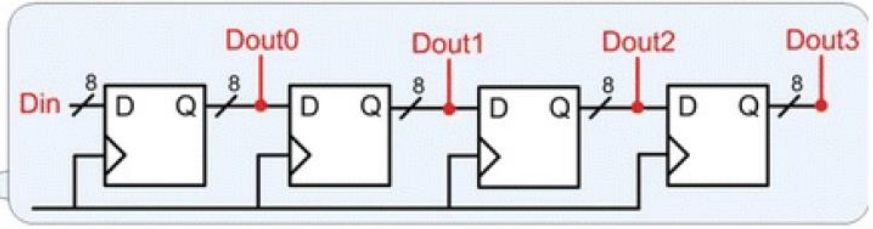
$\overline{R}$	Clk	EN	Reg_Out	
0	X	X	x"00"	Reset
1	X	0	Last Reg_Out	Disabled (ignore clock)
1	0	1	Last Reg_Out	Store
1	1	1	Last Reg_Out	Store
1	↓	1	Reg_In	Update

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

entity reg is
    port    (Clock          : in    std_logic;
             Reset          : in    std_logic;
             Reg_In         : in    std_logic_vector(7 downto 0);
             EN             : in    std_logic;
             Reg_Out        : out   std_logic_vector(7 downto 0));
end entity;

architecture reg_arch of reg is

 begin

  Reg_Proc : process (Clock, Reset)
    begin
      if (Reset = '0') then
         Reg_Out <= x"00";
      elsif (Clock'event and Clock='1') then
          if (EN = '1') then
            Reg_Out <= Reg_In;
          end if;
      end if;
   end process;

end architecture;
```

Shift регистри у VHDL-у

Example: RTL Model of a 4-Stage, 8-Bit Shift Register in VHDL






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

entity Shift_Register is
 port (Clock, Reset : in  std_logic;
       Din          : in  std_logic_vector(7 downto 0);
       Dout0, Dout1 : out std_logic_vector(7 downto 0);
       Dout2, Dout3 : out std_logic_vector(7 downto 0));
end entity;

architecture Shift_Register_arch of Shift_Register is

 signal D0, D1, D2, D3 : std_logic_vector(7 downto 0);

 begin

  SHIFT : process (Clock, Reset)
    begin
      if (Reset = '0') then
        D0 <= x"00"; D1 <= x"00"; D2 <= x"00"; D3 <= x"00";
      elsif (Clock'event and Clock='1') then
        D0 <= Din; D1 <= D0; D2 <= D1; D3 <= D2;
      end if;
  end process;

  Dout3 <= D3; Dout2 <= D2; Dout1 <= D1; Dout0 <= D0;

end architecture;
```