**1 ФОРМУВАННЯ VHDL ОПИСУ ФРАГМЕНТУ СХЕМИ**

1.1 Мета роботи

Вивчення структури VHDL опису комбінаційної схеми – базового компонента SoC. Отримання практичних навичок створення поведінкового опису комбінаційної схеми та її моделювання засобами мови VHDL.

1.2 Методичні вказівки з організації самостійної роботи студентів

Під час підготовки до лабораторної роботи слід ознайомитися з літературою [1-2], матеріалами конспекту лекцій та інформацією, наведеною в підрозділі 1.4.

1.3 Опис лабораторної установки

Cloud service for design of digital devices

1. Login Page:

<https://www.edaplayground.com/login>

A screenshot of a social media post

Description automatically generated

2. The main window.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

Open Wave Form after simulation

Options of debug and simulation

Tools and simulators

Libraries

Hardware description language (VHDL or Verilog)

Run simulation

Fig. 1. Main window

Download file after simulation

Examples of the projects

1.4 Порядок виконання роботи

Приклад опису фрагмента схеми пристрою (рис. \_\_).

A picture containing monitor, screen, television, sitting

Description automatically generated

Listing of VHDL code for Testbench and project of the circuit are shown below.

|  |  |
| --- | --- |
| -- Testbench for Simple And-OR circuit design  library IEEE;  use IEEE.std\_logic\_1164.all;    entity testbench is  -- empty  end testbench;  architecture tb of testbench is  -- DUT component  component and\_or\_gate is  port(  a: in std\_logic;  b: in std\_logic;  c: in std\_logic;  q: out std\_logic;  q2: out std\_logic);  end component;  signal a\_in, b\_in, c\_in, q\_out, q2\_out: std\_logic;  begin  -- Connect DUT  DUT: and\_or\_gate port map(a\_in, b\_in, c\_in, q\_out, q2\_out);  process  begin  a\_in <= '0';  b\_in <= '0';  c\_in <= '0';  wait for 1 ns;  assert(q\_out='1') report "Fail 0/0" severity error;    a\_in <= '0';  b\_in <= '1';  c\_in <= '1';  wait for 1 ns;  assert(q\_out='0') report "Fail 0/1" severity error;  a\_in <= '1';  b\_in <= 'X';  c\_in <= '1';  wait for 1 ns;  assert(q\_out='0') report "Fail 1/X" severity error;  a\_in <= '1';  b\_in <= '1';  c\_in <= '1';  wait for 1 ns;  assert(q\_out='0') report "Fail 1/1" severity error;    -- Clear inputs  a\_in <= '0';  b\_in <= '0';  c\_in <= '0';  assert false report "Test done." severity note;  wait;  end process;  end tb; | -- Simple And-OR circuit design  library IEEE;  use IEEE.std\_logic\_1164.all;  entity and\_or\_gate is  port(  a: in std\_logic;  b: in std\_logic;  c: in std\_logic;  q: out std\_logic;  q2: out std\_logic);  end and\_or\_gate;  architecture rtl of and\_or\_gate is  begin  process(a, b, c) is  begin  q <= a and b;  q2 <= q or c;  end process;  end rtl; |

A screenshot of a computer

Description automatically generated

Рис. \_\_ Результати моделювання

1.4 Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання.

2. Здійснити опис функціональності.

3. Виконати моделювання проекту засобами мови VHDL.

4. Оформити звіт.

1.5 Варіанти завдання

1) Text

Description automatically generated

2) A picture containing text, clipart

Description automatically generated

3) A picture containing text, clipart

Description automatically generated

4) A picture containing text

Description automatically generated

5) A picture containing text, clipart

Description automatically generated

6) A picture containing text

Description automatically generated

7) A picture containing shape

Description automatically generated

1.6 Контрольні запитання

1. Дайте визначення системи на кристалі. Опишіть основні компоненти SoC.

2. Опишіть 3D-архітектури систем на кристалах.

3. Опишіть процес проектування систем на кристалах.

4. Опишіть криву графіка створення систем на кристалах, що включає фази проектування, налагодження і виробництва.

5. Опишіть технологію виготовлення і тестування цифрових систем в пакетах кристалів.

6. Опишіть нові тренди Gartner's Hype Emerging Technologies Cycle 2021.

7. Опишіть структуру мобільного спеціалізованого цифрового пристрою.

8. Опишіть архітектури кристалів програмованої логіки (PLD).

9. Опишіть архітектуру вентильних матриць FPGA.

10. Опишіть технології проектування цифрових систем на кристалах.

11. Опишіть особливості проектування спеціалізованих інтегральних схем ASIC.

**2 ФОРМУВАННЯ VHDL ОПИСУ СУМАТОРА**

2.1 Мета роботи

Вивчення структури VHDL опису суматора – базового компонента SoC. Отримання практичних навичок створення поведінкового опису суматора та його моделювання засобами мови VHDL.

2.2 Методичні вказівки з організації самостійної роботи студентів

Під час підготовки до лабораторної роботи слід ознайомитися з літературою [1-4], матеріалами конспекту лекцій та інформацією, наведеною в підрозділі 2.4.

2.3 Опис лабораторної установки

Каскадний суматор (Ripple-carry adder) – це логічна схема, яка виконує додавання багатобітових двійкових чисел.

2.3.1 Приклад VHDL опису масштабованого каскадного суматора (Scalable Ripple Carry Adder) з використанням параметризації наведено нижче. В якості параметра використовується g\_WIDTH – кількість розрядів або ширина вхідного слова.

|  |
| --- |
| library IEEE;  use IEEE.std\_logic\_1164.all;  entity full\_adder is  port (i\_bit1, i\_bit2, i\_carry: in std\_logic; -- Inputs  o\_sum, o\_carry: out std\_logic); -- Outputs  end full\_adder;  architecture Equations of full\_adder is  signal A1, A2, A3 : std\_logic; -- input signals  begin -- Parallel assignment statements  o\_sum <= i\_bit1 xor i\_bit2 xor i\_carry;  A1 <= i\_bit1 and i\_bit2; -- gives g  A2 <= i\_bit1 and i\_carry;  A3 <= i\_bit2 and i\_carry;  o\_carry <= A1 or A2 or A3;  end Equations;  -------------------------------------------------------------------------------  -- File Downloaded from <http://www.nandland.com>  -------------------------------------------------------------------------------  library ieee;  use ieee.std\_logic\_1164.all;    entity ripple\_carry\_adder is    generic (      g\_WIDTH : natural := 2      );    port (      i\_add\_term1  : in std\_logic\_vector(g\_WIDTH-1 downto 0);      i\_add\_term2  : in std\_logic\_vector(g\_WIDTH-1 downto 0);      --      o\_result   : out std\_logic\_vector(g\_WIDTH downto 0)      );  end ripple\_carry\_adder;    architecture rtl of ripple\_carry\_adder is      component full\_adder is      port (        i\_bit1  : in  std\_logic;        i\_bit2  : in  std\_logic;        i\_carry : in  std\_logic;        o\_sum   : out std\_logic;        o\_carry : out std\_logic);    end component full\_adder;      signal w\_CARRY : std\_logic\_vector(g\_WIDTH downto 0);    signal w\_SUM   : std\_logic\_vector(g\_WIDTH-1 downto 0);    begin      w\_CARRY(0) <= '0';                    -- no carry input on first full adder      SET\_WIDTH : for ii in 0 to g\_WIDTH-1 generate      i\_FULL\_ADDER\_INST : full\_adder        port map (          i\_bit1  => i\_add\_term1(ii),          i\_bit2  => i\_add\_term2(ii),          i\_carry => w\_CARRY(ii),          o\_sum   => w\_SUM(ii),          o\_carry => w\_CARRY(ii+1)          );    end generate SET\_WIDTH;      o\_result <= w\_CARRY(g\_WIDTH) & w\_SUM;  -- VHDL Concatenation    end rtl; |

A close up of a device

Description automatically generated

**VHDL Testbench:**

|  |
| --- |
| -------------------------------------------------------------------------------  -- File Downloaded from http://www.nandland.com  -------------------------------------------------------------------------------  library ieee;  use ieee.std\_logic\_1164.all;    entity ripple\_carry\_adder\_tb is  end ripple\_carry\_adder\_tb;      architecture rtl of ripple\_carry\_adder\_tb is    constant c\_WIDTH : integer := 2;    signal r\_ADD\_1 : std\_logic\_vector(c\_WIDTH-1 downto 0) := (others => '0');  signal r\_ADD\_2 : std\_logic\_vector(c\_WIDTH-1 downto 0) := (others => '0');  signal w\_RESULT : std\_logic\_vector(c\_WIDTH downto 0);    component ripple\_carry\_adder is  generic (  g\_WIDTH : natural  );  port (  i\_add\_term1 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  i\_add\_term2 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  --  o\_result : out std\_logic\_vector(g\_WIDTH downto 0)  );  end component ripple\_carry\_adder;    begin    -- Instantiate the Unit Under Test (UUT)  UUT : ripple\_carry\_adder  generic map (  g\_WIDTH => c\_WIDTH  )  port map (  i\_add\_term1 => r\_ADD\_1,  i\_add\_term2 => r\_ADD\_2,  o\_result => w\_RESULT  );    -- Test bench is non-synthesizable  process is  begin  r\_ADD\_1 <= "00";  r\_ADD\_2 <= "01";  wait for 10 ns;  r\_ADD\_1 <= "10";  r\_ADD\_2 <= "01";  wait for 10 ns;  r\_ADD\_1 <= "01";  r\_ADD\_2 <= "11";  wait for 10 ns;  r\_ADD\_1 <= "11";  r\_ADD\_2 <= "11";  wait ;  end process;    end rtl; |

2.3.2 Приклад VHDL опису масштабованого суматора з прискореним перенесенням (Carry Lookahead Adder) з використанням параметризації наведено нижче. В якості параметра використовується g\_WIDTH – кількість розрядів або ширина вхідного слова.

Carry Lookahead Adder обчислює один або кілька бітів перенесення до виконання підсумовування, що зменшує час очікування для обчислення результату. Він призначений для паралельного формування бітів перенесення при додаванні двійкових чисел в суматор.

|  |
| --- |
| library IEEE;  use IEEE.std\_logic\_1164.all;  entity full\_adder is  port (i\_bit1, i\_bit2, i\_carry: in std\_logic; -- Inputs  o\_sum, o\_carry: out std\_logic); -- Outputs  end full\_adder;  architecture Equations of full\_adder is  signal A1, A2, A3 : std\_logic; -- input signals  begin -- Parallel assignment statements  o\_sum <= i\_bit1 xor i\_bit2 xor i\_carry;  A1 <= i\_bit1 and i\_bit2; -- gives g  A2 <= i\_bit1 and i\_carry;  A3 <= i\_bit2 and i\_carry;  o\_carry <= A1 or A2 or A3;  end Equations;  -------------------------------------------------------------------------------  -- File Downloaded from http://www.nandland.com  -------------------------------------------------------------------------------  library ieee;  use ieee.std\_logic\_1164.all;    entity carry\_lookahead\_adder is  generic (  g\_WIDTH : natural -- все неотрицательные числа  );  port (  i\_add1 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  i\_add2 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  --  o\_result : out std\_logic\_vector(g\_WIDTH downto 0)  );  end carry\_lookahead\_adder;    architecture rtl of carry\_lookahead\_adder is    component full\_adder is  port (  i\_bit1 : in std\_logic;  i\_bit2 : in std\_logic;  i\_carry : in std\_logic;  o\_sum : out std\_logic;  o\_carry : out std\_logic);  end component full\_adder;    signal w\_G : std\_logic\_vector(g\_WIDTH-1 downto 0); -- Generate  signal w\_P : std\_logic\_vector(g\_WIDTH-1 downto 0); -- Propagate  signal w\_C : std\_logic\_vector(g\_WIDTH downto 0); -- Carry    signal w\_SUM : std\_logic\_vector(g\_WIDTH-1 downto 0);    begin    -- Create the Full Adders  GEN\_FULL\_ADDERS : for ii in 0 to g\_WIDTH-1 generate  FULL\_ADDER\_INST : full\_adder  port map (  i\_bit1 => i\_add1(ii),  i\_bit2 => i\_add2(ii),  i\_carry => w\_C(ii),  o\_sum => w\_SUM(ii),  o\_carry => open  );  end generate GEN\_FULL\_ADDERS;    -- Create the Generate (G) Terms: Gi=Ai\*Bi  -- Create the Propagate Terms: Pi=Ai+Bi  -- Create the Carry Terms:  GEN\_CLA : for jj in 0 to g\_WIDTH-1 generate  w\_G(jj) <= i\_add1(jj) and i\_add2(jj);  w\_P(jj) <= i\_add1(jj) or i\_add2(jj);  w\_C(jj+1) <= w\_G(jj) or (w\_P(jj) and w\_C(jj));  end generate GEN\_CLA;    w\_C(0) <= '0'; -- no carry input    o\_result <= w\_C(g\_WIDTH) & w\_SUM; -- VHDL Concatenation    end rtl; |

A close up of a device

Description automatically generated

**VHDL Testbench:**

|  |
| --- |
| -------------------------------------------------------------------------------  -- File Downloaded from http://www.nandland.com  -------------------------------------------------------------------------------  library ieee;  use ieee.std\_logic\_1164.all;    entity carry\_lookahead\_adder\_tb is  end carry\_lookahead\_adder\_tb;    architecture behave of carry\_lookahead\_adder\_tb is    constant c\_WIDTH : integer := 3;    signal r\_ADD\_1 : std\_logic\_vector(c\_WIDTH-1 downto 0) := (others => '0');  signal r\_ADD\_2 : std\_logic\_vector(c\_WIDTH-1 downto 0) := (others => '0');  signal w\_RESULT : std\_logic\_vector(c\_WIDTH downto 0);    component carry\_lookahead\_adder is  generic (  g\_WIDTH : natural  );  port (  i\_add1 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  i\_add2 : in std\_logic\_vector(g\_WIDTH-1 downto 0);    o\_result : out std\_logic\_vector(g\_WIDTH downto 0)  );  end component carry\_lookahead\_adder;    begin    -- Instantiate the Unit Under Test (UUT)  UUT : carry\_lookahead\_adder  generic map (  g\_WIDTH => c\_WIDTH  )  port map (  i\_add1 => r\_ADD\_1,  i\_add2 => r\_ADD\_2,  o\_result => w\_RESULT  );    -- Test bench is non-synthesizable  process is  begin  r\_ADD\_1 <= "000";  r\_ADD\_2 <= "001";  wait for 10 ns;    r\_ADD\_1 <= "100";  r\_ADD\_2 <= "010";  wait for 10 ns;    r\_ADD\_1 <= "010";  r\_ADD\_2 <= "110";  wait for 10 ns;    r\_ADD\_1 <= "111";  r\_ADD\_2 <= "111";  wait ;  end process;    end behave; |

2.4 Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання.

2. Здійснити опис функціональності.

3. Виконати моделювання проекту засобами мови VHDL.

4. Оформити звіт.

2.5 Варіанти завдання

1) п. 2.3.1, значення параметра g\_WIDTH : natural := 4.

2) п. 2.3.1, значення параметра g\_WIDTH : natural := 8.

3) п. 2.3.1, значення параметра g\_WIDTH : natural := 16.

4) п. 2.3.2, значення параметра g\_WIDTH : natural := 4.

5) п. 2.3.2, значення параметра g\_WIDTH : natural := 8.

6) п. 2.3.2, значення параметра g\_WIDTH : natural := 16.

2.6 Контрольні запитання

1. Опишіть алгоритм роботи каскадного суматора.

2. Опишіть алгоритм роботи суматора з прискореним перенесенням.

3. Опишіть алгоритм роботи напівсуматора.

4. Опишіть алгоритм роботи мультиплексора та варіанти опису поведінки мультиплексора мовою VHDL.

5. Опишіть алгоритм роботи демультиплексора та варіанти опису поведінки демультиплексора мовою VHDL.

**3 ФОРМУВАННЯ VHDL ОПИСУ ФРАГМЕНТУ ПОСЛІДОВНОСТНОЇ СХЕМИ**

3.1 Мета роботи

Вивчення структури VHDL опису послідовностної схеми – базового базового компонента SoC. Отримання практичних навичок створення поведінкового опису послідовностної схеми та її моделювання засобами мови VHDL.

3.2 Методичні вказівки з організації самостійної роботи студентів

Під час підготовки до лабораторної роботи слід ознайомитися з літературою [1-4], матеріалами конспекту лекцій та інформацією, наведеною в підрозділі 2.4.

3.3 Опис лабораторної установки

3.3.1 Регістр зсуву - це регістр, який використовується для зсуву збереженого біта в одному або обох напрямках. У наведеному нижче прикладі реалізовано регістр зсуву, який можна використовувати для зсуву даних в обох напрямках. Далі він може використовуватися як паралельно-послідовний або послідовно-паралельний перетворювач.

|  |  |
| --- | --- |
| -- Testbench automatically generated online  -- at https://vhdl.lapinoo.net  -- Generation date : 27.9.2020 21:52:10 UTC  library ieee;  use ieee.std\_logic\_1164.all;  entity tb\_shift\_register is  end tb\_shift\_register;  architecture tb of tb\_shift\_register is  component shift\_register  generic (  n : natural  );  port (clk : in std\_logic;  reset : in std\_logic;  ctrl : in std\_logic\_vector (1 downto 0);  data : in std\_logic\_vector (n-1 downto 0);  q\_reg : out std\_logic\_vector (n-1 downto 0));  end component;    constant n : integer := 8;  signal clk : std\_logic;  signal reset : std\_logic;  signal ctrl : std\_logic\_vector (1 downto 0);  signal data : std\_logic\_vector (n-1 downto 0);  signal q\_reg : std\_logic\_vector (n-1 downto 0);  constant TbPeriod : time := 100 ns; -- EDIT Put right period here  signal TbClock : std\_logic := '0';  signal TbSimEnded : std\_logic := '0';  begin  dut : shift\_register  generic map (  n => n  )  port map (clk => clk,  reset => reset,  ctrl => ctrl,  data => data,  q\_reg => q\_reg);  -- Clock generation  TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';  -- EDIT: Check that clk is really your main clock signal  clk <= TbClock;  stimuli : process  begin  -- EDIT Adapt initialization as needed  ctrl <= (others => '0');  data <= (others => '0');  -- Reset generation  -- EDIT: Check that reset is really your reset signal  reset <= '1';  wait for 10 ns;  reset <= '0';  wait for 10 ns;  -- EDIT Add stimuli here  ctrl <="00";  data <= "11010101";  wait for 10 \* TbPeriod;    ctrl <="01";  data <= "10010101";  wait for 10 \* TbPeriod;    ctrl <="10";  data <= "10001111";  wait for 10 \* TbPeriod;    ctrl <="11";  data <= "10101111";  wait for 10 \* TbPeriod;  -- Stop the clock and hence terminate the simulation  TbSimEnded <= '1';  wait;  end process;  end tb; | -- shift\_register.vhd  -- created by : Meher Krishna Patel  -- date : 22-Dec-16  -- Functionality:  -- load data and shift it data to left and right  -- parallel to serial conversion (i.e. first load, then shift)  -- serial to parallel conversion (i.e. first shift, then read)  -- inputs:  -- ctrl : to load-data and shift operations (right and left shift)  -- data : it is the data to be shifted  -- q\_reg : store the outputs    library ieee;  use ieee.std\_logic\_1164.all;  entity shift\_register is  generic (N :integer :=8);  port(  clk, reset : in std\_logic;  ctrl : in std\_logic\_vector(1 downto 0);  data : in std\_logic\_vector(N-1 downto 0);  q\_reg : out std\_logic\_vector(N-1 downto 0)  );  end shift\_register;  architecture arch of shift\_register is  signal s\_reg, s\_next : std\_logic\_vector(N-1 downto 0);  begin  process(clk, reset)  begin  if(reset='1') then  s\_reg <= (others=>'0'); -- clear the content  elsif (clk'event and clk='1') then  s\_reg <= s\_next; -- otherwise save the next state  end if;  end process;  process (ctrl, s\_reg)  begin  case ctrl is  when "00" =>  s\_next <= s\_reg; -- no operation (to read data for serial to parallel)  when "01" =>  s\_next <= data(N-1) & s\_reg(N-1 downto 1); -- right shift  when "10" =>  s\_next <= s\_reg(N-2 downto 0) & data(0); -- left shift  when others =>  s\_next <= data; -- load data (for parallel to serial)  end case;  end process;    q\_reg <= s\_reg;  end arch; |

3.3.2 Регістр зсуву з лінійним зворотним зв’язком (LFSR) можна використовувати як "генератор псевдовипадкових чисел", які генеруються на основі початкових значень для LFSR. Поліном зворотного зв'язку реалізується відповідно до значення N (табл.\_\_). Для N=3 поліном зворотного зв'язку реалізований як x^3 + x^2 + 1. Щоб змінити поліном зворотного зв'язку, необхідно використовувати нову кількість бітів (N) та змінити значення feedback\_value відповідно до нового значення «N».

A screenshot of a cell phone

Description automatically generated

|  |  |
| --- | --- |
| -- Testbench automatically generated online  -- at https://vhdl.lapinoo.net  -- Generation date : 21.9.2020 21:50:40 UTC  library ieee;  use ieee.std\_logic\_1164.all;  entity tb\_rand\_num\_generator is  end tb\_rand\_num\_generator;  architecture tb of tb\_rand\_num\_generator is  component rand\_num\_generator  port (clk : in std\_logic;  reset : in std\_logic;  q : out std\_logic\_vector (3 downto 0));  end component;  signal clk : std\_logic;  signal reset : std\_logic;  signal q : std\_logic\_vector (3 downto 0);  constant TbPeriod : time := 1000 ns; -- EDIT Put right period here  signal TbClock : std\_logic := '0';  signal TbSimEnded : std\_logic := '0';  begin  dut : rand\_num\_generator  port map (clk => clk,  reset => reset,  q => q);  -- Clock generation  TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';  -- EDIT: Check that clk is really your main clock signal  clk <= TbClock;  stimuli : process  begin  -- EDIT Adapt initialization as needed  -- Reset generation  -- EDIT: Check that reset is really your reset signal  reset <= '1';  wait for 100 ns;  reset <= '0';  wait for 100 ns;  -- EDIT Add stimuli here  wait for 100 \* TbPeriod;  -- Stop the clock and hence terminate the simulation  TbSimEnded <= '1';  wait;  end process;  end tb;  -- Configuration block below is required by some simulators. Usually no need to edit.  configuration cfg\_tb\_rand\_num\_generator of tb\_rand\_num\_generator is  for tb  end for;  end cfg\_tb\_rand\_num\_generator; | -- rand\_num\_generator.vhd  -- created by : Meher Krishna Patel  -- date : 22-Dec-16  -- Feedback polynomial : x^3 + x^2 + 1  -- maximum length : 2^3 - 1 = 7  -- if generic value is changed,  -- then choose the correct Feedback polynomial i.e. change 'feedback\_value' pattern    library ieee;  use ieee.std\_logic\_1164.all;  entity rand\_num\_generator is  generic (N :integer := 3);  port(  clk, reset : in std\_logic;  q : out std\_logic\_vector(N downto 0) -- output of LFSR i.e. random number  );  end rand\_num\_generator;  architecture arch of rand\_num\_generator is  signal r\_reg, r\_next : std\_logic\_vector(N downto 0);  signal feedback\_value : std\_logic; -- based on feedback polynomial  begin  process(clk, reset)  begin  if(reset='1') then  -- set initial value to '1'.  r\_reg(0) <= '1'; -- 0th bit = 1  r\_reg(N downto 1) <= (others=>'0'); -- other bits are 0  elsif (clk'event and clk='1') then  r\_reg <= r\_next; -- otherwise save the next state  end if;  end process;    -- N = 3  -- Feedback polynomial : x^3 + x^2 + 1  -- total sequences (maximum) : 2^3 - 1 = 7  feedback\_value <= r\_reg(3) xor r\_reg(2) xor r\_reg(0);    -- N = 4  -- feedback\_value <= r\_reg(4) xor r\_reg(3) xor r\_reg(0);  -- N = 5, maximum length = 28 (not 31)  -- feedback\_value <= r\_reg(5) xor r\_reg(3) xor r\_reg(0);    -- N = 9  -- feedback\_value <= r\_reg(9) xor r\_reg(5) xor r\_reg(0);    r\_next <= feedback\_value & r\_reg(N downto 1);  q <= r\_reg;  end arch; |

3.4 Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання.

2. Здійснити опис функціональності.

3. Виконати моделювання проекту засобами мови VHDL.

4. Оформити звіт.

3.5 Варіанти завдання

1) п. 3.3.1, значення параметра N :integer :=4.

2) п. 3.3.1, значення параметра N :integer :=8.

3) п. 3.3.1, значення параметра N :integer :=16.

4) п. 3.3.2, значення параметра N :integer :=3.

2) п. 3.3.2, значення параметра N :integer :=4.

6) п. 3.3.2, значення параметра N :integer :=5.

7) п. 3.3.2, значення параметра N :integer :=9.

3.6 Контрольні запитання

1. Опишіть алгоритм роботи регістра зсуву.

2. Опишіть алгоритм роботи регістра зсуву з лінійним зворотним зв’язком (LFSR).

3. Опишіть поведінкові моделі системного рівня проектування.

4. Опишіть структурні моделі системного рівня проектування.

5. Опишіть поведінкові моделі процесорного рівня проектування.

6. Опишіть структурні моделі процесорного рівня проектування.

7. Опишіть процедуру синтезу на процесорному рівні.

8. Опишіть завдання, які вирішуються при перетворенні поведінкової моделі в оптимізовану системну платформу.

**4 ФОРМУВАННЯ VHDL ОПИСУ МОДУЛЯ ПАМ’ЯТІ**

4.1 Мета роботи

Вивчення структури VHDL опису модуля пам’яті – базового компонента SoC. Отримання практичних навичок створення поведінкового опису модуля пам’яті та її моделювання засобами мови VHDL.

4.2 Методичні вказівки з організації самостійної роботи студентів

Під час підготовки до лабораторної роботи слід ознайомитися з літературою [1-4], матеріалами конспекту лекцій та інформацією, наведеною в підрозділі 2.4.

4.3 Опис лабораторної установки

4.3.1 ПЗУ – це пристрої, які використовуються для постійного зберігання інформації. У наступному прикладі ПЗУ реалізовано на FPGA для зберігання шаблону відображення для семисегментного пристрою.

|  |  |
| --- | --- |
| -- Testbench automatically generated online  -- at https://vhdl.lapinoo.net  -- Generation date : 1.10.2020 20:29:40 UTC  library ieee;  use ieee.std\_logic\_1164.all;  entity tb\_ROM\_sevenSegment is  end tb\_ROM\_sevenSegment;  architecture tb of tb\_ROM\_sevenSegment is  component ROM\_sevenSegment  generic(  addr\_bits : integer := 4;  data\_width : natural := 7  );  port (addr : in std\_logic\_vector (addr\_bits-1 downto 0);  data : out std\_logic\_vector (data\_width-1 downto 0));  end component;  constant addr\_bits : integer := 4;  constant data\_width : integer := 7;    signal addr : std\_logic\_vector (addr\_bits-1 downto 0);  signal data : std\_logic\_vector (data\_width-1 downto 0);  constant TbPeriod : time := 100 ns; -- EDIT Put right period here  signal TbClock : std\_logic := '0';  signal TbSimEnded : std\_logic := '0';  begin  dut : ROM\_sevenSegment  generic map (  addr\_bits => addr\_bits, -- required bits to store 16 elements  data\_width => data\_width -- each element has 7-bits  )  port map (addr => addr,  data => data);  -- Clock generation  TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';  stimuli : process  begin  -- EDIT Adapt initialization as needed  addr <= (others => '0');  -- EDIT Add stimuli here  addr <= "1111";  wait for 10 \* TbPeriod;    addr <= "1011";  wait for 10 \* TbPeriod;  -- Stop the clock and hence terminate the simulation  TbSimEnded <= '1';  wait;  end process;  end tb;  -- Configuration block below is required by some simulators. Usually no need to edit.  configuration cfg\_tb\_ROM\_sevenSegment of tb\_ROM\_sevenSegment is  for tb  end for;  end cfg\_tb\_ROM\_sevenSegment; | --ROM\_sevenSegment.vhd  -- created by : Meher Krishna Patel  -- date : 25-Dec-16  -- Functionality:  -- seven-segment display format for Hexadecimal values (i.e. 0-F) are stored in ROM  -- ports:  -- addr : input port for getting address  -- data : ouput data at location 'addr'  -- addr\_width : total number of elements to store (put exact number)  -- addr\_bits : bits requires to store elements specified by addr\_width  -- data\_width : number of bits in each elements    library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity ROM\_sevenSegment is  generic(  addr\_width : integer := 16; -- store 16 elements  addr\_bits : integer := 4; -- required bits to store 16 elements  data\_width : integer := 7 -- each element has 7-bits  );  port(  addr : in std\_logic\_vector(addr\_bits-1 downto 0);  data : out std\_logic\_vector(data\_width-1 downto 0)  );  end ROM\_sevenSegment;  architecture arch of ROM\_sevenSegment is  type rom\_type is array (0 to addr\_width-1) of std\_logic\_vector(data\_width-1 downto 0); -- ROM type    signal sevenSegment\_ROM : rom\_type := (  "1000000", -- 0, active low i.e. 0:display & 1:no display  "1111001", -- 1  "0100100", -- 2  "0110000", -- 3  "0011001", -- 4  "0010010", -- 5  "0000010", -- 6  "1111000", -- 7  "0000000", -- 8  "0010000", -- 9  "0001000", -- a  "0000011", -- b  "1000110", -- c  "0100001", -- d  "0000110", -- e  "0001110" -- f  );  begin  data <= sevenSegment\_ROM(to\_integer(unsigned(addr)));  end arch; |

4.3.2 Однопортова оперативна пам’ять має один вхідний порт (рядок адреси), який використовується як для зберігання, так і для отримання даних.

|  |  |
| --- | --- |
| -- Testbench automatically generated online  -- at https://vhdl.lapinoo.net  -- Generation date : 4.10.2020 21:22:48 UTC  library ieee;  use ieee.std\_logic\_1164.all;  entity tb\_single\_port\_RAM is  end tb\_single\_port\_RAM;  architecture tb of tb\_single\_port\_RAM is  component single\_port\_RAM  generic(  addr\_width : integer := 2;  data\_width : integer := 3  );  port (clk : in std\_logic;  we : in std\_logic;  addr : in std\_logic\_vector (addr\_width-1 downto 0);  din : in std\_logic\_vector (data\_width-1 downto 0);  dout : out std\_logic\_vector (data\_width-1 downto 0));  end component;  constant addr\_width : integer := 2;  constant data\_width : integer := 3;  signal clk : std\_logic;  signal we : std\_logic;  signal addr : std\_logic\_vector (addr\_width-1 downto 0);  signal din : std\_logic\_vector (data\_width-1 downto 0);  signal dout : std\_logic\_vector (data\_width-1 downto 0);  constant TbPeriod : time := 1000 ns; -- EDIT Put right period here  signal TbClock : std\_logic := '0';  signal TbSimEnded : std\_logic := '0';  begin  dut : single\_port\_RAM  generic map (  addr\_width => addr\_width,  data\_width => data\_width  )  port map (clk => clk,  we => we,  addr => addr,  din => din,  dout => dout);  -- Clock generation  TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';  -- EDIT: Check that clk is really your main clock signal  clk <= TbClock;  stimuli : process  begin  -- EDIT Adapt initialization as needed  we <= '0';  addr <= (others => '0');  din <= (others => '0');  -- EDIT Add stimuli here  we <= '1';  addr <= "01";  din <= "111";  wait for 10 \* TbPeriod;    we <= '0';  addr <= "01";    wait for 10 \* TbPeriod;  -- Stop the clock and hence terminate the simulation  TbSimEnded <= '1';  wait;  end process;  end tb; | -- single\_port\_RAM.vhd  -- created by : Meher Krishna Patel  -- date : 26-Dec-16  -- Functionality:  -- store and retrieve data from single port RAM  -- ports:  -- we : write enable  -- addr : input port for getting address  -- din : input data to be stored in RAM  -- data : output data read from RAM  -- addr\_width : total number of elements to store (put exact number)  -- addr\_bits : bits requires to store elements specified by addr\_width  -- data\_width : number of bits in each elements  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity single\_port\_RAM is  generic (  addr\_width : integer := 2;  data\_width : integer := 3  );    port(  clk: in std\_logic;  we : in std\_logic;  addr : in std\_logic\_vector(addr\_width-1 downto 0);  din : in std\_logic\_vector(data\_width-1 downto 0);  dout : out std\_logic\_vector(data\_width-1 downto 0)  );  end single\_port\_RAM;  architecture arch of single\_port\_RAM is  type ram\_type is array (2\*\*addr\_width-1 downto 0) of std\_logic\_vector (data\_width-1 downto 0);  signal ram\_single\_port : ram\_type;  begin  process(clk)  begin  if (clk'event and clk='1') then  if (we='1') then -- write data to address 'addr'  --convert 'addr' type to integer from std\_logic\_vector  ram\_single\_port(to\_integer(unsigned(addr))) <= din;  end if;  end if;  end process;  -- read data from address 'addr'  -- convert 'addr' type to integer from std\_logic\_vector  dout<=ram\_single\_port(to\_integer(unsigned(addr)));  end arch; |

4.3.3 Двопортова оперативна пам'ять використовує два вхідні порти для операцій читання та запису.

|  |  |
| --- | --- |
| -- Testbench automatically generated online  -- at https://vhdl.lapinoo.net  -- Generation date : 4.10.2020 21:41:20 UTC  library ieee;  use ieee.std\_logic\_1164.all;  entity tb\_dual\_port\_RAM is  end tb\_dual\_port\_RAM;  architecture tb of tb\_dual\_port\_RAM is  component dual\_port\_RAM  generic(  addr\_width : integer := 2;  data\_width : integer := 3  );  port (clk : in std\_logic;  we : in std\_logic;  addr\_wr : in std\_logic\_vector (addr\_width-1 downto 0);  addr\_rd : in std\_logic\_vector (addr\_width-1 downto 0);  din : in std\_logic\_vector (data\_width-1 downto 0);  dout : out std\_logic\_vector (data\_width-1 downto 0));  end component;  constant addr\_width : integer := 2;  constant data\_width : integer := 3;  signal clk : std\_logic;  signal we : std\_logic;  signal addr\_wr : std\_logic\_vector (addr\_width-1 downto 0);  signal addr\_rd : std\_logic\_vector (addr\_width-1 downto 0);  signal din : std\_logic\_vector (data\_width-1 downto 0);  signal dout : std\_logic\_vector (data\_width-1 downto 0);  constant TbPeriod : time := 1000 ns; -- EDIT Put right period here  signal TbClock : std\_logic := '0';  signal TbSimEnded : std\_logic := '0';  begin  dut : dual\_port\_RAM  generic map (  addr\_width => addr\_width,  data\_width => data\_width  )  port map (clk => clk,  we => we,  addr\_wr => addr\_wr,  addr\_rd => addr\_rd,  din => din,  dout => dout);  -- Clock generation  TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';  -- EDIT: Check that clk is really your main clock signal  clk <= TbClock;  stimuli : process  begin  -- EDIT Adapt initialization as needed  we <= '0';  addr\_wr <= (others => '0');  addr\_rd <= (others => '0');  din <= (others => '0');  -- EDIT Add stimuli here  we <= '1';  addr\_wr <= "01";  din <= "011";  wait for 10 \* TbPeriod;    we <= '1';  addr\_wr <= "10";  din <= "111";  wait for 10 \* TbPeriod;    we <= '0';  addr\_rd <= "01";  wait for 10 \* TbPeriod;    we <= '0';  addr\_rd <= "10";  wait for 10 \* TbPeriod;  -- Stop the clock and hence terminate the simulation  TbSimEnded <= '1';  wait;  end process;  end tb; | -- dual\_port\_RAM.vhd  -- created by : Meher Krishna Patel  -- date : 26-Dec-16  -- Functionality:  -- store and retrieve data from single port RAM  -- ports:  -- we : write enable  -- addr\_wr : address for writing data  -- addr\_rd : address for reading  -- din : input data to be stored in RAM  -- data : output data read from RAM  -- addr\_width : total number of elements to store (put exact number)  -- addr\_bits : bits requires to store elements specified by addr\_width  -- data\_width : number of bits in each elements  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity dual\_port\_RAM is  generic (  addr\_width : integer := 2;  data\_width : integer := 3  );    port(  clk: in std\_logic;  we : in std\_logic;  addr\_wr, addr\_rd : in std\_logic\_vector(addr\_width-1 downto 0);  din : in std\_logic\_vector(data\_width-1 downto 0);  dout : out std\_logic\_vector(data\_width-1 downto 0)  );  end dual\_port\_RAM;  architecture arch of dual\_port\_RAM is  type ram\_type is array (2\*\*addr\_width-1 downto 0) of std\_logic\_vector (data\_width-1 downto 0);  signal ram\_dual\_port : ram\_type;  begin  process(clk)  begin  if (clk'event and clk='1') then  if (we='1') then -- write data to address 'addr\_wr'  -- convert 'addr\_wr' type to integer from std\_logic\_vector  ram\_dual\_port(to\_integer(unsigned(addr\_wr))) <= din;  end if;  end if;  end process;  -- get address for reading data from 'addr\_rd'  -- convert 'addr\_rd' type to integer from std\_logic\_vector  dout<=ram\_dual\_port(to\_integer(unsigned(addr\_rd)));  end arch; |

4.4 Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання.

2. Здійснити опис функціональності.

3. Виконати моделювання проекту засобами мови VHDL.

4. Оформити звіт.

4.5 Варіанти завдання

1) п. 4.3.1.

2) п. 4.3.2.

3) п. 4.3.3.

4.6 Контрольні запитання

1. Опишіть алгоритм роботи модуля пам’яті.
2. Поясніть архітектуру і опис регістрових файлів мовою VHDL.
3. Поясніть опис семисегментного дисплею мовою VHDL.
4. Поясніть опис ROM (ПЗУ) мовою VHDL.
5. Поясніть опис однопортової RAM мовою VHDL.
6. Поясніть опис двопортової RAM мовою VHDL.
7. Поясніть опис модуля FIFO мовою VHDL.