

«Дослідження та проектування комп'ютерних систем та мереж»

(частина 1)

ЗМІСТ

ЛР № 1. САПР Xilinx WebPack.....	3
ЛР № 2. Імплементування VHDL моделей операційного та керуючого пристроїв (автоматів).....	18
ЛР № 3А. Імплементування VHDL моделей двохнаправленої шини та LVDS шини.....	30
ЛР № 3Б. Синтез та дослідження DLL/DCM тактування проектів на ПЛІС.....	36
ЛР № 4. Імплементування та дослідження VHDL моделі машини з архітектурою MIPS.....	42
ЛР № 5. Імплементування та дослідження VHDL моделі софтверного контролера XILINX PicoBlaze.....	49
ЛР № 6. Імплементування та дослідження VHDL моделі софтверного контролера XESS Gnome.....	57
ЛР № 7. Імплементування та дослідження софтверного контролера з шиною wishbone	62
ЛР № 8А. Дослідження SystemC моделі автомата.....	90
ЛР № 8Б. Синтез та дослідження SystemC моделі RISC процесора.....	106

Узагальнена мета виконання циклу лабораторних робіт з дисципліни «Дослідження та проектування комп'ютерних систем та мереж» (частина 1) полягає в опануванні технікою синтезу (із залученням сучасних САПР) типових апаратних комп'ютерних засобів з наступним імплементуванням створених VHDL або SystemC моделей до ПЛІС фірми Xilinx.

До кожної лабораторної роботи надається базова VHDL модель вузла, що досліджується, та результати її синтезу, імплементування і симуляційної верифікації. Після відпрацювання в лабораторному експерименті базової моделі, її змінюють за певними вимогами, а лабораторні дослідження виконують ще раз з врахуванням змін. Результати досліджень заносять до звіту, який потрібно захистити.

ЛР № 1. САПР Xilinx WebPack**Мета лабораторної роботи:**

Опанувати методами роботи в САПР Xilinx WebPack. Дослідити властивості, поведінку та варіанти використання базового примітивного елементу ПЛІС, що отримав назву функційної таблиці (в оригіналі – Look-Up Table (LUT)).

Завдання:

В САПР WebPack/ModelSim імплементувати в ПЛІС Virtex-II власні модифікації наданого базового проекту «Функційна таблиця». Запропоновані імплементування верифікувати. Скласти звіт з виконання лабораторних досліджень та захистити його.

Зауваження

Проведення лабораторної роботи передбачає ознайомлення з системою виконання проектних робіт на ПЛІС фірми Xilinx, що має назву Xilinx ISE WebPack (розповсюджується фірмою Xilinx безкоштовно: її можна отримати або на сайті xilinx.com, або на кафедрі; обсяг системи складає 1,5 ГБ для версії 9.2i).

Методичний матеріал до виконання цієї частини містить вступний розділ. Прототип VHDL моделі, що досліджують відповідно до завдання роботи, розташований в розділі «Базовий проект «Функційна таблиця».

Вступ. Інструмент Xilinx WebPack

Комбінаційну логіку в ПЛІС реалізують таблицями. Таблиці наповнюються під час конфігурування матриці. Дамо приклад табличної реалізації схеми інвертуючої схеми І на два входи за допомогою табличного примітиву LUT2 (2-Bit Look-Up-Table with General Output, тобто, функційна таблиця на два бітових входи і на один бітовий вихід; вона містить пам'ять на $2^2 = 4$ біти). Цей примітив присутній в ПЛІС Xilinx Virtex-2. Існують також табличні примітиви на чотири входи і один вихід з пам'яттю на 16 біт.

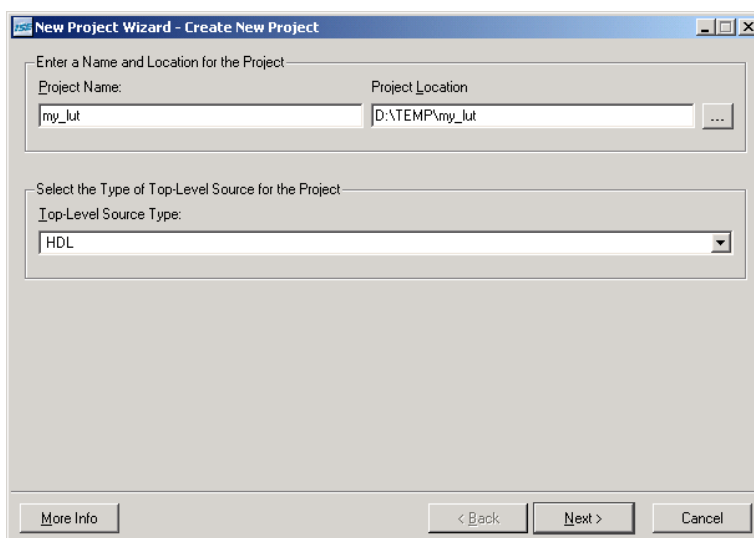


Рис. 1.1 – Створення проекту my_lut з розташуванням в D:\TEMP\my_lut

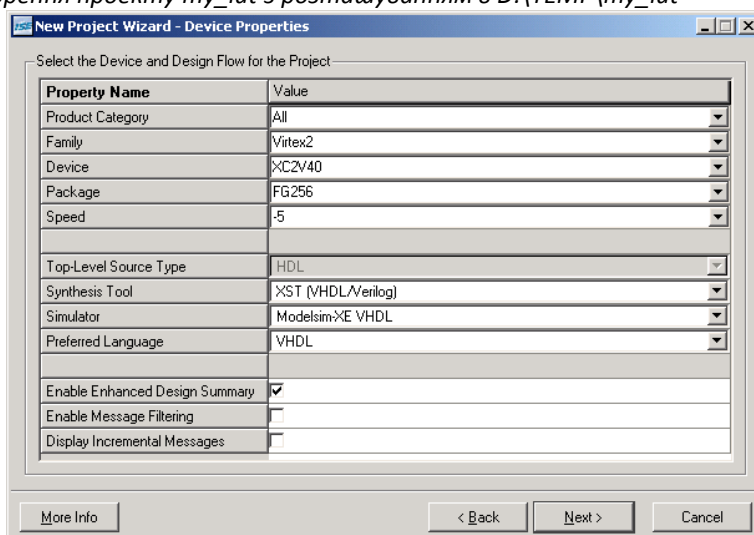


Рис. 1.2 – Вибір цільової ПЛІС та зовнішнього щодо WebPack симулятора Modelsim-XE VHDL

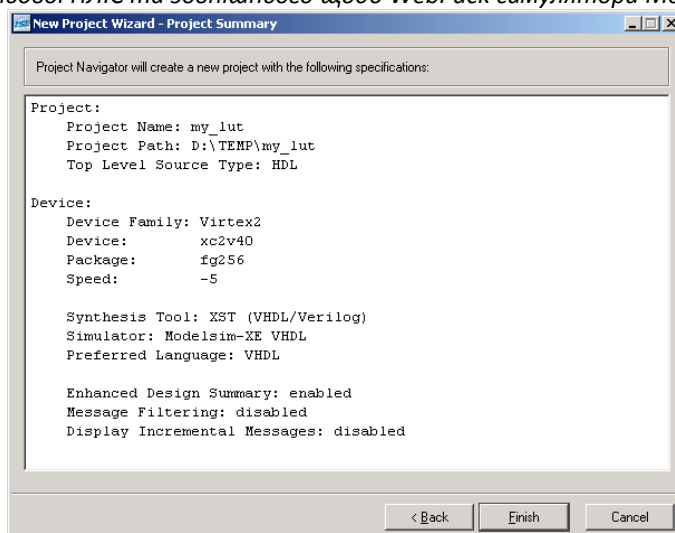


Рис. 1.3 – Автоматично згенероване резюме проекту my_lut

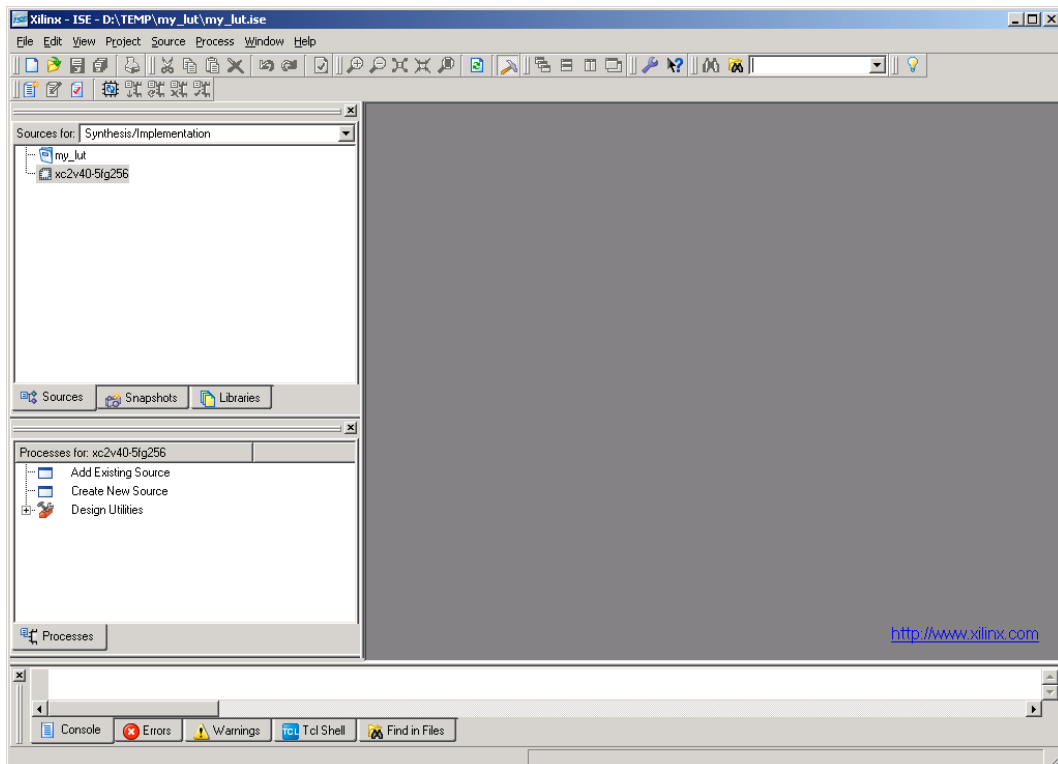


Рис. 1.4 – Вікно навігатора проектів WebPack з новим проектом *my_lut*, але поки що без VHDL моделі

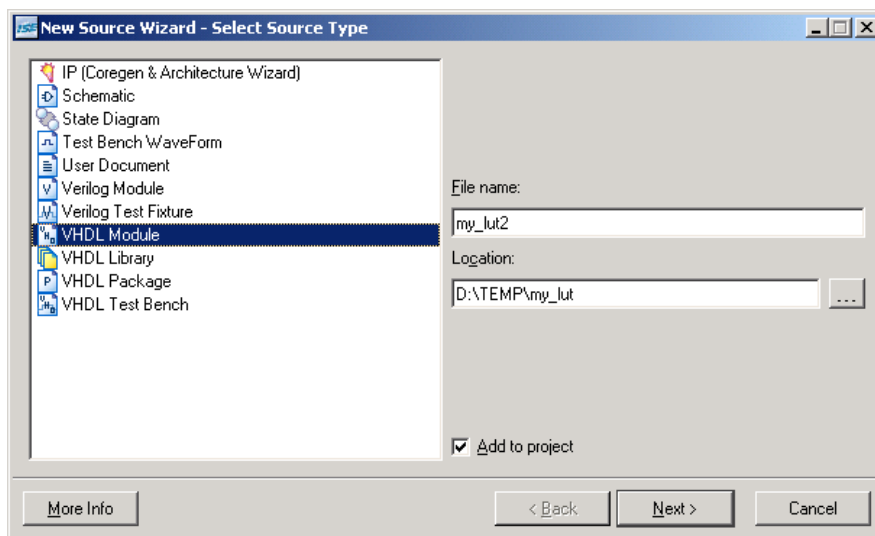


Рис. 1.5 – Початок створення VHDL моделі проекту *my_lut*

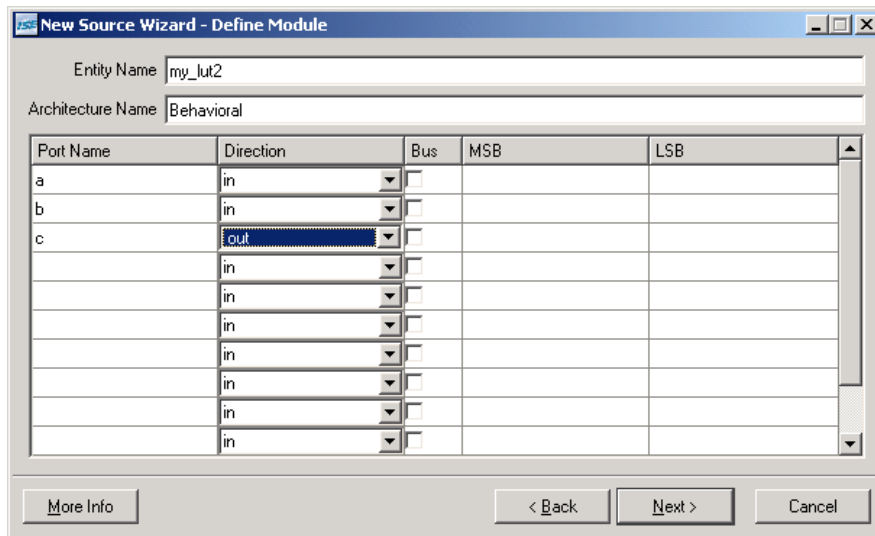


Рис. 1.6 – Визначення входів/виходів VHDL моделі проекту

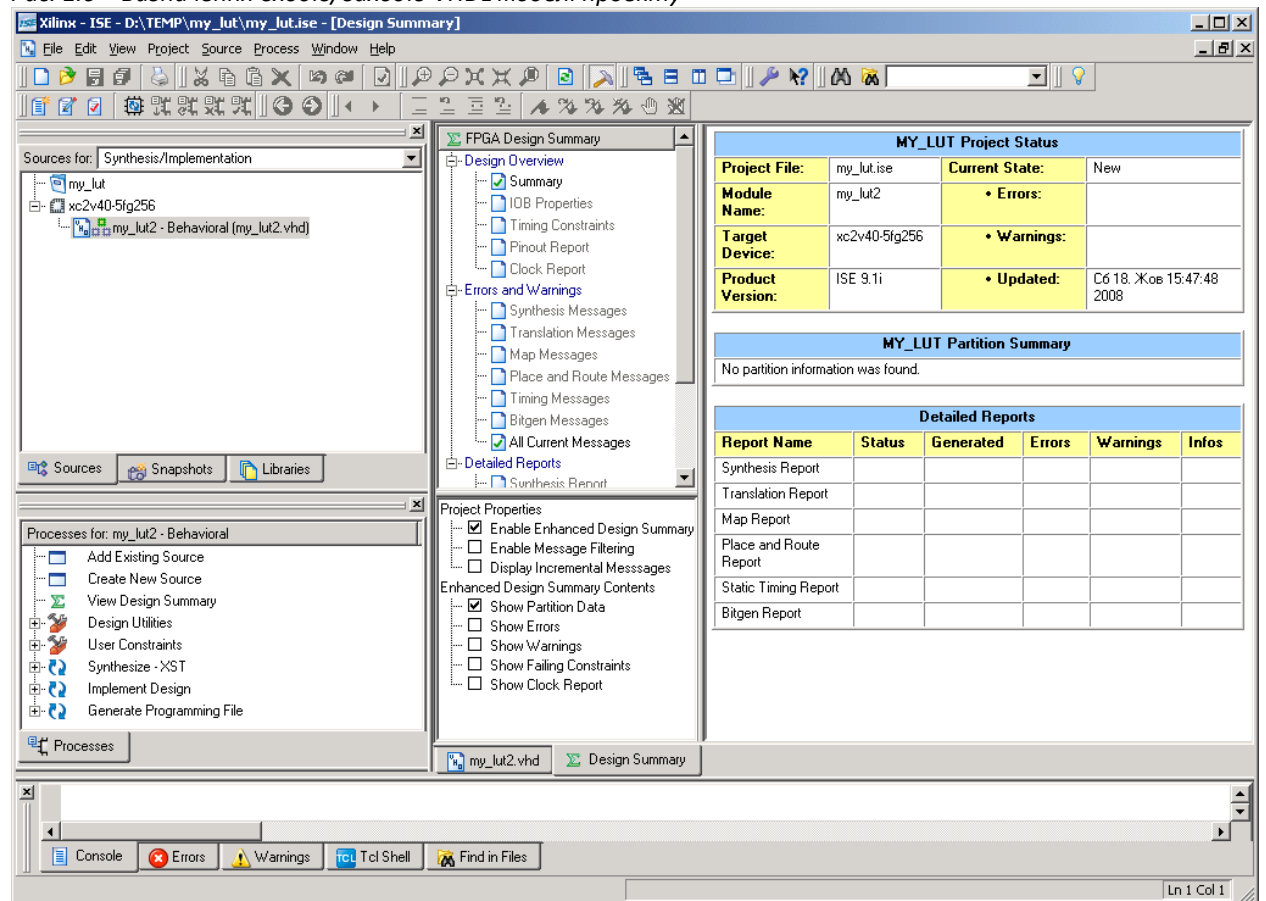


Рис. 1.7 – Вікно навігатора проектів для проекту my_lut; праворуч бачимо вікно резюме

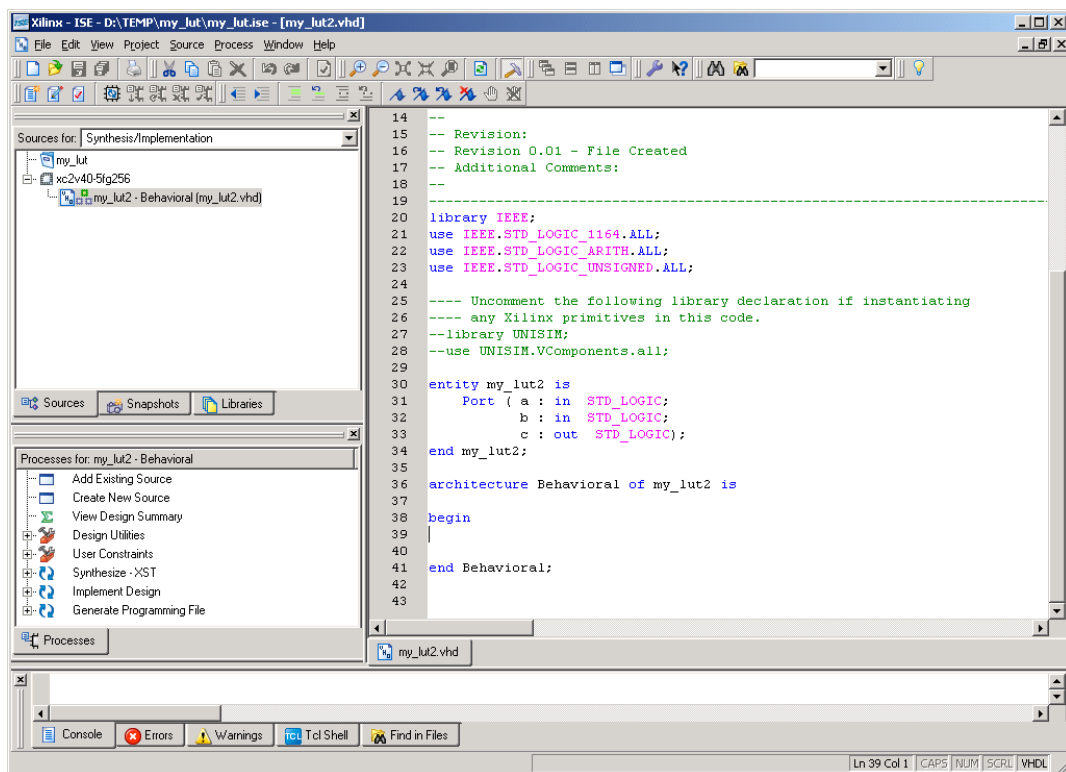


Рис. 1.8 – Навігатора для проекту my_lut; праворуч – автоматично згенерований (проте порожній) шаблон VHDL моделі

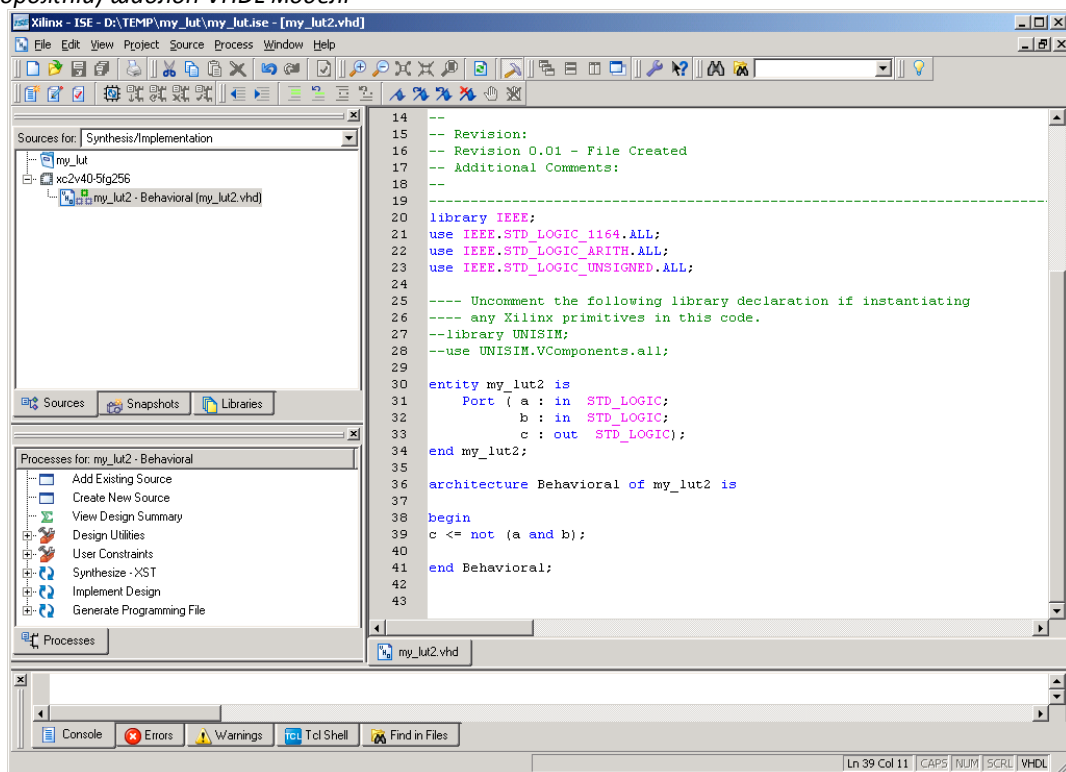


Рис. 1.9 – Вікно навігатора для проекту my_lut; праворуч бачимо заповнений вручну розділ архітектури, з формальним визначенням поведінки (behave) інвертуючого вентиля І на два входи

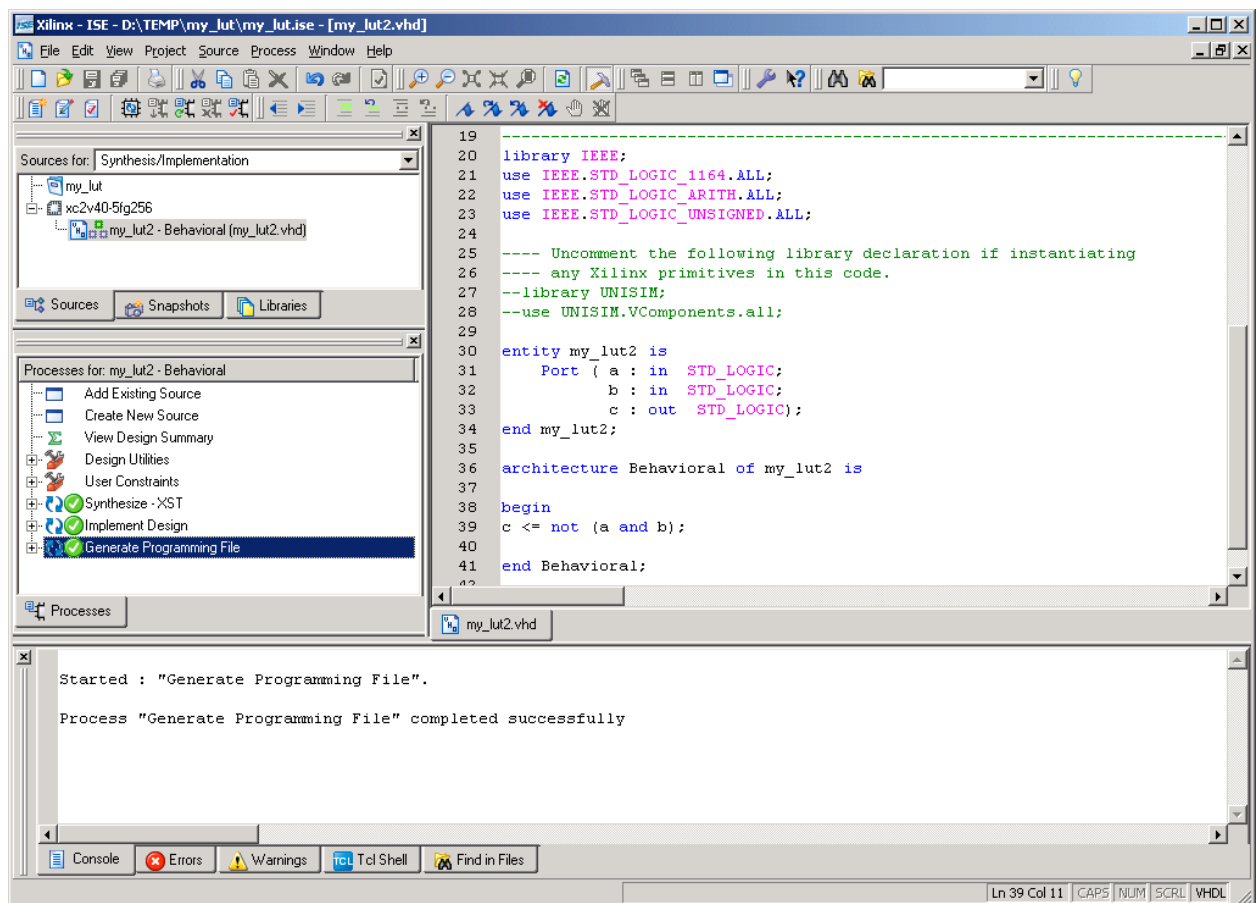


Рис. 1.10 – Вікно навігатора для проекту my_lut із позитивними результатами (зелені гави) синтезу моделі, її імплементації в ПЛІС та програмування (генерації бінарного файла конфігурування ПЛІС)

На етапі синтезу створено звіт, витяг з якого подаємо.

Final Results (синтезу)

RTL Top Level Output File Name : my_lut2.ngf
 Top Level Output File Name : my_lut2
 Output Format : NGC
 Optimization Goal : Speed
 Keep Hierarchy : NO

Design Statistics

IOs : 3
 Cell Usage :
 # BELS : 1
 # LUT2 : 1
 # IO Buffers : 3
 # IBUF : 2
 # OBUF : 1

Device utilization summary:

Selected Device : 2v40fg256-5

Number of Slices:	1	out of	256	0%
Number of 4 input LUTs:	1	out of	512	0%
Number of IOs:	3			

Number of bonded IOBs:

3 out of 88 3%

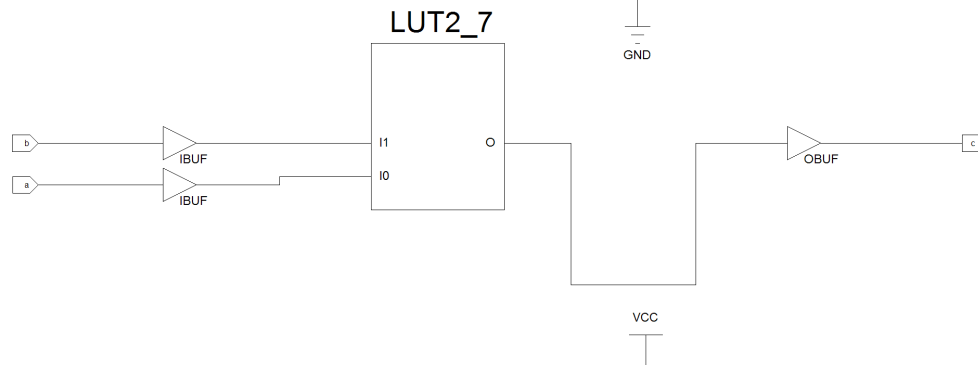


Рис. 1.11 – Технологічна схема проекту *my_lut*; в центрі розташована функційна таблиця на два входи (LUT2_7); входи знаходяться ліворуч, їх надсилають через входні підсилювачі IBUF; вихід видають назовні через підсилювач OBUF. Наявні на структурі рівні логічного нуля (GND) та одиниці (VCC) не використовують

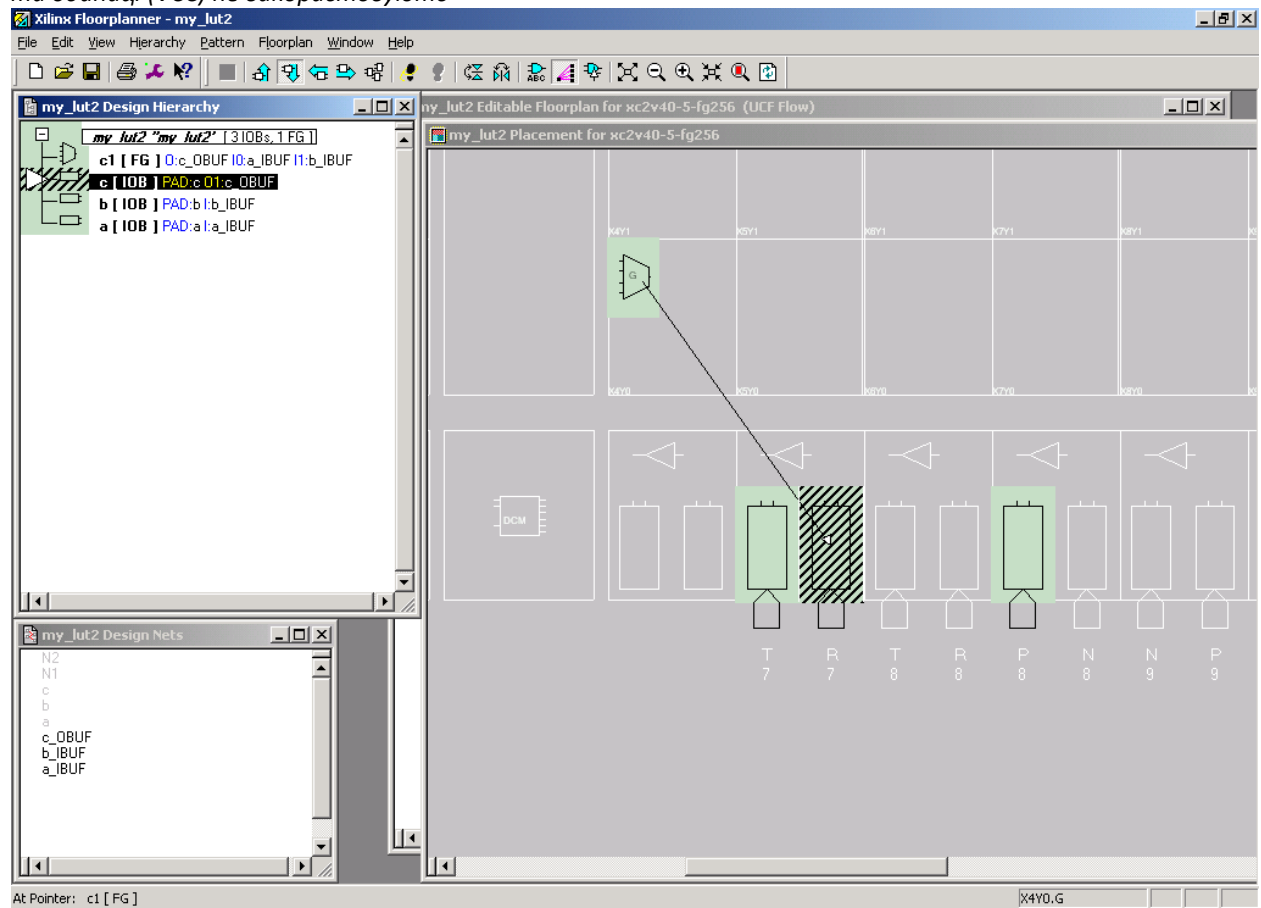


Рис. 1.12 – Вікно утиліти Floorplanner зі складу Xilinx ISE WebPack, що подає розташування імплементації проекту на поверхні кристала ПЛІС; видно які контакти ПЛІС та під які сигнали задіяв автомат розведення; проте призначення може задавати розробник додатковим файлом проекту з розширенням *.ucf, де ucf – User Constrains FILE, який є файлом обмежень (вимог) користувача; наповнення файлу *.ucf виконують спеціальною мовою обмежень

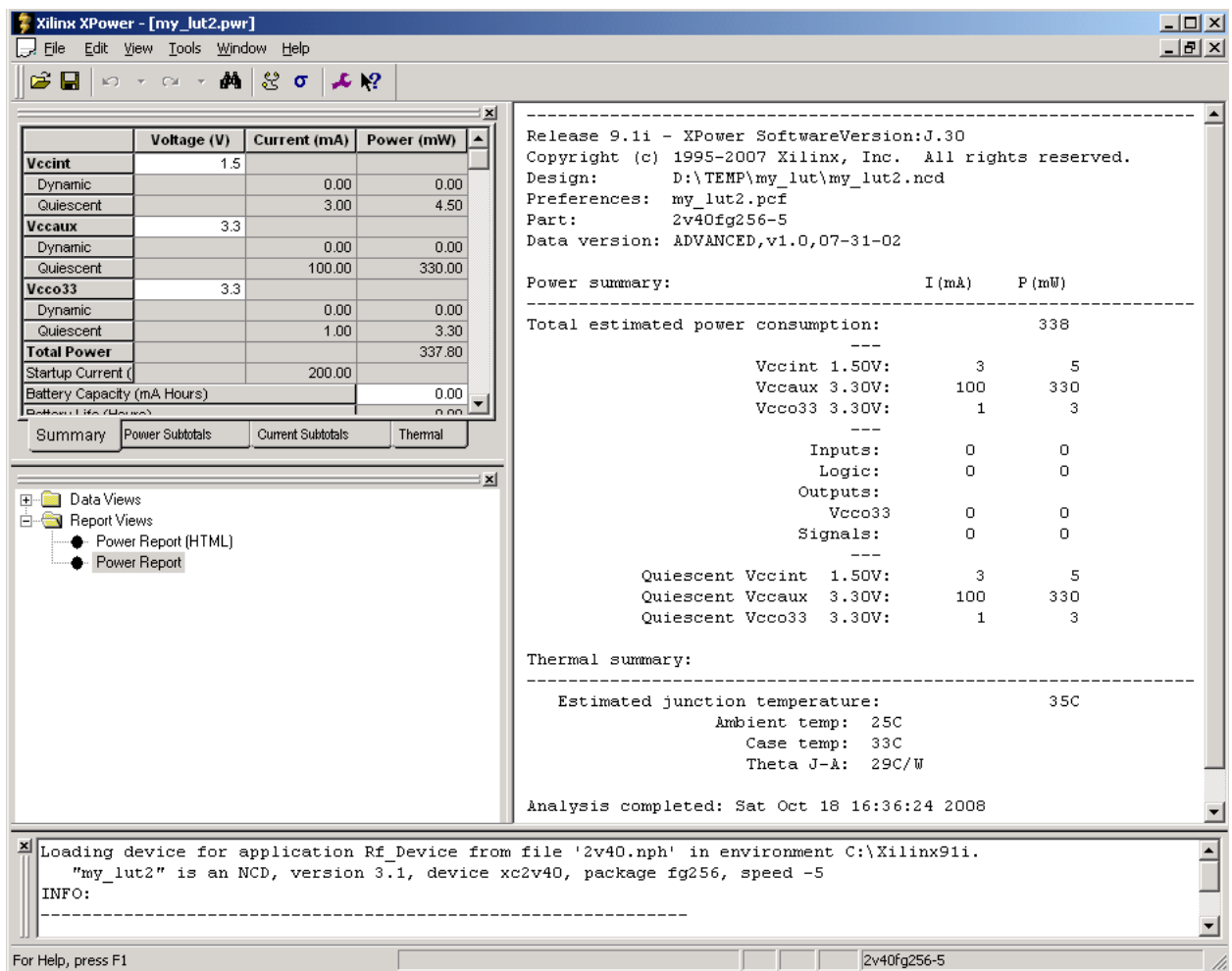


Рис. 1.13 – Вікно утиліти Xilinx Xpower про енергоспоживання проекту my_lut2

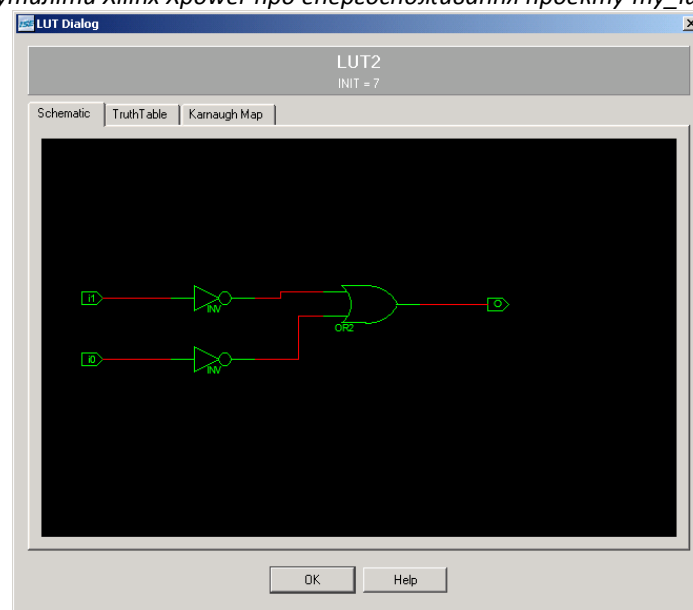


Рис. 1.14 – Еквівалентне подання функційною схемою наповнення функційної таблиці проекту my_lut2

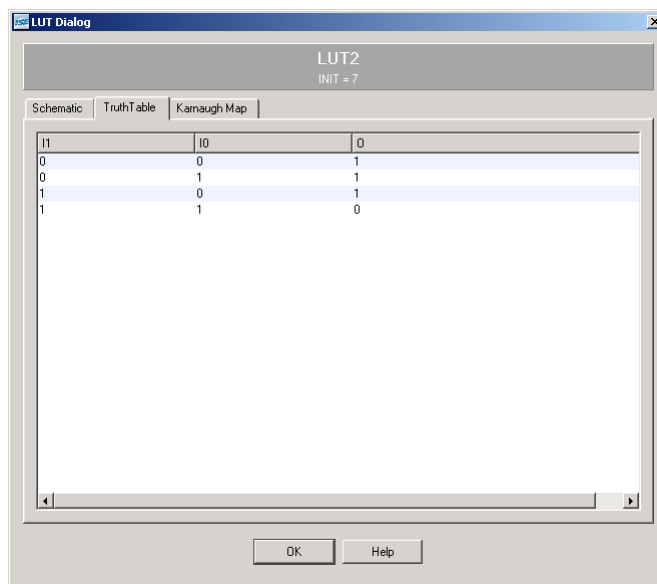


Рис. 1.15 – Наповнення функційної таблиці проекту my_lut2

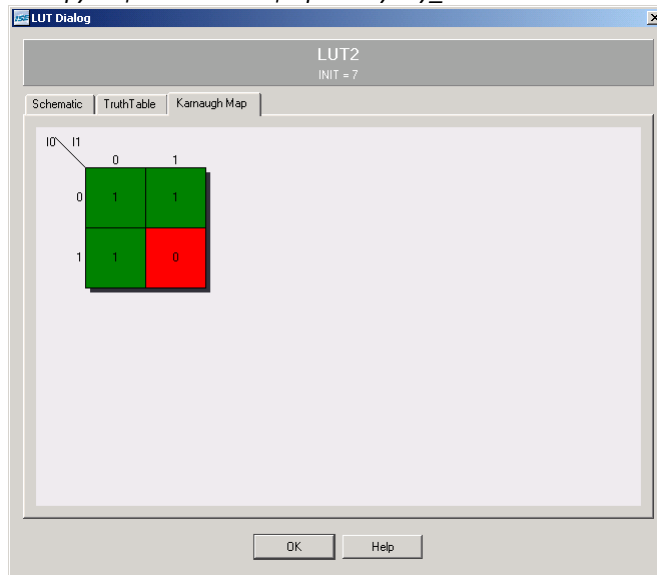


Рис. 1.16 – Еквівалентне подання наповнення функційної таблиці проекту my_lut2 картою Карно

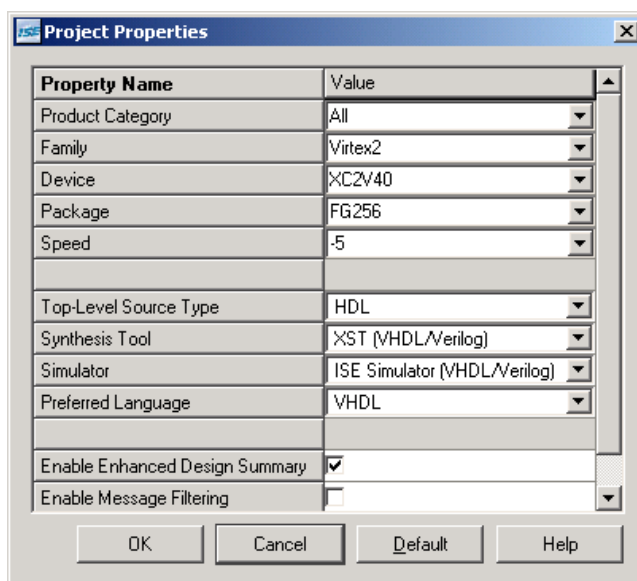


Рис. 1.17 – Заміна (заради зручності) зовнішнього симулятора проекту ModelSim на вбудований до Xilinx ISE WebPack так званий ISE Simulator

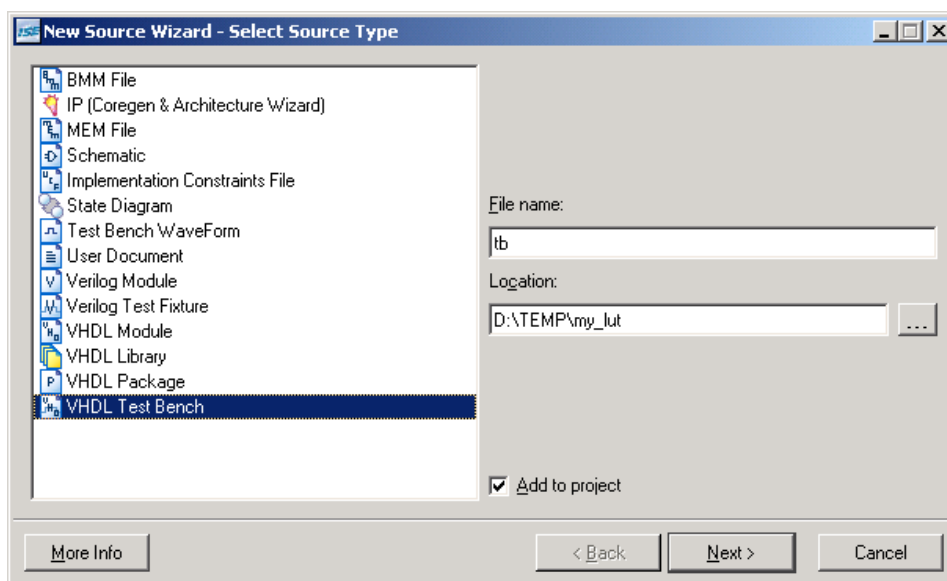


Рис. 1.18 – Створення нового додаткового файлу проекту типу TestBench з назвою tb, чого вимагає подальше проведення Post Place&Route (часової, а не поведінкової) симуляції, що відбувається на рівні вентилів, тлбто, з врахуванням затримок елементів і сполучень)

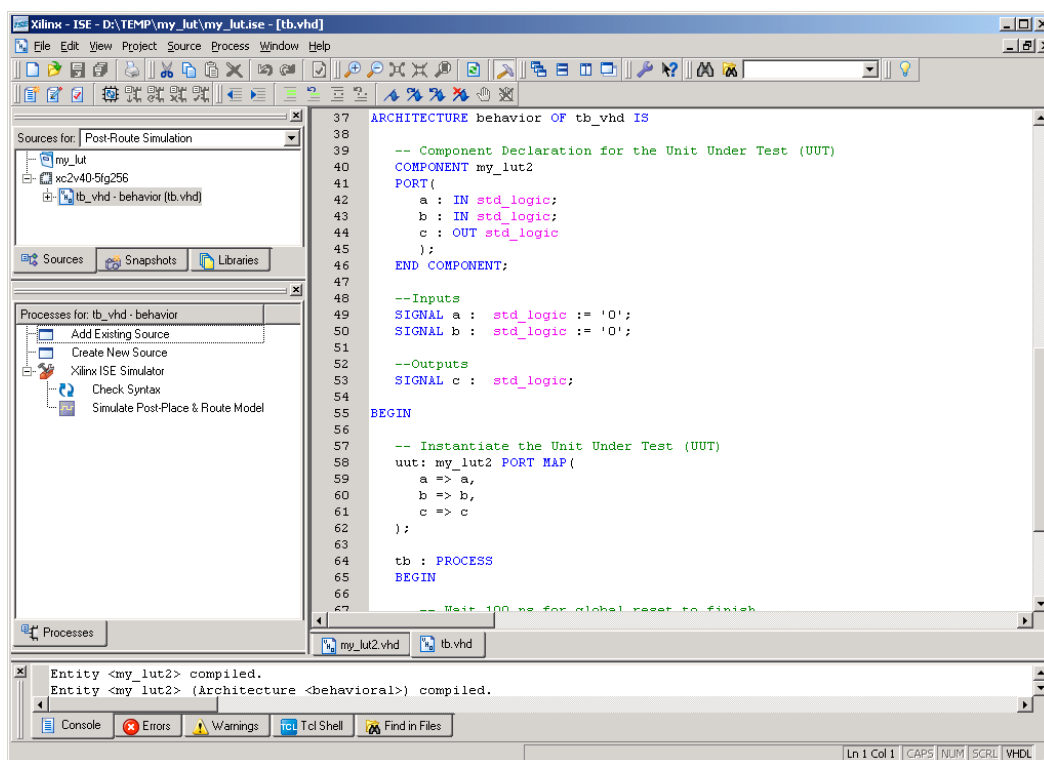


Рис. 1.19 – Первинний текст автоматично згенерованого тест-бенч файла; наш вентиль інстальовано, входні сигнали a,b занулені, їхні зміни в часі автоматом не визначені

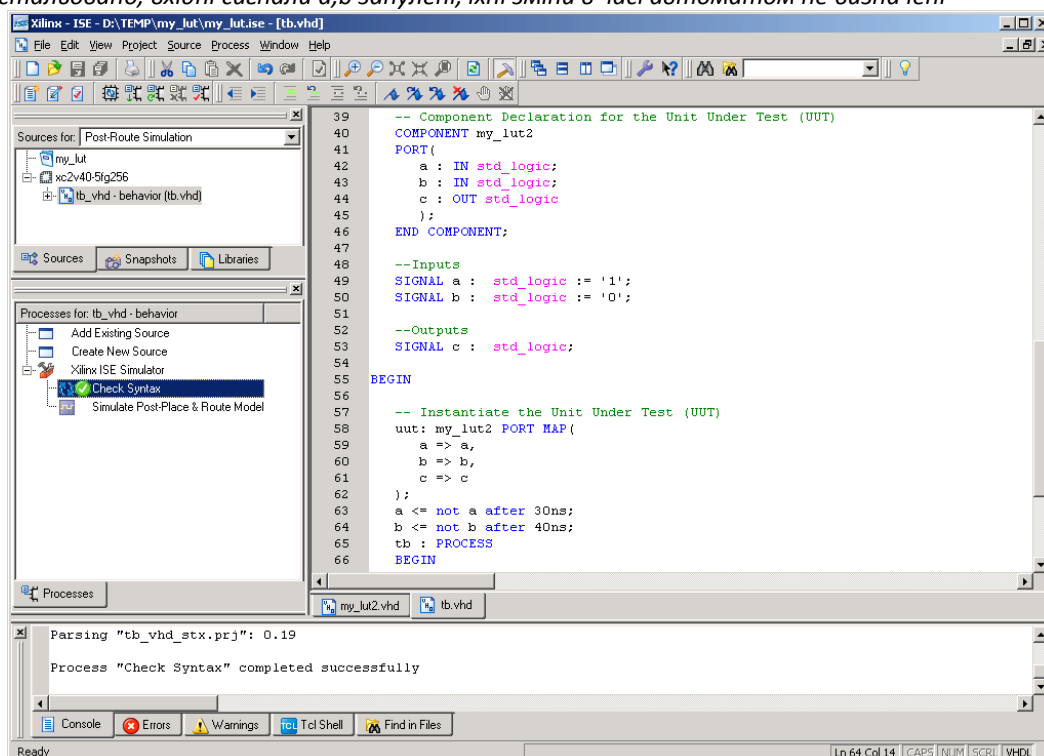


Рис. 1.20 – Змінений уручну текст тест-бенч файла; початкове значення сигналу a змінено з нуля на одиницю, сигнал a змінює власний рівень кожні 30 нс, а сигнал b – кожні 40 нс; перевірено синтаксис файла тест-бенч після принесених змін

Подамо ASCII варіант вмістимого файла конфігурування (bitstream) ПЛІС. Його бінарний (а не ASCII) варіант з розширенням *.bit завантажується до ПЛІС, що змушує ПЛІС виконувати проектну функцію. Отож, отримання конфігураційного файлу є для нас досягнутою метою і завершенням апаратного проекту.

```

0011000000000000011000000000000001
0000000000000000000000000000000000
0011000000000000010000000000000001
000000000000000000000000000001001
0011000000000000010000000000000001
0000000000000000000000000000000000
0011000000000000010000000000000001
0000000000000000000000000000000001
0011000000000000010000000000000000
010100000000000000010100100100010
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000

```

Всі перетворення, що виконані в проекті вкладаються в низку переходів:

VHDL → netlist (через синтез) → bitstream (через імплементацію, або ж втілення).

Зараз перейдемо до розгляду базової VHDL моделі, що має модифікуватися, імплементуватися і досліджуватися студентом при виконанні лабораторної роботи.

Базовий проект «Функційна таблиця»

При складанні VHDL моделі з низькорівневою архітектурою можливі прямі маніпуляції з функційною таблицею, як то подає наступний приклад:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity and2_lut2 is
    Port ( i0 : in std_logic;
          i1 : in std_logic;
          outp : out std_logic);
end and2_lut2;
architecture low_level of and2_lut2 is
    -- 2-Bit Look-Up-Table with General Output
    component LUT2
        port (
            I0 : in std_logic;
            I1 : in std_logic;
            O : out std_logic );
    end component;
    -- Attribute applied to instantiation
    attribute INIT : string;

```

```

attribute INIT of U1 : label is "8";
-- Attribute applied to component
-- attribute INIT : string;
-- attribute INIT of LUT2 : component is "B";
begin
U1: LUT2
  port map (I0 => i0,
            I1 => i1,
            O => outp);
end low_level;

```

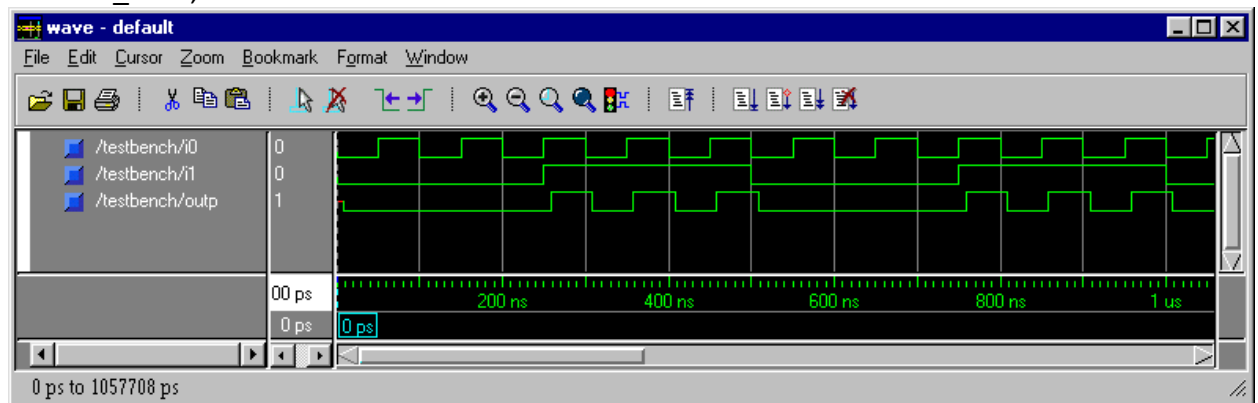


Рис. 1.22 – Часове симулювання табличної реалізації функції «І» на два входи

Пояснимо таблицю, чому чисельне значення VHDL атрибуту для $z = \text{and2}(x, y)$ має наведено в моделі значення.

Таблиця 1.1 – Атрибут табличної реалізації ФАЛ

x	y	$z = \text{and2}(x, y)$	атрибут
0	0	0	8 = 1000
0	1	0	
1	0	0	
1	1	1	

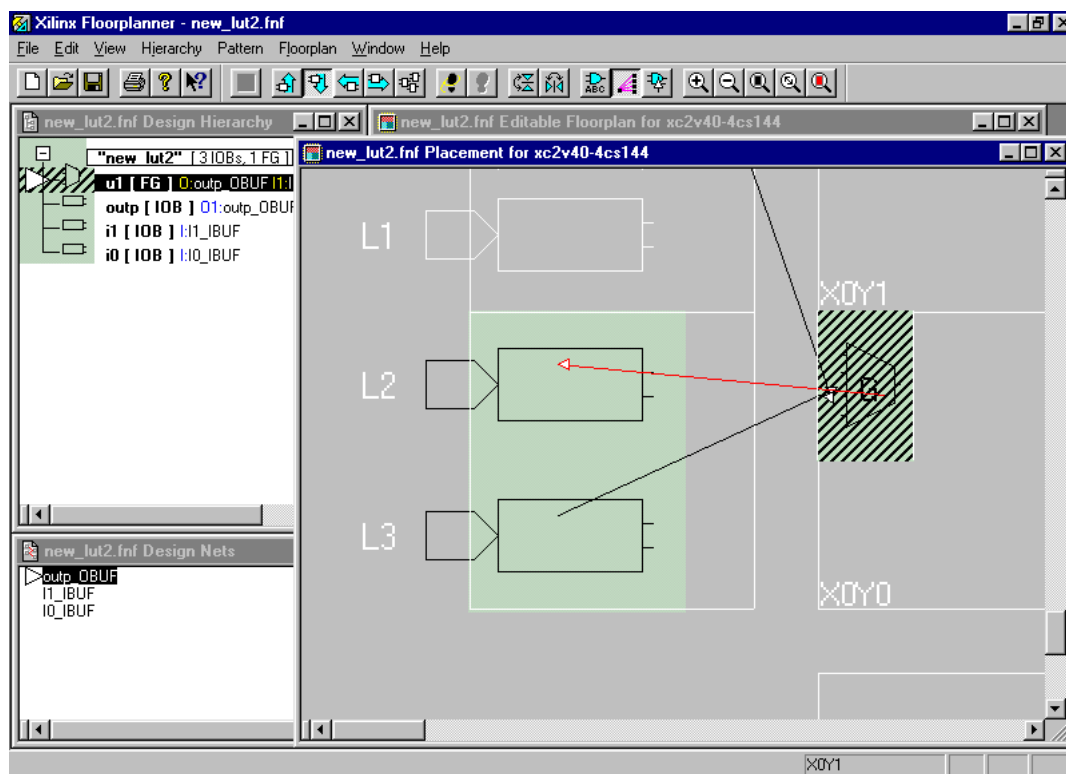


Рис. 1.23 – Топологія табличної реалізації логічної функції and2 на ПЛІС Virtex-2

ЛР № 2. Імплементування VHDL моделей операційного та керуючого пристроїв (автоматів)

Мета:

Опанування технікою VHDL проектування основних комп'ютерних автоматів

Завдання:

Засобами САПР WebPack/ModelSim імплементувати розширення базового проекту "Операційний пристрій". Запропоновану, власну імплементацію дослідити та верифікувати. Скласти звіт з виконання лабораторних досліджень та захистити його.

Теоретичні відомості

Спочатку розглянемо теоретичні питання, що стосуються реалізації комп'ютерної арифметики, а вже потім безпосередньо операційний пристрій.

Арифметика

У VHDL арифметичні операції зі знаковими та з беззнаковими значеннями уводять підключенням відповідних бібліотек. Коли проектують знакову арифметику, тоді бібліотеки VHDL створюють схеми, що використовують доповняльний код (two's complement) чисел.

Коли оперують із беззнаковими числами, тоді підключають наступні бібліотеки:

library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;

use IEEE.std_logic_unsigned.all;

Коли оперують із знаковими числами, тоді підключають наступні бібліотеки:

library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;

use IEEE.std_logic_signed.all;

Арифметичні маніпуляції

if clk'event **and** clk = '1' **then**

-- if the reset is asserted, set the counter value to zeros

if reset = '1' **then**

sum <= (**others** => '0');

-- if the set is asserted, set the counter value to ones

elsif set = '1' **then**

sum <= (**others** => '1');

-- if the load signal is asserted, load what is on the load input

elsif load = '1' **then**

sum <= load_value;

-- finally, if the adder/subtractor is enabled, do the operation,

-- which is dependant on the subtract input signal.

elsif enable = '1' **then**

```

    if subtract = '1' then
        sum <= ('0'&a) - ('0'&b);
    else
        sum <= ('0'&a) + ('0'&b);
    end if;
end if;
end if;

```

Числа зі знаком

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
entity magcomp is
port (
    a: in signed (7 downto 0);
    b: in signed (7 downto 0);
    a_gtet_b: out std_logic;
);
end magcomp;
architecture magcomp_arch of magcomp is
begin
    a_gtet_b <= '1' when a >= b else '0';
end magcomp_arch;

```

Порівняння чисел

```

-- VHDL module for a comparator
-- this module contains two implementations for the
-- comparsion. Not all synthesis tools will accept
-- both implementations. However, these two implementation
-- have been noticed to work with all tests synthesis tools.
-- input(s): a, b
-- output(s): a_et_b_1, a_et_b_2
-- include these three standard IEEE libraries.
-- they include arithmetic operations and conversion functions
-- necessary for mathematical operations.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- Note that since both the arith and unsigned libraries are used
-- in this design, use the explicit switch for compiling this design
-- in ModelSim eg. vcom -explicit comparator.vhd
entity comp is
-- define input and output ports
-- make the inputs 8 bits wide. can actually have any size here

```

```

-- output is always only one bit
port (
  a: in std_logic_vector (7 downto 0);
  b: in std_logic_vector (7 downto 0);
  a_et_b_1: out std_logic;
  a_et_b_2: out std_logic
);
end comp;
architecture comp_arch of comp is
  signal int: std_logic_vector (7 downto 0);
begin
  -- here a simple comparsion operator is done to test the equality.
  -- a one is assigned to the output is the values are equal.
  -- this method does not work for all synthesizers.
  -- Another method can be used if this coding style does not work
  -- with your synthesis tool.
  a_et_b_1 <= '1' when (a = b) else '0';
  -- in the following case a subtraction is used
  -- to test the equality. This explicitly indicates
  -- that a mathematical operation is to be used and
  -- thus it will utilize the carry chain.
  -- the assignment to a_et_b_2 is a check to see if the
  -- result of the subtraction is a zero. If it is zero the
  -- inputs are equal and the output signal is asserted.
  int <= a - b;
  a_et_b_2 <= '1' when (int = 0) else '0';
end comp_arch;

```

Порівняння модулів чисел

```

-- VHDL module for an unsigned magnitude comparator
-- input(s): a, b
-- output(s): a_gtet_b
-- include these three standard IEEE libraries.
-- they include arithmetic operations and conversion functions
-- necessary for mathematical operations.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- Note that since both the arith and unsigned libraries are used
-- in this design, use the explicit switch for compiling this design
-- in ModelSim
-- eg. vcom -explicit magcomp.vhd
entity magcomp is
  -- define input and output ports

```

```

port (
  a: in std_logic_vector (7 downto 0);
  b: in std_logic_vector (7 downto 0);
  a_gtet_b: out std_logic
);
end magcomp;
architecture magcomp_arch of magcomp is
begin
  -- here the output assignment can be a simple evaluation of the input
  -- data. This coding style will produce the desired circuit in
  -- an efficient and compact way.
  a_gtet_b <= '1' when a >= b else '0';
end magcomp_arch;

```

Доповняльний код

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- Note that since both the arith and unsigned libraries are used
-- in this design, use the explicit switch for compiling this design
-- in ModelSim
-- eg. vcom -explicit twoscomp.vhd
entity twoscomp is
port ( d: in std_logic_vector (7 downto 0);
  complement: in std_logic;
  s: out std_logic_vector (7 downto 0)
);
end twoscomp;
architecture twoscomp_arch of twoscomp is
begin
  -- Below is one type of implementation
  add_sub: process(complement, d)
  begin
    if (complement = '1') then
      s <= (d XOR "11111111") + '1';
    else
      s <= d;
    end if;
  -- Below here is a second implementation, much like an adder/subtractor
  if (complement = '1') then
    s <= "00000000" - d;
  else
    s <= "00000000" + d;
  end if;
end process;
end twoscomp_arch;

```

```

end if;
end process add_sub;
end twoscomp_arch;

```

Базова поведінкова модель операційного пристрою

Текст проекту (поведінкова модель) комбінаційного чотирьох бітового операційного пристрою на вісім операцій подано нижче. Реалізовано наступні операції:

- генерація константи *нуль*,
- генерація константи *одиниця*,
- логічний добуток,
- логічна сума,
- логічний зсув праворуч на один біт,
- логічний зсув ліворуч на один біт.

VHDL модель проекту.

-- Copyright © 1997, XILINX Inc., USA

-- CONSTANT_ALU.VHD Version 1.0

```

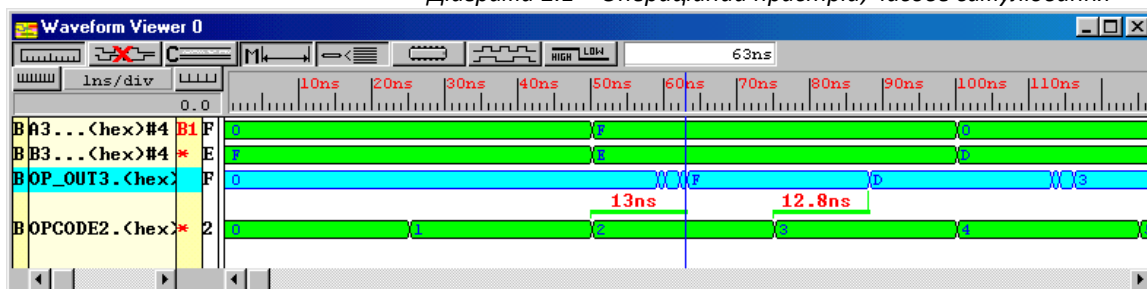
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity constant_alu is
    port (OPCODE : in STD_LOGIC_VECTOR(2 downto 0);
          A, B : in STD_LOGIC_vector(3 downto 0);
          OP_OUT : out STD_LOGIC_vector(3 downto 0));
end constant_alu;
architecture alu_BEHAV of constant_alu is
    constant ZERO : STD_LOGIC_VECTOR (2 downto 0) := "000";
    constant A_AND_B : STD_LOGIC_VECTOR (2 downto 0) := "001";
    constant A_OR_B : STD_LOGIC_VECTOR (2 downto 0) := "010";
    constant ONE : STD_LOGIC_VECTOR (2 downto 0) := "111";
    constant A_PL_B : STD_LOGIC_VECTOR (2 downto 0) := "011";
    constant A_MI_B : STD_LOGIC_VECTOR (2 downto 0) := "100";
    constant A_SRL : STD_LOGIC_VECTOR (2 downto 0) := "101";
    constant A_SLL : STD_LOGIC_VECTOR (2 downto 0) := "110";
begin
    process (OPCODE, A, B)
    begin
        if (OPCODE = A_AND_B) then OP_OUT <= A and B;
        elsif (OPCODE = A_OR_B) then OP_OUT <= A or B;
        elsif (OPCODE = A_PL_B) then OP_OUT <= A + B;
        elsif (OPCODE = A_MI_B) then OP_OUT <= A - B;
        elsif (OPCODE = A_SRL) then OP_OUT <= '0' & A(3 downto 1);

```

```
    elsif (OPCODE = A_SLL) then OP_OUT <= A(2 downto 0) & '0';  
    elsif (OPCODE = ONE) then OP_OUT <= x"f";  
    else OP_OUT <= x"0";  
    end if;  
end process;  
end alu_BEHAV;
```

Результати симулювання синтезованої та імплементованої базової моделі подано наступною симуляційною часовою діаграмою.

Діаграма 2.1 - Операційний пристрій, часове симулювання



Риску вимірювача часу поставлено на поділку 63 нс. На 63-й наносекунді змінився результат відповідно до коду операції 2 (логічне додавання). Отримано результат $F \text{ and } E = E$ із затримкою 13 нс. Потім із затримкою 12.8 нс отримано суму (код операції 3) $F + E = D$ (перевірка: $15 + 14 = 29$). Отже, АЛП функціонує коректно.

Проект реалізовано та верифіковано засобами інструментальної системи Xilinx Foundation™.

Приклад проведення експерименту для проекту «Операційний пристрій»

Дослідження апаратного множення виконано в спосіб зміни базової VHDL моделі (див. наступний фрагмент VHDL коду):

```
architecture alu_BEHAV of constant_alu is
    constant ZERO : STD_LOGIC_VECTOR (2 downto 0) := "000";
    constant A_AND_B: STD_LOGIC_VECTOR (2 downto 0) := "001";
    constant A_OR_B : STD_LOGIC_VECTOR (2 downto 0) := "010";
    constant ONE   : STD_LOGIC_VECTOR (2 downto 0) := "111";
    constant A_PL_B : STD_LOGIC_VECTOR (2 downto 0) := "011";
    constant A_MI_B : STD_LOGIC_VECTOR (2 downto 0) := "100";
    constant A_SRL  : STD_LOGIC_VECTOR (2 downto 0) := "101";
    constant A_SLL  : STD_LOGIC_VECTOR (2 downto 0) := "110";
begin
    process (OPCODE, A, B)
    begin
        if (OPCODE = A_AND_B) then OP_OUT <= A and B;
        elsif (OPCODE = A_OR_B) then OP_OUT <= A or B;
        elsif (OPCODE = A_PL_B) then OP_OUT <= A + B;
        -- заміна віднімання на апаратне множення
        elsif (OPCODE = A_MI_B) then OP_OUT <= A * B;
        elsif (OPCODE = A_SRL) then OP_OUT <= '0' & A(3 downto 1);
        elsif (OPCODE = A_SLL) then OP_OUT <= A(2 downto 0) & '0';
        elsif (OPCODE = ONE) then OP_OUT <= x"f";
        else
            OP_OUT <= x"0";
        end if;
    end process;
end;
```

```
end process;
end alu_BEHAV;
```

Ясно, що вказані зміни не є коректними, адже не було змінено розрядність результату OP_OUT. Отож, старшу цифру добутку втрачено. Проте цією заміною типу виконуваної операції і після виконання синтезу на автоматично згенерованій технологічній схемі можна виявити присутність примітиву апаратного перемножувача з назвою MULT18x18.

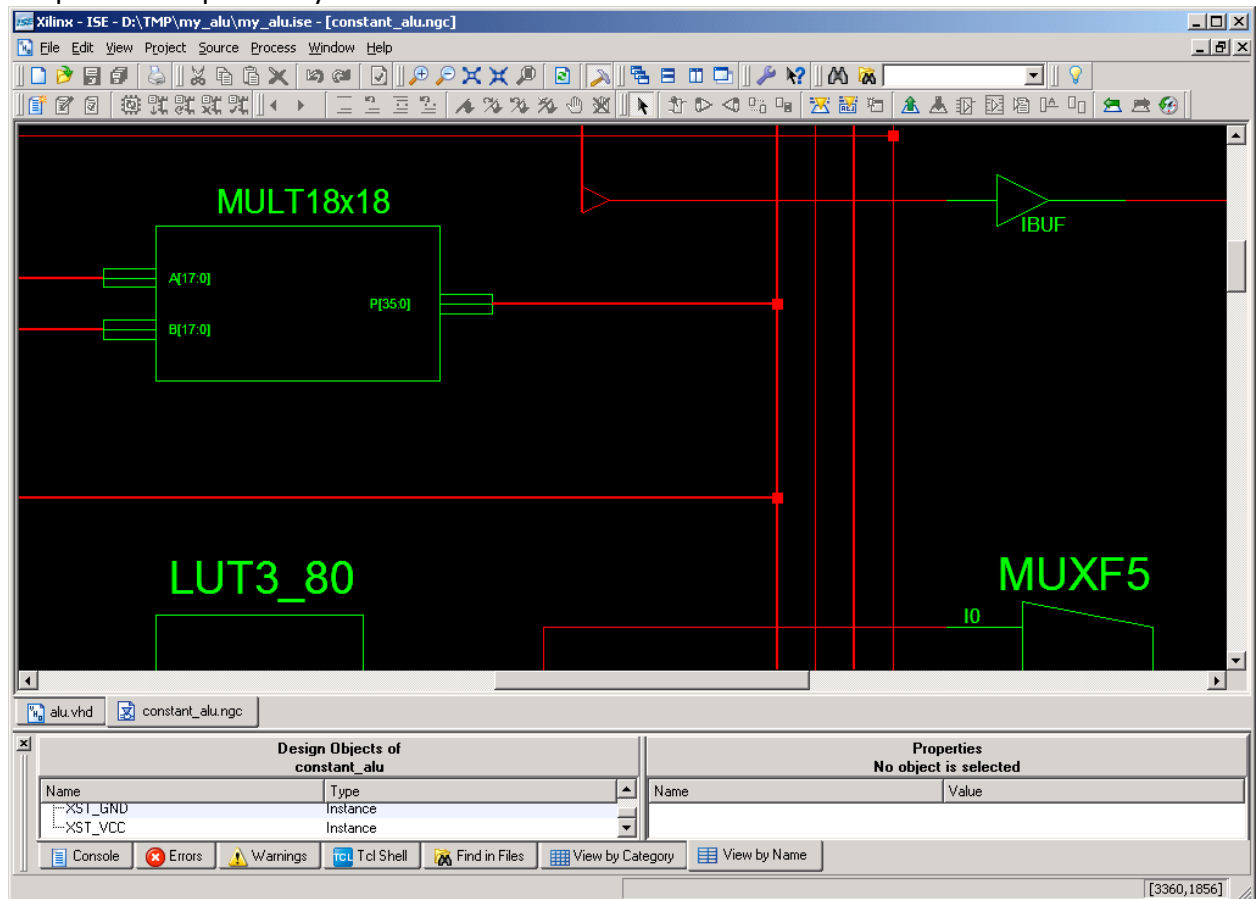


Рис.2.1 – Апаратний перемножувач (примітив ПЛІС, або ж бібліотечний елемент) MULT18x18 на технологічній схемі зміненого введення операції множення операційного пристрою

Необхідні зміни в тексті автоматично згенерованого файлу тест-бенч

Автоматично згенерований тест-бенч містить нульові стартові значення всіх вхідних сигналів. Їх потрібно замінити на ненульові значення наступним чином. Коду операції надаємо значення, відповідне операції, що тестуємо (при цьому пам'ятаємо, що ми замінили операцію віднімання на множення, але не змінили розрядності та назви констант). Отже, код операції множення тепер є 4. а значення чотирибітових операндів $A = 0xF$ та $B = 7$ вибираємо навмання, так само, як і моменти набуття операндами цих значень. Після привнесення змін отримуємо наступне вміст файлу тест-бенч:

--Фрагмент test-bench

--Inputs

SIGNAL OPCODE : std_logic_vector(2 downto 0) := (others=>'0');

SIGNAL A : std_logic_vector(3 downto 0) := (others=>'0');

SIGNAL B : std_logic_vector(3 downto 0) := (others=>'0');

--Outputs

SIGNAL OP_OUT : std_logic_vector(3 **downto** 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: constant_alu PORT MAP(

 OPCODE => OPCODE,

 A => A,

 B => B,

 OP_OUT => OP_OUT

);

OPCODE <= "100" after 200 ns; -- привнесена уручну зміна

A <= "1111" after 150 ns; -- привнесена уручну зміна

B <= "0111" after 220 ns; -- привнесена уручну зміна

tb : **PROCESS**

BEGIN

...

Часове симулювання зміненого проекту «Операційний пристрій»

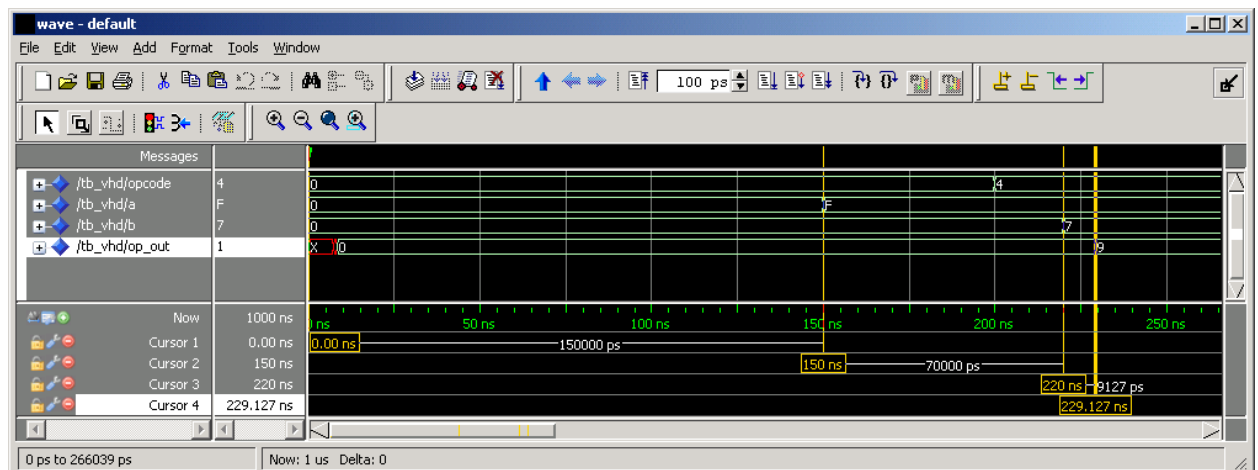


Рис.2.2 – Верифікаційні часові діаграма для зміненого проекту

На 150 нс змінна A набула значення 0xF. На 220 нс операнд B набув значення 0x7. На 200 нс код операції opcode набув значення 4 (після внесення змін до VHDL моделі код 4 відповідає не відніманню, а множенню). З затримкою в 9,127 нс на виході отримали молодшу цифру добутку $9 = 0xF \times 7 = 15 \times 7 = 105 = 9 +$

16×6). Отже, молодша цифра добутку виявилася очікуваною. Верифікація відбулася.

Автомат Мура

Завдання

Засобами САПР Xilinx ISE WebPack/ModelSim імплементувати проект «Автомат Мура» на основі дослідження і розширення наданої базової (прототипної) моделі. Змінену студентом базову модель імплементувати та верифікувати. Скласти звіт з виконання цієї частини лабораторних досліджень та захистити його.

Базова VHDL модель автомата

Фінітну машину станів (ФМС, скінчений автомат) використовують як модель пристроїв керування. Класичними ФМС є скінчені автомати Мілі (Mealy) та Мура (Moore).

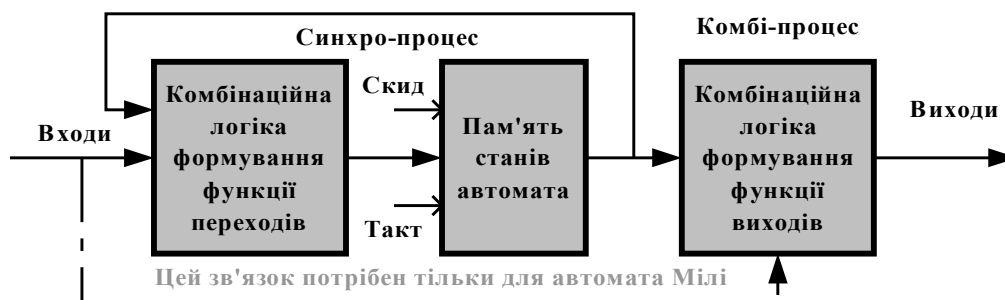


Рис. 2.3 – Структурні автомати (ФМС) Мура і Мілі

Далі подамо прототипну VHDL модель автомата Мура, що містить три процеси, а саме:

- синхропроцес, що зростаючим фронтом тактового імпульсу змінює поточний внутрішній стан автомата на наступний,
- процес формування функції переходів, тобто, комбінаційного обчислення наступного стану автомата в залежності від поточного стану та поточного вхідного сигналу,
- процес формування вихідної функції, що залежить в автоматі Мура виключно від поточного внутрішнього стану.

Зауважимо, що VHDL модель софтверного процесора Gnome, що використовують в курсовому проектуванні, також містить автомат (фінітну машину станів – FSM) з трьома вище зазначеними процесами.

-- Moore State Machine with three processes.

```
library IEEE;
```

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

```
entity fsm_3 is
```

```
    port ( clk, reset, x1 : IN std_logic;
```

```

        outp      : OUT std_logic);
end entity;
architecture beh1 of fsm_3 is
    type state_type is (s1,s2,s3,s4);
    signal state, next_state: state_type ;
begin
    synchro_process: process (clk,reset)
    begin
        if (reset ='1') then
            state <=s1;
        elsif (clk='1' and clk'Event) then
            state <= next_state;
        end if;
    end process synchro_process;
    transition_function_process : process (state, x1)
    begin
        case state is
            when s1 => if x1='1' then
                next_state <= s2;
            else
                next_state <= s3;
            end if;
            when s2 => next_state <= s4;
            when s3 => next_state <= s4;
            when s4 => next_state <= s1;
        end case;
    end process transition_function_process;
    output_function_process : process (state)
    begin
        case state is
            when s1 => outp <= '1';
            when s2 => outp <= '1';
            when s3 => outp <= '0';
            when s4 => outp <= '0';
        end case;
    end process output_function_process;
end beh1;

```

Часова верифікація базової моделі автомата

Подамо отримані часовою симуляцією результати верифікації прототипної моделі. Сутність та пояснення щодо отриманих з власної модифікації автомата часових діаграм потрібно надати в звіті з виконання лабораторної роботи.

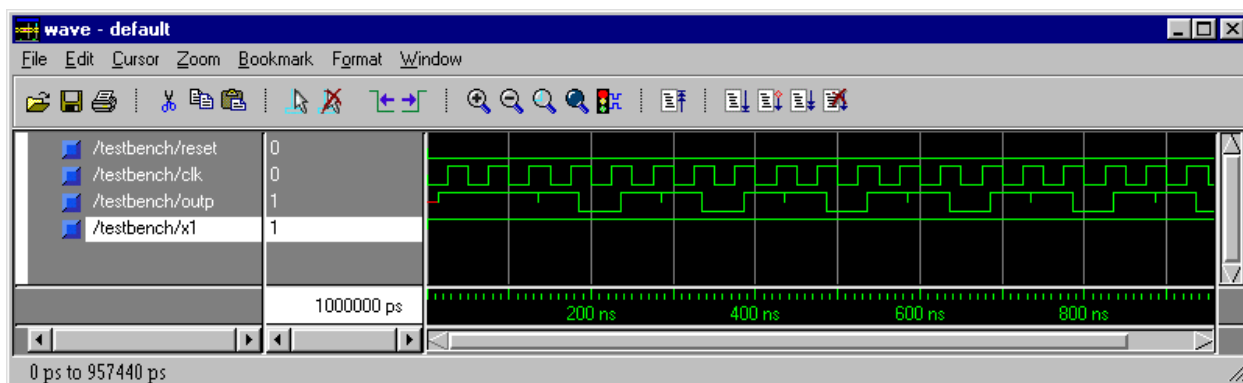


Рис. 2.4 – Автомат Мура. Часове симулювання за умови, що $x1=1$

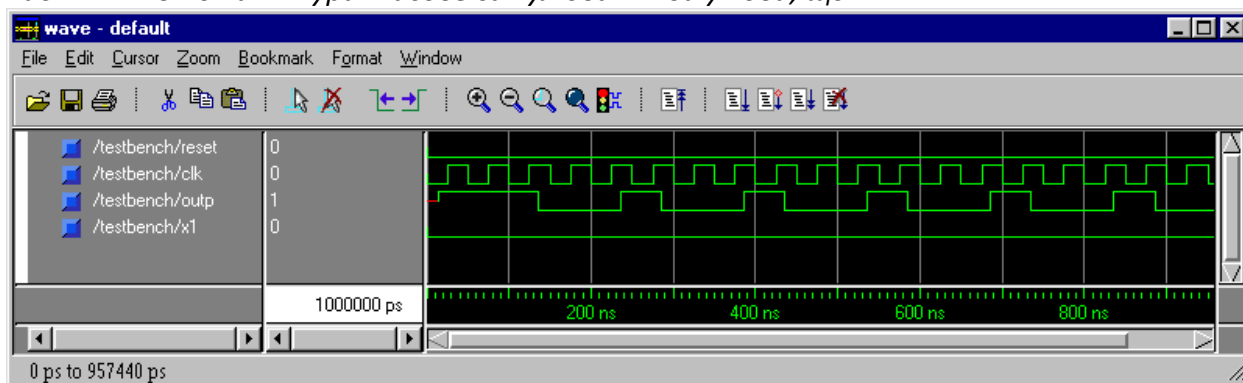


Рис. 2.5– Автомат Мура. Часове симулювання за умови, що $x1=0$

Вихідний сигнал автомата, як це бачимо з діаграм, відповідає моделі. За умови, коли $x1=1$, автомат двічі поспіль видає на виході одиничний рівень сигналу та ще один раз – нульовий рівень. Все це циклічно повторюється. За умови, коли $x1=0$, послідовність станів вихідного сигналу є оберненою. Отже, поведінка імплементації є очікуваною, коректною.

ЛР № 3А. Імплементування VHDL моделей двохнаправленої шини та LVDS шини

Мета:

Опанування технікою використання шин в системних проектах на ПЛІС

Завдання

В САПР ISE WebPack/ModelSim імплементувати власний проект двохнаправленої шини та власний проект LVDS шини. Власні імплементування синтезувати та верифікувати. Скласти звіт про виконання лабораторних досліджень та захистити його.

Базовий варіант виконання лабораторної роботи

Розглянемо проект двохнаправленої шини із трьома станами, що має наступну поведінкову VHDL модель, де високоімпедансний («третій») стан позначено символом «Z».

-- V. Trotsenko, 2002

library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_unsigned.all;

entity tristate **is**

port (

 clk: **in** STD_LOGIC;

 rstn: **in** STD_LOGIC;

 read_state: **out** std_logic;

 write_state: **out** std_logic;

 bus_out: **out** STD_LOGIC_VECTOR (7 downto 0); -- standard output bus

 z_bus: **inout** STD_LOGIC_VECTOR (7 downto 0)); -- tristate bus

end tristate;

architecture tristate_arch **of** tristate **is**

signal chronometer: std_logic_vector(7 downto 0);

signal bus_register: std_logic_vector(7 downto 0);

signal write_to_bus, read_from_bus : std_logic;

begin

 write_to_bus <= '1' **when** chronometer > "00000011" **and** chronometer
 < "00010000" **else**

 '0'; -- multiplexor

 read_from_bus <= '1' **when** chronometer > "00011111" **and** chronometer
 < "00111111" **else**

 '0'; -- multiplexor

 z_bus <= "01110111" **when** write_to_bus = '1' **else**

 (others => 'Z'); -- multiplexor, write cycle

 bus_out <= bus_register; -- out from chip

 write_state <= write_to_bus; -- write-gate signal, chip output

```

read_state <= read_from_bus; -- read-gate signal, chip output
U1: process (clk, rstn, read_from_bus) -- reading from tristate bus
begin
  if rstn = '0' then
    bus_register <= (others => '0');
  elsif read_from_bus = '1' then
    if clk'event and clk = '1' then
      bus_register <= z_bus; -- read cycle
    else
      null;
    end if;
  else null;
  end if;
end process;
U2: process (clk, rstn)
begin
  if rstn = '0' then
    chronometer <= (others => '0');
  elsif clk'event and clk = '1' then
    chronometer <= chronometer + '1';
  else null;
  end if;
end process;
end tristate_arch;

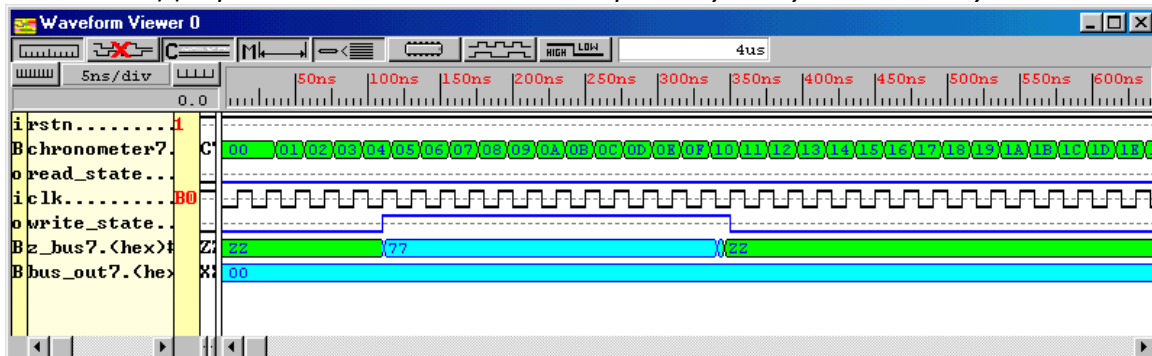
```

Зауважимо наступне. Аби користуватися двохнаправленою шиною достатньо реалізувати запис на неї та читання з неї. За допомогою хронометра формуємо часові ворота на запис та на читання так, аби вони не перетиналися в часі. Хронометр (таймер) виконує лише цю допоміжну функцію, тому послідовність зміни його станів в часі є непринциповою.

Цикл запису

Як засвідчує діаграма 3.15 запису, ми дійсно пишемо на шину код 77_{16} , тимчасово перериваючи стан 'Z' на шині протягом часового інтервалу після 100 нс і до 350 нс. Тут двонаправлену шину ми, фактично, перетворюємо на звичайну вихідну шину, принаймні, на зазначеному часовому інтервалі.

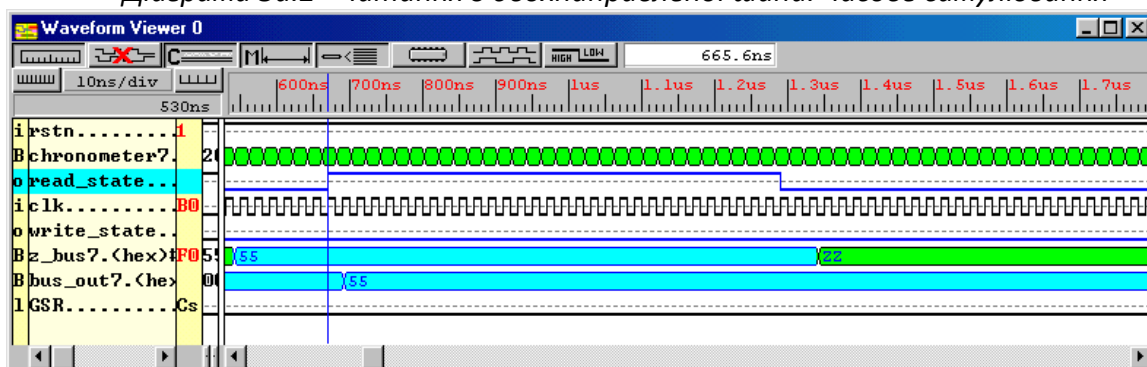
Діаграма 3а.1 - Запис на двохнаправлену шину. Часове симулювання



Цикл читання

Діаграма читання з шини ілюструє той факт, що виводячи примусово (від зовнішнього щодо ПЛІС джерела) на шину код 55_{16} , із перериванням її поточного 'Z' стану під час існування воріт читання, ми маємо можливість прочитати (в кристалі ПЛІС та засобами ПЛІС) цей код на деяку іншу (в нашому прикладі – вивідну з кристала ПЛІС, зараз вже однонаправлену, спрощену в користуванні) шину. Це виконуємо з деякою затримкою по відношенню до старту воріт читання.

Діаграма За.2 - Читання з двохнаправленої шини. Часове симулювання



Ясно, що VHDL модель двохнаправленої шини є нескладною; вона придатна до синтезу. Сучасні ПЛІС дозволяють реалізувати двохнаправлену шину як безпосередньо на кристалі, так і ззовні кристала. Внутрішня реалізація двохнаправленої шини є корисною для ПЛІС високого рівня інтеграції, які використовують у великих проектах, наприклад, в проектах так званих систем на кристалі (SoC).

Шина LVDS

LVDS (низьковольтові диференційні сигнали) is a popular and powerful high-speed interface used in many systems applications. Virtex-II I/Os are designed to comply with the IEEE 1596.3 electrical specification for LVDS, making system and board design easier.

Розглянемо прототипну реалізацію зв'язку поміж двома ПЛІС, розташованими на тих самих або, навіть, на різних друкованих узлах (в межах одного блоку) засобами послідовної шини стандарту LVDS. Цей стандарт дозволяє пересилати сигнали з інтенсивністю до 0.7 Гб/сек.

Далі подано вірцевий VHDL проект, де використовують примітиви LVDS приймача **IBUFDS_LVDS_25** та LVDS передавача **OBUFDS_LVDS_25** з тристабільним виходом, що можна імплементувати, наприклад, в ПЛІС Віртекс-2.

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

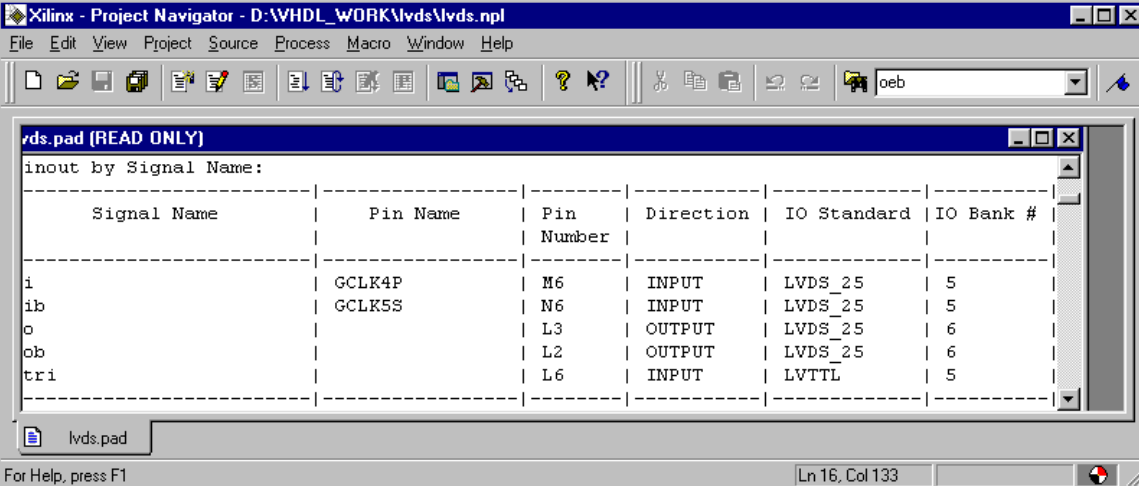
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

-- library unisim;
-- use unisim.vcomponents.all;
entity lvds is
  Port ( i : in std_logic;
        ib : in std_logic;
        tri: in std_logic;
        o : out std_logic;
        ob : out std_logic);
end lvds;
architecture low_level of lvds is
  component IBUFDS_LVDS_25
    port(
      I      : in std_logic;
      IB     : in std_logic;
      O      : out std_logic );
  end component;
  component OBUFTDS_LVDS_25
    port(
      I      : in std_logic;
      T      : in std_logic;
      O      : out std_logic;
      OB     : out std_logic
    );
  end component;
  signal wire_in : std_logic;
  signal wire_out : std_logic;
begin
  U0: wire_in <= not wire_out;
  U1: IBUFDS_LVDS_25
    port map ( I => i, -- P-Channel input to LVDS in_buffer
              IB => ib, -- N-Channel input to LVDS in_buffer
              O => wire_out -- Output to FPGA fabric );
  U2: OBUFTDS_LVDS_25
    port map (
      I => wire_in, -- Input from FPGA fabric
      T => tri,    -- 3-State control input to LVDS out-buffer
      O => o,      -- P-Channel Output of LVDS buffer
      OB => ob     -- N-Channel Output of LVDS buffer
    );
end low_level;

```

За результатами імплементації САПР автоматично призначила контакти сигналам.



lvds.pad (READ ONLY)

Pinout by Signal Name:

Signal Name	Pin Name	Pin Number	Direction	IO Standard	IO Bank #
i	GCLK4P	M6	INPUT	LVDS_25	5
ib	GCLK5S	N6	INPUT	LVDS_25	5
o		L3	OUTPUT	LVDS_25	6
ob		L2	OUTPUT	LVDS_25	6
tri		L6	INPUT	LVTTL	5

lvds.pad

For Help, press F1

Ln 16, Col 133

Рис. 3а.1 – Автоматичний розподіл сигналів проекту по контактах ПЛІС

Видно, що отримана імплементація проекту до ПЛІС використовує для сигналів контакти в стандартах LVTTTL (для сигналу tri) та LVDS-25 (для диференційних сигналів i, ib, o, ob). В цьому ми переконатися.

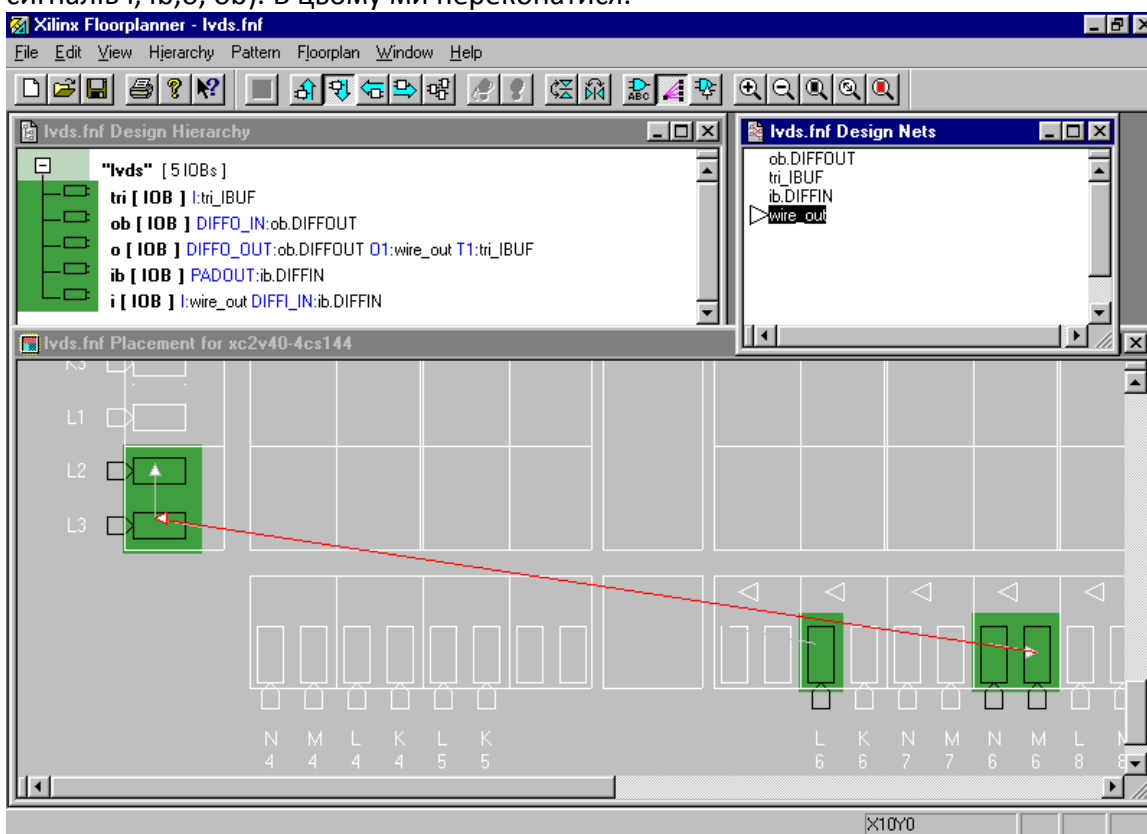


Рис. 3а.2 – Топологія проекту «Шина LVDS»

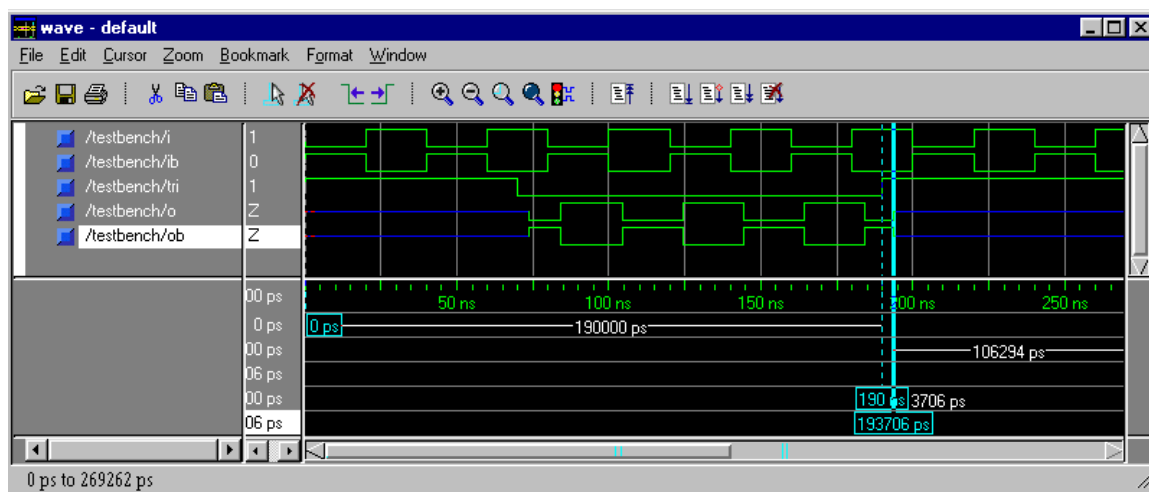


Рис. За.4 – Часове симулювання функціонування LVDS шини (ПЛІС Віртекс-2)

Синіми лініями на часовій діаграмі позначений високоімпедансний стан дротів вихідного каналу LVDS. Цим станом керує сигнал *tri*.

ЛР № 3Б. Синтез та дослідження DLL/DCM тактування проектів на ПЛІС

Мета:

Опанування технікою тактування проектів на ПЛІС

Завдання:

В САПР ISE WebPack/ModelSim імплементувати в ПЛІС Віртекс базовий і власний розширений проект «DLL тактування». Імплементатії верифікувати. Скласти звіт з виконання лабораторних досліджень та захистити його.

Базовий варіант виконання лабораторної роботи

Досягнення найвищої продуктивності проекту на ПЛІС вимагає, як правило, використання апаратури цифрового автоналаштування частоти (АНЧ, digital delay-lock loops, DLLs). Апаратура DLL дозволяє досягнути прецизійної синхронізації зовнішньої та внутрішньої тактових частот проекту. Фірма Ксайлінкс була першою в царині програмованої логіки, яка вбудувала до кожної ПЛІС типу Віртекс чотири 200 МГц DLLs. ПЛІС Віртекс-Е мала вже вісім DLL, спроможних функціонувати до частоти 311 МГц.

DLL ПЛІС Віртекс дозволяють виробляти і прецизійно використовувати розставлені розробником в часі фронти тактових імпульсів за умови застосування ним накристалічних засобів прецизійної затримки фронтів тактових сигналів та прецизійного множення або ділення частоти цих тактових сигналів. Наприклад, прецизійного налаштування тактової частоти вимагають такі високопродуктивні апікації, як контролери DDR (Double Data Rate) синхронної динамічної пам'яті. Без цього продуктивність розроблених проектів на ПЛІС драматично зменшується. Блок-схему DLL подає наступний рисунок.

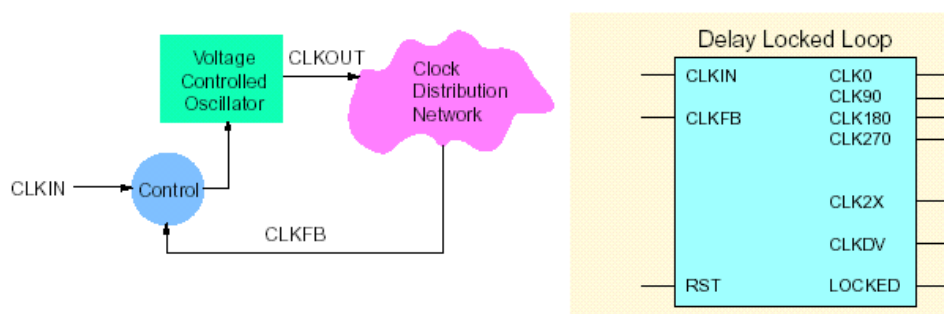


Рис. 36.1 – Блок-схема системи цифрового автоналаштування тактової частоти та умовне позначення блоку DLL (праворуч)

Подано VHDL код базового (прототипного) варіанту проекту. Наголосимо, що в ньому VHDL код є апаратно залежним (тому скористалися структурною архітектурою). Після синтезу VHDL коду отримали та нижче подали RTL схему проекту. Пропонуємо студентам самостійно порівняти відомий поведінковий та наданий нижче структурний VHDL код та зробити відповідні висновки.

-- XAPP132 DLL 1X and 2X Example

```

library ieee;
use ieee.std_logic_1164.all;
entity dll_standard is
    port (CLKIN : in std_logic;
          RESET : in std_logic;
          CLK0  : out std_logic;
          CLK2X : out std_logic;
          LOCKED: out std_logic);
end dll_standard;
architecture structural of dll_standard is
component IBUFG
    port(
        O :    out STD_ULOGIC;
        I :    in  STD_ULOGIC);
end component;
component IBUF
    port(
        O :    out STD_ULOGIC;
        I :    in  STD_ULOGIC);
end component;
component CLKDLL
    port ( CLKIN  : in std_ulogic := '0';
          CLKFB  : in std_ulogic := '0';
          RST    : in std_ulogic := '0';
          CLK0   : out std_ulogic := '0';
          CLK90  : out std_ulogic := '0';
          CLK180 : out std_ulogic := '0';
          CLK270 : out std_ulogic := '0';
          CLK2X  : out std_ulogic := '0';
          CLKDV  : out std_ulogic := '0';
          LOCKED : out std_ulogic := '0');
end component;
component BUFG
    port(
        O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;
component OBUF
    port(
        O :    out STD_ULOGIC;
        I :    in  STD_ULOGIC);
end component;
signal CLKIN_w, RESET_w, CLK0_dll, CLK0_g, CLK2X_dll, LOCKED_dll : std_logic;

```

begin

clkpad : IBUFG **port map** (I=>CLKIN, O=>CLKIN_w);

rstpad : IBUF **port map** (I=>RESET, O=>RESET_w);

dll : CLKDLL **port map** (CLKIN=>CLKIN_w, CLKFB=>CLK0_g, RST=>RESET_w,
CLK0=>CLK0_dll, CLK90=>**open**, CLK180=>**open**,
CLK270=>**open**, CLK2X=>CLK2X_dll, CLKDV=>open,
LOCKED=>LOCKED_dll);

clk0 : BUFG **port map** (I=>CLK0_dll, O=>CLK0_g);

clk2xg : BUFG **port map** (I=>CLK2X_dll, O=>CLK2X_g);

lckpad : OBUF **port map** (I=>LOCKED_dll, O=>LOCKED_g);

CLK0 <= CLK0_g;

end structural;

Поданий код узято з САПР Xilinx ISE WebPack (пункт меню Language Templates).

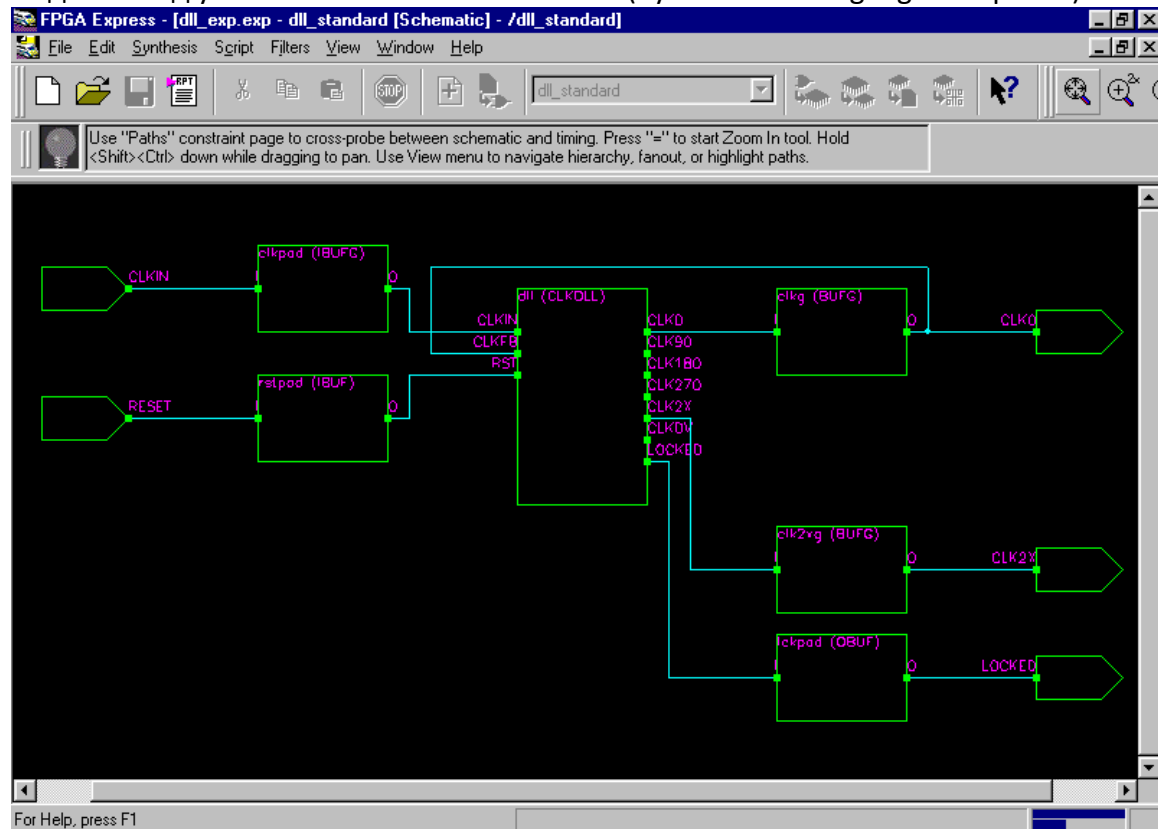
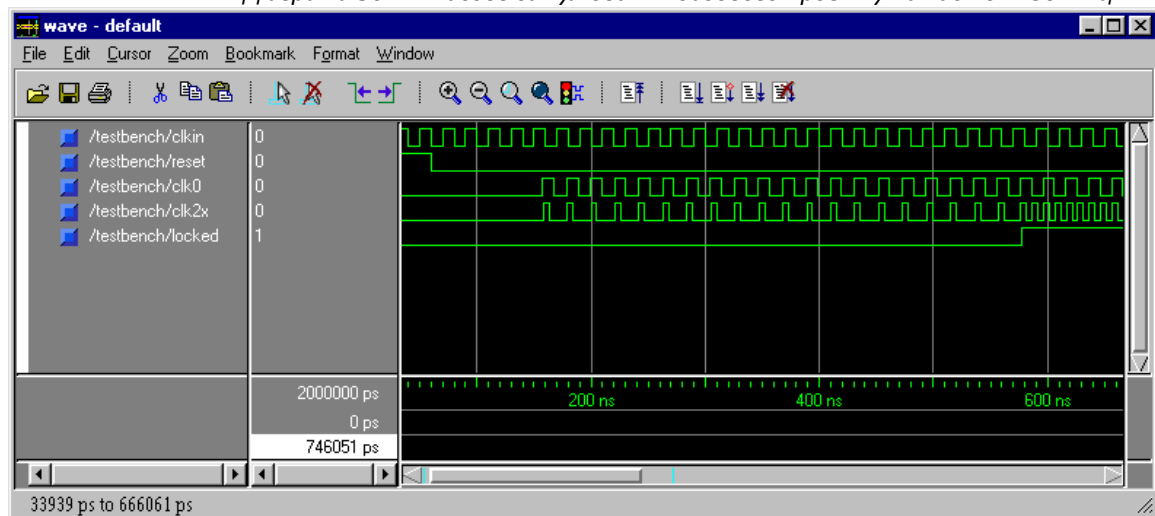


Рис. 36.2 – Автоматично синтезована схема менеджера тактування

Зауважимо, що завдяки використанню в проекті глобальних буферів всі тактові сигнали залишаються на металевих доріжках (з малою + передбачуваною затримкою, тобто, усі апаратні елементи мають синхронізовані поміж собою годинники)). Адже ми не дозволяємо цим сигналам розповсюджуватися полікремнієвими доріжками (з великою і практично не визначальною затримкою). Так чи інакше, проте не можна імпульсні сигнали (фронти) опрацьовувати VHDL кодом так само, як і повільні потенціальні сигнали (рівні).

Часове симулювання базового проекту

Діаграма 36.1 - Часове симулювання базового проекту на частоті 50 МГц



Часові діаграми чітко подають деталі процесу генерування синхронної з основною, проте подвоєної тактової частоти.

Далі подамо копії двох сторінок довідника lib.pdf з бібліотечних елементів ПЛІС Xilinx, що стосуються елементу DCM (менеджера тактових сигналів).

DCM

Digital Clock Manager

Architectures Supported

DCM	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

DCM is a digital clock manager that provides multiple functions. It can implement a clock delay locked loop, a digital frequency synthesizer, digital phase shifter, and a digital spread spectrum.

Note: All unused inputs must be driven Low. The program will automatically tie the inputs Low if they are unused.

Clock Delay Locked Loop (DLL)

DCM includes a clock delay locked loop used to minimize clock skew for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X devices. DCM synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specified time (ps) of each other.

DCM supports two frequency modes for the DLL. By default, the DLL_FREQUENCY_MODE attribute is set to Low and the frequency of the clock signal at the CLKIN input must be in the Low (DLL_CLKIN_MIN) to High (DLL_CLKIN_MAX) range.

Рис. 36.3 - Копія сторінки довідника бібліотечних елементів ПЛІС Xilinx (DCM)

Перша копія містить початок детального опису елемента, а друга копія подає вірєць залучення цього елемента до VHDL моделі проекту на ПЛІС.

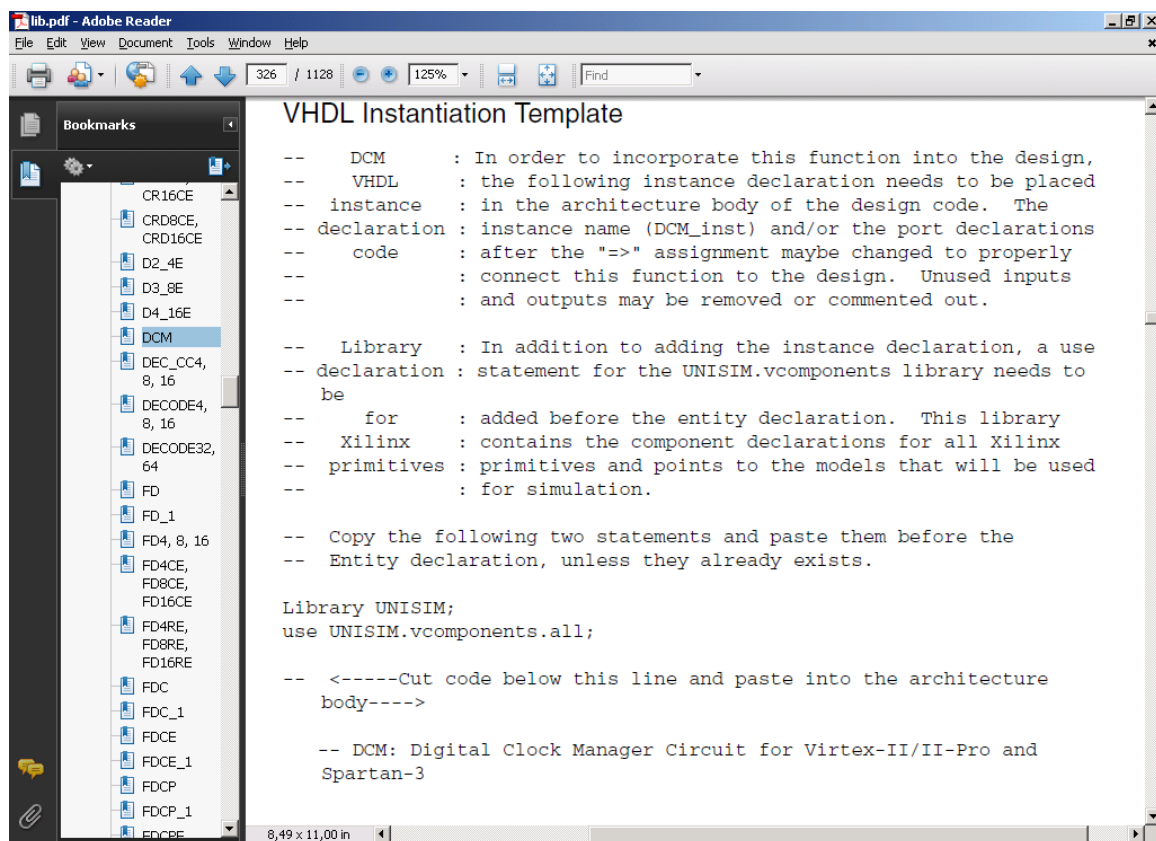


Рис. 3б.4 - Копія іншої сторінки довідника бібліотечних елементів ПЛІС Xilinx (DCM). Взірцевий приклад використання DCM у VHDL моделі

Цей та інші довідники (електронні документи) надаються студентам безкоштовно. Бачимо, що сучасні САПР ефективно допомагають розробнику VHDL моделі, знімаючи з нього рутинну роботу щодо запам'ятовування стандартних фактів.

Апаратно залежні низькорівневі VHDL моделей досить розповсюджені. Їм завжди притаманне намагання повністю вичерпувати можливості апаратних засобів ПЛІС.

ЛР № 4. Імплементування та дослідження VHDL моделі машини з архітектурою MIPS

Мета: опанування технікою створення та використання софтверних контролерів з архітектурою MIPS.

Лабораторні роботи з номерами 4, 5, 6, 7 передбачають створення системних продуктів (софтверних контролерів та машин). Тому перед виконанням необхідно ознайомитися з архітектурою, форматами машинних інструкцій та форматами даних, а також з організацією відповідної машини або контролера. Ці матеріали не уведені до збірки лабораторних робіт, проте надаються студенту в інший спосіб. Ясно і те, що надважливо опанувати VHDL текстами моделі, над якою виконують лабораторні дослідження.

Завдання:

1. Завантажити до САПР наданий базовий варіант MIPS машини.
2. Зафіксувати тип цільової ПЛІС.
3. Виявити VHDL компонент, що містить тестову програму, та показати зв'язок поміж машинним кодом цієї програми та її асемблерним кодом. Визначити функцію тестової програми.
4. Пояснити ієрархію проектних VHDL файлів та зафіксувати функцію кожного з цих файлів.
5. Синтезувати наданий базовий варіант MIPS машини.
6. Верифікувати результат синтезу часовим симулюванням. Встановити зв'язок поміж динамікою теоретично виявленого порядку виконання тестової програми та трасами сигналів на відповідній симуляційній часовій діаграмі.
7. Проаналізувати автоматично створену RTL схему синтезованої MIPS машини.
8. Проаналізувати автоматично створені САПР звіти з виконання етапів синтезу, імплементування і програмування.
9. Проаналізувати фрагменти автоматично створеного САПР файла програмування ПЛІС.
10. Внести зміни до наданого базисного варіанту проекту (такі зміни до програми, які вимагають використання, наприклад, додавання порту введення (виведення) або реалізації ще одної машинної інструкції); змінити цільову ПЛІС.
11. Синтезувати власний проект (= базовий проект + зміни).
12. Експериментом визначити максимально досяжну частоту тактування машини.
13. Дослідити та пояснити поведінку та результати симуляції власного варіанту MIPS машини.

14. Скласти звіт з виконання лабораторної роботи і захистити його.

Допоміжний матеріал

Цей розділ містить інформацію, що полегшує виконання базової частини лабораторної роботи.

1. Організація одноциклової машини з архітектурою MIPS

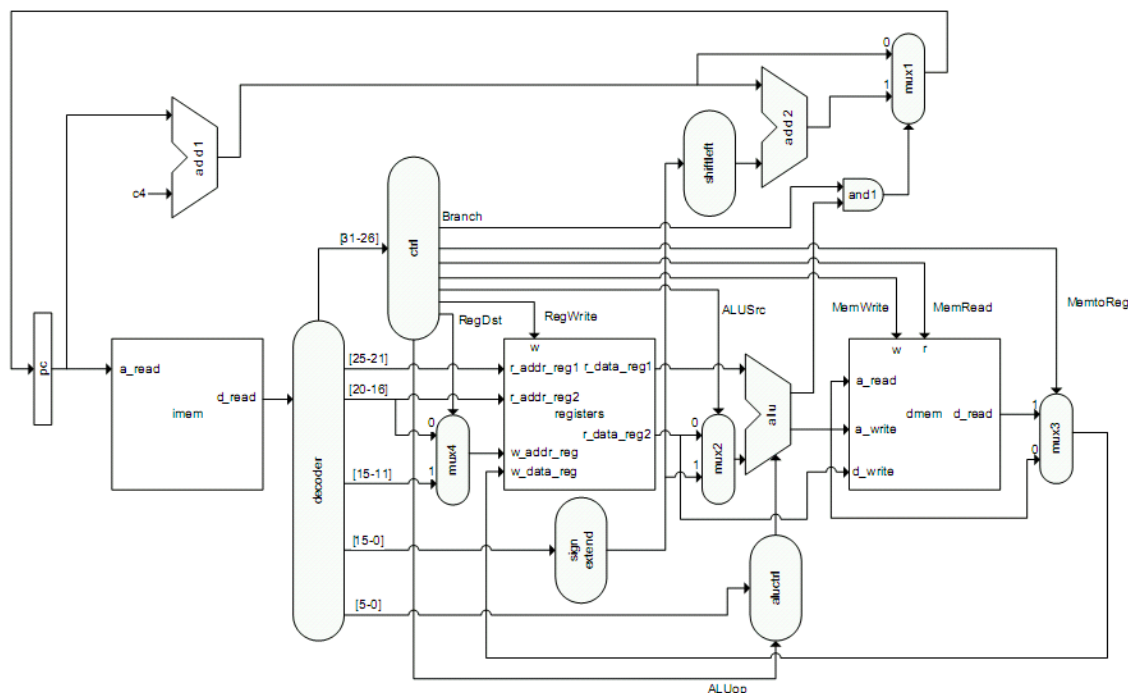


Рис. 4.1. Організація машини MIPS (гардварька архітектура)

В цілому подана рис. 4.1 структура є відомою з архітектурних дисциплін. В одноцикловому варіанті, що зараз розглядається, конвеєра нема. Тому реалізація структури у вигляді VHDL моделі значно спрощена. Додаткові ресурсні спрощення досягнуто скороченням довжини формату даних з 32 до 8 бітів, а це дозволяє вибирати цільову ПЛІС Virtex-II з невисоким рівнем інтеграції. Проте формат машинних інструкцій залишаємо незмінним, з довжиною 32 біти (аби не переробляти асемблер).

В одноцикловому варіанті кожним фронтом тактового імпульсу завершують виконання поточної машинної інструкції записом результату або до комірки регістрового файлу (registers), або до комірки пам'яті даних (dmem) + завжди оновлюють вміст програмного лічильника (pc). Останнє дозволяє вибирання цим самим фронтом наступної інструкції з пам'яті інструкцій (imem). Коли фронтом завершують виконання інструкції умовного переходу (branch), тоді, залежно від невиконання/виконання умови, оновленням програмного лічильника є або адреса наступної за чергою інструкції (поточна адреса + 4), або цільова адреса інструкції умовного переходу (поточна адреса + 4 + безпосередня константа інструкції переходу; проаналізуйте зв'язки на структурі

MIPS машини). Одноцикловий варіант вимагає комбінаційного пристрою керування (control), складність якого є низькою.

Виконання інструкції безумовного переходу не розглядаємо.

Подано сенс керуючих сигналів MIPS машини (мікрооперацій) :

RegDst – вибирає одну з двох можливих адрес комірок регістрового файлу, куди пишуть результат виконання машинної інструкції;

RegWrite – мікронаказ запису до комірки регістрового файлу;

ALUSrc – мікронаказ вибору одного з двох можливих «нижніх» операндів АЛП: або вмістиме комірки регістрового файлу, або безпосередній операнд;

ALUOp – мікронаказ, що разом з розрядами коду операції [5-0] задає операцію для АЛП;

MemWrite – мікронаказ запису до комірки пам'яті даних;

MemRead – мікронаказ читання з комірки пам'яті даних;

MemToReg – мікронаказ пропускання до регістрового файлу виходу пам'яті даних;

Branch – ознака того, що поточно виконуваною є машинна інструкція умовного переходу.

1. Завантаження VHDL проекту MIPS машини до САПР

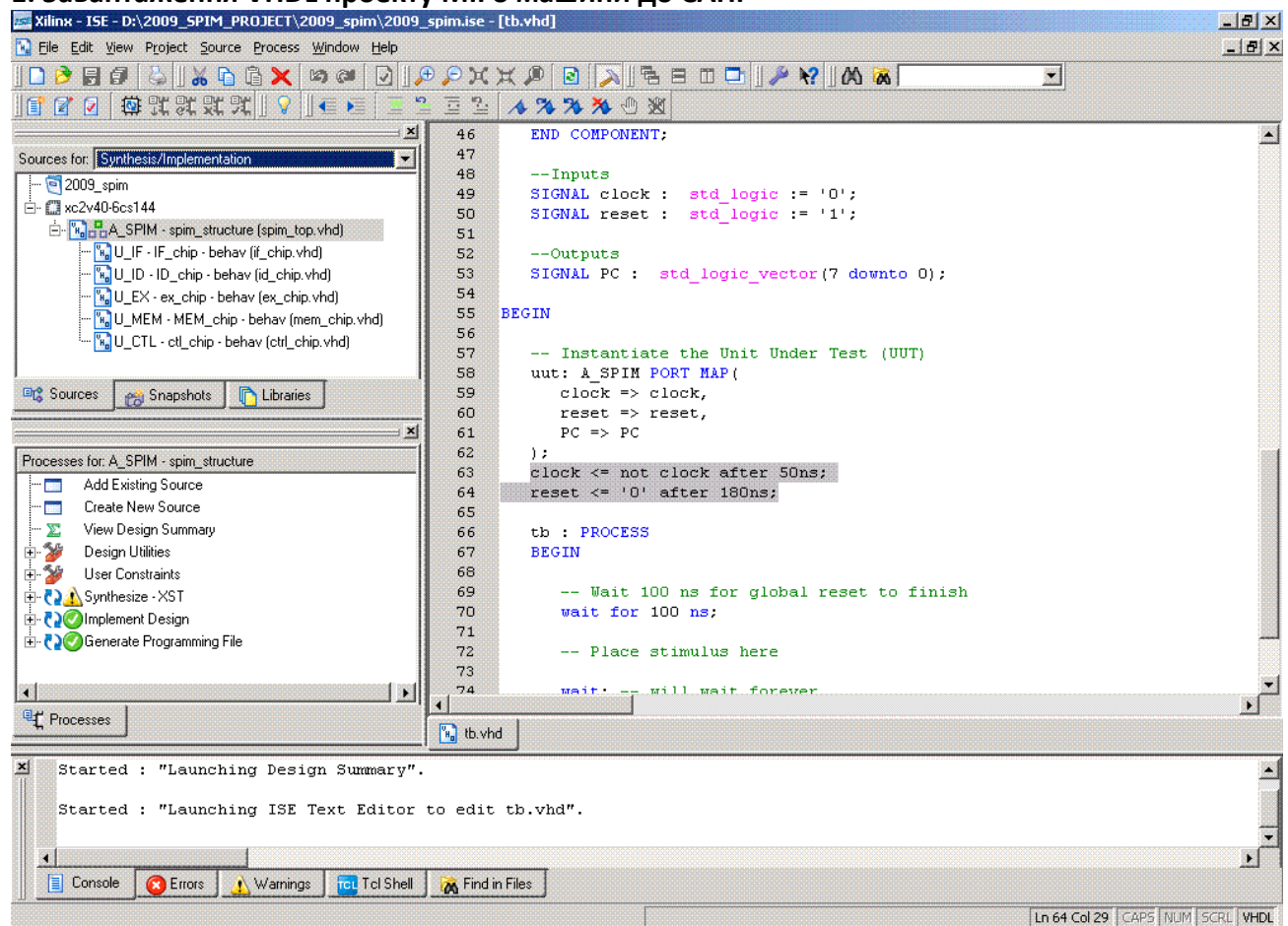


Рис. 4.2. Завантажений до WebPack VHDL проект MIPS машини

У лівому верхньому вікні рис. 4.2 подані проектні джерельні файли, що утворюють ієрархію проекту. В правому вікні (редактор текстів) подано фрагмент тестбенч файла, що містить автоматично задані нульові початкові значення вхідних сигналів (вручну перероблено на 1 для скиду). Темним фоном виділено два дописані вручну рядки, що задають зміни цих вхідних змінних, а саме: на 17 нс сигнал скиду приймає значення 0 і припиняє свою дію; через кожні 50 нс рівень такту інвертується, отже такт має період 100 нс, а частоту зміни $1/100\text{нс} = 10\text{ МГц}$.

2. Інтерфейс MIPS машини

Інтерфейс утворюють два вхідні сигнали такту і скидання + вихідний сигнал, що відтворює вмістиме програмного лічильника PC.

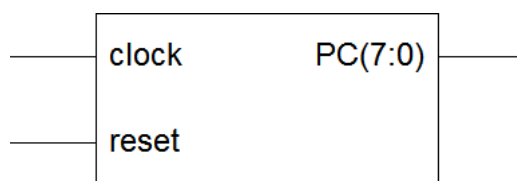


Рис. 4.3. Інтерфейсні сигнали MIPS машини

Призначення такту та скидання є ясным. Трасу сигналу PC спостерігають під час симуляційного виконання тестової програми, аби впевнитися, що ця траса відповідає наперед передбаченому (теоретично) порядку виконання програми. Цей інтерфейс автоматично креслить САПР на етапі синтезу проекту (див. опцію RTL schematic view).

3. Симуляційна часова діаграма виконання тестової програми

Софтмікроконтролер синтезують разом із тестовою програмою, що знаходиться і відокремленій програмній пам'яті (гарвардська архітектура машини). Цільова ПЛІС Virtex 2 містить примітив (фізичну структуру) так званої зблокованої (зосередженої) пам'яті, що і використовують як пам'ять програми. При цьому VHDL код не містить прямих вказівок на застосування саме цього примітиву пам'яті, але синтезатор XST розпізнає, тому і не синтезує "з нуля" програмну пам'ять. Адже це є недоцільним. Ефекту розпізнавання досягають коректною формою подання VHDL коду пам'яті. Приклад «коректного» VHDL коду пам'яті надає сама САПР WebPack.

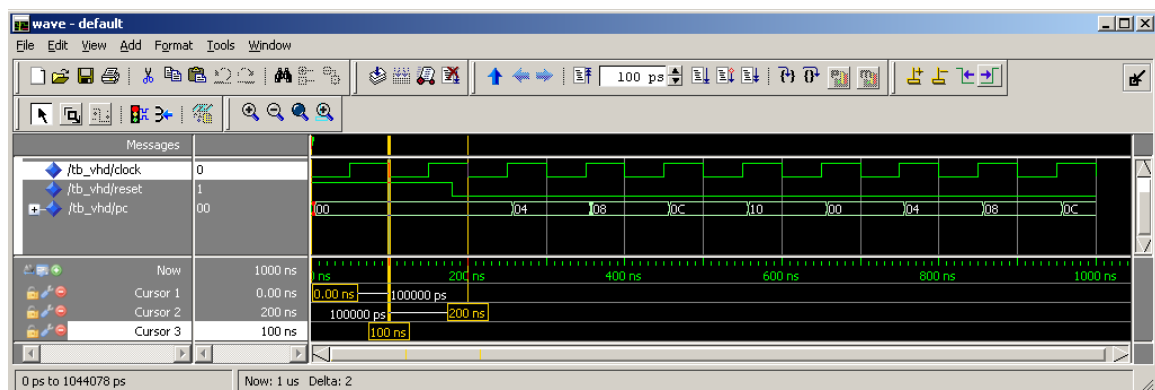


Рис. 4.4. Часова діаграма симуляційного виконання програмного коду в архітектурі Xilinx PicoBlaze

Першим (нагорі рис. 4.4) подано трасу зміни сигнала скиду, потім розташовано трасу такту. Далі подано трасу змін вмістимого програмного лічильника. Порівнюючи симуляційне вмістиме РС з теоретично передбаченим, а це вимагає аналізу тестової програми, доходимо висновку про коректне виконання програми базовим взірцем машини.

На діаграмі видно, що період такту складає 100 нс (10 МГц тактування).

4. Функційна (RTL) схема MIPS машини

Автоматично створену RTL схему MIPS машини містить рисунок 4.5.

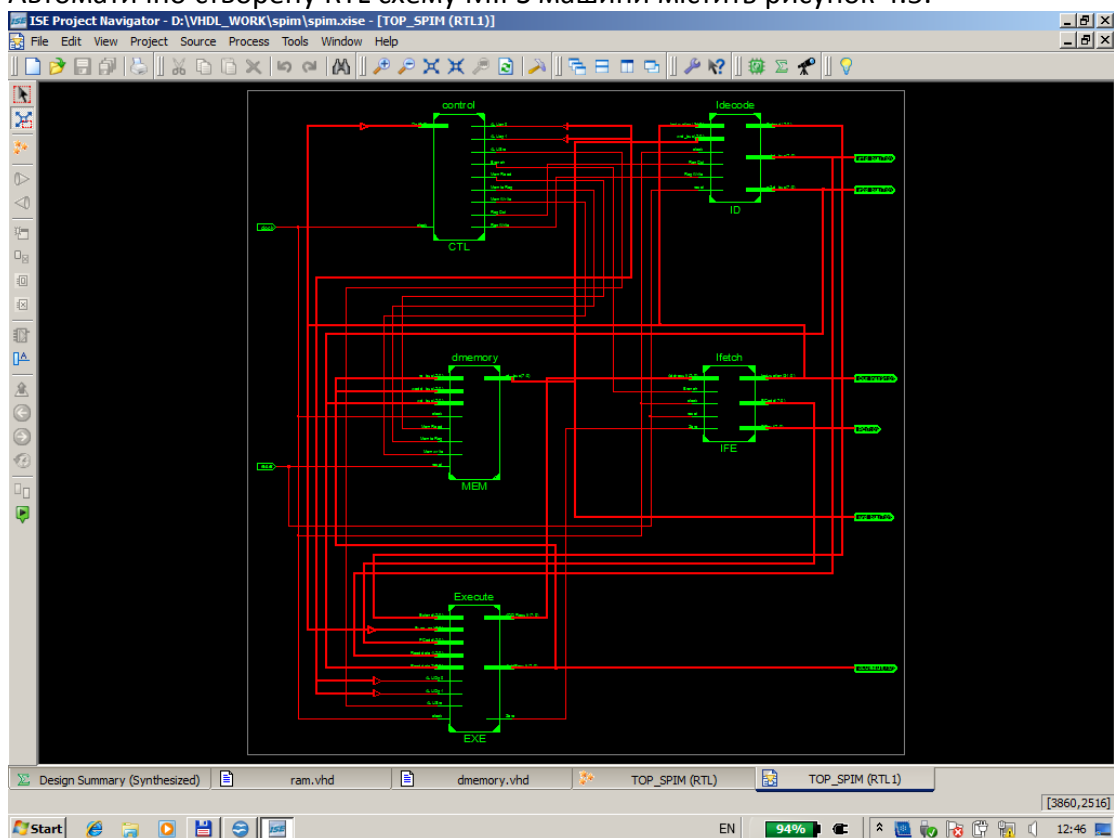


Рис. 4.5. Автоматично згенерована функційна (RTL) схема MIPS машини

Графіку цієї схеми можна перенести до Windows Paint. Тут в збільшеному варіанті легко розпізнати всі основні вузли, що містить VHDL модель, та перевірити коректність зв'язків поміж вузлами, користуючись VHDL кодом.

ЛР № 5. Імплементування та дослідження VHDL моделі софтверного контролера XILINX PicoBlaze

Мета: опанування технікою створення і використання 8 бітових софтверних контролерів з архітектурою Xilinx PicoBlaze.

Завдання:

1. Завантажити до САПР наданий базовий варіант VHDL моделі контролера.
2. Зафіксувати тип цільової ПЛІС.
3. Виявити VHDL компонент, що містить тестову програму, і показати зв'язок поміж машинним кодом цієї програми та її асемблерним кодом. Визначити функцію тестової програми.
4. Пояснити ієрархію проектних VHDL файлів та зафіксувати функцію кожного з цих файлів.
5. Синтезувати наданий базовий варіант софтверного контролера.
6. Верифікувати результат синтезу часовим симулюванням. Встановити зв'язок поміж динамікою теоретично виявленим порядком виконання тестової програми і трасами сигналів на симуляційній часовій діаграмі.
7. Проаналізувати автоматично створені RTL і технологічну схеми синтезованого контролера.
8. Проаналізувати автоматично створені САПР звіти з виконання етапів синтезу, імплементування і програмування.
9. Проаналізувати фрагменти створеного автоматично САПР файлу програмування ПЛІС.
10. Внести зміни до наданого базисного варіанту проекту (такі зміни до програми, що вимагають використання ще і порту введення; змінити цільову ПЛІС).
11. Синтезувати власний проект (= базовий проект + зміни).
12. Дослідити і пояснити поведінку софтверного контролера симуляцією власного варіанту.
13. Скласти звіт з виконання лабораторної роботи і захистити його.

Допоміжний матеріал

Цей розділ містить інформацію, що полегшує виконання базової частини лабораторної роботи.

1. Тестова програма для PicoBlaze

```
Load s7, 01          ; init shifter reg
Output s7, 04
SL0 s7                ; rotate left
Jump NZ, 01
```

Jump 00

2. VHDL модель програмної пам'яті з тестовою програмою

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity demo_test is
    port( address : in std_logic_vector(7 downto 0);
          clk : in std_logic;
          dout : out std_logic_vector(15 downto 0));
end;

architecture v1 of demo_test is

    constant ROM_WIDTH: INTEGER:= 16;
    constant ROM_LENGTH: INTEGER:= 256;

    subtype rom_word is std_logic_vector(ROM_WIDTH-1 downto 0);
    type rom_table is array (0 to ROM_LENGTH-1) of rom_word;

    constant rom: rom_table := rom_table'(
        "0000011100000001",
        "1000111100000100",
        "1010011100000110",
        "1101010100000001",
        "1101000000000000",
        "0000000000000000",
        "0000000000000000",
        .
        .
        .
        "0000000000000000",
        "0000000000000000");

begin
    process (clk)
    begin
        if clk'event and clk = '1' then
            dout <= rom(conv_integer(address));
        end if;
    end process;
end v1;

```

4. Завантаження VHDL проекту до САПР

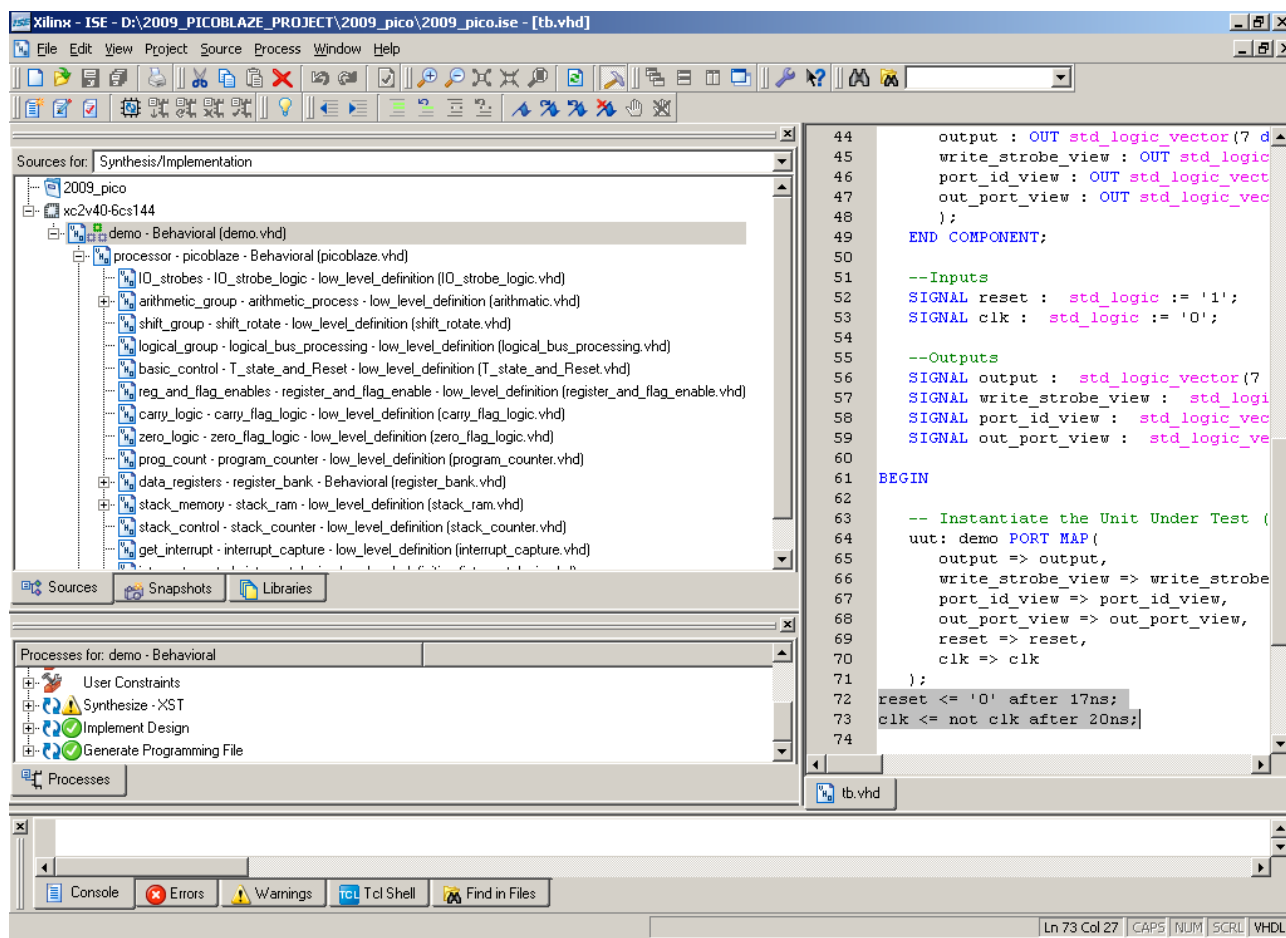


Рис. 5.1. Завантажений до WebPack VHDL проект софтверного контролера Xilinx PicoBlaze

У лівому верхньому вікні рис. 5.1 подані проектні джерельні файли, що утворюють ієрархію проекту. В правому вікні редактора текстів подано фрагмент тестбенч файлу, що містить автоматично задані нульові початкові значення вхідних сигналів (вручну задано 1 для скиду). Темним фоном виділено два рядки, що записані вручну і задають бажані зміни цих вхідних змінних, а саме: на 17 нс сигнал скиду приймає значення 0 і припиняє свою дію; через кожні 20 нс рівень такту інвертується; отже такт має період 40 нс, відповідно частоту зміни $1/40\text{нс} = 25\text{ МГц}$.

Лабораторні роботи з номерами 4, 5, 6, 7 передбачають створення системних продуктів (софтверних контролерів та машин). Тому перед виконанням необхідно ознайомитися з архітектурою, форматами машинних інструкцій і форматами даних, а також з організацією відповідної машини або контролера. Ці матеріали не уведено до збірки лабораторних робіт, але надаються студенту іншими документами. Ясно і те, що надважливо

розуміти VHDL тексти моделі, з якою виконуються лабораторні дослідження.

5. Інтерфейс софтверного контролера

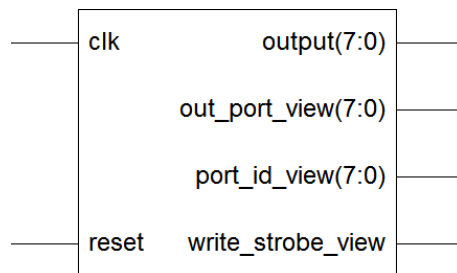


Рис. 5.2. Інтерфейсні сигнали софтверного контролера з архітектурою Xilinx PicoBlaze

Є (рис. 5.2) два вхідні сигнали (clk, reset) та чотири вихідні сигнали (output + три сигнали для контрольного спостереження на симуляційній часовій діаграмі: out_port_view – інформація, що виводиться на порт, port_id_view – номер порту, write_strobe_view – звернення до порту запису). СENS шойно занотованих сигналів виявляють аналізом VHDL кодів проекту. Цей інтерфейс автоматично створює САПР на етапі синтезу проекту (див. опцію RTL schematic view).

6. Симуляційна часова діаграма виконання тестової програми

Софтверний контролер синтезують разом із тестовою програмою, що знаходиться в окремій програмній пам'яті. Цільова ПЛІС Virtex 2 містить примітив (фізичну існуючу структуру) зблокованої пам'яті, що і використовується як пам'ять програми. При цьому VHDL модель не містить прямих вказівок на застосування саме цього примітиву. Разом з цим САПР не синтезує «з нуля» програмну пам'ять, що є недоцільним. Ефект досягають коректною формою запису VHDL коду цієї пам'яті. Приклад «правильного» коду пам'яті містить безпосередньо САПР WebPack.

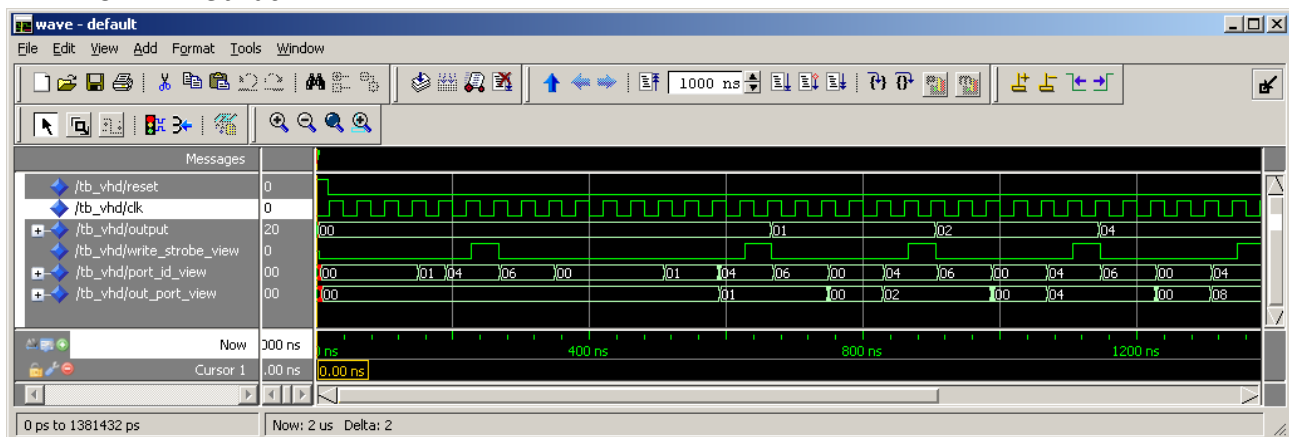


Рис. 5.3. Часова діаграма симуляційного виконання програмного коду в архітектурі Xilinx PicoBlaze

Першим нагорі (рис. 5.3) подано трасу зміни сигналу скиду, потім розташовано трасу такту, далі йде траса вихідного сигналу, що інкрементується кожного разу

з виконанням запису до порту 4. Нижче розташовані траси нгомера порту і сигналу, що видається на порт виводу. Видно, що період такту складає 40 нс.

7. Функційна (RTL) схема софтверола

RTL схему софтверола з архітектурою Xilinx PicoBlaze містить рисунок 5.4.

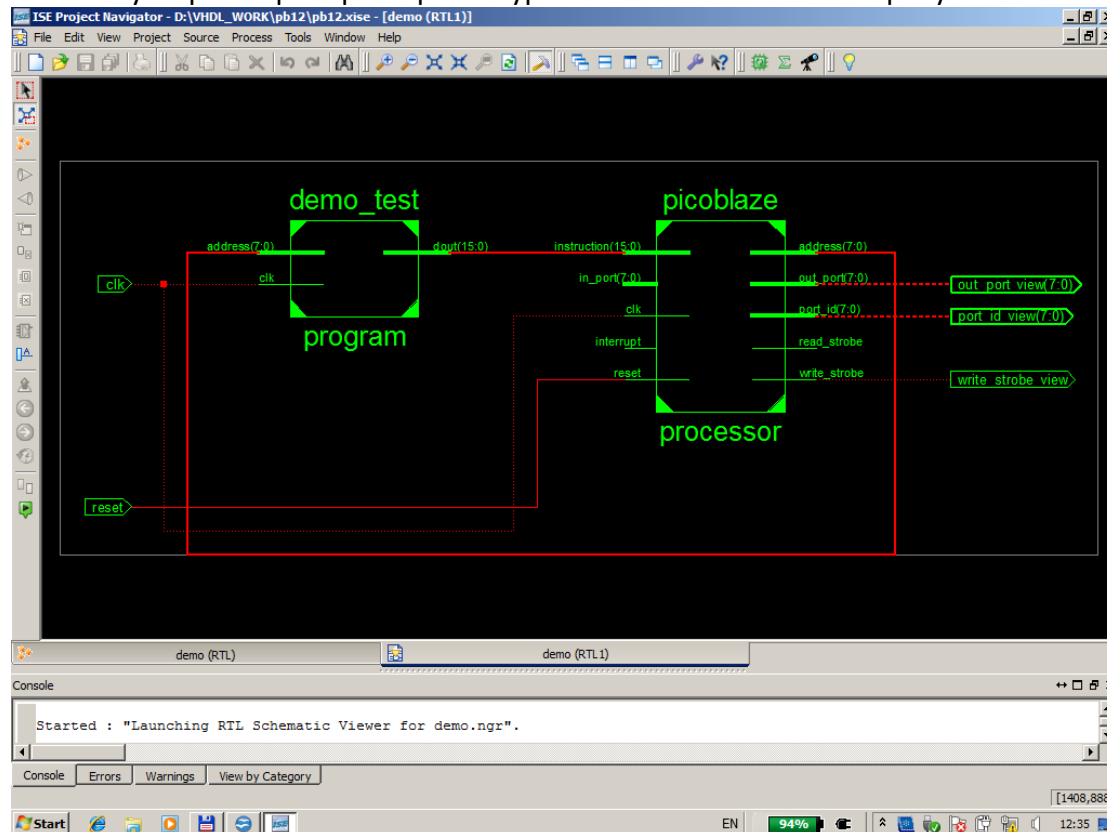


Рис. 5.4. Автоматично створена функційна (RTL) схема софтверола з архітектурою PicoBlaze

Надолі розташовані програмна пам'ять і софтвероцесор з архітектурою PicoBlaze. На горі міститься порт виводу (фактично вісім D – тригерів з дозволом тактування CE; узагальна назва кожного тригера – FDE тригер; він є примітивом, тобто, реальним елементом і не синтезується).

ЛР № 6. Імплементування та дослідження VHDL моделі софтверного контролера XESS Gnome

Мета: опанування технікою створення і використання 4 бітового софтверного контролера з архітектурою XESS Gnome.

Завдання:

1. Завантажити до САПР наданий базовий варіант VHDL моделі контролера.
2. Зафіксувати тип цільової ПЛІС.
3. Виявити VHDL компонент, що містить тестову програму, і показати зв'язок поміж машинним кодом цієї програми та її асемблерним кодом. Визначити функцію тестової програми.
4. Пояснити ієрархію проектних VHDL файлів та зафіксувати функцію кожного з цих файлів.
5. Синтезувати наданий базовий варіант софтверного контролера.
6. Верифікувати результат синтезу часовим симулюванням. Встановити зв'язок поміж динамікою теоретично виявленим порядком виконання тестової програми і трасами сигналів на симуляційній часовій діаграмі.
7. Проаналізувати автоматично створені RTL і технологічну схеми синтезованого контролера.
8. Проаналізувати автоматично створені САПР звіти з виконання етапів синтезу, імплементування і програмування.
9. Проаналізувати фрагменти створеного автоматично САПР файлу програмування ПЛІС.
10. Внести зміни до наданого базисного варіанту проекту (такі зміни до програми, що вимагають використання ще і порту введення; змінити цільову ПЛІС).
11. Синтезувати власний проект (= базовий проект + зміни).
14. Дослідити і пояснити поведінку софтверного контролера симуляцією власного варіанту.
15. Скласти звіт з виконання лабораторної роботи і захистити його.

Допоміжний матеріал

Цей розділ містить інформацію, що полегшує виконання базової частини лабораторної роботи.

1. Завантаження VHDL проекту XESS Gnome до САПР

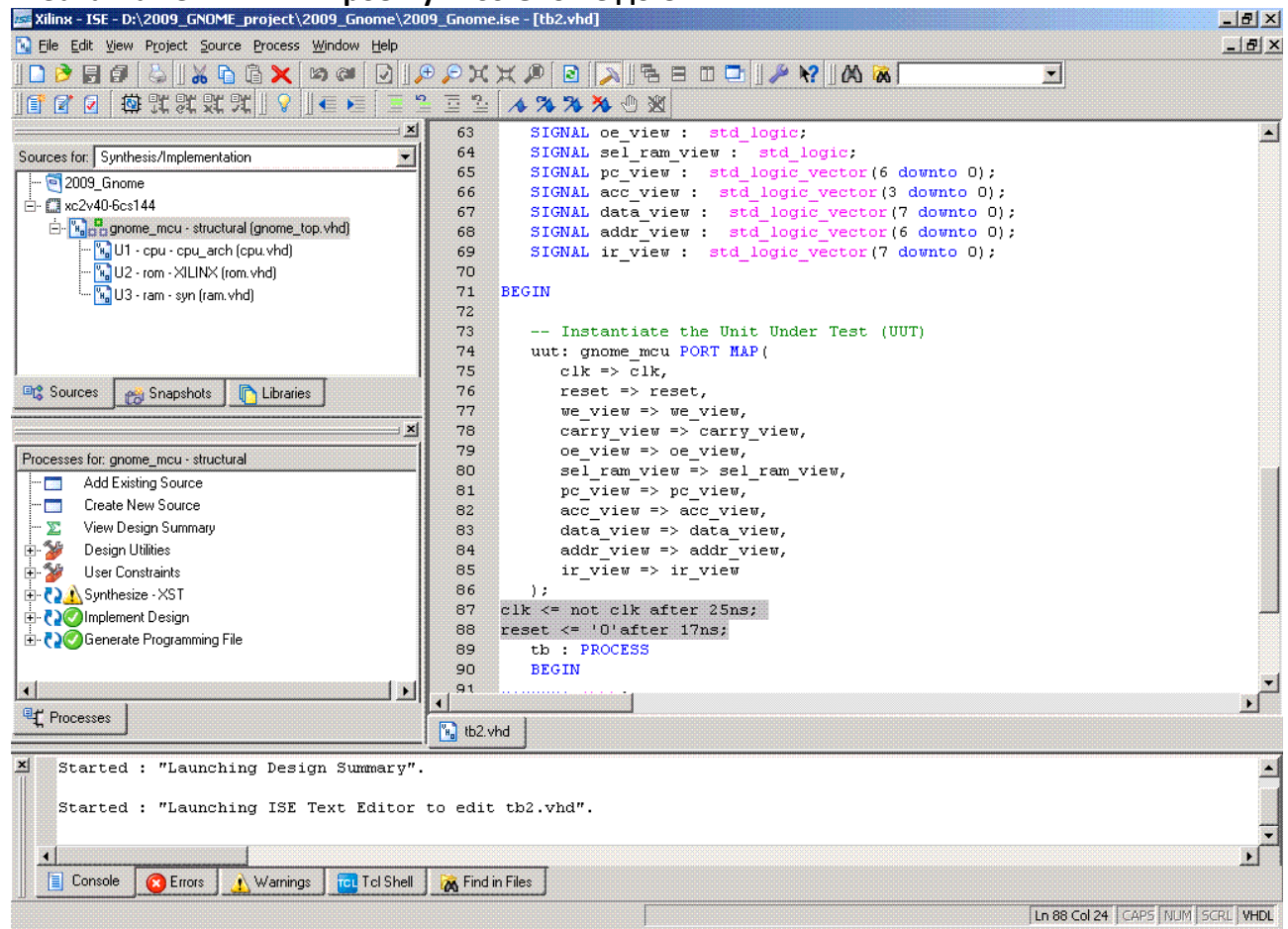


Рис. 6.1. Завантажений до WebPack VHDL проект софтверного контролера XESS Gnome

В лівому верхньому вікні рис. 6.1 подані проектні джерельні файли, що утворюють ієрархію проекту. В правому вікні редактора текстів подано фрагмент тестбенч файлу, що містить автоматично задані нульові початкові значення вхідних сигналів (вручну задано 1 для скиду). Темним фоном виділено два рядки, що записані вручну і задають бажані зміни цих вхідних змінних, а саме: на 17 нс сигнал скиду приймає значення 0 і припиняє свою дію; через кожні 25 нс рівень такту інвертується; отже такт має період 50 нс, відповідно частоту зміни $1/50\text{нс} = 20\text{ МГц}$.

2. Інтерфейс софтверного контролера

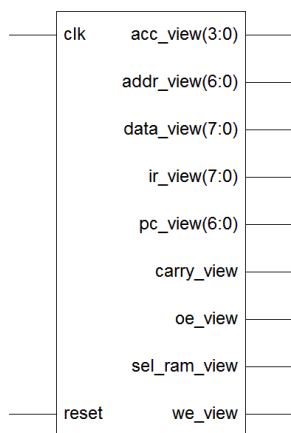


Рис. 6.2. Інтерфейсні сигнали софтверного контролера з архітектурою Xilinx PicoBlaze

Праворуч розташовані два вхідні сигнали (clk, reset). Реалізовано не весь софтверний контролер, а лише його ядро. Підсистеми введення/виведення немає. Її має додати студент. Всі вихідні сигнали розташовано праворуч. Присутність в назві сигналу слова view свідчить про те, що це є сигнал, трасу якого тз метою налаштування можна побачити на симуляційній часовій діаграмі. Перелічимо вихідні сигнали:

acc_view – вихід акумулятора;

addr_view – шина адреси;

data_view – шина даних;

ir_view – вихід регістра інструкцій;

pc_view – вихід програмного лічильника;

carry_view – сигнал переносу;

sel_ram_view – сигнал вибору пам'яті даних (регістрового файлу);

we_view – сигнал дозволу запису до пам'яті даних.

Лабораторні роботи з номерами 4, 5, 6, 7 та 8 передбачають створення системних продуктів (софтверних контролерів і машин). Тому перед виконанням необхідно ознайомитися з архітектурою, форматами машинних інструкцій і форматами даних, а також з організацією відповідної машини або контролера. Ці матеріали не уведені до збірки лабораторних робіт, але надаються студенту іншими документами. Ясно і те, що надважливо розуміти VHDL тексти моделі, з якою виконуються лабораторні дослідження.

3. Симуляційна часова діаграма виконання тестової програми

Софтверний контролер синтезують разом із тестовою програмою, що знаходиться в окремій програмній пам'яті. Цільова ПЛІС Virtex 2 містить примітив (фізичну

існуючу структуру) заблокованої пам'яті, що і використовується як пам'ять програми. При цьому VHDL модель не містить прямих вказівок на застосування саме цього примітиву. Разом з цим САПР не синтезує «з нуля» програмну пам'ять, що є недоцільним. Ефект досягають коректною формою запису VHDL коду цієї пам'яті. Приклад «правильного» коду пам'яті містить безпосередньо САПР WebPack.

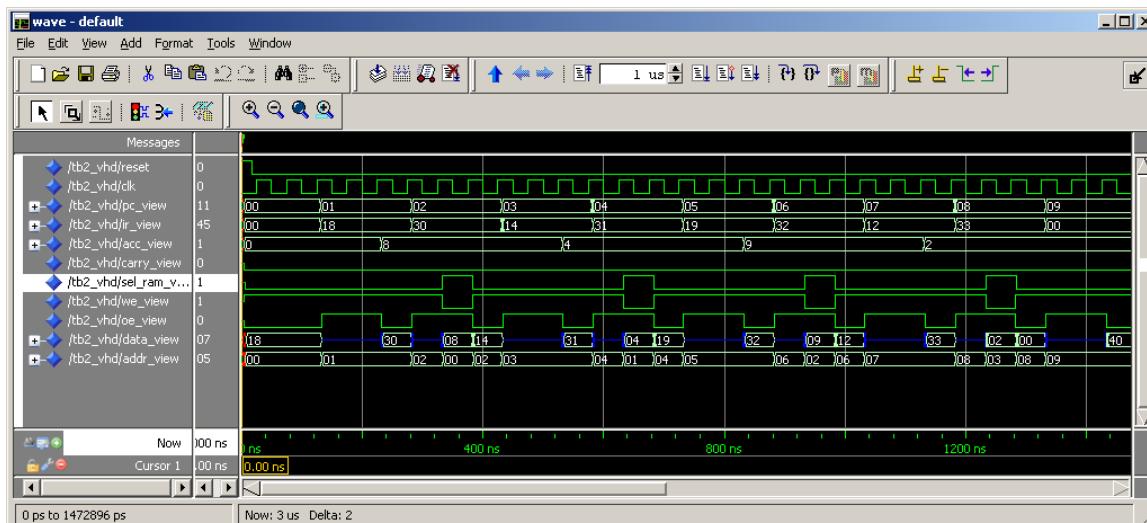


Рис. 6.3.А Симуляційна діаграма виконання тестової програми ядром Gnome (початок трас)

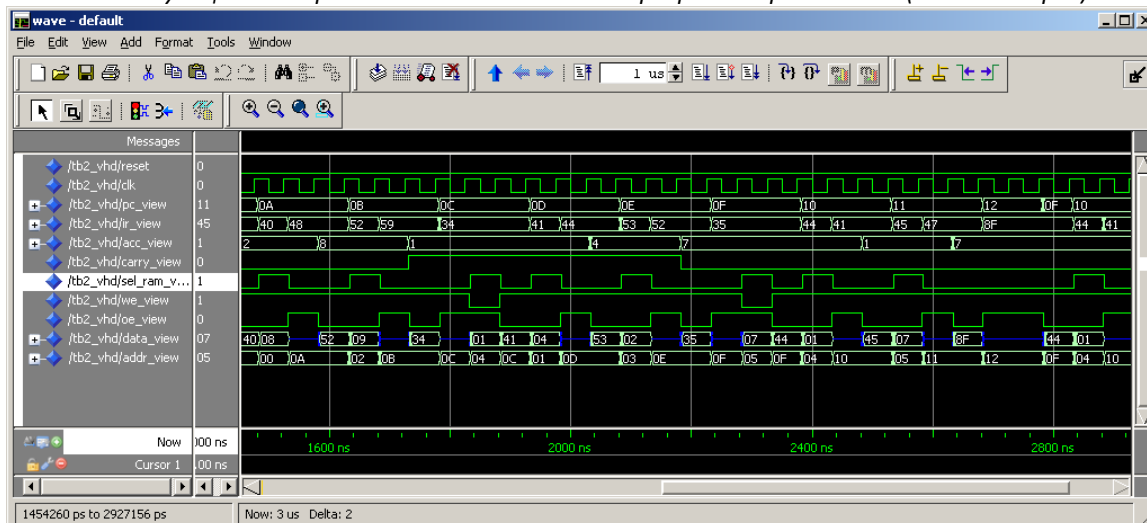


Рис. 6.3.Б Симуляційна діаграма виконання тестової програми ядром Gnome (продовження трас)

Першою (нагорі рис. 6.3) подано трасу зміни сигнала скиду, потім розташовано трасу такту. Далі йдуть траси всіх вихідних сигналів, що присутні в інтерфейсі ядра. Видно, що період такту складає 50 нс, частота тактування дорівнює 20МГц.

4. Функційна (RTL) схема софтверного контролера

RTL схему ядра софтверного контролера з архітектурою XESS Gnome містить рисунок 6.4.

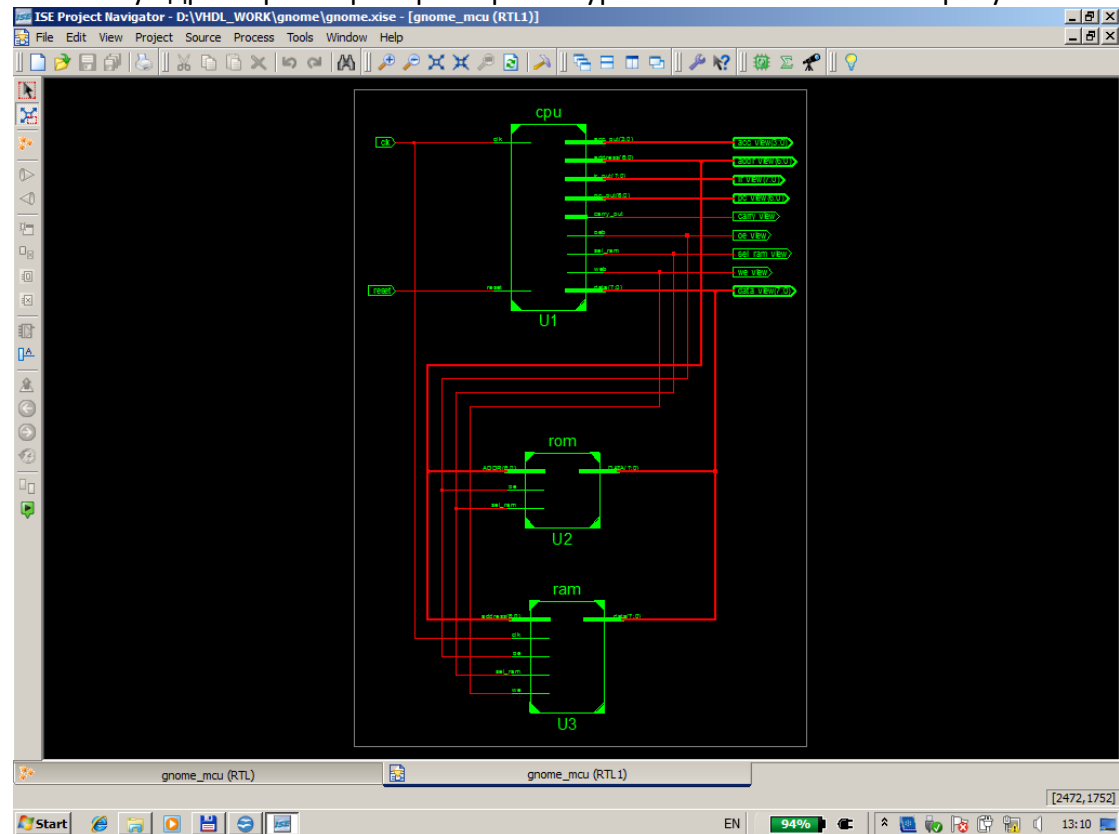


Рис. 6.4. Функційна (RTL) схема софтверного контролера з архітектурою XESS Gnome

Знизу розташовані пам'ять даних та програмна пам'ять, нагорі – софтверний процесор з архітектурою XESS Gnome.

ЛР № 7. Імплементація та дослідження софтверного контролера з шиною *wishbone*

Мета: опанування технікою створення і використання машин на основі стандартної системної шини *wishbone*

Завдання:

В САПР WebPack/ModelSim імплементувати до ПЛІС Virtex-II проект софтверного контролера «Micro8», побудований на системній шині *wishbone*. Результат проектування верифікувати методом часової симуляції. Необхідними змінами в VHDL коді моделі забезпечити візуалізацію с подальшим аналізом часових діаграм циклів, що відбуваються на шині *wishbone*. Скласти звіт з проведених лабораторних досліджень і захистити його.

Основні відомості і варіант виконання лабораторної роботи

Передусім розглянемо систему машинних інструкцій того варіанту софтверного контролера Micro8, що розглядається в лабораторній роботі. Зауважимо, що відомі декілька модифікацій софтверного контролера.

Перший варіант софтверного контролера мав наступні інструкції: **ADD** (додавання), **NOR** (логічного додавання з запереченням), **STA** та **JCC** (Jump on Carry, збереження акумулятора в пам'яті і умовний перехід за ознакою переносу). It had a single carry bit which was reset by the **JCC** instruction. При цьому більш складні машинні інструкції інших контролерів вдається запрограмувати кодами софтверного контролера Micro8.

Зараз всі машинні інструкції мають наступний двохбайтовий формат.

B7	B6	B5	B4	B3	B2	B1	B0
регістр	інструкція		Режим адресування		адреса high / зсув high		

B7	B6	B5	B4	B3	B2	B1	B0
адреса low / зсув low / 8 бітів безпосереднього операнда							

Рис. 7.1 – Двохбайтовий формат інструкції процесора контролера Micro8
Старшу частину адреси (high) формують біти B2, B1 та B0 першого байту формату машинної інструкції of the opcode, а молодші біти (low) адреси або безпосереднє значення (immediate value) задають біти від B7 до B0 другого байту формату. Отже, можна адресувати $2^{(3+8=11)} \times 1 \text{ byte} = 2 \text{ KB}$ пам'яті, тоді як первинний варіант дозволяв адресування (шістьма бітами однобайтового формату інструкції) лише $2^6 = 64$ байти пам'яті. До першого варіанту внесено наступні зміни:

- Додано 8 бітовий індексний регістр.
- Bit 7 визначає регістр, а саме: акумулятор **A** (коли B7=0) або індексний регістр **X** (коли B7=1).

- Чотири первинні інструкції залишилися без змін:

Біти B6.B5	Тлумачення	
00	ADD	Додати пам'ять до регістра
01	NOR	<i>nor</i> регістр з пам'яттю
10	STO	Зберегти регістр в пам'яті
11	Bcc	Перехід за умовою cc

- Додані чотири режими адресування (спочатку існував лише один режим абсолютної адреси):

Біти B4.B3	Тлумачення	
00	I	Безпосередня (8 біт значення)
01	A	Абсолютная (11 біт зсуву)
10	X	Індексная (11 біт зсуву)
11	P	Відносно PC (11 біт зсуву)

- Додано 8 інструкцій умовного переходу (відносно програмного лічильника PC)

In Tim's computer, there was only one carry bit which was reset by the jump. This was so that subsequent jump instructions were always taken. The jump instruction also used absolute addressing. On my version, only ADD, NOR and STO affect the condition codes, not the branches. The branch instructions on my computer have a 11 bit displacements so can jump anywhere in the address range. It would be nice to have an absolute JMP instruction, to jump to fixed locations but there is not enough room in the opcode map.

Біти B7.B6.B5.B4.B3	Мнемокод	Тлумачення інструкцій умовного переходу
01100	BRA	Branch Always
01101	BEQ	Branch if Zero flag set
01110	BCS	Branch carry flag set
01111	BMI	Branch if Negative flag set (Negative)
11100	BRN	Branch never
11101	BNE	Branch never 11110 - BCC -
11110	BCC	Branch carry clear
11111	BPL	Branch if Negative flag Clear (Positive)

Макроси на основі машинних інструкцій

Очищення регістру:

NORAI #FF

Завантаження регістру з пам'яті:

NORAI #FF

ADDAA address

Інвертування:
NORAI #00

Тестування вмістимого (наприклад) біту з номером 0:

NORAI #FE ; *set all other bits and invert*

BEQ branch_address ; *test for opposite state*

I have not tested all the instructions but the code prints out "Hello World" on the miniuart and waits for a character to be entered. Note that the system was tested with a 4.915 MHz clock, so you either have to set the jumpers of the ICST525-01 accordingly or hack the miniUART code. The Wishbone MiniUart operates at 9600 baud given a 4.915 MHz clock input. The default baudrate divider was 130 for a 5MHz clock, however I have modified it to 128 for a 4.915 MHz clock.

Далі ми розглянемо тестову програму, що виконує Micro8. Подамо текст програми, що «зашита» до програмної пам'яті мікроконтролера Micro8 та імплементується разом з мікроконтролером:

```
-- FILE NAME: bootrom.vhdl
-- ENTITY NAME: boot_rom
-- ARCHITECTURE NAME: behave
-- REVISION: A
--
-- DESCRIPTION: 64 byte x 8 bit ROM to down a Monitor
-- program on reset
--
--Written by John Kent for the micro8 processor
```

library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;

entity boot_rom is

port (

WB_ADR_I : in std_logic_vector(5 **downto** 0);

WB_DAT_O : out std_logic_vector(7 **downto** 0);

WB_STB_I : in std_logic;

WB_ACK_O : out std_logic

);

end entity boot_rom;

architecture basic **of** boot_rom is

constant width : integer := 8;

constant memsize : integer := 64;

type rom_array is **array**(0 to memsize-1) **of** std_logic_vector(width-1 **downto** 0);

constant rom_data : rom_array :=

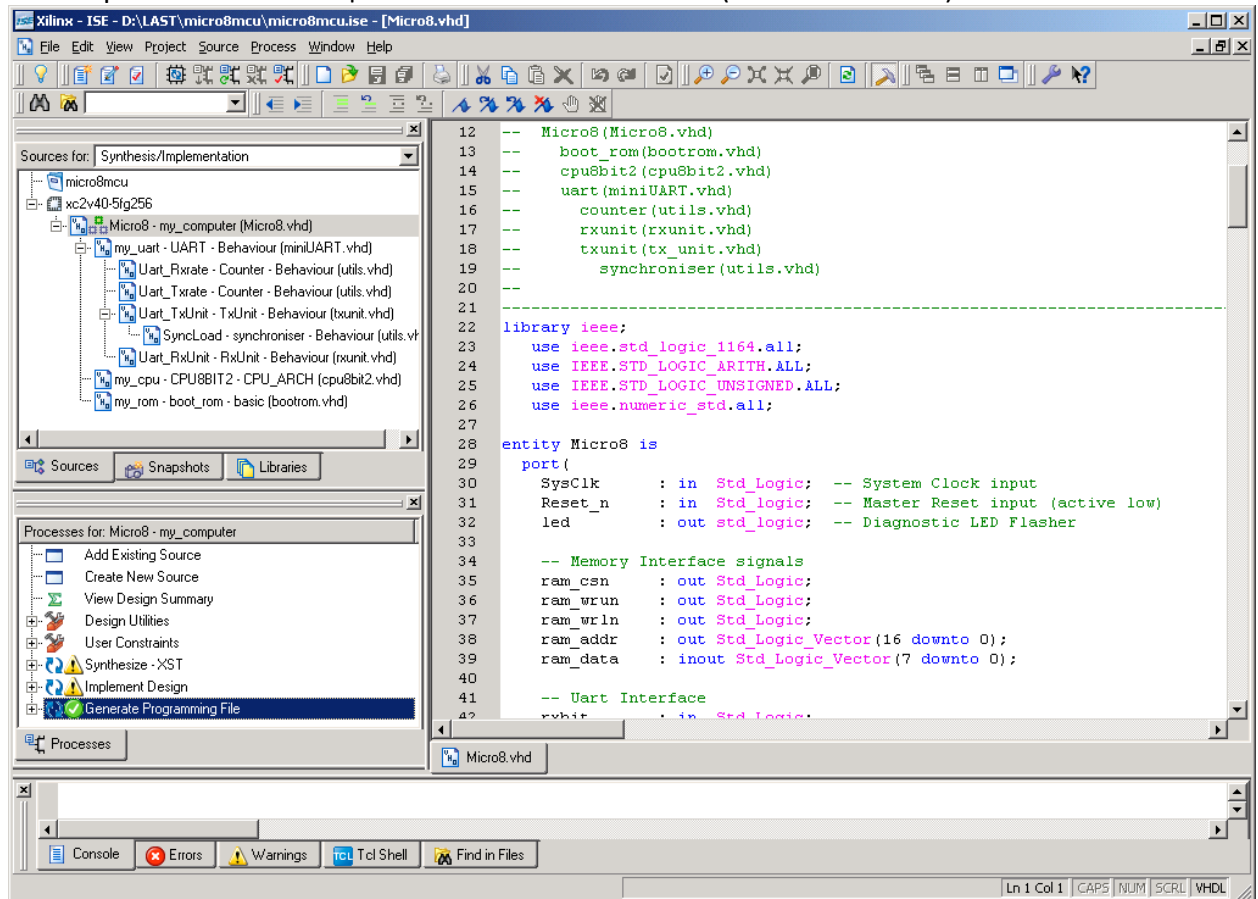
```

( "10100000", "11111111",      -- 0000 - a0 ff  RESET NORX #$FF
  "00100000", "11111111",      -- 0002 - 20 ff  POLL1 NORA #$FF
  "00001111", "11111101",      -- 0004 - 0f fd  ADDA UARTCR
  "00100000", "11111110",      -- 0006 - 20 fe  NORA #not(TXBE)
  "11101111", "11111000",      -- 0008 - ef f8  BNE POLL1
  "00100000", "11111111",      -- 000a - 20 ff  NORA #$FF
  "00010000", "00110000",      -- 000c - 10 30  ADDA MSG,X
  "01101000", "00000110",      -- 000e - 68 06  BEQ POLL2
  "10000000", "00000001",      -- 0010 - 80 01  ADDX #1
  "01001111", "11111100",      -- 0012 - 4f fc  STA UARTDR
  "11101111", "11101100",      -- 0014 - ef ec  BNE POLL1
  "00100000", "11111111",      -- 0016 - 20 ff  POLL2 NORA #$FF
  "00001111", "11111101",      -- 0018 - 0f fd  ADDA UARTCR
  "00100000", "11111101",      -- 001a - 20 fd  NORA #not(RXBF)
  "11101111", "11111000",      -- 001c - ef f8  BNE POLL2
  "00100000", "11111111",      -- 001e - 20 ff  NORA #$FF
  "00001111", "11111100",      -- 0020 - 0f fc  ADDA UARTDR
  "01100111", "11011100",      -- 0022 - 67 dc  BRA RESET
  "00000000", "00000000",      -- 0024 - 00 00  fcb $00,$00
  "00000000", "00000000",      -- 0026 - 00 00  fcb $00,$00
  "00000000", "00000000",      -- 0028 - 00 00  fcb $00,$00
  "00000000", "00000000",      -- 002a - 00 00  fcb $00,$00
  "00000000", "00000000",      -- 002c - 00 00  fcb $00,$00
  "00000000", "00000000",      -- 002e - 00 00  fcb $00,$00
  "01001000", "01100101", "01101100", -- 0030 - 48 65 6c MSG FCC "Hel"
  "01101100", "01101111", "00100000", -- 0033 - 6c 6f 20 FCC "lo "
  "01010111", "01101111", "01110010", -- 0036 - 57 6f 72 FCC "Wor"
  "01101100", "01100100",      -- 0039 - 6c 64 FCC "ld"
  "00001010", "00001101", "00000000", -- 003b - 0a 0d 00 FCB LF,CR,NULL
  "00000000", "00000000",      -- 003e - 00 00 fcb null,null
);
begin
  WB_DAT_O <= rom_data(conv_integer(WB_ADR_I));
  WB_ACK_O <= WB_STB_I;
end architecture basic;

```

Проект Micro8 містить асинхронний приймач-передавач UART, що дозволяє в нашому випадку виводити з мікроконтролера послідовним асинхронним кодом повідомлення Hello World. **Rx** позначає приймач UART, на вхід якого ми в симуляції задамо рівень rxbit = '1' (нема сигналу, тобто, пасивний стан входу). Послідовний код повідомлення hello World спостерігає на бітовому виході передавача **Tx**, який позначено через txbit. VHDL модель універсального асинхронного приймача-передавача містять файли miniUART.vhd (а це є локальний топ-файл вкладеного проекту), txunit.vhd, rxunit.vhd та utils.vhd. VHDL

модель процесора містить файл `cpu8bit2.vhd`, а програму, що виконує процесор, містить файл `bootrom.vhd`. Зараз подамо копію вікна навігатора із завантаженням VHDL проектом `Micro8`. Цільова ПЛІС – Xilinx Virtex2 (40 тис. вентилів).



1. Рис. 7.2 - Копія вікна навігатора з завантаженням проектом `Micro8`

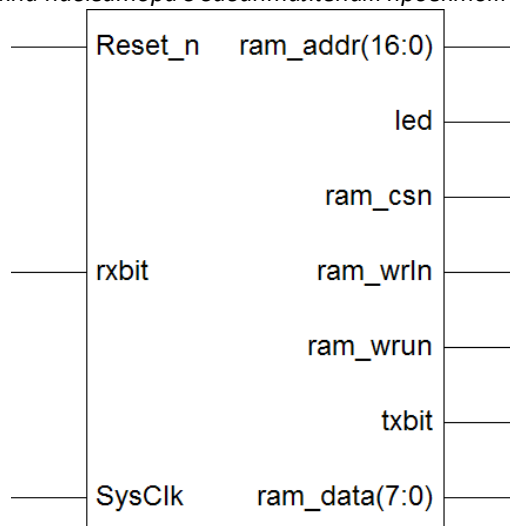


Рис. 7.3 - Інтерфейс мікроконтролера `Micro8`

В симуляції на вхід `rxbit` треба подати одиницю, на вхід `Reset_n` (низькоактивний скид) на короткий час (декілька періодів тактових імпульсів, але не на ціле число їхнє число) треба подати нуль. На вхід `SysClk` треба подати

тактові імпульси (в нас – частотою 10 МГц, період 100 нс). Вихідні сигнали треба спостерігати (на часових симуляційних діаграмах).

Маємо наступні вихідні сигнали:

- **ram_address(16 : 0)** є адресою зовнішньої щодо мікроконтролера пам'яті,
- **led** є вхідним сигналом зовнішнього світлодіода, що розташований на прототипній платі, засобами якою емулюють проект (в нас прототипної плати нема, сигнал можна не спостерігати),
- **ram_cs** є сигналом вибору кристала зовнішньої пам'яті,
- **ram_wrin**
- **ram_wrun**
- **ram_data(7:0)** є двохнаправленою шиною поміж ram та Micro8.

Подано витяг із звіту про синтез проекту (мікростатистика витрат базових елементів ПЛІС). Звіт отримано по завершенню синтезу.

HDL Synthesis Report

Macro Statistics

# ROMs	: 1
64x8-bit ROM	: 1
# Adders/Subtractors	: 6
11-bit adder	: 2
2-bit adder	: 1
4-bit adder	: 2
9-bit adder	: 1
# Counters	: 3
2-bit down counter	: 1
24-bit up counter	: 1
7-bit down counter	: 1
# Registers	: 46
1-bit register	: 33
10-bit register	: 1
11-bit register	: 1
2-bit register	: 1
4-bit register	: 2
5-bit register	: 1
8-bit register	: 7
# Comparators	: 1
4-bit comparator greatequal	: 1
# Tristates	: 1
8-bit tristate buffer	: 1

Витяг зі звіту про імплементування містить відомості про утилізацію апаратних ресурсів ПЛІС отримано по завершенню імплементування.

Device Utilization Summary:

Number of BUFGMUXs	1 out of 16	6%
Number of External IOBs	33 out of 88	37%
Number of LOCed IOBs	0 out of 33	0%
Number of SLICES	224 out of 256	87%

Далі подамо автоматично здійснений САПР план розташування апаратних засобів мікроконтролера на кристалі ПЛІС.

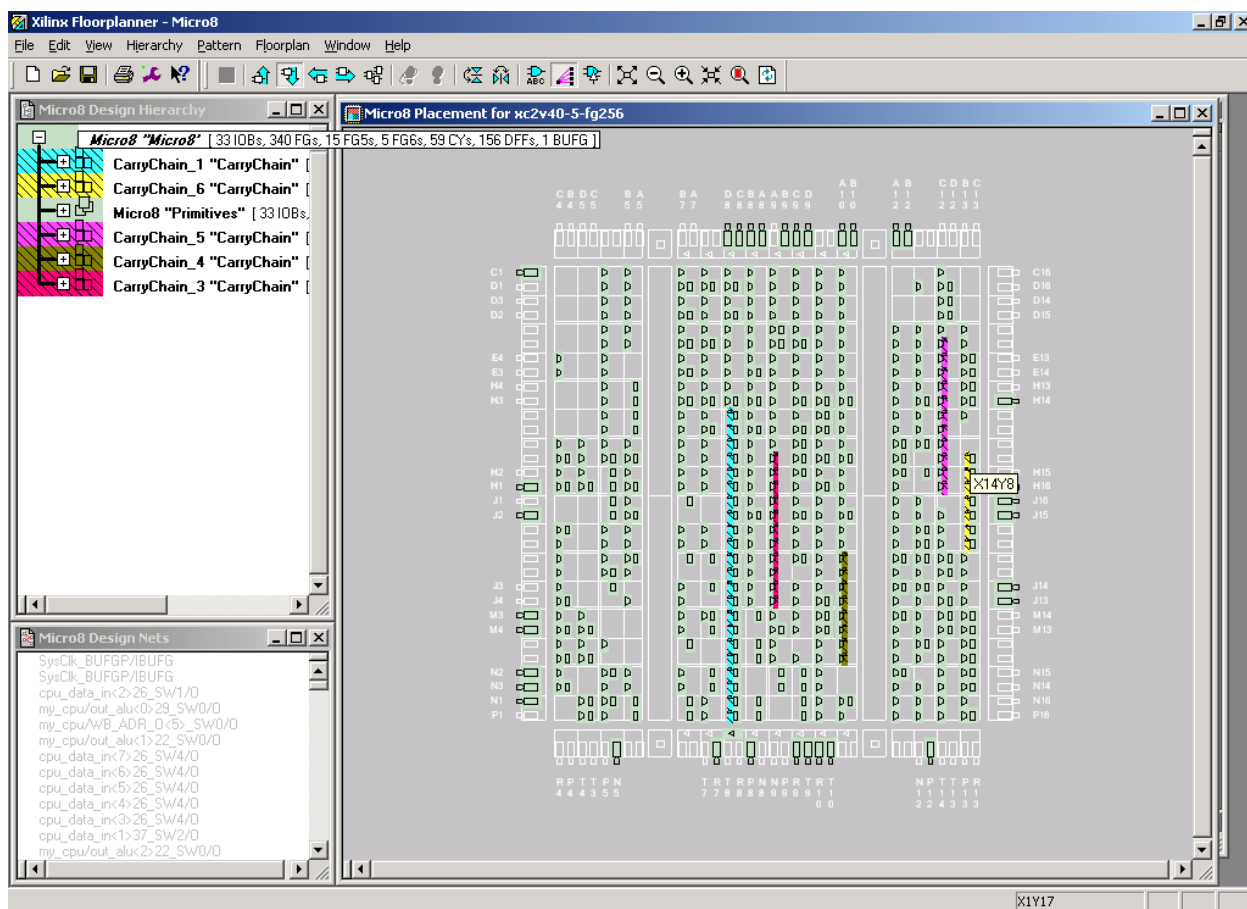


Рис. 7.4 - Топологія мікроконтролера Micro8 (утиліта Floorplanner зі складу WebPack)

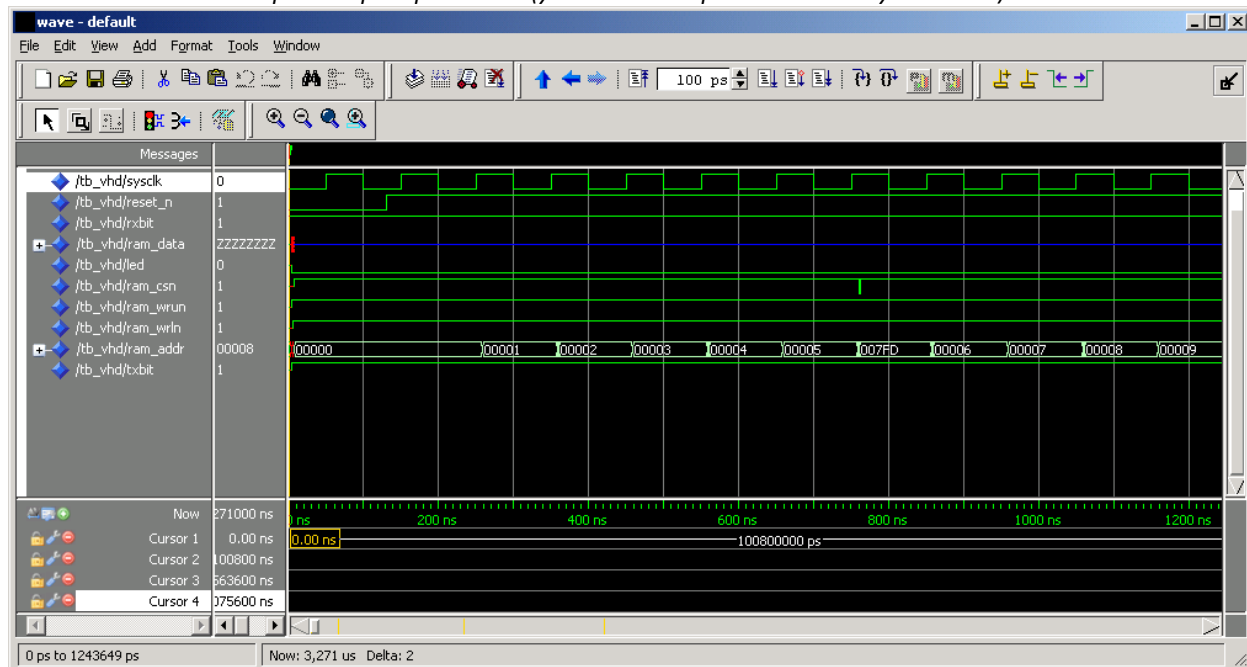


Рис. 7.5 - Часова поведінка мікроконтролера Micro8 на початку виконання програми

Розглянемо VHDL модель пам'яті стартової програми, а ця модель ще містить код повідомлення *Hello World*, що виводиться на бітову лінію txbit передавача Тх вихідного каналу UART, що є в складі мікроконтролера.

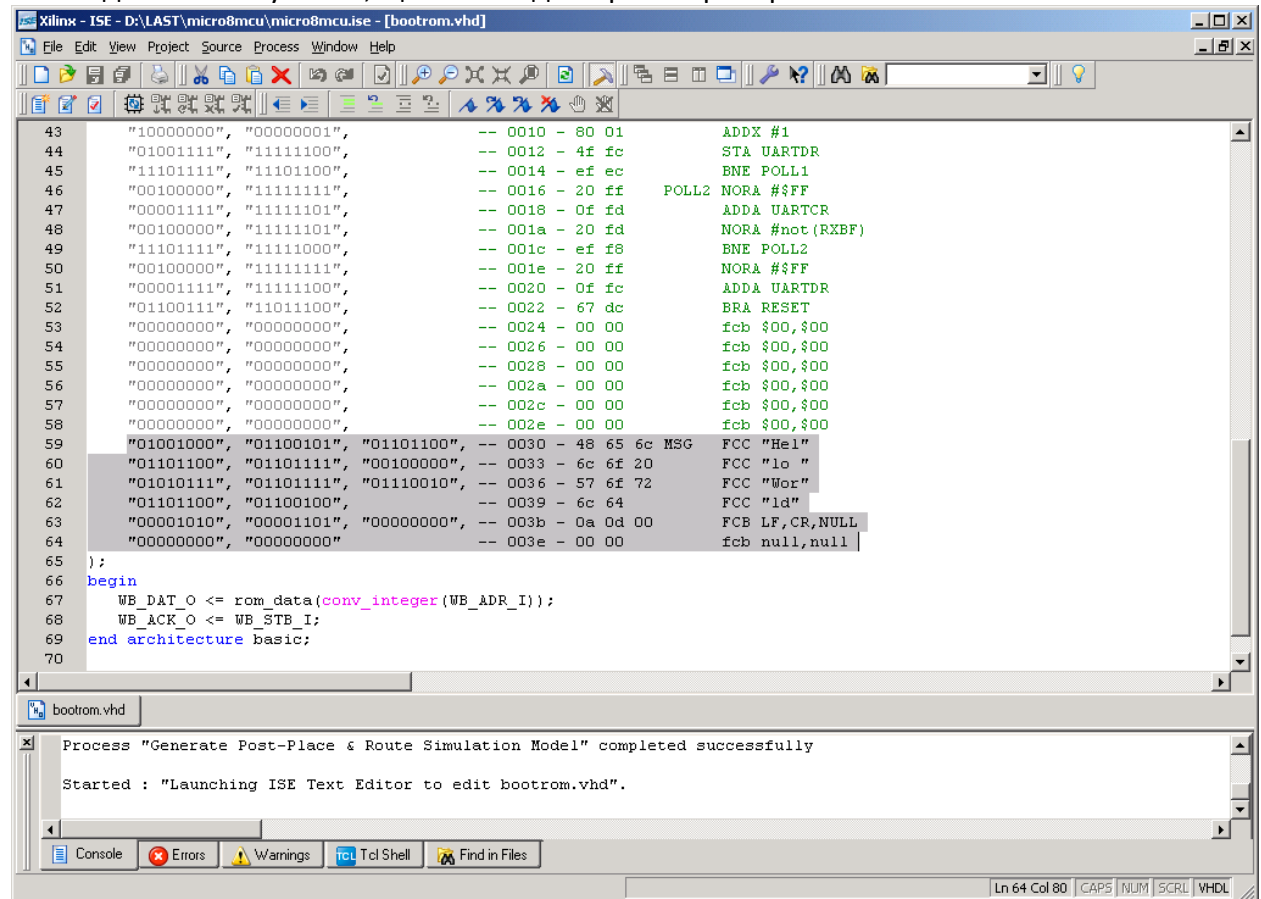


Рис. 7.6 - Код повідомлення Hello World в пам'яті стартової програми

Копію коду повідомлення Hello World подано нижче.

```

"01001000", "01100101", "01101100", -- 0030 - 48 65 6c MSG FCC "Hel"
"01101100", "01101111", "00100000", -- 0033 - 6c 6f 20      FCC "lo "
"01010111", "01101111", "01110010", -- 0036 - 57 6f 72      FCC "Wor"
"01101100", "01100100",      -- 0039 - 6c 64      FCC "ld"
"00001010", "00001101", "00000000", -- 003b - 0a 0d 00      FCB LF,CR,NULL
"00000000", "00000000",      -- 003e - 00 00      fcb null,null

```

Залишається пересвідчитися в тому, що поданий симуляційною часовою діаграмою код відповідає саме коду повідомленню *Hello World*.

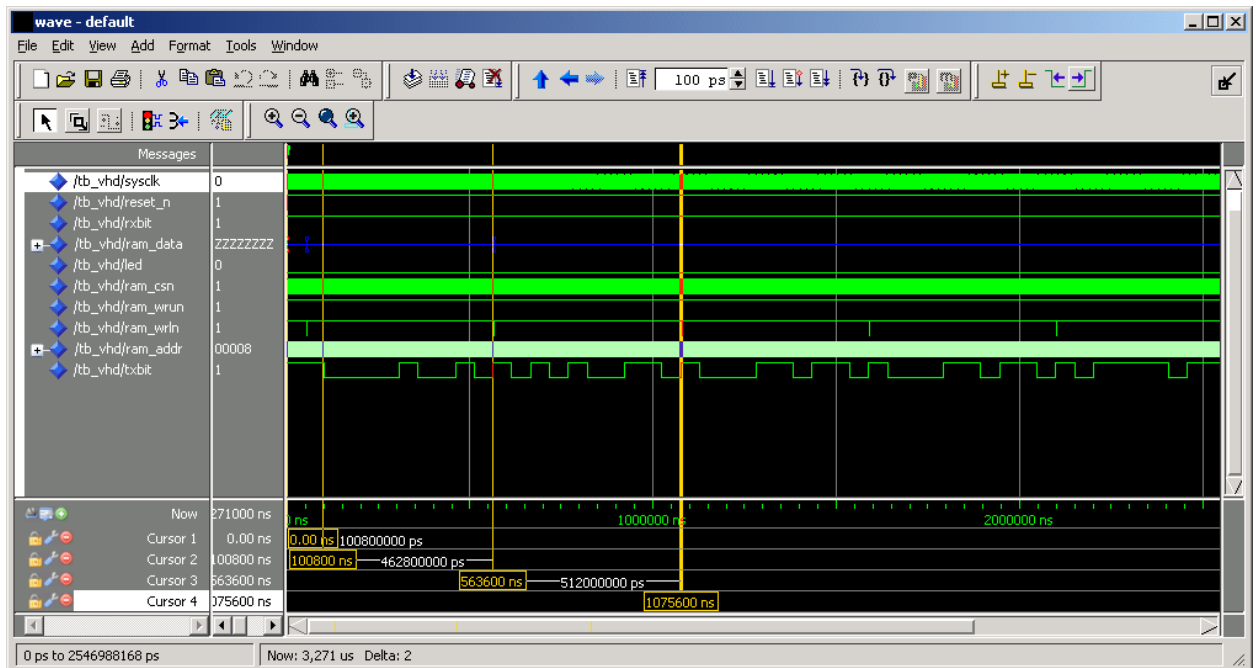


Рис. 7.7 - Симуляційна часова діаграма виконання програми Hello World комп'ютера Micro8

Надамо пояснення щодо часової діаграми для перших двох символів **He** повідомлення **Hello World** на виході *txbit*, яка від самого початку знаходиться в стані «1» (немає передачі в асинхронному послідовному інтерфейсі).

Таблиця 7.1 – Код послідовності **He** в вихідній лінії послідовного каналу UART

Stop	Start	H (48h ASCII)								Stop	Start	e (65h ASCII)								Stop
		4h				8h						6h				5h				
		8h (UART)				4h (UART)						5h (UART)				6h (UART)				
1	0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	1	1	0	1



Рис. 7.8 - Відповідний кодовий послідовності фрагмент симуляційної часової діаграми

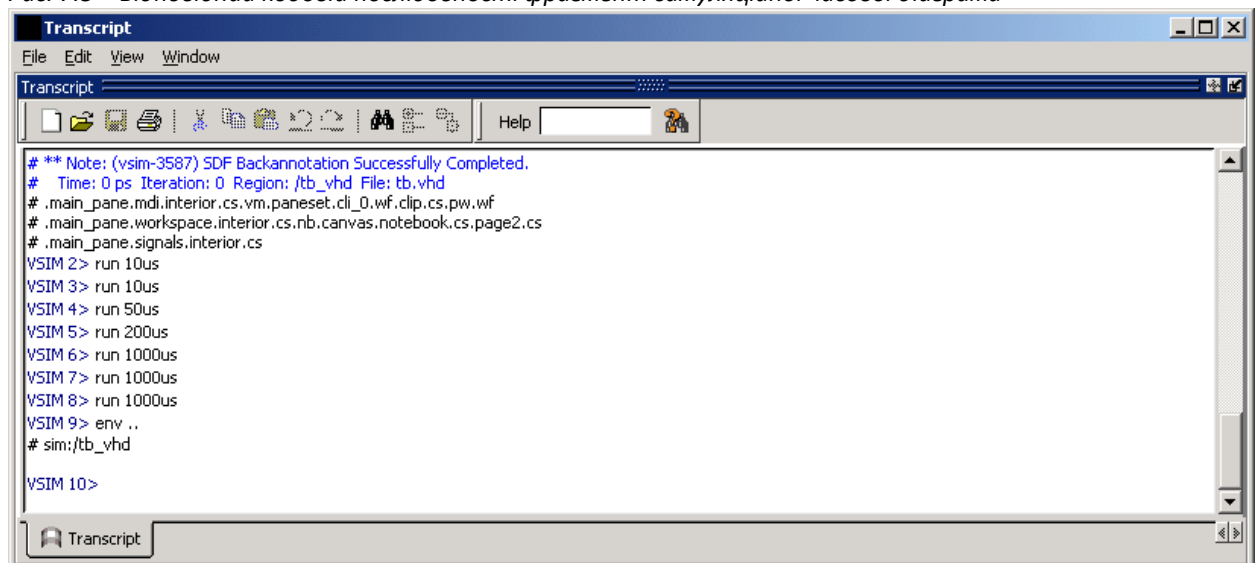


Рис. 7.9 - Скриптове вікно симулятора (бачимо, що повідомлення генерується і виводиться послідовним каналом UART дуже повільно, а саме: за мілісекунди на тактовій частоті 10 МГц)

Аби спостерігати на симуляційних часових діаграмах цикли внутрішньо кристальної шини **wishbone**, на якій побудовано мікроконтролер, достатньо проаналізувати топ-файл VHDL проекту, внести до нього потрібні зміни, потім пересинтезувати і переімплементувати проект та знову запустити симуляцію. При цьому зауважимо, що рис. 7.2 (Інтерфейс мікроконтролера Micro8) подає інтерфейс мікроконтролера назовні, він не є внутрішньою шиною wishbone, на якій і побудовано власне контролер. Отже, топ-файл проекту Micro8 має наступне вмістиме:

```
library ieee;
  use ieee.std_logic_1164.all;
  use IEEE.STD_LOGIC_ARITH.ALL;
  use IEEE.STD_LOGIC_UNSIGNED.ALL;
  use ieee.numeric_std.all;
entity Micro8 is
  port(
    SysClk    : in Std_Logic; -- System Clock input
    Reset_n   : in Std_Logic; -- Master Reset input (active low)
    led       : out std_logic; -- Diagnostic LED Flasher
    -- Memory Interface signals
    ram_csn   : out Std_Logic;
    ram_wrun  : out Std_Logic;
    ram_wrln  : out Std_Logic;
    ram_addr  : out Std_Logic_Vector(16 downto 0);
    ram_data  : inout Std_Logic_Vector(7 downto 0);
    -- Uart Interface
    rxbit     : in Std_Logic;
    txbit     : out Std_Logic
  );
end Micro8;
-- Architecture for memio Controller Unit
architecture my_computer of Micro8 is
  -----
  -- Signals
  -----
  -- BOOT ROM
  signal rom_data_out : Std_Logic_Vector(7 downto 0);
  signal rom_stb      : Std_Logic;
  signal rom_ack      : Std_Logic;
  -- UART Interface signals
  signal uart_data_out : Std_Logic_Vector(7 downto 0);
```

```

signal uart_stb      : Std_Logic;
signal uart_ack      : Std_Logic;
-- RAM
signal ram_stb       : std_logic; -- memory chip select strobe
signal ram_ack       : std_logic; -- memory chip select acknowledge
signal ram_wr        : std_logic; -- memory write
signal ram_data_out  : std_logic_vector(7 downto 0);
-- Sequencer Interface signals
signal cpu_reset     : Std_Logic;
signal cpu_we        : std_logic;
signal cpu_stb       : std_logic;
signal cpu_ack       : std_logic;
signal cpu_addr      : Std_Logic_Vector(10 downto 0);
signal cpu_data_in   : Std_Logic_Vector(7 downto 0);
signal cpu_data_out  : Std_Logic_Vector(7 downto 0);
signal cpu_irq_0     : std_logic;
signal cpu_irq_1     : std_logic;
-- Memory Transfer signals
signal countL       : std_logic_vector(23 downto 0);
-----

-- Open Cores Mini UART
-----

component UART is
  port (
    -- Wishbone signals
    WB_CLK_I : in std_logic; -- clock
    WB_RST_I : in std_logic; -- Reset input
    WB_ADR_I : in std_logic_vector(1 downto 0); -- Adress bus
    WB_DAT_I : in std_logic_vector(7 downto 0); -- DataIn Bus
    WB_DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
    WB_WE_I  : in std_logic; -- Write Enable
    WB_STB_I : in std_logic; -- Strobe
    WB_ACK_O : out std_logic; -- Acknowledge
    -- process signals
    IntTx_O  : out std_logic; -- Transmit interrupt: indicate waiting for Byte
    IntRx_O  : out std_logic; -- Receive interrupt: indicate Byte received
    BR_Clk_I : in std_logic; -- Clock used for Transmit/Receive
    TxD_PAD_O : out std_logic; -- Tx RS232 Line
    RxD_PAD_I : in std_logic; -- Rx RS232 Line
  )
end component UART;

component CPU8BIT2 is
  port (
    WB_CLK_I : in std_logic; -- clock

```

```

WB_RST_I : in std_logic; -- Reset input
WB_ADR_O : out std_logic_vector(10 downto 0); -- Adress bus
WB_DAT_I : in std_logic_vector( 7 downto 0); -- DataIn Bus
WB_DAT_O : out std_logic_vector( 7 downto 0); -- DataOut Bus
WB_WE_O : out std_logic; -- Write Enable
WB_STB_O : out std_logic; -- Strobe
WB_ACK_I : in std_logic -- Acknowledge
);
end component CPU8BIT2;

component boot_rom is
port (
    WB_ADR_I : in Std_Logic_Vector(5 downto 0); -- 64 byte boot rom
    WB_DAT_O : out Std_Logic_Vector(7 downto 0);
    WB_STB_I : in Std_Logic;
    WB_ACK_O : out Std_Logic
);
end component boot_rom;

begin
    -----
    -- Instantiation of internal components
    -----

my_uart : UART port map (
-- Wishbone signals
    WB_CLK_I => SysClk,
    WB_RST_I => cpu_reset,
    WB_ADR_I => cpu_addr(1 downto 0),
    WB_DAT_I => cpu_data_out,
    WB_DAT_O => uart_data_out,
    WB_WE_I  => cpu_we,
    WB_STB_I => uart_stb,
    WB_ACK_O => uart_ack,
-- process signals
    IntTx_O  => cpu_irq_1,
    IntRx_O  => cpu_irq_0,
    BR_Clk_I => SysClk,
    TxD_PAD_O => txbit,
    RxD_PAD_I => rxbit
);
my_cpu : CPU8BIT2 port map (
    WB_CLK_I => SysClk, -- clock
    WB_RST_I => cpu_reset, -- Reset input
    WB_ADR_O => cpu_addr(10 downto 0),

```

```

WB_DAT_I => cpu_data_in,
WB_DAT_O => cpu_data_out,
WB_WE_O => cpu_we,    -- Write Enable
WB_STB_O => cpu_stb,  -- Strobe
WB_ACK_I => cpu_ack    -- Acknowledge
);

my_rom : boot_rom port map (
    WB_ADR_I => cpu_addr(5 downto 0),
    WB_DAT_O => rom_data_out,
    WB_STB_I => rom_stb,
    WB_ACK_O => rom_ack
);
-----
-- Processes to decode memory devices
-----

decode: process( Reset_n,
    cpu_addr, cpu_we, cpu_stb,
    ram_stb, ram_ack,
    uart_stb, uart_ack,
    rom_data_out, uart_data_out, ram_data_out, cpu_data_out )
begin
    case cpu_addr(10 downto 6) is
        when "00000" =>
            cpu_data_in <= rom_data_out;
            rom_stb <= cpu_stb;
            ram_stb <= '0';
            uart_stb <= '0';
        when "11111" =>
            cpu_data_in <= uart_data_out;
            rom_stb <= '0';
            ram_stb <= '0';
            uart_stb <= cpu_stb;
        when others =>
            cpu_data_in <= ram_data_out;
            rom_stb <= '0';
            ram_stb <= cpu_stb;
            uart_stb <= '0';
    end case;
    cpu_reset <= not Reset_n; -- CPU reset is active high
    cpu_ack <= ram_ack or uart_ack or rom_ack;
end process;
-----

```

-- Processes to read and write RAM

```

ram: process( Reset_n,
             cpu_addr, cpu_we, cpu_stb,
             ram_stb, ram_wr,
             ram_data_out, cpu_data_out )
begin
    ram_wr <= Reset_n and cpu_stb and cpu_we;
    ram_wrln <= not ram_wr;
    ram_wrun <= '1';

    ram_csn <= not ram_stb;
    ram_ack <= ram_stb;

    ram_addr(16 downto 11) <= "0000000";
    ram_addr(10 downto 0) <= cpu_addr;
    ram_data_out(7 downto 0) <= ram_data(7 downto 0);
    if ram_stb = '1' then
        if ram_wr = '1' then
            ram_data(7 downto 0) <= cpu_data_out;
        else
            ram_data(7 downto 0) <= "ZZZZZZZZ";
        end if;
    else
        ram_data(7 downto 0) <= "ZZZZZZZZ";
    end if;
end process;

```

-- flash led to indicate code is working

```

increment: process (SysClk, CountL )
begin
    if Rising_Edge(SysClk) then
        countL <= countL + 1;
    end if;
    LED <= countL(21);
end process;

end my_computer; -- End of architecture

```

Сигнали шини wishbone подані нижче. Зауважимо, що назви всіх сигналів, що належать цій шині починаються префіксом WB та мають заключний суфікс I (вхідний сигнал) та (O)

```
WB_CLK_I => SysClk,    -- системні тактові імпульси (clock)
WB_RST_I => cpu_reset, -- вхід сигналу скиду мікроконтролера (Reset input)
WB_ADR_O => cpu_addr(10 downto 0), -- вихідні сигнали адреси
WB_DAT_I => cpu_data_in, -- вхідний сигнал на прийом даних
WB_DAT_O => cpu_data_out, -- вихідний сигнал на втвід даних
WB_WE_O => cpu_we,    -- вихідний сигнал дозволу запису (Write Enable)
WB_STB_O => cpu_stb,  -- вихідний стробовий (дозвільний) сигнал (Strobe)
WB_ACK_I => cpu_ack   -- вхідний сигнал підтвердження (Acknowledge)
```

Додатки

Короткі відомості про шину wishbone

Архітектура шини wishbone вирішує фундаментальну проблему задачі проектування інтегральних схем, а саме: як з'єднати поміж собою схемно реалізовані функції системи в нескладний спосіб. Зазвичай, схемно реалізовані функції додають до проекту в формі так званих інтелектуальних ядер (IP Cores - Intellectual Property Cores), які системні інтерпратори (в тому числі і розробники систем на кристалі) можуть придбати або самостійно розробити. З використанням архітектури wishbone IP перетворюються на будівельні блоки, з яких монтують систему. Цими IP блоками є мікропроцесори, послідовні порти, дискові інтерфейси, мережеві контролери тощо. Зазвичай, IP ядра проектують незалежно одне від іншого, а збирають їх до єдиної системи незалежні розробники. Це є можливим лише за умови стандартизації інтерфейсу IP ядер. Варіантом такої стандартизації є шина wishbone.

Вступ

Шина wishbone використовує MASTER/SLAVE архітектуру. Це значить, що MASTER ініціює обмін даними через шину зі SLAVE-ом. Рис. Д-1 показано, як MASTERS та SLAVES комунікують поміж собою через інтерфейс з назвою INTERCON. INTERCON найкраще уявляти собі як хмарку, що містить певні схеми. Саме ці схеми і дозволяють майстрам MASTER-ам комунікувати з SLAVE-ми.

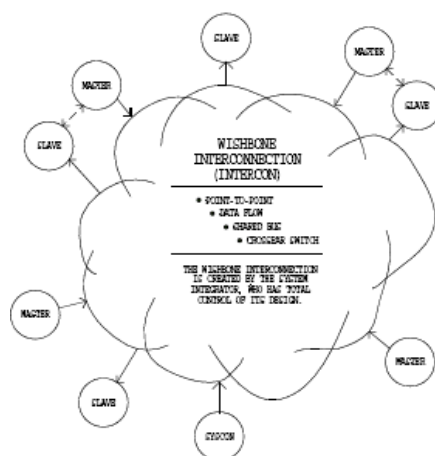


Рис. Д-1. З'єднання в системі на основі шини WISHBONE

Досягні різні варіанти з'єднань дозволяють проектувальнику оптимізувати побудову системи на основі шини wishbone, яка за властивостями є відмінною від інших шин, наприклад, PCI, cPCI, VMEbus та ISA.

Реалізують wishbone як синхронну схему. При цьому wishbone не накладає обмежень на часові специфікації (вони забезпечує асинхронний обмін).

Власник шини wishbone фірма Silicore Corporation визначила з'єднання на шині WISHBONE в формі тестів, що викладені мовою VHDL. Це дозволило фірмі абсолютно точно специфікувати комунікації і комунікаційні процеси в wishbone.

Типи з'єднань на шині wishbone

Шина WISHBONE дозволяє різні варіанти з'єднань ядер з інтелектуальними властивостями (IP cores) поміж собою. Дозволені наступні чотири типи з'єднань на шині WISHBONE:

- Point-to-point (точка-точка)
- Data flow (потік даних)
- Shared bus (розділена шина)
- Crossbar switch (перехрестний комутатор)

З'єднання точка-точка (point-to-point)

Цей вид з'єднання є найпростішим способом комутації двох wishbone IP ядер. Рис. Д-2 подає, як комутують поміж собою інтерфейс пристрою MASTER (здатний керувати шиною) та інтерфейс SLAVE (підлеглий, не здатний керувати шиною). Наприклад, інтерфейс MASTER може належити мікропроцесорному ядру (IP core), а інтерфейс SLAVE – послідовному порту введення-виведення (serial I/O port).

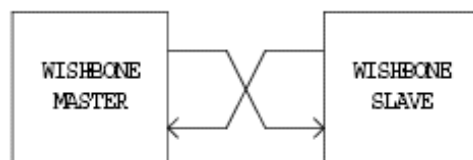


Рис. Д-2. З'єднання точка-точка (point-to-point interconnection)

З'єднання потоком даних (Data Flow)

Комутування потоком даних (data flow interconnection) використовують тоді, коли дані можна пересилати від одного пристрою до іншого за правилами конвеєра. Рис. Д-3 показано, що кожне ядро (IP core) в архітектурі потоку даних має обидва типи інтерфейси на wishbone, а саме: MASTER та SLAVE інтерфейси. Дані розповсюджуються від ядра до ядра (from core-to-core), тобто, за принципом конвеєра (pipeline).

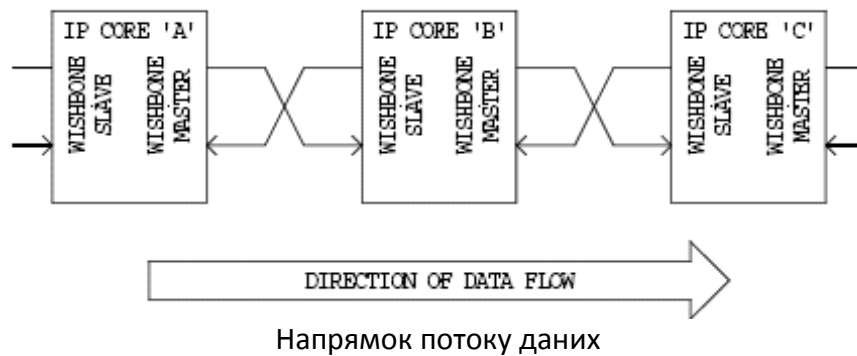


Рис. Д-3 - З'єднання потоком даних

Архітектура потоку даних використовує паралелізм заради прискорення. Наприклад, коли на поданому рисунком Д-3 кожний вузол є готовим ядром (IP core) для виконання операцій з рухомою комою, тоді така система надає втричі вищу продуктивність в порівнянні з системою на одному ядрі. Ясно, що наявною є можливість виконання дій в конвеєрному режимі та що всі складові операції мають вирівнену тривалість виконання в часі.

З'єднання розділеною шиною (Shared Bus)

Розділена шина переважає, коли потрібно зкомутувати два або більше майстрів MASTER с двома або більше SLAVE-ми. Відповідну структуру подає рис.Д-4. Тут MASTER ініціює (запускає) цикл шини для цільового (обраного ним) SLAVE-а. Обраний SLAVE може комутуватися з MASTER- протягом одного чи більшого числа розташованих поспіль циклів шини.

Так званий Арбітр, що не позначений на структурі, кожного разу вирішує, якому з декількох «зацікавлених» в володінні шиною майстрів надається керування шиною. В цей спосіб уникають конфліктів поміж майтрами в змаганнях за володіння шиною. Наприклад, арбітр може надавати майстрам шину за певними пріоритетами або за правилом циклічного розподілу (the shared bus can use a priority or a round robin arbiter).

Головною перевагою розділеної шини є заощадження логічних ресурсів на реалізацію, за що сплачують невисокою швидкодією системи в цілому.

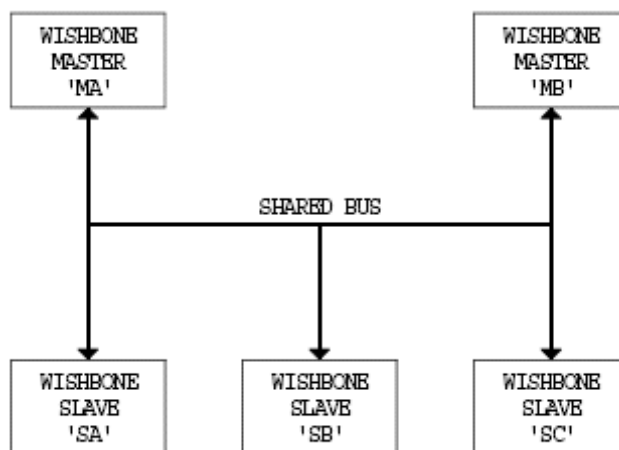


Рис Д-4 - З'єднання шиною, що розділяється.

Вимоги стандарту шини wishbone не обмежують імплементацію розділеної шини. Її можна здійснити, наприклад, з використанням мультиплексора або засобами, що мають високоімпедансний («третій») стан. This gives the system integrator additional flexibility, as some logic chips work better with multiplexor logic, and some work better with three-state buses. Зауважимо, що ідею розділеної шини втілили ще PCI та VMEbus.

Інтерфейсні сигнали шини WISHBONE

Інтерфейси пристроїв wishbone MASTER та пристроїв wishbone SLAVE повинні забезпечити багато варіантність зв'язку. Тому інтерфейсні сигнали на шині wishbone, так само, як і цикли на цій шині, є достатньо гнучкими. Перелічимо наступні властивості сигналів шини wishbone:

- Сигнали дозволяють інтерфейсам пристроїв MASTER і SLAVE підтримувати комунікації point-to-point, потоку даних (data flow), розділеної шини (shared bus) та перехрещеного комутатора (crossbar switch).
- Сигнали дозволяють виконувати три типи циклів шини (basic types of bus cycle), а саме:
 - а) SINGLE READ/WRITE одноразове читання/запис,
 - б) BLOCK READ/WRITE блокове читання/запис,
 - в) RMW (read-modify-write, прочитати-змінити-записати).
- Підтримується механізм асинхронного обміну (handshaking mechanism – рукопотискання). Це значить, що на WISHBONE всі цикли шини вирівнені на швидкодію найповільнішого інтерфейсу.
- Асинхронний механізм дозволяє SLAVE приймати/видавати сигнали на шину, повторювати видачу даних, коли трапилися помилки та просити MASTER повторювати цикл шини. При цьому SLAVE генерує сигнали вихідного підтвердження [ACK_O], вихідний сигнал помилки [ERR_O] та вихідний сигнал запитування на повтор циклу на шині [RTY_O], відповідно. Всі реалізації шини мусять підтримувати сигнал [ACK_O], але підтримка інших сигналів не є обов'язковою.
- Всі сигнали на інтерфейсах MASTER-а і SLAVE-а є або вхідними, або вихідними, але ніколи не двохнаправленими (three-state). І це лише тому, що деякі ПЛІС (FPGA) та технології виготовлення замовних мікросхем класу ASIC, не підтримують двохнаправлені сигнали (на кристалах самих мікросхем). Але самі по собі двохнаправлені сигнали є ефективними в комунікаціях.
- Ширину шин адреси і даних можна змінювати, аби пристосуватися до задачі. Дозволені 8, 16, 32 та 64-бітові шини даних і 0-64 бітові шини адреси.
- Як подає рис. Д-5, всі сигнали організовані так, що інтерфейси MASTER і SLAVE можуть напряму комутуватися один з іншим в формі нескладного з'єднання класу точка-точка. Це надзвичайно спрощує реалізацію з'єднання на шині wishbone. Саме так, наприклад, можна приєднати ззовні до системи wishbone стороннє мікропроцесорне ядро (microprocessor IP Core). Проте і на кристалі реалізація шини є ефективною.
- Дозволено додавати до стандарту wishbone користувацькі сигнали в формі тегів («tags», позначок, ярликів). Наприклад, розробник (ним, як

правило, є системний інтегратор) може додати біт парності до шин даних і адреси.

Повний перелік сигналів на WISHBONE містять специфікації шини.

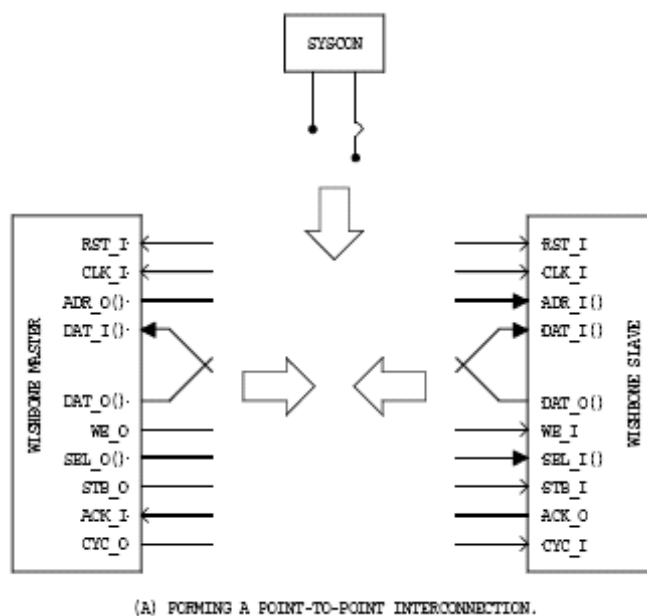


Рис. Д-5 (початок) - Напрямки пересилання інформації (а) шиною WISHBONE і сигнали шини в з'єднанні точка-точка (b)

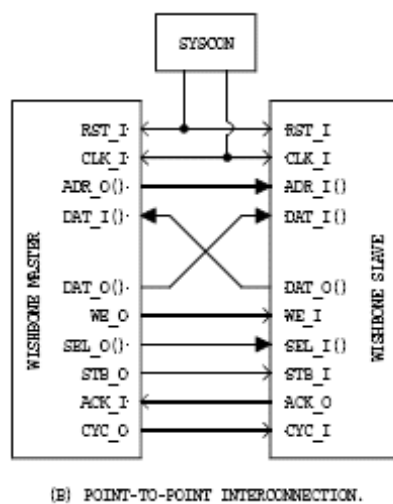


Рис. Д-5 (продовження) - Напрямки пересилання інформації (а) шиною WISHBONE і сигнали шини в з'єднанні точка-точка (b)

Цикли шини WISHBONE

Нагадаємо, що існують три типи циклів на шині WISHBONE:

- SINGLE READ/WRITE (одноразовий цикл читання/запис)
- BLOCK READ/WRITE (блокове читання/запис, множинні читання/записи)

- READ MODIFY WRITE (RMW) (читання/модифікація (зміна)/запис

Одноразовий цикл читання/запис (SINGLE READ/WRITE Cycle)

Цикл SINGLE READ/WRITE є найпоширенішим типом циклу на шині wishbone. Його використовують для пересилання шиною одного операнду (single data operand). Рис. Д-6 подає типовий цикл SINGLE READ.

Специфікації шини wishbone містять часові діаграми всіх типів циклів, як для інтерфейсу MASTER, так і для інтерфейсу SLAVE в варіанті конфігурації точка-точка (point-to-point configuration). Розглянемо інтерфейсні сигнали з боку пристрою MASTER. Це дозволить подати інтерфейс без розгляду цілої системи. Наприклад, сигнал з назвою [ACK_I] є вхідним для інтерфейсу пристрою MASTER. Ясно, що його прямо з'єднують з сигналом [ACK_O], що генерується підпорядкованим пристроєм SLAVE. Коли розглядають часову діаграму з погляду пристрою SLAVE, тоді мусять розглядати сигнал підтвердження [ACK_O], що генерує цей SLAVE.

В циклі SINGLE READ відбувається наступне :

1. В відповідь на фронт такту 0, інтерфейс MASTERa формує сигнали на вихідних лініях [ADR_O()], [WE_O], [SEL_O], [STB_O] та [CYS_O].
2. SLAVE розпізнає новий цикл шини спостереженням зміни сигналу вхідного стробу [STB_I] та адресних входів, а потім формує коректні дані на власних вихідних лініях [DAT_O()]. Коли система має конфігурацію точка-точка, тоді вихідні сигнали SLAVE [DAT_O()] під'єднані до вхідних сигналів MASTER [DAT_I()].
3. SLAVE повідомляє майстру, що він розташував коректні дані на шині за допомогою власного вихідного сигналу підтвердження [ACK_I]. При цьому SLAVE має право затримати пересилання, уводячи один чи більше станів очікування на шині. В цьому випадку SLAVE відкладає в часі видачу сигналу підтвердження для майстра. Можливість уведення на шині станів очікування позначають на часових діаграмах скороченням -WSS- (Wait StateS).
4. MASTER спостерігає стан власної вхідної лінії [ACK_I], аби дізнатися, коли SLAVE підтвердить пересилання на фронті такту (at clock edge 1).
5. MASTER заціпіє [DAT_I()] та інвертує власний вихідний строб [STB_O] signal в відповідь на вхідний сигнал [ACK_I].

Одноразовий цикл запису SINGLE WRITE виконують аналогічно, хіба за винятком того, що MASTER генерує вихідний сигнал дозволу запису [WE_O] та розміщує дані, що мають писатися, на власних вихідних лініях [DAT_O]. При цьому SLAVE генерує власний вихідний сигнал підтвердження [ACK_O] після того, як заціпіє вхідні дані на запис.

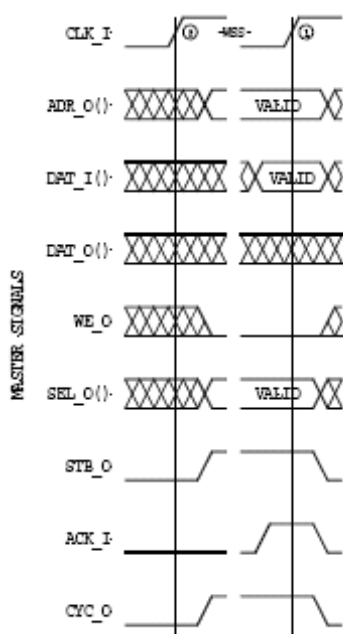


Рис. Д-6 - Цикл одиночного читання (SINGLE READ)

Цикл блокового читання/запису

Зблоковані цикли BLOCK READ/WRITE дуже схожі на одиночні цикли SINGLE READ/WRITE. Зблоковані цикли BLOCK можна мислити як два або більше циклів SINGLE, що розташовані поспіль. Індивідуальні цикли, які разом і складають блок, називають фазами.

Початок і завершення циклів BLOCK позначають, відповідно, підтвердженням і зняттям сигналу від майстра з назвою [CYC_O]. Сигнал [CYC_O] використовують розділені шини і перехрещені комутатори тому, що він інформує системну логіку про те, що MASTER має намір використовувати шину.

Цикл зчитування-модифікації-запису READ-MODIFY-WRITE (RMW)

Шинний цикл READ-MODIFY-WRITE використовують мультипроцесорні і мультизадачні (multiprocessor and multitasking) системи. Цей спеціальний тип циклу дозволяє програмним процесам розділяти спільні ресурси через семафори. А це є потрібним для інтерфейсів дискових контролерів, послідовних портів і інтерфейсів до пам'яті. Як то впливає з назви, цикл READ-MODIFY-WRITE читає і після модифікації пише дані до комірки пам'яті в одному циклі шини (цикл шини може складатися з десяти і більше тактових інтервалів процесора). Це дозволяє одному програмному процесу виконувати роботу за два (або більше) програмні процеси. При цьому цикл READ-MODIFY-WRITE треба мислити як один неподільний цикл.

Порцію читання цього циклу називають фазою читання (read phase), відповідно, порцію запису – фазою запису (write phase). Коли аналізувати часові діаграми

цього виду циклу шини, тоді ці діаграми можна сприймати як дві фази циклу типу BLOCK, де перша фаза читає, а друга фаза пише.

Цикл READ-MODIFY-WRITE треба мислити неподільним ще і тому, що арбітр, надавши шину на обмежений час певному майстру, ніколи не перериває такий цикл, якщо він вже розпочався. Це дозволяє майстру-мікропроцесору прочитати дані, модифікувати їх, а потім записати результат модифікації. І все це робиться в одному циклі, з мінімумом накладених (на організацію циклу) часових витрат.

ЛР № 8А. Дослідження SystemC моделі автомата

Мета: опанування технікою дослідження імплементованих SystemC моделей типових комп'ютерних пристроїв

Завдання

Промодельювати поведінку та верифікувати поведінку створеної на основі базової моделі моделі керуючого автомата Мура. Роботу виконати в системі SystemC_Win (www.systemc.org). За результатами виконання лабораторних досліджень скласти звіт та захистити його.

Техніка роботи в SystemC Win

Аби продемонструвати роботу системи SystemC Win та дії розробника в цій системі розглянемо нескладний приклад моделювання поведінки D-тригера на основі його SystemC моделі. Цей приклад розглядаємо як навчальний (ілюстративний). Він не має прямого відношення до предмету досліджень цієї лабораторної роботи.

В межах годин самостійної роботи, що визначені програмою дисципліни «Дослідження і проектування КСМ» (а це є $71-4=67$ академічних годин в 2010 році), студент отримує систему SystemC Win та компілятор C++ фірми Borland версії 5.5, що розповсюджуються вільно. Всі необхідні файли присутні в наступній копії вікна файлового менеджера.

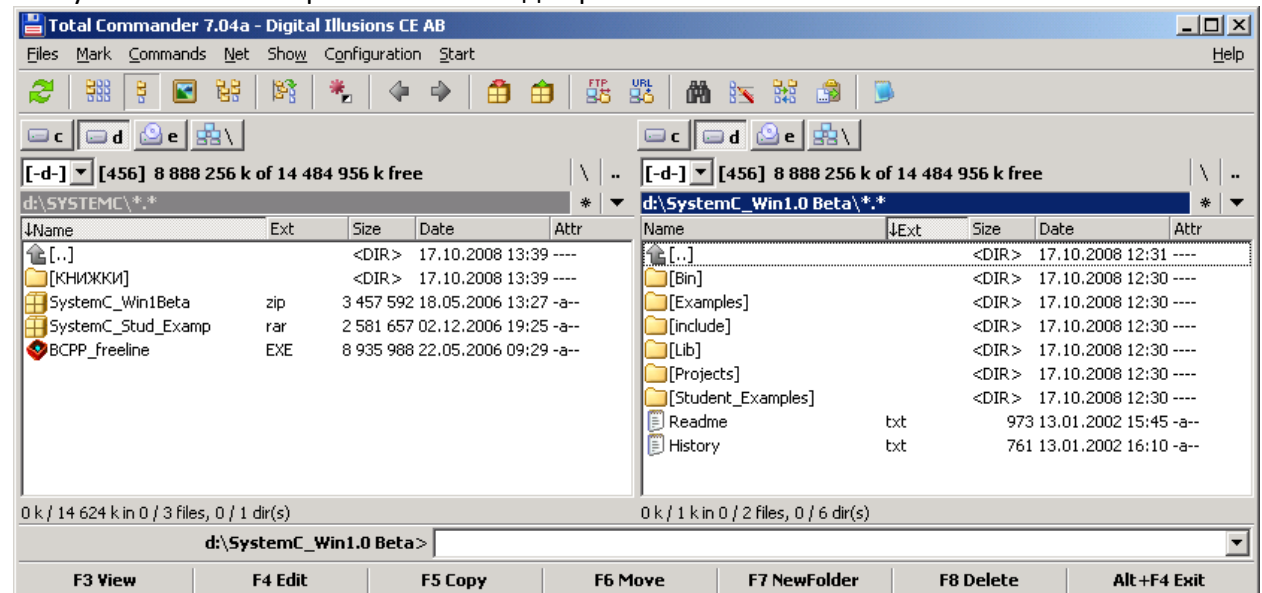


Рис. 8а.1 – Копія вікна файлового менеджера з файлами системи SystemC Win

Ліворуч на копії вікна в директорії SYSTEMC розташоване наступне:

- SystemC_Win1Beta.zip – архів з системою SystemC Win;
- SystemC_Stud_Exam.rar – архів з прикладами для вивчення студентами;
- BCPP_freeline – безкоштовний компілятор мови C++ фірми Borland;

- КНИЖКИ – директорія, що містить дві книжки з питань застосування SystemC.

Праворуч на копії вікна бачимо директорії системи SystemC Win з наступним призначенням:

- Bin – директорія містить ехе-файл системи SystemC Win; цей файл не інсталюється, а просто викликається для запуску системи;
- Examples – директорія зі штатними для організації OSCI (open systemc initiative, див. www.systemc.org) прикладами (тут є приклад **risc_cpu** для наступної лабораторної роботи);
- Student_Examples – директорія, що містить приклади, призначені студентам; саме тут розташована директорія з ілюстративним прикладом D-тригера (**dff**).

Інші директорії містять службову інформацію системи SystemC.

Отже, потрібно спочатку проінсталювати на власному ПК компілятор Borland, а вже потім розархівувати SystemC Win. Далі треба знайти в SystemC Win файл Systemc_Win.exe та запустити його. Після запуску системи SystemC_win потрібно вказати, де розташована інсталяція компілятора Borland.

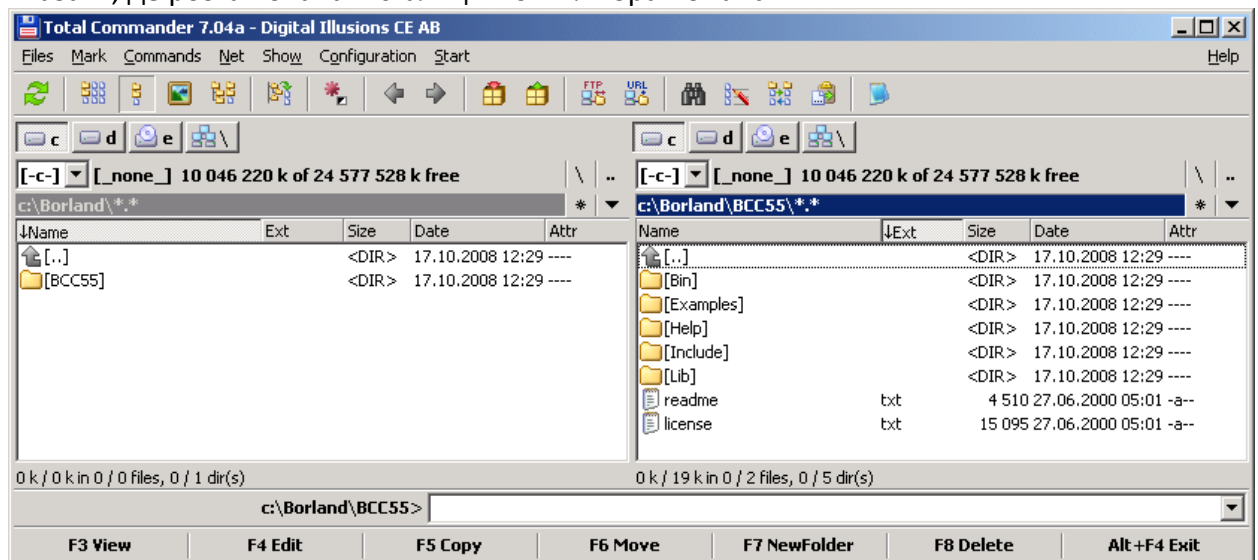


Рис. 8а.2 – Копія вікна файлового менеджера з файлами компілятора Borland C++

Після цього все готове до моделювання. Потрібно лише завантажити до системи приклад з моделлю тригера і можна розпочинати дослідження.

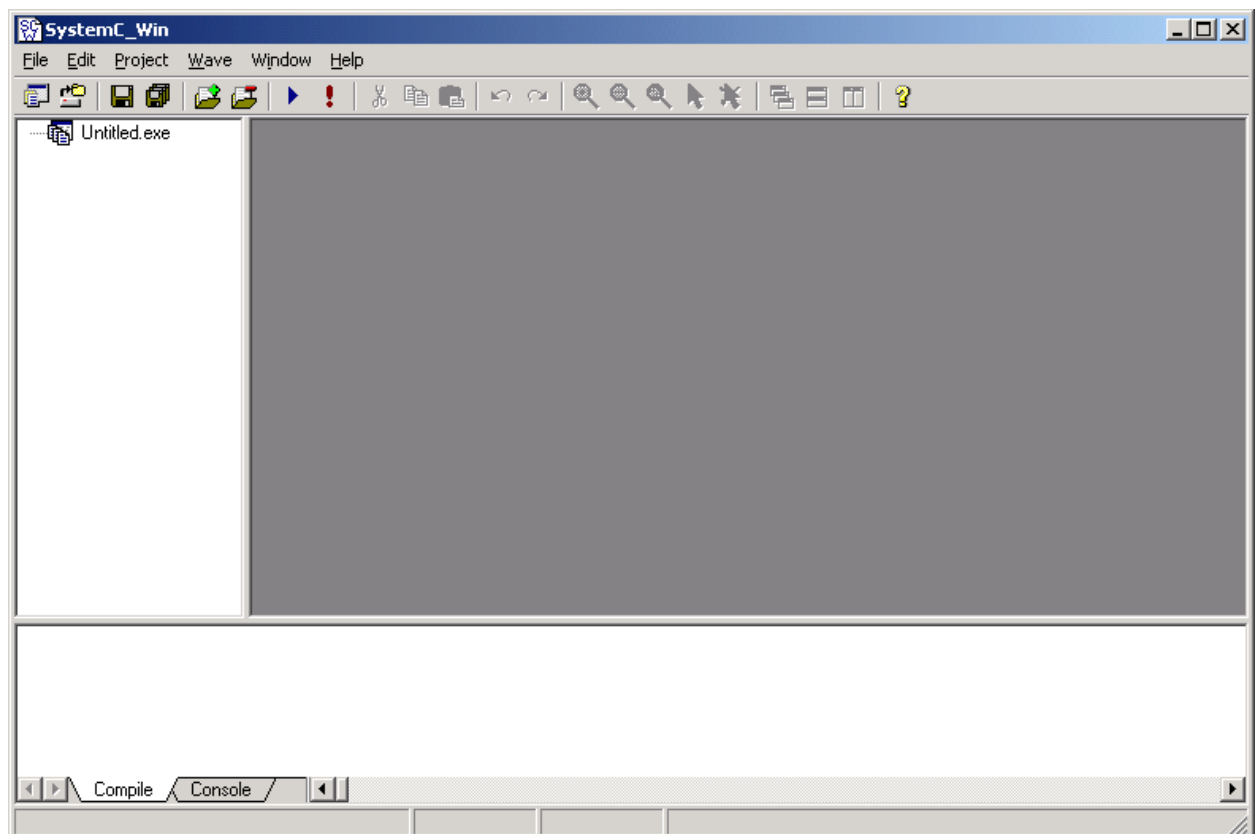


Рис. 8а.3 – Активований навігатор проектів SystemC Win

Далі розглянемо модель D-тригера, що в готовому виді є консольною програмою для Windows. Модель складають п'ять файлів наступного призначення:

- клас тригера;
- метод тригера;
- клас (допоміжної) консолі для стимулювання тригера та для спостереження за його реакцією на стимули;
- методи консолі;
- тест-бенч (дослідницька лабораторія, де створюється система з двох мікросхем тригера та консолі, а створена система приводиться до дії і моделює поведінку тригера.

Далі подамо вміст файлів моделі.

Клас D-тригера

// файл my_dff.h D-FF Declaration

#include "systemc.h"

```
SC_MODULE(my_dff) {
    sc_in<sc_logic> din_pin;
    sc_out<sc_logic> dout_pin;
    sc_in_clk clk_pin;
```

```
void doit();
```

```
SC_CTOR(my_dff) {
    SC_METHOD(doit);
    sensitive_pos << clk_pin;
}
};
```

Метод D-тригера

// файл my_dff.cpp D-FF Implementation

```
#include "my_dff.h"
void my_dff::doit(){
    dout_pin->write(din_pin->read());
}
```

Клас консольної мікросхеми

// File mon_stim.h Declaration

```
#include <systemc.h>
SC_MODULE(console) {
    sc_in_clk clk_pin;
    sc_out<sc_logic> din_out;
    sc_in<sc_logic> dout_in;
    bool stimul;
```

// Stimulus and Monitor processes

```
void monitor();
void stim();
```

// Constructor

```
SC_CTOR(console) {
    SC_METHOD(monitor);
    sensitive_neg << clk_pin;
    SC_THREAD(stim);
    sensitive_neg << clk_pin;
}
};
```

Методи консольної мікросхеми

// Implementation. File mon_stim.cpp

```
#include "console.h"
void console :: monitor () {
    std::cout << "Time is : " << sc_time_stamp()
        << ", din = " << din_out
```

```

        << ", dout = " << dout_in
        << std::endl;
    }

void console :: stim () {
    sc_logic stimul_logic;
    stimul = false;
    stimul_logic = sc_logic_0;
    din_out -> write(stimul_logic);
    while (true) {
        din_out -> write(stimul_logic);
        stimul = !stimul;
        if (stimul)
            stimul_logic = sc_logic_0;
        else
            stimul_logic = sc_logic_1;
        wait();
    }
}

```

Тест-бенч проекту «D-триггер»

// File main.cpp Testbench

```

#include "systemc.h"
#include "my_dff.h"
#include "console.h"
int sc_main(int argc, char* argv[]) {
    sc_signal<sc_logic> din_wire;
    sc_signal<sc_logic> dout_wire;
    sc_clock clk("Clock", 10, 0.5, 0);

    // Initialize input
    dout_wire = sc_logic_0;
    din_wire = sc_logic_0;

    // Instantiate modules and signals connect
    my_dff dff_inst("dff_inst");
    dff_inst.din_pin(din_wire);
    dff_inst.dout_pin(dout_wire);
    dff_inst.clk_pin(clk);

    console console_inst("console_inst");
    console_inst.din_out(din_wire);
    console_inst.dout_in(dout_wire);
    console_inst.clk_pin(clk);
}

```

```

//tracing
sc_trace_file* tf;
tf = sc_create_vcd_trace_file ("dff_wave");
if (!tf)
    cout << "There was an file creating error." << endl;
else
    cout << "Create vcd file!" << endl;
sc_trace(tf, clk.signal(), "Clock");
sc_trace(tf, din_wire, "din");
sc_trace(tf, dout_wire, "dout");
sc_start(300);
sc_close_vcd_trace_file(tf);
cout << "Close vcd file. Bye!" << endl;
sc_stop();
return(0);
}

```

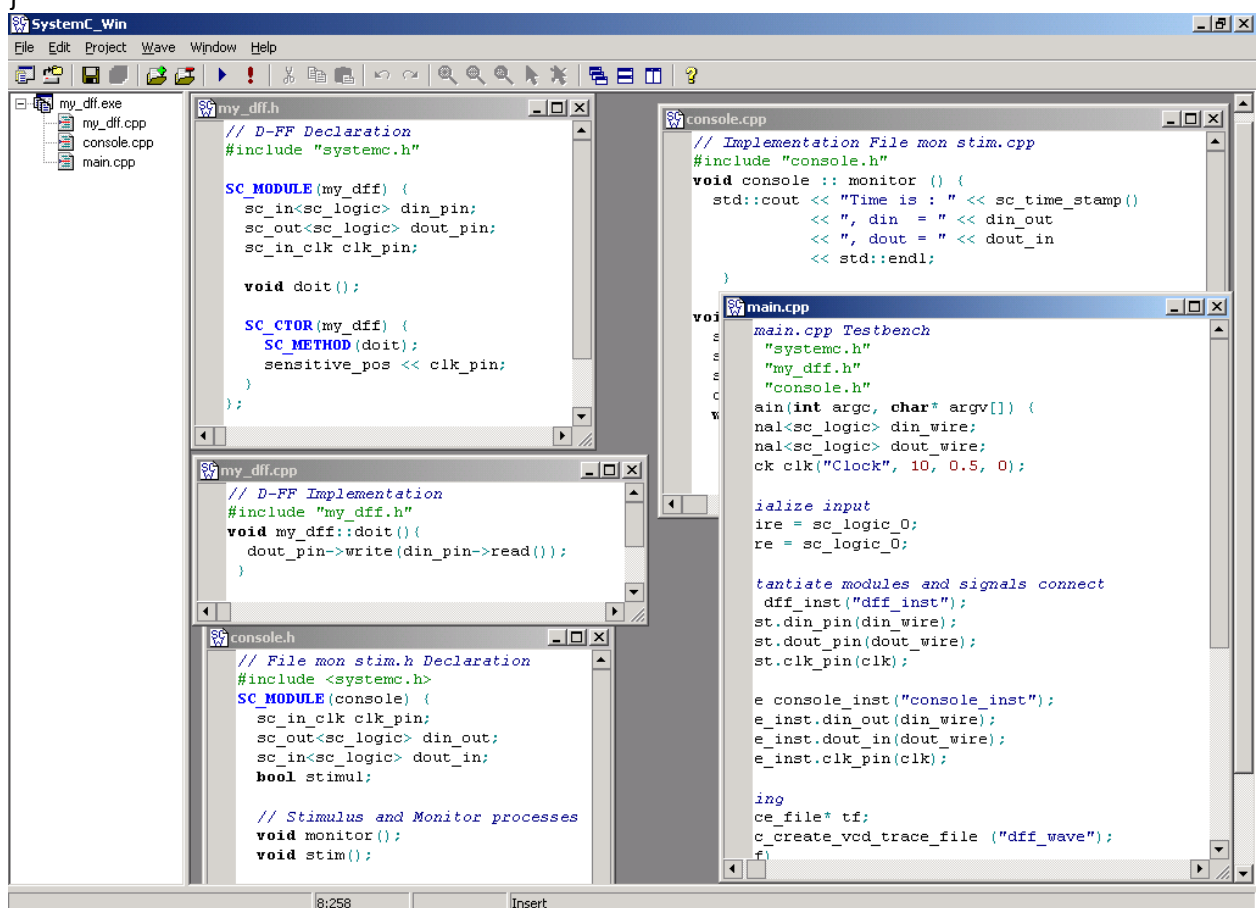


Рис. 8а.4 – Навігатор проектів SystemC Win із завантаженим проектом “D-тригер”

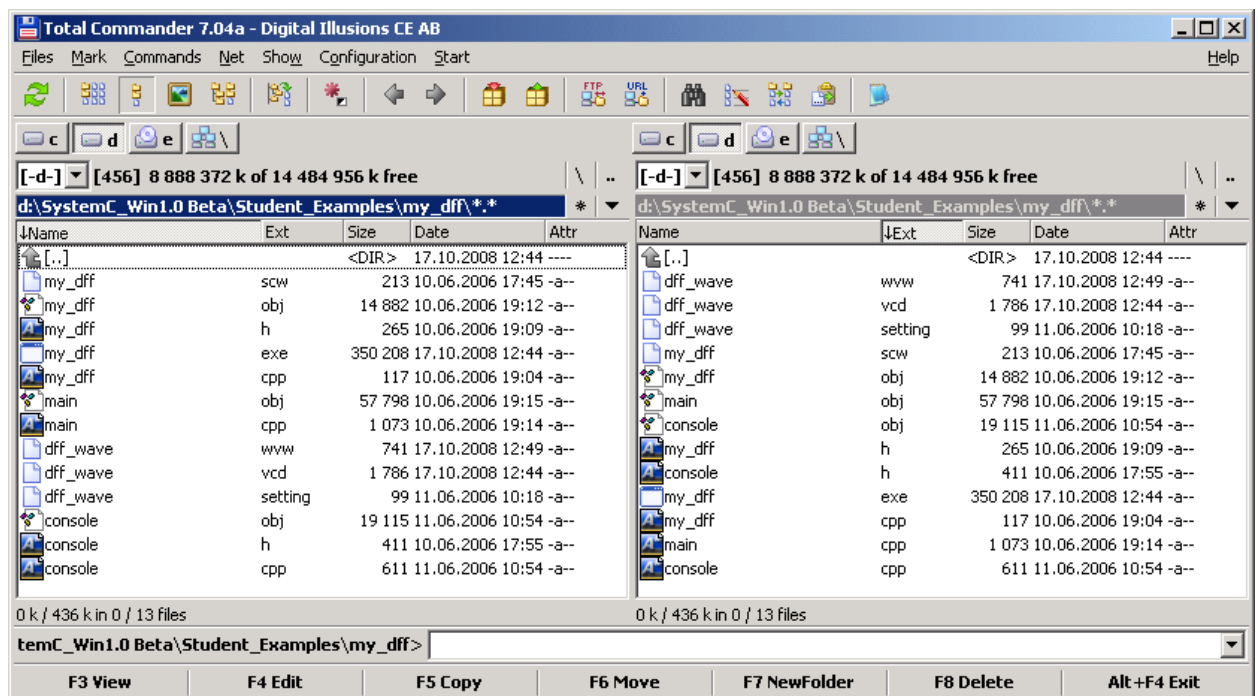


Рис. 8а.5 – Вікно файлового менеджера з файлами скопійованого проекту тригера

Призначення файлів проекту є наступним:

- my_dff.h – файл, що містить клас мікросхем D-тригера my_dff;
- my_dff.cpp – файл, що містить методи класу my_dff;
- my_dff.vcd – файл з часовими діаграмами поведінки моделі тригера;
- my_dff.exe – власне модель (консольна аплікація в термінах Windows) D-тригера my_dff;
- console.h – файл, що містить клас допоміжних мікросхем console, за допомогою яких формуються сигнали на вході тригера та спостерігаються відповідні входам реакції на виході тригера;
- my_dff.cpp – файл, що містить методи класу my_dff;
- main.cpp – так званий тест-бенч, тобто, майстерня, де на основі двох мікросхем (тригер і консоль) створено схему, що спроможна промодельовувати поведінку тригера.

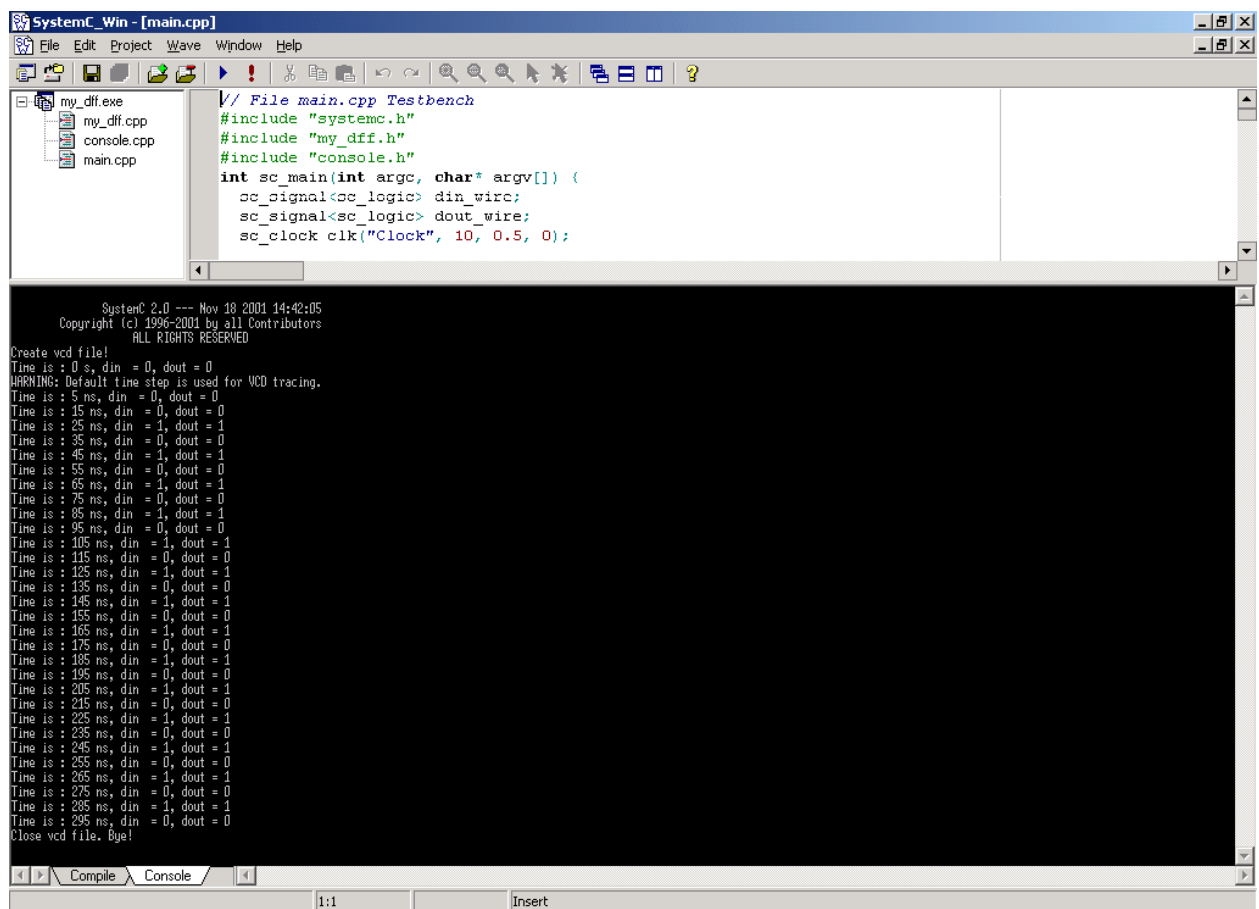


Рис. 8а.6 – Консольне вікно навігатора проєктів SystemC Win. Результати моделювання тригера

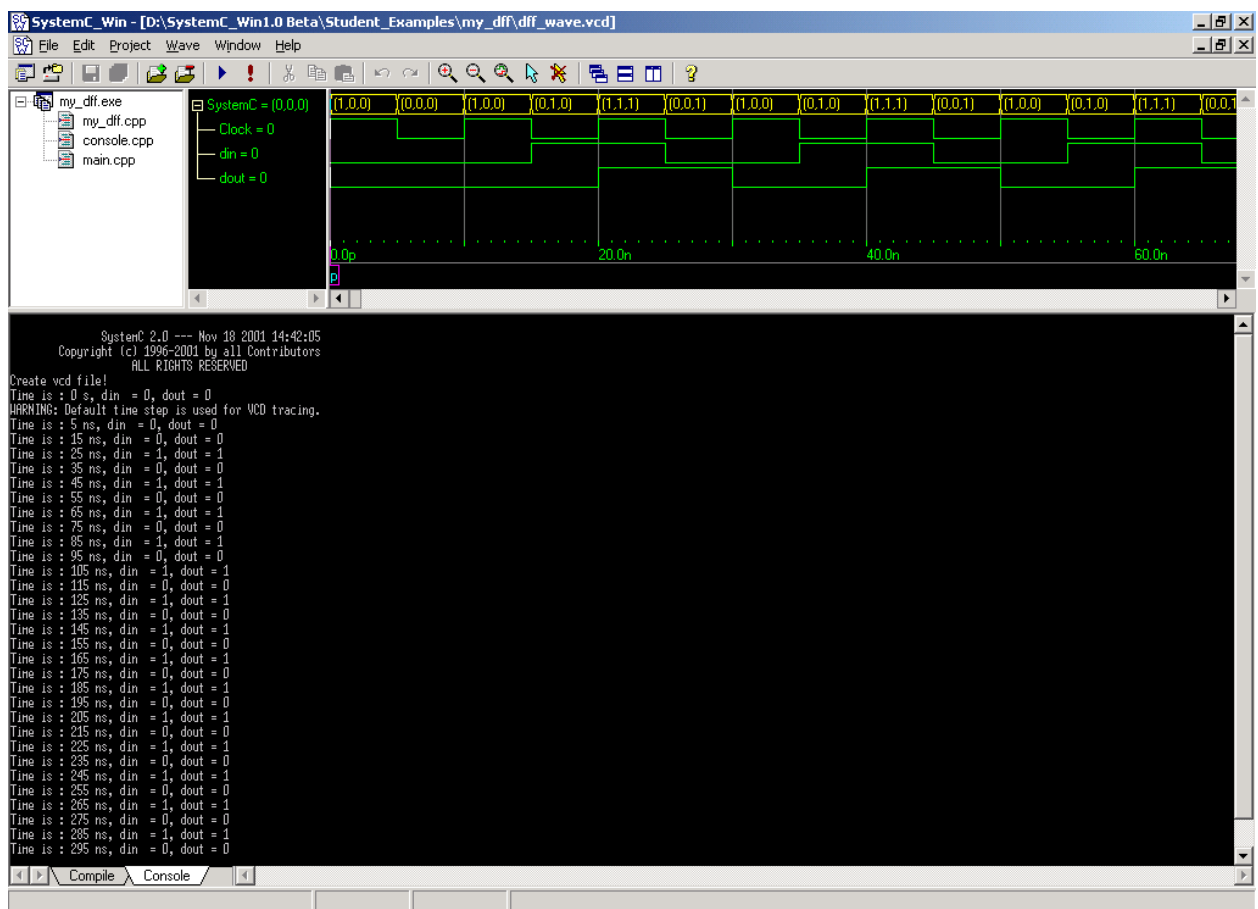


Рис. 8а.7 – Файлове вікно і консольне вікно навігатора проектів SystemC Win. Результати моделювання тригера

У файловому вікні навігатора бачимо візуалізовані часові діаграми, які є вмістимим файла dff_wave.vcd. Часові діаграми сформовані (створеною засобами компілятора BorlandC та на основі наявного SystemC опису) моделлю (консольною програмою Windows) my_dff.exe після її запуску з вікна командного рядка ОС Windows. Саме модель my_dff.exe створила файл dff_wave.vcd nf по завершенню моделювання занесла до нього дані про часові діаграми.

Повертаємося до теми досліджень поточної лабораторної роботи. Зараз ясно, що базова SystemC модель автомата також складатиметься з п'яти файлів.

Базова SystemC модель керуючого автомата Мура

Проект моделі керуючого автомата, що пропонується як прототип (база) для виконання лабораторних досліджень складається з 5 файлів. Два файли *.h містять проголошення класів, відповідно, цільового керуючого автомата Мура (moore.h) та монітора-стимулятора (mon_stim.h), призначеного для

генерування стимулів для автомата та для спостереження (моніторингу) реакцій автомата на вхідні стимули.

Ще два файли **moore.cpp**, **mon_stim.cpp**) містять C++ опис реалізації методів для автомата і для монітора-стимулятора, тобто, надають реалізації цих двох класів. Останній файл **main.cpp** є тест-бенчем проекту, тобто, тестовою лабораторією, де спочатку беруться по одному примірнику автомата і монітор-стимулятора, дрти, генератор тактових імпульсів, а потім збирається схема з цих двох мікросхем, що запускається на функціонування протягом 300 тактових інтервалів. Додатково створюється файл, що містить траси змін в часі певних сигналів (їхні часові діаграми). Файл трас має розширення **vcd**.

Ясно, що все функціонування відбувається віртуально, за допомогою вбудованого до бібліотеки SystemC симулятора. Сама створена в проекті модель представляє собою ехе файл, що є консольною аплікацією Windows. Він створений засобами вільно розповсюджуваного компілятора C++ фірми Borland. Після активації модель виводить повідомлення на консоль про числові значення стимулів і відповідних ним реакцій, а також створює файл трас, що містить часові діаграми зміни сигналів в часі. Є спеціальний в'ювер для цих трас. Він є вмонтованим до системи SystemC Win. Консоль з повідомленнями також вмонтована до SystemC Win.

Ясно, що порядок дій є наступним:

- Прокомпілювати модель в системі SystemC Win і отримати ехе файл моделі.
- Викликати модель на виконання, отримати повідомлення на консолі і файл трас.
- Проаналізувати отриману інформацію і порівняти її з результатом теоретичного дослідження поведінки наданої базової моделі автомата Мура.
- За умови збіжності отриманих теоретично і в експерименті результатів, модифікувати базову модель і провести нові дослідження. Отримані результати подати звітом, що має захищатися.

Далі йде текст п'яти файлів, що складають базову SystemC модель керуючого автомата Мура.

Клас **moore**

// File moore.h Class declaration

#include "systemc.h"

```
SC_MODULE(moore) {
    sc_in<sc_logic> a_in_pin, reset_in_pin; // input ports
    sc_in_clk clk_in_pin; // clock input
    sc_out<sc_logic> z_out_pin; // output port
    sc_out<sc_uint<2>> state_out_pin; // port for monitoring
```

```

// Internal variable
enum state_type {s0, s1, s2, s3};
sc_signal<state_type> moore_state;
sc_uint<2> state_int;

// FSM processes
void state(); // synchro process
void output(); // combi process
// Constructor
SC_CTOR(moore) {
    SC_METHOD(state);
    sensitive_pos << clk_in_pin;
    SC_METHOD(output);
    sensitive << clk_in_pin;
}
};

```

Методи класу moore

// File moore.cpp Implementation

```
#include "moore.h"
```

```

void moore::state(){
    bool cond1, cond2;
    if (reset_in_pin->read()== sc_logic_1) cond1 = true; else cond1 = false;
    if (a_in_pin->read()== sc_logic_1) cond2 = true; else cond2 = false;

    if (cond1) //->read()
        moore_state = s0;
    else
        switch(moore_state){
            case s0: moore_state = cond2 ? s0:s2; break;
            case s1: moore_state = cond2 ? s0:s2; break;
            case s2: moore_state = cond2 ? s2:s3; break;
            case s3: moore_state = cond2 ? s1:s3; break;
        }
}

void moore::output(){
    switch(moore_state){
        case s3: z_out_pin->write(sc_logic_1); state_int = 3; break;
        case s0: z_out_pin->write(sc_logic_1); state_int = 0; break;
        case s1: z_out_pin->write(sc_logic_0); state_int = 1; break;
        case s2: z_out_pin->write(sc_logic_0); state_int = 2; break;
    }
}

```

```
state_out_pin->write(state_int);
}
```

Клас монітору

// File mon_stim.h Declaration

```
#include <systemc.h>
SC_MODULE(mon_stim) {
    sc_in_clk clk_in;
    sc_out<sc_logic> a_out;
    sc_out<sc_logic> reset_out;
    sc_in<sc_logic> z_in;
    sc_in<sc_uint<2>> state_in;
    bool stimul;
    bool reset_int;
    // Stimulus and Monitor processes
    void monitor();
    void stim();
    // Constructor
    SC_CTOR(mon_stim) {
        SC_METHOD(monitor);
        sensitive_neg << clk_in;
        SC_THREAD(stim);
        sensitive_neg << clk_in;
    }
};
```

Методи монітору-стимулятора

// Implementation File mon_stim.cpp

```
#include <math.h>
#include "mon_stim.h"
void mon_stim::monitor () {
    std::cout << "Time is : " << sc_time_stamp()
        << ", reset = " << reset_out
        << ", state = " << state_in
        << ", z = " << z_in
        << ", a = " << a_out
        << std::endl;
}
```

```
int clknum = 0 ;
```

```
void mon_stim::stim (){
    sc_logic stimul_logic, reset_int_logic;
    stimul = false;
```

```

stimul_logic = sc_logic_0;
reset_int = true;
reset_int_logic = sc_logic_1;
a_out->write(stimul_logic);
reset_out->write(reset_int_logic);
clknum ++;
while (true) {
    a_out->write(stimul_logic);
    reset_out->write(reset_int_logic);

    if (stimul)
        {stimul = false; stimul_logic = sc_logic_0;}
    else
        {stimul = true; stimul_logic = sc_logic_1;}

    if (clknum < 5 )
        {reset_int = true; reset_int_logic = sc_logic_1;}
    else
        {reset_int = false; reset_int_logic = sc_logic_0;}

    clknum ++;
    wait();
}
}

```

Тест-бенч проекту

// File main.cpp. Testbench

```

#include "systemc.h"
#include "moore.h"
#include "mon_stim.h"
int sc_main(int argc, char* argv[]) {
    sc_signal<sc_logic> a_wire;
    sc_signal<sc_logic> reset_wire;
    sc_signal<sc_logic> z_wire;
    sc_signal<sc_uint<2>> state_wire;
    sc_clock clk("Clock", 10, 0.5, 0);
    //Initialize input
    a_wire = sc_logic_0;
    reset_wire = sc_logic_0;
    state_wire=0;
    // Instantiate modules and signals connect
    moore moore_inst("moore_inst");
    moore_inst.clk_in_pin(clk);
    moore_inst.reset_in_pin(reset_wire);

```

```

moore_inst.a_in_pin(a_wire);
moore_inst.z_out_pin(z_wire);
moore_inst.state_out_pin(state_wire);
mon_stim mon_stim_inst("mon_stim_inst");
mon_stim_inst.a_out(a_wire);
mon_stim_inst.reset_out(reset_wire);
mon_stim_inst.clk_in(clk);
mon_stim_inst.z_in(z_wire);
mon_stim_inst.state_in(state_wire);
//tracing
sc_trace_file *tf = sc_create_vcd_trace_file ("moore_wave");
sc_trace(tf, clk.signal(), "Clock");
sc_trace(tf, a_wire, "a input");
sc_trace(tf, z_wire, "z output");
sc_trace(tf, reset_wire, "reset");
sc_trace(tf, state_wire, "state");
sc_start(300);
return(0);
}

```

```

C:\WINDOWS\system32\cmd.exe

SystemC 2.1.v1 --- Oct 20 2005 16:11:09
Copyright (c) 1996-2005 by all Contributors
ALL RIGHTS RESERVED
Time is : 0 s, reset = 0, state = 0, z = X, a = 0
WARNING: Default time step is used for UCD tracing.
Time is : 5 ns, reset = 1, state = 2, z = 0, a = 0
Time is : 15 ns, reset = 1, state = 0, z = 1, a = 1
Time is : 25 ns, reset = 1, state = 0, z = 1, a = 0
Time is : 35 ns, reset = 1, state = 0, z = 1, a = 1
Time is : 45 ns, reset = 1, state = 0, z = 1, a = 0
Time is : 55 ns, reset = 0, state = 0, z = 1, a = 1
Time is : 65 ns, reset = 0, state = 0, z = 1, a = 0
Time is : 75 ns, reset = 0, state = 2, z = 0, a = 1
Time is : 85 ns, reset = 0, state = 2, z = 0, a = 0
Time is : 95 ns, reset = 0, state = 3, z = 1, a = 1
Time is : 105 ns, reset = 0, state = 1, z = 0, a = 0
Time is : 115 ns, reset = 0, state = 2, z = 0, a = 1
Time is : 125 ns, reset = 0, state = 2, z = 0, a = 0
Time is : 135 ns, reset = 0, state = 3, z = 1, a = 1
Time is : 145 ns, reset = 0, state = 1, z = 0, a = 0
Time is : 155 ns, reset = 0, state = 2, z = 0, a = 1
Time is : 165 ns, reset = 0, state = 2, z = 0, a = 0
Time is : 175 ns, reset = 0, state = 3, z = 1, a = 1
Time is : 185 ns, reset = 0, state = 1, z = 0, a = 0
Time is : 195 ns, reset = 0, state = 2, z = 0, a = 1
Time is : 205 ns, reset = 0, state = 2, z = 0, a = 0
Time is : 215 ns, reset = 0, state = 3, z = 1, a = 1
Time is : 225 ns, reset = 0, state = 1, z = 0, a = 0
Time is : 235 ns, reset = 0, state = 2, z = 0, a = 1
Time is : 245 ns, reset = 0, state = 2, z = 0, a = 0
Time is : 255 ns, reset = 0, state = 3, z = 1, a = 1
Time is : 265 ns, reset = 0, state = 1, z = 0, a = 0
Time is : 275 ns, reset = 0, state = 2, z = 0, a = 1
Time is : 285 ns, reset = 0, state = 2, z = 0, a = 0
Time is : 295 ns, reset = 0, state = 3, z = 1, a = 1

D:\TEMP\my_moore\my_moore\OSCI_win32-msvc.net>

```

Рис. 8а.8 – Консольні повідомлення базової моделі керуючого автомата Мура

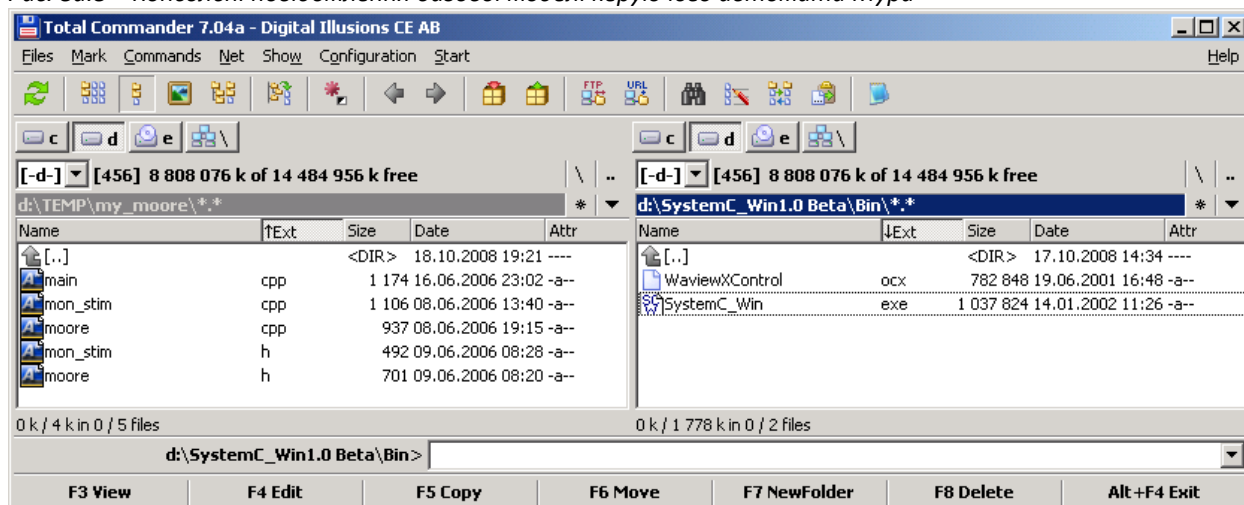


Рис. 8а.9 – Файли проекту (ліворуч), файли системи SystemC Win (праворуч)

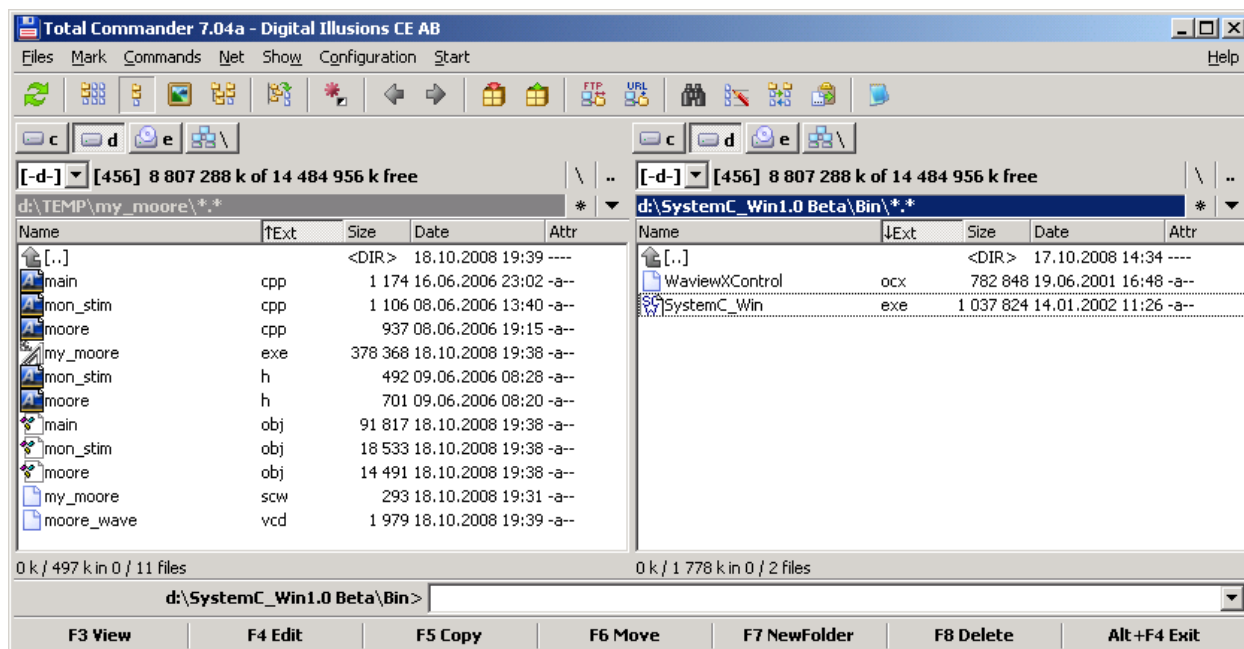


Рис. 8а.10 – Файли прокомпільованого проекту (ліворуч); модель – це файл my_moore.exe

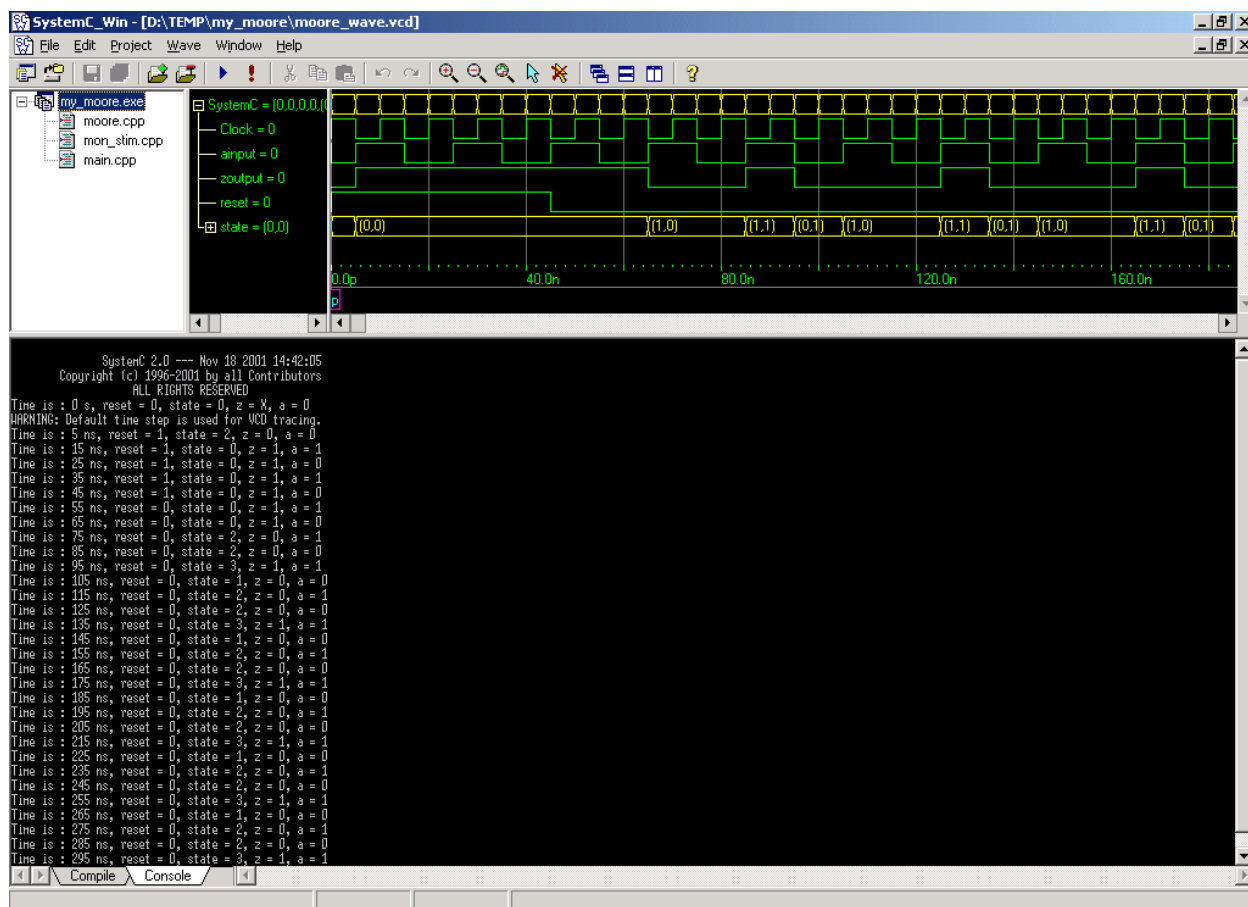


Рис. 8а.11 – Часові діаграми (нагорі) та консольні повідомлення моделі (знизу)

ЛР № 8Б. Синтез та дослідження SystemC моделі RISC процесора

Мета: опанування технікою високорівневого моделювання та проектування на основі мови SystemC

Завдання

Промодельовати та верифікувати поведінку наданої SystemC моделі RISC процесора з штатною програмою. Виконати зміни в штатній програмі та знову промодельовати і верифікувати поведінку зміненої SystemC моделі. Отримані результати порівняти. Роботу виконати в системі SystemC_Win (www.systemc.org). За результатами виконання лабораторної роботи скласти звіт і захистити його.

Іншими словами, завдання лабораторної роботи полягає в аналізі з подальшим поясненням поведінки (запозиченої від OSCI) моделі RISC процесора.

Зауваження:

SystemC модель RISC процесора міститься в системі SystemC_Win. Код цієї моделі є громіздким, тому не надається. Систему SystemC_Win разом з численними SystemC поведінковими моделями комп'ютерних пристроїв можна отримати або на кафедрі ЕОМ, або на сайті Інтернет www.systemc.org. Модель можна дослідити на власному ПК.

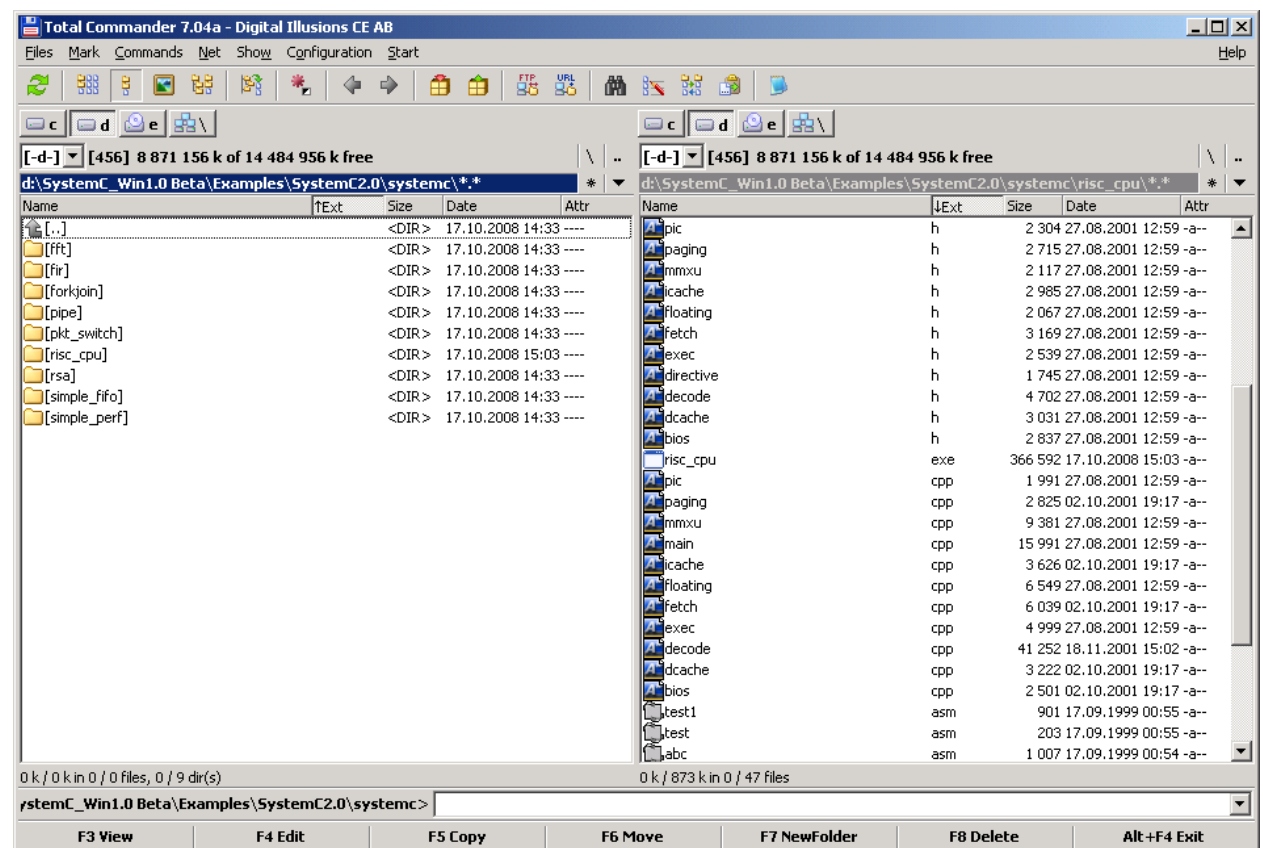


Рис. 8б.1 – Файловий менеджер і проект SystemC risc_cpu

Техніка проведення експерименту

1. Завантаження проекту до системи SystemC Win

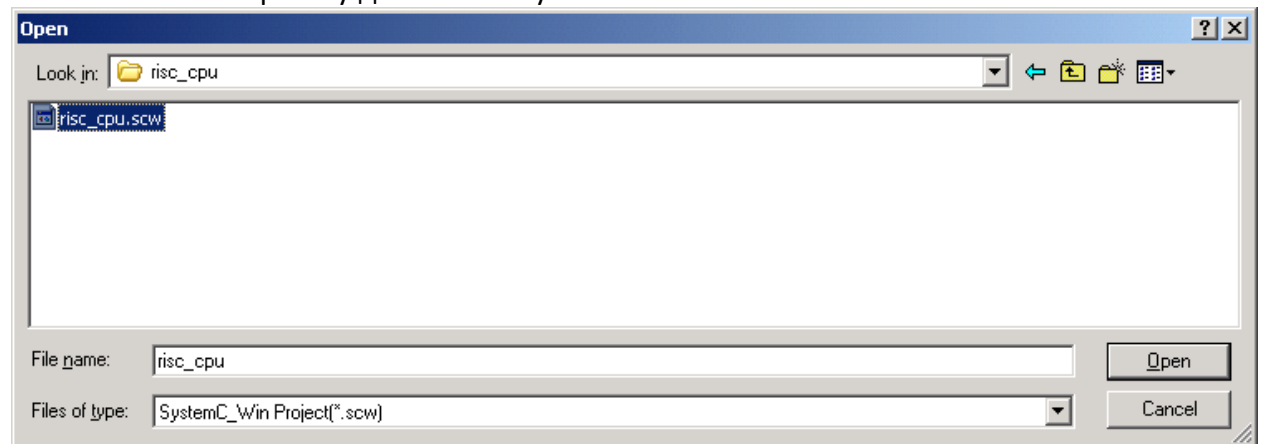


Рис. 86.2 – Завантаження проекту *risc_cpu*

2. Система SystemC Win з завантаженою моделлю RISC процесора

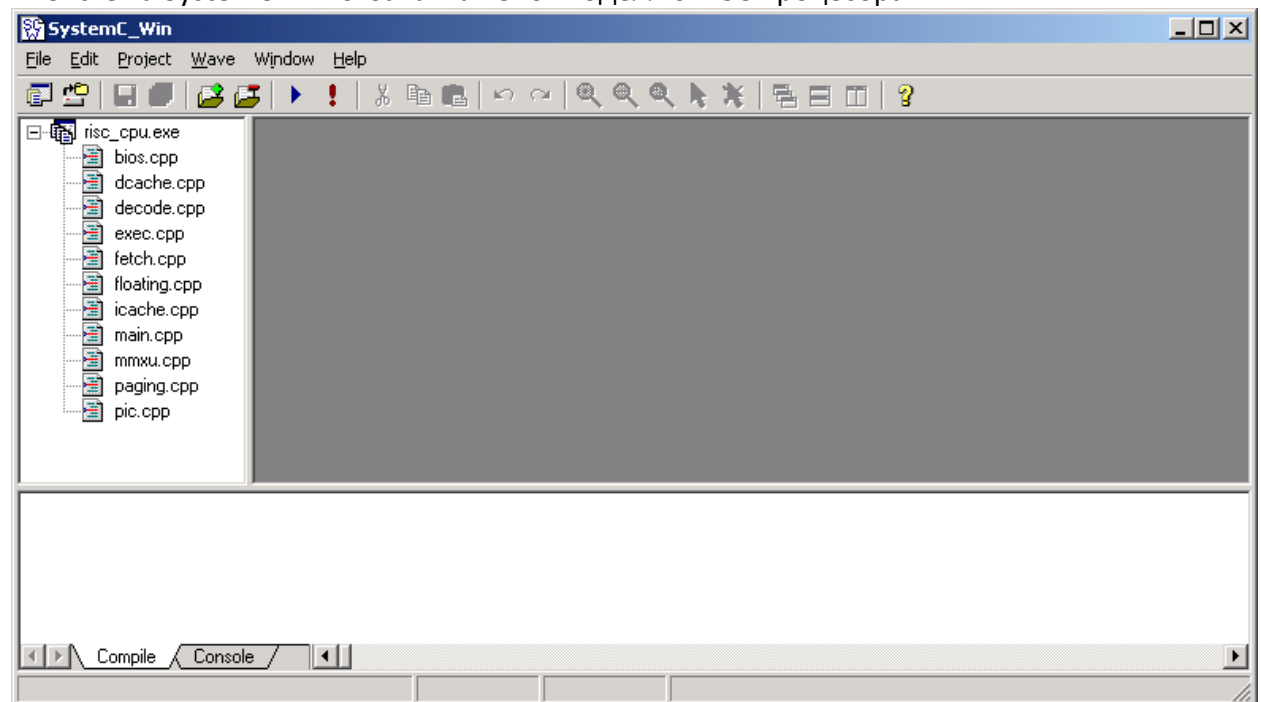


Рис. 86.3 – Завантажений проект *risc_cpu*

Метою компіляції проекту є отримання файлу моделі RISC_CPU.exe з подальшим запуском моделі на виконання як консольної аплікації Windows. Під час виконання модель виводить повідомлення до консольного вікна. З аналізу текстів SystemC моделі та повідомлень роблять висновки щодо коректності поведінки моделі RISC процесора, яка виконує певну процесорну програму. Цю останню програму повинна містити модель ще до її компіляції. Отже, в текстах моделі потрібно винайти та проаналізувати програму для RISC CPU, а вже потім

аналізувати теоретично передбачені та реально отримані моделюванням повідомлення текстового вікна на відповідність бажаній поведінці процесора.

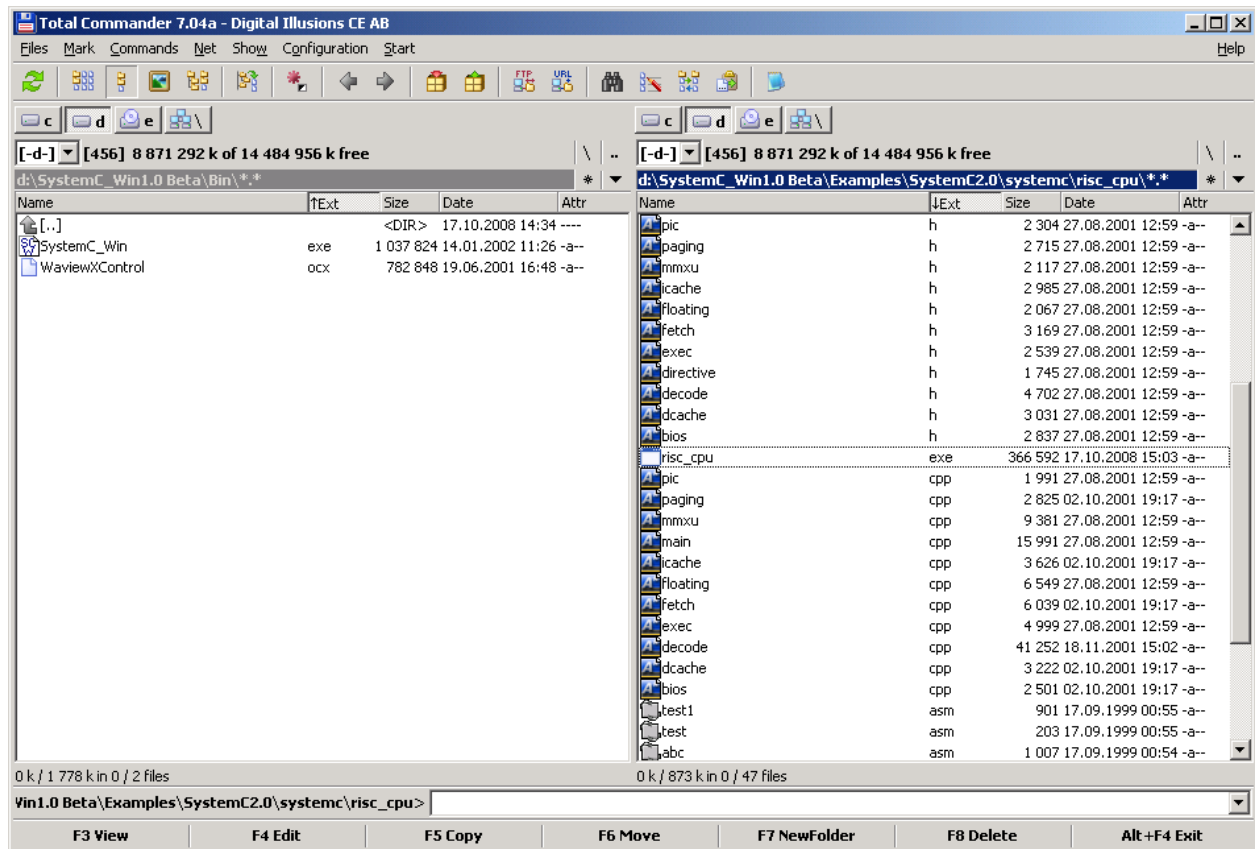


Рис. 86.5 – Скопійована модель (консольна програма `risc_cpu.exe`)


```

: at CSIM 260 ns
-----
ALU : op= 3 A= -1 B= 4
ALU : R= 3-> R2 at CSIM 262 ns
-----
ID: R2=0x3(3) fr ALU at CSIM 263 ns
-----
IFU : mem=0x0
IFU : pc= 24 at CSIM 265 ns
-----
*****
ID: REGISTERS DUMP at CSIM 267 ns
*****
REG :=====
R 0(00000000) R 1(ffff00e1) R 2(00000003) R 3(ffffffff)
R 4(00000004) R 5(00000006) R 6(0000000a) R 7(fcf0fdef)
R 8(00000008) R 9(00000009) R10(00000010) R11(0000ff31)
R12(0000ff12) R13(00000013) R14(00000014) R15(00000015)
R16(00000016) R17(00ffe0117) R18(00ffe0118) R19(00ffe0119)
R20(00ffe0220) R21(00ffe0321) R22(00ffe0322) R23(00ffe0423)
R24(00ff0524) R25(00ff0625) R26(00ff0726) R27(00ff0727)
R28(00ff0728) R29(00000029) R30(00000030) R31(00000031)
=====
IFU : mem=0xffffffff
IFU : pc= 25 at CSIM 272 ns
-----
ID: - SHUTDOWN - at CSIM 274 ns
ID: - PLEASE WAIT ..... -
-----
////////////////////////////////////
SystemC: simulation stopped by user.
Time for simulation = 1

```

Результат “роботи” моделі