Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ



Звіт

з лабораторної роботи № 3

з дисципліни: «Тестування та діагностика програмно-апаратних засобів»

на тему: «Тестування цифрового автомату »

Варіант 3

Виконав:

ст. гр. КІ-301

Порубайміх О.Є.

Перевірив:

Хомуляк М.О.

Львів – 2024

Зміст

[1. Мета 3](#__RefHeading___Toc4821_1098515561)

[2. Теоретична частина 4](#__RefHeading___Toc4823_1098515561)

[3. Індивідуальне завдання 6](#__RefHeading___Toc4825_1098515561)

[4. Виконання завдання 7](#__RefHeading___Toc4827_1098515561)

[4.1. Створення проєкту та модуля. 7](#__RefHeading___Toc4829_1098515561)

[4.2. Реалізація модулів та генерування схем 9](#__RefHeading___Toc4831_1098515561)

[4.3. З’єднав модулі та добавив вхід та виходи. 12](#__RefHeading___Toc4833_1098515561)

[4.4. Створив та реалізував testbench для виявлення помилкі. 13](#__RefHeading___Toc4835_1098515561)

[4.5. Виявлення помилки 14](#__RefHeading___Toc4837_1098515561)

[5. Висновки 15](#__RefHeading___Toc4839_1098515561)

[6. Перелік літератури 16](#__RefHeading___Toc4841_1098515561)

[Додаток 1. VHD файли 17](#__RefHeading___Toc4843_1098515561)

Перелік рисунків

[Рис. 4.1. Створення проєкту 7](#%25252525D0%25252525A0%25252525D0%252525)

[Рис. 4.2. Створення графічного файлу (схеми) з розширенням .sch 7](#%25252525D0%25252525A0%25252525D0%252521)

[Рис. 4.3. Створення текстового файлу (мовою VHDL) з розширенням .vhd для опису справного вузла 8](#%25252525D0%25252525A0%25252525D0%252522)

[Рис. 4.4. Створення VHDL файлу для опису вузла із помилкою 8](#%25252525D0%25252525A0%25252525D0%252523)

[Рис. 4.5. Створення VHDL файлу для опису схеми порівняння 9](#%25252525D0%25252525A0%25252525D0%252524)

[Рис. 4.6. Реалізація справного вузла (DA\_G) 10](#%25252525D0%25252525A0%25252525D0%252526)

[Рис. 4.7. Вигляд DA\_G на схемі 10](#%25252525D0%25252525A0%25252525D0%252527)

[Рис. 4.8. Реалізація вузла із помилкою (DA\_B) 11](#%25252525D0%25252525A0%25252525D0%252528)

[Рис. 4.9. Вигляд DA\_B на схемі 11](#%25252525D0%25252525A0%25252525D0%252529)

[Рис. 4.10. Реалізація схеми порівняння 12](#%25252525D0%25252525A0%25252525D0%25252a)

[Рис. 4.11. Вигляд comparator на схемі 12](#%25252525D0%25252525A0%25252525D0%25252b)

[Рис. 4.12. Вигляд схеми 13](#%25252525D0%25252525A0%25252525D0%25252c)

[Рис. 4.13. Створення генератора тестових послідовностей (TestBench) 13](#%25252525D0%25252525A0%25252525D0%25252d)

[Рис. 4.14. Реалізація testbench.vhd 14](#%25252525D0%25252525A0%25252525D0%25252e)

[Рис. 4.15. Результат симуляції – часова діаграмма. 14](#%25252525D0%25252525A0%25252525D0%25252f)

# Мета

Ознайомлення із загальною схемою тестування цифрової техніки. Засвоєння методів та засобів тестування цифрових схем з пам’яттю на прикладі цифрового автомата (ЦА).

# Теоретична частина

Особливості тестування цифрових автоматів

Цифровий автомат є ключовим елементом сучасних цифрових систем і призначений для виконання визначеного алгоритму обробки інформації залежно від поточного стану та вхідних сигналів. Основними характеристиками цифрового автомата є набір вхідних і вихідних сигналів, набір внутрішніх станів, початковий стан, а також правила формування вихідних сигналів і наступних станів.

**Типи цифрових автоматів**

Залежно від логіки роботи, цифрові автомати поділяються на автомати Мура та Мілі. У автоматах Мура вихідні сигнали визначаються виключно поточним станом автомата, що робить їх прогнозованими та легкими для аналізу. Водночас, автомати Мілі враховують як поточний стан, так і вхідні сигнали, що надає їм більшої гнучкості у проектуванні складних систем.

**Методи тестування**

Тестування цифрових автоматів є критичним етапом в процесі розробки, яке забезпечує відповідність реалізації заданим специфікаціям. Тестування допомагає виявити помилки у логіці роботи автоматів, невідповідності у формуванні вихідних сигналів та переходів між станами. Основні методи тестування включають:

* **Табличний метод**, який передбачає використання таблиць станів і переходів, дозволяє визначити відповідність між вхідними сигналами, поточними станами та очікуваними виходами.
* **Часові діаграми** демонструють динаміку зміни станів і сигналів в часі, сприяючи кращому розумінню процесів в автоматі.
* **Аналітичний метод** застосовується для математичного опису поведінки автомата, включно з використанням мов опису апаратури, таких як VHDL або Verilog.
* **Моделювання алгоритму роботи автомата** дозволяє візуалізувати його поведінку за різних вхідних умов.
* **Використання графів** станів для наглядного представлення переходів між станами та визначення можливих шляхів роботи автомата.

Розвиток технологій і зростання складності цифрових систем вимагають від методів тестування цифрових автоматів не тільки високої точності, але й гнучкості у виявленні та діагностиці потенційних проблем. Одним з передових напрямків є використання **формальної верифікації**. Цей метод базується на математичному доведенні правильності роботи автоматів відповідно до їх специфікації, що дозволяє виявити помилки, які можуть залишитися непоміченими при класичних підходах тестування.

Ще один важливий метод – **динамічна перевірка**, яка включає в себе створення тестових середовищ, що імітують реальні умови експлуатації автомата. Такий підхід дозволяє не тільки перевірити правильність роботи в стандартних ситуаціях, але й оцінити поведінку системи при крайніх значеннях вхідних сигналів, спонтанних перериваннях, збоях живлення та інших нестандартних умовах.

**Застосування**

Цифрові автомати знаходять широке застосування у різноманітних галузях, включно з автоматизованими системами управління, телекомунікаціями, електронікою та комп'ютерною технікою. Вони є основою для мікропроцесорів, контролерів, систем цифрової обробки сигналів та інших складових сучасних цифрових систем.

Тестування цифрових автоматів є важливим етапом проектування, що забезпечує надійність та ефективність функціонування цифрових систем. Використання різноманітних методів тестування дозволяє глибоко аналізувати поведінку автоматів та оптимізувати їх роботу для задоволення специфічних вимог проекту.

# Індивідуальне завдання

Змоделювати діаграму станів та протестувати її при всіх можливих комбінаціях.

3 Варіант: Закоротка вхідного сигналу на живлення +5В

# Виконання завдання

## Створення проєкту та модуля.

Створюю проєкт Рис. 4.1 та налаштовую його параметри. Після цього створюю схему Рис. 4.2, працюючиий модуль Рис. 4.3, модуль з помилкою Рис. 4.4 та модуль порівняння Рис. 4.5.

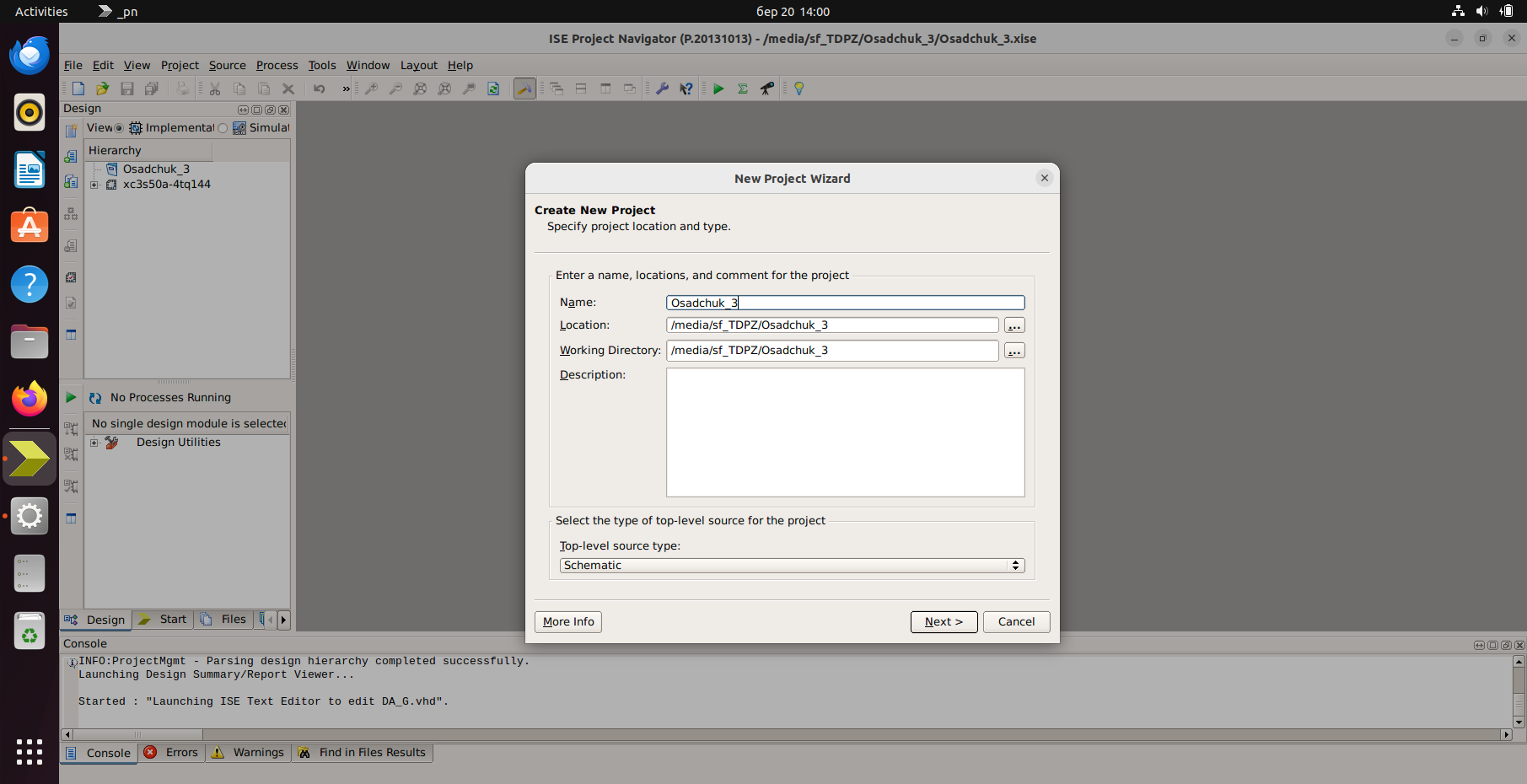


Рис. 4.1. Створення проєкту

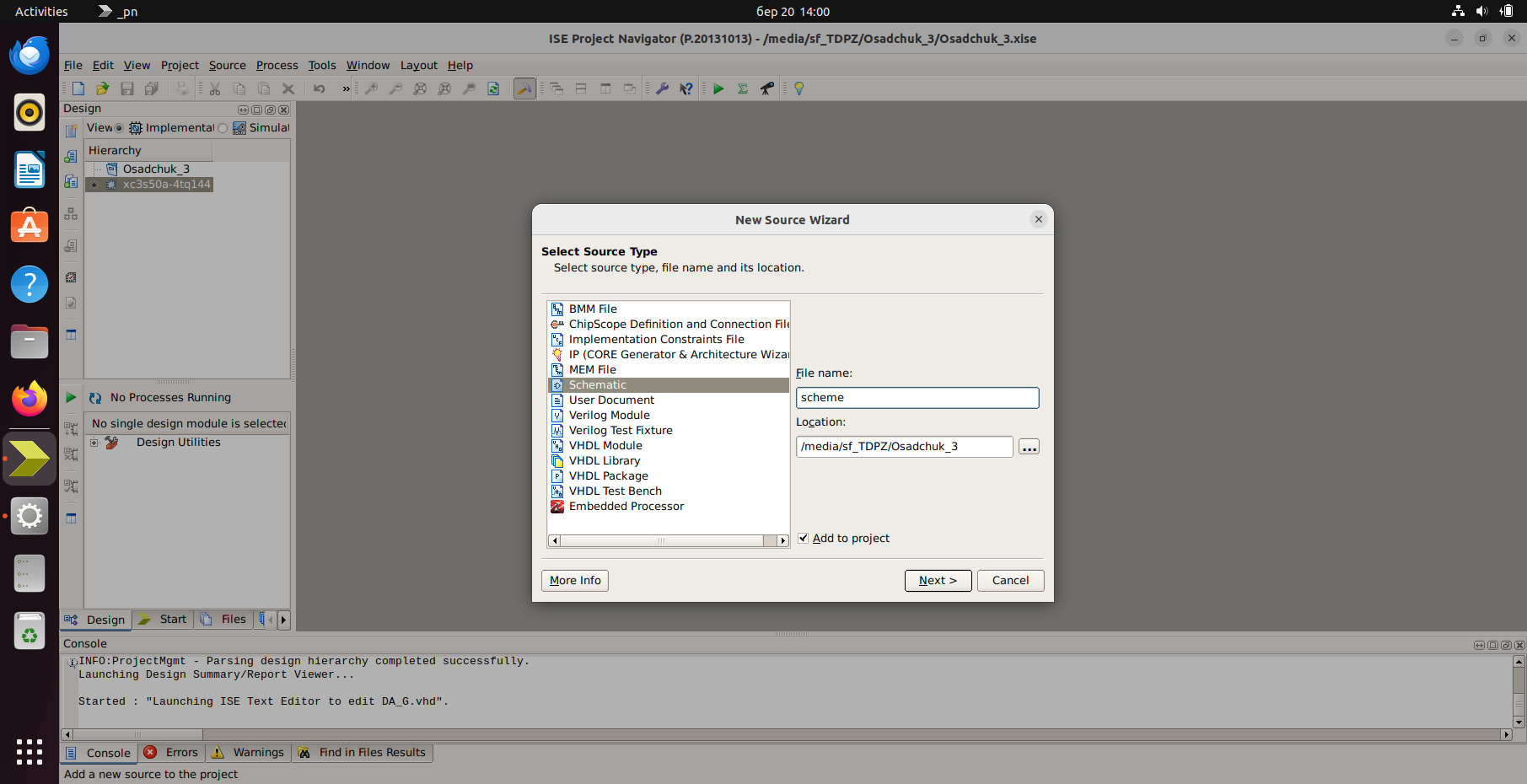


Рис. 4.2. Створення графічного файлу (схеми) з розширенням .sch

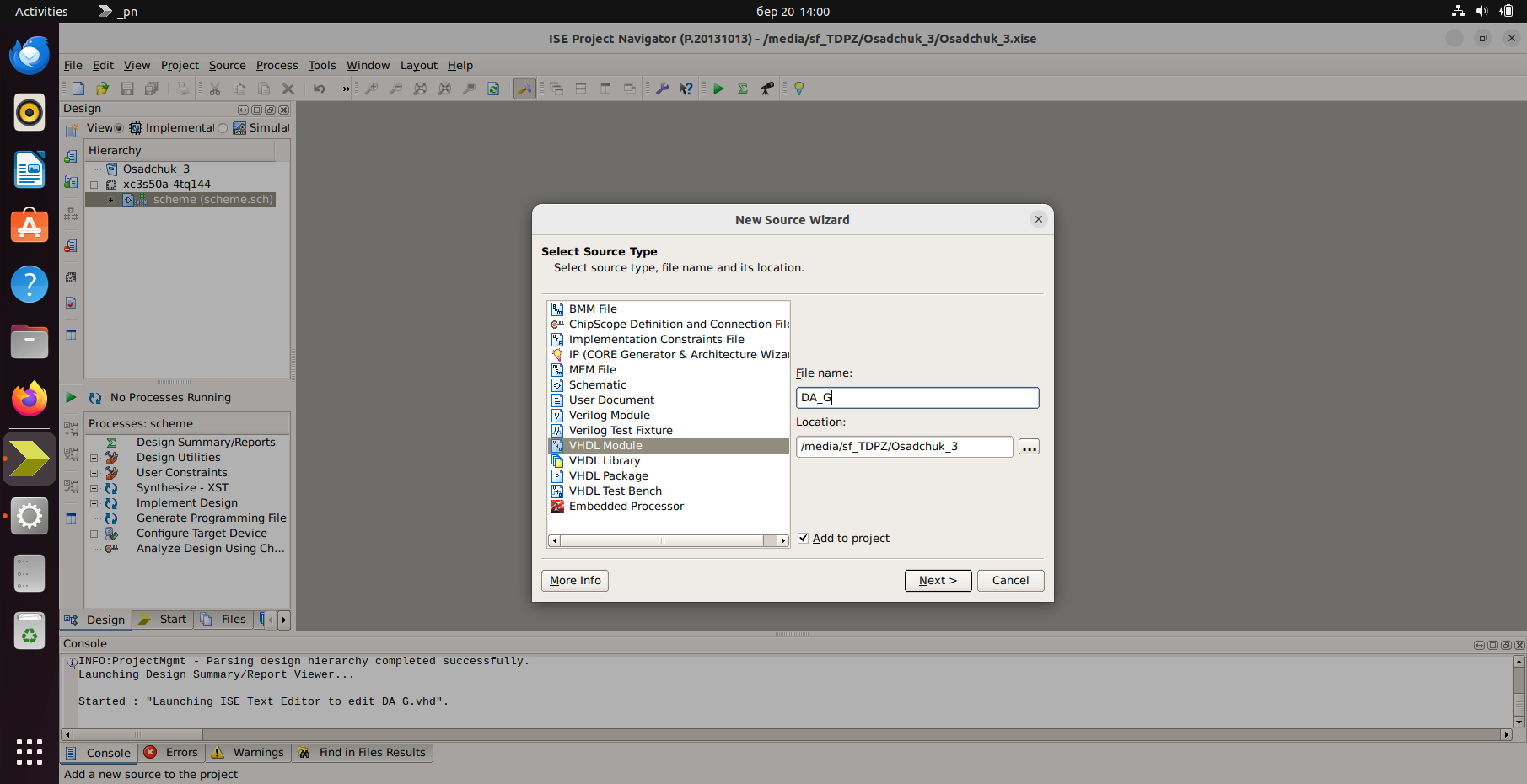


Рис. 4.3. Створення текстового файлу (мовою VHDL) з розширенням .vhd для опису справного вузла

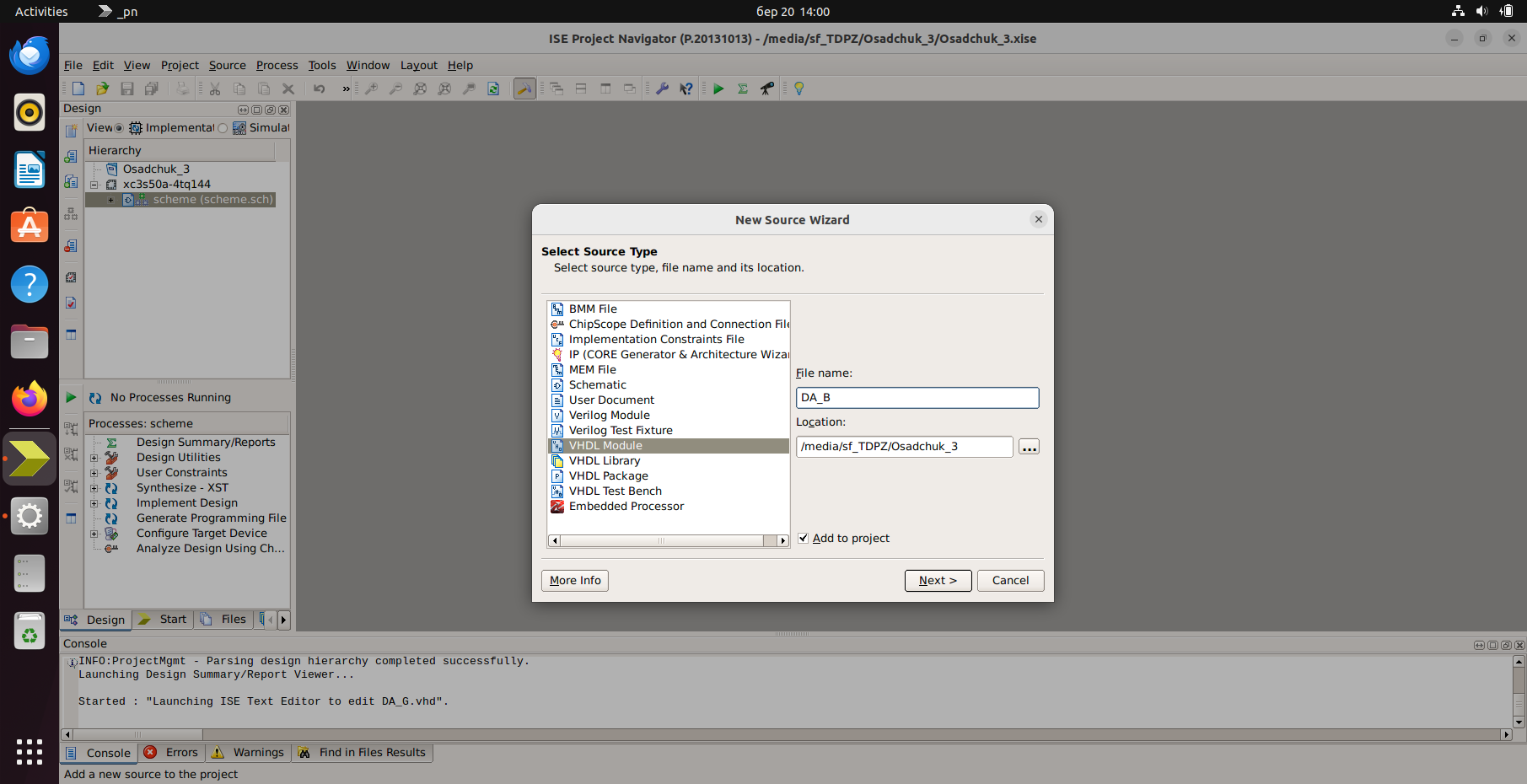


Рис. 4.4. Створення VHDL файлу для опису вузла із помилкою

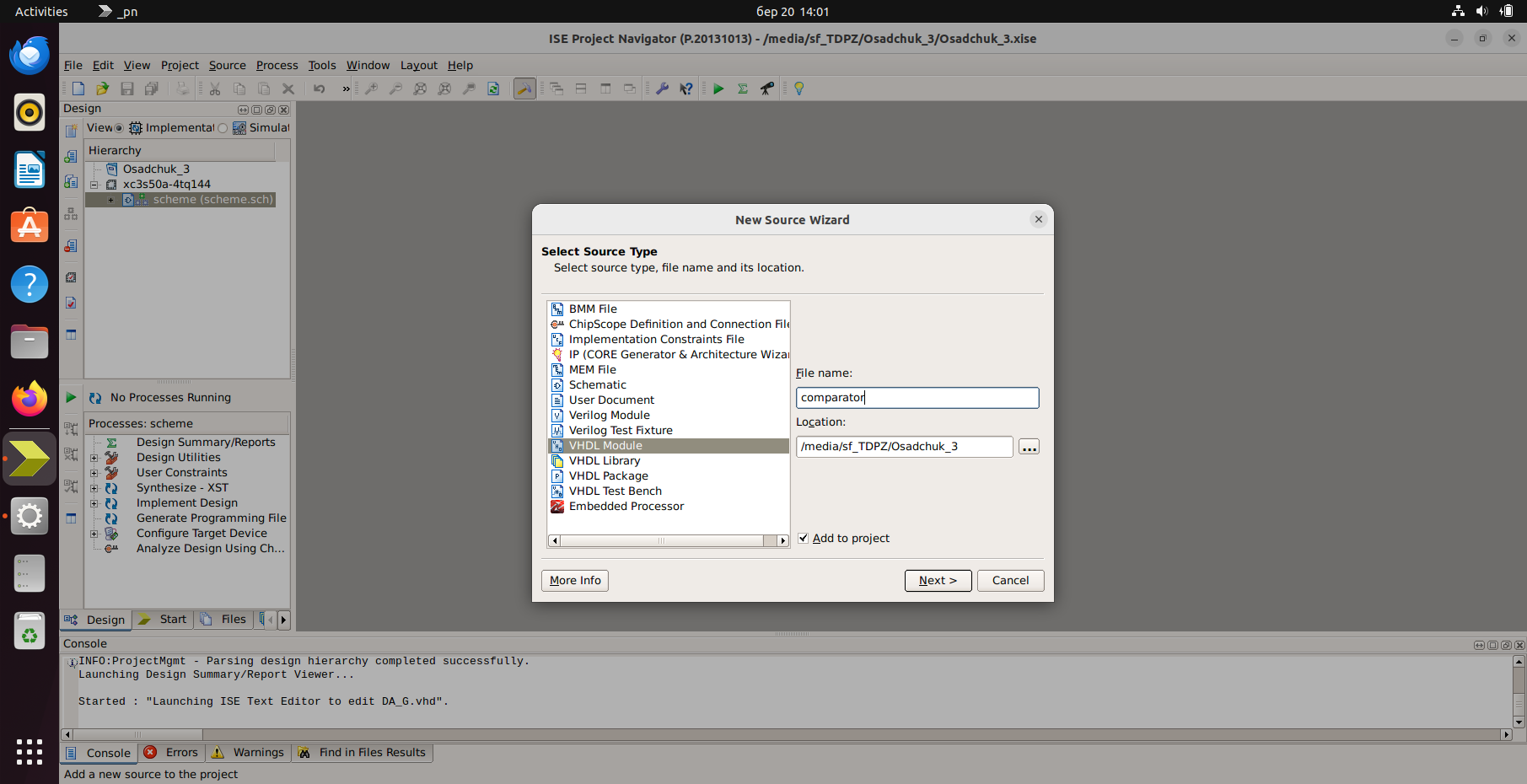


Рис. 4.5. Створення VHDL файлу для опису схеми порівняння

## Реалізація модулів та генерування схем

Написав код для еталоного модуля [(див. Рис. 4.6.)](#Рисунок_6|graphic) та добавив його на схему [(див. Рис. 4.7.)](#_toc146). В модулі з помилкою симулював закорочення вхідного сигналу на живлення +5В [(див. Рис. 4.8.)](#Рисунок_7|graphic) та добавив його на схему [(див. Рис. 4.9.).](#Рисунок_7|graphic) В модулі comparator порівняв виходи [(див. Рис. 4.10.)](#_toc158) та добавив вузол на схему [(див. Рис. 4.11.)](#_toc154).

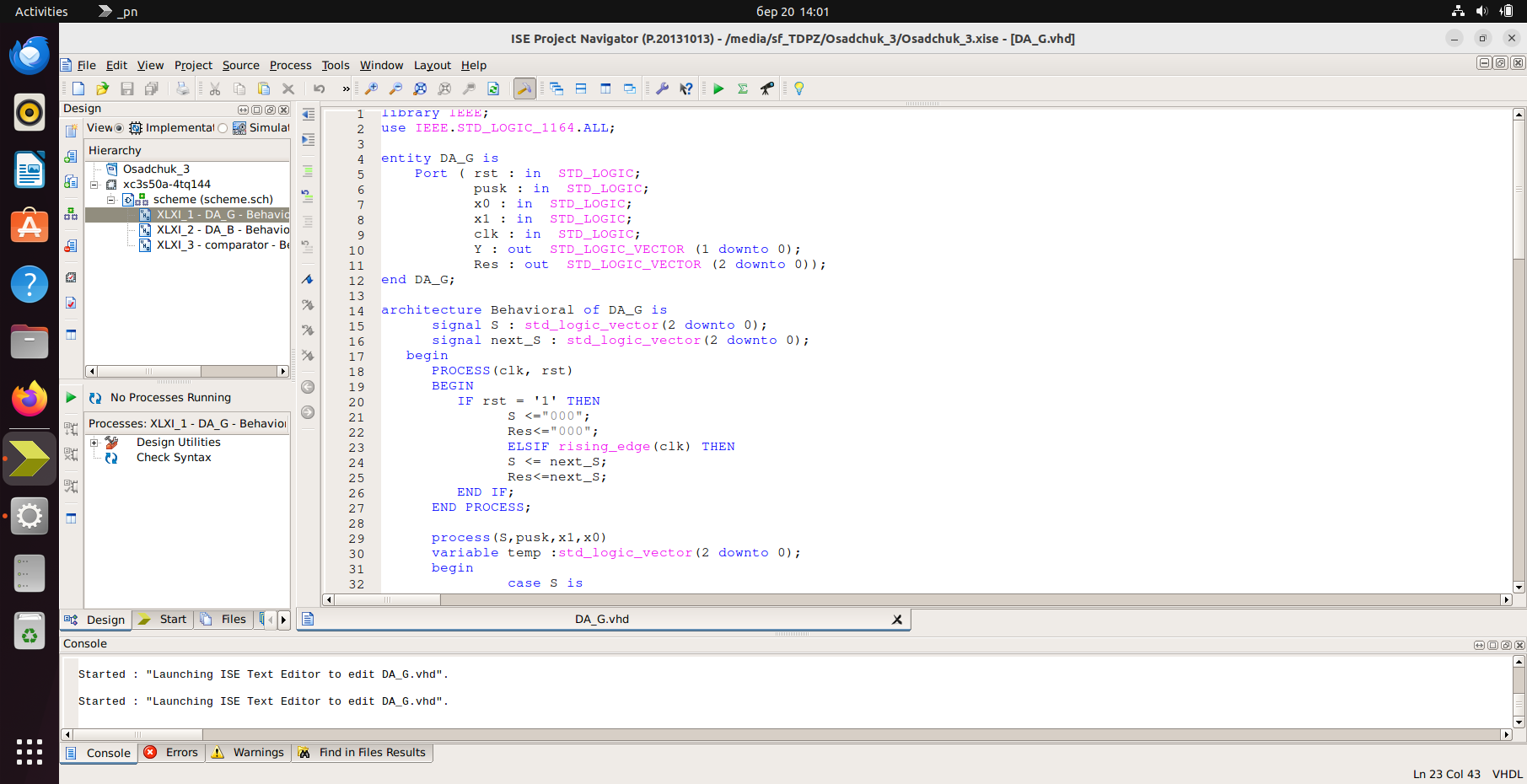


Рис. 4.6. Реалізація справного вузла (DA\_G)

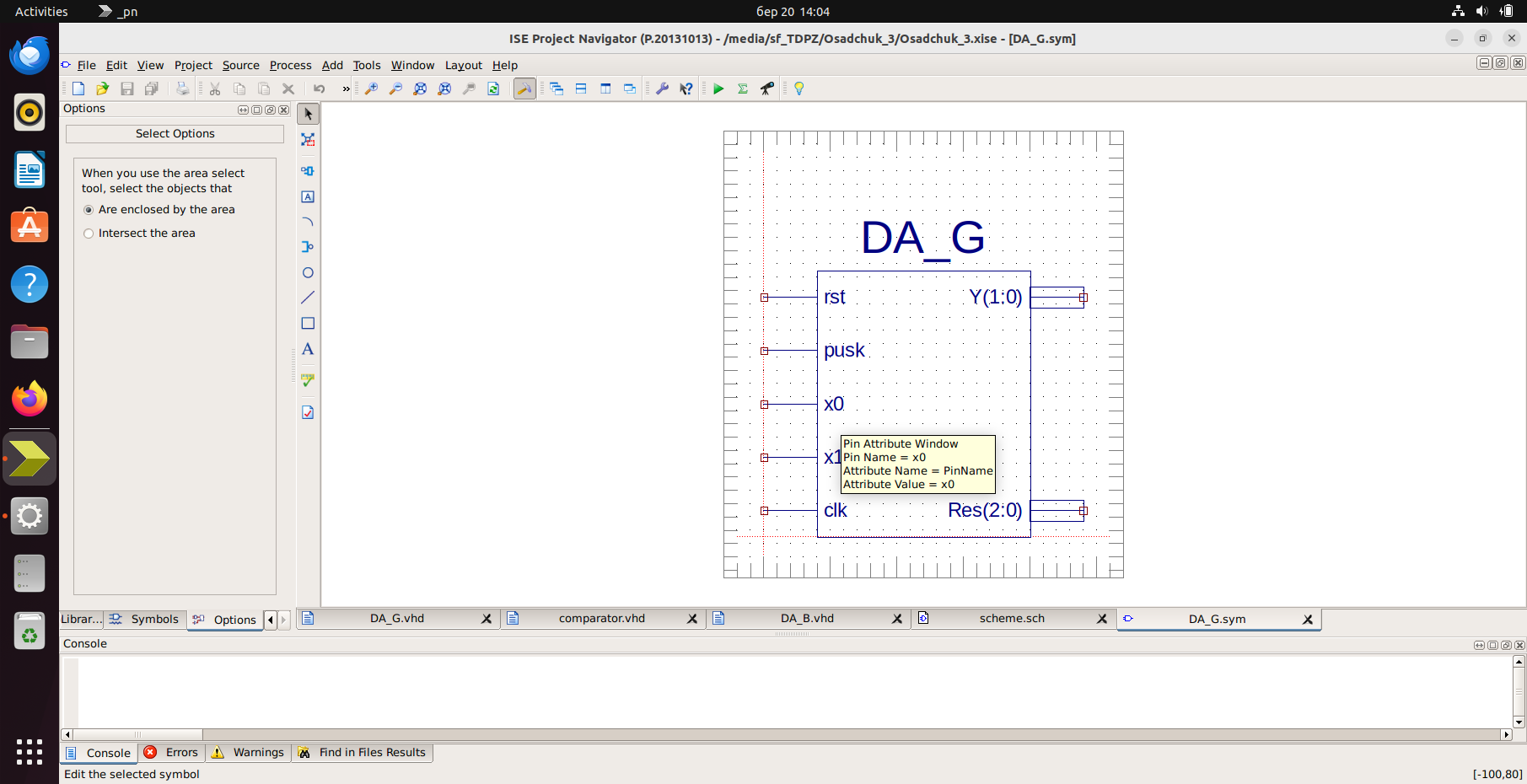


Рис. 4.7. Вигляд DA\_G на схемі

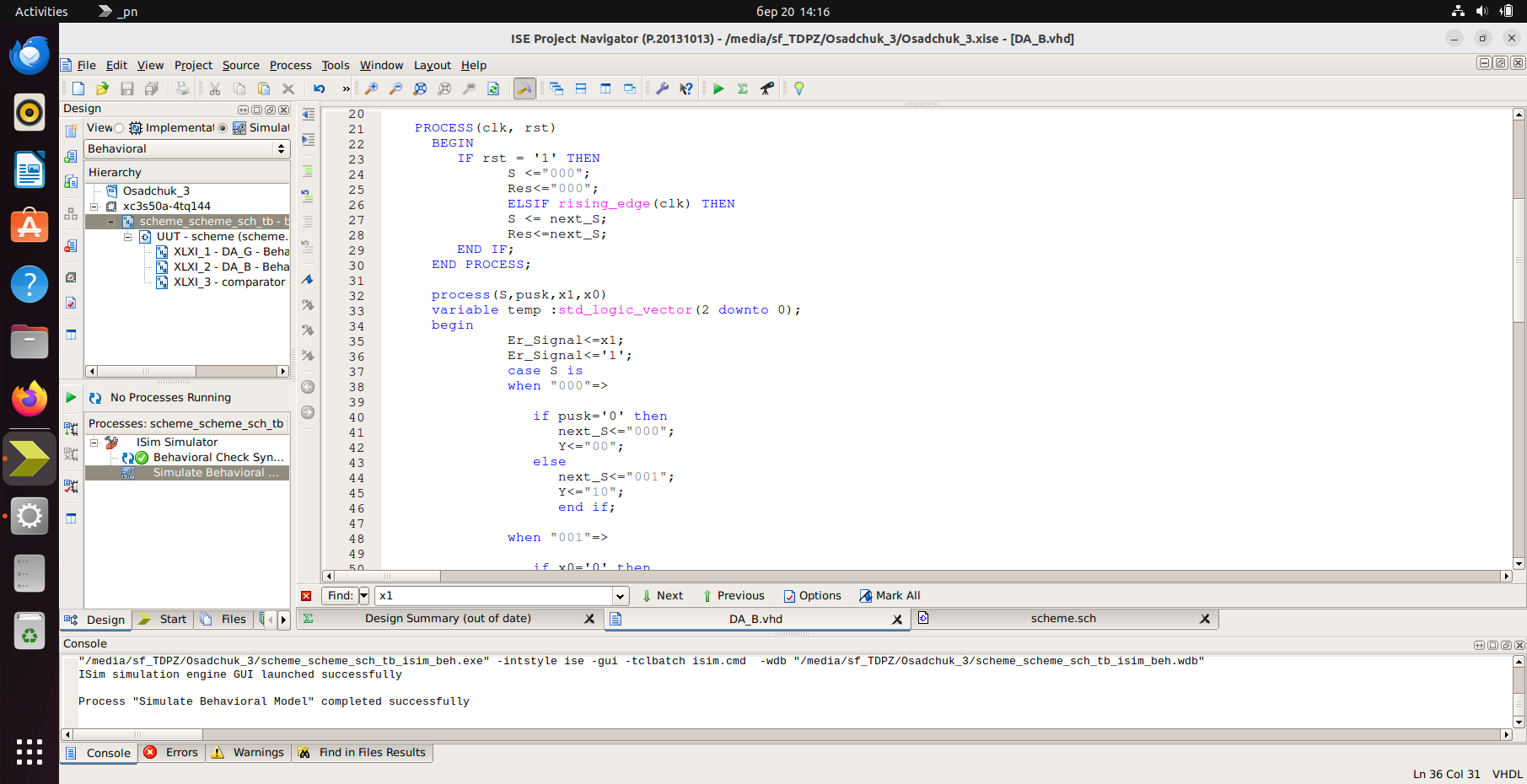


Рис. 4.8. Реалізація вузла із помилкою (DA\_B)

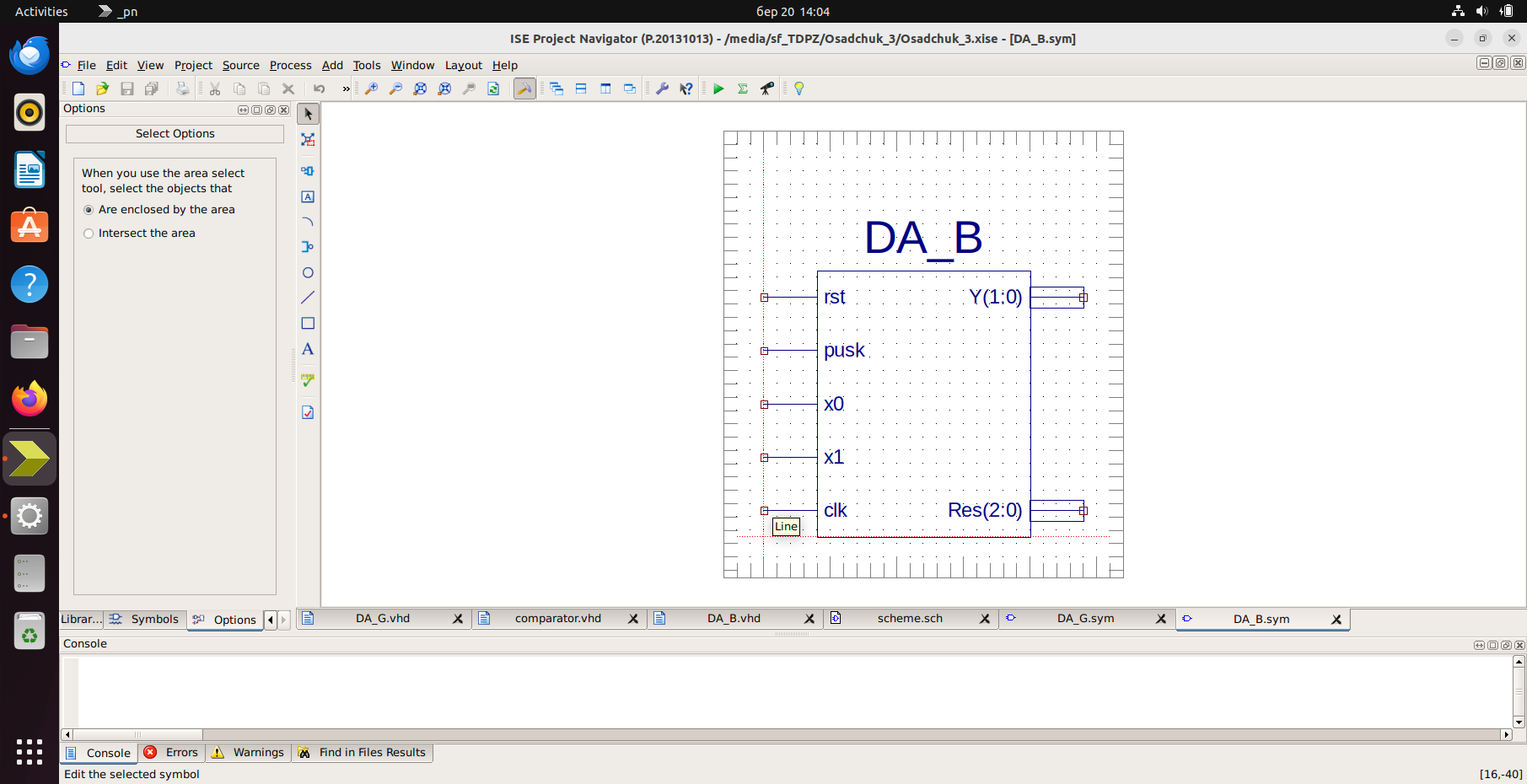


Рис. 4.9. Вигляд DA\_B на схемі

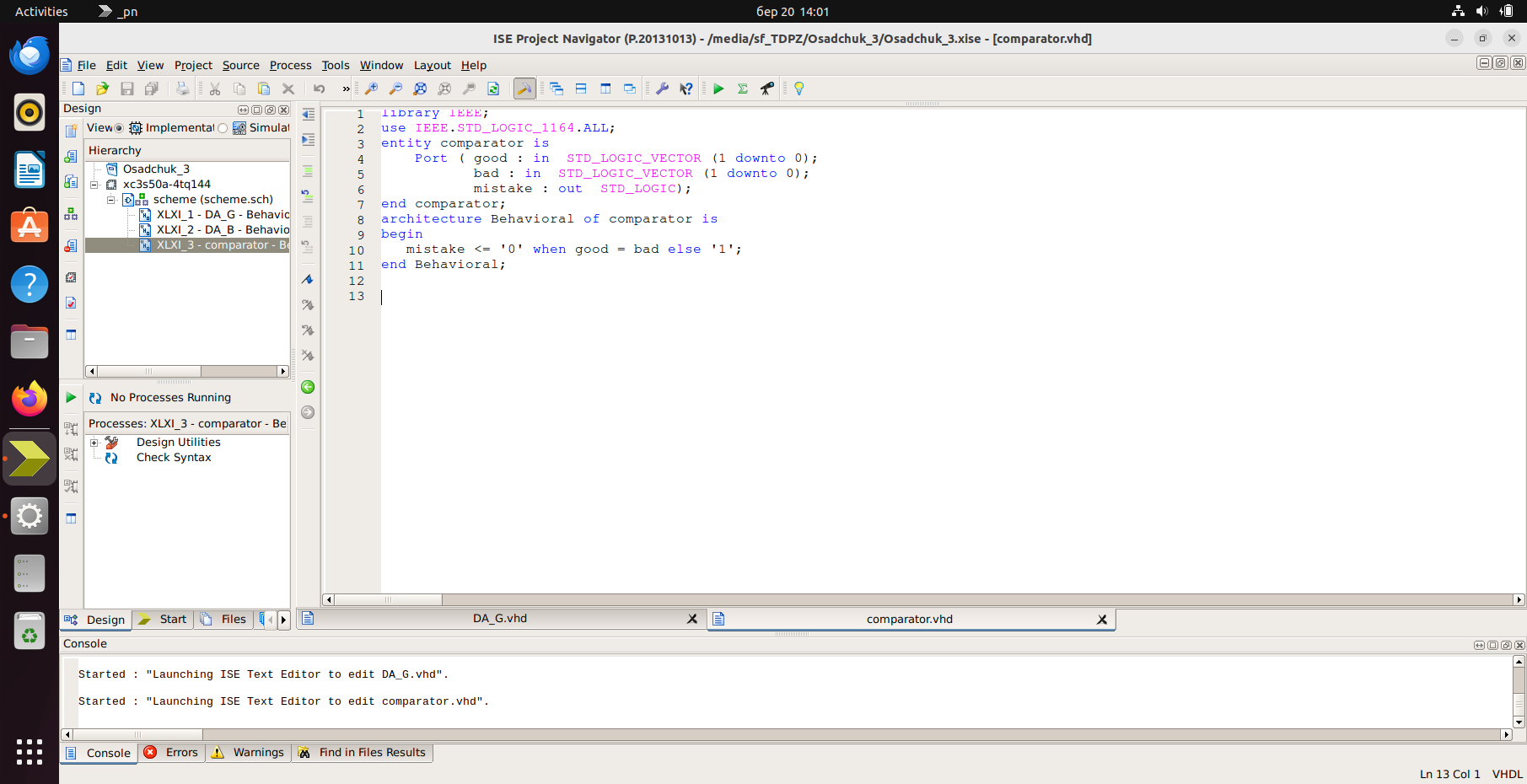


Рис. 4.10. Реалізація схеми порівняння

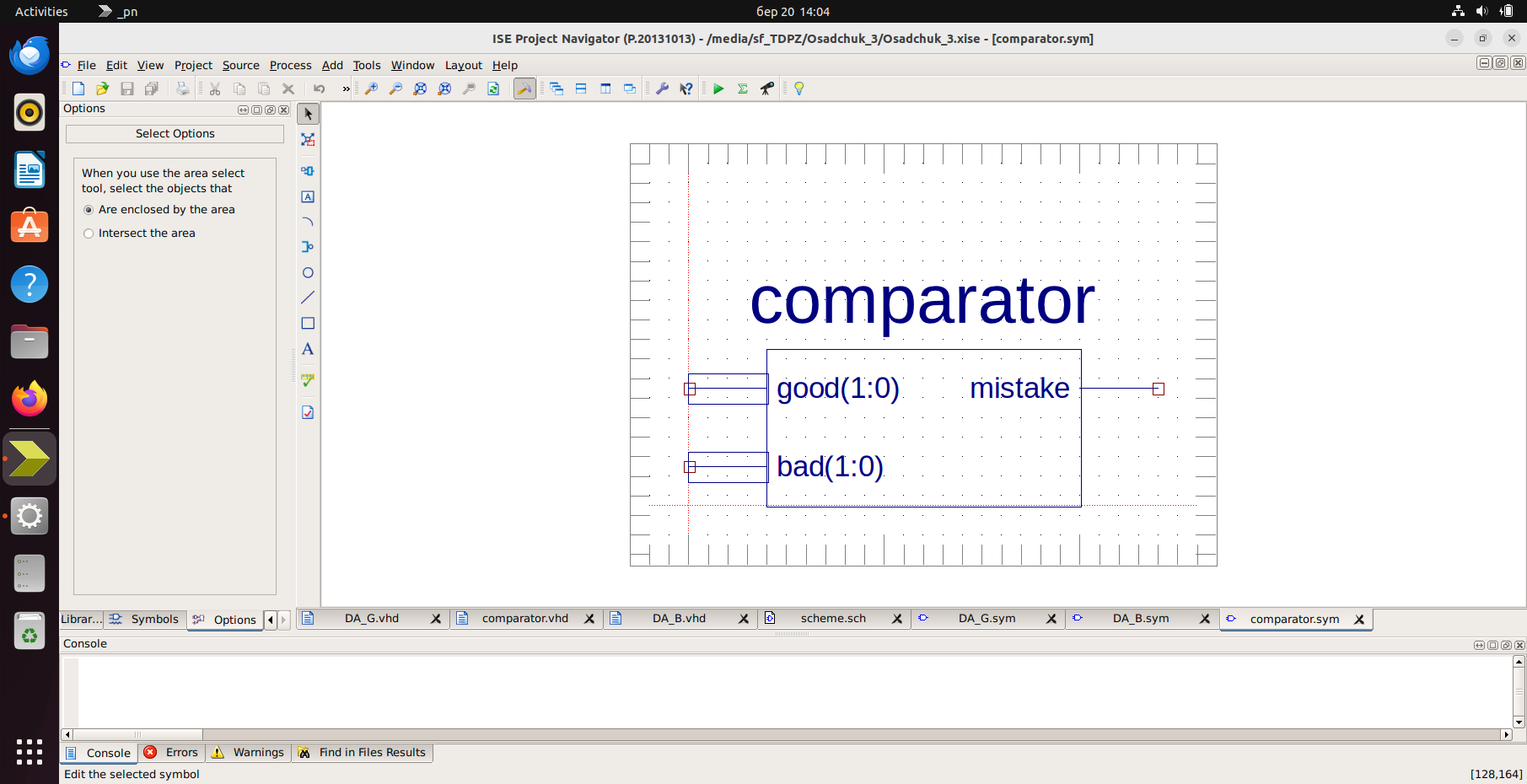


Рис. 4.11. Вигляд comparator на схемі

## З’єднав модулі та добавив вхід та виходи.

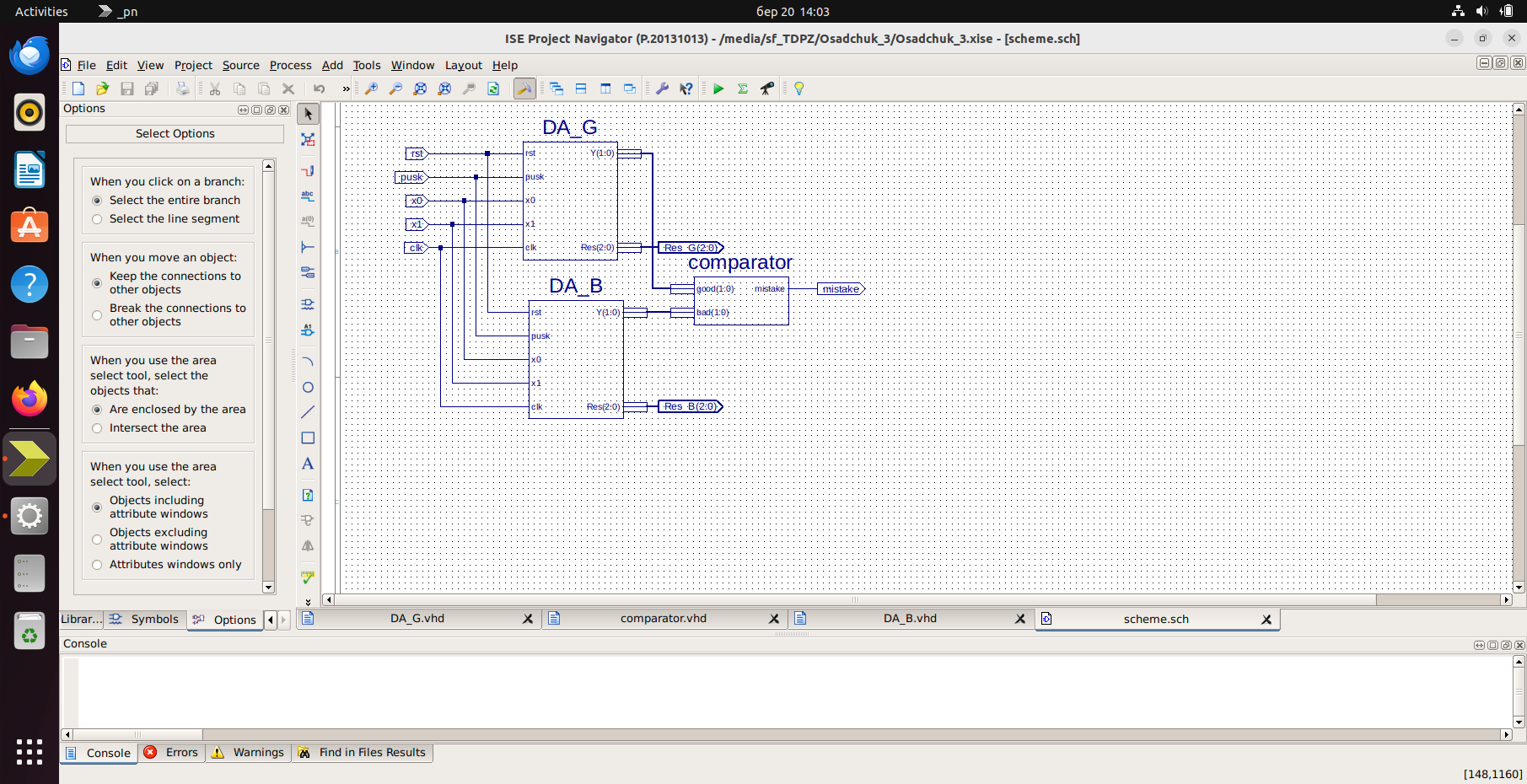


Рис. 4.12. Вигляд схеми

## Створив та реалізував testbench для виявлення помилкі.

В testbench прописав тестову послідовність [(див. Рис. 4.15.)](#_toc162), за допомогою якого і виявлю помилку та оставив паузу в 15 ns для більш зручного виявлення на часовій діаграмі.

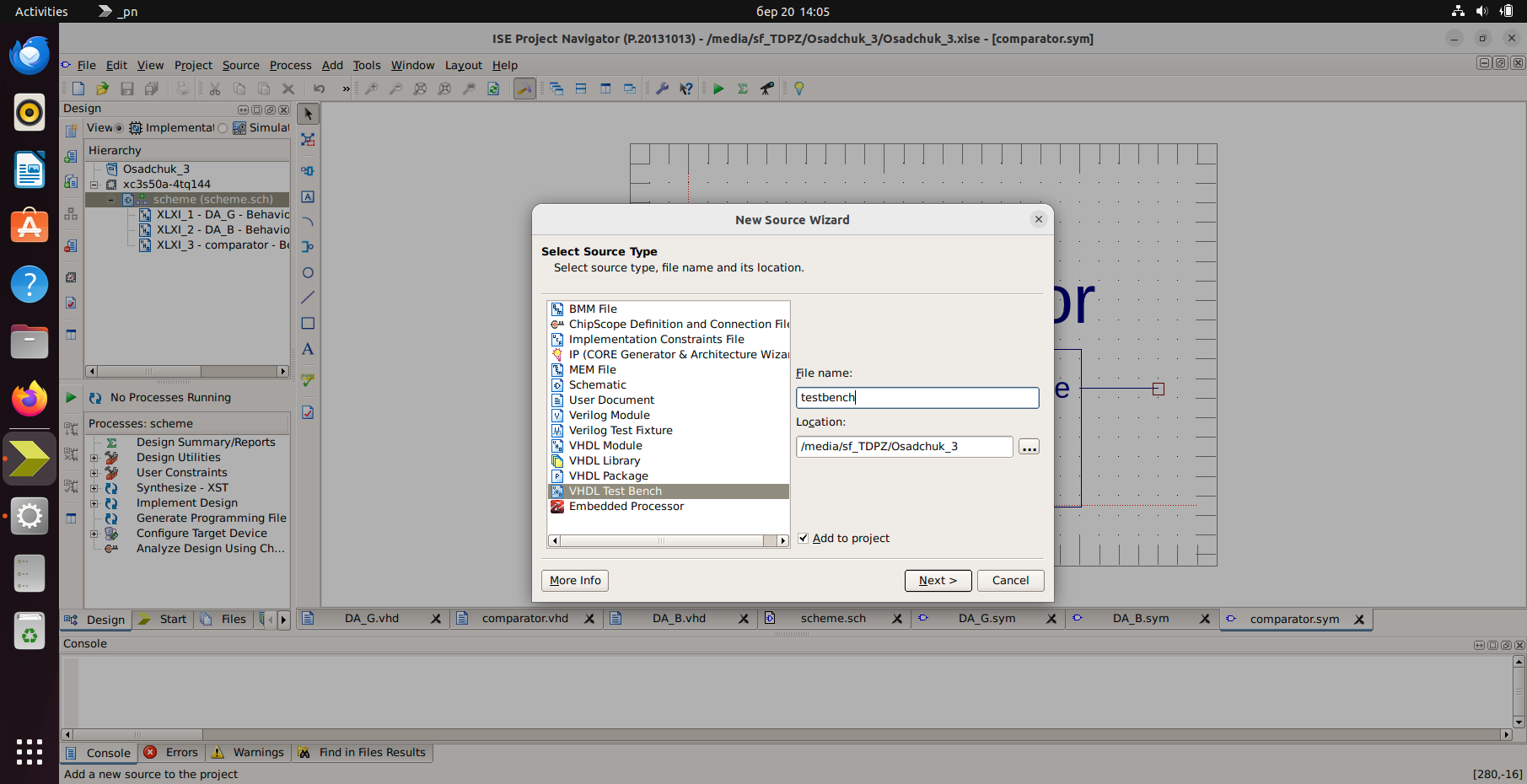


Рис. 4.13. Створення генератора тестових послідовностей (TestBench)

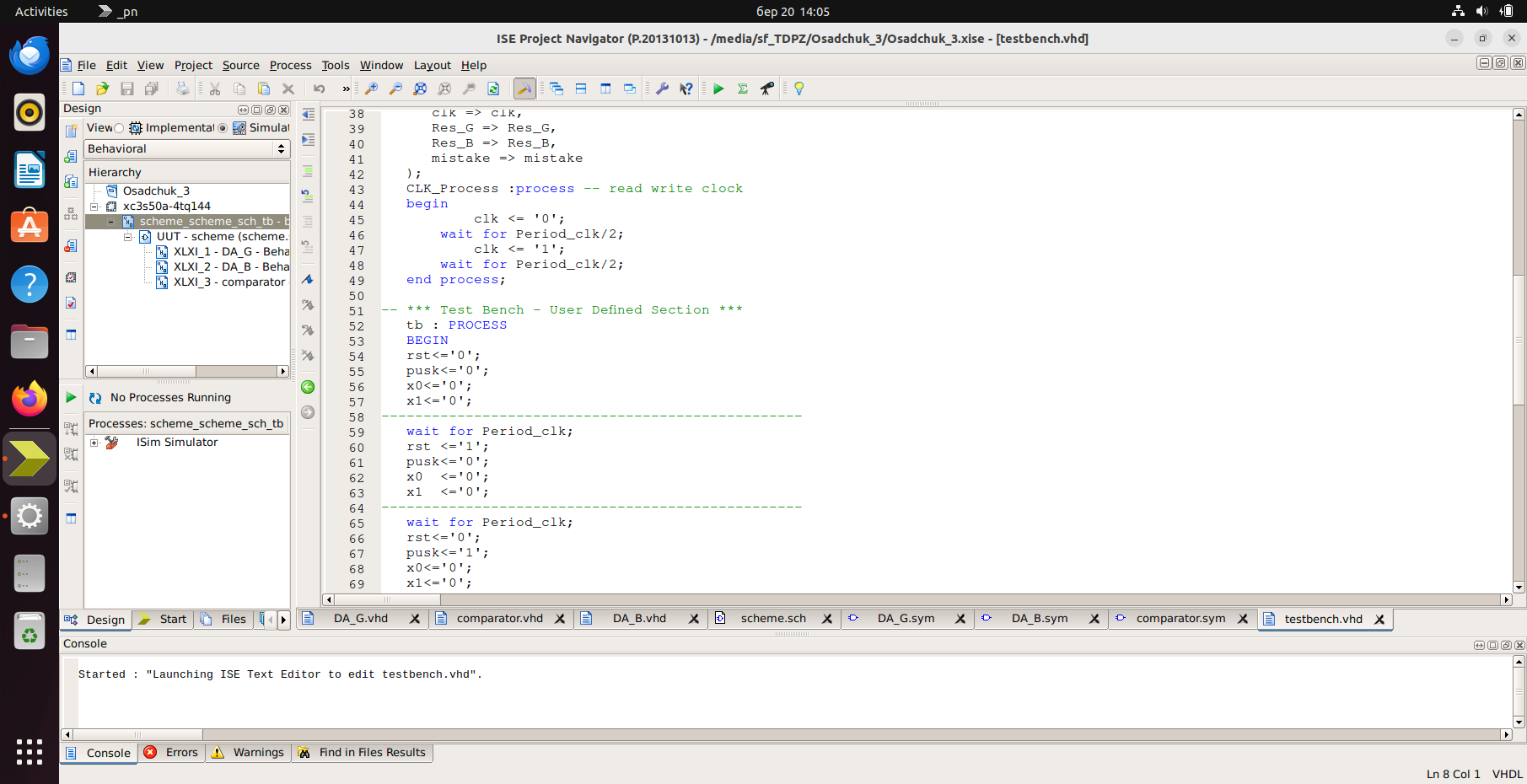


Рис. 4.14. Реалізація testbench.vhd

## Виявлення помилки

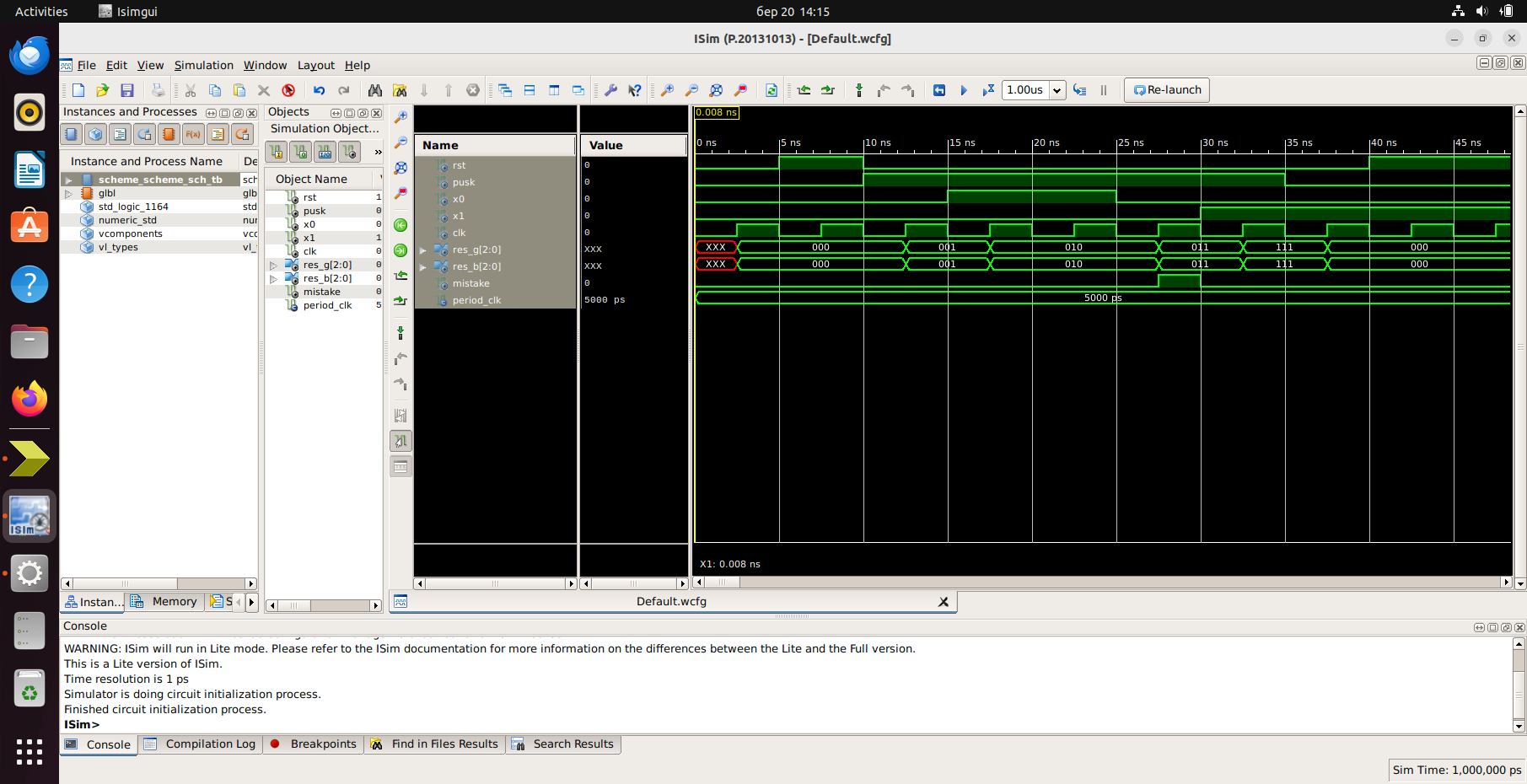


Рис. 4.15. Результат симуляції – часова діаграмма.

# Висновки

Ознайомився із загальною схемою тестування цифрових пристроїв та методики тестування цифрових схем з пам’яттю через призму оперативного запам'ятовувального пристрою (ОЗП). Провів симуляцію "Закоротка вхідного сигналу на живлення +5В" і використав тестову послідовність для ідентифікації проблем.

# Перелік літератури

1. Тестування і діагностика програмно-апаратних засобів : лабораторний практикум для студентів спеціальності 123 “Комп’ютерна інженерія” / В. С. Глухов, М. О. Хомуляк, Г. В. Бойко, І. М. Жолубак. – Львів : Видавництво Національного університету “Львівська політехніка”, 2021. – 120 с

2. ISE In-Depth Tutorial <https://docs.xilinx.com/v/u/en-US/ise_tutorial_ug695> 25.10.2023

1. VHD файли

DA\_G.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity DA\_G is

Port ( rst : in STD\_LOGIC;

pusk : in STD\_LOGIC;

x0 : in STD\_LOGIC;

x1 : in STD\_LOGIC;

clk : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (1 downto 0);

Res : out STD\_LOGIC\_VECTOR (2 downto 0));

end DA\_G;

architecture Behavioral of DA\_G is

signal S : std\_logic\_vector(2 downto 0);

signal next\_S : std\_logic\_vector(2 downto 0);

begin

PROCESS(clk, rst)

BEGIN

IF rst = '1' THEN

S <="000";

Res<="000";

ELSIF rising\_edge(clk) THEN

S <= next\_S;

Res<=next\_S;

END IF;

END PROCESS;

process(S,pusk,x1,x0)

variable temp :std\_logic\_vector(2 downto 0);

begin

case S is

when "000"=>

if pusk='0' then

next\_S<="000";

Y<="00";

else

next\_S<="001";

Y<="10";

end if;

when "001"=>

if x0='0' then

next\_S<="101";

Y<="10";

else

next\_S<="010";

Y<="01";

end if;

when "010"=>

if x0='0' then

next\_S<="011";

Y<="01";

else

next\_S<="010";

Y<="01";

end if;

when "011"=>

if x1='0' then

next\_S<="100";

Y<="00";

else

next\_S<="111";

Y<="01";

end if;

when "100"=>

next\_S<="101";

Y<="10";

when "101"=>

next\_S<="110";

Y<="00";

when "110"=>

if x1='0' then

next\_S<="010";

Y<="01";

else

next\_S<="111";

Y<="01";

end if;

when "111"=>

if pusk='0' then

next\_S<="000";

Y<="00";

else

next\_S<="111";

Y<="01";

end if;

when others =>

next\_S<="000";

Y<="00";

end case;

end process;

end Behavioral;

DA\_B.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity DA\_B is

Port ( rst : in STD\_LOGIC;

pusk : in STD\_LOGIC;

x0 : in STD\_LOGIC;

x1 : in STD\_LOGIC;

clk : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (1 downto 0);

Res : out STD\_LOGIC\_VECTOR (2 downto 0));

end DA\_B;

architecture Behavioral of DA\_B is

signal S: std\_logic\_vector(2 downto 0);

signal next\_S : std\_logic\_vector(2 downto 0);

signal Er\_Signal : std\_logic;

begin

PROCESS(clk, rst)

BEGIN

IF rst = '1' THEN

S <="000";

Res<="000";

ELSIF rising\_edge(clk) THEN

S <= next\_S;

Res<=next\_S;

END IF;

END PROCESS;

process(S,pusk,x1,x0)

variable temp :std\_logic\_vector(2 downto 0);

begin

Er\_Signal<=x1;

Er\_Signal<='0';

case S is

when "000"=>

if pusk='0' then

next\_S<="000";

Y<="00";

else

next\_S<="001";

Y<="10";

end if;

when "001"=>

if x0='0' then

next\_S<="101";

Y<="10";

else

next\_S<="010";

Y<="01";

end if;

when "010"=>

if x0='0' then

next\_S<="011";

Y<="01";

else

next\_S<="010";

Y<="01";

end if;

when "011"=>

if x1='0' then

next\_S<="100";

Y<="00";

else

next\_S<="111";

Y<="01";

end if;

when "100"=>

next\_S<="101";

Y<="10";

when "101"=>

next\_S<="110";

Y<="00";

when "110"=>

if Er\_Signal='0' then

next\_S<="010";

Y<="01";

else

next\_S<="111";

Y<="01";

end if;

when "111"=>

if pusk='0' then

next\_S<="000";

Y<="00";

else

next\_S<="111";

Y<="01";

end if;

when others =>

next\_S<="000";

Y<="00";

end case;

end process;

end Behavioral;

COMPARATOR.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity comparator is

Port ( good : in STD\_LOGIC\_VECTOR (1 downto 0);

bad : in STD\_LOGIC\_VECTOR (1 downto 0);

mistake : out STD\_LOGIC);

end comparator;

architecture Behavioral of comparator is

begin

mistake <= '0' when good = bad else '1';

end Behavioral;

TESTBENCH.vhd

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

LIBRARY UNISIM;

USE UNISIM.Vcomponents.ALL;

ENTITY scheme\_scheme\_sch\_tb IS

END scheme\_scheme\_sch\_tb;

ARCHITECTURE behavioral OF scheme\_scheme\_sch\_tb IS

COMPONENT scheme

PORT( rst : IN STD\_LOGIC;

pusk : IN STD\_LOGIC;

x0 : IN STD\_LOGIC;

x1 : IN STD\_LOGIC;

clk : IN STD\_LOGIC;

Res\_G : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0);

Res\_B : OUT STD\_LOGIC\_VECTOR (2 DOWNTO 0);

mistake : OUT STD\_LOGIC);

END COMPONENT;

SIGNAL rst : STD\_LOGIC;

SIGNAL pusk : STD\_LOGIC;

SIGNAL x0 : STD\_LOGIC;

SIGNAL x1 : STD\_LOGIC;

SIGNAL clk : STD\_LOGIC;

SIGNAL Res\_G : STD\_LOGIC\_VECTOR (2 DOWNTO 0);

SIGNAL Res\_B : STD\_LOGIC\_VECTOR (2 DOWNTO 0);

SIGNAL mistake : STD\_LOGIC;

constant Period\_clk : time := 5 ns;

BEGIN

UUT: scheme PORT MAP(

rst => rst,

pusk => pusk,

x0 => x0,

x1 => x1,

clk => clk,

Res\_G => Res\_G,

Res\_B => Res\_B,

mistake => mistake

);

CLK\_Process :process -- read write clock

begin

clk <= '0';

wait for Period\_clk/2;

clk <= '1';

wait for Period\_clk/2;

end process;

-- \*\*\* Test Bench - User Defined Section \*\*\*

tb : PROCESS

BEGIN

rst<='0';

pusk<='0';

x0<='0';

x1<='0';

--------------------------------------------------

wait for Period\_clk;

rst <='1';

pusk<='0';

x0 <='0';

x1 <='0';

--------------------------------------------------

wait for Period\_clk;

rst<='0';

pusk<='1';

x0<='0';

x1<='0';

--------------------------------------------------

wait for Period\_clk;

rst<='0';

pusk<='1';

x0<='1';

x1<='0';

--------------------------------------------------

wait for Period\_clk;

rst<='0';

pusk<='1';

x0<='1';

x1<='0';

--------------------------------------------------

wait for Period\_clk;

rst<='0';

pusk<='1';

x0<='0';

x1<='0';

--------------------------------------------------

wait for Period\_clk;

rst<='0';

pusk<='1';

x0<='0';

x1<='1';

--------------------------------------------------

wait for Period\_clk;

rst<='0';

pusk<='0';

x0<='0';

x1<='1';

--------------------------------------------------

wait for Period\_clk;

rst<='1';

pusk<='0';

x0<='0';

WAIT; -- will wait forever

END PROCESS;

-- \*\*\* End Test Bench - User Defined Section \*\*\*

END;