# Лабораторна робота № 1

## Реалізація логічних функцій.

### Мета: навчитися реалізовувати логічну функцію за допомогою схемотехнічного редактора та мови Verilog, побудова найпростішого тесту.

#### 1.1 Побудова логічної функції за таблицею істиності та її реалізація

Запишіть дату свого народження у двійковій формі у форматі день (а0, а1, а2, а3, а4) та місяць (а5, а6, а7, а8).

*Наприклад, 24 березня: 24 – 11000, березень – 03 – 0011. Результат: 1 1000 0011*

Для побудови логічної функції використовуйте таблицю істиності, наведену у таблиці 1.1. Використовуючи таблицю дійсності побудуйте логічне рівняння.

Мінімізуйте логічне рівняння за допомогою будь-якого алгоритму.

Побудуйте принципову схему за допомогою елементів серії 555.

Використовуючи електричну прикципову схему побудуйте часові діаграми у двох випадках – без врахування затримок та з урахуванням затримок.

**Таблиця 1.1 – Таблиця істиності функції**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| х1 | х2 | х3 | х4 | y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | а0 |
| 0 | 0 | 1 | 1 | а1 |
| 0 | 1 | 0 | 0 | а2 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | а3 |
| 1 | 0 | 0 | 0 | а4 |
| 1 | 0 | 0 | 1 | а5 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | а6 |
| 1 | 1 | 1 | 0 | а7 |
| 1 | 1 | 1 | 1 | а8 |

#### 1.2 Створення опису логічних рівнянь за допомогою мови Verilog.

Створіть новий проект. Опишіть рівняння, що отримані у пункті 1.1, за допомогою мови Verilog і проведіть компіляцію проекту.

Основним елементом Verilog HDL є модуль (**module**). Кожен модуль описує якийсь пристрій або ж його частину. Вихідний код модуля, показаний в лістингу 1.1.

Назва модуля записується після слова «module». Ім'я модуля повинно співпадати з ім'ям файлу. Будь-яке цифровий пристрій містить зовнішні контакти. Аналогом цього в Verilog HDL є входи (**inputs**) і виходи (**outputs**). Для нашого пристрою - це «KEY» і «LED».

**Листинг 1.1 – Схема з логічними елементами**

**module** lab1

(

**input** [**1**:**0**] **KEY**, // KEYs

**output** [**3**:**0**] **LED** // LEDs

);

**wire** a = ~ **KEY** [**0**];

**wire** b = ~ **KEY** [**1**];

// Basic gates AND, OR and NOT

**assign** **LED** [**0**] = a & b;

**assign** **LED** [**1**] = a | b;

**assign** **LED** [**2**] = ~ a;

// Building XOR only using AND, OR and NOT

**assign** **LED** [**3**] = ( a | b ) & ~ ( a & b );

**endmodule**

Входи і виходи в пристрої можуть бути як однорозрядні, так і багаторозрядні. Приклад однорозрядного входу:

**input** **rst\_n**,

Для багаторозрядного входу або виходу запис буде виглядати так:

**input** [**1**: **0**] **KEY**, // KEYs

**output** [**3**: **0**] **LED** // LEDs

У квадратних дужках вказують кількість біт – два для входу **KEY** і чотири для виходу **LED**. Це означає, що два сигналу від кнопок будуть підключені до входів модуля, а чотири вихідні сигнали будуть підключені до світлодіодів.

Всередині модуля вхідний сигнал управляє провідником (**wire**). У нашому прикладі це два сигнали: «**a**» і «**b**». Ці сигнали надходять після інверсії вхідних сигналів з кнопок. Сигнал «**а**» - як інверсія **key [0**], сигнал «**b**» - як інверсія **key [1]**:

**wire** a = ~ **KEY** [**0**];

**wire** b = ~ **KEY** [**1**];

Наведемо схему з використанням логічних рівнянь. Наприклад, опис логічного І виглядає так:

**assign** **LED** [**0**] = a & b;

Це означає, що вихід «**led [0]**» буде управлятися логічним І від «**a**» і «**b**».

Оператор «**assign**» задає безперервне присвоювання значення сигналу. «Безперервне» означає, що будь-яка зміна одного з операндів в правій частині виразу відразу викликає зміна значення вихідного сигналу. Цей вид присвоювання завжди синтезується в комбінаційну схему.

Створіть тестбенч для вашої лабораторної роботи. **Тестбенч** (Testbench) – це тестове оточення для перевірки правильності функціонування вашого проекту. Тестбенч генерує набір сигналів для нашого пристрою, а потім отримує сигнали від пристрою. Розглянемо код тестбенчу (Лістинг 1.2) для моделювання схеми, що описується таблицею дійсності 1.1.

**Лістинг 1.2 Модуль 'testbench.v'**

`timescale 1ns / 1ps

module testbench;

// input and output test signals

reg [1:0] key;

wire [3:0] led;

// creating the instance of the module

// we want to test

// lab2 - module name

// dut - instance name

//('dut' means 'device under test')

lab1 dut ( key, led );

// do at the beginning of the simulation

initial

begin

key = 2'b00; // set test signals value

#10; // pause

key = 2'b01; // set test signals value

#10; // pause

key = 2'b10; // set test signals value

#10; // pause

key = 2'b11; // set test signals value

#10; // pause

end

// do at the beginning of the simulation

// print signal values on every change

initial

$monitor("key=%b led=%b", key, led);

// do at the beginning of the simulation

initial

$dumpvars; //iverilog dump init

endmodule

Тестбенч починається з завдання масштабу часу і точності обчислення інтервалів:

**`timescale** **1**ns / **1**ps

Далі слід декларація модуля. Якби це було описом будь-якого пристрою, то далі було б опис портів модуля, але тестбенч не містить зовнішніх портів, а тільки внутрішні сигнали:

reg [1:0] key;

wire [9:0] led;

Опис «провідника» «wire» вже раніше зустрічалося. Провідник не зберігає інформацію на відміну від «reg» - елемента зберігання даних, який зберігає значення сигналу до тих пір, поки йому не буде присвоєно нове значення.

Потім оголошується екземпляр модуля, який буде тестуватися. Тестована схема називається тестованим пристроєм (DUT, device under test) або testing модулем (UUT, unit under test). У нашому тестовому оточенні DUT описується наступним рядком:

lab2 dut (key, led);

Наступні рядки запускають initial блок:initial

begin

...

end

Кожен initial блок буде працювати тільки один раз, на початку моделювання. Якщо ж описано кілька початкових блоків, то вони будуть працювати паралельно.

В цьому розділі напишемо найпростіший тестбенч. Цей тестбенч буде подавати сигнали на входи модуля, що тестується через певні проміжки часу. У DUT є два входи, на які будуть подаватися послідовні сигнали з інтервалом в 10 одиниць модельного часу:

key = 2'b00; // set test signals value

#10; // pause

Формат чисел, що подаються на входи, наступний: **2'b00**. Це означає, що 2-бітний вектор з усіма бітами, рівними 0. **2'b11**, відповідає 2-бітного вектору зі значенням 11 в бінарному вигляді. Синтаксис «**#**» використовується для вказівки затримки. Це означає, що спочатку сигнал «**key**» буде встановлено в значення «**00**». Потім (після затримки в 10 одиниць модельного часу) буде виконана наступна по порядку команда.

Наступний рядок містить команду:

initial $monitor("key=%b led=%b", key, led);

Вона дає команду симулятора виводити всі зміни сигналів «**led**» і «**key**» на консоль.

В цій лабораторній роботі ми робимо досить простий тестбенч для того, щоб ви зрозуміли принцип побудови тестбенчу і формування тестових сигналів. В наступних роботах ми будем роботи формування тестових сигналів більш компактним.

#### 1.3 Симуляція за допомогою ModelSim

Після виконання попередніх кроків ми маємо дві програми на мові Verilog – lab1.v та testbench.v. Також у нас встановлена програма ModelSim. Тепер за її допомогою виконанаємо симуляцію нашого проекту. Для цього виконаємо наступні дії:

Перейдемо до папки simulation, де запустимо скрипт симуляції у ModelSim (Листинг 1.2):

>01\_simulate\_with\_modelsim.bat

У результаті виконання цього скрипта ви побачите результати моделювання вашої лабораторної роботи (рисунок 1.1).

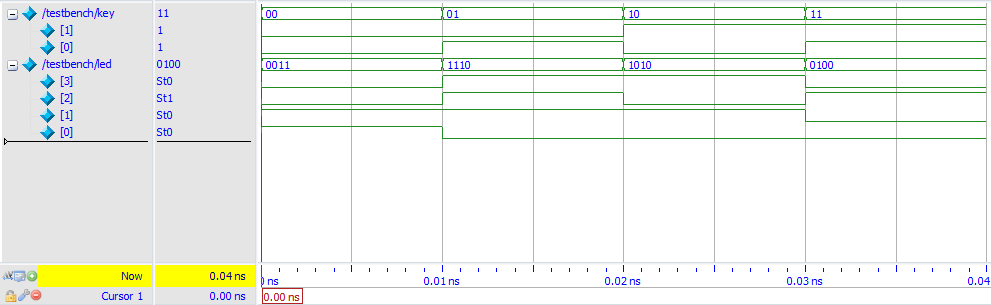


Рисунок 1.1 – Результати симуляції в пакеті ModelSim

Скрип з лістингу 1.3 запускає скрипт моделювання у пакеті ModelSim (Лістинг 1.4). Він написаний на мові TCL – скриптовій мові, що використовується пакетом ModelSim у якості внутрішньої мови для керування цим пакетом. Всі команди цього скрипта мають аналоги у графічному інтерфейсі пакету ModelSim, однак використання скриптів дозволяє значно скоротити роботу по симуляції та позбавитисть виконання однакових дій та зменшити час тестування.

Листинг 1.3 Скрипт '02\_simulate\_with\_modelsim.bat'

rem recreate a temp folder for all the simulation files

rd /s /q sim

md sim

cd sim

rem start the simulation

vsim -do ../modelsim\_script.tcl

rem return to the parent folder

cd ..

Листинг 1.4 Скрипт 'modelsim\_script.tcl`

# create modelsim working library

vlib work

# compile all the Verilog sources

vlog ../testbench.v ../../lab2.v

# open the testbench module for simulation

vsim work.testbench

# add all testbench signals to time diagram

add wave sim:/testbench/\*

# run the simulation

run -all

# expand the signals time diagram

wave zoom full

#### Завдання для самостійного виконання.

Перед проведенням симуляції змініть налаштування кольорової схеми у пакеті ModelSim. Для цього використовуйте інструкцію з файлу ModelSim\_color\_scheme, який розташований у розділі з лабораторними роботами курсу «Схемотехніка радіоелектронних апаратів» на Moodle.

Для використання кольору при формленні лістингів можна використовувати сайт <http://hilite.me>. Для роботи з мовою Verilog необхідно обрати у пункті «**Language**» мову Verilog.

Використовуючи тестбенч (лістинг 1.2) проведіть моделювання вашої програми. Порівняйте результати симуляції та часові діаграми, що були отримані при виконанні п. 1.1.