Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота № 6

з дисципліни «Комп’ютерна схемотехніка»

**«ПРИСТРОЇ ДЛЯ ПЕРЕТВОРЕННЯ ЧИСЕЛ.
ТИПОВІ ВУЗЛИ КОМП’ЮТЕРА»**

Виконав:
студент групи ІО-32
Душко Р.В.
Номер залікової книжки: 3206

Перевірив:
Викладач Нікольський С.С.

Київ 2025 р.

Лабораторна робота № 6

Тема: Пристрої для перетворення чисел. Типові вузли комп’ютера.

Мета роботи: Ознайомитись з методами побудови арифметичних комбінаційних пристроїв, зокрема багаторозрядних суматорів. Реалізувати 6-розрядний суматор двома способами — структурно, через каскадування однорозрядних суматорів, та поведінково — з використанням оператора додавання.

Хід роботи

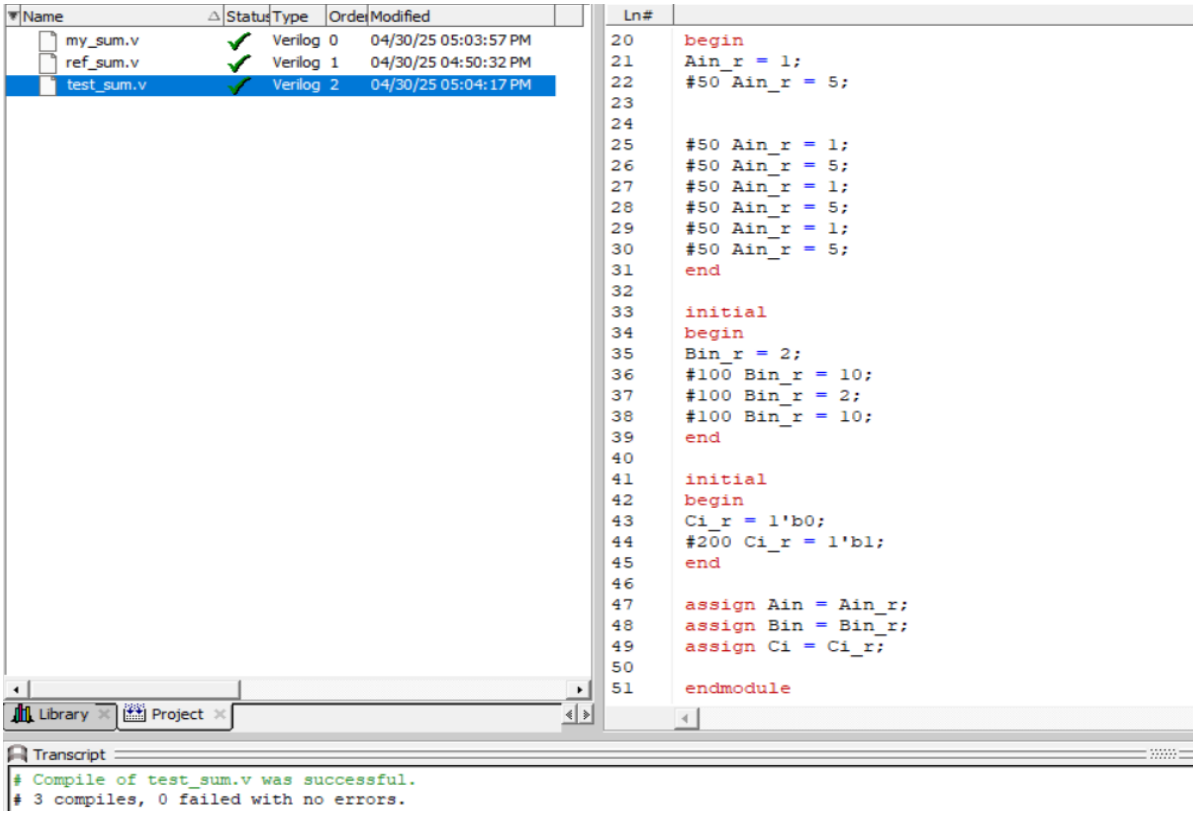
Варіант:

3206_{10} – 110010100110_2, звідси :

0	1	1	0	0	1
h1	h2	h3	h4	h5	h6

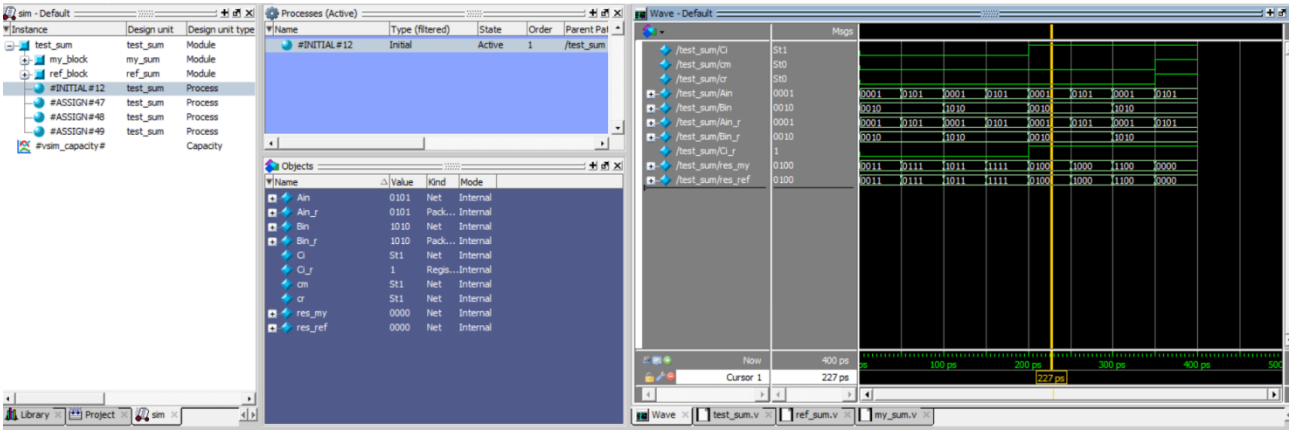
h3 h2 h1	Розрядність суматора
1 1 0	7

Створюємо проект, а також файли на мові Verilog відповідно методички. Прописуємо код та проводимо компіляцію.



Компіляція пройшла успішно

Перейдемо у режим моделювання :



Результати з консолі:

```
VSIM 4> run -all
#                Time Ain Bin Ci res_my cm res_ref cr
#                   0    1   2    0    3        0      3       0
#                  50    5   2    0    7        0      7       0
#                 100    1  10    0   11        0     11       0
#                 150    5  10    0   15        0     15       0
#                 200    1   2    1    4        0      4       0
#                 250    5   2    1    8        0      8       0
#                 300    1  10    1   12        0     12       0
#                 350    5  10    1    0        1      0       1
# ** Note: $finish    : C:/Users/User/Desktop/KC/6/Lab_6/test_sum.v(15)
#    Time: 400 ps  Iteration: 0  Instance: /test_sum
# 1
# Break in Module test_sum at C:/Users/User/Desktop/KC/6/Lab_6/test_sum.v line 15
```

Ми бачимо, що на входи суматора подаються операнди Ain, Bin та сигнал переносу Ci. Результат додавання виводиться на res_my — це результат нашого суматора, і cm — сигнал переносу з нього. Також паралельно працює модуль ref_sum, який реалізований на поведінковому рівні, і виводить res_ref та cr.Під час симуляції бачимо, що для кожної комбінації входів результати res_my і res_ref повністю збігаються, як і сигнали переносу cm і cr. Це підтверджує, що реалізація мого суматора my_sum працює правильно. Симуляція виконана коректно.

Створимо новий проєкт та створимо там файл на мові Verilog та напишемо там код для 7-ти розрядного суматора. За основу візьмемо код, який використовувався у прикладі вище . У результаті маємо отакий код:

```
Ln#
 1
 2   module sum_1r (A, B, Cin, S, Cout);
 3       input A, B, Cin;
 4       output S, Cout;
 5
 6       wire Res, c1, c2;
 7
 8       xor (Res, A, B);
 9       and (c1, A, B);
10       xor (S, Cin, Res);
11       and (c2, Cin, Res);
12       or  (Cout, c1, c2);
13   endmodule
```

У цьому модулі реалізовано логіку повного однорозрядного суматора з урахуванням вхідного переносу. Сума обчислюється через два XOR, а перенос — через два AND і один OR. Цей модуль є базовим елементом для побудови багаторозрядного суматора.

```
 1   module sum_7r (Ain, Bin, Ci, Sout, Co);
 2       input  [6:0] Ain, Bin;
 3       input        Ci;
 4       output [6:0] Sout;
 5       output       Co;
 6
 7       wire [6:0] C;
 8
 9       sum_1r sum1 (Ain[0], Bin[0], Ci,     Sout[0], C[0]);
10       sum_1r sum2 (Ain[1], Bin[1], C[0],   Sout[1], C[1]);
11       sum_1r sum3 (Ain[2], Bin[2], C[1],   Sout[2], C[2]);
12       sum_1r sum4 (Ain[3], Bin[3], C[2],   Sout[3], C[3]);
13       sum_1r sum5 (Ain[4], Bin[4], C[3],   Sout[4], C[4]);
14       sum_1r sum6 (Ain[5], Bin[5], C[4],   Sout[5], C[5]);
15       sum_1r sum7 (Ain[6], Bin[6], C[5],   Sout[6], C[6]);
16
17       assign Co = C[6];
18   endmodule
```

У цьому модулі реалізовано 7-розрядний суматор на основі послідовного з'єднання 7 повних однорозрядних суматорів (sum_1r). Кожен розряд враховує перенос із попереднього, що дозволяє точно передавати перенесення по всьому розряду. Структурна побудова дозволяє краще уявити внутрішню логіку додавання.

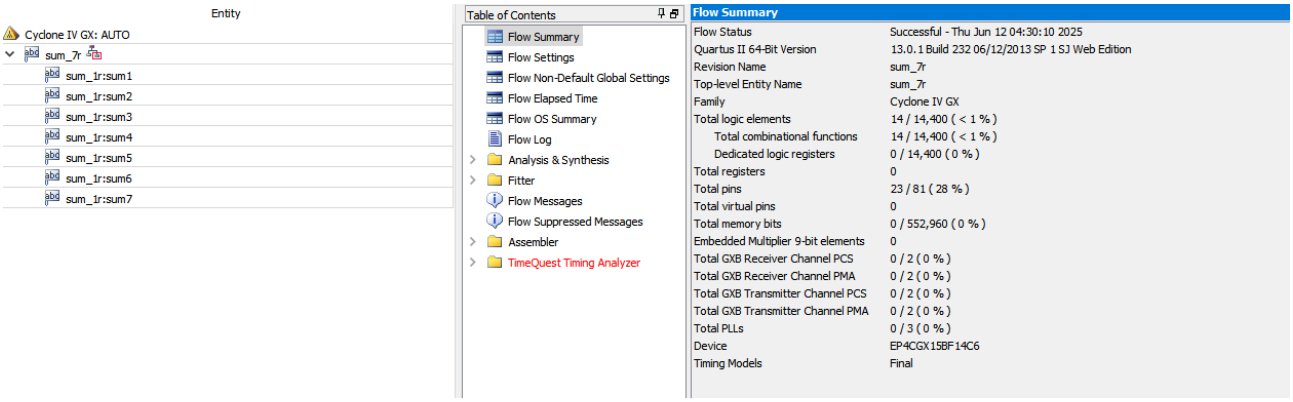
```
1   module ref_7r_sum (Ain, Bin, Ci, Sout, Co);
2       input  [6:0] Ain, Bin;
3       input        Ci;
4       output [6:0] Sout;
5       output       Co;
6
7       reg [7:0] S;
8
9       always @(*) begin
10          S = Ain + Bin + Ci;
11      end
12
13      assign Sout = S[6:0];
14      assign Co   = S[7];
15  endmodule
```

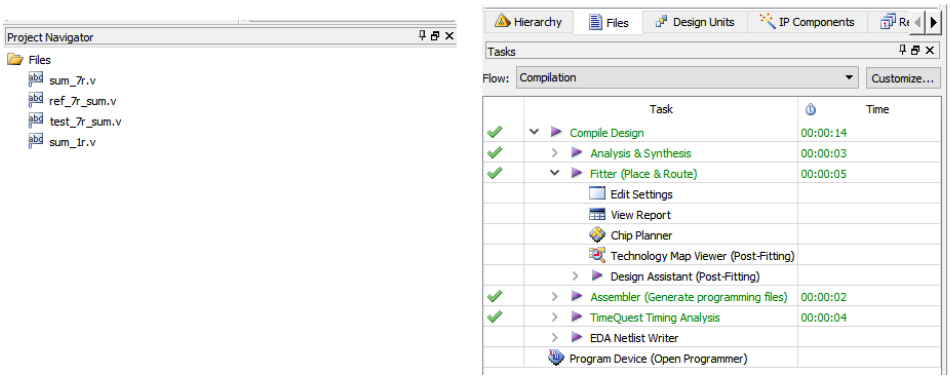
У даному модулі використовується поведінковий опис суматора, де результат обчислюється за допомогою оператора +. Це дозволяє швидко і просто реалізувати додавання з урахуванням переносу, без логічної деталізації. Такий модуль зручно використовувати як еталон для перевірки.

```
module test_7r_sum;
    wire C1, C2;
    wire [6:0] Ain, Bin;
    reg  [6:0] Ain_r, Bin_r;
    reg        Ci_r;
    wire [6:0] res_my, res_ref;

    sum_7r     my_block  (Ain, Bin, Ci_r, res_my, C1);
    ref_7r_sum ref_block (Ain, Bin, Ci_r, res_ref, C2);

    initial begin
        $display("\tTime\tAin\tBin\tCi\tres_my\tC1\tres_ref\tC2");
        $monitor("%t\t%b\t%b\t%b\t%b\t%b\t%b\t%b",
                 $time, Ain, Bin, Ci_r, res_my, C1, res_ref, C2);
        #500 $finish;
    end

    initial begin
        Ain_r = 7'd5;
        #50 Ain_r = 7'd10;
        #50 Ain_r = 7'd20;
        #50 Ain_r = 7'd33;
        #50 Ain_r = 7'd15;
    end

    initial begin
        Bin_r = 7'd3;
        #100 Bin_r = 7'd12;
        #100 Bin_r = 7'd20;
    end

    initial begin
        Ci_r = 1'b0;
        #200 Ci_r = 1'b1;
    end

    assign Ain = Ain_r;
    assign Bin = Bin_r;

endmodule
```

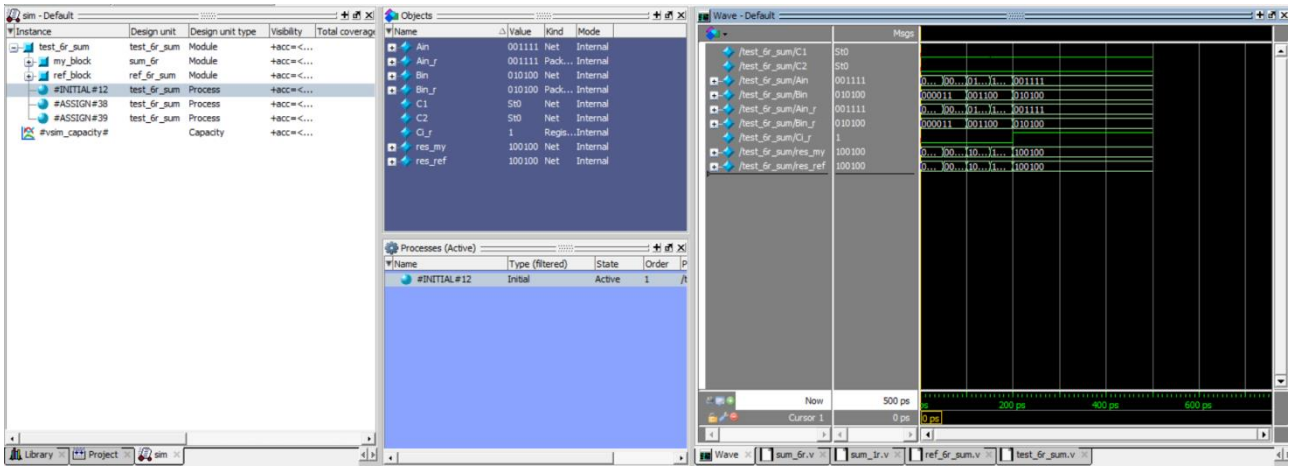
У модулі тестбенча реалізовано перевірку роботи суматора шляхом подачі різних комбінацій вхідних значень Ain, Bin та Ci. Результати виводяться в консоль через $monitor. Порівняння результатів структурної та поведінкової реалізацій дозволяє впевнено перевірити правильність роботи власного суматора.

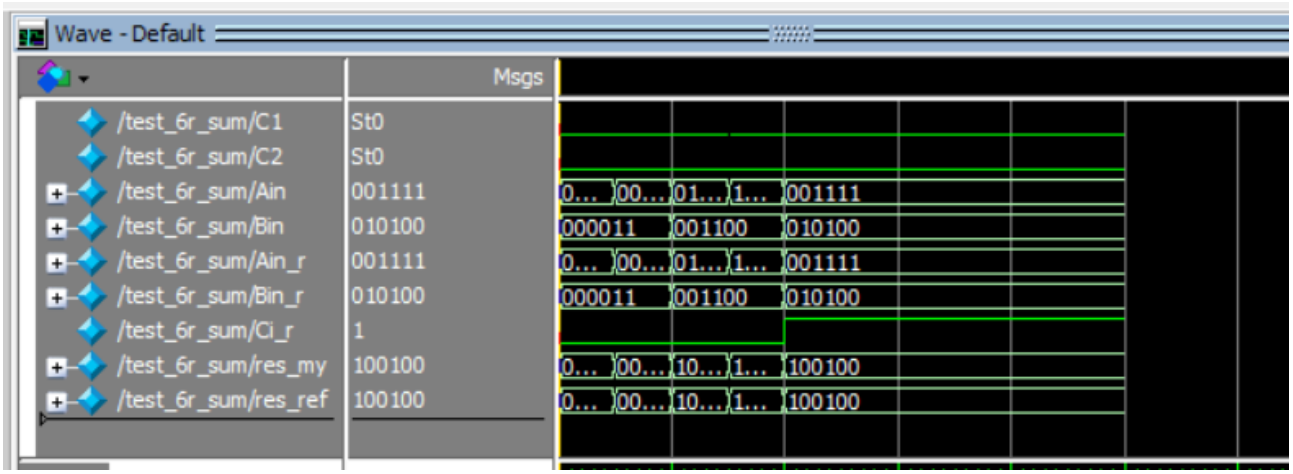
Тепер скомпілюємо всі ці файли .





Перейдемо у режим модулювання та перевіримо чи правильні результати:





```
#         Time Ain Bin Ci res_my cm res_ref cr
#                    0     1     2     0     3       0       3       0
#                   50     5     2     0     7       0       7       0
#                  100     1    10     0    11       0      11       0
#                  150     5    10     0    15       0      15       0
#                  200     1     2     1     4       0       4       0
#                  250     5     2     1     8       0       8       0
#                  300     1    10     1    12       0      12       0
#                  350     5    10     1     0       1       0       1
# ** Note: $finish    : C:/Users/User/Desktop/KC/6/Lab_6/test_sum.v(15)
#    Time: 570 ps  Iteration: 0  Instance: /test_sum
# 1
# Break in Module test_sum at C:/Users/User/Desktop/KC/6/Lab_6/test_sum.v line 15
# WARNING: No extended dataflow license exists
```

Все виконано вірно, відповідно до теоретичних даних (вимог).

Висновки: У даній лабораторній роботі було реалізовано 7-розрядний суматор двома способами: структурно та поведінково. Структурна модель (sum_7r) була побудована з семи однорозрядних повних суматорів (sum_1r), з’єднаних послідовно. Поведінкова реалізація (ref_7r_sum) виконувала додавання за допомогою оператора "+". Для перевірки обох підходів було створено тестовий модуль (test_7r_sum), який подавав різні вхідні дані та порівнював результати. Симуляція у ModelSim показала повний збіг результатів, що підтверджує правильність роботи структурної моделі.

Посилання на git_hub репозиторій:

https://github.com/Romchik235/Circuit-Design/tree/main/Lab-6