

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

Перевірів:
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

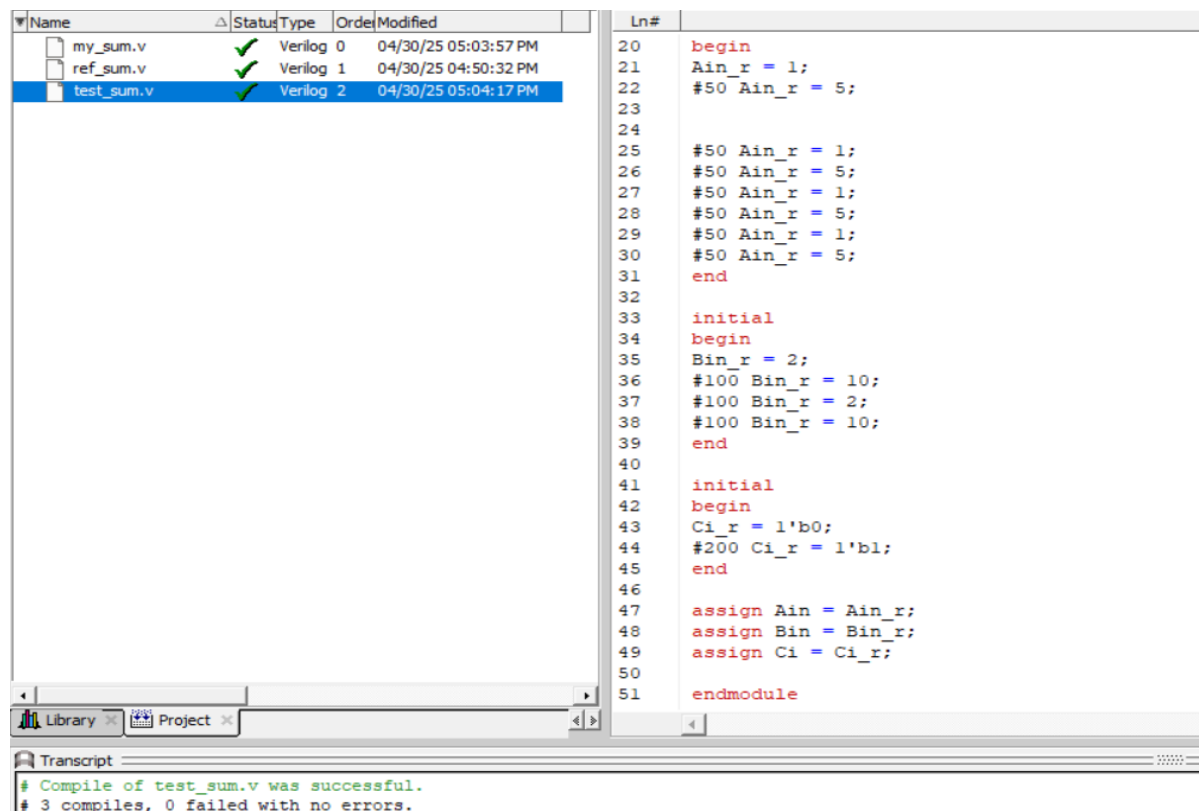
Варіант:

$3206_{10} - 110010100110_2$, звідси :

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

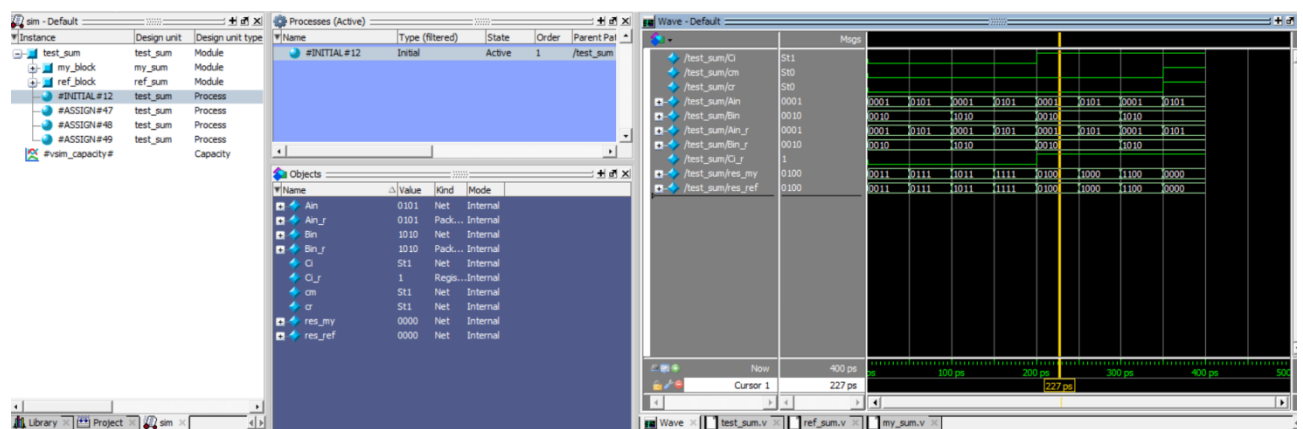
h3 h2 h1	Розрядність суматора
1 1 0	7

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:

```

VSIM4> 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	<code>module sum_lr (A, B, Cin, S, Cout);</code>
3	<code>input A, B, Cin;</code>
4	<code>output S, Cout;</code>
5	
6	<code>wire Res, c1, c2;</code>
7	
8	<code>xor (Res, A, B);</code>
9	<code>and (c1, A, B);</code>
10	<code>xor (S, Cin, Res);</code>
11	<code>and (c2, Cin, Res);</code>
12	<code>or (Cout, c1, c2);</code>
13	<code>endmodule</code>

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

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 .

Entity

Cyclone IV GX: AUTO

- sum_7r
 - sum_1r:sum1
 - sum_1r:sum2
 - sum_1r:sum3
 - sum_1r:sum4
 - sum_1r:sum5
 - sum_1r:sum6
 - sum_1r:sum7

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Flow Messages
- Flow Suppressed Messages
- Assembler
- TimeQuest Timing Analyzer

Flow Summary

Flow Status: Successful - Thu Jun 12 04:30:10 2025

Quartus II 64-Bit Version: 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition

Revision Name: sum_7r

Top-level Entity Name: sum_7r

Family: Cyclone IV GX

Total logic elements: 14 / 14,400 (< 1 %)

Total combinational functions: 14 / 14,400 (< 1 %)

Dedicated logic registers: 0 / 14,400 (0 %)

Total registers: 0

Total pins: 23 / 81 (28 %)

Total virtual pins: 0

Total memory bits: 0 / 552,960 (0 %)

Embedded Multiplier 9-bit elements: 0

Total GX8 Receiver Channel PCS: 0 / 2 (0 %)

Total GX8 Receiver Channel PMA: 0 / 2 (0 %)

Total GX8 Transmitter Channel PCS: 0 / 2 (0 %)

Total GX8 Transmitter Channel PMA: 0 / 2 (0 %)

Total PLLs: 0 / 3 (0 %)

Device: EP4CGX15BF14C6

Timing Models: Final

Project Navigator

- Files
 - sum_7r.v
 - ref_7r_sum.v
 - test_7r_sum.v
 - sum_1r.v

Hierarchy Files Design Units IP Components Rt

Tasks

Flow: Compilation Customize...

Task	Time
Compile Design	00:00:14
Analysis & Synthesis	00:00:03
Fitter (Place & Route)	00:00:05
Edit Settings	
View Report	
Chip Planner	
Technology Map Viewer (Post-Fitting)	
Design Assistant (Post-Fitting)	
Assembler (Generate programming files)	00:00:02
TimeQuest Timing Analysis	00:00:04
EDA Netlist Writer	
Program Device (Open Programmer)	

Перейдемо у режим модулювання та перевіримо чи правильні результати:

sim - Default

Instance	Design unit	Design unit type	Visibility	Total coverage
test_6r_sum	test_6r_sum	Module	+acc<...	
my_block	sum_6r	Module	+acc<...	
ref_block	ref_6r_sum	Module	+acc<...	
#INITIAL#12	test_6r_sum	Process	+acc<...	
#ASSIGN#38	test_6r_sum	Process	+acc<...	
#ASSIGN#39	test_6r_sum	Process	+acc<...	
#vsm_capacity#	test_6r_sum	Capacity	+acc<...	

Objects

VName	Value	Kind	Mode
Ain	001111	Net	Internal
Ain_r	001111	Pack...	Internal
Bin	010100	Net	Internal
Bin_r	010100	Pack...	Internal
C1	S0	Net	Internal
C2	S0	Net	Internal
Cl_r	1	Regs...	Internal
res_my	100100	Net	Internal
res_ref	100100	Net	Internal

Processes (Active)

Name	Type (filtered)	State	Order
#INITIAL#12	Initial	Active	1

Wave - Default

Wave	Time
/test_6r_sum/C1	S0
/test_6r_sum/C2	S0
/test_6r_sum/Ain	001111
/test_6r_sum/Bin	010100
/test_6r_sum/Ain_r	001111
/test_6r_sum/Bin_r	010100
/test_6r_sum/Cl_r	1
/test_6r_sum/res_my	100100
/test_6r_sum/res_ref	100100

Wave - Default

Wave	Time
/test_6r_sum/C1	S0
/test_6r_sum/C2	S0
/test_6r_sum/Ain	001111
/test_6r_sum/Bin	010100
/test_6r_sum/Ain_r	001111
/test_6r_sum/Bin_r	010100
/test_6r_sum/Cl_r	1
/test_6r_sum/res_my	100100
/test_6r_sum/res_ref	100100

```

#      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>