КОМП’ЮТЕРНА ЛОГІКА

Верифікація та підготовка 4 бітного ALU до реалізації на FPGA

SystemVerilog, self-checking testbench, EPWave та прогноз ресурсів

ЛАБОРАТОРНА РОБОТА 8

Державний університет «Житомирська політехніка»
2026–2027 навчальний рік

Призначення роботи

Робота поєднує лекцію 13 про верифікацію та лекцію 14 про FPGA. Студент реалізує чотирибітний арифметико-логічний пристрій, створює самоперевірний testbench, аналізує waveform і визначає, які FPGA-ресурси потрібні після синтезу.

Мета

- описати ALU конструкціями always_comb і case
- правильно формувати carry/borrow, zero і negative
- створити автоматичні перевірки PASS/FAIL
- проаналізувати EPWave
- пов’язати RTL із LUT і carry chain

Програмне середовище

Параметр	Значення
Сайт	https://www.edaplayground.com/
Testbench + Design	SystemVerilog/Verilog
Simulator	Icarus Verilog 12.0
Open EPWave after run	увімкнути

Підготовка EDA Playground

1. Увійдіть у EDA Playground через Log In.
2. Оберіть SystemVerilog/Verilog і Icarus Verilog 12.0.
3. Увімкніть Open EPWave after run і дозвольте спливні вікна.
4. Правий редактор використовуйте для design.sv, лівий — для testbench.sv.
5. Перед вставленням виділіть початковий текст Ctrl+A і видаліть його.

Короткі теоретичні відомості

ALU (Arithmetic Logic Unit, арифметико-логічний пристрій) виконує вибрану арифметичну або логічну операцію. Двобітний op обирає одну з чотирьох гілок case. Вихід y має чотири біти. Прапорець flag має узгоджене значення: перенос для ADD та INC, позика для SUB та DEC, біт що вийшов для SHL і SHR, нуль для інших операцій. zero дорівнює одиниці за y=0, negative копіює y[3].

Конструкція	Апаратний зміст
always_comb	комбінаційна мережа
case(op)	вибір операції та мультиплексування
a+b	carry chain і LUT
a&b, a|b, a^b	LUT
zero, negative	логіка прапорців

Стандартний приклад

op	Операція
00	AND
01	OR
10	ADD
11	SUB

Крок 1 Код design.sv

```
module alu4(input logic [3:0] a,b, input logic [1:0] op,
   output logic [3:0] y, output logic flag,zero,negative);
 logic [4:0] tmp;
 always_comb begin
  y='0; flag=0; tmp='0;
  case(op)
    2'b00: y = a & b;
    2'b01: y = a | b;
    2'b10: begin tmp={1'b0,a}+{1'b0,b}; y=tmp[3:0]; flag=tmp[4]; end
    2'b11: begin y=a-b; flag=(a<b); end
    default: begin y='0; flag=0; end
  endcase
 end
 assign zero=(y=='0);
 assign negative=y[3];
endmodule
```

Значення за замовчуванням y, flag і tmp задаються до case. Додавання виконується у п’яти бітах. Віднімання залишає молодші чотири біти результату й окремо визначає позику.

Крок 2 Код testbench.sv

```
`timescale 1ns/1ps
module testbench;
 logic [3:0] a,b,y; logic [1:0] op;
 logic flag,zero,negative; int errors=0;
 alu4 dut(.*);
 task automatic check(input logic[3:0] ta,tb,input logic[1:0] to,
   input logic[3:0] ey,input logic ef,ez,en);
   begin a=ta;b=tb;op=to;#1;
    $display("a=%0d b=%0d op=%b | y=%0d f=%b z=%b n=%b",a,b,op,y,flag,zero,negative);
    if({y,flag,zero,negative}!=={ey,ef,ez,en}) begin $error("FAIL"); errors++; end
   end
 endtask
 initial begin
   $dumpfile("dump.vcd");$dumpvars(0,testbench);
   check(5,3,2'b00,1,0,0,0); check(5,3,2'b01,7,0,0,0);
   check(5,3,2'b10,8,0,0,1); check(5,3,2'b11,2,0,0,0);
   check(15,1,2'b10,0,1,1,0); check(0,1,2'b11,15,1,0,1);
   if(errors==0)$display("TEST PASSED"); else $display("TEST FAILED %0d",errors);
   $finish;
 end
endmodule
```

Задача check подає набір, очікує 1 нс, друкує результат і порівнює одразу чотири вихідні ознаки. У стандартному прикладі перевіряються всі операції, перенос 15+1 та позика 0-1.

Крок 3 Запуск

1. Натисніть Run або Ctrl+Enter.
2. У Log знайдіть шість рядків і TEST PASSED.
3. За помилки компіляції перевірте ;, begin/end, назви портів і розрядності.
4. За TEST FAILED зіставте фактичний рядок з очікуваним викликом check.

Крок 4 EPWave

1. Натисніть Get Signals.
2. Додайте a, b, op, y, flag, zero і negative.
3. Для шин виберіть Binary або Unsigned.
4. Перевірте, що кожний інтервал 1 нс відповідає одному виклику check.
5. Знайдіть інтервали 15+1 і 0-1 та поясніть прапорці.

Виконання власного варіанта

1. Знайдіть чотири операції та тестові a і b.
2. Замініть гілки case у design.sv.
3. Розрахуйте y, flag, zero і negative для всіх чотирьох op.
4. Створіть чотири виклики check для заданих a і b.
5. Додайте два граничні тести, доречні для операцій варіанта.
6. Досягніть TEST PASSED.
7. Збережіть EPWave з усіма сигналами.
8. У звіті спрогнозуйте ресурси: LUT, carry chain і мультиплексування.

№	00	01	10	11	a	b
1	AND	XOR	ADD	SUB	9	6
2	NAND	NOR	ADD	SUB	3	5
3	OR	XNOR	INC	DEC	14	3
4	XOR	SHL	SHR	ADD	11	5
5	MAX	MIN	ADD	SUB	7	12
6	AND	OR	INC	SUB	13	4
7	NOTA	NOTB	ADD	XOR	15	9
8	NOR	NAND	SHL	SHR	10	3
9	AND	XNOR	MAX	MIN	9	6
10	SHL	SHR	ADD	SUB	13	7
11	OR	XOR	INC	DEC	4	11
12	INC	DEC	ADD	SUB	15	1

№	00	01	10	11	a	b
13	NOTA	NOTB	AND	OR	2	11
14	ADD	XOR	OR	AND	8	7
15	MAX	MIN	XNOR	XOR	12	9
16	SHL	NOR	ADD	AND	13	5
17	XNOR	SHR	SUB	OR	1	15

Вимоги до звіту

- номер варіанта і таблиця операцій
- ручні очікувані результати шести тестів
- design.sv і testbench.sv
- Log із TEST PASSED
- EPWave
- прогноз FPGA-ресурсів
- короткий висновок

Запитання для захисту

1. Чому ALU є комбінаційним блоком?
2. Для чого потрібні значення за замовчуванням в always_comb?
3. Як двобітний op кодує чотири операції?
4. Навіщо a і b розширювати до п’яти бітів під час ADD?
5. Що означає flag для ADD, SUB і SHL/SHR?
6. Як формуються zero та negative?
7. Чим !== відрізняється від != у testbench?
8. Які FPGA-ресурси очікуються після синтезу ALU?
9. Чому тест на платі не замінює testbench?
10. Який граничний випадок є найважливішим у вашому варіанті?