**Лабораторна робота №4**

**Тема:** Розробка 32-бітного суматора.

**Мета:** отримати знання щодо форматів представлення даних в арифметичних операціях; отримати навики проектування суматорів з використанням поведінкової моделі.

**Теоретичні відомості**

В цифрових пристроях інформація різного характеру представляється двійковими послідовностями з різним числом розрядів. Окремі частини цих послідовностей мають певні призначення. В залежності від типу інформації використовується той чи інший формат даних.

Формати даних можна розбити на дві групи: цілочисельні формати та формати для чисел із плаваючою комою. **Цілочисельні формати** використовуються для представлення цілих чисел і інформації, що може бути закодована такими числами. **Формат із плаваючою комою** використовується для представлення дробових або дійсних чисел. Цілочисельні формати в свою чергу можна поділити на знакові та беззнакові. В цілому **беззнаковому** числі всі розряди використовуються для кодування значення. В **знаковому** числі один з розрядів (як правило,старший) використовується для формування знаку числа, а решта – для кодування значення. Основною характеристикою цілочисельних форматів при виконанні арифметичних операцій, є діапазон представлень чисел (N), що прямо залежить від розрядності формату (b): для беззнакових форматів 0≤N≤2 b -1, для форматів з врахуванням знаку: −2 b-1 ≤N≤2 b-1 -1.

Знакові цілі числа можуть бути представлені в зворотному або додатковому коді. **Зворотний код** для перетворення позитивного числа в негативне та навпаки використовує операцію інверсії. Але у такому випадку числу 0 відповідають 2 коди – „позитивний” 0 та „негативний” 0. Це призводить до ускладнення арифметичних операцій. **Додатковий код** для зміни знаку числа використовує інверсію та додавання 1 до результату інверсії. В обох кодах старший розряд числа представляє знак. Якщо старший розряд дорівнює 0, то це позитивне число, а якщо 1 – негативне.

**Суматор** – це схема, яка призначена для підсумовування двох вхідних двійкових n-розрядних кодів. Операція віднімання замінюється складанням у зворотному або додатковому коді. Операції множення і ділення зводяться до реалізації багаторазових складань і зрушень. Тому суматор є важливим компонентом цифрових пристроїв. Для можливості каскадного з’єднання суматорів використовуються біти переносу, які забезпечують облік результатів додавання молодших частин числа у результатах додавання старших частин.

**Завдання**

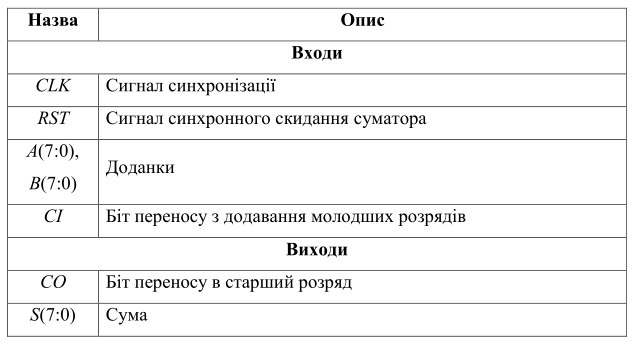
1. Розробити програму 8-розрядного суматора цілих беззнакових чисел з інтерфейсом, який наведено у табл. 1.

2. Провести моделювання розробленої програми.

3. Модифікувати раніше розроблену програму для обробки цілих знакових чисел.

4. Провести моделювання модифікованої програми.

Таблиця 4.1 Інтерфейс суматора

****

**Код програми**

**// ОПРЕДЕЛЕНИЯ КОНСТАНТ**

`define DEL 1 // Задержка распространения сигнала в

// сумматоре. Нулевая задержка может

// привести к проблемам.

`define BITS 32 // Разрядность операндов сумматора

**// ГЛАВНЫЙ МОДУЛЬ**

module Adder(

clk,

a,

b,

reset\_n,

add\_en,

out,

cout,

valid);

**// ВХОДЫ**

input clk; // Тактирующий импульс

input [`BITS-1:0] a; // Входной сигнал операнда A

input [`BITS-1:0] b; // Входной сигнал операнда B

input reset\_n; // Срабатывающий по спаду

// синхронный сигнал сброса

input add\_en; // Синхронный сигнал разрешения работы

**// ВЫХОДЫ**

output [`BITS-1:0] out; // Выход (сумма)

output cout; // Переполнение выхода

output valid; // Сигнал готовности выхода

**// ОБЪЯВЛЕНИЯ СИГНАЛОВ**

wire clk;

wire [`BITS-1:0] a;

wire [`BITS-1:0] b;

wire reset\_n;

wire add\_en;

reg [`BITS-1:0] out;

reg cout;

reg valid;

// ОСНОВНОЙ КОД

// Отслеживание событий на сигнале reset\_n

always @(reset\_n) begin

//Если сигнал сброса установлен в 1, то отключить выходы от

//непрерывного присваивания из следующего блока

if (reset\_n) begin

deassign out;

deassign cout;

cout = 1’b0;

out = `BITS’h0;

end

// Ожидание возрастания тактирующего сигнала

@(posedge clk);

// Если reset установлен в 0 в момент возрастания clk,

// установить нули на выходе устройства (с задержкой `DEL)

if (~reset\_n) begin

#`DEL

assign cout = 1’b0;

assign out = `BITS’h0;

end

end

// Ожидание возрастания тактирующего сигнала

always @(posedge clk) begin

if (add\_en)

//Вычисление суммы с учетом бита переноса

{cout, out} <= #`DEL a+b;

// Выход становится корректным после сброса или суммирования

valid <= #`DEL ~reset\_n | add\_en | valid;

end

endmodule // Конец модуля Adder

//

//

/\* Код уровня регистровых передач представлен ниже: \*/

//

//

**// ОПРЕДЕЛЕНИЯ КОНСТАНТ**

`define DEL 1 // Задержка распространения сигнала в

// сумматоре. Нулевая задержка может

// привести к проблемам.

// ГЛАВНЫЙ МОДУЛЬ

module Adder(

clk,

a,

b,

add\_en,

add\_en,

out,

cout,

valid);

**// ВХОДЫ**

input clk; //Тактирующий импульс

input [31:0] a; //32-битный входной сигнал операнда A

input [31:0] b; //32-битный входной сигнал операнда B

input reset\_n; // Срабатывающий по спаду

// синхронный сигнал сброса

input add\_en; // Синхронный сигнал разрешения работы

**// ВЫХОДЫ**

output [31:0] out; // 32-битный выход (сумма)

output cout; // Переполнение выхода

output valid; // Сигнал готовности выхода

**// ОБЪЯВЛЕНИЯ СИГНАЛОВ**

wire clk;

wire [31:0] a;

wire [31:0] b;

wire reset\_n;

wire add\_en;

wire [31:0] out;

wire cout;

wire valid;

wire [7:0] cout4; // Сигналы переполнения выходов

// 4-битных сумматоров

reg [2:0] valid\_cnt; // Счетчик, определяющий готовность

// выходов

// Операторы assign – модель потоков данных

assign #`DEL cout = cout4[7];

assign #`DEL valid = ~|valid\_cnt;

// Основной код

// Включение восьми 4-битных сумматоров

Adder\_4bit Add0(

.clk(clk),

.a(a[3:0]),

.b(b[3:0]),

.cin(1’b0),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[3:0]),

.cout(cout4[0]));

Adder\_4bit Add1(

.clk(clk),

.a(a[7:4]),

.b(b[7:4]),

.cin(cout4[0]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[7:4]),

.cout(cout4[1]));

Adder\_4bit Add2(

.clk(clk),

.a(a[11:8]),

.b(b[11:8]),

.cin(cout4[1]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[11:8]),

.cout(cout4[2]));

Adder\_4bit Add3(

.clk(clk),

.a(a[15:12]),

.b(b[15:12]),

.cin(cout4[2]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[15:12]),

.cout(cout4[3]));

Adder\_4bit Add4(

.clk(clk),

.a(a[19:16]),

.b(b[19:16]),

.cin(cout4[3]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[19:16]),

.cout(cout4[4]));

Adder\_4bit Add5(

.clk(clk),

.a(a[23:20]),

.b(b[23:20]),

.cin(cout4[4]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[23:20]),

.cout(cout4[5]));

Adder\_4bit Add6(

.clk(clk),

.a(a[27:24]),

.b(b[27:24]),

.cin(cout4[5]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[27:24]),

.cout(cout4[6]));

Adder\_4bit Add7(

.clk(clk),

.a(a[31:28]),

.b(b[31:28]),

.cin(cout4[6]),

.reset\_n(reset\_n),

.add\_en(add\_en),

.out(out[31:28]),

.cout(cout4[7]));

// Ожидание возрастания тактирующего сигнала

always @(posedge clk) begin

if (~reset\_n) begin

// Инициализировать счетчик valid\_cnt

valid\_cnt <= #`DEL 3’h0;

end

else if (((valid\_cnt == 3’h0) && (add\_en == 1’b1)) ||

(valid\_cnt != 3’h0)) begin

// Увеличить счетчик, если сигналы valid и add\_en = 1

// или valid = 0

valid\_cnt <= #`DEL valid\_cnt + 1;

end

end

endmodule // Конец модуля Adder

**// ПОДЧИНЕННЫЙ МОДУЛЬ**

module Adder\_4bit(

clk,

a,

b,

add\_en,

cin,

out,

cout);

**// ВХОДЫ**

input clk; // Тактирующий импульс

input [3:0] a; // 4-битный входной сигнал операнда A

input [3:0] b; // 4-битный входной сигнал операнда B

input cin; // Бит переноса на входе

input reset\_n; // Срабатывающий по спаду

// синхронный сигнал сброса

input add\_en; // Синхронный сигнал разрешения работы

**// ВЫХОДЫ**

output [3:0] out; // 4-битный выход (сумма)

output cout; // Переполнение выхода

**// ОБЪЯВЛЕНИЯ СИГНАЛОВ**

wire clk;

wire [3:0] a;

wire [3:0] b;

wire cin;

wire reset\_n;

wire add\_en;

reg [3:0] out;

reg cout;

**// ОСНОВНОЙ КОД**

// Ожидание возрастания тактирующего сигнала

always @(posedge clk) begin

if (~reset\_n) begin

{cout,out} <= #`DEL 33’h00000000;

end

else if (add\_en) begin

{cout, out} <= #`DEL a+b+cin;

end

end

endmodule // Конец модуля Adder\_4bit

**Висновок:** під час виконання лабораторної роботи ми отримали знання щодо форматів представлення даних в арифметичних операціях; отримали навики проектування суматорів з використанням поведінкової моделі. Розроблено програму 8-розрядного суматора цілих беззнакових чисел з інтерфейсом, який наведено у табл. 1. Проведено моделювання розробленої програми.