**Лабораторная работа №4**

**Разработка сумматора**

**Цель:** Получить навыки проектирования сумматоров с использованием поведенческой модели.

**Теория**

Табл.1. Интерфейс сумматора

|  |  |
| --- | --- |
| **Название** | **Описание** |
| **Входы** | |
| *CLK* | сигнал синхронизации |
| *RST* | Сигнал синхронного сброса сумматора |
| *A*(7:0),  *B*(7:0) | слагаемые |
| *CI* | Бит переноса с добавлением младших разрядов |
| **Выходы** | |
| *CO* | Бит переноса в старший разряд |
| *S*(7:0) | сумма |

В цифровых устройствах информация различного характера представляется двоичными последовательностями с различным числом разрядов. Отдельные части этих последовательностей имеют определенные назначения. В зависимости от типа информации используется тот или иной формат данных. Форматы данных можно разбить на две группы: целочисленные форматы и форматы для чисел с плавающей запятой. целочисленные форматы используются для представления целых чисел и информации, которая может быть закодирована такими числами. Формат с плавающей запятой используется для представления дробных или действительных чисел.

Целочисленные форматы в свою очередь можно разделить на знаковые и беззнаковое. В целом беззнаковую числе все разряды используются для кодирования значения. В знаковом числе один из разрядов (как правило, старший) используется для формирования знака числа, а остальные - для кодирования значения. Основной характеристикой целочисленных форматов при выполнении арифметических операций, является диапазон представлений чисел (N),

что напрямую зависит от разрядности формата (b): для беззнаковых форматов 0≤*N*≤2*b-1*,

для форматов с учетом знака: −2b-1≤*N*≤2b-1-1.

Знаковые целые числа могут быть представлены в обратном или дополнительном коде. Обратная код для преобразования положительного числа в отрицательное и наоборот использует операцию инверсии. Но в таком случае числу 0 соответствуют 2 коды - "позитивный" 0 и "негативный" 0. Это приводит к усложнению арифметических операций. дополнительный код для изменения знака числа использует инверсию и добавления 1 к результату инверсии. В обоих кодах старший разряд числа представляет знак. если старший разряд равен 0, то это положительное число, а если 1 - отрицательное. Сумматор - это схема, которая предназначена для суммирования двух входных двоичных n-разрядных кодов. Операция вычитания заменяется сборкой в ​​обратном или дополнительном коде. Операции умножения и деления сводятся к реализации многократных сборок и сдвигов. Поэтому сумматор является важным компонентом цифровых устройств. для возможности каскадного соединения сумматоров используются биты переноса, которые обеспечивают учет результатов добавления младших частей числа в результатах добавления старших частей.

**Код программы:**

// 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

**Вывод.** На лабораторной работе получили навыки проектирования сумматоров с использованием поведенческой модели. В цифровых устройствах информация различного характера представляется двоичными последовательностями с различным числом разрядов. Отдельные части этих последовательностей имеют определенные назначения. В зависимости от типа информации используется тот или иной формат данных. Форматы данных можно разбить на две группы: целочисленные форматы и форматы для чисел с плавающей запятой. целочисленные форматы используются для представления целых чисел и информации, которая может быть закодирована такими числами. Формат с плавающей запятой используется для представления дробных или действительных чисел.