Lab8_1

задание для самостоятельного выполнения

Плата для аппаратной отладки проекта

Аппаратная отладка проекта ориентирована на плату MiniDiLaB-CIV

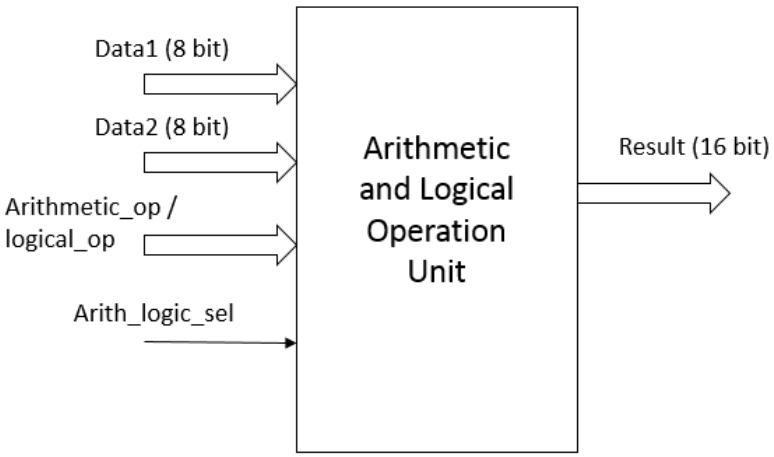
MiniDiLaB-CIV: Микросхема - **EP4CE6E22C8**, Вход тактового сигнала (25МГц) – 23.

Описание проекта Lab8_1

Рабочая папка Lab8_1. Имя проекта – Lab8_1. Имя модуля верхнего уровня – Lab8_1. Файл с описанием – Lab8_1.sv. Файл теста – tb_Lab8_1.sv. Файл с описанием для отладки – db_Lab8_1.sv.

Алгоритм работы:

The arith_logic SystemVerilog model used in this lab works as a miniature 8-bit ALU. It performs four arithmetic or four logical operations on 8-bit operands.



The enumerated type variable:

- arithmetic_op chooses between the four arithmetic operations that are available: addition, subtraction, multiplication, and division.
- logical_op variable decides between the four logical operations that are available: nand, nor, not, and xor.

These operations are done on two data signals, which are specified in the packed structure called arith_logic_info.
The final result is of 16 bits.
The arith_logic_sel signal selects between the arithmetic and logical operations.
You will find the following signals in this lab:

- arith_logic_sel – Selects between arithmetic and logical operations.
- operation – Selects which arithmetic or logical operation is to be done.
- ip_data1 and ip_data2 – Two 8-bit data inputs.
- data_out – Output of the selected operation.

Apart from these, there are two objects created in this module, through which the structures and their members can be accessed:

- arith_data – Object of the packed structure arith_logic_info performing arithmetic operations.
- logic_data – Object of the packed structure arith_logic_info performing logical operations.

Пример исходного кода с описанием АЛУ представлен ниже:

```
`timescale 1ns / 1ps
    // User defined enumerated data type declaration
    typedef enum logic[1:0] {add=0, sub, mul, div} arith_operation;
    typedef enum logic[1:0] {nand_op=0, nor_op, not_op, xor_op} logic_operation;
    // Creating one Packed and one Unpacked Structure here
    // User defined Packed Structure declaration
    typedef struct packed { arith_operation arithmetic_op;
                            logic_operation logical_op;
                            logic [7:0] data1;
                            byte data2;
                            logic [15:0] arith_result;
                            logic [15:0] logic_result;
                           } arith_logic_info;

module arith_logic (    input logic arith_logic_sel,         // Selects between arith or logic operations
                        input logic [1:0] operation,         // Selects which arith or logic operations is to be done
                        input logic [7:0] ip_data1, ip_data2,// Two data inputs
                        output logic [15:0] data_out);       // Output of the selected operation

    always @ (*)
    begin
        arith_logic_info arith_data, logic_data;    // Creating two objects arith_data and logic_data

        if (arith_logic_sel==0)
        // Selecting Arithmetic operation
        begin
            //Read the input data
            arith_data.arithmetic_op = arith_operation'(operation);
            arith_data.logical_op = logic_operation'(0);
            arith_data.data1 = ip_data1;
            arith_data.data2 = ip_data2;

            case (arith_data.arithmetic_op)         // Accessing enum data type through arith_data input
            // As per the value of enum, do the arithmatic operations
            // Accesing the Packed Structure for two datas and storing the result in an Packed Union
                add : arith_data.arith_result = arith_data.data1 + arith_data.data2;
                sub : arith_data.arith_result = arith_data.data1 - arith_data.data2;
                mul : arith_data.arith_result = arith_data.data1 * arith_data.data2;
                div : arith_data.arith_result = arith_data.data1 / arith_data.data2;
            endcase
            data_out = arith_data.arith_result; //Write the result to the output
        end
        else
        // Selecting Logical operation
        begin
            logic_data.arithmetic_op = arith_operation'(0);
            logic_data.logical_op = logic_operation'(operation);

            logic_data.data1 = ip_data1;
            logic_data.data2 = ip_data2;

            case (logic_data.logical_op)        // Accessing enum data type through logic_data input
            // As per the value of enum, do the logic operations
            // Accesing the Packed Structure for two datas and storing the result in an Packed Union
                nand_op : logic_data.logic_result = ~((logic_data.data1) & (logic_data.data2));
                nor_op  : logic_data.logic_result = ~((logic_data.data1) | (logic_data.data2));
                not_op  : logic_data.logic_result = ~ (logic_data.data1);
                xor_op  : logic_data.logic_result =  ((logic_data.data1) ^ (logic_data.data2));
            endcase
            data_out = logic_data.logic_result;
        end
    end

endmodule
```

Программа работы

- Разработать описание устройства АЛУ – модуль Lab8_1
  - В качестве основы можно использовать приведенный выше пример.
  - ***К описанию надо добавить***
    - *Вход CLK – вход тактового сигнала*
    - *Вход RST – вход синхронного сброса,* сброс (=1)/работа (=0)
    - Вход ENA – вход разрешающий (=1)/запрещающий (=0) работу устройства.
  - ***К описанию надо добавить***
    - На всех входах (кроме ENA и RST) должны быть использованы регистры (с разрешением работы и синхронным сбросом)
    - на выходе Result(16 bit) должен быть использован регистр(с разрешением работы и синхронным сбросом).
- Разработать тест tb_Lab8_1 для проверки АЛУ (тест первого класса – без автоматической проверки).
  - Пример теста приведен в отдельном разделе ниже – на его базе надо разработать свой тест.
- По результатам моделирования в ModelSim необходимо доказать работоспособность устройства АЛУ (продемонстрировать выполнение всех операций для нескольких наборов данных).
- Разработать модуль верхнего уровня для отладки db_Lab8_1, содержащий: модуль Lab8_1; модуль SP_unit (модуль, обеспечивающий возможность: задания входных сигналов без использования кнопок на плате; отображения результата). Модуль должен обеспечивать подключение к тактовому сигналу на плате.
- Провести анализ работы db_Lab8_1 и доказать (зафиксировав результаты снимками экрана), что:
  - Модуль управляется входными сигналами
  - Правильно реализуется алгоритм работы

Содержание отчета

- Отчет должен быть оформлен по правилам, принятым в Высшей Школе.
- Отчет должен содержать все этапы работы, все созданные исходные коды, необходимые снимки экрана. Все рисунки и полученные на них результаты должны быть прокомментированы.

Пример теста

Ниже приведен пример теста для проверки АЛУ. На его основе надо разработать собственный тест.

```
1   `timescale 1ns / 1ps
2   module tb_arith_logic;
3
4       // Inputs
5       logic arith_logic_sel;
6       logic [7:0] ip_data1, ip_data2;
7       logic [1:0] operation;
8
9       // Output
10      logic [15:0] data_out;
11
12      // instantiate the unit under test (uut)
13      arith_logic inst (  .arith_logic_sel(arith_logic_sel),
14                          .operation(operation),
15                          .ip_data1(ip_data1),
16                          .ip_data2(ip_data2),
17                          .data_out(data_out)
18                       );
19
20      initial
21      begin
22          // initialize inputs
23          arith_logic_sel = 0;
24          operation = 2'b00;
25          ip_data1 = 0;
26          ip_data2 = 0;
27      end
28
29      initial
30      begin
31          arith_logic_sel = 0;
32          ip_data1 = 8'd20;
33          ip_data2 = 8'd10;
34
35          operation = 2'b00;
36
37          #10;
38          operation = 2'b01;
39
40          #10;
41          operation = 2'b10;
42
43          #10;
44          operation = 2'b11;
45
46          #10;
47          arith_logic_sel = 1;
48          ip_data1 = 8'd60;
49          ip_data2 = 8'd15;
50
51          operation = 2'b00;
52
53          #10;
54          operation = 2'b01;
55
56          #10;
57          operation = 2'b10;
58
59          #10;
60          operation = 2'b11;
61      end
62  endmodule
```