Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

Отчёт по лабораторной работе № 6

Дисциплина: Автоматизация проектирования дискретных устройств (на английском языке).

Выполнил студент гр. 5130901/10101 _____ Д.Л. Симоновский
(подпись)

Руководитель _____ А.А. Федотов
(подпись)

“13” марта 2024 г.

Санкт-Петербург
2024

Оглавление

1. Список иллюстраций:

2. Цель упражнения:

Пройти цикл проектирования в рамках пакетов Quartus и ModelSim, включая следующие этапы:

- Создание проекта.
- Разработка описания модулей с использованием конструкций расширения SystemVerilog.
- Разработка теста на языке SystemVerilog и моделирование.
- Отладка проекта.

3. Алгоритм работы проекта:

АЛУ (арифметико-логическое устройство) с параметризированной разрядностью (width=8, по умолчанию). Реализуется как комбинационная схема. Выполняет знаковые операции:

- ADD – сложение.
- SUB – вычитание.
- MUL – умножение.
- DIV – деление.
- OP_18 – среднее арифметическое.

Выводы устройства:

- ops – вход кода операции
- op_a – вход операнда A: знаковый (разрядность width).
- op_b – вход операнда B: знаковый (разрядность width).
- ALU_out – выход результата: знаковый (разрядность width).
- CLK – вход тактового сигнала, для синхронизации ISSPE и SignalTapII.

4. Решение:

Первым делом необходимо создать пакет для типов данных, используемых в проекте:

```
package lab_MS_SV4_pack;
  parameter WIDTH = 8;
  typedef enum bit [2:0] { ADD, SUB, MUL, DIV, OP_18 } opcode_t;
  typedef bit signed [WIDTH-1:0] data_y;

  typedef struct packed {
    opcode_t opc;
    data_y   op_a;
    data_y   op_b;
  } INST_t;
endpackage : lab_MS_SV4_pack
```

Здесь мы видим перечисление кодов операций и общий тип данных для входов и выходов. Эти типы объединены в структуру.

На основе этого пакета создадим модуль АЛУ:

```
`timescale 1ns / 1ns
import lab_MS_SV4_pack::*;

module lab_MS_SV4 (
  input  INST_t INST,
  output data_y ALU_out
);

  always_comb begin
    case (INST.opc)
      ADD: ALU_out = INST.op_a + INST.op_b;
      SUB: ALU_out = INST.op_a - INST.op_b;
      MUL: ALU_out = INST.op_a * INST.op_b;
      DIV: ALU_out = INST.op_a / INST.op_b;
      OP_18: ALU_out = (INST.op_a + INST.op_b) / 2;
    endcase
  end
endmodule
```

В качестве входных данных принимается структура INST_t, созданная ранее, а выходным параметром является выход АЛУ, который аналогично имеет ранее созданный тип данных. Выполним компиляцию и убедимся в том, что была создана именно комбинационная схема:

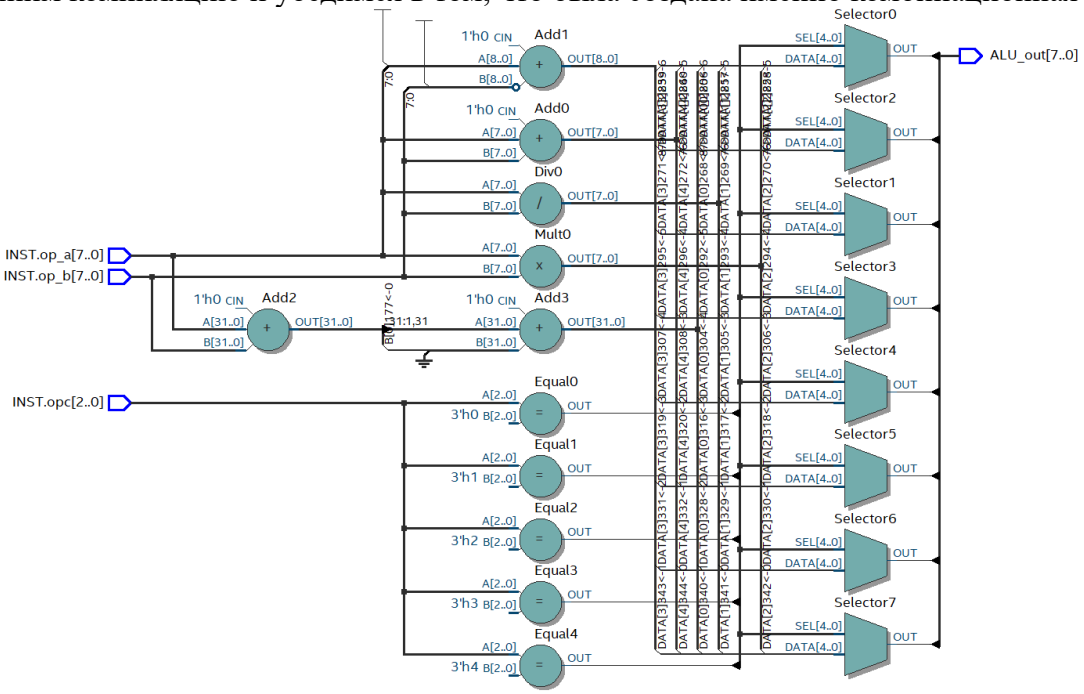

Рис. 4.1. RTL схема разработанного АЛУ.

Проверим корректность разработанного модуля, используя тест первого класса:

```
`timescale 1ns / 1ns
import lab_MS_SV4_pack::*;

module tb_lab_MS_SV4 ();
  INST_t INST;
  data_y ALU_out;

  lab_MS_SV4 DUT (.*);

  task test(
    input data_y a,
    input data_y b
  );
    INST.opc = INST.opc.first();
    do begin
      INST.op_a = a;
      INST.op_b = b;
      #10 INST.opc = INST.opc.next();
    end while (INST.opc != INST.opc.last());
    #10;
    $display("\n");
  endtask

  initial begin
    test(10, -5);
    test(127, 127);
    test(-128, -128);
    test(10, 0);
    $display("\n");
    $stop;
  end
  initial
    $monitor(
        "instruction=%p \top_a=%d \top_b=%d \tALU_out=%d",
        INST.opc,
        INST.op_a,
        INST.op_b,
        ALU_out
    );
endmodule
```

Этот тест проходит по всем операциям и подает на вход различные комбинации цифр, чтоб проверить, что операции корректно переключаются. Запустим его и получим следующий результат:

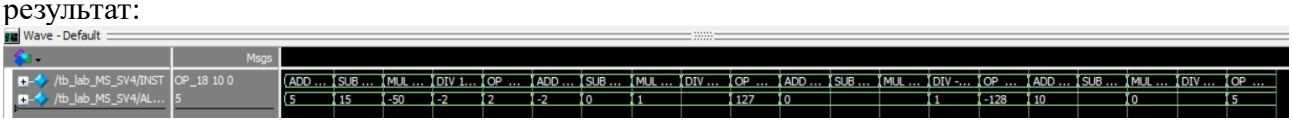

Рис. 4.2. Waveform для тестовой последовательности.

```
# instruction=ADD  op_a=  10  op_b=  -5  ALU_out=   5
# instruction=SUB  op_a=  10  op_b=  -5  ALU_out=  15
# instruction=MUL  op_a=  10  op_b=  -5  ALU_out= -50
# instruction=DIV  op_a=  10  op_b=  -5  ALU_out=  -2
# instruction=OP_18  op_a=  10  op_b=  -5  ALU_out=   2
#
#
# instruction=ADD  op_a= 127  op_b= 127  ALU_out=  -2
# instruction=SUB  op_a= 127  op_b= 127  ALU_out=   0
# instruction=MUL  op_a= 127  op_b= 127  ALU_out=   1
# instruction=DIV  op_a= 127  op_b= 127  ALU_out=   1
# instruction=OP_18  op_a= 127  op_b= 127  ALU_out= 127
#
#
# instruction=ADD  op_a=-128  op_b=-128  ALU_out=   0
# instruction=SUB  op_a=-128  op_b=-128  ALU_out=   0
# instruction=MUL  op_a=-128  op_b=-128  ALU_out=   0
# instruction=DIV  op_a=-128  op_b=-128  ALU_out=   1
# instruction=OP_18  op_a=-128  op_b=-128  ALU_out=-128
#
#
# instruction=ADD  op_a=  10  op_b=   0  ALU_out=  10
# instruction=SUB  op_a=  10  op_b=   0  ALU_out=  10
# instruction=MUL  op_a=  10  op_b=   0  ALU_out=   0
# instruction=DIV  op_a=  10  op_b=   0  ALU_out=   0
# instruction=OP_18  op_a=  10  op_b=   0  ALU_out=   5
```

Рис. 4.3. Результат запуска в консоли.

Как мы видим все операции работают корректно, в соответствии с заданием.

Разработаем модуль для отладки программы на плате:

```
import lab_MS_SV4_pack::*;

module db_lab_MS_SV4 (
  (* altera_attribute = "-name IO_STANDARD \"3.3-v LVCMOS\"", chip_pin = "23" *)
  input CLK
);

  INST_t INST;
  data_y ALU_out;

  lab_MS_SV4 DUT (
    .ALU_out,
    .INST
  );

  SP_unit SP_ (
    .source    (INST),
    .probe     (ALU_out),
    .source_clk(CLK)
  );
endmodule
```

Для ввода значений в модуль будем использовать ISSP, там же будем смотреть результат. Дополнительно добавим Signal Tap II, в котором будем получать значения по изменению типа операции и получать результат в виде сегментов:

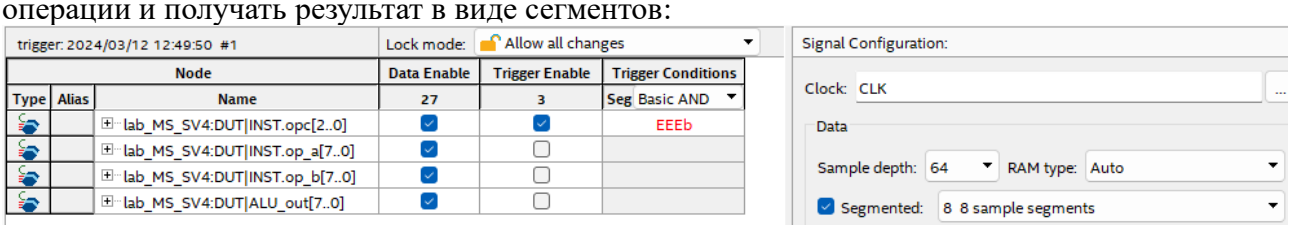

Рис. 4.4. Настройки Signal Tap II.

Запишем наш проект на плату и начнем тестирование:

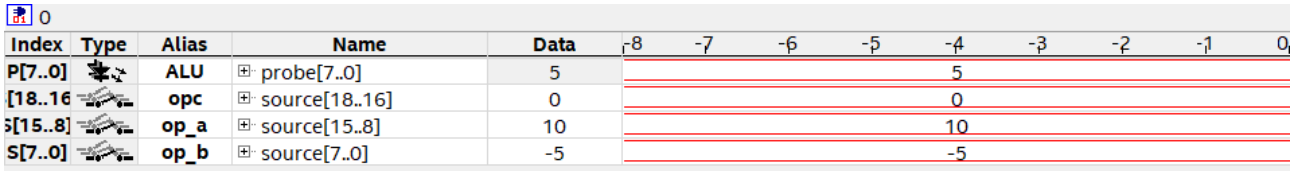

Рис. 4.5. ISSP операция ADD.

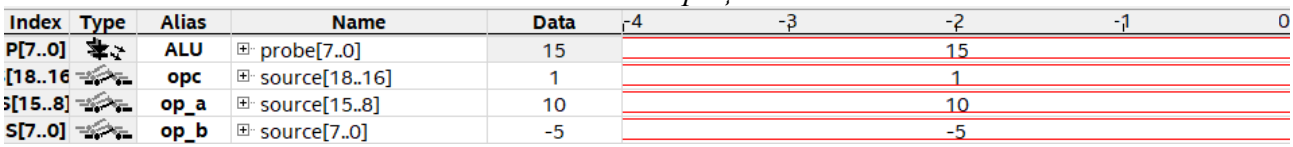

Рис. 4.6. ISSP операция SUB.

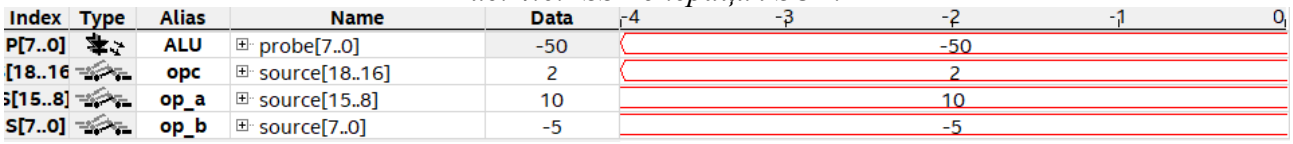

Рис. 4.7. ISSP операция MUL.

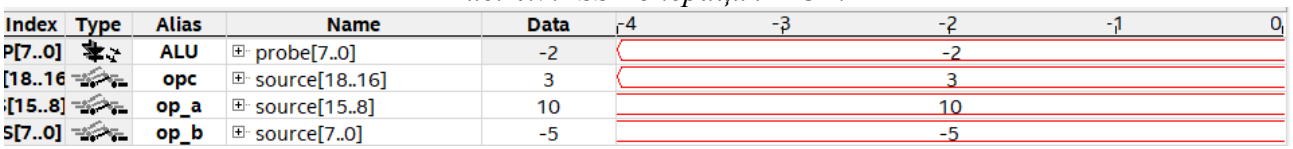

Рис. 4.8. ISSP операция DIV.

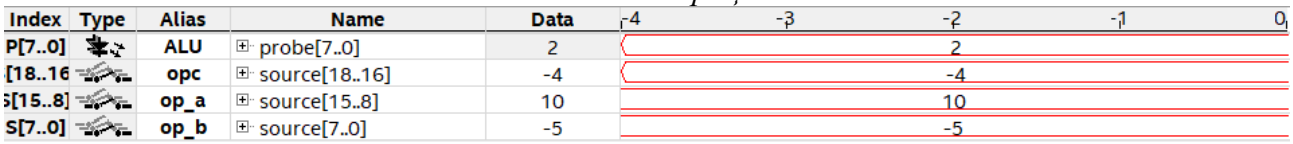

Рис. 4.9. ISSP операция OP 18.

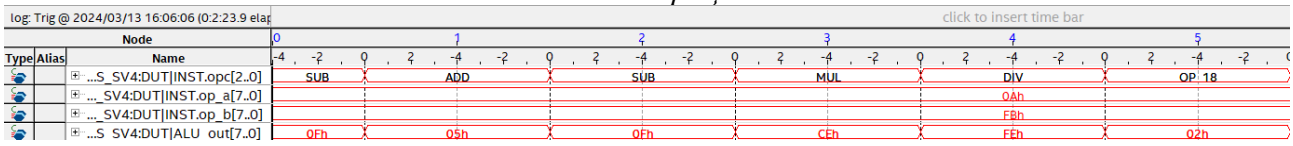

Рис. 4.10. Результат в Signal Tap II.

Как видно по рисункам, приведенным выше, устройство работает корректно в соответствии с заданием, что было продемонстрированно преподавателю практики.

5. Вывод:

В ходе лабораторной работы успешно пройден цикл проектирования, начиная с создания проекта и разработки модулей с использованием расширений SystemVerilog. Использование SystemVerilog предоставило широкий спектр новых возможностей по сравнению с Verilog, облегчая процесс разработки и улучшая читаемость кода.

Отладка проекта осуществлялась с помощью инструментов In-System Sources and Probes Editor и SignalTap II, что значительно повысило эффективность процесса. Эти инструменты позволили быстро выявить и исправить ошибки, что является ключевым аспектом при работе с любым проектом.