

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**

<b>1. Список иллюстраций: .....</b>	<b>2</b>
<b>2. Цель упражнения: .....</b>	<b>3</b>
<b>3. Алгоритм работы проекта: .....</b>	<b>3</b>
<b>4. Решение: .....</b>	<b>3</b>
<b>5. Вывод: .....</b>	<b>6</b>

## 1. Список иллюстраций:

Рис. 4.1. RTL схема разработанного АЛУ. ....	4
Рис. 4.2. Waveform для тестовой последовательности. ....	5
Рис. 4.3. Результат запуска в консоли. ....	5
Рис. 4.4. Настройки Signal Tap II. ....	5
Рис. 4.5. ISSP операция ADD. ....	6
Рис. 4.6. ISSP операция SUB. ....	6
Рис. 4.7. ISSP операция MUL. ....	6
Рис. 4.8. ISSP операция DIV. ....	6
Рис. 4.9. ISSP операция OP_18. ....	6
Рис. 4.10. Результат в Signal Tap II. ....	6

## 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 А: знаковый (разрядность width).
- op\_b – вход операнда В: 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:

```
1 package lab_MS_SV4_pack;
2 parameter WIDTH = 8;
3 typedef enum bit [2:0] { ADD, SUB, MUL, DIV, OP_18 } opcode_t;
4 typedef bit signed [WIDTH-1:0] data_y;
5
6 typedef struct packed {
7     opcode_t opc;
8     data_y op_a;
9     data_y op_b;
10 } INST_t;
11 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:

```
1 `timescale 1ns / 1ns
2 import lab_MS_SV4_pack::*;
3
4 module lab_MS_SV4 (
5     input INST_t INST,
6     output data_y ALU_out
7 );
8
9     always_comb begin
10         case (INST.opc)
11             ADD: ALU_out = INST.op_a + INST.op_b;
12             SUB: ALU_out = INST.op_a - INST.op_b;
13             MUL: ALU_out = INST.op_a * INST.op_b;
14             DIV: ALU_out = INST.op_a / INST.op_b;
15             OP_18: ALU_out = (INST.op_a + INST.op_b) / 2;
16         endcase
17     end
18 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:

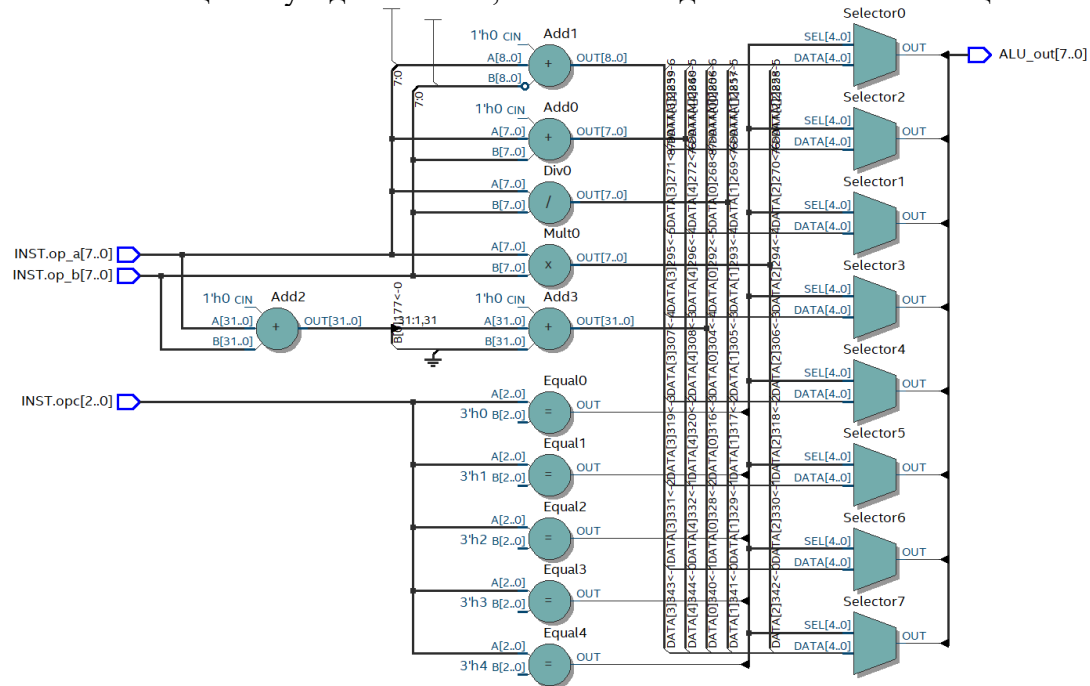


Рис. 4.1. RTL схема разработанного АЛУ.

Проверим корректность разработанного модуля, используя тест первого класса:

```

1  `timescale 1ns / 1ns
2  import lab_MS_SV4_pack::*;
3
4  module tb_lab_MS_SV4 ();
5      INST_t INST;
6      data_y ALU_out;
7
8      lab_MS_SV4 DUT (.*);
9
10     task test(
11         input data_y a,
12         input data_y b
13     );
14         INST.opc = INST.opc.first();
15         do begin
16             INST.op_a = a;
17             INST.op_b = b;
18             #10 INST.opc = INST.opc.next();
19         end while (INST.opc != INST.opc.last());
20         #10;
21         $display("\n");
22     endtask
23
24     initial begin
25         test(10, -5);
26         test(127, 127);
27         test(-128, -128);
28         test(10, 0);
29         $display("\n");
30         $stop;
31     end
32     initial
33         $monitor(
34             "instruction=%p \top_a=%d \top_b=%d \tALU_out=%d",
35             INST.opc,
36             INST.op_a,
37             INST.op_b,
38             ALU_out
39         );
40 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:

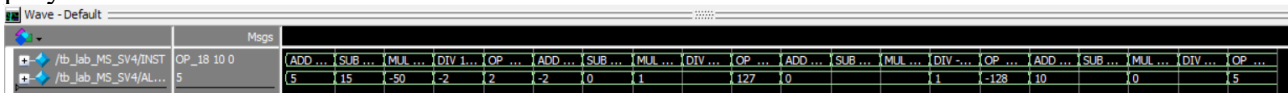


Рис. 4.2. Waveform для тестовой последовательности.

```
# 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:

```
1 import lab_MS_SV4_pack::*;
2
3 module db_lab_MS_SV4 (
4     (* altera_attribute = "-name IO_STANDARD \"3.3-v LVCMS\" , chip_pin = \"23\" *)
5     input CLK
6 );
7
8 INST_t INST;
9 data_y ALU_out;
10
11 lab_MS_SV4 DUT (
12     .ALU_out,
13     .INST
14 );
15
16 SP_unit SP_ (
17     .source (INST),
18     .probe (ALU_out),
19     .source_clk(CLK)
20 );
21 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:

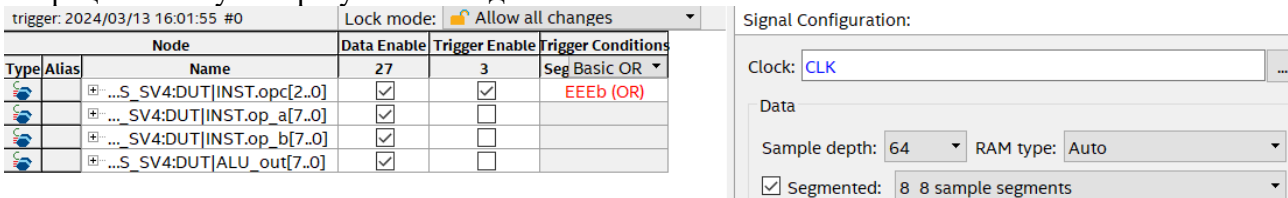


Рис. 4.4. Настройки Signal Tap II.

Запишем наш проект на плату и начнем тестирование:





Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
P[7..0]		ALU	⊕ probe[7..0]	5					5				
[18..16]		opc	⊕ source[18..16]	0					0				
S[15..8]		op_a	⊕ source[15..8]	10					10				
S[7..0]		op_b	⊕ source[7..0]	-5					-5				

Рис. 4.5. ISSP операция ADD.





Index	Type	Alias	Name	Data	-4	-3	-2	-1	0
P[7..0]		ALU	⊕ probe[7..0]	15			15		
[18..16]		opc	⊕ source[18..16]	1			1		
S[15..8]		op_a	⊕ source[15..8]	10			10		
S[7..0]		op_b	⊕ source[7..0]	-5			-5		

Рис. 4.6. ISSP операция SUB.





Index	Type	Alias	Name	Data	-4	-3	-2	-1	0
P[7..0]		ALU	⊕ probe[7..0]	-50			-50		
[18..16]		opc	⊕ source[18..16]	2			2		
S[15..8]		op_a	⊕ source[15..8]	10			10		
S[7..0]		op_b	⊕ source[7..0]	-5			-5		

Рис. 4.7. ISSP операция MUL.





Index	Type	Alias	Name	Data	-4	-3	-2	-1	0
P[7..0]		ALU	⊕ probe[7..0]	-2			-2		
[18..16]		opc	⊕ source[18..16]	3			3		
[15..8]		op_a	⊕ source[15..8]	10			10		
S[7..0]		op_b	⊕ source[7..0]	-5			-5		

Рис. 4.8. ISSP операция DIV.





Index	Type	Alias	Name	Data	-4	-3	-2	-1	0
P[7..0]		ALU	⊕ probe[7..0]	2			2		
[18..16]		opc	⊕ source[18..16]	-4			-4		
S[15..8]		op_a	⊕ source[15..8]	10			10		
S[7..0]		op_b	⊕ source[7..0]	-5			-5		

Рис. 4.9. ISSP операция ОР 18.

Timing diagram showing clock and data signals. The clock signal is a square wave. The data signals are labeled with operations: SUB, ADD, SUB, MUL, DIV, and OP:18. The diagram shows the timing of these operations relative to the clock cycles.

Рис. 4.10. Результат в Signal Tap II.

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