

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

Отчёт по лабораторной работе Lab_MS_SV_3

Дисциплина: Автоматизация проектирования дискретных устройств (на
английском языке)

Выполнил студент гр. 5130901/10101 _____ М.Т. Непомнящий
(подпись)

Руководитель _____ А.А. Федотов
(подпись)

Санкт-Петербург
2024

Оглавление

1.	Задание	4
1.1.	Цель работы	4
2.	Ход решения	5
2.1.	Создание LFSR модуля	5
2.2.	Создание теста первого класса	6
2.3.	Создания модуля для тестирования на плате	9
2.4.	Настройка Signal Tap II	10
2.5.	Тестирование на плате средствами Signal Tap II	11
3.	Вывод	13

Список иллюстраций

Рис. 1 – Пакет для типов данных	5
Рис. 2 – Модуль АЛУ	5
Рис. 3 – Структура модуля АЛУ в RTL Viewer	6
Рис. 4 – Тестовый файл для модуля АЛУ (1).....	6
Рис. 5 – Тестовый файл для модуля АЛУ (2).....	7
Рис. 6 – Моделирование тестового файла средствами ModelSim (wave)	7
Рис. 7 – Моделирование тестового файла средствами ModelSim (cmd).....	7
Рис. 8 – Модифицированный модуль для теста первого класса.....	8
Рис. 9 – Общая вывод после симуляции модифицированного tb	8
Рис. 10 – Результат SLU_out > 127	9
Рис. 11 – Обработка случая остатка от деления на 0	9
Рис. 12 – Обработка случая деления на 0.....	9
Рис. 13 – Модуль db для тестирования на плате	9
Рис. 14 – Настройка окна Signal Tap II	10
Рис. 15 – Настройка Signal Probe	10
Рис. 16 – Мнемоническая таблица.....	10
Рис. 17 – Симуляция в Signal Probe (op_a = 10, op_b = 3)	11
Рис. 18 – Результат SP (ADD, op_a = 10, op_b = 3)	11
Рис. 19 – Результат SP (SUB, op_a = 10, op_b = 3)	11
Рис. 20 – Результат SP (MUL, op_a = 10, op_b = 3).....	11
Рис. 21 – Результат SP (DIV, op_a = 10, op_b = 3).....	11
Рис. 22 – Результат SP (VAR, op_a = 10, op_b = 3)	11
Рис. 23 – Результат SP (ADD, op_a = 127, op_b = 127)	12
Рис. 24 – Временные характеристики устройства.....	12

1. Задание

1.1. Цель работы

Создать 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.
- `VAR` – по варианту

Выводы устройства:

- `ops` – вход кода операции (**3** бита – в Вашем проекте, в примерах ниже – 2 бита).
- `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.

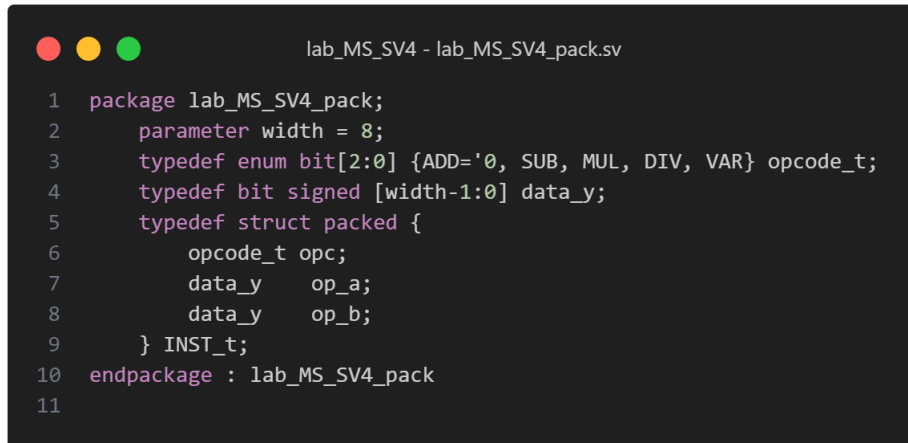
1.2. Программа работы:

1. Разработать описание устройства АЛУ (модуль `lab_MS_SV4`) на SystemVerilog, используя пакет и структуру.
 - Создать пакет (`lab_MS_SV4_pack.sv`) с определением структуры и другими необходимыми конструкциями.
 - В основном файле (`lab_MS_SV4.sv`) передать структуру в модуль.
2. Разработать тест (`tb_lab_MS_SV4.sv`) для проверки АЛУ.
 - Создать тест первого класса без автоматической проверки. Выбрать значения данных, демонстрирующие правильность работы всех операций, включая дополнительные.
 - Представить результаты теста в отчете: временная диаграмма и вывод в консоль.
3. Разработать модуль верхнего уровня для отладки (`db_lab_MS_SV4`), содержащий модуль `lab_MS_SV4` и модуль `SP_unit`.
 - Модуль `SP_unit` обеспечивает возможность задания входных сигналов, синхронизируемых тактовым сигналом `CLK`, без использования кнопок на плате, и отображения результата.
4. Используя ISSPE, провести анализ работы `lab_MS_SV4` и зафиксировать работоспособность устройства АЛУ, демонстрируя выполнение всех операций, включая дополнительные.
5. Настроить SignalTapII для получения временной диаграммы, аналогичной примеру, и отобразить соответствующую дополнительную операцию.

2. Ход решения

2.1. Создание LFSR модуля

Создадим пакет для типов данных, которые будут использоваться в проекте:

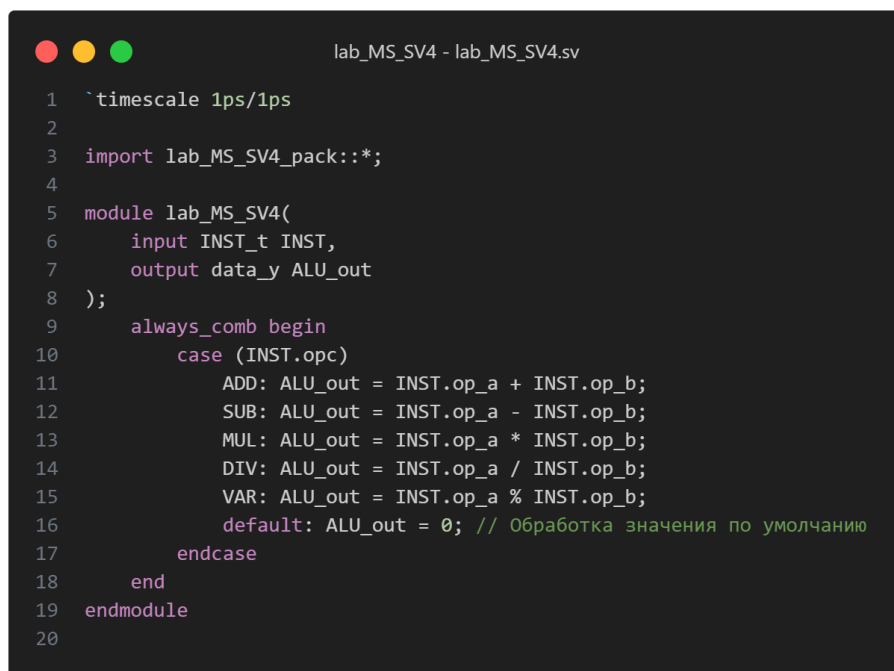


```
lab_MS_SV4 - lab_MS_SV4_pack.sv
1 package lab_MS_SV4_pack;
2     parameter width = 8;
3     typedef enum bit[2:0] {ADD='0, SUB, MUL, DIV, VAR} opcode_t;
4     typedef bit signed [width-1:0] data_y;
5     typedef struct packed {
6         opcode_t opc;
7         data_y    op_a;
8         data_y    op_b;
9     } INST_t;
10 endpackage : lab_MS_SV4_pack
11
```

Рис. 1 – Пакет для типов данных

В этом пакете мы храним коды операций (ADD='0, SUB, MUL, DIV, VAR) и общий тип данных для их входов и выходов. Эта информация помещена в один общий пакет.

Теперь, в соответствии с только что созданным модулем и вариантом задания создадим модуль АЛУ, где пропишем то, как работают операции с арифметической точки зрения:



```
lab_MS_SV4 - lab_MS_SV4.sv
1 `timescale 1ps/1ps
2
3 import lab_MS_SV4_pack::*;
4
5 module lab_MS_SV4(
6     input INST_t INST,
7     output data_y ALU_out
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             VAR: ALU_out = INST.op_a % INST.op_b;
16             default: ALU_out = 0; // Обработка значения по умолчанию
17         endcase
18     end
19 endmodule
20
```

Рис. 2 – Модуль АЛУ

*Здесь VAR – это операция нахождения остатка от деления нацело `op_a` на `op_b`, заданная по варианту.

После компиляции данного модуля можем увидеть, что его структура в RTL Viewer выглядит следующим образом:

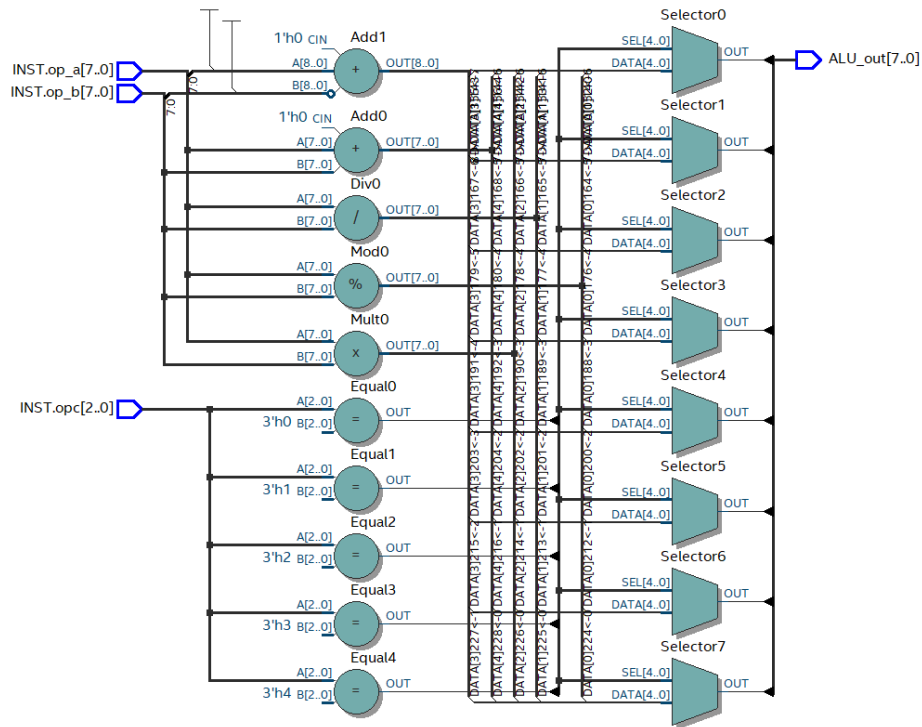


Рис. 3 – Структура модуля АЛУ в RTL Viewer

2.2. Создание теста первого класса

Напишем тест первого класса для только что созданного модуля АЛУ:

```

lab_MS_SV4 - tb_lab_MS_SV4.sv

1  `timescale 1ps/1ps
2
3  import lab_MS_SV4_pack::*;
4
5  module tb_lab_MS_SV4();
6      INST_t INST;
7      data_y ALU_out;
8
9      lab_MS_SV4 UUT (.ALU_out, .INST);
10
11     initial begin
12         for (int op = 0; op < 5; op++) begin
13             for (int i = 0; i < 256; i++) begin
14                 for (int j = 0; j < 256; j++) begin
15                     INST.op_a = i;
16                     INST.op_b = j;
17                     #10;
18                     case (INST.opc)
19                         ADD: begin
20                             ALU_out = INST.op_a + INST.op_b;
21                             $display("\tADD: \tALU_out = %0d + %0d = %0d;", INST.op_a, INST.op_b, ALU_out);
22                         end
23                         SUB: begin
24                             ALU_out = INST.op_a - INST.op_b;
25                             $display("\tSUB: \tALU_out = %0d - %0d = %0d;", INST.op_a, INST.op_b, ALU_out);
26                         end
27                         MUL: begin
28                             ALU_out = INST.op_a * INST.op_b;
29                             $display("\tMUL: \tALU_out = %0d * %0d = %0d;", INST.op_a, INST.op_b, ALU_out);
30                         end
31                         DIV: begin
32                             if (INST.op_b != 0) begin
33                                 ALU_out = INST.op_a / INST.op_b;
34                                 $display("\tDIV: \tALU_out = %0d / %0d = %0d;", INST.op_a, INST.op_b, ALU_out);
35                             end else begin
36                                 // Выводим сообщение об ошибке на waveform
37                                 $error("%t Division by zero occurred: Error %0d DIV %0d", $realtime, INST.op_b, ALU_out);
38                             end
39                         end
40                     end
41                 end
42             end
43         end
44     end
45 end

```

Рис. 4 – Тестовый файл для модуля АЛУ (1)

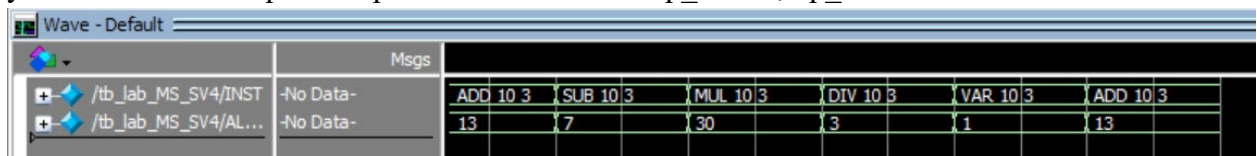
```

40         VAR: begin
41             if (INST.op_b != 0) begin
42                 ALU_out = INST.op_a % INST.op_b;
43                 $display("\tVAR: \tALU_out = %d %% %d = %d;", INST.op_a, INST.op_b, ALU_out);
44             end else begin
45                 // Выводим сообщение об ошибке на waveform
46                 $error("%t Modulo by zero occurred: Error %d VAR %d", $realtime, INST.op_b, ALU_out);
47             end
48         end
49         default: begin
50             // Выводим сообщение об ошибке на waveform
51             $error("%t Invalid operation occurred: Error %d operation(%d) %d", $realtime, INST.op_b, INST.opc, ALU_out);
52         end
53     endcase
54 end
55 end
56 INST.opc = INST.opc.next();
57 end
58 while (INST.opc != INST.opc.first());
59     #10;
60     $stop;
61 end
62 endmodule
63

```

Рис. 5 – Тестовый файл для модуля АЛУ (2)

Данный тест позволит пройти по всем обрабатываемым в модуле АЛУ операциям и выведет их результат в консоль. Чтобы проверить корректность работы данного теста, укажем в качестве рассматриваемых значений $op_a = 10$, $op_b = 3$.



Wave - Default		Msgs					
/tb_lab_MS_SV4/INST	-No Data-	ADD 10 3	SUB 10 3	MUL 10 3	DIV 10 3	VAR 10 3	ADD 10 3
/tb_lab_MS_SV4/AL...	-No Data-	13	7	30	3	1	13

Рис. 6 – Моделирование тестового файла средствами ModelSim (wave)

```

# .main_pane.objects.interior.cs.body.tree
# run -all
#     ADD:   ALU_out = 10 + 3 = 13;
#     SUB:   ALU_out = 10 - 3 = 7;
#     MUL:   ALU_out = 10 * 3 = 30;
#     DIV:   ALU_out = 10 / 3 = 3;
#     VAR:   ALU_out = 10 % 3 = 1;
# ** Note: $stop      : D:/Users/Legion/Quartus an

```

Рис. 7 – Моделирование тестового файла средствами ModelSim (cmd)

Заметим, что данный тест ограничен количеством одной парой значений. Чтобы провести полноценное тестирование модифицируем файл `tb` таким образом, чтобы он охватывал все возможные пары значений op_a и op_b (реализуем с помощью вложенного цикла `for`):

```

lab_MS_SV4 - tb_lab_MS_SV4.sv

1  `timescale 1ps/1ps
2
3  import lab_MS_SV4_pack::*;
4
5  module tb_lab_MS_SV4();
6      INST_t INST;
7      data_y ALU_out;
8
9      lab_MS_SV4 UUT (.ALU_out, .INST);
10
11     initial begin
12         for (int i = 0; i < 256; i++) begin
13             for (int j = 0; j < 256; j++) begin
14                 INST.op_a = i;
15                 INST.op_b = j;
16                 #10;
17                 for (int op = 0; op < 5; op++) begin
18                     case (INST.opc)
19                         ADD: $display("\tADD: \tALU_out =
20 +                      SUB: $display("\tSUB: \tALU_out =
21 =                      MUL: $display("\tMUL: \tALU_out =
22 ;#, INST.op_a, INST.op_b, INST.opc, ALU_out);
23 ;#, INST.op_a, INST.op_b, INST.opc, ALU_out);
24 ;#, INST.op_a, INST.op_b, INST.opc, ALU_out);
25 ;#, INST.op_a, INST.op_b, INST.opc, ALU_out);
26 ;\n", INST.op_a, INST.op_b, INST.opc, ALU_out);
27                 end
28             end
29         end
30         while (INST.opc != INST.opc.first());
31         #10;
32         $stop;
33     end
34 endmodule
35

```

Рис. 8 – Модифицированный модуль для теста первого класса

Теперь смоделируем созданный файл средствами ModelSim и проанализируем полученные данные:

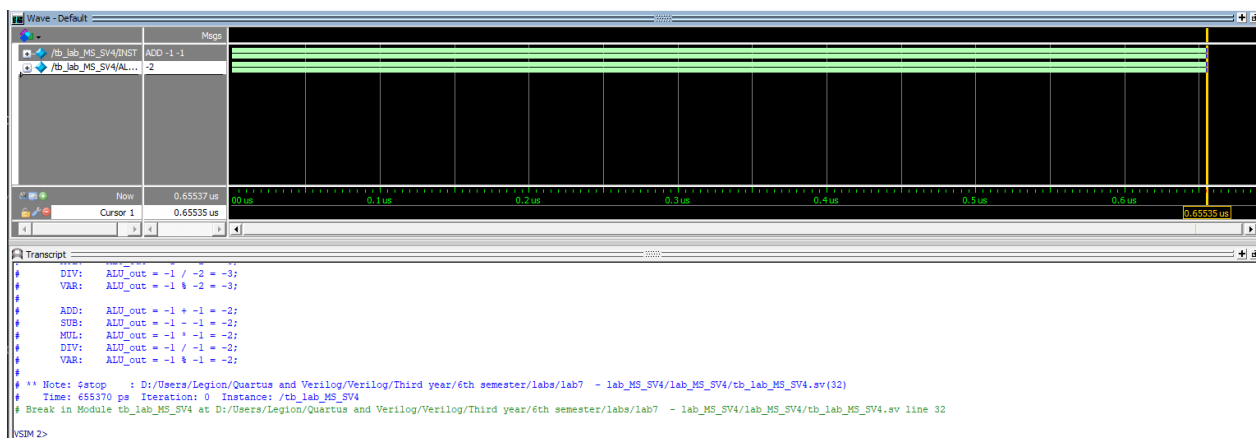


Рис. 9 – Общая вывод после симуляции модифицированного tb

Рассмотрим отдельно краевые значения:

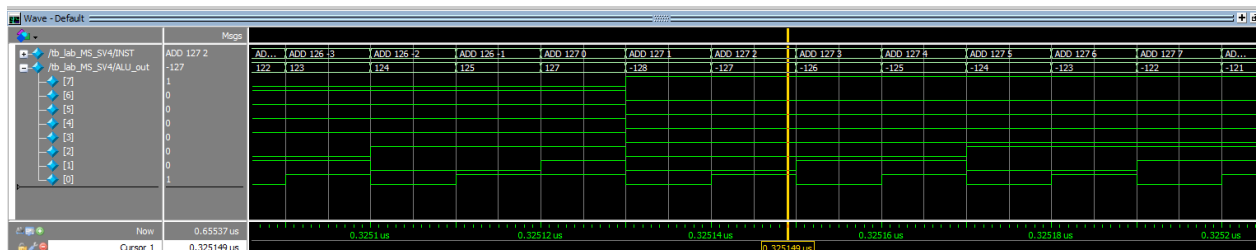


Рис. 10 – Результат SLU_out > 127

На рисунке выше можем увидеть, что выводится отрицательное число, однако, если раскрыть его и посмотреть на его бинарный код, то станет понятно, что это связано с тем, что мы выводим значения в дополнительном коде.

Также, в код добавлены обработки специальных случаев, которые выводятся в консоль. На рисунке ниже представлен пример такого вывода для обработки ошибочных значений при выполнении операции, заданной по варианту (остаток от деления на ноль):

```
# VAR: ALU_out = -12 % -3 = 0;
# VAR: ALU_out = -12 % -2 = 0;
# VAR: ALU_out = -12 % -1 = 0;
** Error: 3248650 Modulo by zero occurred: Error 0 VAR 0
# Time: 3248650 ps Scope: tb_lab_MS_SV4 File: D:/Users/Legion/Quartus and Verilog/Verilog/Third year/6th semester/labs/lab7 - lab_MS_SV4/lab_MS_SV4/tb_lab_MS_SV4.sv Line: 46
# VAR: ALU_out = -11 % 1 = 0;
# VAR: ALU_out = -11 % 2 = -1;
```

Рис. 11 – Обработка случая остатка от деления на 0

```
# DIV: ALU_out = -81 / -2 = 40;
# DIV: ALU_out = -81 / -1 = 81;
** Error: 2416650 Division by zero occurred: Error 0 DIV 0
# Time: 2416650 ps Scope: tb_lab_MS_SV4 File: D:/Users/Legion/Quartus and Verilog/Verilog/Third year/6th semester/labs/lab7 - lab_MS_SV4/lab_MS_SV4/tb_lab_MS_SV4.sv Line: 37
# DIV: ALU_out = -80 / 1 = -80;
# DIV: ALU_out = -80 / 2 = -40;
```

Рис. 12 – Обработка случая деления на 0

Остальные граничные случаи также обрабатываются корректно, однако в конкретном тесте сложно найти их выводы из-за обработки всех возможных значений (синтаксис обработки таких случаев схож с приведенным выше примером, поэтому можно просто посмотреть код, чтобы увидеть, что будет выводиться в конкретном случае).

Как видим, все операции работают корректно и выводят верный результат.

2.3. Создания модуля для тестирования на плате

Теперь разработаем модуль для тестирования программы на плате:

```
lab_MS_SV4 - db_lab_MS_SV4.sv

1 import lab_MS_SV4_pack::*;
2
3 module db_lab_MS_SV4(
4     (* altera attribute = "-name IO_STANDART \"3.3-V LVCMOS\"", chip_pin = "23" *)
5     input CLK
6 );
7     INST_t INST;
8     data_y ALU_out;
9
10 lab_MS_SV4 UUT (.ALU_out, .INST);
11
12     sp_unit SP_ (
13         .source (INST), // sources.source
14         .probe (ALU_out), // probes.probe
15         .source_clk (CLK) // source_clk.clk
16     );
17 endmodule
18
```

Рис. 13 – Модуль db для тестирования на плате

2.4. Настройка Signal Tap II

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:

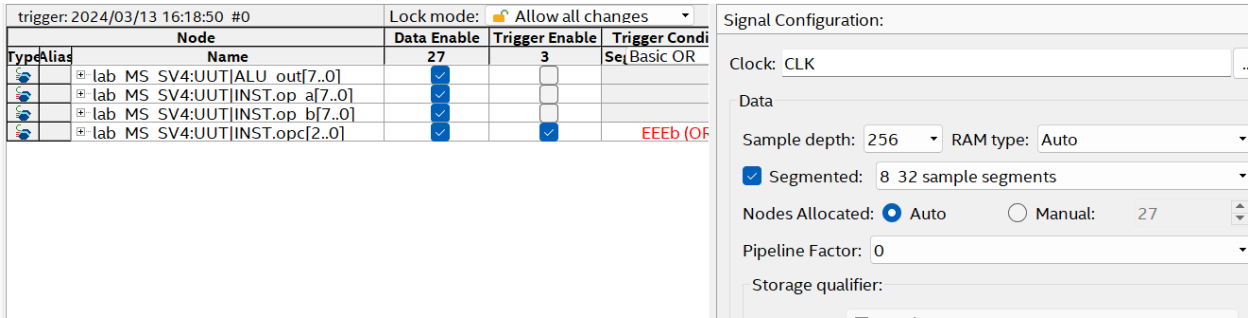


Рис. 14 – Настройка окна Signal Tap II

Перегруппируем S[18..0] сигналы в SP так, чтобы первая подгруппа (S[18..16]) отвечала за номер операции в пакете, вторая подгруппа (S[15..8]) отвечала бы за первое число op_a, а третья подгруппа (S[7..0]) – за число op_b:

0			
Index	Type	Alias	Name
P[7..0]			ALU out
S[18..16]			operation
S[15..8]			op a
S[7..0]			op b

Рис. 15 – Настройка Signal Probe

Составим мнемоническую таблицу, чтобы при симуляции отображался не просто код операции, а её название:

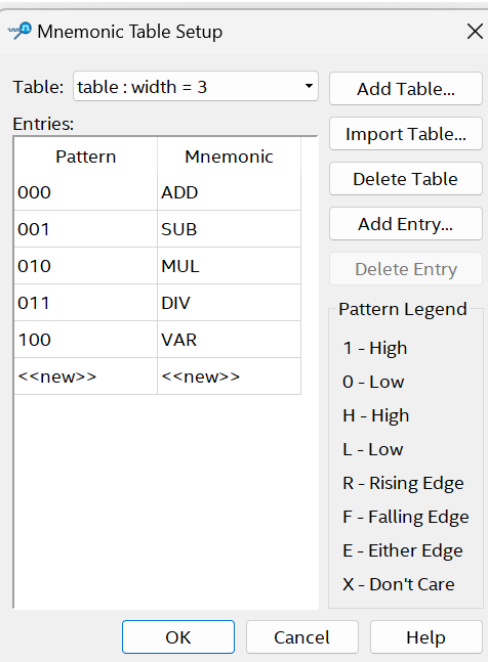


Рис. 16 – Мнемоническая таблица

2.5. Тестирование на плате средствами Signal Tap II

Теперь запустим и выполним проверку корректности работы программы на плате. Выполним загрузку разработанного модуля на плату и запустим тестирование:

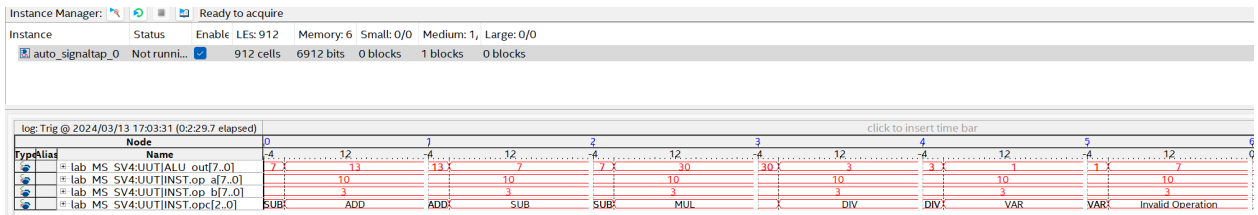


Рис. 17 – Симуляция в Signal Probe (op_a = 10, op_b = 3)

Как видим, значение было посчитано верно.

Теперь будем последовательно выполнять ввод номера операции, а также сами значения переменных непосредственно в редакторе SP, чтобы значение ALU_out выводилось в режиме реального времени. Примеры таких тестов для одной пары чисел представлены ниже (op_a = 10, op_b = 3):

Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
P[7..0]			ALU out	13					13				
S[18..16]			operation	0					0				
S[15..8]			op a	10					10				
S[7..0]			op b	3					3				

Рис. 18 – Результат SP (ADD, op_a = 10, op_b = 3)

Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
P[7..0]			ALU out	7					7				
S[18..16]			operation	1					1				
S[15..8]			op a	10					10				
S[7..0]			op b	3					3				

Рис. 19 – Результат SP (SUB, op_a = 10, op_b = 3)

Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
P[7..0]			ALU out	30					30				
S[18..16]			operation	2					2				
S[15..8]			op a	10					10				
S[7..0]			op b	3					3				

Рис. 20 – Результат SP (MUL, op_a = 10, op_b = 3)

Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
P[7..0]			ALU out	3					3				
S[18..16]			operation	3					3				
S[15..8]			op a	10					10				
S[7..0]			op b	3					3				

Рис. 21 – Результат SP (DIV, op_a = 10, op_b = 3)

Index	Type	Alias	Name	Data	-8	-7	-6	-5	-4	-3	-2	-1	0
P[7..0]			ALU out	1					1				
S[18..16]			operation	4					4				
S[15..8]			op a	10					10				
S[7..0]			op b	3					3				

Рис. 22 – Результат SP (VAR, op_a = 10, op_b = 3)

Протестируем случай с переполнением:

0						-8	-7	-6	-5	-4	-3	-2	-1
Index	Type	Alias	Name	Data									
P[7..0]			ALU out	0									
S[18..16]			operation	0									
S[15..8]			op a	-127									
S[7..0]			op b	127									

Рис. 23 – Результат SP (ADD, op_a = 127, op_b = 127)

Заметим, что результат выдаётся неверный. Это связано с тем, что происходит переполнение, поэтому при вводе есть ограничение на вводимые числа. Чтобы её вычислить надо рассмотреть операцию умножения, так как эта операция позволяет получить наибольшую разрядность выходного числа при одних и тех же входных переменных (размерности переменных складываются).

Выполним полную компиляцию. В отчете о компиляции видно, что устройство удовлетворяет временным параметрам.

Slow 1000mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	71.16 MHz	71.16 MHz	altera_reserved_tck	

Рис. 24 – Временные характеристики устройства

3. Вывод

В результате создания проекта на разработку АЛУ с параметризированной разрядностью и использованием пакетирования в SystemVerilog было достигнуто несколько значимых результатов.

Во-первых, использование пакета для хранения типов данных и определения кодов операций позволило сделать код более читаемым и модульным. Это упростило разработку и отладку проекта, так как изменения в типах данных или кодах операций можно внести в едином месте, и они автоматически отразятся на всех остальных частях проекта.

Во-вторых, модульность проекта была улучшена благодаря пакетированию. Каждый модуль (ALU, тестовое окружение, модуль для тестирования на плате) был разделен на отдельные компоненты, что сделало проект более гибким и масштабируемым. Это позволяет легко расширять функциональность проекта или переиспользовать его компоненты в других проектах.

Наконец, использование пакета способствует повышению производительности и уменьшению вероятности ошибок. За счет того, что типы данных и коды операций определены один раз и использованы во всех частях проекта, уменьшается вероятность ошибок при внесении изменений и обеспечивается единообразие данных в проекте.

Таким образом, применение пакетирования в проекте на разработку АЛУ существенно облегчило процесс разработки, повысило его модульность и гибкость, а также способствовало повышению производительности и уменьшению вероятности ошибок.