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_2

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

Список иллюстраций

1. Задание

1.1. Цель работы

Лабораторная работа выполняется по индивидуальным заданиям. Теория и приведенные ниже (на английском языке) примеры могут быть использованы как образцы для составления собственных исходных кодов проектируемых модулей и тестов. В работе следует использовать конструкции SystemVerilog.

1.2. Программа работы:

- Разработать оптисание LFSR по индивидуальному заданию (номер задания = номеру в списке группы)
- Разработать тест для проверки LFSR и провести моделирование
- По результатам моделирования в ModelSim **необходимо доказать**, что период повторения равен 2N-1, где N – максимальная степень полинома из задания (ОБРАТИТЕ ВНИМАНИЕ: в некоторых вариантах задания, теоретически, может быть ошибка, тогда этот период будет меньше 2N-1. Если у Вас так получилось, то надо, прежде всего, проверить правильность своего описания, и только после этого обсудить это с преподавателем).
- Разработать модуль для построения гистограммы
- Разработать тест для проверки модуля построения гистограммы и провести моделирование
- По результатам моделирования в ModelSim необходимо доказать, что для входных данных, поступающих со счетчика (шаг счета = номеру студента в группе) получена правильная гистограмма в модуле памяти.
- Разработать модуль верхнего уровня lab_MS_SV3, содержащий LFSR и модуль построения гистограммы.
- Разработать тест для проверки модуля верхнего уровня иерархии (tb_lab_MS_SV3).
- По результатам моделирования необходимо доказать, что период повторения LSFR равен 2N-1, где N – максимальная степень полинома из задания и модуль построения гистограммы создает правильную гистограмму.
- Разработать модуль верхнего уровня для отладки db_lab_MS_SV3, содержащий: модуль lab_MS_SV3; модуль SP_unit (модуль, обеспечивающий возможность задания входных управляющих сигналов без использования кнопок на плате). Модуль должен содержать подключение только к тактовому сигналу на плате.
- Настроить логический анализатор для проведения исследования и отладки реализуемого на плате db_lab_MS_SV3.
- Провести анализ работы db_lab_MS_SV3 и доказать (зафиксировав результаты снимками экрана), что:
    - Модуль управляется входными сигналами RST и ENA
    - Правильно формируется гистограмма
    - Формируемые псевдослучайные данные отображаются аналогично изображению на Figure 17.

- Изменение модуля histogram_unit

- В модуле histogram_unit (см. Figure 7) замените память mem_in, созданную как массив, на параметризированный (IP) блок RAM:1_PORT
  - 10 бит – разрядность слова, 1024 слова, Read-During Write – OLD DATA;
  - Включите Allow in-System-Memory-Content Editor и задайте Instance ID = RAM0

- Осуществите моделирование в пакете ModelSim (не забудьте подключить библиотеку altera_mf_ver) и убедитесь в правильности работы модуля.

- В модуле db_lab_MS_SV3 настройте в логическом анализаторе 16 сегментов (по 16 отсчетов, положение – в центре) условие захвата – достижение генератором значения, равного Вашему номеру в списке группы.
- Запустите ISSP, убедитесь, что RST=1 и ENA= 1;
- Запустите IMCE: считайте данные из модуля памяти с Instance ID = RAM0. Зафиксируйте их и объясните полученные результаты.
- Откройте окно Логического анализатора SignalTapII и запустите захват данных
- Откройте окно ISSP, установите RST=0
- Откройте окно IMCE: считайте данные из модуля памяти с Instance ID = RAM0. Зафиксируйте их и объясните полученные результаты.
- Откройте окно Логического анализатора SignalTapII, зафиксируйте и приведите в отчете временные диаграммы, убедитесь в том, что:
  - временная диаграмма похожа на приведенную ниже
  - формируемые генератором данные, повторяются во всех 16-х сегментах.
  - объясните почему на выходе mem_out мы видим числа 0, 1, 2, …15.
    - если увеличить количество сегментов для захвата данных, то до какого занчения будут увеличиваться эти числа?
      - Как это объяснить?
      - Докажите это с использованием SignalTapII

1.3. Вариант по заданию

№ варианта	**Полином для реализации**	**Тип реализации** **Тип логического элемента обратной связи**
6	$x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + x^1 + 1$	Фибоначчи XOR

2. Ход решения

2.1. Создание LFSR модуля

В соответствии с заданием разработаем описание LSFR модуля на языке SystemVerilog для следующего полинома:

$$x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + x^1$$

lab_MS_SV3 - LFSR_Fibonacci_8764321.sv

```
`timescale 1ns/1ns
module LFSR_Fibonacci_8764321 (
    input bit CLK,
    input bit RST,
    input bit ENA,
    output bit [8:1] LFSR_out
);
always_ff @(posedge CLK, posedge RST)
if (RST) LFSR_out <= 8'd1;
else if (ENA)
    if (LFSR_out == '0)
        LFSR_out <= 8;
    else
        LFSR_out <= {LFSR_out[7:1],LFSR_out[8]*LFSR_out[7]*LFSR_out[6]*LFSR_out[4]*LFSR_out[3]*LFSR_out[2]*LFSR_out[1]};
endmodule

```

Рис. 1 – LFSR модуль

Данный модуль – сдвигающий регистр, с асинхронным сбросом. Сброс выполняется в 1 т. к. в противном случае LFSR перестанет работать.

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

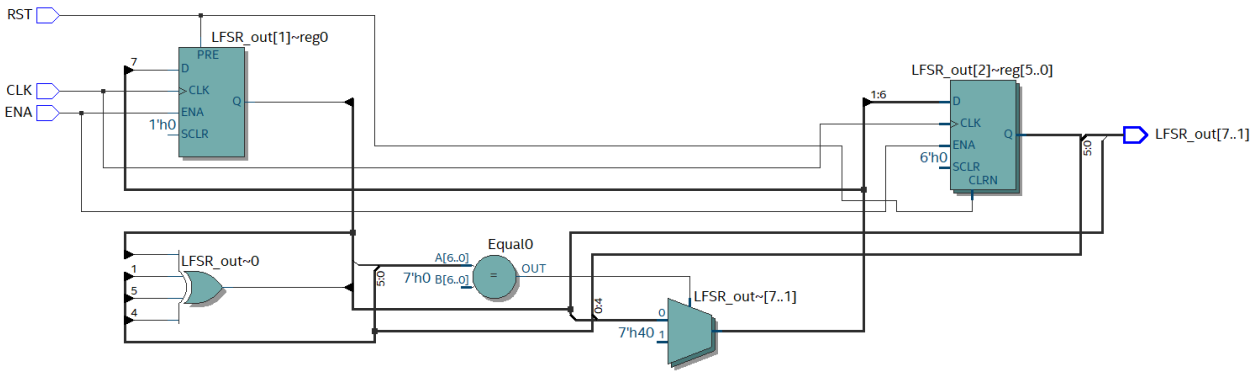

Рис. 2 – Структура LFSR модуля в RTL Viewer

Напишем тест для только что созданного модуля LFSR:

lab_MS_SV3 - tb_LFSR_8764321_F.sv

```
1  `timescale 1ns / 1ns
2  module tb_LFSR_8764321_F ();
3    bit       CLK;
4    bit       RST;
5    bit       ENA;
6    bit [8:1] LFSR_out;
7    bit LFSR_CYCLE;
8
9    LFSR_8764321_F LFSR_8764321_F_inst (.*);
10
11   localparam CLK_PERIOD = 20;
12
13   initial forever #(CLK_PERIOD / 2) CLK = ~CLK;
14
15   bit [8:1] CNT_int = '0;
16
17   bit [8:1] LFSR_out_start = '0;
18
19   initial begin
20     RST = '1;
21     #(CLK_PERIOD * 3 / 4);
22     RST = '0;
23     #(CLK_PERIOD * 5 / 4);
24     ENA = '1;
25     forever begin
26       @(negedge CLK);
27       if (CNT_int == '0) LFSR_out_start = LFSR_out;
28       else
29       if (LFSR_out_start == LFSR_out) begin
30         LFSR_CYCLE = '1;
31         break;
32       end
33       CNT_int ++;
34    end
35     #(CLK_PERIOD * 5);
36     $stop;
37   end
38
39 endmodule
40
```

Рис. 3 – Тестовый файл для модуля LFSR

Данный модуль позволит нам посчитать период. Т. к. степень полинома 8, мы ожидаем период равный $2^8 - 1 = 256 - 1 = 255$.

Для проверки ожидаемых результатов протестируем работу модуля средствами ModelSim. При этом получим следующую waveform'у:

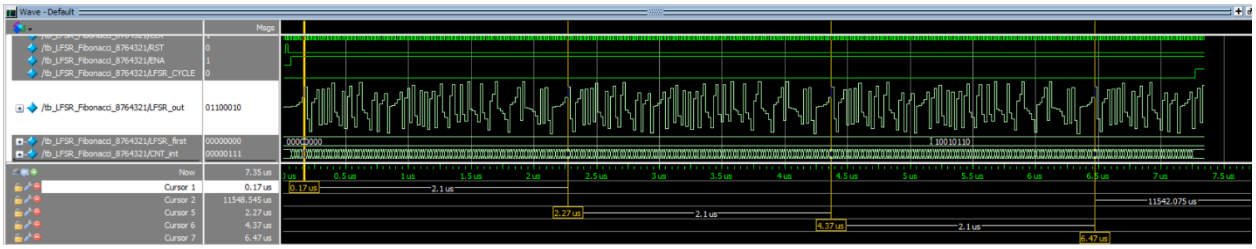

Рис. 4 – Моделирование тестового файла средствами ModelSim (вар. 1)

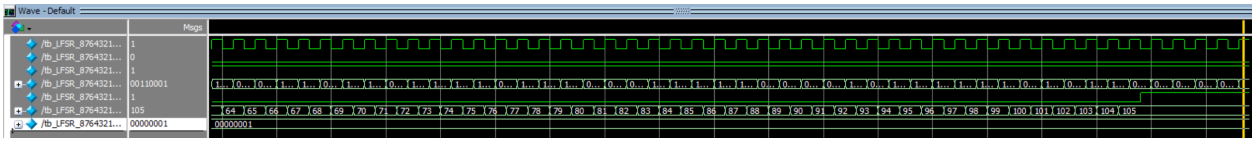

Рис. 5 – Моделирование тестового файла средствами ModelSim (вар. 2)

Как видно из полученной waveform, ожидаемый период не был получен. Однако, сигнал LFSR_CYCLE (3-й снизу сигнал) при подаче последнего сигнала перешёл в

состояние 1. Это значит, что был пройден полный цикл. Фрагмент кода с модулем LFSR_CYCLE представлен ниже:

lab_MS_SV3 - tb_LFSR_8764321_F.sv

```
27  if (CNT_int == '0) LFSR_out_start = LFSR_out;
28        else
29        if (LFSR_out_start == LFSR_out) begin
30          LFSR_CYCLE = '1;
31          break;
32        end
33        CNT_int ++;
```

Рис. 6 – Проверка на полный проход цикла (сигнал LFSR_CYCLE)

2.2. Создание модуля гистограммы

Теперь создадим модуль для создания гистограммы:

lab_MS_SV3 - histogram_unit.sv

```
1  `timescale 1ns / 1ns
2  module histogram_unit #(
3    parameter MAX_NUMBER = ((1 << 8) - 1),
4    parameter SIZE       = 8
5  ) (
6    input  bit                            CLK,
7    input  bit [$clog2(MAX_NUMBER) - 1:0] d_in,
8    input  bit                            RST,
9    input  bit                            ENA,
10   output bit [             SIZE - 1:0] mem_out
11 );
12
13   bit [SIZE - 1:0] mem_in;
14   bit [$clog2(MAX_NUMBER) - 1:0] adr_in, adr_clear;
15
16   bit [$clog2(MAX_NUMBER) - 1:0] d_in_temp;
17
18   bit                            clk_50;
19   bit [SIZE - 1:0] mem_out_2;
20
21   RAM RAM_inst (
22     .address(adr_in),
23     .data   (mem_in),
24     .clock  (!clk_50),
25     .wren   (ENA),
26     .q      (mem_out_2)
27   );
28
29   always_ff @(posedge CLK) d_in_temp <= d_in;
30   always_ff @(negedge CLK) mem_out <= mem_out_2;
31
32   PLL PLL_inst (
33     .inclk0(CLK),
34     .c0    (clk_50)
35   );
36
37   assign mem_in = RST ? '0 : (ENA ? mem_out_2 + !CLK : mem_out_2);
38   assign adr_in = RST ? adr_clear : d_in_temp;
39
40   always_ff @(negedge clk_50) begin : clearing_array
41     if (~RST) adr_clear <= '0;
42     else adr_clear <= adr_clear + 1'b1;
43   end
44
45 endmodule
46
```

Рис. 7 – Модуль гистограммы

Здесь MAX_NUMBER – параметр, определяющий максимальное число в гистограмме, а SIZE – размерность данных в гистограмме.

Данный модуль использует память mem_arr, в которой каждый такт по введенному адресу (d_in) добавляется единица, также присутствует возможность отчистки памяти, однако для полной отчистки понадобится MAX_NUMBER тактов.

Получившаяся RTL схема выглядит следующим образом:

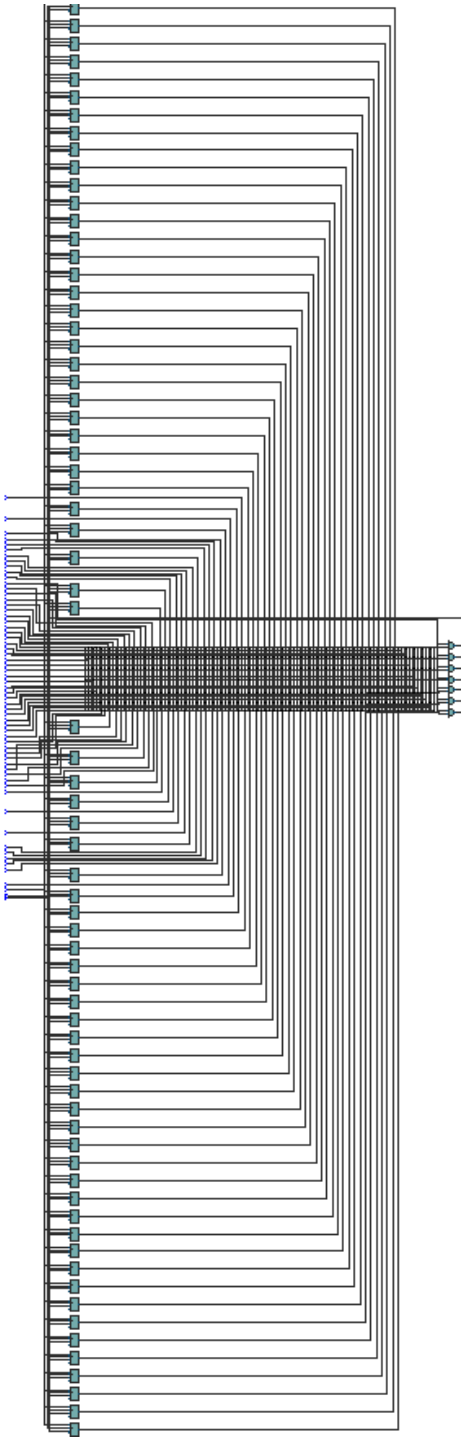

Рис. 8 – Структура модуля гистограммы в RTL Viewer

Как можно заметить, тут нет как таковой памяти, все строится на регистрах. Причина этого в том, что мы пытаемся одновременно читать значения из памяти и записывать в неё, из-за чего Quartus решает, что это лучше сделать, используя регистровые схемы.

Теперь напишем тест для созданного модуля:

lab_MS_SV3 - tb_histogram_unit.sv

```
`timescale 1ns / 1ns
module tb_histogram_unit ();
  parameter SIZE = 8;
  parameter MAX_NUMBER = 255;
  bit [$clog2(MAX_NUMBER) - 1:0] d_in;
  bit                            CLK;
  bit                            ENA;
  bit                            RST;
  bit [             SIZE - 1:0] mem_out;

  localparam CLK_PERIOD = 20;

  initial forever #(CLK_PERIOD / 2) CLK = ~CLK;

  histogram_unit #(
    .SIZE      (SIZE),
    .MAX_NUMBER(MAX_NUMBER)
  ) histogram_unit_inst (
    .*
  );

  initial begin
    ENA = '0;
   #(CLK_PERIOD * 10);
    ENA = '1;
   #(CLK_PERIOD);
    for (int i = 0; i < (MAX_NUMBER + 1) * 8; i++) begin
      @(negedge CLK) d_in += 6;
    end
   @(negedge CLK)
   d_in += 1;
   #(CLK_PERIOD * 3 / 2);
    RST = '1;
    @(negedge CLK)
   @(negedge CLK)
    $stop;
  end

endmodule
```

Рис. 9 – Тестовый файл для модуля гистограммы

При прогонке данного тестового модуля средствами ModelSim получим следующую временную диаграмму:

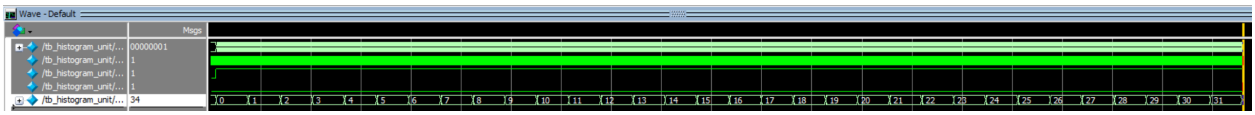

Рис. 10 – Результаты моделирования тестового файла модуля гистограммы

Здесь же, открыв вкладку Memory List, можем увидеть данные в памяти:

00000000	0	0	0	0	32	0
00000006	32	0	32	0	32	0
0000000c	32	0	32	0	32	0
00000012	32	0	32	0	32	0
00000018	32	0	32	0	32	0
0000001e	32	0	32	0	32	0
00000024	32	0	32	0	32	0
0000002a	32	0	32	0	32	0
00000030	32	0	32	0	32	0
00000036	32	0	32	0	32	0
0000003c	32	0	32	0	32	0
00000042	32	0	32	0	32	0
00000048	32	0	32	0	32	0
0000004e	32	0	32	0	32	0
00000054	32	0	32	0	32	0
0000005a	32	0	32	0	32	0
00000060	32	0	32	0	32	0
00000066	32	0	32	0	32	0
0000006c	32	0	32	0	32	0
00000072	32	0	32	0	32	0
00000078	32	0	32	0	32	0
0000007e	32	0				

Рис. 11 – Данные в памяти

Как мы видим, модуль гистограммы корректно обрабатывает входную последовательность, а также очищает данные по RST.

2.3. Создание модуля верхнего уровня

Теперь напишем модуль верхнего уровня, объединив LFSR и модуль гистограммы, чтоб проверить, что в LFSR все случайные числа равновероятны:

lab_MS_SV3 - lab_MS_SV_3.sv

```
1  `timescale 1ns / 1ns
2  module lab_MS_SV_3 (
3    input  bit       CLK,
4    input  bit       RST,
5    input  bit       ENA,
6    output bit [7:0] mem_out
7  );
8
9    bit [7:0] LFSR_out;
10   bit [7:0] d_in;
11
12   LFSR_8764321_F LFSR_8764321_F_inst (
13     .CLK,
14     .RST,
15     .ENA,
16     .LFSR_out
17   );
18
19   assign d_in = LFSR_out;
20
21   histogram_unit histogram_unit_inst (
22     .CLK,
23     .d_in,
24     .RST,
25     .ENA,
26     .mem_out
27   );
28
29 endmodule
30
```

Рис. 12 – Модуль верхнего уровня

Аналогично тому. Как делали с предыдущими модулями, напишем тестовый модуль:

lab_MS_SV3 - tb_lab_MS_SV_3.sv

```
1  `timescale 1ns / 1ns
2  module tb_lab_MS_SV_3 ();
3    bit       CLK;
4    bit       RST;
5    bit       ENA;
6    bit [7:0] mem_out;
7
8    lab_MS_SV_3 lab_MS_SV_3_unit (.*);
9
10   localparam CLK_PERIOD = 20;
11
12   initial forever #(CLK_PERIOD / 2) CLK = ~CLK;
13
14   initial begin
15     RST = '1;
16     ENA = '1;
17     #(CLK_PERIOD / 2);
18     RST = '0;
19     repeat(255 * 5) @(negedge CLK);
20    @(posedge CLK)
21     $stop;
22   end
23
24 endmodule
25
```

Рис. 13 – Тестовый файл для модуля верхнего уровня

Мы 12 раз повторяем период, который был получен ранее, ожидая, что в гистограмме все значения от 1 до 255 будут равны 5. Проверим это:

00000000	12	12	12	12	12	12
00000006	12	12	12	12	12	12
0000000c	12	12	12	12	12	12
00000012	12	12	12	12	12	12
00000018	12	12	12	12	12	12
0000001e	12	12	12	12	12	12
00000024	12	12	12	12	12	12
0000002a	12	12	12	12	12	12
00000030	12	12	12	12	12	12
00000036	12	12	12	12	12	12
0000003c	12	12	12	12	12	12
00000042	12	12	12	12	12	12
00000048	12	12	12	12	12	12
0000004e	12	12	12	12	12	12
00000054	12	12	12	12	12	12
0000005a	12	12	12	12	12	12
00000060	12	12	12	12	12	12
00000066	12	12	12	12	12	12
0000006c	12	12	12	12	12	12
00000072	12	12	12	12	12	12
00000078	12	12	12	12	12	12
0000007e	12	0				

Рис. 14 – Данные в памяти гисторграммы

Как мы видим, результат соответствует ожиданиям.

2.4. Создания модуля для тестирования на плате

Теперь разработаем модуль для тестирования на плате:

lab_MS_SV3 - db_lab_MS_SV_3.sv

```
module db_lab_MS_SV_3 (
  (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23" *)
  input bit CLK
);

  bit       RST = 1'b1;
  bit       ENA = 1'b1;
  bit [7:0] mem_out;

  lab_MS_SV_3 lab_MS_SV_3_ints (.*);
  SP_unit SP_unit_inst (
    .source    ({RST, ENA}),
    .source_clk(CLK)
  );
endmodule

```

Рис. 15 – Модуль db для тестирования на плате

2.5. Настройка Signal Tap II

Для отображения выводов конечного автомата создадим мнемоническую. Зададим следующие настройки Signal Tap II:

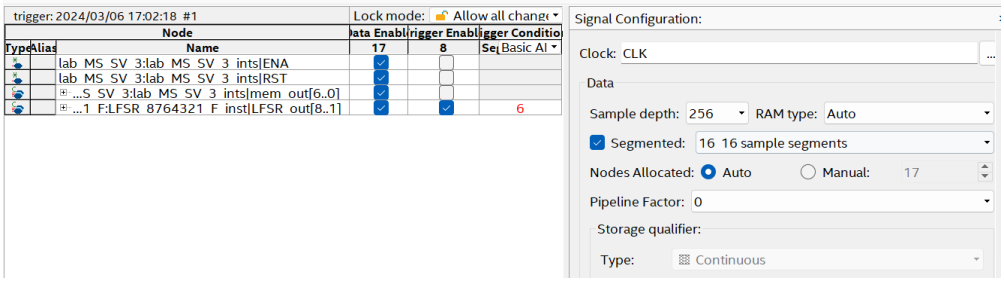

Рис. 16 – Настройка окна Signal Tap II

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

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	58.13 MHz	58.13 MHz	altera_reserved_tck	
2	84.86 MHz	84.86 MHz	CLK	

Рис. 17 – Временные характеристики устройства

Теперь запустим и выполним проверку корректности работы программы на плате. Выполним загрузку разработанного модуля на плату и запустим тестирование, переведя RST в 0:

0

Index	Type	Alias	Name	Data
S1		RST	source[1]	0
S0		ENA	source[0]	1

Рис. 18 – Настройка ISSP

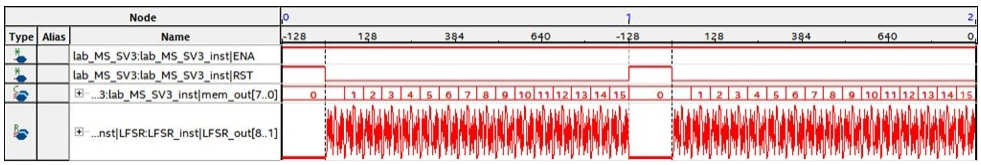

Рис. 19 –

Как мы видим, LFSR выдает случайные числа, а mem_out увеличивается с 0 до 13 т.е. ведется подсчет циклов с начала работы.

2.6. Доработка модуля гисторграммы

Теперь в соответствии с заданием в модуле histogram необходимо заменить mem_arr на однопортовую память. Проблема состоит в том, что в один такт необходимо выполнить считывание значения из памяти и на основании этого записать в ту же ячейку новые данные. Выполнить это за один такт нереально, поэтому необходимо добавить PLL, который умножит внутреннюю частоту. Схема будет следующей: на фронте clk сохраняется значение на входе d_in, после чего на спаде clk_50 (clk с частотой в 2 раза большей) мы загрузим на вход памяти адрес, а на второй спад мы на вход памяти поместим обновленные данные:

lab_MS_SV3 - histogram_unit.sv

```
`timescale 1ns / 1ns
module histogram_unit #(
  parameter MAX_NUMBER = ((1 << 8) - 1),
  parameter SIZE       = 8
) (
  input  bit                            CLK,
  input  bit [$clog2(MAX_NUMBER) - 1:0] d_in,
  input  bit                            RST,
  input  bit                            ENA,
  output bit [              SIZE - 1:0] mem_out
);

  bit [SIZE - 1:0] mem_in;
  bit [$clog2(MAX_NUMBER) - 1:0] adr_in, adr_clear;

  bit [$clog2(MAX_NUMBER) - 1:0] d_in_temp;

  bit                            clk_50;
  bit [SIZE - 1:0] mem_out_2;

  RAM RAM_inst (
    .address(adr_in),
    .data   (mem_in),
    .clock  (!clk_50),
    .wren   (ENA),
    .q      (mem_out_2)
  );

  always_ff @(posedge CLK) d_in_temp <= d_in;
  always_ff @(negedge CLK) mem_out <= mem_out_2;

  PLL PLL_inst (
    .inclk0(CLK),
    .c0    (clk_50)
  );

  assign mem_in = RST ? '0 : (ENA ? mem_out_2 + !CLK : mem_out_2);
  assign adr_in = RST ? adr_clear : d_in_temp;

  always_ff @(negedge clk_50) begin : clearing_array
    if (~RST) adr_clear <= '0;
    else adr_clear <= adr_clear + 1'b1;
  end

endmodule

```

Рис. 20 – исправленный модуль гистограммы

Теперь память будет стираться в 2 раза быстрее (по 2 адреса за такт). Посмотрим на то, как выглядит RTL схема модуля:

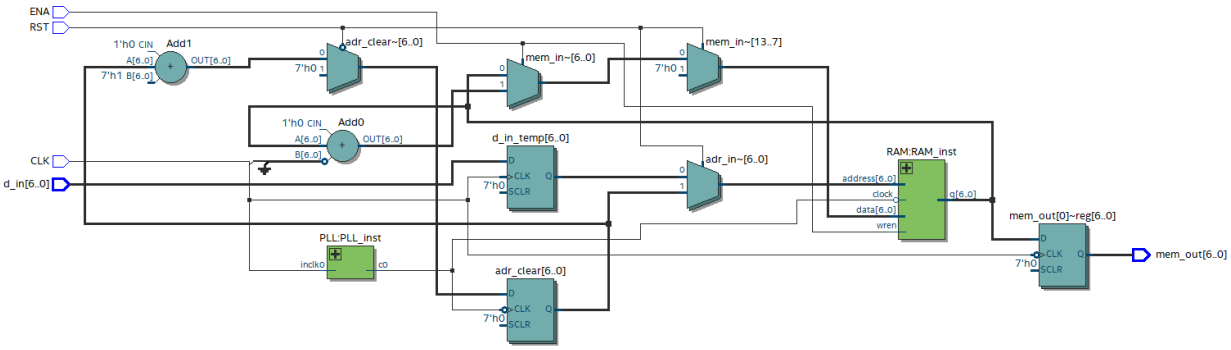

Рис. 21 – Схема исправленного модуля гистограммы в RTL Viewer

Как мы видим, такой способ действительно помог избавиться от регистровой схемы памяти, однако сильно усложнил проект т. к. требует PLL.

Теперь необходимо повторить тестирование этого модуля, чтоб проверить корректность его работы:

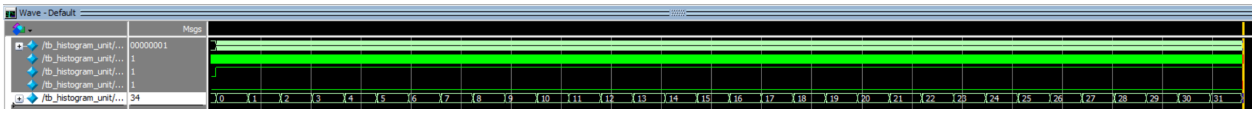

Рис. 22 – Результаты моделирования обновлённого тестового файла модуля гистограммы

Здесь же, открыв вкладку Memory List, можем увидеть данные в памяти:

00000000	0	0	0	0	32	0
00000006	32	0	32	0	32	0
0000000c	32	0	32	0	32	0
00000012	32	0	32	0	32	0
00000018	32	0	32	0	32	0
0000001e	32	0	32	0	32	0
00000024	32	0	32	0	32	0
0000002a	32	0	32	0	32	0
00000030	32	0	32	0	32	0
00000036	32	0	32	0	32	0
0000003c	32	0	32	0	32	0
00000042	32	0	32	0	32	0
00000048	32	0	32	0	32	0
0000004e	32	0	32	0	32	0
00000054	32	0	32	0	32	0
0000005a	32	0	32	0	32	0
00000060	32	0	32	0	32	0
00000066	32	0	32	0	32	0
0000006c	32	0	32	0	32	0
00000072	32	0	32	0	32	0
00000078	32	0	32	0	32	0
0000007e	32	0				

Рис. 23 – Данные в памяти (2)

Как мы видим, модуль гистограммы корректно обрабатывает входную последовательность, а также очищает данные по RST.

Тестирование модуля верхнего уровня также повторило предыдущие тесты, поэтому перейдем сразу к тестированию на плате.

2.7. Тестирование на плате

Важной особенностью новой памяти является то, что её можно посмотреть через ISMC. Если сделать это при RST = 1, мы увидим следующий результат:

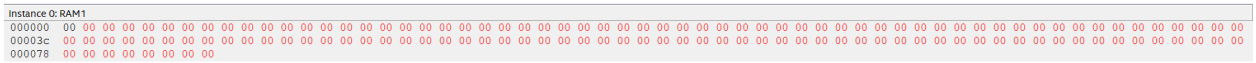

Рис. 24 – Память при RST = 1

И это логично т. к. память постоянно сбрасывается. Теперь поставим RST в 0 и считаем память:

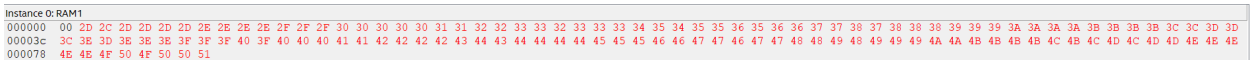

Рис. 25 – Память при RST = 0

Как мы видим, 0 ячейка осталась 0 т. к. наш генератор не должен выдавать это число, а вот остальные ячейки имеют случайные значения, хотя ожидалось что они будут одинаковыми. Это связано с тем, что ISMC не работает на достаточных частотах и не может выгрузить всю память сразу, а делает это постепенно, откуда и возникают отличия.

В Signal Tap II увидим:

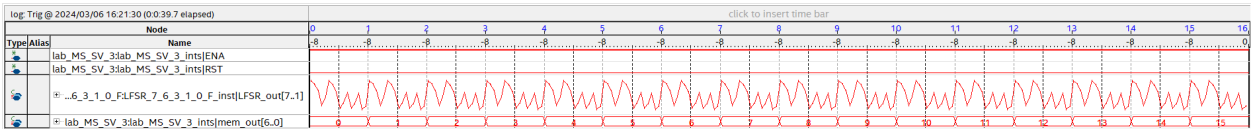

Рис. 26 – Результаты моделирования средствами Signal Tap II

Мы видим, что было зафиксировано 16 отрезков, когда встречалось требуемое число. Оно встречается раз в цикл (как и ожидалось), а также соседние измерения имеют одинаковую форму т. к. цикл формирует одинаковую последовательность чисел.

3. Вывод

В результате выполнения лабораторной работы были успешно разработаны модули с использованием SystemVerilog, что позволило использовать параметризацию для легкой адаптации кода под различные конфигурации. Это значительно упростило разработку, делая ее более гибкой и обеспечивая легкость поддержки проекта при внесении изменений.

Дополнительное удобство заключается в эффективном использовании инструментов In-System Sources and Probes Editor и SignalTap II для отладки. Эти инструменты обеспечили быстрое выявление и исправление ошибок, что существенно ускорило процесс разработки и обеспечило успешное завершение проекта.

Практическое применение этого опыта раскрывается в контексте разработки сложных цифровых систем, таких как генераторы псевдослучайных последовательностей с последующим анализом данных. Применение SystemVerilog и современных инструментов отладки делает цифровое проектирование более гибким и адаптивным к изменяющимся требованиям проекта.