Simulation, debugging and implementing the design with ModelSim and Quartus Prime Lite

самостоятельная работа с использованием SystemVerilog

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

В РАЗДЕЛЕ

ЗАДАНИЕ

Contents

Prerequisites

You should:

- be familiar with basics of logic and digital design,
- be familiar with ***SystemVerilog*** constructs,
- have Quartus Prime Lite installed.

Objectives

You will familiarize yourself with the following:

1. How to create design in Quartus Prime
2. How to simulate design using ModelSim
3. How to debug design on a board using ISSP and SignalTapII.

Before going forward

Before going forward with the Lab be sure that you have the working folder for this lab (C:\Intel_trn\Q_MS_SV\Lab6_1) and the folder is empty.

Create the project in Quartus Prime

1. Start Quartus Prime **Lite** development tool
2. Create a project
   a. Working directory: **C:\Intel_trn\Q_MS_SV\Lab6_1**
   b. Project name: Lab6_1
   c. Top-Level design entity: Lab6_1
   d. Project Type: Empty project
   e. Add files: No files
   f. Device: EP4CE6E22C8
   g. EDA tools: NONE
3. In Quartus Prime window, select **Assignment => Settings...**
4. In the window appeared:
   a. Select **Compilation Process Settings**
      i Set **Use all available processors**
      ii Set **Use Smart Compilation**
      iii Click **OK**

Lab6_1 design files examples

Developing LFSR unit

Introduction to theory

Linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The initial value of the LFSR is called the seed. The seed could be any value except zero. Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. The following table lists some examples of maximal-length polynomials for shift-register lengths up to 16.

Биты, n	Примитивный многочлен	Период, $2^n - 1$	Число примитивных многочленов
2	$x^2 + x + 1$	3	1
3	$x^3 + x^2 + 1$	7	2
4	$x^4 + x^3 + 1$	15	2
5	$x^5 + x^3 + 1$	31	6
6	$x^6 + x^5 + 1$	63	6
7	$x^7 + x^6 + 1$	127	18
8	$x^8 + x^6 + x^5 + x^4 + 1$	255	16
9	$x^9 + x^5 + 1$	511	48
10	$x^{10} + x^7 + 1$	1023	60
11	$x^{11} + x^9 + 1$	2047	176
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4095	144
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8191	630
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383	756
15	$x^{15} + x^{14} + 1$	32767	1800
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535	2048

Figure 1

Let's explore 16-bit LFSR: $X^{16}+X^{14}+X^{13}+X^{11}+1$ in Fibonacci and Galois configurations. The bit positions that affect the next state are called the ***taps***.

Fibonacci configuration.

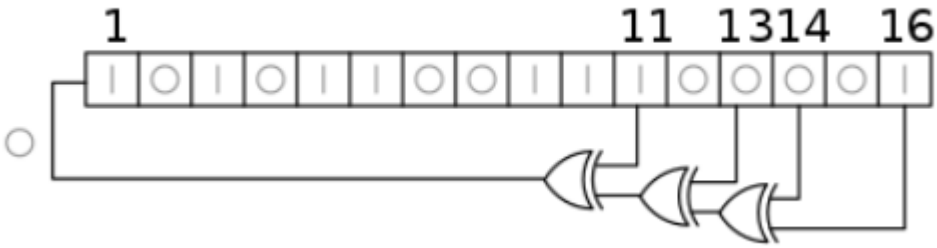

Figure 2

In the diagram the taps are [16,14,13,11]. The rightmost bit of the LFSR is called the output bit. The taps are XOR'd sequentially with the output bit and then fed back into the leftmost bit.
As an alternative to the XOR-based feedback in an LFSR, one can also use XNOR:

- a state with all zeroes is illegal when using XOR feedback.
- a state with all ones is illegal when using an XNOR feedback.

This states are considered illegal because the counter would remain "locked-up" in this state.
This LFSR configuration is also known as standard, external XOR (XNOR) gates.
The alternative is Galois configuration.

It is necessary to ensure that the LFSR never enters an all-zeros state(XOR) or all-ones (XNOR) state, for example by presetting it at start-up to any other state in the sequence.

Galois configuration

$X^{16}+X^{14}+X^{13}+X^{11}+1$ In the Galois configuration, the taps are XORed with the output bit (bit 1) before they are stored in the next position.

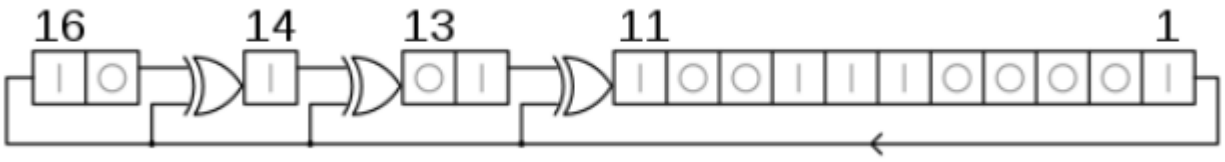

Figure 3

The Galois register shown has the same output stream as the Fibonacci register in the above section. A time offset exists between the streams, so a different start point will be needed to get the same output each cycle.
Galois LFSRs do not concatenate every tap to produce the new input (the XORing is done within the LFSR, and no XOR gates are run in serial, therefore the propagation times are reduced to that of one XOR rather than a whole chain), thus it is possible for each tap to be computed in parallel, increasing the speed of execution.

1 On the Figure 4 you can find an example of LFSR: $X^{10}+X^7+1$ in Fibonacci configuration with XORing.

```
1   `timescale 1ns/1ns
2   module LFSR_10_7_1_F
3       (input bit CLK, input bit RST, input bit ENA, output bit [10:1] LFSR_out);
4
5   always_ff @(posedge CLK, posedge RST)
6   if (RST)    LFSR_out <= 10'd1;
7   else if (ENA)
8           if (LFSR_out == '0)     LFSR_out <= 10'd1;
9           else                    LFSR_out <= {LFSR_out[9:1], LFSR_out[10]^LFSR_out[7]};
10  endmodule
```

Figure 4

2 On the Figure 5 you can find an example of LFSR: $X^{10}+X^7+1$ in Galois configuration with XORing.

```
1   `timescale 1ns/1ns
2   module LFSR_10_7_1_G
3       (input bit CLK, input bit RST, input bit ENA, output bit [10:1] LFSR_out);
4
5   always_ff @(posedge CLK, posedge RST)
6   if (RST)    LFSR_out <= 10'b1111111110;
7   else if (ENA)
8           if (LFSR_out == '0)     LFSR_out <= 10'd1;
9           else                    LFSR_out <= {LFSR_out[1], LFSR_out[10:9], LFSR_out[1]^LFSR_out[8], LFSR_out[7:2] };
10  endmodule
11
```

Figure 5

The both implementations have the same inputs and output.

- Inputs are
  - CLK – clock input.
  - RST – asynchronous reset (when = 1'b1) to the state 10'b0000000001. It is a seed for the LFSR.
  - ENA – enable input (enable to work when is equal to 1'b1).
- Output
  - LFSR_out – 10 bits output of the LFSR.
- Pay attention on:
  - In line 8 the ***if*** is used to ensure that LFSR never enters an all-zeros state.

An example of a source code for the testbench is on Figure 6.

```
1  `timescale 1ns/1ns
2  module tb_LFSR_10_7_1();
3    bit CLK = '0;
4    bit [10:1] LFSR_out    = '0;
5    bit [10:1] LFSR_out_t  = '0;
6    bit [10:1] LFSR_out_G  = '0;
7    bit [10:1] CNT_int     = '0;
8    bit RST = '1;
9    bit ENA = '0;
10
11 LFSR_10_7_1_F LFSR_10_7_1_F_inst (.*);
12 LFSR_10_7_1_G LFSR_10_7_1_G_inst (.LFSR_out (LFSR_out_G), .CLK, .ENA, .RST);
13
14 always  #10 CLK = ~ CLK;
15
16 initial
17 begin
18   #15;
19   RST = '0;
20   #25;
21   ENA = '1;
22 end
23
24 initial
25 begin
26   #45;
27   @(negedge CLK);
28   LFSR_out_t = LFSR_out_G;  //to check Galois
29 //LFSR_out_t = LFSR_out;    //to check Fibonacci
30   forever begin
31     @(negedge CLK);
32     CNT_int++;
33     if (LFSR_out_t == LFSR_out_G) break; //to check Galois
34 //  if (LFSR_out_t == LFSR_out) break;   //to check Fibonacci
35   end
36
37   #20;
38   $stop;
39 end
40 endmodule
```

Figure 6

This simple testbench allows you to check Galois and Fibonacci configuration and be sure that $X^{10}+X^7+1$ is maximal-length polynomial – by the end of simulation the value of CNT_int should be 1023.

Developing the unit

The histogram detection unit allows you to build a histogram for the input data stream. An example of a source code for such unit is on Figure 7.

```
1   `timescale 1ns/1ns
2
3   module histogram_unit(input bit CLK, input bit [9:0] d_in, input bit RST, input bit ENA, output bit [9:0] mem_out);
4   bit [9:0] mem_arr [0:1023];// an array for the histogram
5
6   bit [9:0] mem_in;
7   bit [9:0] adr_in, adr_clear;
8
9   initial
10  begin:initial_val //initial values for the array
11      for (int i=0; i<1024; i++) mem_arr [i] =0;
12  end:initial_val
13
14  assign mem_in   = (RST)? '0         : ((ENA)? (mem_out + 10'd1): mem_out);
15  assign adr_in   = (RST)? adr_clear  : d_in;
16
17  always @(posedge CLK)
18  begin:building_histogram
19      mem_arr [adr_in]    <= mem_in;
20      mem_out             <= mem_arr [adr_in];
21  end:building_histogram
22
23  always_ff @(posedge CLK, negedge RST)
24  begin:clearing_array
25      if (~RST) adr_clear <= '0;
26      else adr_clear <=adr_clear+1'b1;
27  end:clearing_array
28
29  endmodule
```

Figure 7

An example of a source code for the testbench is on Figure 8.

```
1   `timescale 1ns/1ns
2   module tb_histogram_unit();
3   bit [9:0] d_in = '0;
4   bit CLK = '1;
5   bit ENA = '0;
6   bit RST = '0;
7   bit [9:0] mem_out = '0;
8
9   always  #10 CLK = ~ CLK;
10
11  initial
12  begin
13    #25;
14    ENA = '1;
15    #41040;
16   $stop;
17  end
18
19  histogram_unit histogram_unit_inst(.*);
20
21  initial
22  begin
23      for (int i = 0; i < 4095; i++)
24          begin
25              @(negedge CLK)
26              d_in = d_in + 10'd2;
27          end
28
29  end
30  endmodule
```

Figure 8

In this simple testbench an input stream for the histogram_unit is an output of a counter with counting step 2.

A result of simulation proving the right behavior of histogram unit is the content of the memory array (mem_arr). The memory array (mem_arr) should contains for each even address equal values 4.An example is on Figure 9.

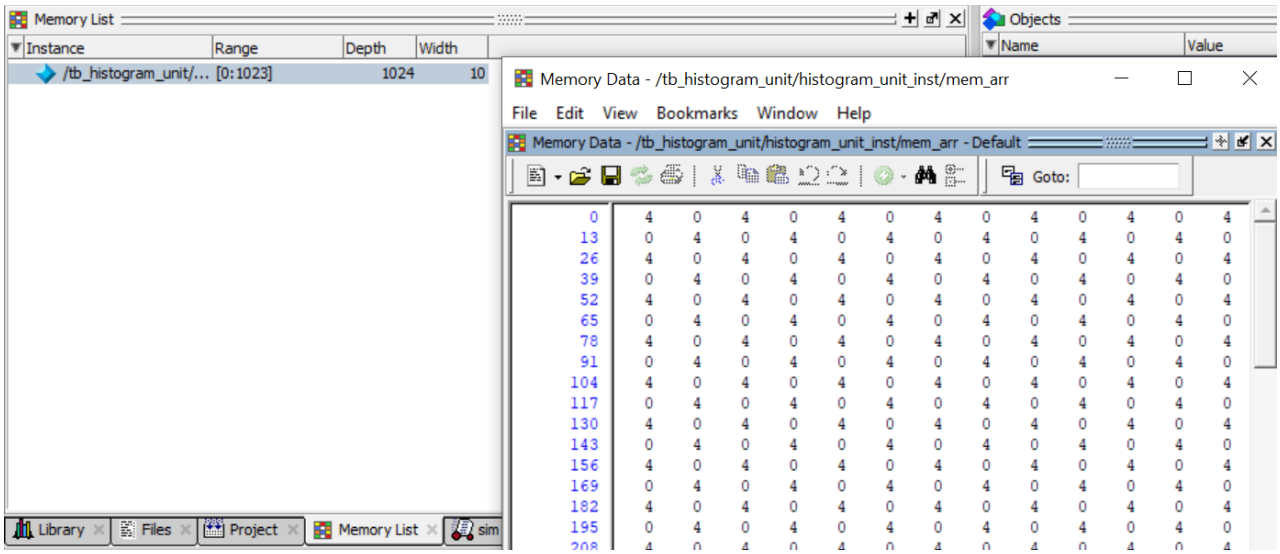

Figure 9

Developing the unit

The top level unit for the Lab contains LFSR unit and Histogram unit connected together. An example of source code is on Figure 10.

```
1   `timescale 1ns/1ns
2
3   module lab6_1(input bit CLK, input bit RST, input bit ENA, output bit [9:0] mem_out);
4
5   bit [9:0] LFSR_out;
6   bit [9:0] d_in;
7
8   assign d_in = LFSR_out;
9
10  LFSR_10_7_1_G LFSR_10_7_1_G_inst (.*);
11  //LFSR_10_7_1_F LFSR_10_7_1_F_inst (.*);
12  histogram_unit histogram_unit_inst(.*);
13
14  endmodule
```

Figure 10

In the top level unit an input stream for the histogram_unit is an output of a LFSR unit (d_in = LFSR_out).

An example of a source code for the testbench is on Figure 11.

```
1   `timescale 1ns/1ns
2
3   module tb_lab6_1();
4
5   bit CLK = '1;
6   bit RST = '0;
7   bit ENA = '0;
8   bit [9:0] mem_out ='0;
9
10  lab6_1 lab6_1_inst (.*);
11  always  #10 CLK = ~ CLK;
12
13  initial
14  begin
15      #25;
16      ENA = '1;
17    #41040;
18   $stop;
19  end
20
21  endmodule
```

Figure 11

A result of simulation proving the right behavior of the top level unit is the content of the memory array (mem_arr). The memory array (mem_arr) should contains for each address equal values 2 (except address 0, which should be 1). An example is on Figure 12.

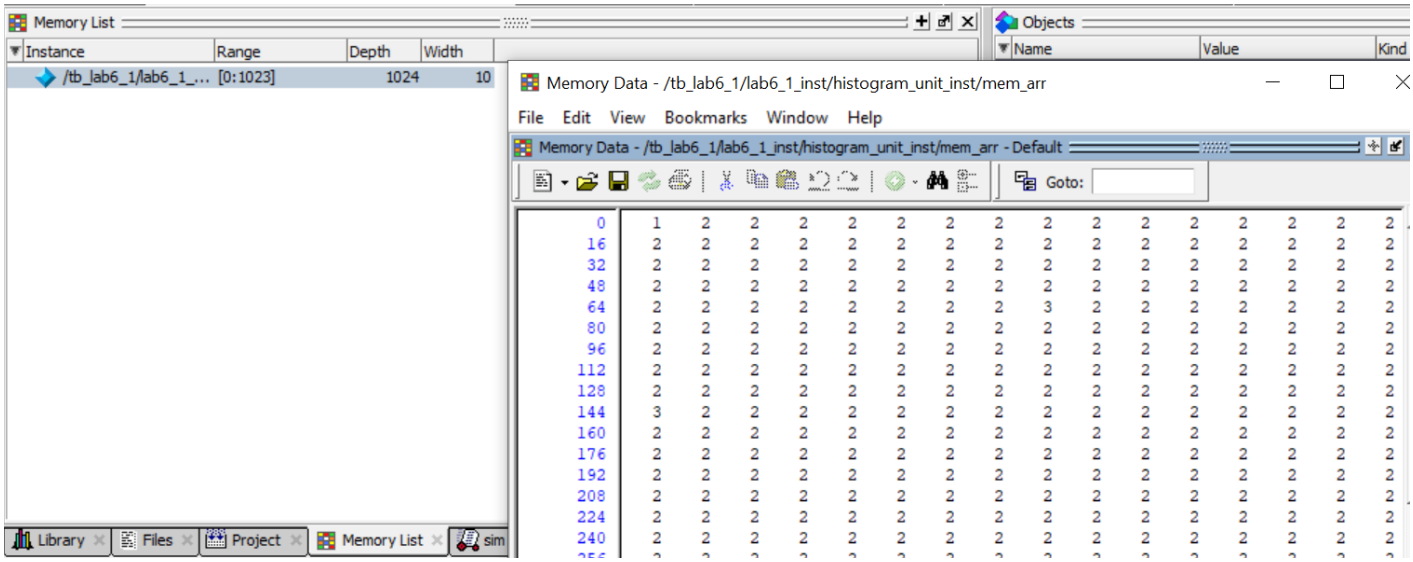

Figure 12

An example of a source code for debugging the unit on MiniDilbCIV board is on Figure 13.

```
1  module db_lab6_1
2  (
3      (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23"  *)
4      input bit CLK
5  );
6
7  bit RST = 1'b1;
8  bit ENA = 1'b1;
9  bit [9:0] mem_out ;
10
11 lab6_1 lab6_1_inst (.*);
12 SP_unit SP_unit_inst (
13     .source     ({RST, ENA} ),
14     .source_clk (CLK        )
15 );
16 endmodule
```

Figure 13

For debugging purposes you need to add **Intel FPGA In-System Source & Probes** – the unit will be used for setting (sourcing) signals **RST** and **ENA** (instead of using the real FPGA pins).

You can find **Intel FPGA In-System Source & Probes** IP function by typing **In-System** in the Find field of IP catalog. If you do not have the IP Catalog already open, go to View menu => Utility Windows => IP Catalog to view the window. Name the file as **SP_unit**. Recommended settings for the unit is on Figure 14.

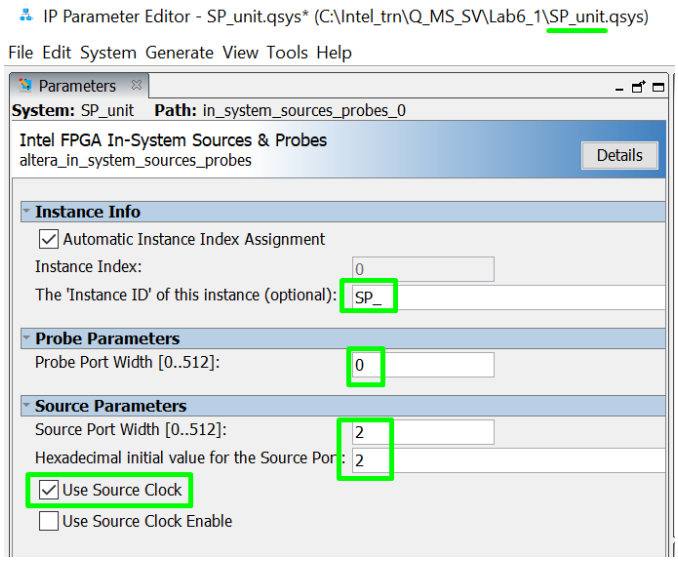

Figure 14

To be sure that you have the right top level project for debugging:

- Set db_lab6_1.sv file as a top level file in Quartus Prime
- Start Analyze and Elaborate procedure.

- Open RTL viewer – you should see something like structure on Figure 15.

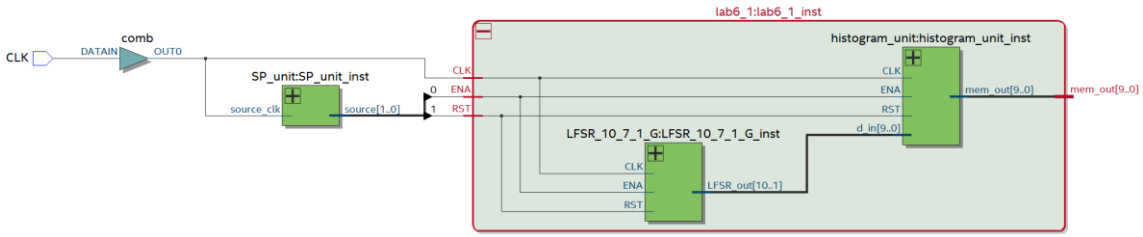

Figure 15

For debugging purposes you need to set-up and add to the project a Signal Tap Logic Analyzer. As an example for setting it up is a Figure 16.

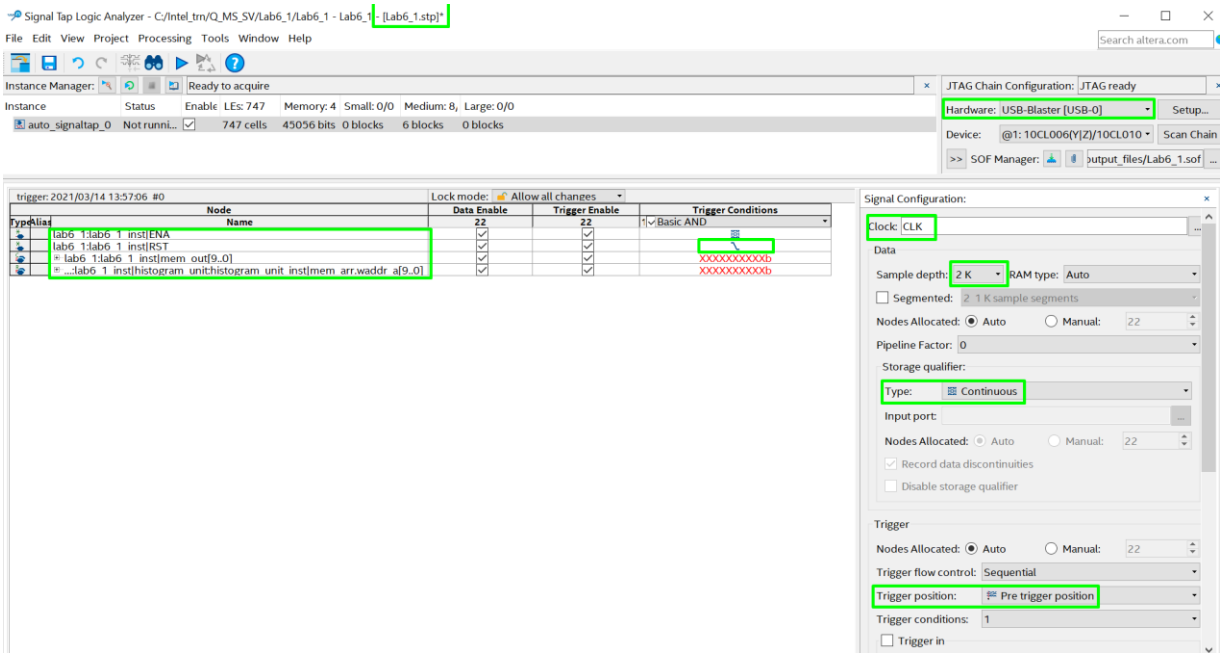

Figure 16

By using **In system Source and Probe** and Signal Tap Logic Analyzer be sure that the project works correctly. An example is on Figure 17. You can see that after setting RST to 0, Signal Tap Logic Analyzer got and displayed results.

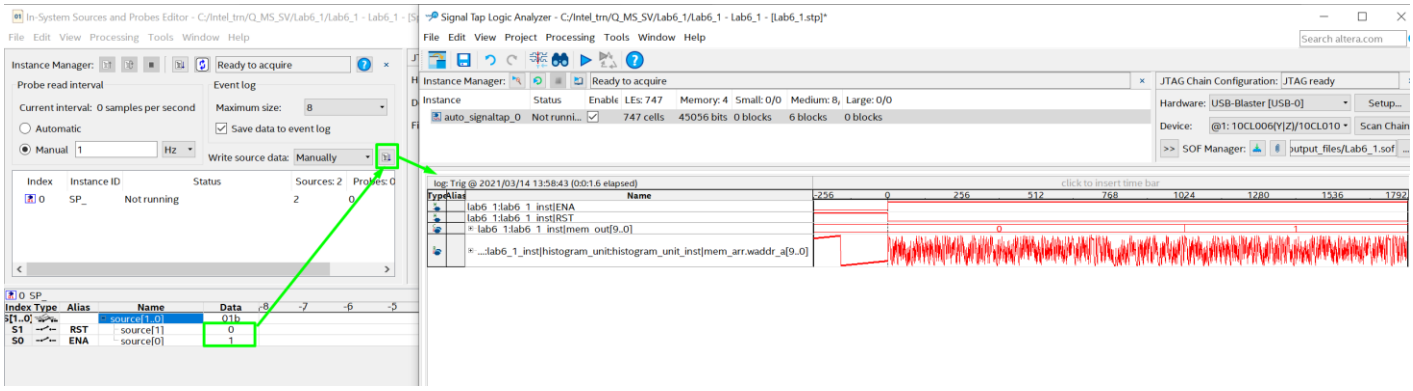

Figure 17

Задание.

Лабораторную работу Lab6_1 студенты выполняют по индивидуальным заданиям. Приведенные выше примеры могут быть использованы как образцы для составления собственных исходных кодов проектируемых модулей и тестов (следует использовать конструкции SystemVerilog)

Программа работы

- Разработать описание LFSR по индивидуальному заданию
- Разработать тест для проверки LFSR
- По результатам моделирования в ModelSim необходимо доказать, что период повторения равен 2^N-1, где N – максимальная степень полинома из задания (ОБРАТИТЕ ВНИМАНИЕ: в некоторых вариантах задания, теоретически, может быть ошибка, тогда этот период будет меньше 2^N-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
- 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 Lab6_1, содержащий LFSR и модуль построения гистограммы.
- Разработать тест для проверки модуля верхнего уровня иерархии.
- По результатам моделирования необходимо доказать, что период повторения LSFR равен 2^N-1, где N – максимальная степень полинома из задания и модуль построения гистограммы создает правильную гистограмму.
- Разработать модуль верхнего уровня для отладки db_Lab6_1, содержащий: модуль Lab6_1; модуль SP_unit (модуль, обеспечивающий возможность задания входных управляющих сигналов без использования кнопок на плате). Модуль д.б. ориентирован на плату MiniDiLabCIV и содержать подключение только к тактовому сигналу на плате.
- Настроить логический анализатор для проведения исследования и отладки реализуемого на плате db_Lab6_1.
- Провести анализ работы db_Lab6_1 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.

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.

Номер задания = номеру студента в списке группы	**Полином для реализации** **(регистр 8 или 7 разрядный, в зависимости от максимальной степени в полиноме)**	**Тип реализации** **Тип логического элемента в обратной связи.**
1.	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1$	Галуа XOR
2.	$x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1$	Фибоначчи XOR
3.	$x^8 + x^7 + x^6 + x^5 + x^4 + x + 1$	Фибоначчи XOR
4.	$x^8 + x^7 + x^6 + x^5 + x^2 + x + 1$	Фибоначчи XNOR
5.	$x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + 1$	Галуа XOR
6.	$x^8 + x^7 + x^6 + x^4 + x^2 + x + 1$	Фибоначчи XOR
7.	$x^8 + x^7 + x^6 + x^3 + x^2 + 1$	Фибоначчи XOR
8.	$x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$	Фибоначчи XNOR
9.	$x^8 + x^4 + x^3 + x + 1$	Галуа XOR
10.	$x^8 + x^4 + x^3 + x^2 + 1$	Фибоначчи XOR
11.	$x^8 + x^5 + x^3 + x + 1$	Фибоначчи XOR
12.	$x^8 + x^5 + x^3 + x^2 + 1$	Фибоначчи XNOR
13.	$x^8 + x^5 + x^4 + x^3 + 1$	Галуа XOR

14.	$x^8 + x^6 + x^3 + x^2 + 1$	Фибоначчи XOR
15.	$x^7 + x^5 + x^2 + x + 1$	Фибоначчи XOR
16.	$x^7 + x^3 + x^3 + x + 1$	Фибоначчи XNOR
17.	$x^7 + x^5 + x^4 + x^3 + 1$	Галуа XOR
18.	$x^7 + x^6 + x^3 + x + 1$	Фибоначчи XOR
19.	$x^7 + x^6 + x^4 + x + 1$	Фибоначчи XOR
20.	$x^7 + x^6 + x^5 + x^2 + 1$	Фибоначчи XNOR
21.	$x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$	Галуа XOR
22.	$x^7 + x^6 + x^5 + x^3 + x^2 + 1$	Фибоначчи XOR
23.	$x^7 + x^4 + 1$	Фибоначчи XOR
24.	$x^7 + x^3 + 1$	Фибоначчи XNOR
25.	$x^7 + x + 1$	Галуа XOR
26.	$x^7 + x^3 + x^2 + x + 1$	Фибоначчи XOR
27.	$x^7 + x^6 + 1$	Фибоначчи XOR
28.	$x^7 + x^6 + x^5 + x^4 + 1$	Фибоначчи XNOR
29.	$x^7 + x^3 + x^3 + x + 1$	Галуа XOR
30.	$x^7 + x^6 + x^5 + x^3 + x^2 + 1$	Фибоначчи XOR