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

# Отчёт курсовой работе. Часть 1

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

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

Руководитель _____ А.А. Федотов
(подпись)

“25” апреля 2024 г.

Санкт-Петербург
2024

**Оглавление**

# 1. Список иллюстраций:

# 2. Структура проекта:

Структура разрабатываемого проекта приведена ниже:

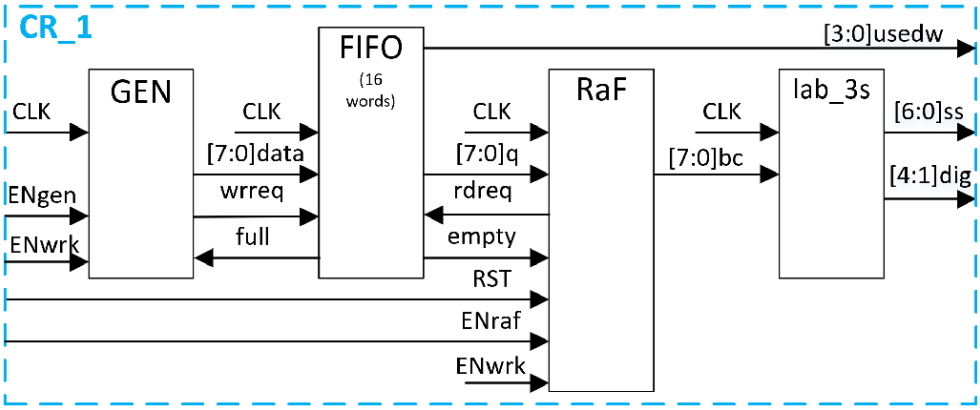


*Рис. 2.1. Структура разрабатываемого проекта.*

Он состоит из следующих элементов:

- GEN – генератор случайных чисел.
- FIFO – IP модуль очереди.
- RaF – модуль для поиска максимального числа, среди поданных на вход.
- lab_3s – модуль для вывода результата модуля RaF на 7-сегментный индикатор.

Для этого модуля разработать тестовый файл, программу отладки на плате и итоговый проект для интеграции устройства.

# 3. Решение:

Начнем разработку данного проекта с модуля GEN. Он должен работать по следующему алгоритму:

- Если ($ENgen = 1$ и $full = 0$ и $Enwrk = 1$) выполняется:
  Модуль формирует сигнал $wrreq = 1$, работает генератор псевдослучайных чисел (данные выдаются на выход $[7:0]$ $data$)
- Если условие ($ENgen = 1$ и $full = 0$ и $Enwrk = 1$) не выполняется:
  Модуль формирует сигнал $wrreq = 0$, генератор псевдослучайных чисел находится в состоянии ожидания (данные выдаются на выход $[7:0]$ $data$)

В качестве генератора псевдослучайных чисел используется модуль LFSR_7_6_3_1_0_F, разработанный ранее в ходе лабораторных:

Verilog_labs - LFSR_7_6_3_1_0_F.sv

```
`timescale 1ns / 1ns
module LFSR_7_6_3_1_0_F (
  input  bit       CLK,
  input  bit       RST,
  input  bit       ENA,
  output bit [7:1] LFSR_out
);
  always_ff @(posedge CLK, posedge RST)
    if (RST) LFSR_out <= 1'b1;
    else if (ENA)
      if (LFSR_out == '0) LFSR_out <= 1'b1;
      else LFSR_out <= {LFSR_out[6:1], LFSR_out[7] ^ LFSR_out[6] ^ LFSR_out[3] ^ LFSR_out[1]};
endmodule
```

В свою очередь модуль GEN является лишь контролирует, когда модуль будет работать, а когда нет. Он будет выглядеть следующим образом:

Verilog_labs - gen.sv

```
module gen (
  input  bit       clk,
  input  bit       ENgen,
  input  bit       ENwrk,
  input  bit       full,
  output bit       wrreq,
  output bit [7:0] data
);

  bit       RST = 1'b0;
  bit       ENA = 1'b0;
  bit [7:1] LFSR_out = 1'b0;

  LFSR_7_6_3_1_0_F u_LFSR_7_6_3_1_0_F (
    .CLK(clk),
    .RST,
    .ENA,
    .LFSR_out
  );

  assign data = {1'b0, LFSR_out};

  always_ff @(posedge clk) begin
    if (ENgen == 1'b1 && full == 0 && ENwrk == 1) begin
      wrreq <= 1'b1;
      ENA   <= 1'b1;
    end else begin
      wrreq <= 1'b0;
      ENA   <= 1'b0;
    end
  end

endmodule
```

На вход он принимает сигналы, которые отражены на *Рис. 2.1*. Далее объявляется модуль LFSR_7_6_3_1_0_F. Сигнал RST у него будет принимать постоянное значение нуля т.е. никогда не будет сброшен, благодаря ENA будет выполнятся остановка генерации, а на выход data будет подаваться результат.

Логика включения ENA и сигнала wrreq написана в блоке always_ff и соответствует поставленному заданию.

Выполним компиляцию и посмотрим на получившуюся RTL схему:

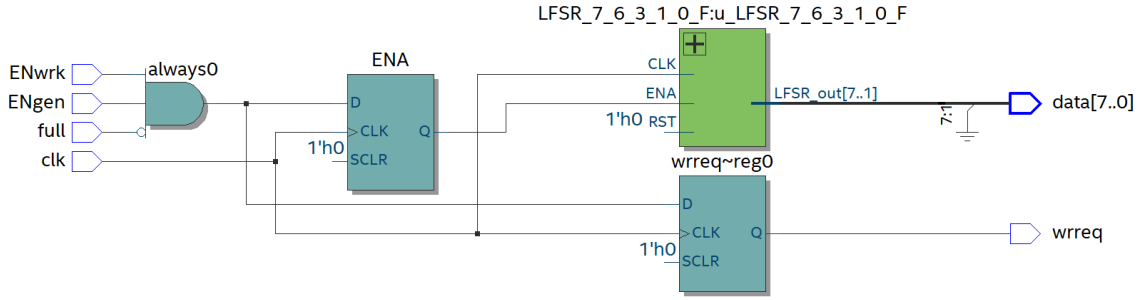


*Рис. 3.1. RTL Viewer модуля GEN.*

Она соответствует ожиданиям, теперь необходимо провести тестирование данного модуля, для этого разработаем тест первого класса, он будет выглядеть следующим образом:

Verilog_labs - gen_tb.sv

```
1  `timescale 1ns / 1ns
2  module tb_gen;
3
4    parameter PERIOD = 10;
5
6    bit       clk = 1'b0;
7    bit       ENgen = 1'b0;
8    bit       ENwrk = 1'b0;
9    bit       full = 1'b0;
10   bit       wrreq;
11   bit [7:0] data;
12
13   initial begin
14     forever #(PERIOD / 2) clk = ~clk;
15   end
16
17   gen u_gen (.*);
18
19   initial begin
20     #(PERIOD * 1);
21     ENgen = 1'b1;
22     #(PERIOD * 1);
23     ENwrk = 1'b1;
24     #(PERIOD * 10);
25     full = 1'b1;
26     #(PERIOD * 2);
27     full = 1'b0;
28     #(PERIOD * 4);
29     $stop;
30   end
31
32 endmodule
```

В этом тесте мы проверяем, что счет происходит только при заданных требованиях, а также продолжается, когда требования вновь выполняются.
Запустим его и посмотрим на результат:

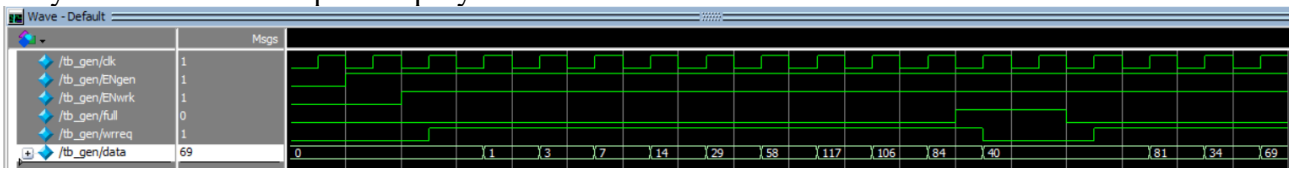


*Рис. 3.2. Тест модуля GEN.*

Как можно заметить, тест соответствует разработанным требованиям.
Следующим разработаем модуль RaF, который будет считывать данные из FIFO и сохранять максимальное число. Алгоритм выглядит следующим образом:

- Если ($ENraf = 1$ и $empty = 0$ и $Enwrk = 1$)
  Модуль считывает данные из FIFO (формирует сигнал $rdreq = 1$)
  Ищет среди принимаемых от FIFO данных максимальное значение
  Передает текущее максимальное значение на выход $[7:0]bc$
- Если условие ($ENraf = 1$ и $empty = 0$ и $Enwrk = 1$) не выполняется
  Модуль формирует сигнал $rdreq = 0$
  Модуль находится в состоянии ожидания (текущее максимальное значение выдается на выход $[7:0]$ $bc$)
- Сигнал RST асинхронно сбрасывает найденное максимальное значение в 0

Его описание на языке System Verilog будет выглядеть следующим образом:

Verilog_labs - RaF.sv

```
module RaF (
  input  bit       CLK,
  input  bit       ENraf,
  input  bit       empty,
  input  bit       Enwrk,
  input  bit       RST,
  input  bit [7:0] q,
  output bit       rdreq,
  output bit [7:0] bc
);

  always_ff @(posedge CLK, posedge RST) begin
    if (RST) bc <= 1'b0;
    else if (ENraf == 1'b1 && empty == 1'b0 && Enwrk == 1'b1) begin
      rdreq <= 1'b1;
      bc    <= (bc < q ? q : bc);
    end else begin
      rdreq <= 1'b0;
    end
  end

endmodule
```

Посмотрим на RTL Viewer разработанного модуля:

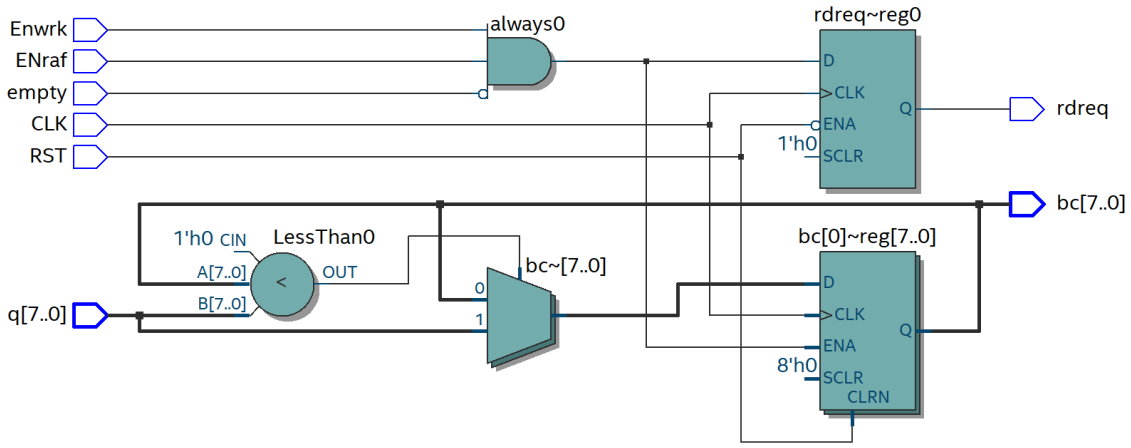


*Рис. 3.3. RTL Viewer модуля RaF.*

Далее необходимо провести тестирование исследуемого модуля. Для этого разработаем тест первого уровня:

Verilog_labs - RaF_tb.sv

```
`timescale 1ns / 1ns
module tb_RaF;

  parameter PERIOD = 10;

  bit       CLK = 1'b0;
  bit       ENraf = 1'b0;
  bit       empty = 1'b0;
  bit       Enwrk = 1'b0;
  bit       RST = 1'b0;
  bit [7:0] q;
  bit       rdreq;
  bit [7:0] bc;

  initial begin
    forever #(PERIOD / 2) CLK = ~CLK;
  end

  RaF u_RaF (.*);

  initial begin
    #(PERIOD * 1);
    q = 8'd7;
    #(PERIOD * 1);
    ENraf = 1'b1;
    #(PERIOD * 1);
    Enwrk = 1'b1;
    #(PERIOD * 2);
    q = 8'd4;
    #(PERIOD * 2);
    RST = 1'b1;
    #(PERIOD * 1);
    RST = 1'b0;
    #(PERIOD * 1);
    q = 8'd120;
    #(PERIOD * 2);
    $stop;
  end

endmodule
```

В этом тесте мы подаем на вход q какое-то значение, но т.к. условия не выполняются записано оно не будет, после чего подаем все значения для выполнения условия и теперь запись должна быть выполнена. После чего пытаемся перезаписать меньшим значением.
Далее происходит сброс и пытаемся записать после сброса другое значение.
Запустим данный тест и посмотрим на результат тестирования:

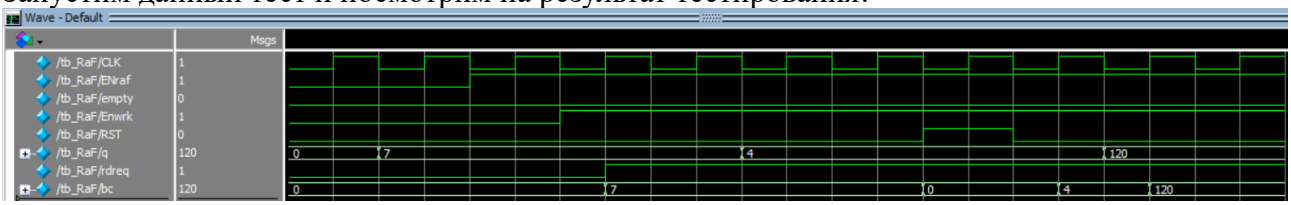


*Рис. 3.4. Результат тестирования модуля RaF.*

Как мы видим все результаты тестирования совпадают с ожиданиями.
Модуль FIFO мы возьмем из IP библиотеки, разрабатывать его не нужно, а модуль для вывода значений на 7-сегментный дисплей мы возьмем из лабораторной работы прошлого семестра, это значит мы готовы перейти к разработке модуля на *Рис. 2.1*.
Данное описание на языке System Verilog приведено ниже:

Verilog_labs - CR_1.sv

```
module CR_1 (
  input  bit       CLK,
  input  bit       ENgen,
  input  bit       Enwrk,
  input  bit       RST,
  input  bit       ENraf,
  output bit [3:0] usedw,
  output bit [6:0] ss,
  output bit [4:1] dig
);

  bit       full;
  bit       wrreq;
  bit [7:0] data;

  gen u_gen (
    .clk  (CLK),
    .ENgen,
    .ENwrk(Enwrk),
    .full,
    .wrreq,
    .data
  );

  bit       rdreq;
  bit       empty;
  bit [7:0] q;

  FIFO u_fifo (
    .clock(CLK),
    .data(data),
    .rdreq(rdreq),
    .wrreq(wrreq),
    .empty(empty),
    .full(full),
    .q(q),
    .usedw(usedw)
  );

  bit [7:0] bc;

  RaF u_RaF (
    .CLK,
    .ENraf,
    .empty,
    .Enwrk,
    .RST,
    .q,
    .rdreq,
    .bc
  );

  lab_3s u_lab_3s (
    .clk(CLK),
    .bc(bc),
    .ss(ss),
    .dig(dig)
  );


endmodule
```

Как можно заметить это просто описание соединение разработанных модулей а также очереди и модуля, лабораторной номер 3.

Очередь будет иметь следующие настройки:

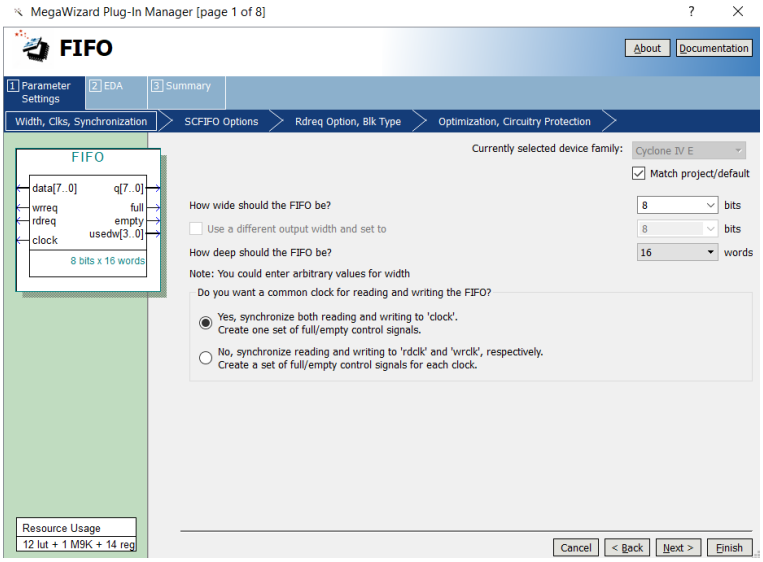


*Рис. 3.5. Настройки модуля FIFO.*

Выполним компиляцию и посмотрим на RTL Viewer, чтоб убедиться в корректности построенной схемы:

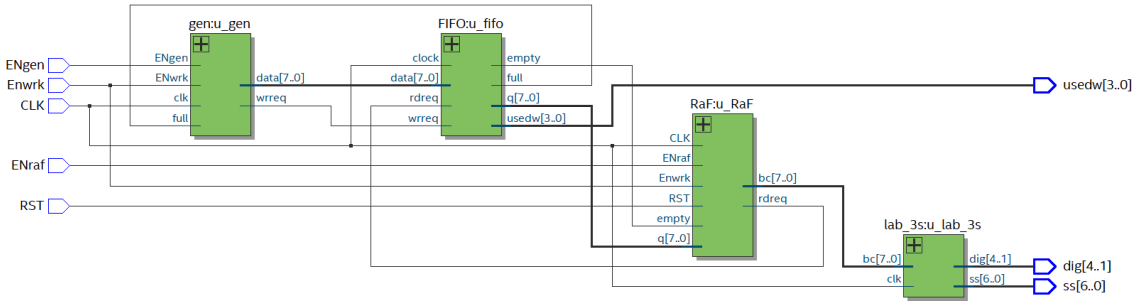


*Рис. 3.6. RTL Viewer модуля CR_1.*

Как мы видим, полученная схема похожа на *Рис. 2.1*. что свидетельствует о корректности разработанного модуля. Теперь выполним тестирование этого модуля, для этого разработает следующий тест первого класса:

Verilog_labs - CR_1_tb.sv

```
`timescale 1ns / 1ns
module tb_CR_1;

  parameter PERIOD = 10;

  bit       CLK = 1'b0;
  bit       ENgen = 1'b0;
  bit       Enwrk = 1'b0;
  bit       RST = 1'b0;
  bit       ENraf = 1'b0;
  bit [3:0] usedw;
  bit [6:0] ss;
  bit [4:1] dig;

  initial begin
    forever #(PERIOD / 2) CLK = ~CLK;
  end

  CR_1 u_CR_1 (.*);

  initial begin
    #(PERIOD * 1);
    ENraf = 1'b0;
    ENgen = 1'b1;
    Enwrk = 1'b1;
    RST = 1'b0;
    #(PERIOD * 18);
    ENraf = 1'b1;
    ENgen = 1'b0;
    Enwrk = 1'b1;
    RST = 1'b0;
    #(PERIOD * 18);
    ENraf = 1'b1;
    ENgen = 1'b1;
    Enwrk = 1'b1;
    RST = 1'b0;
    #(PERIOD * 27);
    ENraf = 1'b1;
    ENgen = 1'b1;
    Enwrk = 1'b0;
    RST = 1'b0;
    #(PERIOD * 3);
    ENraf = 1'b1;
    ENgen = 1'b1;
    Enwrk = 1'b1;
    RST = 1'b1;
    #(PERIOD * 3);
    $stop;
  end

endmodule
```

Данный тест проверяет работу модуля по следующему алгоритму:

- Запись в FIFO до появления сигнала $full$ ($ENraf = 0\ Engen = 1\ Enwrk = 1\ RST = 0$)
- Чтение из FIFO до появления сигнала $empty$ ($ENraf = 1\ Engen = 0\ Enwrk = 1\ RST = 0$)
- Чтение и запись FIFO до появления на выводе $[7:0]bc$ максимального значения ($ENraf = 1\ Engen = 1\ Enwrk = 1\ RST = 0$)
- Запрет работы ($ENraf = 1\ Engen = 1\ Enwrk = 0\ RST = 0$)
- Сброс максимального значения ($ENraf = \text{x}\ Engen = \text{x}\ Enwrk = \text{x}\ RST = 1$)

Проведем первый тест, его результат выглядит следующим образом:

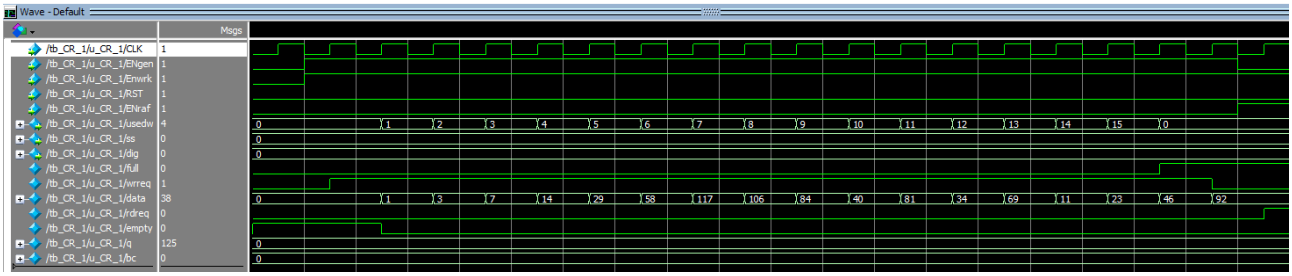


*Рис. 3.7. Тестирование модуля CR_1 до сигнала full.*

Как мы видим, все работает корректно и данные выдаются до появления сигнала full. В bc данные не записываются т.к. он не активирован.

Теперь перейдем к следующему тесту:

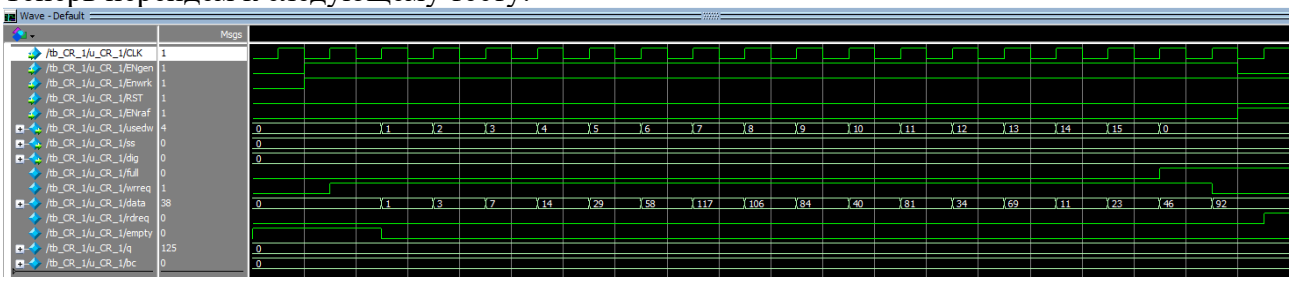


*Рис. 3.8. Тестирования модуля CR_1 до сигнала empty.*

Система корректно показывает и на этом тесте, выдавая содержимое очереди до сигнала empty. В bc данные аналогично не попадают т.к. он не активирован.

Теперь запустим систему в стандартном для неё режиме работы:

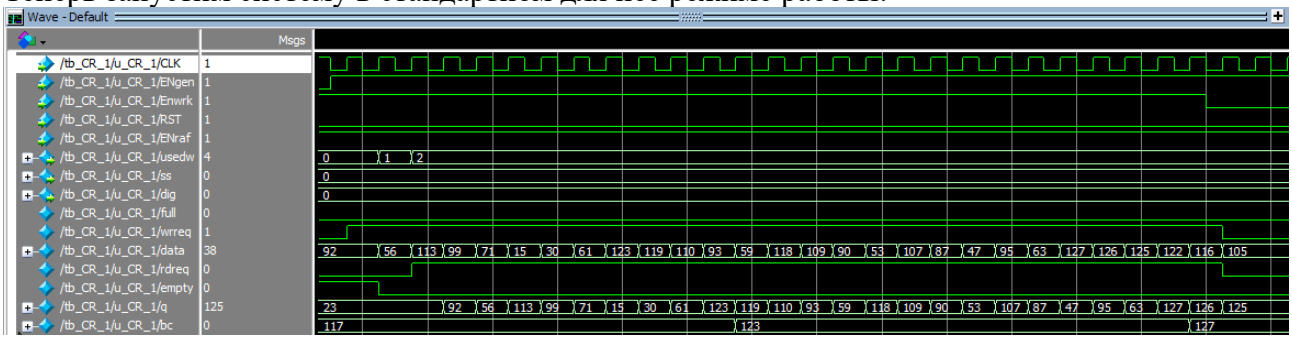


*Рис. 3.9. Тестирование модуля CR_1 до максимального значения генератора.*

Как мы видим, система работает корректно и действительно сохраняет значение максимума.

Теперь проверим сигнал RST:

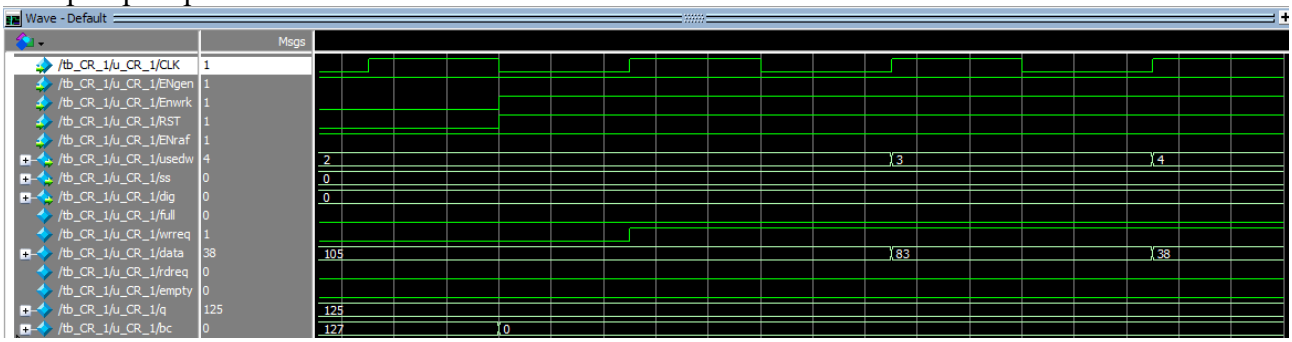


*Рис. 3.10. Тестирования модуля CR_1 на сигнале RST.*

На рисунке выше видно, что устройство корректно работает и с этими сигналами.

Стоит отметить, что во время тестирования выходы ss и dig оставались нулевыми. Это связано с тем, что модуле вывода данных на 7-сегментный индикатор стоит делитель, чтоб избавиться от мигания индикатора от частных обновлений. Его значение составляет около 10000, что много меньше количество тактов в тесте.

Изменим это, передав в него значение 1, чтоб при проверке на плате уже видеть нормальные сигналы.

Перейдем непосредственно к тестированию на плате, для этого создадим следующий модуль, благодаря которому сможем менять входные значение, используя ISSPE и смотреть на результат, используя Signal Tap II:

Verilog_labs - db_CR_1.sv

```
module db_CR_1 (
  (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23" *)
  input bit CLK
);
  bit       ENgen;
  bit       Enwrk;
  bit       RST;
  bit       ENraf;
  bit [3:0] usedw;
  bit [6:0] ss;
  bit [4:1] dig;

  CR_1 u_CR_1 (
    .CLK,
    .ENgen,
    .Enwrk,
    .RST,
    .ENraf,
    .usedw,
    .ss,
    .dig
  );

  SP_unit u0 (
    .source     ({ENgen, Enwrk, RST, ENraf}),  // sources.source
    .source_clk(CLK)                           // source_clk.clk
  );

endmodule

```

Его RTL Viewer приведен ниже:

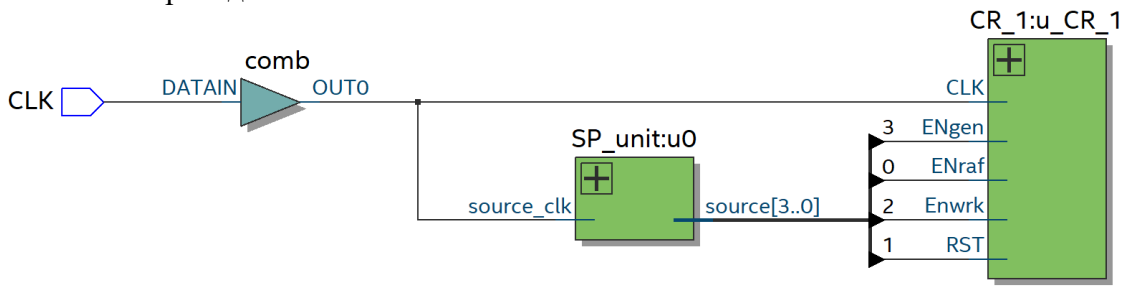


*Рис. 3.11. RTL Viewer для tb_CR_1.*

Выполним настройку Signal Tap II следующим образом:

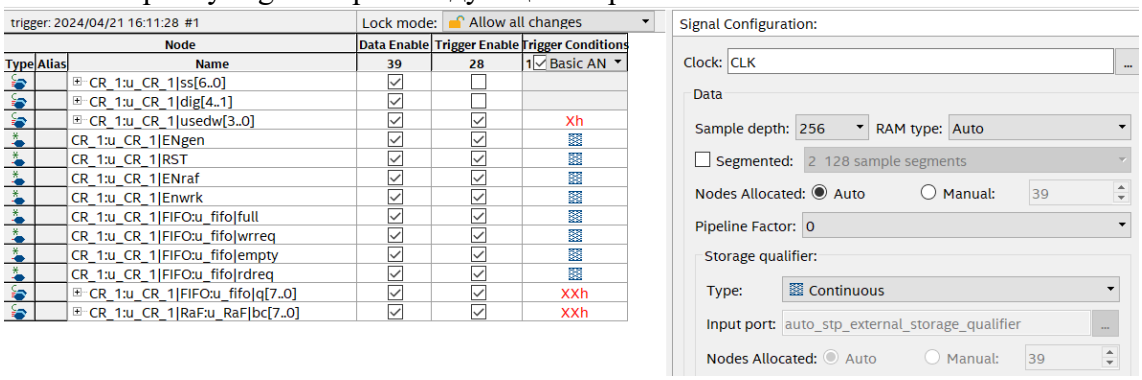


*Рис. 3.12. Настройки Signal Tap II.*

Для выхода ss создадим маску, чтоб смотреть значение, которое он выводит:

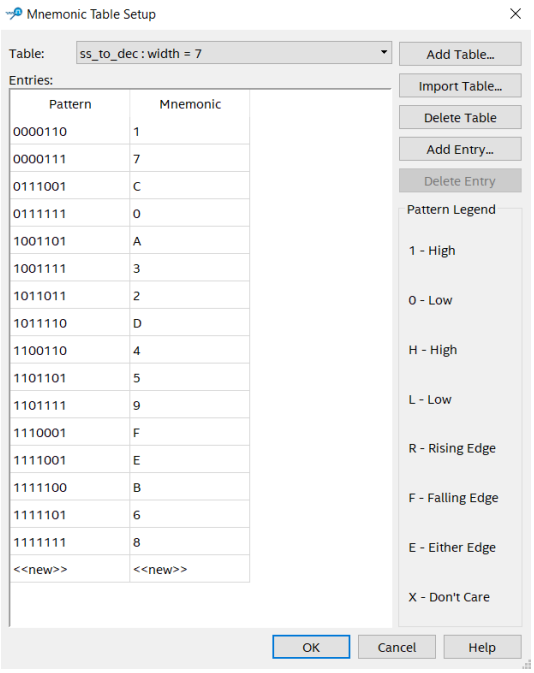


*Рис. 3.13. Mnemonic Tabel Setup.*

Выполним запуск и используя алгоритм для тестирования, приведенный ранее и рассмотрим интересующие нас случаи:

Запись в FIFO до появления сигнала $full$ ($ENraf = 0\ Engen = 1\ Enwrk = 1\ RST = 0$):

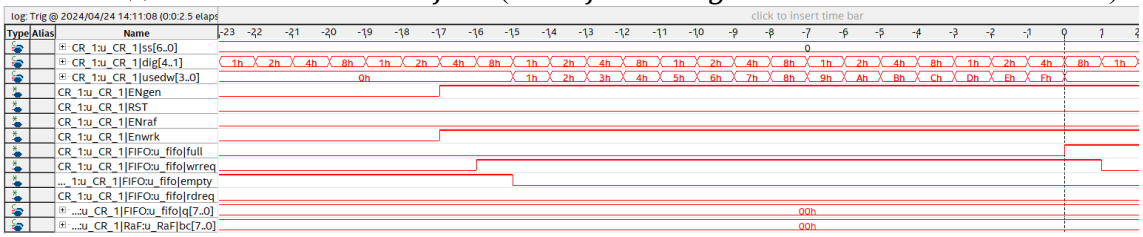

*Рис. 3.14. Появление сигнала full при тестировании на плате.*

Чтение из FIFO до появления сигнала $empty$ ($ENraf = 1\ Engen = 0\ Enwrk = 1\ RST = 0$):

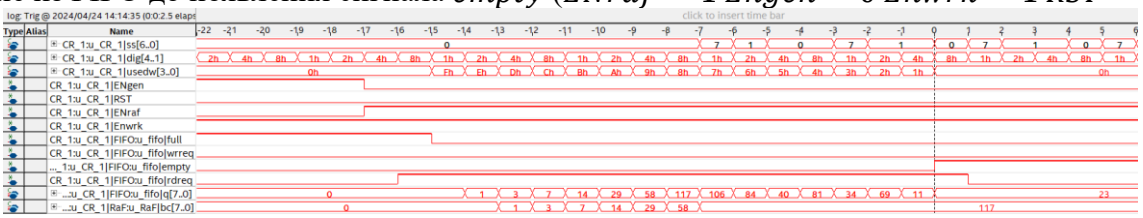

*Рис. 3.15. Появление сигнала empty при тестировании на плате.*

Чтение и запись FIFO до появления на выводе $[7:0]bc$ максимального значения ($ENraf = 1\ Engen = 1\ Enwrk = 1\ RST = 0$):

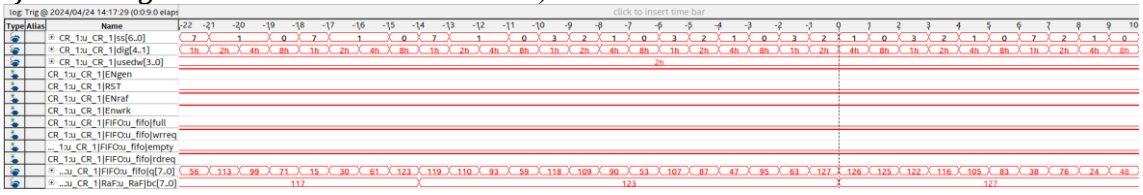

*Рис. 3.16. Появление значения 127 (максимального) при тестировании на плате.*

Запрет работы ($ENraf = 1\ Engen = 1\ Enwrk = 0\ RST = 0$):

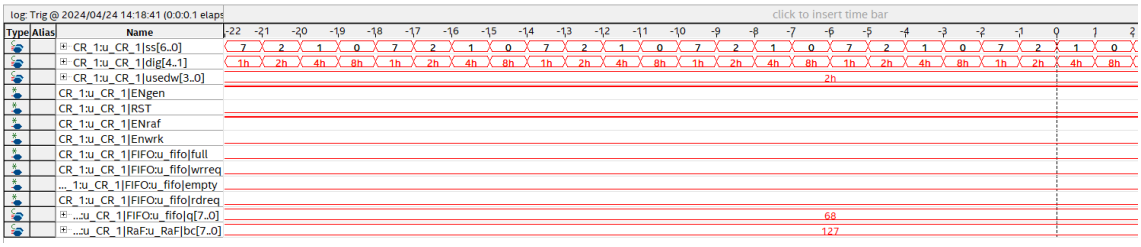


*Рис. 3.17. Отключение работы при тестировании на плате.*

Сброс максимального значения ($ENraf = x\,Engen = x\,Enwrk = x\,RST = 1$):

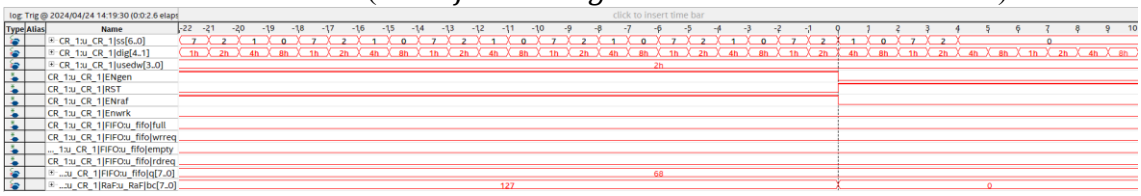


*Рис. 3.18. Сброс при тестировании на плате.*

Как мы видим, все работает корректно. Раз устройство работает верно, перейдем к созданию модуля имплементации:

```
Verilog_labs - impl_CR_1.sv
1  module impl_CR_1 (
2    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23" *)
3    input bit CLK,
4    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "46 25 24" *)
5    input bit [2:0] SW,
6    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "84, 76, 85, 77, 86, 133, 87" *)
7    output bit [6:0] ss,
8    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "73, 80, 74, 83" *)
9    output bit [4:1] dig,
10   (* altera_attribute = "-name IO_STANDARD \"2.5-V\"", chip_pin = "69 70 71 72" *)
11   output bit [3:0] LED
12 );
13   bit Enwrk = 1'b0;
14
15   CR_1 u_CR_1 (
16     .CLK,
17     .ENgen(SW[1]),
18     .Enwrk,
19     .RST  (RST_snh[1]),
20     .ENraf(SW[2]),
21     .usedw(LED),
22     .ss,
23     .dig
24   );
25
26   bit [1:0] RST_snh = 1'b0;
27
28   always_ff @(posedge CLK) begin
29     RST_snh <= {RST_snh[0], SW[0]};
30   end
31
32   int counter = 0;
33   always_ff @(posedge CLK) begin
34     if (counter > 8000000) begin
35       Enwrk   <= 1'b1;
36       counter <= 1'b0;
37     end else begin
38       Enwrk   <= 1'b0;
39       counter <= counter + 1;
40     end
41   end
42
43 endmodule
```

Также не забудем выполнить изменения в модуле вывода значений на 7-сегментный индикатор, поставив делитель обратно на 10000. Выполним компиляцию и посмотрим на получившуюся RTL схему:

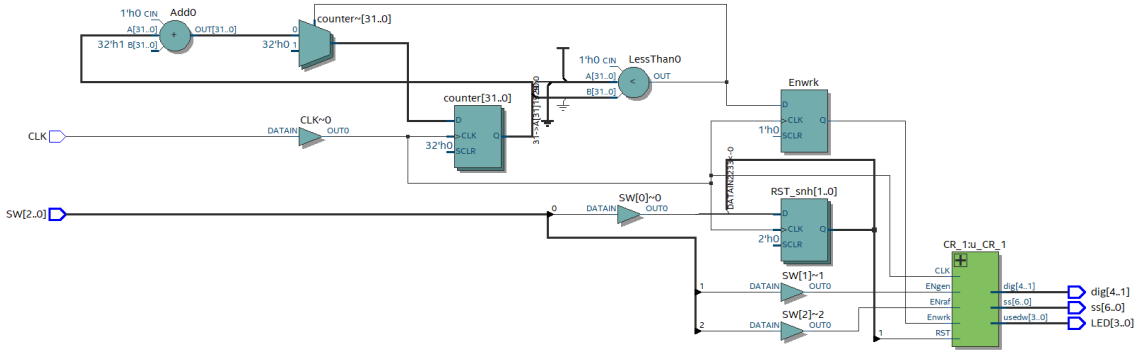


*Рис. 3.19. RTL Viewer для impl_CR_1.*

Запишем получившуюся программу на плату. Результат работы продемонстрирован преподавателю.

# 4. Вывод:

По ходу выполнения курсовой работы была успешно осуществлена разработка заданного устройства, используя System Verilog в качестве основного инструмента. Кроме того,

проведено комплексное тестирование устройства как на симуляторе, так и на плате. Этот процесс включал в себя все этапы разработки - от первоначального проектирования до завершения работы над функционально полноценным устройством, готовым к использованию. Как итог, была достигнута стабильная работоспособность созданного устройства, что подтверждает успешное завершение проекта.