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

# Отчёт по лабораторной работе № 3

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

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

Руководитель _____ А.А. Федотов
(подпись)

“21” февраля 2024 г.

Санкт-Петербург
2024

**Оглавление**

# 1. Список иллюстраций:

# 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:

- Два экземпляра (cntA и cntB) 10-разрядного счетчика (cnt_10bits) на сложение, по модулю, заданному на входах cntA_Module и cntB_Module, формируют:
  - текущие значения: cntA[9:0], cntB[9:0]
  - сигналы переноса (по достижению заданного модуля счета): CoutA, CoutB
- Компаратор (cpm_eq) на равенство сравнивает значения двух счетчиков и формирует выходной сигнал cntA_EQ_cntB.
- Конечный автомат (fsm) анализирует сигналы переноса двух счетчиков и формирует сигналы:
  - AeqB – равное количество сигналов переноса от счетчика A и счетчика B.
  - AmB – сигналы переноса от счетчика A появляются чаще.
  - BmA – сигналы переноса от счетчика B появляются чаще.
- Сигнал асинхронного сброса (rst_n) обеспечивает сброс всех устройств с памятью.

Структура проекта приведена на рисунке ниже:

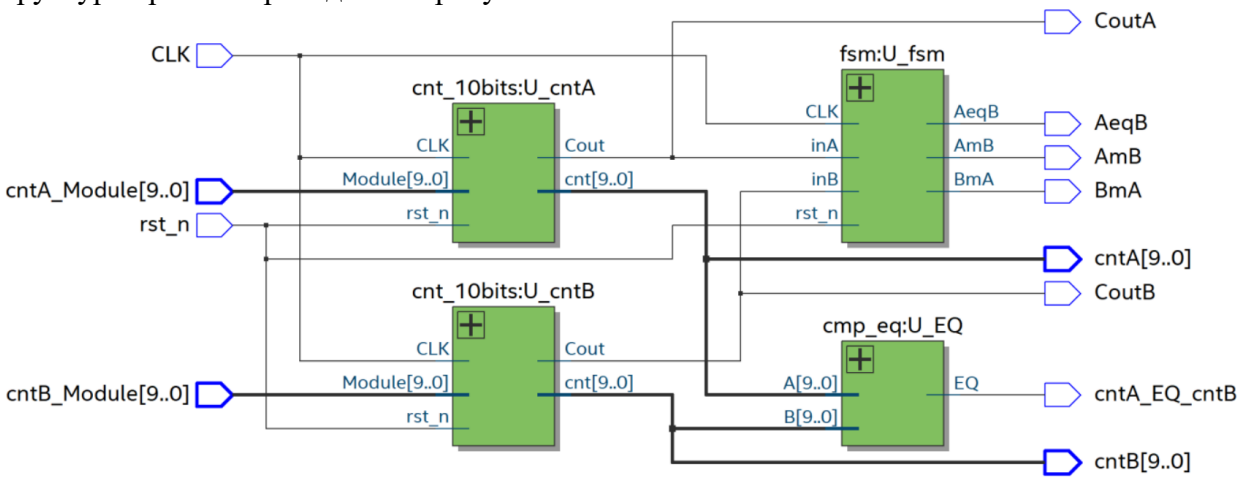


*Рис. 3.1. Структура разрабатываемого проекта.*

# 4. Решение:

Выполним создание первого модуля – cnt_10bits, используя средства System Verilog:

```
module cnt_10bits (
  input  bit       CLK,
  input  bit [9:0] Module,
  input  bit       rst_n,
  output bit       Cout,
  output bit [9:0] cnt
);

  always_ff @(posedge CLK, negedge rst_n)
    if (!rst_n) cnt <= 10'd0;
    else cnt <= cnt < (Module - 10'd1) ? cnt + 10'd1 : 10'd0;

  assign Cout = cnt == (Module - 10'd1);

endmodule
```

Это счетчик делитель с делителем, подаваемым на вход Module.

Далее реализуем модуль cmp_eq, как комбинационную схему средствами System Verilog:

```
module cmp_eq (
    input bit [9:0] A,
    input bit [9:0] B,
    output bit EQ
);

always_comb EQ = A == B;

endmodule
```

И последний модуль, необходимый для реализации – конечный автомат для фиксации, какие сигналы переноса появляются чаще – со счетчика A или B:

```
module fsm (
  input  bit CLK,
  input  bit rst_n,
  input  bit inA,
  input  bit inB,
  output bit AeqB,
  output bit AmB,
  output bit BmA
);

  enum bit [1:0] {EQ, A_, B_} states;

  always_ff @(posedge CLK, negedge rst_n) begin
    if (~rst_n) states <= EQ;
    else
      case (states)
        EQ: if (inA & ~inB) states <= A_;
            else if (~inA & inB) states <= B_;
        A_: if (~inA & inB) states <= EQ;
        B_: if (inA & ~inB) states <= EQ;
      endcase
  end

  always_comb begin
    AeqB = 1'd0;
    AmB  = 1'd0;
    BmA  = 1'd0;
    case (states)
      EQ: AeqB = 1'd1;
      A_: AmB = 1'd1;
      B_: BmA = 1'd1;
    endcase
  end

endmodule
```

Теперь создадим модуль верхнего уровня, собрав все модули вместе:

```
module lab_MS_SV1 (
  input  bit       CLK,
  input  bit [9:0] cntA_Module,
  input  bit [9:0] cntB_Module,
  input  bit       rst_n,
  output bit       CoutA,
  output bit       CoutB,
  output bit [9:0] cntA,
  output bit [9:0] cntB,
  output bit       cntA_EQ_cntB,
  output bit       AeqB,
  output bit       AmB,
  output bit       BmA
);

  cnt_10bits U_cntA (
    .CLK   (CLK),
    .Module(cntA_Module),
    .rst_n (rst_n),
    .Cout  (CoutA),
    .cnt   (cntA)
  );
  cnt_10bits U_cntB (
    .CLK   (CLK),
    .Module(cntB_Module),
    .rst_n (rst_n),
    .Cout  (CoutB),
    .cnt   (cntB)
  );
  cmp_eq U_EQ (
    .A (cntA),
    .B (cntB),
    .EQ(cntA_EQ_cntB)
  );
  fsm U_fsm (
    .CLK  (CLK),
    .rst_n(rst_n),
    .inA  (CoutA),
    .inB  (CoutB),
    .AeqB (AeqB),
    .AmB  (AmB),
    .BmA  (BmA)
  );

endmodule
```

Выполним полную компиляцию и проверим, что получившаяся схема соответствует поставленному заданию:

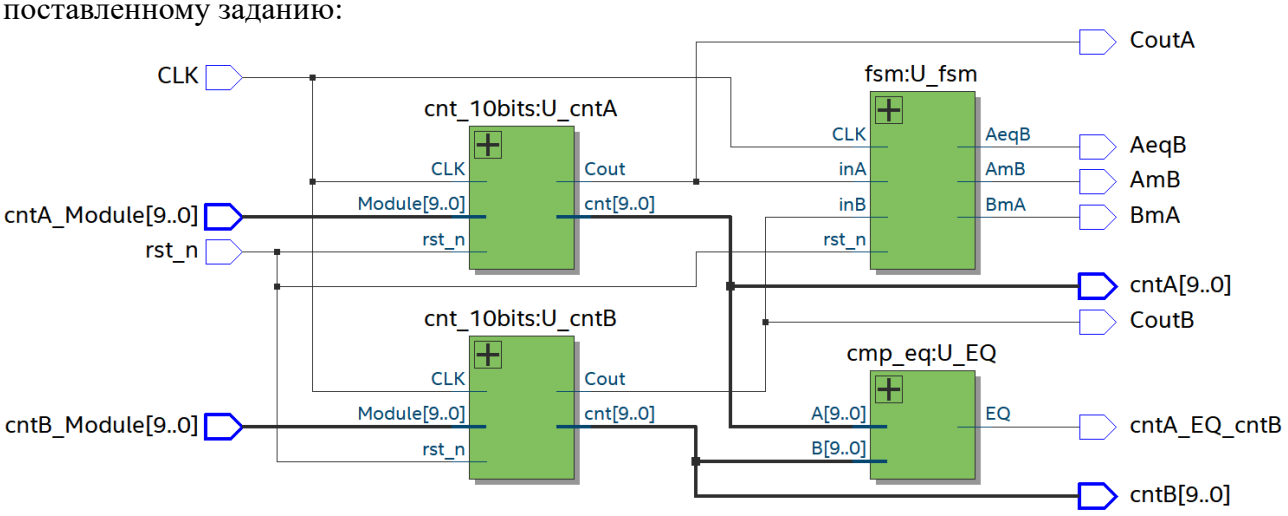


*Рис. 4.1. RTL Viewer.*

Как мы видим, получившаяся схема полностью идентична поставленной задаче, далее необходимо выполнить тестирование.

Разработаем тест первого уровня:

```
`timescale 1ns / 1ns
module tb_lab_MS_SV1 ();
  bit CLK = 1'b0;
  bit rst_n;
  bit CAout, CBout;
  bit AeqB, AmB, BmA;
  bit CoutA, CoutB;
  bit cntA_EQ_cntB;
  bit [9:0] cntA_Module, cntB_Module;
  bit [9:0] cntA, cntB;

  lab_MS_SV1 u_lab_MS_SV1 (
    .CLK         (CLK),
    .cntA_Module (cntA_Module),
    .cntB_Module (cntB_Module),
    .rst_n       (rst_n),
    .CoutA       (CoutA),
    .CoutB       (CoutB),
    .cntA        (cntA),
    .cntB        (cntB),
    .cntA_EQ_cntB(cntA_EQ_cntB),
    .AeqB        (AeqB),
    .AmB         (AmB),
    .BmA         (BmA)
  );

  localparam PERIOD = 20;

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


  initial begin
    rst_n = 1'd0;
    #(PERIOD * 4);
    rst_n = 1'd1;
    cntA_Module = 10'd10;
    cntB_Module = 10'd10;
    #(PERIOD * 28);

    rst_n = 1'd0;
    #(PERIOD * 4);
    rst_n = 1'd1;
    cntA_Module = 10'd5;
    cntB_Module = 10'd10;
    #(PERIOD * 28);

    rst_n = 1'd0;
    #(PERIOD * 4);
    rst_n = 1'd1;
    cntA_Module = 10'd10;
    cntB_Module = 10'd5;
    #(PERIOD * 28);

    rst_n = 1'd0;
    #(PERIOD * 4);
    rst_n = 1'd1;
    cntA_Module = 10'd5;
    cntB_Module = 10'd5;
    #(PERIOD * 28);
    $stop;
  end


endmodule
```

Выполним запуск этого теста, средствами Quartus и получим следующий результат:

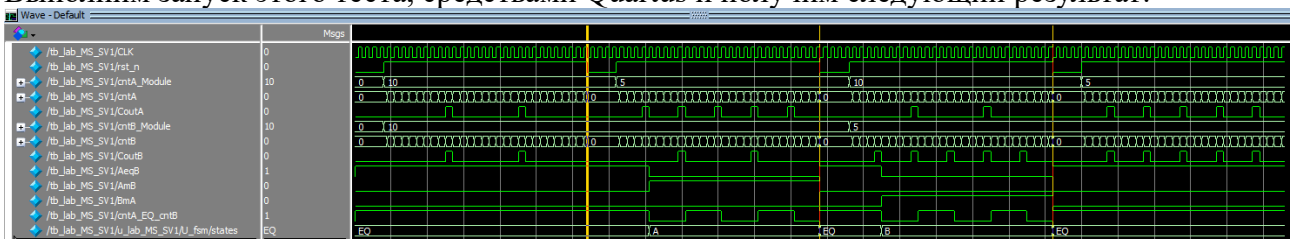


*Рис. 4.2. Результат моделирования.*

Видим, что результат моделирования полностью соответствует ожиданиям, тогда перейдем к моделированию на плате, для этого создадим следующий модуль верхнего уровня:

```
module db_lab_MS_SV1 (
  (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23" *)
  input CLK
);

  bit rst_n;
  bit AeqB, AmB, BmA;
  bit CoutA, CoutB;
  bit cntA_EQ_cntB;
  bit [9:0] cntA_Module, cntB_Module;
  bit [9:0] cntA, cntB;

  lab_MS_SV1 U1 (
    .CLK         (CLK),
    .cntA_Module (cntA_Module),
    .cntB_Module (cntB_Module),
    .rst_n       (rst_n),
    .CoutA       (CoutA),
    .CoutB       (CoutB),
    .cntA        (cntA),
    .cntB        (cntB),
    .cntA_EQ_cntB(cntA_EQ_cntB),
    .AeqB        (AeqB),
    .AmB         (AmB),
    .BmA         (BmA)
  );

  SP_unit U2 (
    .source    ({rst_n, cntA_Module, cntB_Module}),
    .source_clk(CLK)
  );

endmodule
```

Добавим SDC файл, чтоб задать требования тактовому сигналу, со следующим содержанием:

```
create_clock -name {CLK} -period 20.000 -waveform { 0.000 10.000 } [get_ports {CLK}]
derive_clock_uncertainty
```

Для отладки на устройстве будем использовать ISSP, с помощью которого будем подавать сигнал сброса и значения для счетчиков и Signal Tap II, которым будем отслеживать результаты. ISSP будет иметь следующие настройки:

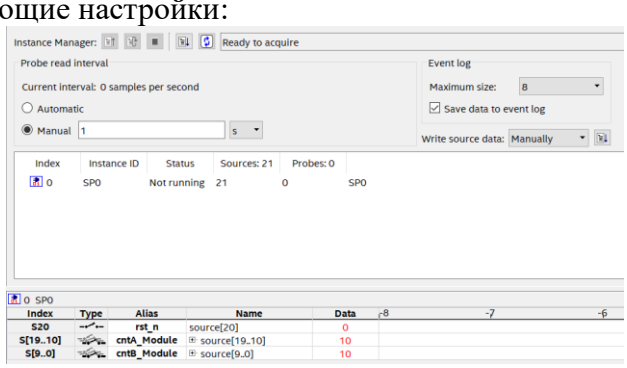


*Рис. 4.3. Настройки ISSP.*

А настройки Signal Tap II выглядят следующим образом:

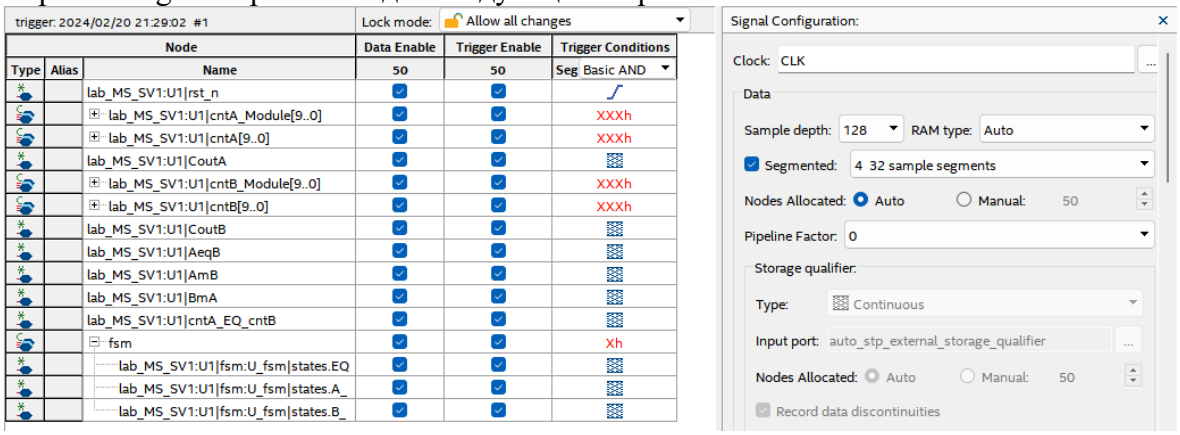


*Рис. 4.4. Настройки Signal Tap II.*

Как можно заметить, захват происходит по фронту сигнала сброса. Используя ISSP, запишем на устройство в cntA_Module сначала 10, потом 5, потом 10, потом 5, а в cntB_Module 10, 10, 5, 5. Между переключениями необходимо сбрасывать значения.
В результате ожидаем EQ, A, B, EQ, что мы видели при моделировании:

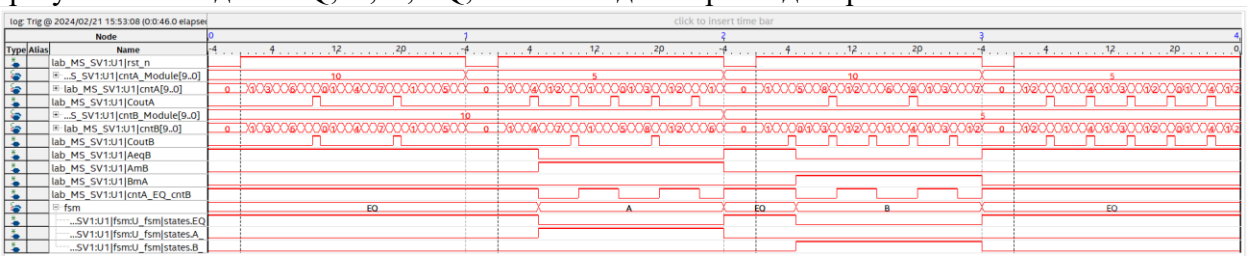

*Рис. 4.5. Результат измерений в Signal Tap II.*

Как можно заметить, *Рис. 4.2* и *Рис. 4.5* полностью совпадают, что свидетельствует о корректности работы устройства на ПЛИС.
После этого поменяем настройки Signal Tap II следующим образом:

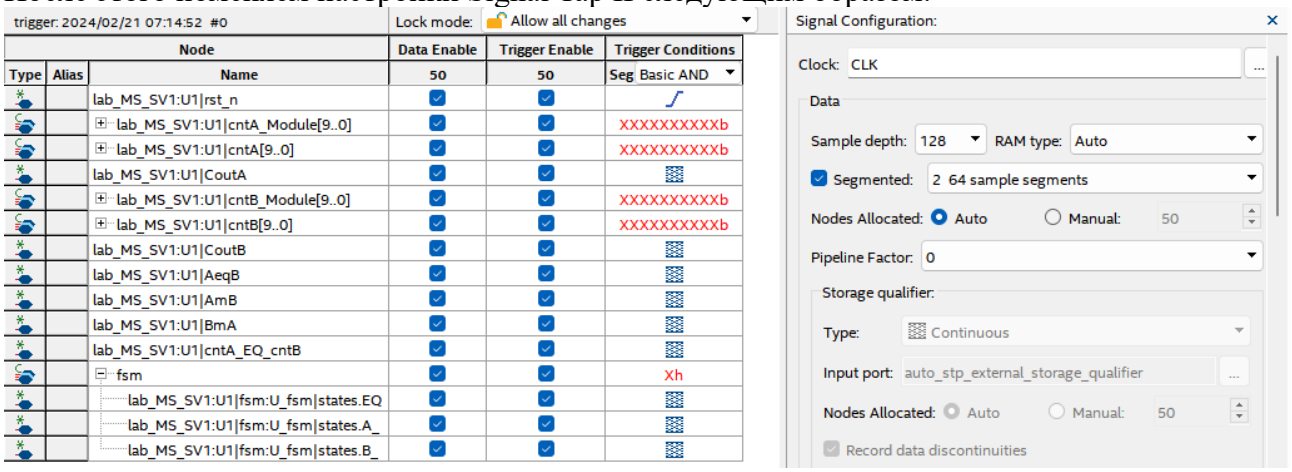


*Рис. 4.6. Новые настройки Signal Tap II.*

Таким образом будет выполняться захват 2 сегментов, вместо 4, как было ранее.
Запишем этот проект на плату и используя ISSP подадим в соответствии с вариантом, сначала 20, 40, а потом 40, 20. Ожидаем получить B, A. Между переключениями необходимо выполнить сброс.
На Signal Tap II получаем следующую диаграмму:

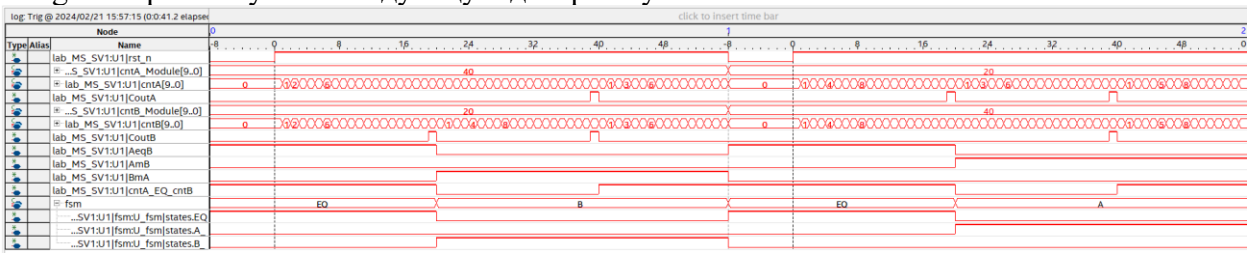

*Рис. 4.7. Результат измерений в Signal Tap II.*

Как мы видим, она полностью соответствует ожиданиям. Сначала при подаче комбинации 40; 20 мы получаем B, а после изменения их местами, получаем A.

# 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.