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)

“14” февраля 2024 г.

Санкт-Петербург
2024

Оглавление

1. Список иллюстраций:

2. Задача:

Изучить:

- Как создавать устройства в Quartus Prime.
- Как моделировать созданное устройство, используя модели.
- Как отладить созданное устройство используя ISSP и SignalTap II.
- Как реализовывать созданные устройства на плате.

3. Решение:

Возьмем код устройства из приложения к лабораторной:

```
`timescale 1ns / 1ns
module Lab3_1 #(
  parameter div_par = 25'd4
) (
  input            CLK,
  input            RST,
  input            DIR,
  output     [3:0] DIG,
  output reg [6:0] HEX
);
  reg  [24:0] div_cnt = 25'd1;  //Clock divider
  wire        cout;  //Carry out
  reg  [ 3:0] cnt_val;  //value to count
  reg  [ 1:0] rst_int = 2'd0;  //Synchronized reset
  reg  [ 3:0] Counter = 4'd0;  //Counter
  //=========================================================
  // Reset
  //
  always @(posedge CLK) rst_int <= {rst_int[0], RST};
  //=========================================================
  // Clock Divider
  //
  always @(posedge CLK, negedge rst_int[1])
    if (!rst_int[1]) div_cnt <= 25'd1;
    else div_cnt <= div_cnt + 25'd1;

  assign cout = (div_cnt == div_par);
  //=========================================================
  // Counter
  //
  always @(posedge CLK) cnt_val = (DIR) ? (-4'd1) : (4'd1);  //value for counting

  always @(posedge CLK, negedge rst_int[1])
    if (!rst_int[1]) Counter <= 4'd0;
    else if (cout) begin
      Counter <= Counter + cnt_val;
      case ({
        DIR, Counter
      })
        5'b10000: Counter <= 4'd9;
        5'b01001: Counter <= 4'd0;
      endcase
    end
  //=========================================================
  // Coder
  //
  always @(posedge CLK, negedge rst_int[1])
    if (!rst_int[1]) HEX <= 7'b0111111;
    else
      case (Counter)
        4'b0000: HEX <= 7'b0111111;  // "0"
        4'b0001: HEX <= 7'b0000110;  // "1"
        4'b0010: HEX <= 7'b1011011;  // "2"
        4'b0011: HEX <= 7'b1001111;  // "3"
        4'b0100: HEX <= 7'b1100110;  // "4"
        4'b0101: HEX <= 7'b1101101;  // "5"
        4'b0110: HEX <= 7'b1111101;  // "6"
        4'b0111: HEX <= 7'b0000111;  // "7"
        4'b1000: HEX <= 7'b1111111;  // "8"
        4'b1001: HEX <= 7'b1101111;  // "9"
        default: HEX <= 7'b0111111;  // "0"
      endcase
  //=========================================================
  // Constant value
  //
  assign DIG = 4'b1000;
  //=========================================================
endmodule


```

Данное устройство реализует счетчик от 0 до 9 на семисегментном индикаторе.
Протестируем это устройство тестом первого класса:

```
`timescale 1ns / 1ns
module tb_Lab3_1 ();
  reg        tb_clk;
  reg  [5:0] tb_mem   [0:127];
  reg        tb_dir;
  wire [3:0] tb_dig;
  wire [6:0] tb_hex;
  reg        tb_reset;
  wire [6:0] tb_ss;

  localparam CLK_PERIOD = 20;

  initial begin : clock_gen
    tb_clk = 1'b0;
    forever #(CLK_PERIOD / 2) tb_clk = ~tb_clk;
  end

  Lab3_1 #(25'd4) Lab3_1_inst (
    .CLK(tb_clk),
    .RST(tb_reset),
    .DIR(tb_dir),
    .DIG(tb_dig),
    .HEX(tb_hex)
  );

  initial begin : reset_gen
    tb_reset = 1'b0;
    #(CLK_PERIOD * 5) tb_reset = 1'b1;
  end

  initial begin : control_gen
    tb_dir = 1'b0;
    #(CLK_PERIOD * 55) tb_dir = 1'b1;
    #(CLK_PERIOD * 44);
  end

  initial begin : mem_reading
    $readmemb("ss_to_ascii.txt", tb_mem);
    #(CLK_PERIOD * 111) $stop;
  end

  assign tb_ss = tb_mem[tb_hex];

endmodule
```

Как можно заметить, сначала выполняется счет вверх, потом вниз. При этом первые 5 периодов CLK подается сигнал сброса. Для удобства добавлен модуль, который переводит из семисегментного кода в HEX.
Выполним компиляцию и посмотрим на результат запуска этого тестового модуля:

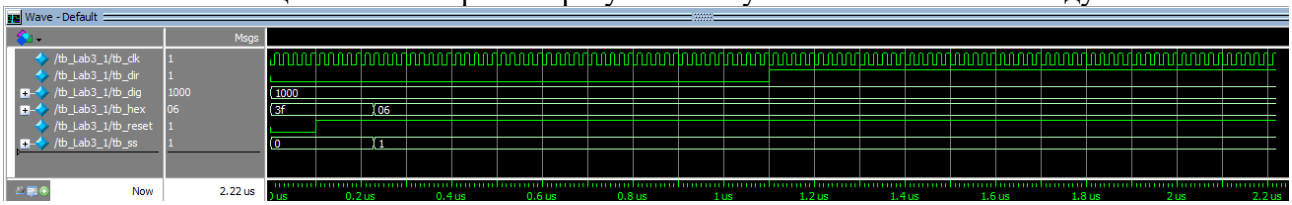

Рис. 3.1. Результат тестирования моудля.

Как мы видим, полученный результат отличается от ожидаемого. Добавим внутренние сигналы тестируемого модуля и найдем ошибку:

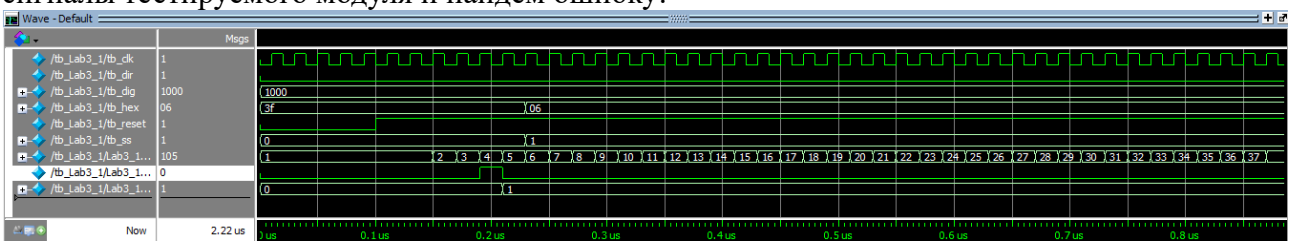

Рис. 3.2. Результаты моделирования с внутренними сигналами.

Как видно на этом скриншоте, счетчик-делитель вместо того, чтоб сброситься продолжает считать, поэтому в дальнейшем числа не меняются. Поправим эту ошибку, добавив в модуль счетчика сброс:

```
//=============================================================
// Clock Divider
//
always @(posedge CLK, negedge rst_int[1])
  if (!rst_int[1]) div_cnt <= 25'd1;
  else div_cnt <= cout ? 25'd1 : div_cnt + 25'd1;

assign cout = (div_cnt == div_par);
//=============================================================
```

Повторим тестирование модуля:

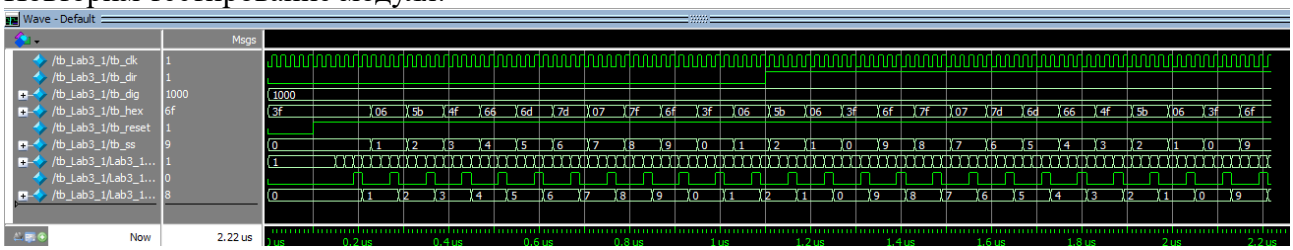

Рис. 3.3. Результат тестирования после исправления багов.

Как видим исправление этого модуля помогло и устройство работает корректно.

После моделирования устройство необходимо выполнить тестирование на плате. Разработаем для этого модуль, который позволит управлять всеми входами устройства с компьютера, а также отслеживать выходы:

```
module db_Lab3_1 (
  (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23"  *)
  input CLK
);
  wire [6:0] HEX;
  wire [3:0] DIG;
  wire       db_reset;
  wire       db_dir;
  wire       db_clk_high;

  Lab3_1 #(25'd25000000) Lab3_1_inst (
    .CLK(CLK),
    .RST(db_reset),
    .DIR(db_dir),
    .DIG(DIG),
    .HEX(HEX)
  );
  SP_unit SP_unit_inst (
    .source    ({db_reset, db_dir}),
    .probe     ({HEX, DIG}),
    .source_clk(CLK)
  );
  PLL_unit PLL_unit_inst (
    .inclk0(CLK),
    .c0    (db_clk_high)
  );
endmodule
```

Используя ISSP мы будем управлять входами направления счета и сброса, а отслеживать значение, передаваемое на счетчик.

Также создаем PLL, который выдает частоту в 2 раза больше заданной, чтоб используя SignalTap II видеть тактовый сигнал, однако пока SignalTap II не настроен, db_clk_high исчезнет при компиляции:

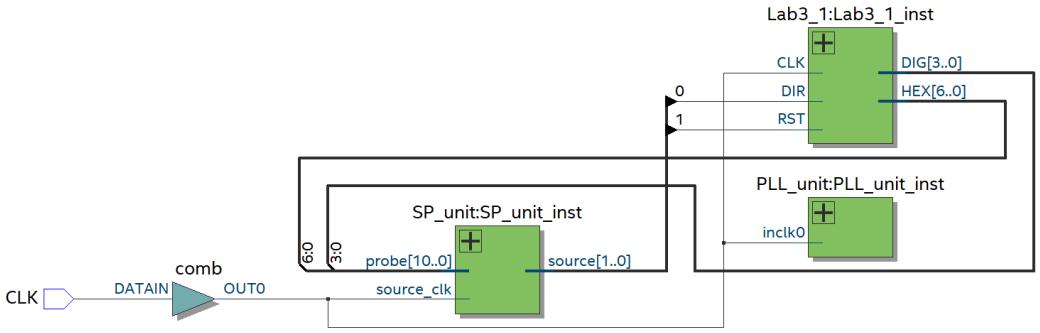

Рис. 3.4. RTL-Viewer модуля отладки.

Выполним настройку SignalTap II следующим образом:

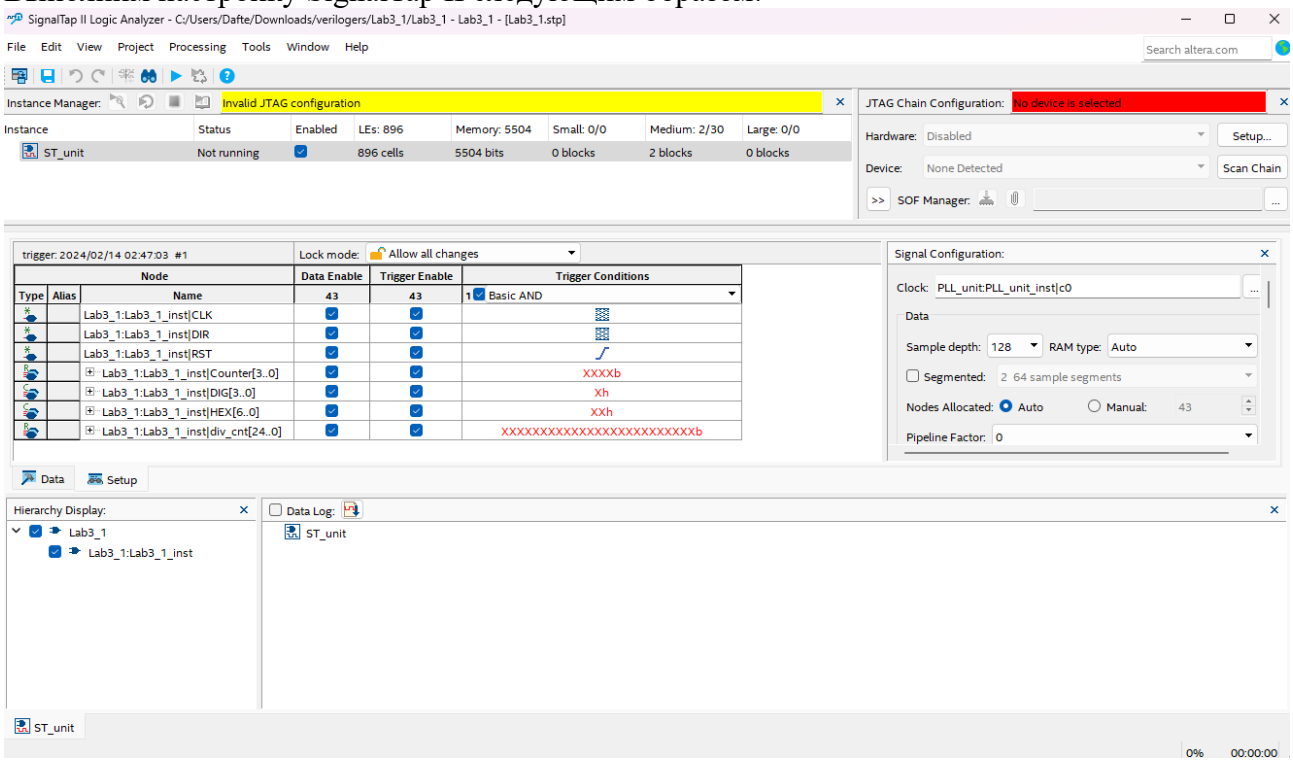

Рис. 3.5. Настройка SignalTap II.

Добавим к проекту SDC файл и выполним полную компиляцию. Получим следующий Fmax для разработанного устройства:

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	54.29 MHz	54.29 MHz	altera_reserved_tck
2	149.95 MHz	149.95 MHz	PLL_unit_inst|altpll_component|auto_generated|pll1|clk[0]
3	150.11 MHz	150.11 MHz	clock_in

Рис. 3.6. Slow 1200mV 85C Model Fmax Summary.

Здесь важно, чтоб clock_in работал со скоростью минимум 25 МГц (частота платы). Как мы видим, это действительно выполняется.

Запишем разработанное устройство на плату для дальнейшей отладки.

Выполним настройку In-System Source and Probes:

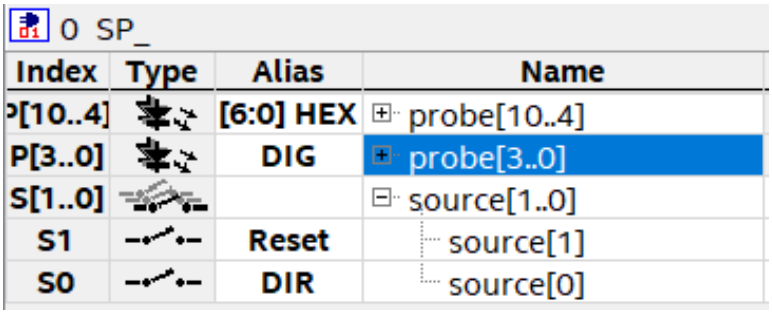

Рис. 3.7. In-System Source and Probes.

Запустим его, выставим Reset в 1 (устройство начнет работу) и посмотрим на результат:

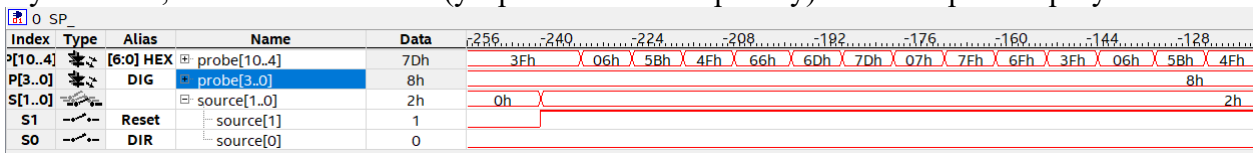

Рис. 3.8. In-System Source and Probes. Работа. DIR = 0.

Как мы видим, устройство корректно начало считать вверх, как и было при тестировании. Изменим направление счета и посмотрим, что устройство корректно считает и в обратную сторону:

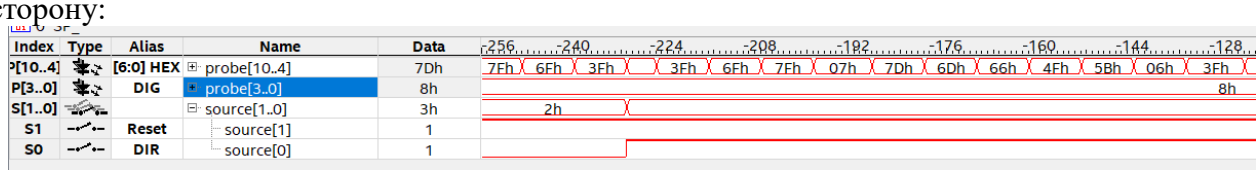

Рис. 3.9. In-System Source and Probes. Работа. DIR = 1.

Перейдем в SignalTap II и настроим таким образом, чтоб сигнал захватывался при div_cnt = 2500000 т.е. параметр, переданный в модуль ранее (момент переключения на следующее значение). После этого выполним запуск и проучим следующий результат:

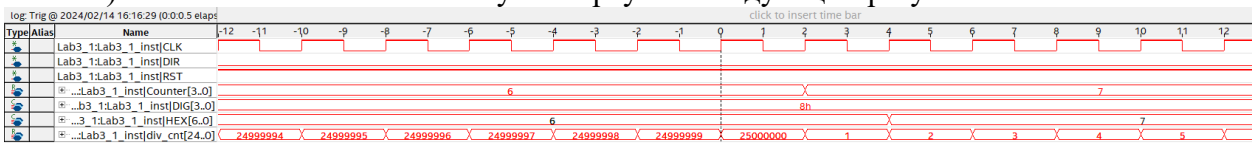

Рис. 3.10. SignalTap II. Захват при div_cnt = 2500000.

Мы захватили момент перехода к следующей цифре, что можно заметить на скриншоте. Так же важно отметить, что благодаря PLL мы корректно наблюдаем сигнал CLK.

Теперь изменим настройки SignalTap II следующим образом:

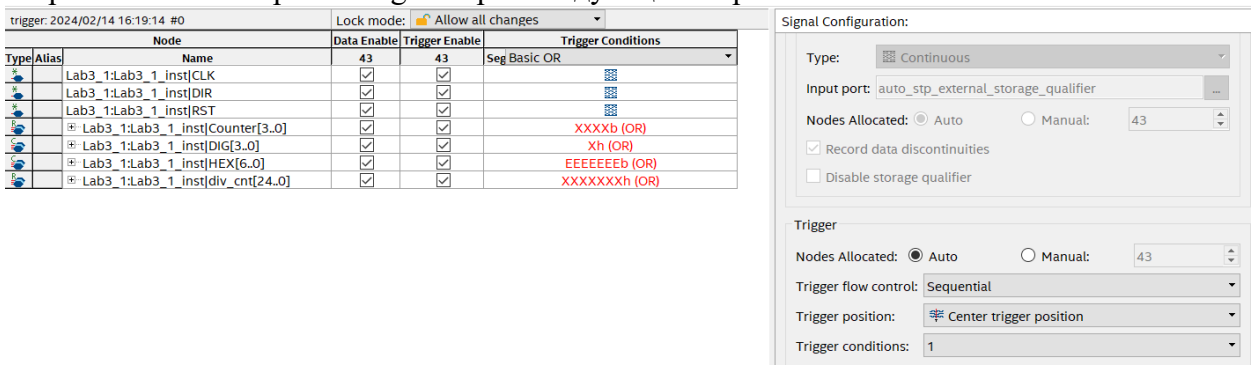

Рис. 3.11. Измененные настройки SignalTap II.

Они позволят нам захватывать именно изменения HEX выхода и так 16 раз, в каждом по 8 измерений, это удобно позволит отслеживать изменения на этом выходе.

Поставим Reset = 1, DIR = 0 и запустим захват данных на SignalTap II:

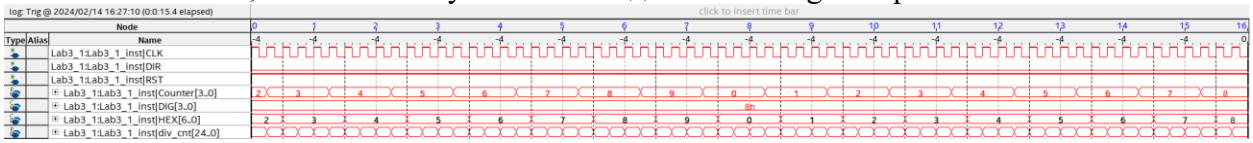

Рис. 3.12. SignalTap II. Захват 16 сегментов по 8 измерений. DIR = 0.

Как и ожидалось мы видим как значение растет на отладке.
Повторим измерения, установив DIR = 1:

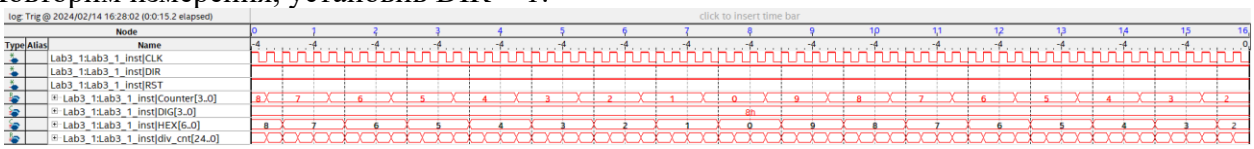

Рис. 3.13. SignalTap II. Захват 16 сегментов по 8 измерений. DIR = 1.

А теперь мы получаем направление счета вниз.
Теперь создадим модуль, который будет полноценно реализовывать модуль на плате:

```
module impl_Lab3_1 (
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23"  *)
  input CLK,
    (* altera_attribute = "-name IO_STANDARD \"2.5 V\"", chip_pin = "64" *)
  input RST,
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "88" *)
  input DIR,
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "84, 76, 85, 77, 86, 133, 87" *)
  output [6:0] HEX,
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "73, 74, 80, 83" *)
  output [3:0] DIG
);

  Lab3_1 #(25'd25000000) Lab3_1_inst (
    .CLK(CLK),
    .RST(RST),
    .DIR(DIR),
    .DIG(DIG),
    .HEX(HEX)
  );

endmodule
```

Рис. 3.14. Модуль реализующий проект.

Его RTL схема приведена ниже:

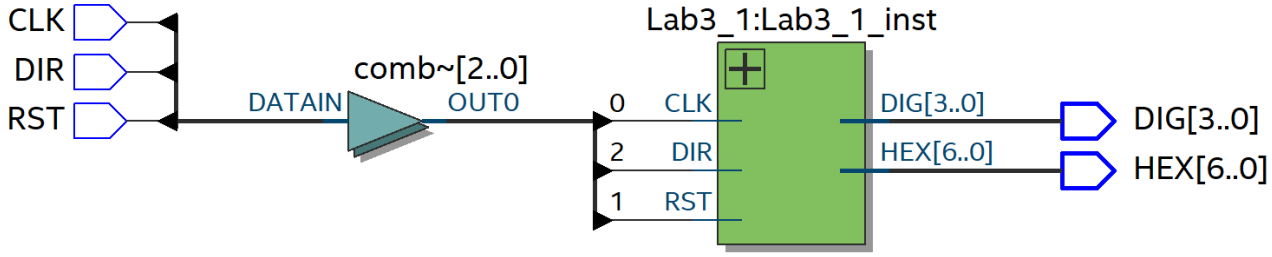

Рис. 3.15. RTL Viewer итогового проекта.

Как по нему видно, этот файл действительно только задает входы и выходы созданному модулю.
Также добавим SDC файл для временных требований, включим SignalTap II и запустим полную компиляцию.
Используя Timing Analyzer, убедимся, что все временные требования выполняются, а также что SignalTap II действительно выключился:

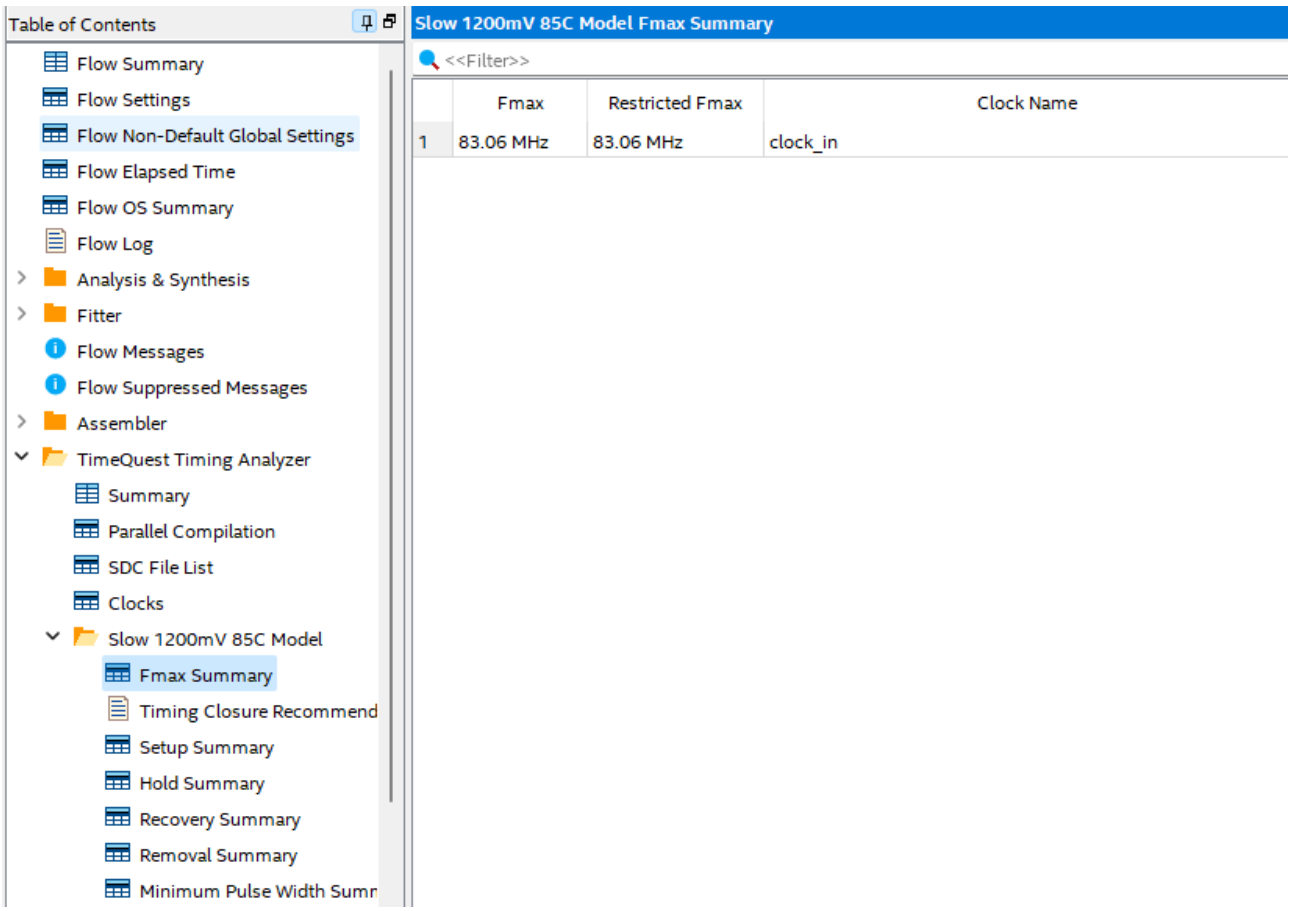

Рис. 3.16. Timing Analyzer report.

Запишем полученный проект на плату. Данное устройство работает корректно и было продемонстрированно преподавателю.

4. Вывод:

В ходе лабораторной работы были получены навыки по многоэтапному тестированию устройства, сначала используя моделирование (без платы) средствами ModelSim, после чего тестирование уже на плате, используя In-System Sources and Probes Editor и SignalTap II и последующую реализацию модуля, готового к полноценному использованию на плате.

Данные навыки помогут при разработке как маленьких проектов, так и больших, которые не так просто отлаживать. ModelSim, In-System Sources and Probes Editor и SignalTap II очень сильно ускоряют отладку, что несомненно важно при работе с любым проектом.