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

Отчёт по лабораторной работе № 10

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

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

Руководитель _____ А.П. Антонов
(подпись)

“14” апреля 2024 г.

Санкт-Петербург
2024

Оглавление

1. Список иллюстраций:

2. Алгоритм работы проекта:

Средствами Platform Designer создать структуру проекта, представленную на рисунке ниже:

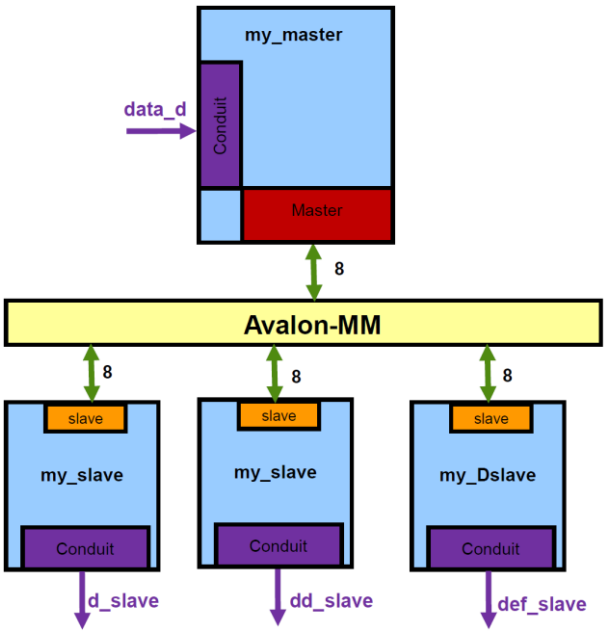

Рис. 2.1. Схема разрабатываемого устройства.

3. Ход работы:

Выполним создание проекта со стандартными настройками, после чего создадим описания модулей my_master, my_slave и my_Dslave.

Модуль my_master будет выглядеть следующим образом:

lab_PD4 - my_master.sv

```
`timescale 1ns/1ns

module my_master (
    // clock and reset
    input bit        csi_clk,             // clock clk
    input bit        rsi_reset,           // reset reset
    // Avalon MM master
    output bit [7:0] avm_m0_address,      // MM master address
    output bit       avm_m0_write,        // MM master write
    output bit [7:0] avm_m0_writedata,    // MM master writedata
    output bit       avm_m0_waitrequest,  // MM master waitrequest
    // conduit
    input bit [7:0]  coe_c0_DA
);

    typedef enum bit [1:0] {initSM, del1, wr1D, del2} fsm_type;
    fsm_type fsm_MM;
    bit [7:0] cnt_intA;

    always_ff @(posedge csi_clk) begin
        if (rsi_reset) begin
            fsm_MM   <= initSM;
            cnt_intA <= 8'd0;
        end
        else begin
            case (fsm_MM)
                initSM: fsm_MM <= del1;
                del1:   fsm_MM <= wr1D;
                wr1D:   if (avm_m0_waitrequest)
                            fsm_MM <= wr1D;
                        else
                            fsm_MM <= del2;
                del2:   begin
                            fsm_MM <= initSM;
                            cnt_intA <= cnt_intA + 8'd1;
                        end
            endcase
        end
    end

    always_comb begin
        case (fsm_MM)
            wr1D:
                begin
                    avm_m0_address     = cnt_intA;
                    avm_m0_write       = 1'd1;
                    avm_m0_writedata   = cnt_intA + coe_c0_DA;
                end
            default:
                begin
                    avm_m0_address     = 8'd255;
                    avm_m0_write       = 1'd0;
                    avm_m0_writedata   = 8'd255;
                end
        endcase
    end
endmodule

```

Модуль my_master функционирует как главное устройство в системе Avalon Memory-Mapped (MM). Он управляет передачей данных от мастера к другим компонентам.
Модуль работает на основе конечного автомата (FSM), который имеет четыре состояния: initSM, del1, wr1D, del2.

- initSM: Начальное состояние.
- del1: Задержка для ожидания данных от мастера Avalon MM (чтобы отделить циклы записи по шине, это не обязательно, но так будет наглядно при просмотре waveform).
- wr1D: Ожидание завершения операции записи данных от мастера Avalon MM.
- del2: Дополнительная задержка после завершения операции записи.

RTL Viewer выглядит следующим образом:

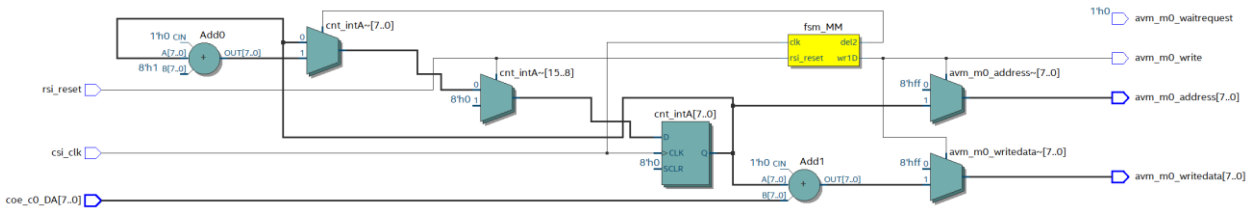

Рис. 3.1. RTL Viewer для my_master.

Модуль my_slave выглядит следующим образом:

lab_PD4 - my_slave.sv

```
1  `timescale 1ns/1ns
2  module my_slave (
3  // clock amd reset
4      input bit        csi_clk, // clocl clk
5      input bit        rsi_reset, // reset reset
6  // MM Slave
7      input bit [7:0]  avs_s0_writedata, // MM Slave writedata
8      input bit        avs_s0_write, // MM Slave write
9      output bit        avs_s0_waitrequest, // MM Slave wairequest
10 // Conduit
11     output bit [7:0] coe_s0_Dout
12 );
13     bit [7:0] rg_DATA;
14     assign avs_s0_waitrequest = 1'b0;
15
16     always_ff @(posedge csi_clk) begin
17         if (rsi_reset)
18             rg_DATA <= 8'd0;
19         else if (avs_s0_write)
20             rg_DATA <= avs_s0_writedata;
21     end
22     assign coe_s0_Dout = rg_DATA;
23 endmodule
24
```

Модуль my_slave в интерфейсе Avalon Memory-Mapped (MM) функционирует как подчинённое устройство, принимая данные от мастера и передавая их через выходной сигнал coe_s0_Dout. Он использует тактовый сигнал csi_clk для синхронизации операций и сигнал сброса rsi_reset для инициализации внутренних состояний. Модуль содержит 8-битный регистр данных rg_DATA, который обновляется при каждом положительном фронте csi_clk, если активирован сигнал записи avs_s0_write. При активации сигнала сброса регистр rg_DATA сбрасывается в ноль. Данные, хранящиеся в регистре rg_DATA, передаются через выходной сигнал coe_s0_Dout. Сигнал avs_s0_waitrequest всегда устанавливается в ноль, что означает отсутствие запроса на ожидание со стороны подчинённого устройства.

Посмотрим, как выглядит диаграмма этого модуля в RTL Viewer:

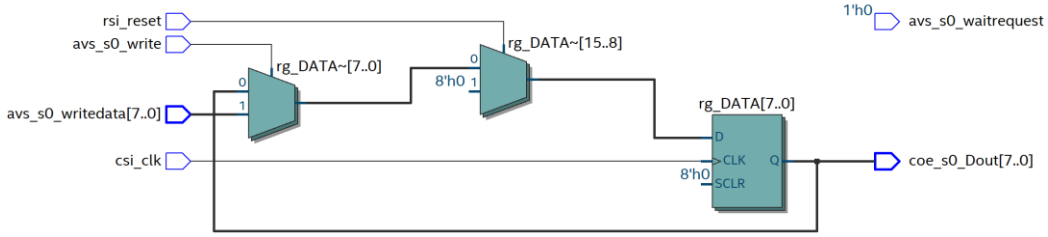

Рис. 3.2. RTL Viewer для my_slave.

Далее разработаем модуль my_Dslave:

lab_PD4 - my_Dslave.sv

```
1  `timescale 1ns/1ns
2  module my_Dslave (
3  // clock amd reset
4      input bit        csi_clk, // clocl clk
5      input bit        rsi_reset, // reset reset
6  // MM Slave
7      input bit [7:0]  avs_s0_writedata, // MM Slave writedata
8      input bit        avs_s0_write, // MM Slave write
9      output bit        avs_s0_waitrequest, // MM Slave wairequest
10 // Conduit
11     output bit [7:0] coe_s0_Dout
12 );
13     bit [7:0] cnt_;
14     assign avs_s0_waitrequest = 1'b0;
15
16     always_ff @(posedge csi_clk) begin
17         if (rsi_reset)
18             cnt_ <= 8'd0;
19         else if (avs_s0_write)
20             cnt_ <= cnt_ + 8'd1;
21     end
22     assign coe_s0_Dout = cnt_;
23 endmodule
24
```

Модуль my_Dslave является простым устройством в системе, которое принимает данные от мастера Avalon MM и передаёт их через интерфейс Conduit. Когда мастер отправляет данные, my_Dslave сохраняет их во внутреннем регистре и затем передаёт через выходной порт coe_s0_Dout через интерфейс Conduit без каких-либо изменений. Это позволяет эффективно передавать данные от мастера Avalon MM к другим частям системы, используя my_Dslave в качестве посредника, без необходимости дополнительной обработки или изменений данных.
RTL Viewer выглядит следующим образом:

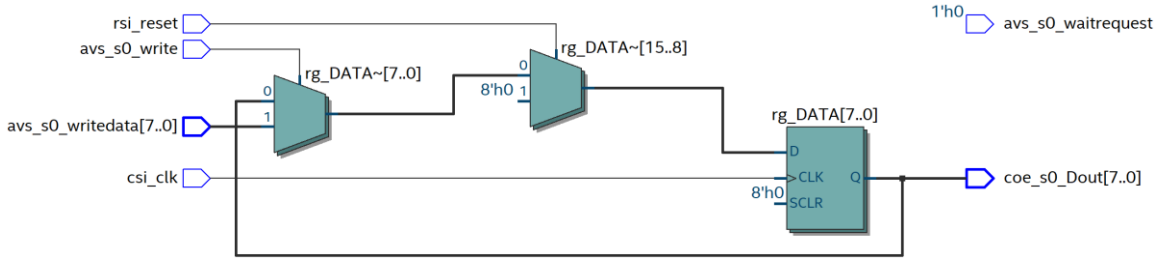

Рис. 3.3. RTL Viewer модуля my_Dslave.

Перейдем в Platform Disigner:

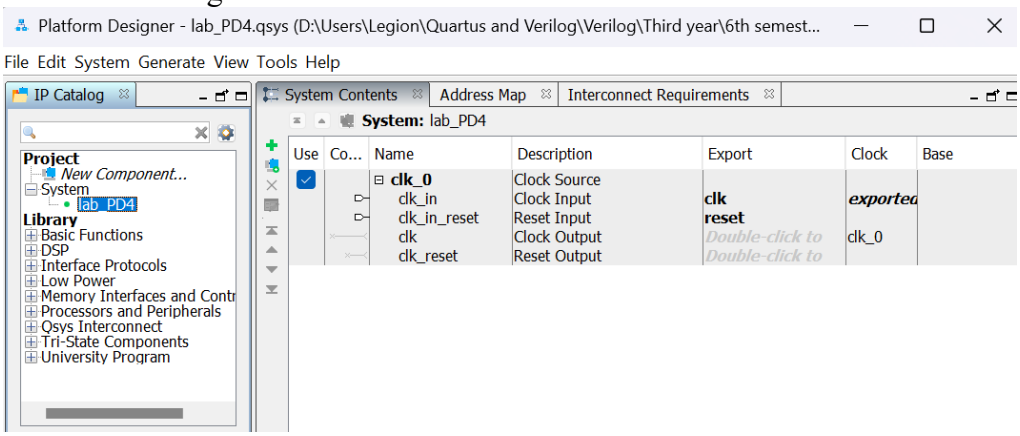

Рис. 3.4. Platform Designer.

Добавим в PD ранее созданные компоненты:

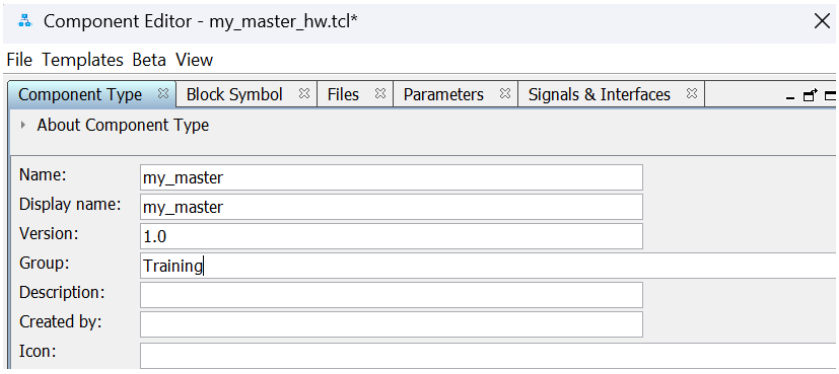

Рис. 3.5. Component Type для my_master.

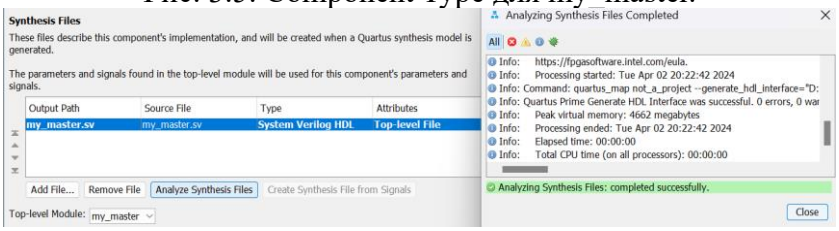

Рис. 3.6. Files для my_master.

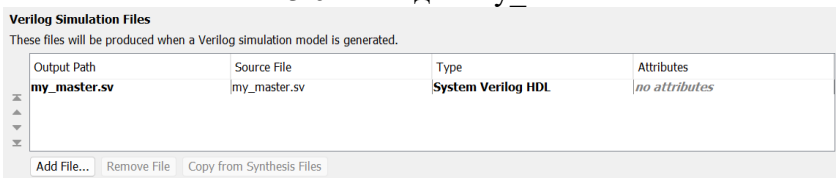

Рис. 3.7. Выбор файла симуляции для my_master.

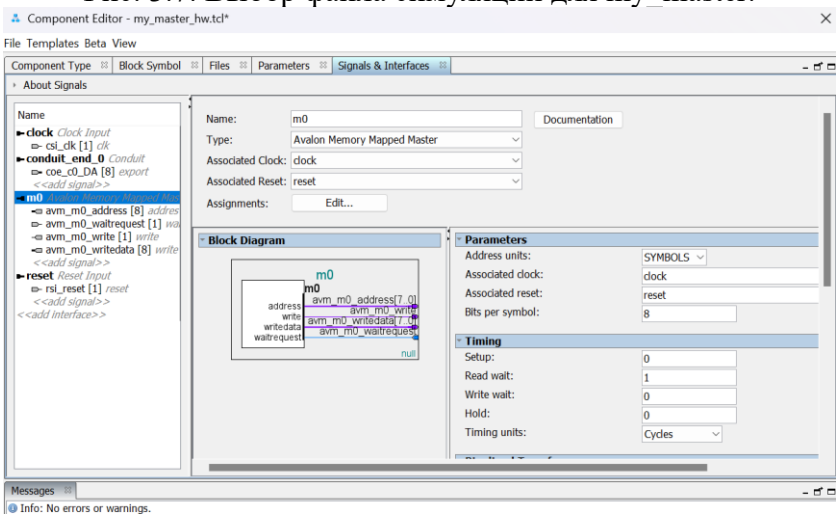

Рис. 3.8. Signals & Interfaces для my_master.

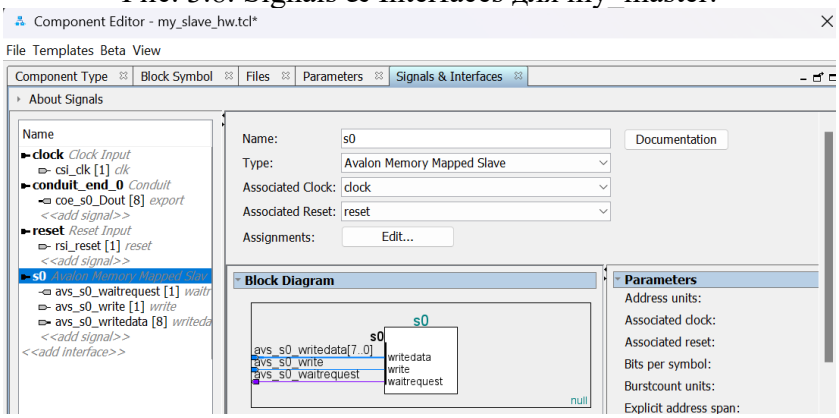

Рис. 3.9. Signal & Interfaces для slave.

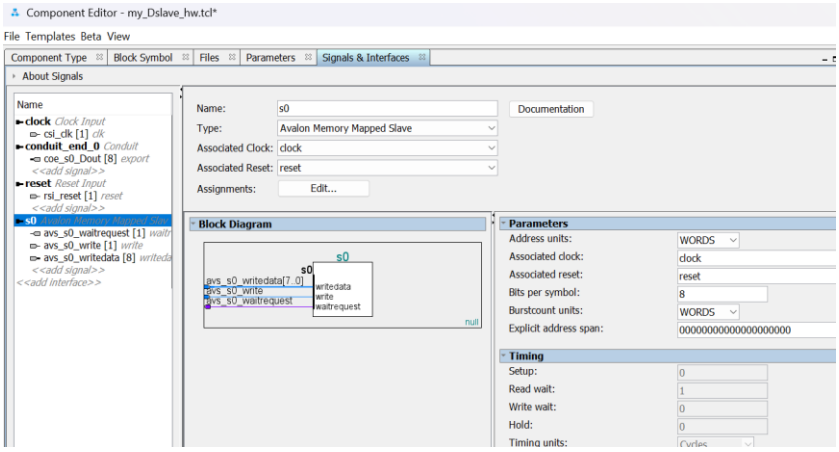

Рис. 3.10. Signal & Interfaces для Dslave.

Теперь добавим модули в систему:

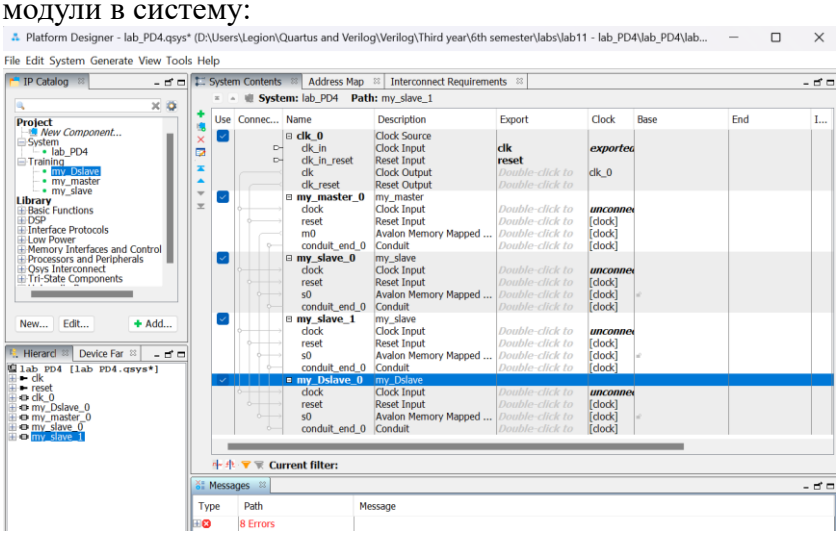

Рис. 3.11. Элементы в Platform Designer.

Выполним настройку моудля clk:

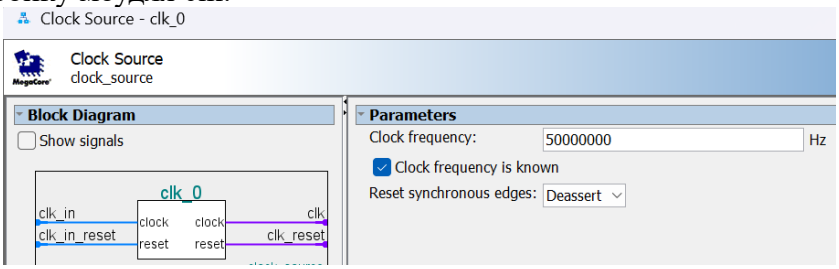

Рис. 3.12. Настройка сигналов clk.

Подключим все компоненты друг к другу:

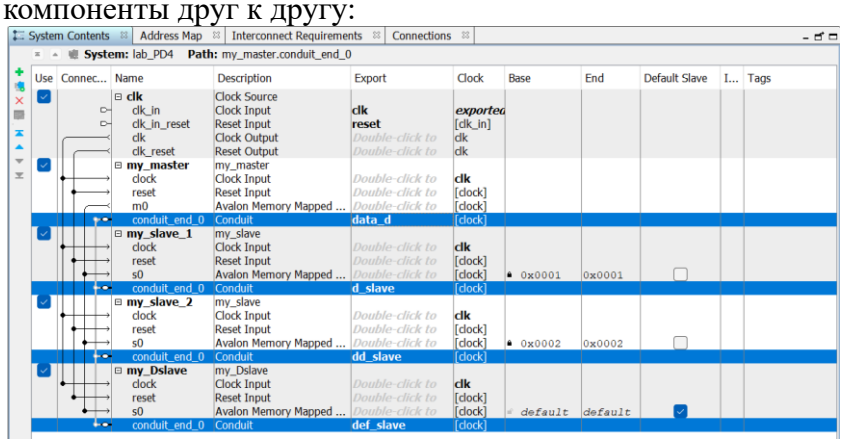

Рис. 3.13. Подключения в модуле.

Произведем анализ разработанного модуля:

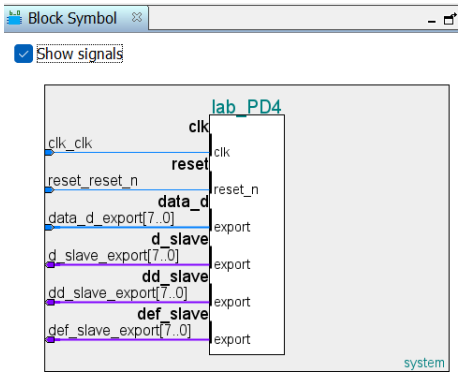

Рис. 3.14. Символ системы.

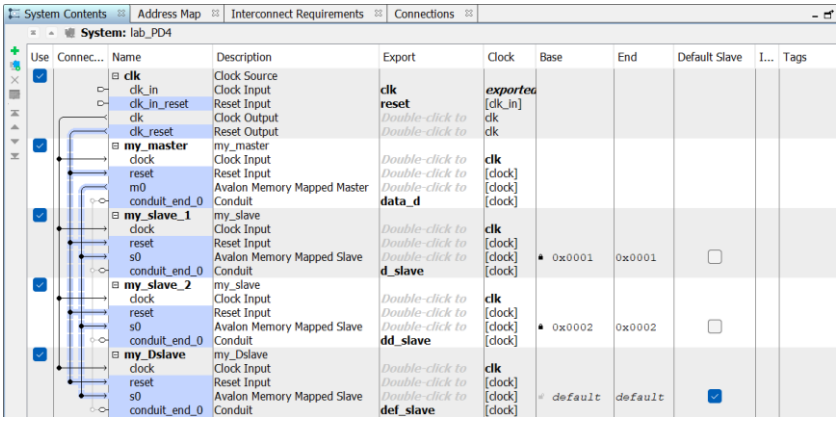

Рис. 3.15. Анализ проблемных подключений.

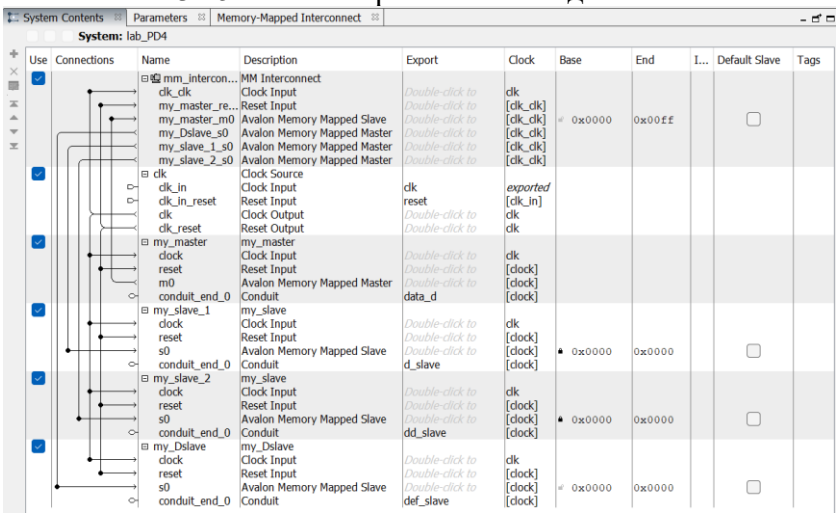

Рис. 3.16. System with PD Interconnect.

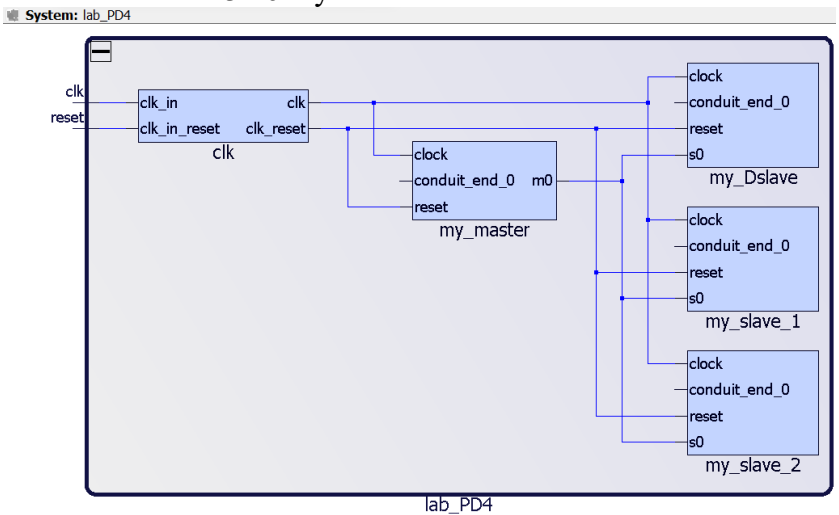

Рис. 3.17. Schematic.

Как видим все выполнено в соответствии с заданием.
Выполним генерацию и подключим получившиеся файлы к проекту:

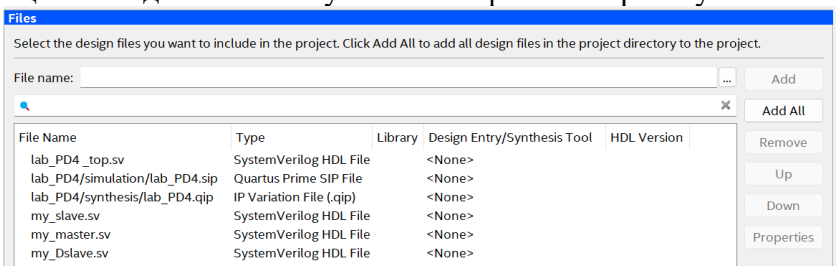

Рис. 3.18. Подключение файлов к проекту.

Далее создадим файл верхнего уровня:

lab_PD4 - lab_PD4 _top.sv

```
`timescale 1ns/1ns
module lab_PD4_top (
    input bit clk,
    input bit reset,
    input bit [7:0] data_d,
    output bit [7:0] dd_slave,
    output bit [7:0] d_slave,
    output bit [7:0] def_slave,
);
    lab_PD4 lab4_sys_inst (
        .clk_clk          (clk),
        .reset_reset_n    (reset),
        .def_slave_export (def_slave),
        .dd_slave_export  (d_slave),
        .d_slave_export   (d_slave),
        .data_d_export    (data_d)
    );
endmodule

```

RTL Viewer данного проекта приведен ниже:

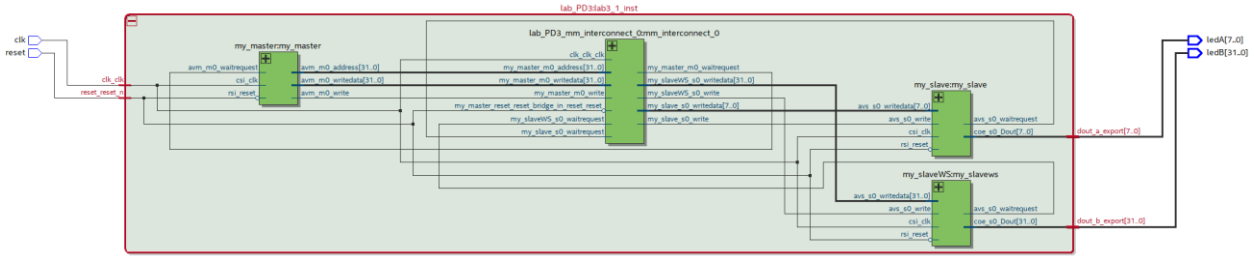

Рис. 3.19. RTL Viewer.

Выполним тестирование разработанного файла, используя следующий тестовый модуль:

lab_PD4 - tb_lab_PD4_top.sv

```
`timescale 1ns/1ns
module tb_lab_PD4_top ();
    bit clk,
    bit reset,
    bit [7:0] data_d,
    bit [7:0] dd_slave,
    bit [7:0] d_slave,
    bit [7:0] def_slave
assign data_d = 8'd6;
always #10 clk = ~ clk;
initial begin
    #20;
    reset = 1'b1;
    reset (4*9) @(negedge clk);
    $stop;
end
    lab_PD4 Lab4_sys_inst (.*);
endmodule

```

Запустим тестовый файл и получим следующий результат:

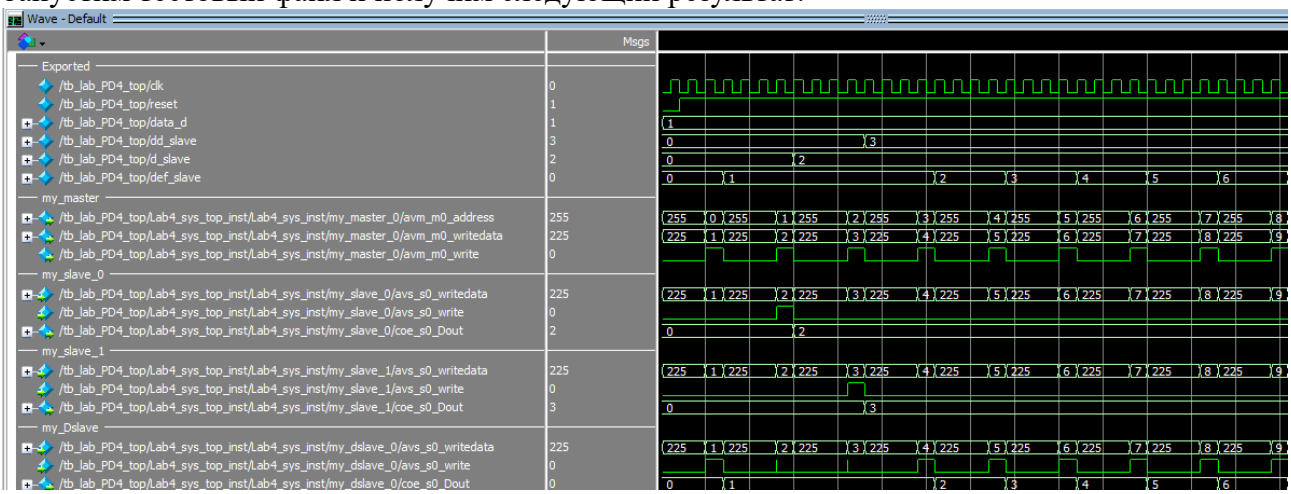

Рис. 3.20. Результат тестирования.

Как мы видим, данные доставляются в соответствии с заданными адресами. А при несовпадении адреса отправляются в my_Dslave, где внутренний счетчик увеличивается на 1, что соответствует заданию.

Поменяем сигнал data_d – смещение записываемого значения, относительно адреса на 18 (в соответствии с вариантом и повторим запуск тестируемого модуля:

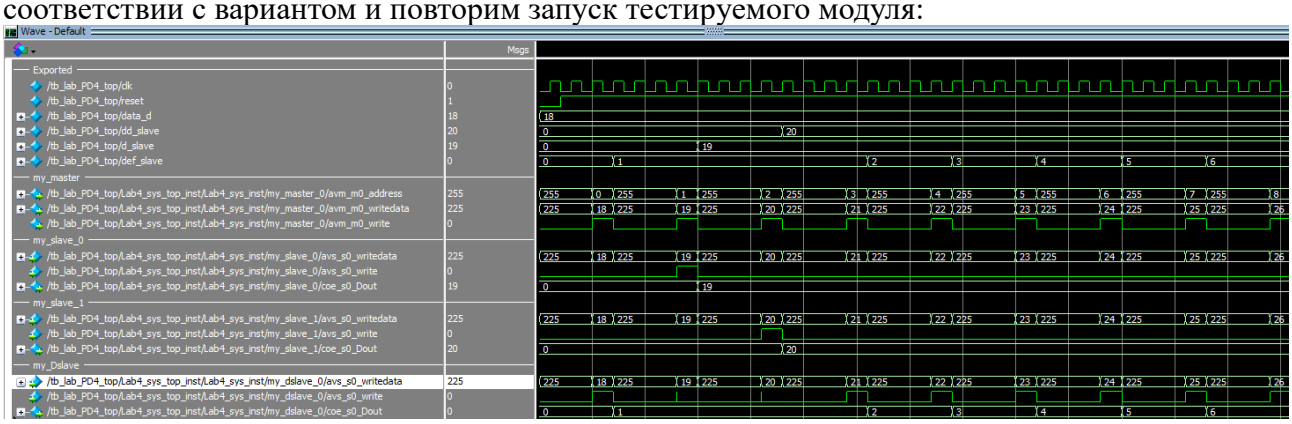

Рис. 3.21. Результат тестирования со смещением.

Как мы видим, все работает корректно. Теперь перейдем к тестированию непосредственно на плате. Для этого разработаем следующий модуль:

lab_PD4 - db_lab_PD4_top.sv

```
1  `timescale 1ns/1ns
2  module db_lab_PD4_top (
3      (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23" *)
4      input bit clk
5  );
6      bit reset;
7      bit [7:0]  data_d;
8      bit [7:0] dd_slave;
9      bit [7:0] d_slave;
10     bit [7:0] def_slave;
11 SP_unit SP_unit_inst (
12     .source ( {reset, data_d} ),
13     .source_clk (clk)
14     );
15 lab_PD4_top lab_PD4_top_inst (.*);
16 endmodule
17
```

RTL_Viewer разработанного модуля выглядит следующим образом:

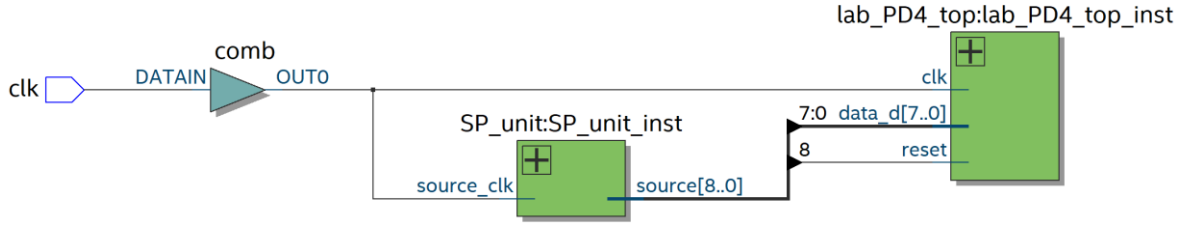

Рис. 3.22. RTL Viewer.

Выполним компиляцию разработанного модуля и посмотрим на результат в Signal Tap:

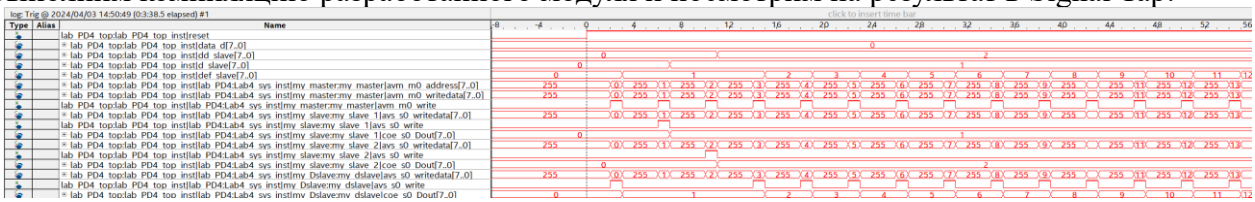

Рис. 3.23. Signal Tap.

Полученный результат совпадает со схемой, полученной в Model Sim, что свидетельствует о корректности разработанного устройства.

4. Вывод:

В ходе выполнения лабораторной работы была создана структура проекта с использованием Platform Designer. Этот инструмент предоставляет графический интерфейс для интеграции и настройки IP-блоков, что упрощает и ускоряет процесс разработки в сравнении с ручным написанием кода.

Преимущества Platform Designer включают в себя:

1. Ускорение разработки: Платформа предлагает готовые IP-блоки, которые могут быть легко интегрированы в проект, сокращая время разработки.
2. Упрощение процесса: Использование графического интерфейса позволяет избежать сложностей в написании и отладке кода, особенно для начинающих разработчиков.
3. Модульность и повторное использование: IP-блоки могут быть использованы в разных проектах, что способствует повторному использованию кода и упрощает поддержку проектов.
4. Визуализация системы: Платформа предоставляет графическое представление структуры проекта, что облегчает понимание и анализ системы.

Таким образом, использование Platform Designer обеспечивает эффективное управление проектом, ускоряет его разработку и повышает его надежность за счет готовых компонентов и удобного интерфейса.