

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

Отчёт по лабораторной работе Lab_MS_SV_5

Дисциплина: Автоматизация проектирования дискретных устройств (на
английском языке)

Выполнил студент гр. 5130901/10101 _____ М.Т. Непомнящий
(подпись)

Руководитель _____ А.А. Федотов
(подпись)

Санкт-Петербург
2024

Оглавление

1.	Задание	4
1.1.	Цель работы	4
1.2.	Алгоритм работы	4
1.3.	Программа работы:	5
2.	Ход решения	6
2.1.	Разработка модулей задания	6
2.1.1.	Модуль lab_MS_SV5_interface	6
2.1.2.	Модуль master	6
2.1.3.	Модуль slave_EVEN	7
2.1.4.	Модуль slave_ODD	8
2.1.5.	Модуль lab_MS_SV5	8
2.1.6.	Структура устройства	9
2.2.	Создание теста первого класса	9
2.3.	Создания модуля для тестирования на плате	10
2.4.	Настройка Signal Tap II	11
2.5.	Тестирование на плате средствами Signal Tap II	11
3.	Дополнительное задание	12
3.1.	Изменение модулей	12
3.1.1.	Изменение модуля интерфейса	12
3.1.2.	Изменение модуля master	12
3.1.3.	Изменение модуля slave_ODD	13
3.1.4.	Изменение модуля slave_EVEN	13
3.2.	Анализ получившегося устройства	13
3.3.	Тестирование на плате средствами Signal Tap II	14
4.	Вывод	16

Список иллюстраций

Рис. 1 – Структура разрабатываемого устройства	4
Рис. 2 – Граф состояний автомата Мура	5
Рис. 3 – Модуль интерфейса lab_MS_SV5_interface	6
Рис. 4 – Модуль master	7
Рис. 5 – Модуль ведомого устройства slave_EVEN	7
Рис. 6 – Модуль ведомого устройства slave_ODD	8
Рис. 7 – Модуль верхнего уровня lab_MS_SV5	8
Рис. 8 – Структура разработанного устройства в RTL Viewer	9
Рис. 9 – Тестовый файл первого класса для устройства	9
Рис. 10 – Моделирование тестового файла средствами ModelSim (wave)	10
Рис. 11 – Модуль db для тестирования на плате	10
Рис. 12 – Настройка окна Signal Tap II	11
Рис. 13 – Мнемоническая таблица	11
Рис. 14 – Временные характеристики устройства	11
Рис. 15 – Результат SignalTap II	11
Рис. 16 – Модифицированный модуль интерфейса	12
Рис. 17 – Модифицированный модуль master	12
Рис. 18 – Модифицированный модуль slave_ODD	13
Рис. 19 – Модифицированный модуль slave_EVEN	13
Рис. 20 – Структура модифицированного устройства в RTL Viewer	14
Рис. 21 – Моделирование мод. устройства средствами ModelSim (wave)	14
Рис. 22 – Настройка окна Signal Tap II (доп. задание)	15
Рис. 23 – Временные характеристики устройства	15
Рис. 24 – Результат SignalTap II	15

1. Задание

1.1. Цель работы

Разработать устройство, структура которого будет выглядеть следующим образом:

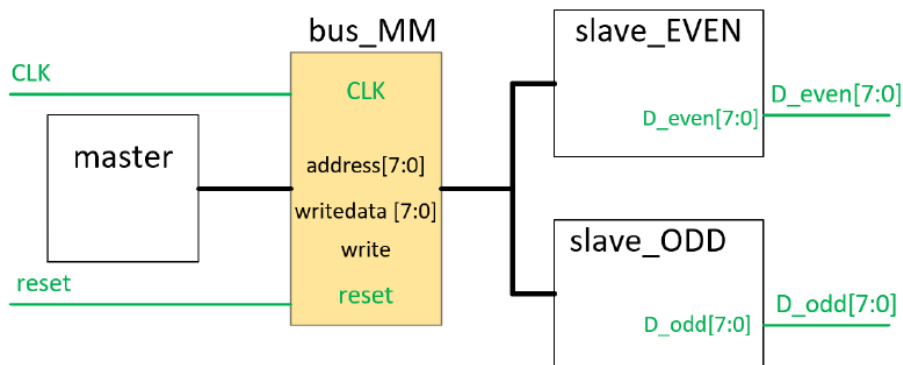


Рис. 1 – Структура разрабатываемого устройства

В состав устройства входят:

- Модуль master – ведущее устройство, формирует обращение к двум ведомым устройствам.
- Модули slave_EVEN и slave_ODD – ведомые устройства, управляемые мастером.
- Модуль bus_MM - экземпляр интерфейса, обеспечивающий подключение мастера и ведомых устройств.

Выходы устройства (выделены зеленым цветом):

- CLK – вход тактового сигнала.
- reset – синхронный сброс всех устройств
- D_odd – восьмиразрядный выход
- D_even – восьмиразрядный выход

1.2. Алгоритм работы

Алгоритм работы разрабатываемого устройства определяется алгоритмами работы его модулей:

- Модуль master:
 - Содержит конечный автомат Мура с тремя состояниями: initSM, por, wr1D (его граф состояний приведён на).
 - Генерирует комбинационные сигналы:
 - address (8 бит) - адреса.
 - writedata (8 бит) - данные.
 - write - разрешение записи.
 - Использует 8-разрядный счетчик cnt для формирования адреса и данных.

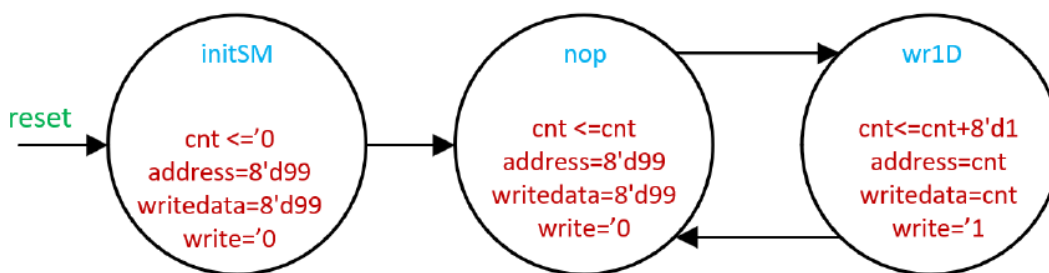


Рис. 2 – Граф состояний автомата Мура

- Модуль slave_EVEN:
 - Записывает значение шины данных во внутренний регистр, если адрес четный и разрешение на запись установлено.
 - Выдаёт значение внутреннего регистра на выход.
- Модуль slave_ODD:
 - Записывает значение шины данных во внутренний регистр, если адрес нечетный и разрешение на запись установлено.
 - Выдаёт значение внутреннего регистра на выход.

1.3. Программа работы:

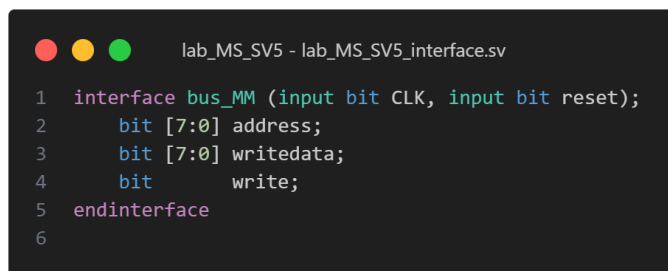
1. Разработать описание интерфейса lab_MS_SV5_interface.sv с сигналами CLK и reset.
2. Создать модуль master в файле master.sv, используя интерфейс.
3. Создать модуль slave_ODD в файле slave_ODD.sv, используя интерфейс.
4. Создать модуль slave_EVEN в файле slave_EVEN.sv, используя интерфейс.
5. Объединить модули в верхнеуровневый модуль lab_ms_sv5 в файле lab_ms_sv5.sv, используя интерфейс.
6. Скомпилировать модуль lab_ms_sv5 в Quartus.
7. Отобразить структуру устройства с помощью RTL Viewer, включая проверку и включение структуры в отчет.
8. Разработать тестовый модуль tb_lab_MS_SV5.sv с временной диаграммой результатов теста.
9. Разработать отладочный модуль db_lab_MS_SV5.sv, содержащий lab_ms_sv5 и SP_unit для задания сигнала reset.
10. Создать файл lab_ms_sv5.stp для SignalTapII с настройками, соответствующими временной диаграмме из моделирования.

2. Ход решения

2.1. Разработка модулей задания

2.1.1. Модуль lab_MS_SV5_interface

Создадим файл lab_MS_SV5_interface.sv, который будет представлять из себя интерфейс для взаимодействия между разными модулями в устройстве:



```
lab_MS_SV5 - lab_MS_SV5_interface.sv
1 interface bus_MM (input bit CLK, input bit reset);
2     bit [7:0] address;
3     bit [7:0] writedata;
4     bit      write;
5 endinterface
6
```

Рис. 3 – Модуль интерфейса lab_MS_SV5_interface

В данном случае интерфейс определяет структуру работы шины bus_MM, которая используется для передачи данных и управляющих сигналов между модулями master, slave_EVEN и slave_ODD.

На входы этого интерфейса поступают сигнал CLK, а также reset, необходимый для перехода в начальное состояние конечного автомата initSM.

На выходы подаётся 3 вида сигналов:

- address – сигнал адреса, используемый для доступа к различным устройствам на шине.
- writedata: данные, передаваемые по шине для записи в устройства.
- write: управляющий сигнал, указывающий, что на шину отправляются данные.

Таким образом, модуль lab_MS_SV5_interface обеспечивает общий интерфейс для всех модулей в системе, что позволяет им взаимодействовать друг с другом посредством передачи данных и управляющих сигналов через шину bus_MM.

2.1.2. Модуль master

Используя созданный выше интерфейс, разработаем модуль master, в котором будет содержаться описание работы автомата Мура, алгоритм работы которого приведён в задании (Рис. 2).

```

lab_MS_SV5 - master.sv

1 `timescale 1 ns / 1 ns
2 module master (bus_MM bus);
3     enum bit[1:0] {initSM, nop, wr1D} fsm_MM;
4     bit [7:0] cnt;
5     always_ff @ (posedge bus.CLK)
6     if (bus.reset) begin
7         fsm_MM <= initSM;
8         cnt <= '0;
9     end else
10    case (fsm_MM)
11        initSM : fsm_MM <= nop;
12        nop : fsm_MM <= wr1D;
13        wr1D : begin
14            fsm_MM <= nop;
15            cnt <= cnt + 8'd1;
16        end
17    endcase
18    always_comb begin
19        case (fsm_MM)
20            wr1D: begin
21                bus.address = cnt;
22                bus.write = '1;
23                bus.writedata = cnt;
24            end
25            default: begin
26                bus.address = 8'd99;
27                bus.write = '1'd0;
28                bus.writedata = 8'd99;
29            end
30        endcase
31    end
32 endmodule
33

```

Рис. 4 – Модуль master

Таким образом, модуль master представляет собой управляющий блок (ведущим устройством), который управляет записью данных на шину bus_MM в соответствии с состоянием автомата fsm_MM. В зависимости от текущего состояния автомата модуль master генерирует адрес, данные и управляющий сигнал записи для шины bus_MM, чтобы передать данные на шину для последующей обработки другими модулями (формирует обращение к двум ведомым устройствам slave_EVEN и slave_ODD).

2.1.3. Модуль slave_EVEN

Создадим модуль ведомого устройства slave_EVEN:

```

lab_MS_SV5 - slave_EVEN.sv

1 `timescale 1 ns / 1 ns
2 module slave_EVEN (
3     bus_MM bus,
4     output bit[7:0] D_even
5 );
6     always_ff @(posedge bus.CLK)
7     if (bus.reset)
8         D_even <= 0;
9     else
10        if ((bus.address ==? 8'b???????) & (bus.write == '1))
11            D_even <= bus.writedata;
12 endmodule
13

```

Рис. 5 – Модуль ведомого устройства slave_EVEN

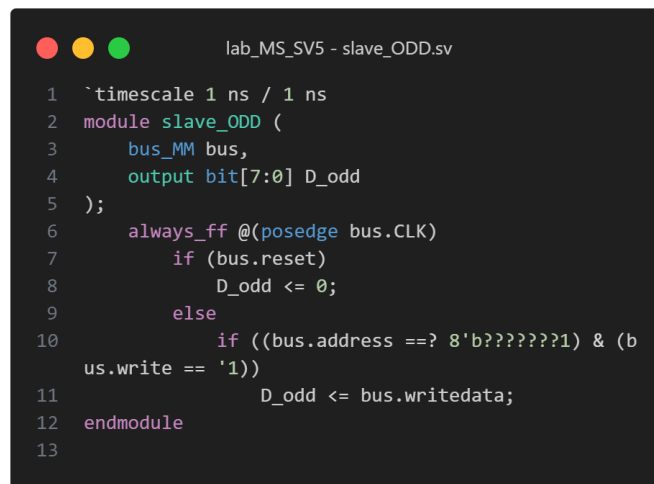
Модуль `slave_EVEN` представляет собой часть системы, которая работает как управляемое устройство и отвечает за обработку данных, переданных по шине `bus_MM`, когда адрес данных указывает на четное значение.

Это устройство ожидает сигнал сброса `reset` и тактового сигнала `CLK`.

- В случае, если модуль не находится в сброшенном состоянии, данные поступают на шину и происходит проверка адреса. Если он чётный, то сигнал активен, данные записываются в выходной порт `D_even`.
- В случае, если модуль сброшен, он ожидает поступления новых сигналов, которые будут обладать чётным адресом

2.1.4. Модуль `slave_ODD`

Создадим аналогичное `slave_EVEN` управляемое устройство `slave_ODD`, которое будет обрабатывать только нечётные сигналы:



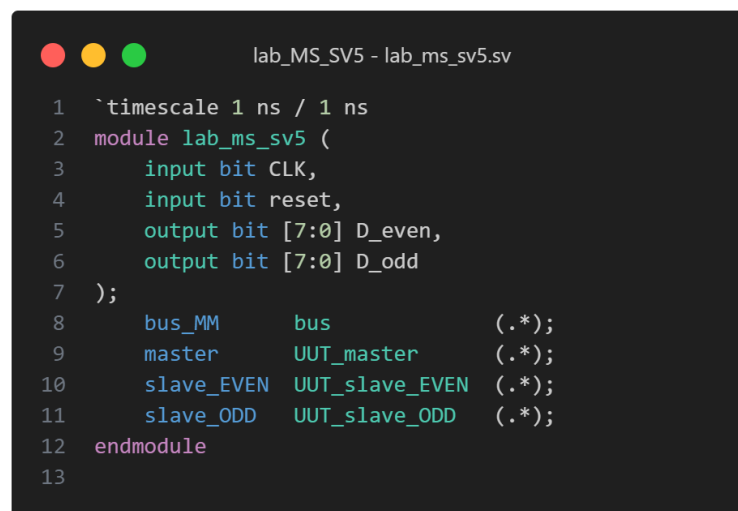
```
lab_MS_SV5 - slave_ODD.sv

1  `timescale 1 ns / 1 ns
2  module slave_ODD (
3      bus_MM bus,
4      output bit[7:0] D_odd
5  );
6      always_ff @(posedge bus.CLK)
7          if (bus.reset)
8              D_odd <= 0;
9          else
10             if ((bus.address ==? 8'b??????1) & (b
11                 us.write == '1))
12                 D_odd <= bus.writedata;
13  endmodule
```

Рис. 6 – Модуль ведомого устройства `slave_ODD`

2.1.5. Модуль `lab_MS_SV5`

Создадим модуль верхнего уровня `lab_MS_SV5`:



```
lab_MS_SV5 - lab_ms_sv5.sv

1  `timescale 1 ns / 1 ns
2  module lab_ms_sv5 (
3      input bit CLK,
4      input bit reset,
5      output bit [7:0] D_even,
6      output bit [7:0] D_odd
7  );
8      bus_MM      bus      (*);
9      master      UUT_master (*);
10     slave_EVEN  UUT_slave_EVEN (*);
11     slave_ODD   UUT_slave_ODD (*);
12 endmodule
13
```

Рис. 7 – Модуль верхнего уровня `lab_MS_SV5`

Модуль lab_MS_SV5 это контроллер, который объединяет и управляет работой: master, slave_EVEN и slave_ODD. Он синхронизирует их работу с CLK и сигналом сброса reset, а также осуществляет передачу данных между ними через шину bus_MM.

2.1.6. Структура устройства

Убедимся в том, что устройство разработано и его структура в RTL Viewer соответствует требованиям задания (Рис. 1):

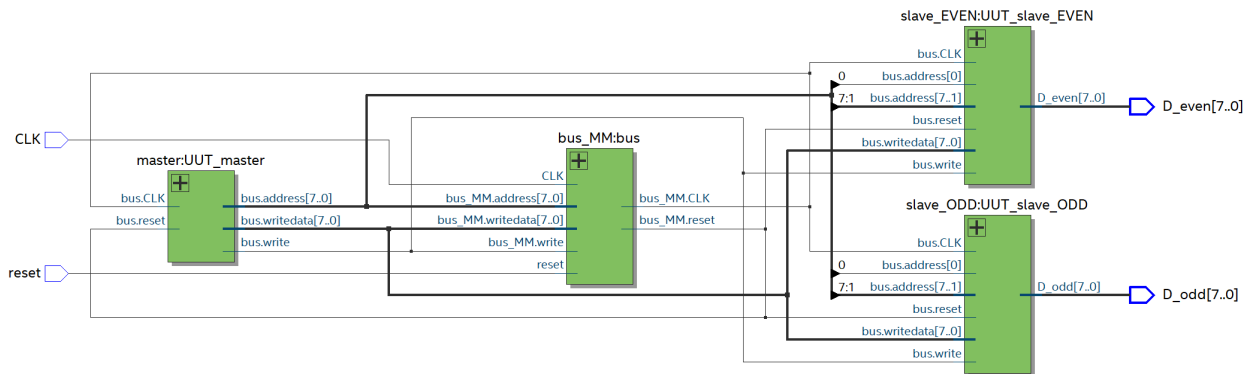


Рис. 8 – Структура разработанного устройства в RTL Viewer

Как видим, полученное устройство имеет структуру, удовлетворяющую требованиям задачи.

2.2. Создание теста первого класса

Напишем тест первого класса для только что созданного устройства:

```
lab_MS_SV5 - tb_lab_MS_SV5.sv

1 `timescale 1 ns / 1 ns
2 module tb_lab_MS_SV5 ();
3     bit CLK;
4     bit reset = '1;
5     bit [7:0] D_even;
6     bit [7:0] D_odd;
7     lab_ms_sv5 UUT (.*);
8     initial
9         forever #5 CLK = ~CLK;
10    initial begin
11        #7;
12        reset = '0;
13        repeat (32) @(negedge CLK);
14        $stop;
15    end
16 endmodule
17
```

Рис. 9 – Тестовый файл первого класса для устройства

Данный тест позволит пройти по всем состояниям конечного автомата. Чтобы проверить корректность работы данного теста, укажем в качестве рассматриваемых значений $op_a = 10$, $op_b = 3$.

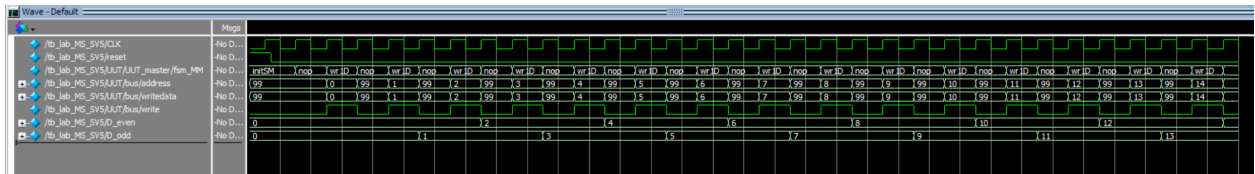


Рис. 10 – Моделирование тестового файла средствами ModelSim (wave)

Проанализируем полученную временную диаграмму:

- Сигнал fsm_MM выводит название состояние конечного автомата в конкретный момент времени.
- Сигналы address и writedata совпадают, так как значение writedata записывается по адресу, который указан в сигнале address.
- Сигнал write работает корректно (запись происходит в состоянии wr1D, а в состоянии por сигнал = 0 и модуль ожидает сигнала или условия, при котором будет необходимо выполнить запись данных).
- Также, заметим, что изменение адреса происходит правильно: он увеличивается после каждого прохода состояния wr1D (т. е. при осуществлении записи) и не увеличивается в состоянии por, т. к. запись не осуществляется.
- Сигналы D_even и D_odd так же работают корректно, т. к. принимают значение 1 только при поступлении чётного и нечётного адресов соответственно.

Как видим, все состояния и сигналы обрабатываются корректно.

2.3. Создания модуля для тестирования на плате

Теперь разработаем модуль для тестирования программы на плате:

```
lab_MS_SV5 - db_lab_MS_SV5.v
1 `timescale 1 ns / 1 ns
2 module db_lab_MS_SV5 (
3     (* altera_attribute = "-name IO_STANDART \"3.3-V LVCMOS\"", chip_pin = "23" *)
4     // "23 for miniDilab-CIV"
5     input bit CLK
6 );
7 bit reset;
8 bit [7:0] D_even;
9 bit [7:0] D_odd;
10 lab_ms_sv5 uut (.*);
11 SP_unit SP_ (
12     .source (reset), // sources.source
13     .source_clk (CLK)
14 );
15 endmodule
16
```

Рис. 11 – Модуль db для тестирования на плате

2.4. Настройка Signal Tap II

Для ввода значений в модуль будем использовать ISSP, там же будем смотреть результат. Дополнительно добавим Signal Tap II, в котором будем получать значения по изменению состояний и получать результат в виде названия состояния и адреса:

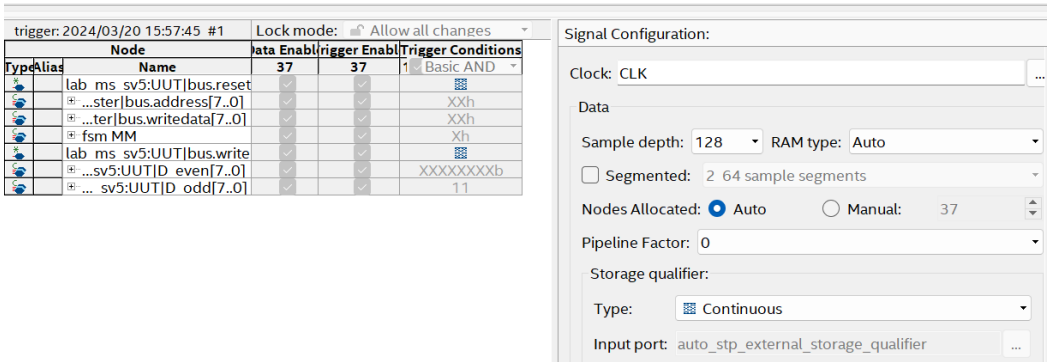


Рис. 12 – Настройка окна Signal Tap II

Составим мнемоническую таблицу, чтобы при симуляции отображался не просто код состояния, а его название:

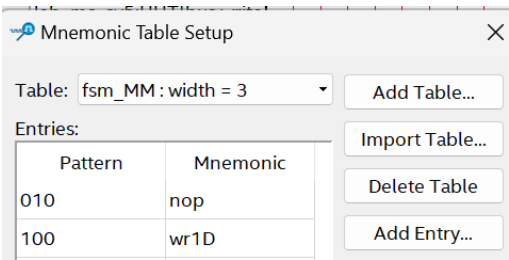


Рис. 13 – Мнемоническая таблица

2.5. Тестирование на плате средствами Signal Tap II

Выполним полную компиляцию. В отчете о компиляции видно, что устройство удовлетворяет временным параметрам.

Slow 1200mV OC Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	455.58 MHz	250.0 MHz	CLK	limit due to minimum period restricti

Рис. 14 – Временные характеристики устройства

Теперь запустим и выполним проверку корректности работы программы на плате. Выполним загрузку разработанного модуля на плату и запустим тестирование:

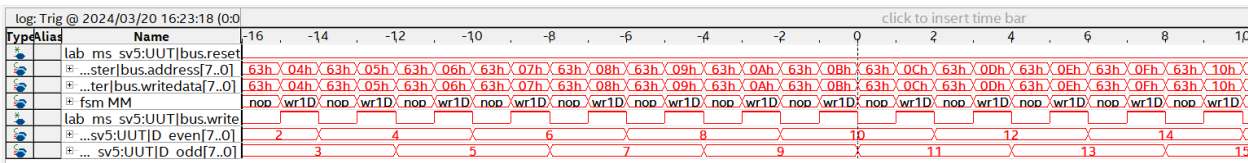


Рис. 15 – Результат SignalTap II

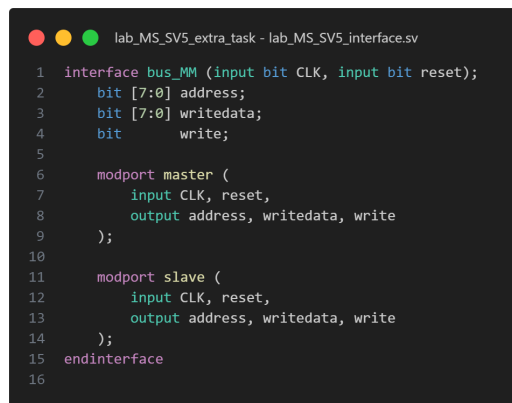
3. Дополнительное задание

3.1. Изменение модулей

В контексте дополнительного задания нам нужно модифицировать устройство из первой части путём добавления `modport`.

3.1.1. Изменение модуля интерфейса

Внесём изменения в описание интерфейса `lab_MS_SV5_interface`, добавив `modport` для master и slave.

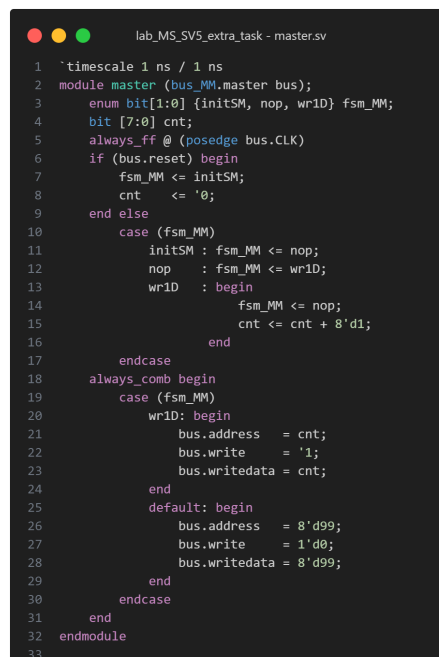


```
lab_MS_SV5_extra_task - lab_MS_SV5_interface.v
1 interface bus_MM (input bit CLK, input bit reset);
2     bit [7:0] address;
3     bit [7:0] writedata;
4     bit      write;
5
6     modport master (
7         input CLK, reset,
8         output address, writedata, write
9     );
10
11     modport slave (
12         input CLK, reset,
13         output address, writedata, write
14     );
15 endinterface
16
```

Рис. 16 – Модифицированный модуль интерфейса

3.1.2. Изменение модуля master

В соответствии с только что модифицированным модулем интерфейса, изменим файл `master.v`:

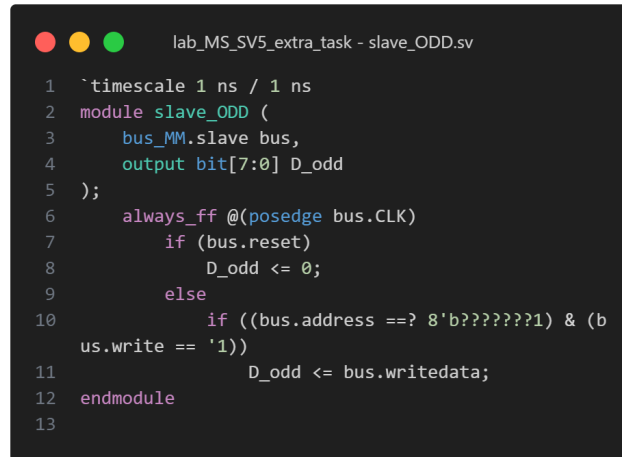


```
lab_MS_SV5_extra_task - master.v
1 `timescale 1 ns / 1 ns
2 module master (bus_MM.master bus);
3     enum bit[1:0] {initSM, nop, wr1D} fsm_MM;
4     bit [7:0] cnt;
5     always_ff @ (posedge bus.CLK)
6     if (bus.reset) begin
7         fsm_MM <= initSM;
8         cnt <= '0;
9     end else
10     case (fsm_MM)
11     initSM : fsm_MM <= nop;
12     nop    : fsm_MM <= wr1D;
13     wr1D   : begin
14         fsm_MM <= nop;
15         cnt <= cnt + 8'd1;
16     end
17     endcase
18     always_comb begin
19     case (fsm_MM)
20     wr1D: begin
21         bus.address = cnt;
22         bus.write   = '1;
23         bus.writedata = cnt;
24     end
25     default: begin
26         bus.address = 8'd99;
27         bus.write   = '1'd0;
28         bus.writedata = 8'd99;
29     end
30     endcase
31     end
32 endmodule
33
```

Рис. 17 – Модифицированный модуль master

3.1.3. Изменение модуля slave_ODD

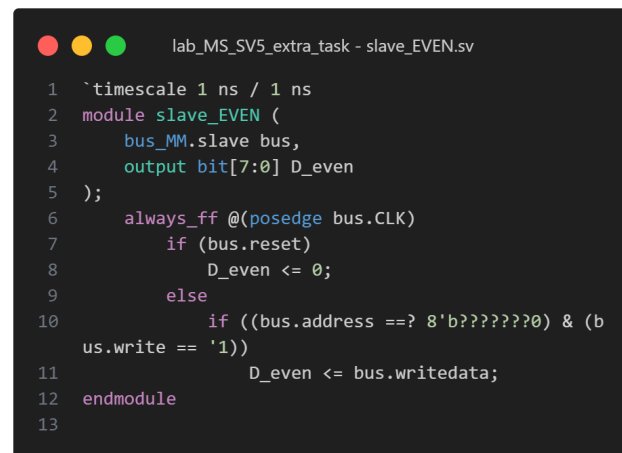
Модифицируем модуль управляемого устройства slave_ODD:



```
lab_MS_SV5_extra_task - slave_ODD.sv
1 `timescale 1 ns / 1 ns
2 module slave_ODD (
3     bus_MM.slave bus,
4     output bit[7:0] D_odd
5 );
6     always_ff @(posedge bus.CLK)
7         if (bus.reset)
8             D_odd <= 0;
9         else
10             if ((bus.address ==? 8'b???????1) & (bus.write == '1))
11                 D_odd <= bus.writedata;
12 endmodule
13
```

Рис. 18 – Модифицированный модуль slave_ODD

3.1.4. Изменение модуля slave_EVEN



```
lab_MS_SV5_extra_task - slave_EVEN.sv
1 `timescale 1 ns / 1 ns
2 module slave_EVEN (
3     bus_MM.slave bus,
4     output bit[7:0] D_even
5 );
6     always_ff @(posedge bus.CLK)
7         if (bus.reset)
8             D_even <= 0;
9         else
10             if ((bus.address ==? 8'b???????0) & (bus.write == '1))
11                 D_even <= bus.writedata;
12 endmodule
13
```

Рис. 19 – Модифицированный модуль slave_EVEN

3.2. Анализ получившегося устройства

Здесь, в отличие от предыдущей части, мы имеем более развёрнутую структуру, т. к. она отражает буферы slave и master, которые были вырезаны из интерфейса.

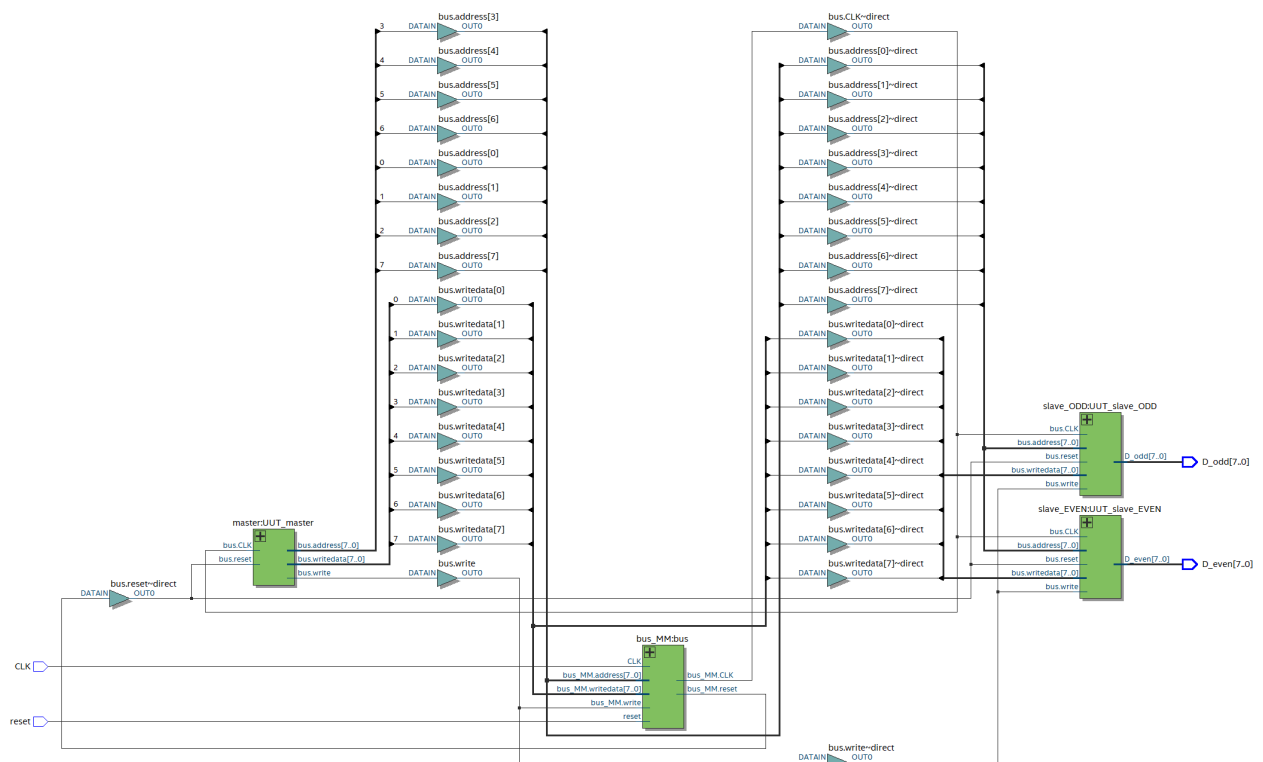


Рис. 20 – Структура модифицированного устройства в RTL Viewer

Тестовый файл остаётся таким же, проверим, что при запуске теста средствами ModelSim полученная временная диаграмма совпадёт с той, что была получена при моделировании тестового модуля первой части (Рис. 10):

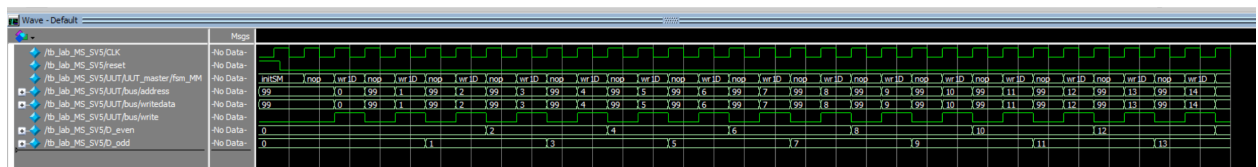


Рис. 21 – Моделирование мод. устройства средствами ModelSim (wave)

Как и ожидалось, работа тестового модуля модифицированного устройства совпала с работой тестового модуля из первой части.

3.3. Тестирование на плате средствами Signal Tap II

Настройка окна SignalTab II выглядит так же, как и в первой части. Единственное, что изменяется, значение по варианту. В данном случае это $6+5=11$ (нечётное число), т. е. в поле сигнала D_odd нужно указать значение = 11. Это число будет определять на каком адресе будет происходить захват.

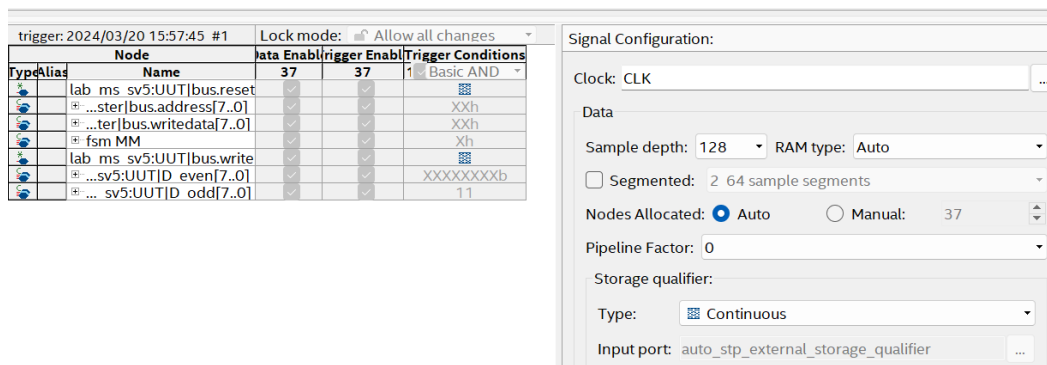


Рис. 22 – Настройка окна Signal Tap II (доп. задание)

Выполним полную компиляцию. В отчете о компиляции видно, что устройство удовлетворяет временным параметрам.

Slow 1200mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	397.61 MHz	250.0 MHz	CLK	limit due ...ggle rate)

Рис. 23 – Временные характеристики устройства

Теперь запустим и выполним проверку корректности работы программы на плате. Выполним загрузку разработанного модуля на плату и запустим тестирование:

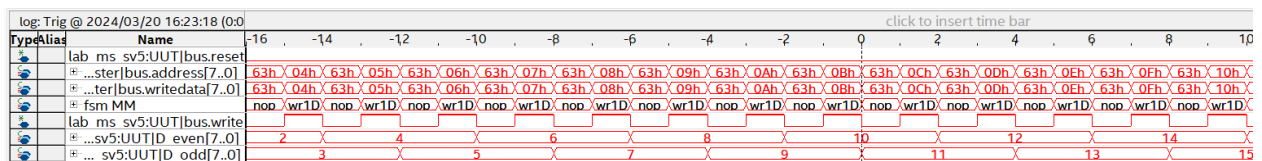


Рис. 24 – Результат SignalTap II

4. Вывод

В результате выполнения лабораторной работы было разработано устройство с использованием модулей `master`, `slave_EVEN` и `slave_ODD`, объединенных через интерфейс `lab_MS_SV5_interface`. Реализовано взаимодействие между модулями посредством шины `bus_MM`, что обеспечивает передачу данных и управляющих сигналов. После внесения модификаций, включая добавление `modport` в интерфейс и соответствующие изменения в модулях, устройство было протестировано на плате с использованием Signal Tap II. В результате была подтверждена корректность работы программы, что подтверждает функциональность и правильность реализации задачи.

Использование интерфейсов в проекте помогло структурировать его и улучшить взаимодействие между компонентами. Это позволило сделать проект более понятным и гибким, облегчая разработку, отладку и поддержку кода. Интерфейсы позволяют явно определить, какие данные и сигналы обмениваются между модулями, что делает код более читаемым и упрощает его использование в других проектах.