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](#_Toc161275562)

[1.1. Цель работы 4](#_Toc161275563)

[2. Ход решения 5](#_Toc161275564)

[2.1. Создание LFSR модуля 5](#_Toc161275565)

[2.2. Создание теста первого класса 6](#_Toc161275566)

[2.3. Создания модуля для тестирования на плате 9](#_Toc161275567)

[2.4. Настройка Signal Tap II 10](#_Toc161275568)

[2.5. Тестирование на плате средствами Signal Tap II 11](#_Toc161275569)

[3. Вывод 13](#_Toc161275570)

**Список иллюстраций**

[Рис. 1 – Пакет для типов данных 5](#_Toc161275571)

[Рис. 2 – Модуль АЛУ 5](#_Toc161275572)

[Рис. 3 – Структура модуля АЛУ в RTL Viewer 6](#_Toc161275573)

[Рис. 4 – Тестовый файл для модуля АЛУ (1) 6](#_Toc161275574)

[Рис. 5 – Тестовый файл для модуля АЛУ (2) 7](#_Toc161275575)

[Рис. 6 – Моделирование тестового файла средствами ModelSim (wave) 7](#_Toc161275576)

[Рис. 7 – Моделирование тестового файла средствами ModelSim (cmd) 7](#_Toc161275577)

[Рис. 8 – Модифицированный модуль для теста первого класса 8](#_Toc161275578)

[Рис. 9 – Общая вывод после симуляции модифицированного tb 8](#_Toc161275579)

[Рис. 10 – Результат SLU\_out > 127 9](#_Toc161275580)

[Рис. 11 – Обработка случая остатка от деления на 0 9](#_Toc161275581)

[Рис. 12 – Обработка случая деления на 0 9](#_Toc161275582)

[Рис. 13 – Модуль db для тестирования на плате 9](#_Toc161275583)

[Рис. 14 – Настройка окна Signal Tap II 10](#_Toc161275584)

[Рис. 15 – Настройка Signal Probe 10](#_Toc161275585)

[Рис. 16 – Мнемоническая таблица 10](#_Toc161275586)

[Рис. 17 – Симуляция в Signal Probe (op\_a = 10, op\_b = 3) 11](#_Toc161275587)

[Рис. 18 – Результат SP (ADD, op\_a = 10, op\_b = 3) 11](#_Toc161275588)

[Рис. 19 – Результат SP (SUB, op\_a = 10, op\_b = 3) 11](#_Toc161275589)

[Рис. 20 – Результат SP (MUL, op\_a = 10, op\_b = 3) 11](#_Toc161275590)

[Рис. 21 – Результат SP (DIV, op\_a = 10, op\_b = 3) 11](#_Toc161275591)

[Рис. 22 – Результат SP (VAR, op\_a = 10, op\_b = 3) 11](#_Toc161275592)

[Рис. 23 – Результат SP (ADD, op\_a = 127, op\_b = 127) 12](#_Toc161275593)

[Рис. 24 – Временные характеристики устройства 12](#_Toc161275594)

# Задание

## Цель работы

Разработать устройство, структура которого будет выглядеть седеющим образом:

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

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.

Выводы устройства (выделены зеленым цветом):

* CLK – вход тактового сигнала.
* reset – синхронный сброс всех устройств
* D\_odd – восьмиразрядный выход
* D\_even – восьмиразрядный выход

## Алгоритм работы

Алгоритм работы разрабатываемого устройства определяется алгоритмами работы его модулей:

* Модуль master:
  + Содержит конечный автомат Мура с тремя состояниями: initSM, nop, 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.

Изображение выглядит как текст, круг, Шрифт, диаграмма

Автоматически созданное описание

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.

## 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.

# Ход решения

## Разработка модулей

### 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:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

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 resest, 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.

### 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).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

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).

### Модуль slave\_EVEN

Создадим модуль ведомого устройства slave\_EVEN:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

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

### Модуль slave\_ODD

Создадим аналогичное slave\_EVEN управляемое устройство slave\_ODD, которое будет обрабатывать только нечётные сигналы:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 6 – Модуль ведомого устройства slave\_ODD

### Модуль lab\_MS\_SV5

Создадим модуль верхнего уровня lab\_MS\_SV5:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

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.

### Структура устройства

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

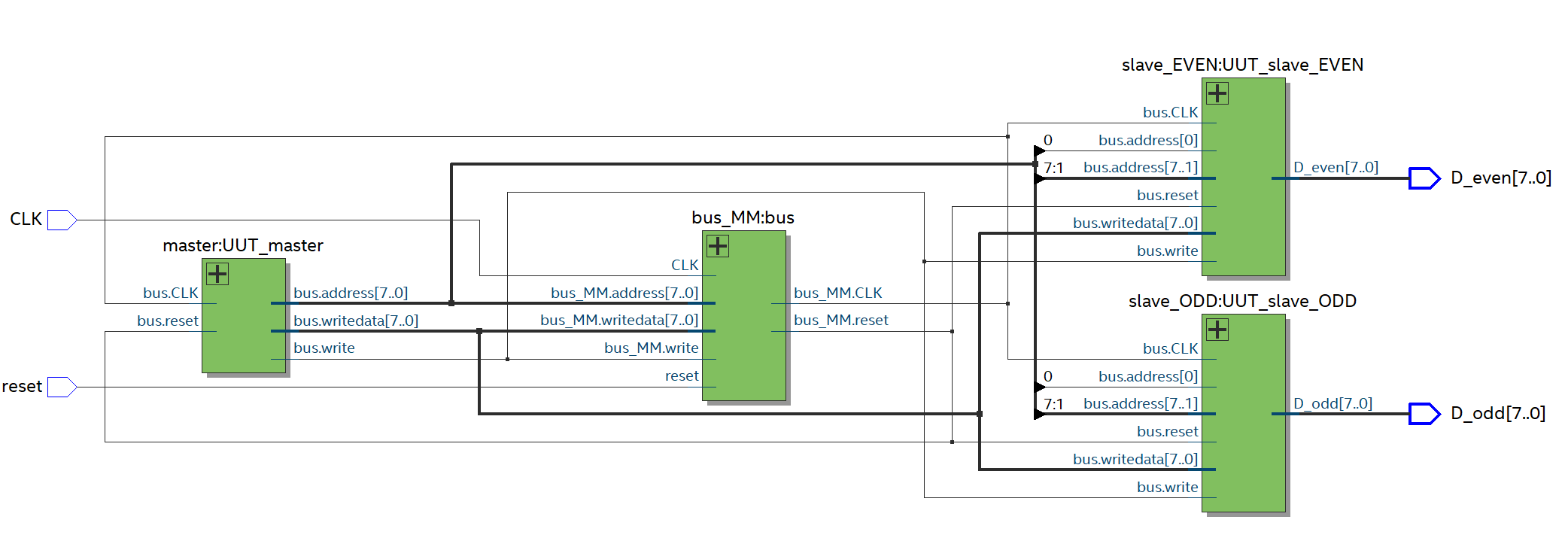


Рис. 8 – Структура разработанного устройства в RTL Viewer

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

## Создание теста первого класса

Напишем тест первого класса для только что созданного устройства:

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рис. 9 – Тестовый файл первого класса для устройства

Данный тест позволит пройтись по всем состояниям конечного автомата. Чтобы проверить корректность работы данного теста, укажем в качестве рассматриваемых значений op\_a = 10, op\_b = 3.

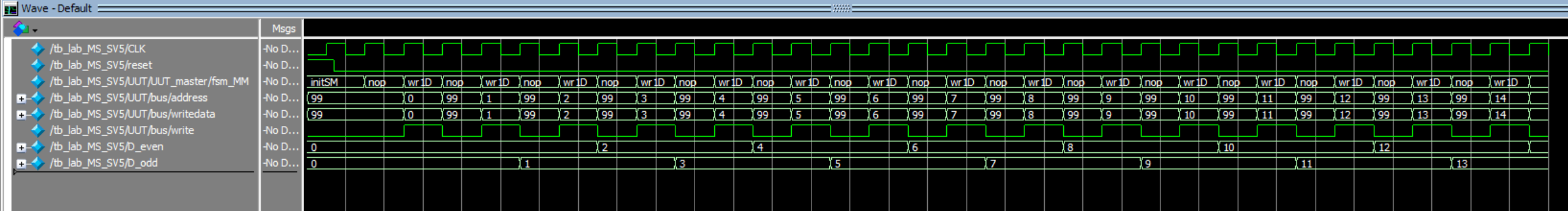


Рис. 10 – Моделирование тестового файла средствами ModelSim (wave)

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 nop 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 nop, 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.
* Остальные граничные случаи также обрабатываются корректно, однако в конкретном тесте сложно найти их выводы из-за обработки всех возможных значений (синтаксис обработки таких случаев схож с приставленным выше примером, поэтому можно просто посмотреть код, чтобы увидеть, что будет выводиться в конкретном случае).

Как видим, все состояния и сигналы обрабатываются корректно.

## Создания модуля для тестирования на плате

Теперь разработаем модуль для тестирования программы на плате:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 17 – Модуль db для тестирования на плате

## 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:

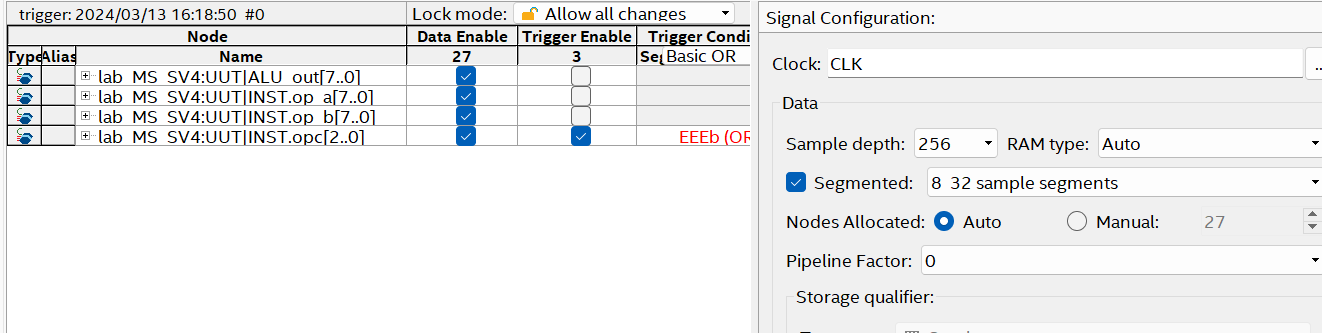


Рис. 18 – Настройка окна Signal Tap II

Перегруппируем сигналы в SP так, чтобы …:

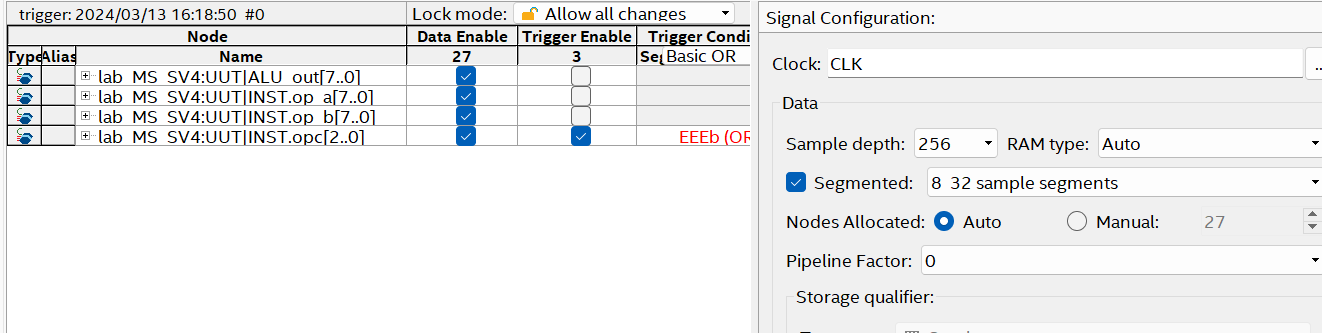


Рис. 19 – Настройка Signal Probe

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

Рис. 20 – Мнемоническая таблица

## Тестирование на плате средствами Signal Tap II

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

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 28 – Временные характеристики устройства

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

Рис. 27 – Результат SP (ADD, op\_a = 127, op\_b = 127)

# Вывод