Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по лабораторной работе Lab\_PD4**

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 5](#_Toc163052034)

[2. Ход работы 6](#_Toc163052035)

[2.1. Создание описания модулей 6](#_Toc163052036)

[Создание модуля my\_master 6](#_Toc163052037)

[Создание модуля my\_slave 9](#_Toc163052038)

[Создание модуля my\_Dslave 10](#_Toc163052039)

[Отличие my\_slave от my\_Dslave 10](#_Toc163052040)

[Принцип работы системы 10](#_Toc163052041)

[2.2. Создание проекта 11](#_Toc163052042)

[Начало работы в PD 11](#_Toc163052043)

[2.3. Настройка сигналов 14](#_Toc163052044)

[Настройка clk 14](#_Toc163052045)

[2.4. Подключение сигналов 14](#_Toc163052046)

[Подключение тактового сигнала 14](#_Toc163052047)

[Подключение сигнала Reset 15](#_Toc163052048)

[2.5. Подключение Avalon-MM интерфейсов 15](#_Toc163052049)

[2.6. Экспорт выводов 17](#_Toc163052050)

[2.7. Анализ системы 17](#_Toc163052051)

[Проверка блока 17](#_Toc163052052)

[Анализ с помощью Schematic 19](#_Toc163052053)

[Генерация системы 19](#_Toc163052054)

[2.8. Подключение файлов к проекту 20](#_Toc163052055)

[3. Тестирование проекта 21](#_Toc163052056)

[3.1. Тестирование средствами ModelSim 21](#_Toc163052057)

[Создание тестового файла 21](#_Toc163052058)

[Симуляция средствами ModelSim 22](#_Toc163052059)

[Изменение тестового файла 23](#_Toc163052060)

[Симуляция средствами ModelSim (все значения) 23](#_Toc163052061)

[3.2. Тестирование средствами Signal Tap II 24](#_Toc163052062)

[Создание файла для отладки 24](#_Toc163052063)

[Настройка Signal Tap II 25](#_Toc163052064)

[Тестирование на плате средствами Signal Tap II 25](#_Toc163052065)

[4. Вывод 27](#_Toc163052066)

**Список иллюстраций**

[Рис. 1 – Структура проекта 5](#_Toc163052067)

[Рис. 2 – Модуль my\_master 6](#_Toc163052068)

[Рис. 3 – Схема конечного автомата 7](#_Toc163052069)

[Рис. 4 – Модуль my\_master в RTL Viewer 8](#_Toc163052070)

[Рис. 5 – Схема конечного автомата в State Machine Viewer 8](#_Toc163052071)

[Рис. 6 – Модуль my\_slave 9](#_Toc163052072)

[Рис. 7 – Модуль my\_slave в RTL Viewer 9](#_Toc163052073)

[Рис. 8 – Модуль my\_Dslave 10](#_Toc163052074)

[Рис. 9 – Модуль my\_slave в RTL Viewer 10](#_Toc163052075)

[Рис. 10 – Исходное окно PD 11](#_Toc163052076)

[Рис. 11 – Закладка Component Type в New Component 11](#_Toc163052077)

[Рис. 12 – Закладка Files в New Component 11](#_Toc163052078)

[Рис. 13 – Выбор файла для симуляции 12](#_Toc163052079)

[Рис. 14 – Закладка Signals & Interfaces в New Component (master) 12](#_Toc163052080)

[Рис. 15 – Проверка успешного добавления мастера в проект 12](#_Toc163052081)

[Рис. 16 – Закладка Signals & Interfaces в New Component (slave) 13](#_Toc163052082)

[Рис. 17 – Закладка Signals & Interfaces в New Component (Dslave) 13](#_Toc163052083)

[Рис. 18 – Проверка успешности добавления модулей 13](#_Toc163052084)

[Рис. 19 – Добавление модулей в систему 14](#_Toc163052085)

[Рис. 20 – Настройка компонента clk 14](#_Toc163052086)

[Рис. 21 – Подключение тактового сигнала (1) 15](#_Toc163052087)

[Рис. 22 – Подключение тактового сигнала (2) 15](#_Toc163052088)

[Рис. 23 – Подключение сигнала Reset 15](#_Toc163052089)

[Рис. 24 – Подключение Avalon-MM интерфейсов 16](#_Toc163052090)

[Рис. 25 – Фиксация адресов 16](#_Toc163052091)

[Рис. 26 – Назначение правильных адресов для компонентов 16](#_Toc163052092)

[Рис. 27 – Проверка корректности адресов 17](#_Toc163052093)

[Рис. 28 – Экспорт выводов 17](#_Toc163052094)

[Рис. 29 – Символ системы 18](#_Toc163052095)

[Рис. 30 – Анализ проблемных подключений 18](#_Toc163052096)

[Рис. 31 – Show System with QSYS Interconnect 18](#_Toc163052097)

[Рис. 32 – Schematic 19](#_Toc163052098)

[Рис. 33 – Предустановки окна Genreration 19](#_Toc163052099)

[Рис. 34 – Проверка успешности генерации HDL 19](#_Toc163052100)

[Рис. 35 – Подключение файлов к проекту 20](#_Toc163052101)

[Рис. 36 – Синтаксис файла lab\_PD4\_top.sv 20](#_Toc163052102)

[Рис. 37 – Схема проекта в RTL Viewer 20](#_Toc163052103)

[Рис. 38 – Тестовый файл tb\_ lab\_PD4\_top.sv 21](#_Toc163052104)

[Рис. 39 – Тестовый файл tb\_lab\_PD2\_top.sv 21](#_Toc163052105)

[Рис. 40 – Моделирование проекта средствами ModelSim 22](#_Toc163052106)

[Рис. 41 – Изменённый тестовый файл 23](#_Toc163052107)

[Рис. 42 – Моделирование проекта средствами ModelSim (все значения) 23](#_Toc163052108)

[Рис. 43 – Файл для отладки модуля верхнего уровня 24](#_Toc163052109)

[Рис. 44 – Схема проекта с добавлением SP\_unit в RTL Viewer 24](#_Toc163052110)

[Рис. 45 – Сигналы логического анализатора 25](#_Toc163052111)

[Рис. 46 – Настройка окна Signal Tap II 25](#_Toc163052112)

[Рис. 47 – Временные характеристики устройства 25](#_Toc163052113)

[Рис. 48 – Настройка окна In-System Sources and Probe Editor 26](#_Toc163052114)

[Рис. 49 – Результат SignalTap II 26](#_Toc163052115)

# Задание

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

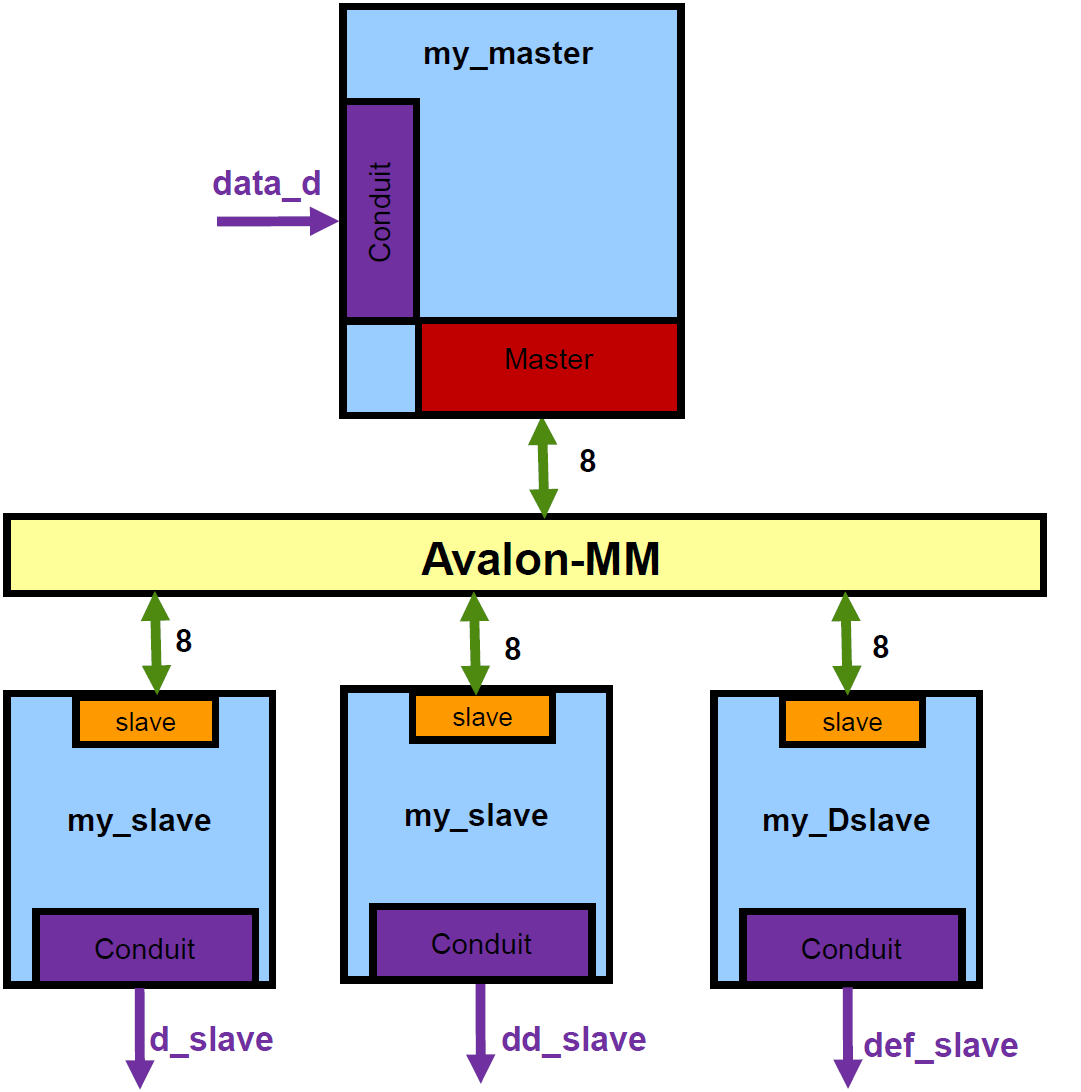


Рис. 1 – Структура проекта

Устройство, которое содержит master и 3 slave: 2 модуля my\_slave и 1 модуль my\_Dslave (default slave)

Master получает некоторые данные через Conduit, через 8-разрядный интерфейс мастер осуществляет адресный доступ к одному из slave’ов, настраивает соответственно slave’s, либо что-то в них записывает, каждый из slave’ов имеет в себе Conduit, который помогает посмотреть на выводе slave’а то, что мы туда записали из мастера.

# Ход работы

## Создание описания модулей

Согласно структуре системы, представленной на Рис. 1, создадим описания модулей my\_master, my\_slave и my\_Dsalve:

### Создание модуля my\_master

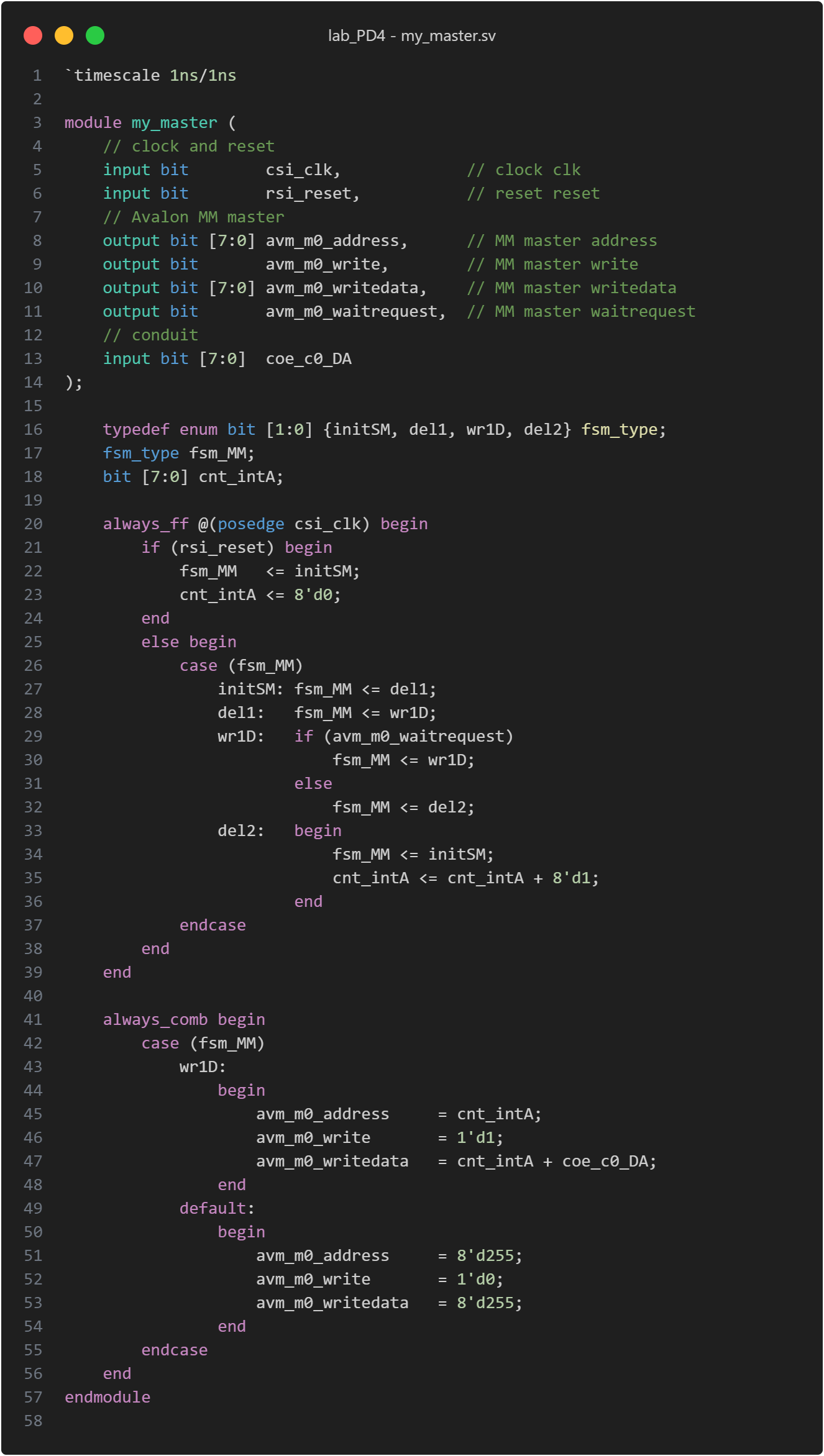


Рис. 2 – Модуль my\_master

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. FSM 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.

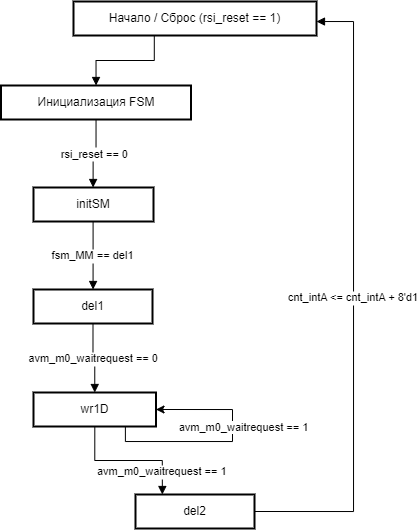


Рис. 3 – Схема конечного автомата

В состоянии wr1D, когда мастер отправляет данные на запись, модуль устанавливает значения выходных сигналов avm\_m0\_address, avm\_m0\_write, avm\_m0\_writedata в соответствии с требуемыми операциями записи.

Пока операция записи выполняется (avm\_m0\_waitrequest == 1), FSM остаётся в состоянии wr1D, ожидая окончания операции записи.

Когда операция записи завершается (когда avm\_m0\_waitrequest == 0), FSM переходит в состояние del2, где инкрементируется счётчик cnt\_intA.

Значение счётчика cnt\_intA выводится через сигнал coe\_c0\_DA для передачи его другим компонентам через интерфейс Conduit.

\*Conduit обеспечивает канал связи между различными компонентами системы, что позволяет им обмениваться данными и взаимодействовать друг с другом.

Таким образом, модуль my\_master обеспечивает правильную передачу данных от мастера к другим устройствам в системе (my\_slave и my\_Dslave).

Схема master средствами RTL Viewer будет выглядеть следующим образом:

Изображение выглядит как линия, диаграмма, снимок экрана, График

Автоматически созданное описание

Рис. 4 – Модуль my\_master в RTL Viewer

Также, откроем State Machine Viewer и убедимся в правильности построенного автомата:

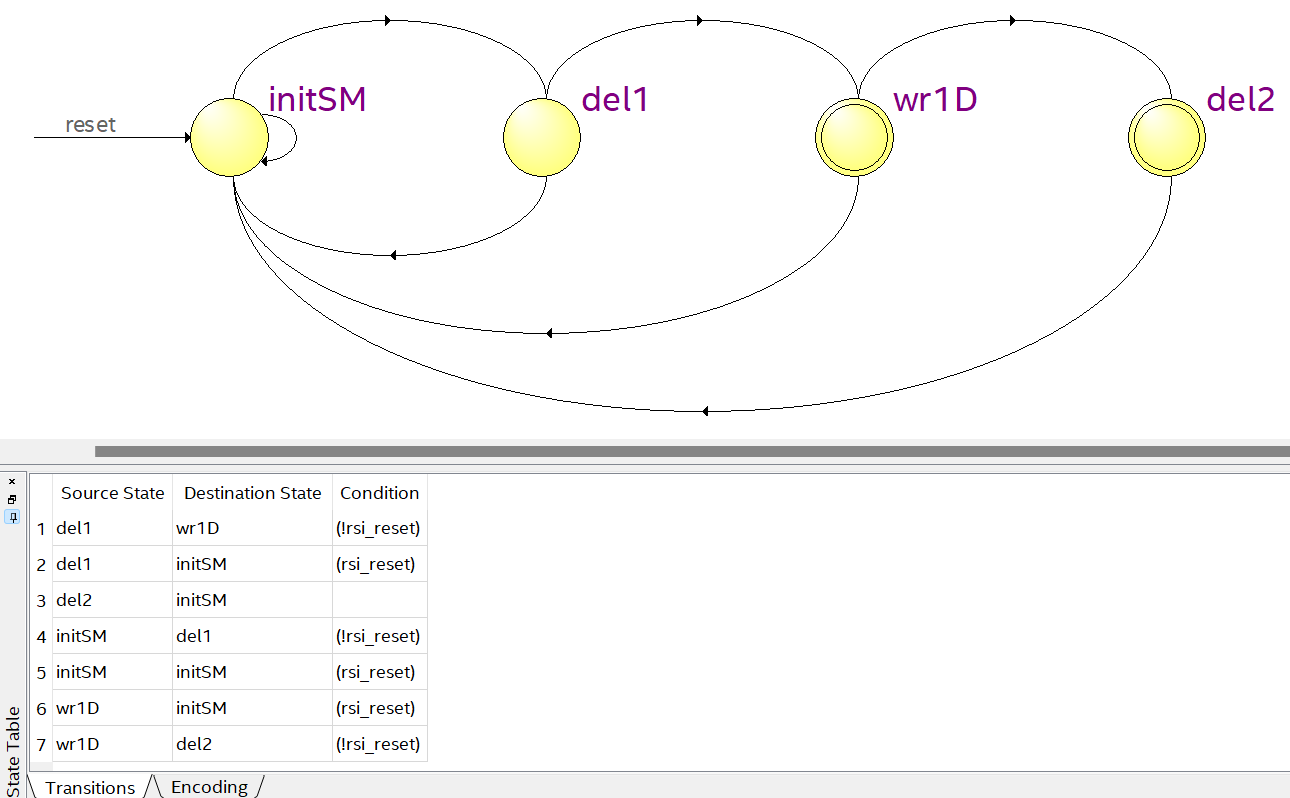


Рис. 5 – Схема конечного автомата в State Machine Viewer

Схема конечного автомата удовлетворяет той, что представлена на Рис. 3.

### Создание модуля my\_slave

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рис. 6 – Модуль my\_slave

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:

Изображение выглядит как линия, диаграмма, текст, снимок экрана

Автоматически созданное описание

Рис. 7 – Модуль my\_slave в RTL Viewer

### Создание модуля my\_Dslave

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рис. 8 – Модуль my\_Dslave

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.

Изображение выглядит как линия, диаграмма, снимок экрана, текст

Автоматически созданное описание

Рис. 9 – Модуль my\_slave в RTL Viewer

### Отличие my\_slave от my\_Dslave

Модули my\_slave и my\_Dslave оба выполняют функцию обработки данных в системе, но есть несколько ключевых различий между ними.

Если обычный slave просто записывает то, что пришло на шину write\_data, то Dslave будет хранить число обращений к нему

### Принцип работы системы

Таким образом, автомат берёт данные с Conduit’а, выдает их на шину master (с 0 по 255 такты цикла), м/у записями (когда не write), формирует признаки того, что он не работает, выдавая 255, когда работает, выдаёт 1.

## Создание проекта

### Начало работы в PD

Откроем PD и сохраним систему:

Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание

Рис. 10 – Исходное окно PD

Через строчку New Component импортируем модули, созданные ранее (my\_master, my\_slave, my\_Dslave):

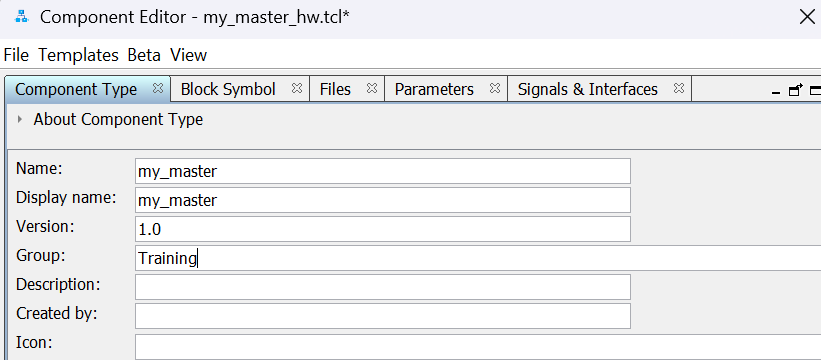


Рис. 11 – Закладка Component Type в New Component

Изображение выглядит как текст, Шрифт, число, линия

Автоматически созданное описание

Рис. 12 – Закладка Files в New Component

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 13 – Выбор файла для симуляции

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рис. 14 – Закладка Signals & Interfaces в New Component (master)

Убедимся в том, что модуль my\_master был успешно добавлен в проект:

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

Автоматически созданное описание

Рис. 15 – Проверка успешного добавления мастера в проект

Теперь проведём аналогичные действия для модулей my\_slave и my\_Dslave:

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

Автоматически созданное описание

Рис. 16 – Закладка Signals & Interfaces в New Component (slave)

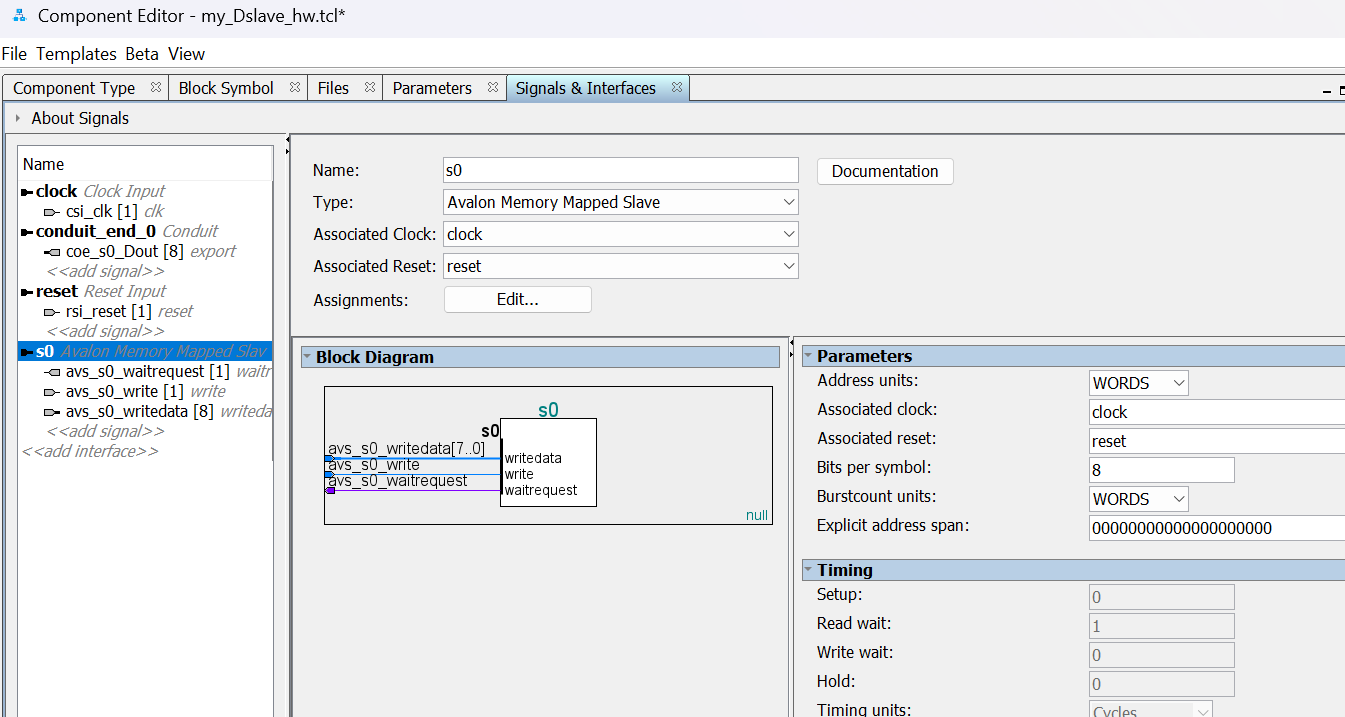


Рис. 17 – Закладка Signals & Interfaces в New Component (Dslave)

Убедимся, что все модули успешно добавились в систему:

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, число

Автоматически созданное описание

Рис. 18 – Проверка успешности добавления модулей

Согласно схеме устройства на Рис. 1 добавим необходимые модули из тех, которые были только что нами созданы и проверим, что система выглядит следующим образом:

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

Автоматически созданное описание

Рис. 19 – Добавление модулей в систему

## Настройка сигналов

### Настройка clk

Зададим значение Reset synchronous edges = Deassert

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, Шрифт

Автоматически созданное описание

Рис. 20 – Настройка компонента clk

## Подключение сигналов

### Подключение тактового сигнала

Выделим интерфейс clk компонента clk\_0, и, открыв его соединения, выберем подключение ко всем тактовым входам:

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

Автоматически созданное описание

Рис. 21 – Подключение тактового сигнала (1)

Переименуем сигналы и выполним Filter → Clock and Reset Interfaces, убедимся, что соединения выполнены корректно:

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

Автоматически созданное описание

Рис. 22 – Подключение тактового сигнала (2)

### Подключение сигнала Reset

Выполним System → Create Global Reset Network и убедимся, что система выглядит корректно, сигнал Reset подключен:

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

Автоматически созданное описание

Рис. 23 – Подключение сигнала Reset

## Подключение Avalon-MM интерфейсов

Выполним Filter → Avalon-MM Interfaces и выберем соединения так, как показано на картинке ниже

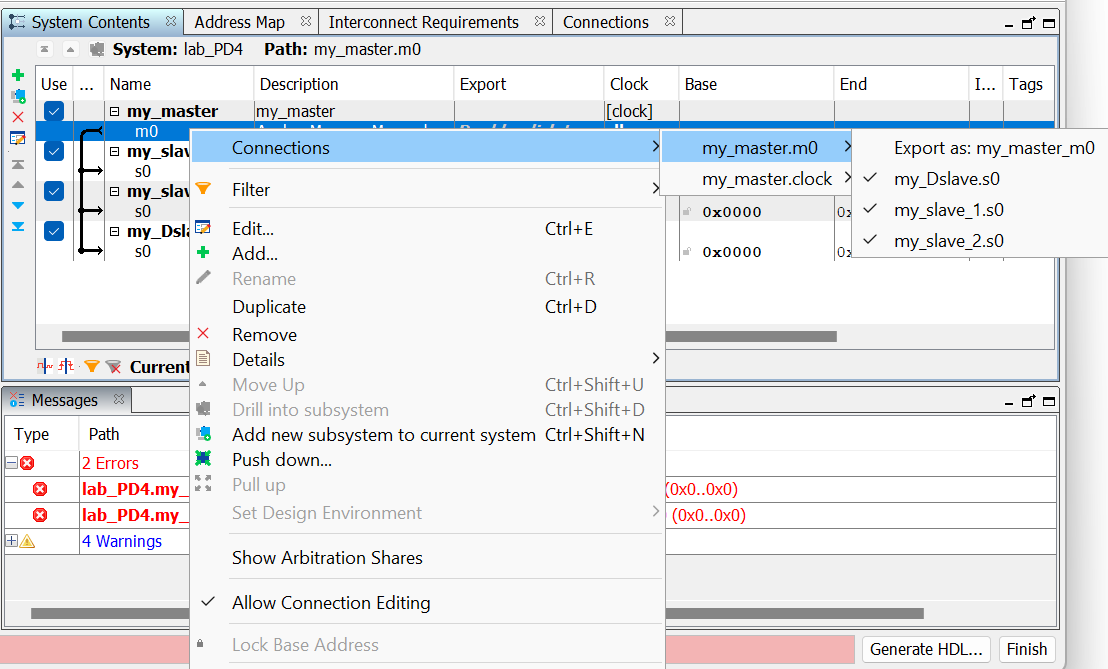


Рис. 24 – Подключение Avalon-MM интерфейсов

В столбце Default Slave (добавлен через ПКМ), поставив галочку, тем самым выполнив chek box:



Рис. 25 – Фиксация адресов

Компоненту my\_slave\_1.s0 назначим базовый адрес = 1, а компоненту my\_slave\_1.s0 – базовый адрес = 2. После этого зафиксируем адреса (замочек), система будет выглядеть следующим образом:

Изображение выглядит как текст, число, линия, Шрифт

Автоматически созданное описание

Рис. 26 – Назначение правильных адресов для компонентов

Вкладка Address Map будет выглядеть следующим образом:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 27 – Проверка корректности адресов

Запомним базовые адреса:

* my\_slave\_1.s0 = 1,
* my\_slave\_2.s0 = 2.

\*Позже он будут указываться при настройке модулей my\_slave\_1 и my\_slave\_2 соответственно

## Экспорт выводов

Проведём экспорт выводов путём задания имён для выделенных модулей в столбце Export (значения data\_d, d\_slave, did\_slave и def\_slave в соответствующем столбце выделенных строк):

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

Автоматически созданное описание

Рис. 28 – Экспорт выводов

Проверим, что окно сообщений не содержит ошибок и пре

## Анализ системы

### Проверка блока

Выполним View → Block Symbol и убедимся в том, что символ системы построен правильно:

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

Автоматически созданное описание

Рис. 29 – Символ системы

Выполним View → Clock domains Beta, выберем режим отображения Reset. Заметим, что проблемных подключений не выявлено:

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рис. 30 – Анализ проблемных подключений

Выполним команду System → Show System with PD Interconnect (Show System with QSYS Interconnect). Проверим, был добавлен только модуль mm\_interconnect\_0.

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

Рис. 31 – Show System with QSYS Interconnect

### Анализ с помощью Schematic

Выполним View → Schematic, в качестве фильтра введём in и убедимся в том, что система синхронизации и каналы ST системы подключены верно:

Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание

Рис. 32 – Schematic

### Генерация системы

Выполним PD → Generate HDL и укажем следующие предустановки для генерации:

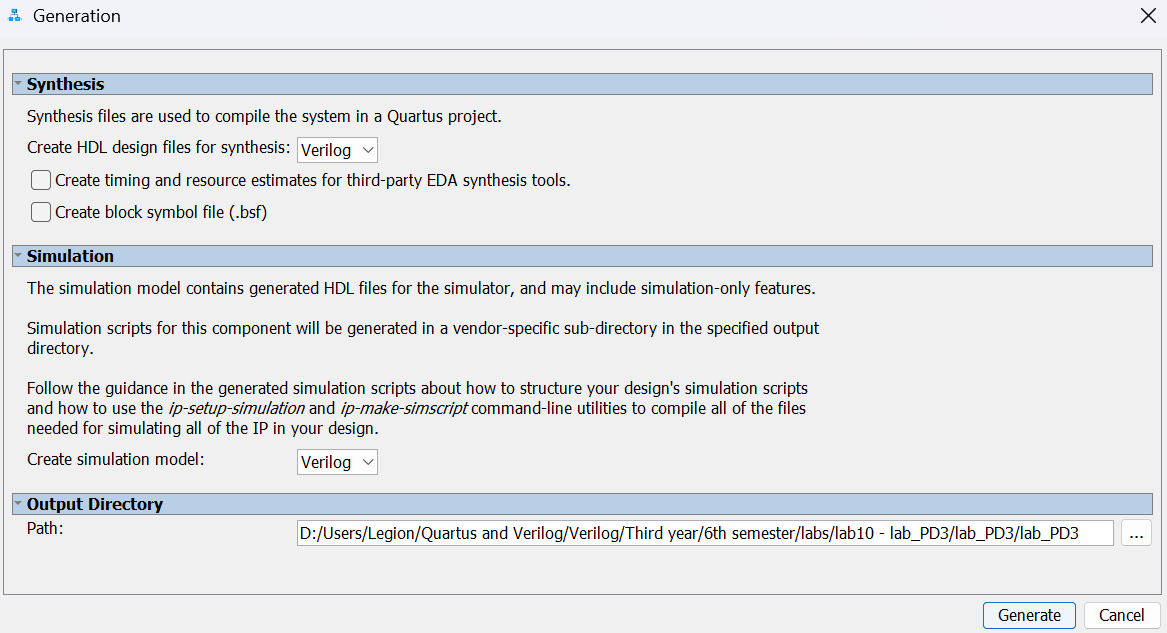


Рис. 33 – Предустановки окна Genreration

Удостоверимся в том, что генерация прошла успешно:

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

Автоматически созданное описание

Рис. 34 – Проверка успешности генерации HDL

## Подключение файлов к проекту

Подключим файлы к проекту в Quartus

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

Автоматически созданное описание

Рис. 35 – Подключение файлов к проекту

Синтаксис файла lab\_PD4\_top.sv:

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

Автоматически созданное описание

Рис. 36 – Синтаксис файла lab\_PD4\_top.sv

Выполним анализ и синтез проекта средствами QP и убедимся в правильности схемы средствами RTL Viewer:

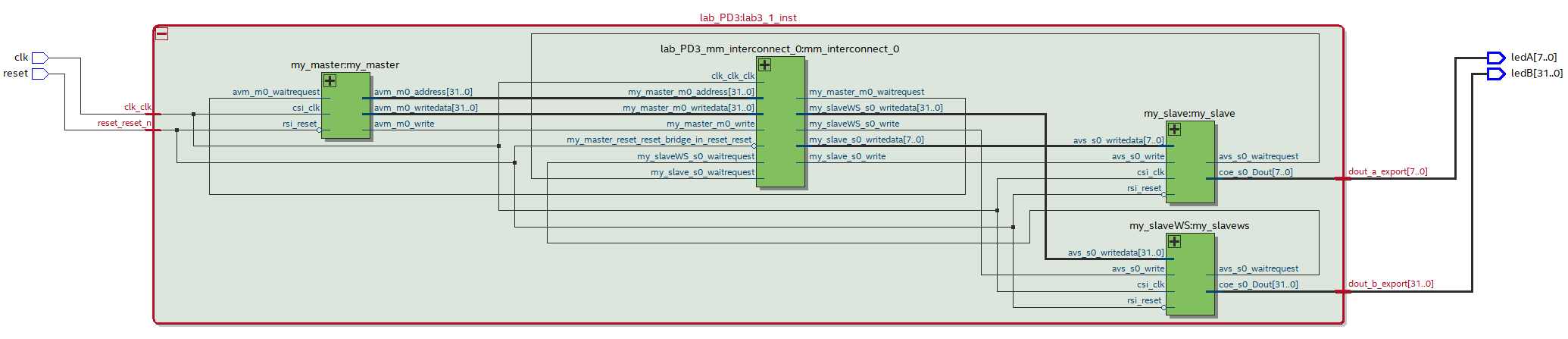


Рис. 37 – Схема проекта в RTL Viewer

Можем увидеть, что полученная в RTL Viewer схема совпадает с той, что была задана по условию (в зелёном блоке отображается тот фрагмент системы, который был создан средствами PD).

# Тестирование проекта

## Тестирование средствами ModelSim

### Создание тестового файла

Добавим тест первого класса для созданного проекта:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рис. 38 – Тестовый файл tb\_ lab\_PD4\_top.sv

Укажем созданный файл в качестве основного тестового файла, который будет выполняться при симуляции средствами ModelSim:

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

Автоматически созданное описание

Рис. 39 – Тестовый файл tb\_lab\_PD2\_top.sv

### Симуляция средствами ModelSim

Выполним компиляцию проекта средствами ModelSim. Для этого запустим wave.do файл:

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

Автоматически созданное описание

Рис. 40 – Моделирование проекта средствами ModelSim

Изначально сигнал reset был равен 0, сделали его = 1, устройство начало работать.

**master** выставляет адрес 0 и пытается записать туда 6 (это данные, которые находятся на шине writedata).

1. Адрес 0:

**slave\_1** и **slave\_2** не смогли распознать сигнал (мы это поняли так как в соответствующей им строке write значение сигнала по прежнему = 0) => сигнал теперь принадлежит (будет записан в) **Dslave** (в этот slave всегда происходит запись, когда остальные не смогли распознать сигналы, т. к. данные передаются прямо с шины, перед этим проходя через slave\_1 и slave\_2). Заметим, что после того, как произошла запись Dout принимает значение = 1. Это связано с тем, что нам не интересно, что записывается в **Dslave**, поэтому на выход просто поставлен счётчик, который увеличивается каждый раз, когда мы что-то выводит на этот slave.

1. Адрес 1:

Адрес 1 был распознан **slave\_1** (write = 1) => сигнал теперь принадлежит **slave\_1**. При этом до **slave\_2** и **Dslave** уже этот сигнал не дойдёт (сигнал идёт в порядке **slave\_1 → slave\_2 → Dslave** и, если распознаётся каким-то из сигналов, то дальше не идёт).

1. Адрес 2:

Адрес 2 не был распознан **slave\_1**, но был распознан **slave\_2** (write = 1), происходит запись в **slave\_2**

1. Адрес 3:

Адрес 3, как и первый, не был распознан **slave\_1** и **slave\_2**, происходит запись в **Dslave** значения счётчика + 1.

1. Оставшиеся от 255 значения, которые не принадлежат, ни **slave\_1**, ни **slave\_2**, отправляются на **Dslave**.

\*За тем, какие данные куда и когда записались удобно следить последним 4 сигналам блока Esported (верхний блок перед my\_master).

### Изменение тестового файла

Изменим тест таким образом, чтобы он обеспечивал генерацию числа тактовых сигналов, необходимых для формирования master’ом шине Avalon-MM 260 сигналов, для этого изменим тестовый файл следующим образом (на 14-й строке вместо 9 поставим 260):

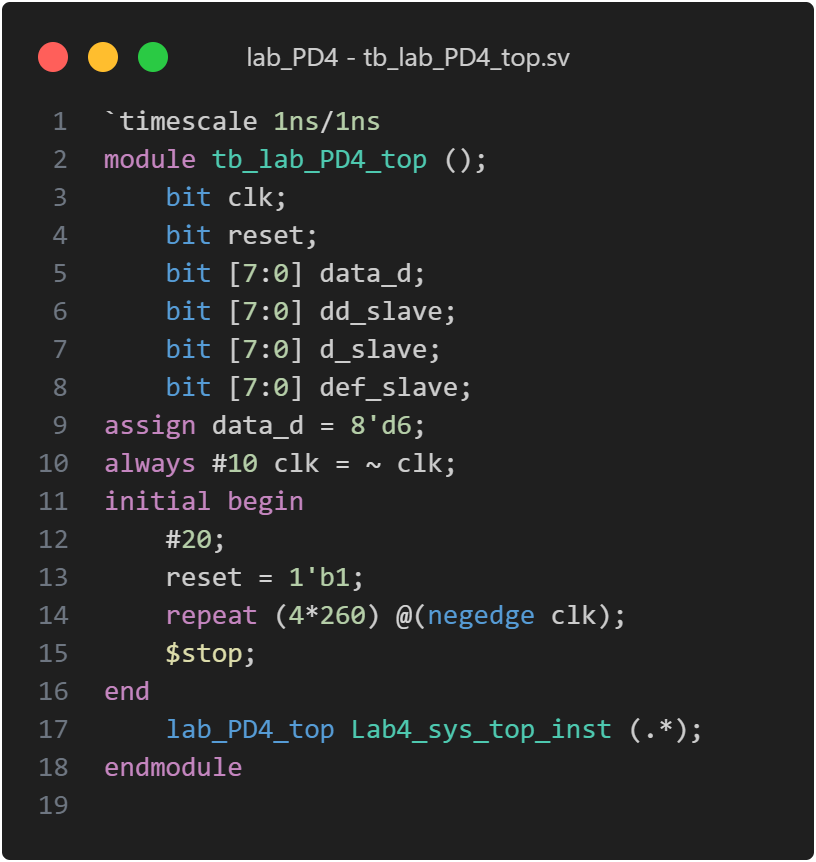


Рис. 41 – Изменённый тестовый файл

### Симуляция средствами ModelSim (все значения)

Получившаяся wave будет во много раз больше, т. к. в ней отображается больше значений, чтобы было наглядно рассмотрим только последнюю её часть, которая представлена на ниже:

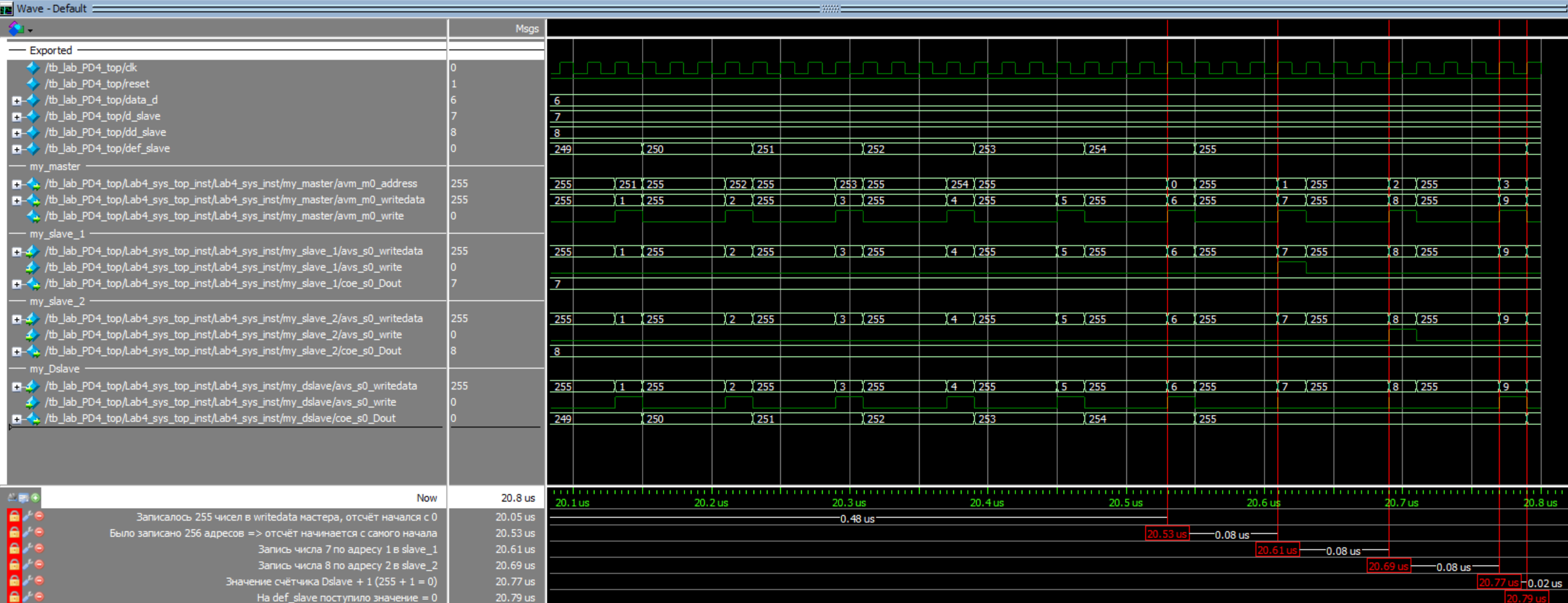


Рис. 42 – Моделирование проекта средствами ModelSim (все значения)

Заметим, что временная диаграмма работает корректно. После записи 255 чисел в writedata мастера, значения пошли снова от 0 до 255. К моменту, когда значение стало равно 6, адрес сбросился и тоже начал идти с 0. Таким образом, цикл завершился и дальше происходит запись чисел аналогично тому, как показано на Рис. 40. Таким образом, число 7 по адресу 1 записалось в slave\_1, число 8 по адресу 2 записалось в slave\_2 и т. д.

Заметим, что за всё это время к модулю Dslave было 256 + 1 обращений = 257. Здесь 256 — это предпоследнее значение в coe\_s0\_Dout, которое увеличивалось на 1 при каждом новом обращении к Dslave, а + 1, т. к. в самом конце coe\_s0\_Dout обнулился из-за ещё одного обращения.

## Тестирование средствами Signal Tap II

### Создание файла для отладки

Создадим файл db\_lab\_PD4\_top.sv для отладки модуля lab\_PD4\_top:

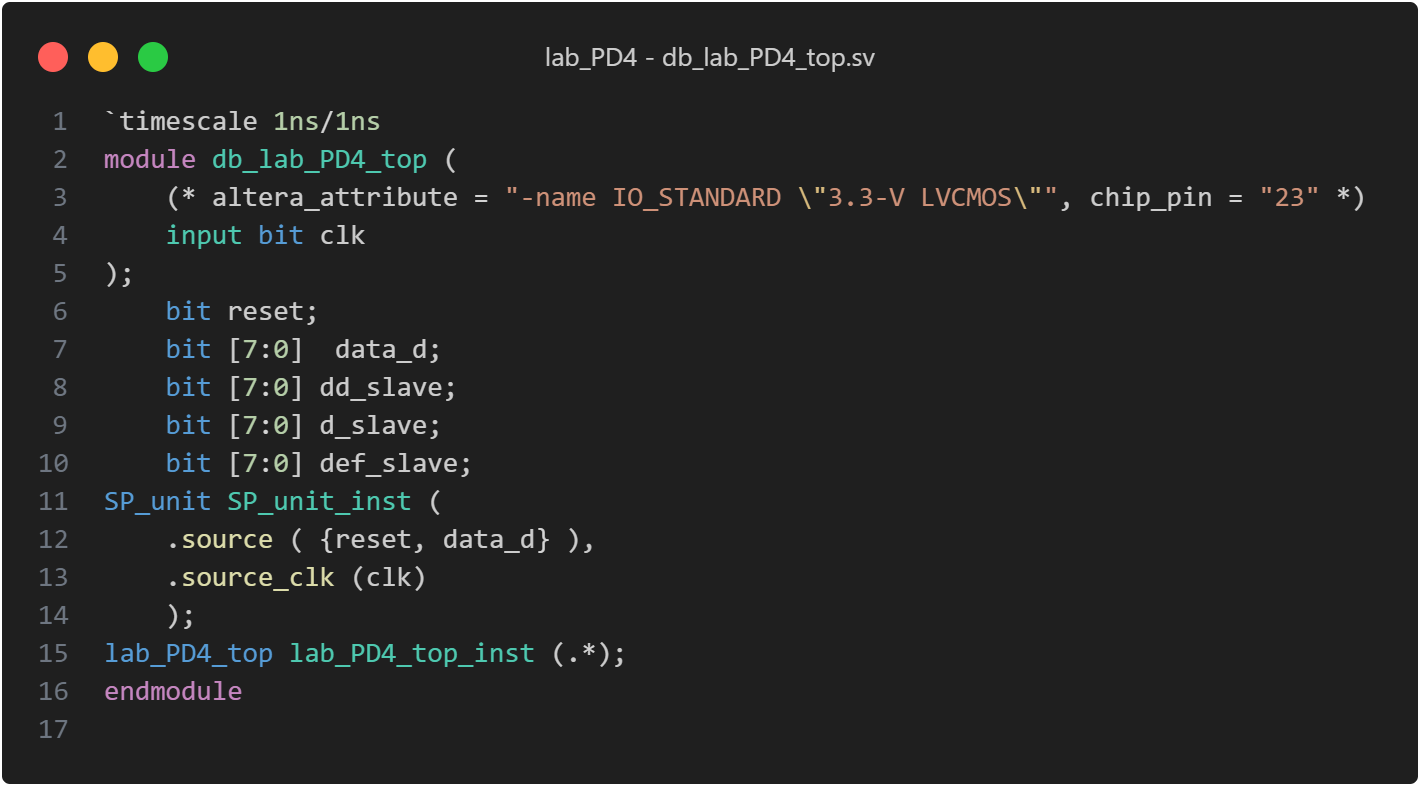


Рис. 43 – Файл для отладки модуля верхнего уровня

Создадим модуль ISSPE, укажем файл db\_lab\_PD4\_top.sv файлом верхнего уровня и убедимся в том, что схема, получаемая в результате компиляции, будет верной:

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рис. 44 – Схема проекта с добавлением SP\_unit в RTL Viewer

Как видно из схемы SP\_unit добавлен корректно.

### Настройка Signal Tap II

Выберем сигналы для логического анализатора:

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

Автоматически созданное описание

Рис. 45 – Сигналы логического анализатора

Настроим окно Signal Tap II так, как показано на рисунке ниже:

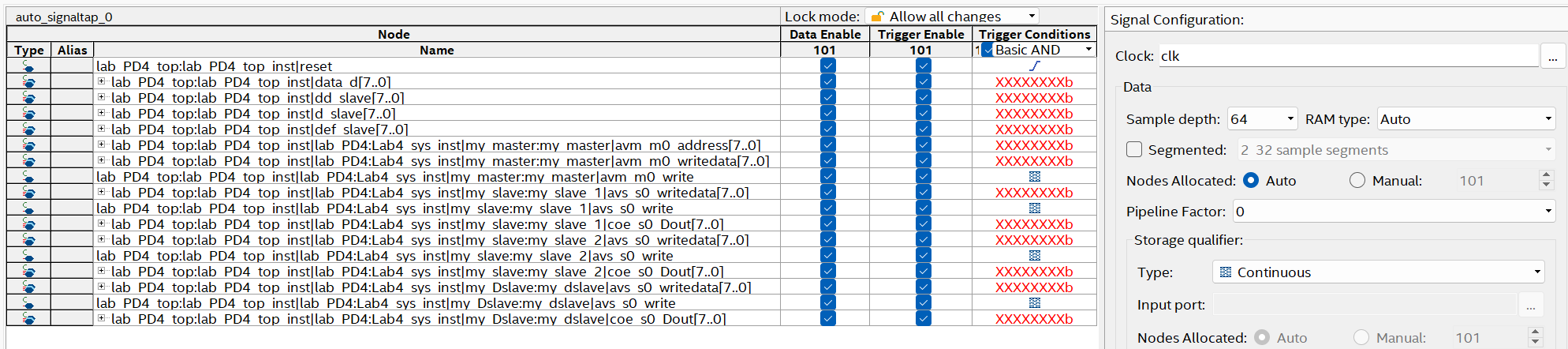


Рис. 46 – Настройка окна Signal Tap II

### 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. 47 – Временные характеристики устройства

Выполним настройку окна In-System Sources and Probe Editor:

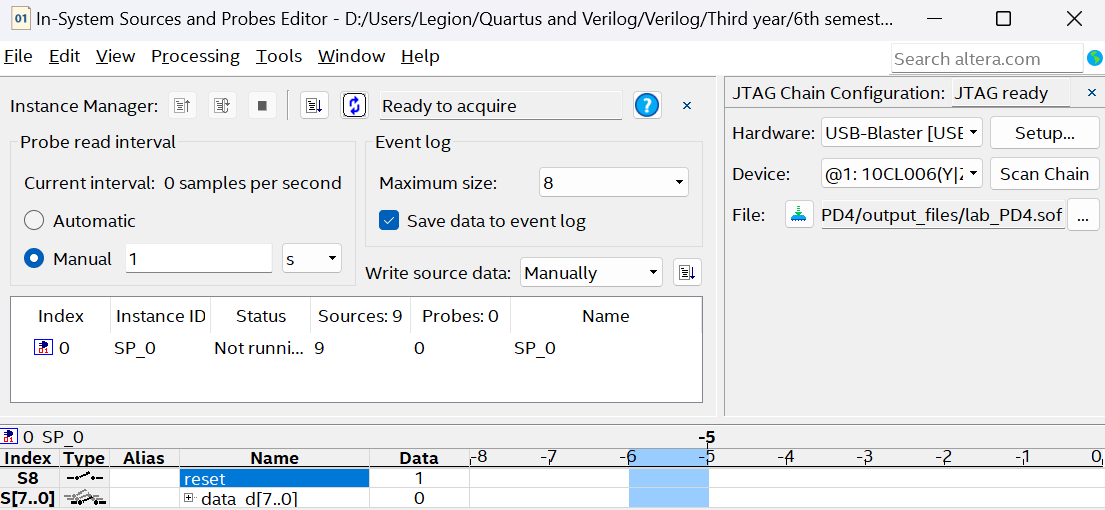


Рис. 48 – Настройка окна In-System Sources and Probe Editor

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

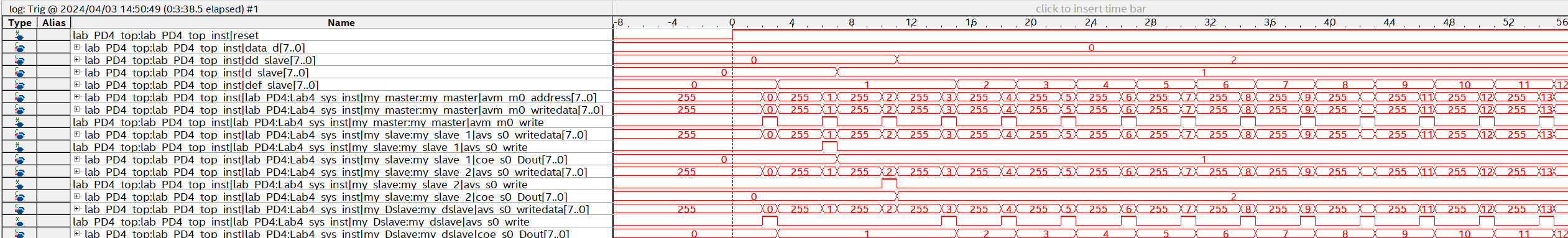


Рис. 49 – Результат SignalTap II

Полученная временная диаграмма совпадает с той, что была получена в ходе тестирования проекта средствами ModelSim (Рис. 40 – Рис. 42). Данные поступают и передаются на приёмник корректно.

# Вывод

В ходе лабораторной работы было создано устройство, включающее мастер Avalon MM и три слейва - два модуля my\_slave и один модуль my\_Dslave. Мастер осуществляет адресный доступ к слейвам и управляет передачей данных. Слейвы принимают данные от мастера и, в зависимости от адреса, либо сохраняют их, либо передают дальше.

Преимущество такого устройства заключается в его гибкости и расширяемости. Мастер может направлять данные на различные устройства в зависимости от их адреса, а слейвы могут быть легко добавлены или удалены по мере необходимости. Это делает систему универсальной и адаптивной к различным потребностям.

Такие устройства широко используются во встраиваемых системах, где требуется управление и обмен данными между различными компонентами. Примерами могут служить устройства для управления промышленным оборудованием, автомобильными системами, сетевыми устройствами и другими приложениями, где требуется эффективная передача данных и управление периферийными устройствами.