Министерство образования и науки Российской Федерации
ТОМСКИЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

КАФЕДРА ТЕЛЕКОММУНИКАЦИЙ И ОСНОВ РАДИОТЕХНИКИ (ТОР)

УТВЕРЖДАЮ
Зав. кафедрой ТОР
А.Я. Демидов

Методические указания
к лабораторным работам

по дисциплине «**Программирование логических интегральных схем**»

РАЗРАБОТЧИКИ:
_______ Д.А. Покаместов,
___________ Я.В. Крюков,
_________ Ж.Т. Эрдынеев,
аспиранты каф. ТОР

ТОМСК – 2014

Оглавление

Введение..1

Лабораторная работа №1...2

Лабораторная работа №2...8

Лабораторная работа №3...10

Лабораторная работа №4...14

Лабораторная работа №5...23

Лабораторная работа №6...30

Лабораторная работа №7...39

Введение

Данный лабораторный практикум знакомит студентов с основами проектирования цифровых устройств на базе ПЛИС типа FPGA. В качестве основного метода рассмотрен язык описания Verilog. Проектирование производится в пакете Quartus II, с использованием основных его компонент. В качестве средства верификации и симуляции проектов используется ModelSim.

Программируемые логические интегральные схемы - класс устройств, способных менять логику работы в соответствии с прошивкой. Для программирования используются программаторы и отладочные среды, позволяющие задать структуру устройства с помощью описания на специальных языках: Verilog, VHDL, AHDL.

Результаты проделанной работы необходимо внести в отчет и представить на защиту.

Указания к отчету по лабораторной работе.

Отчет должен содержать следующие разделы:

1. Название, цель работы.
2. Исходные данные.
3. Описание выполненных лабораторных заданий, с выводами по каждому заданию

Лабораторная работа №1.

«Создание проекта в Quartus ll. Логические схемы»

Цель работы: Знакомство с пакетом Quartus II. Создание проекта с помощью графического описания схем.

Лабораторное задание

Изучить пакет Quartus II. Реализовать с помощью графического программирования логические примитивы Освоить методику симуляции проектов.

Порядок выполнения работы

Создание проекта

1) Откройте Quartus ll
2) Создайте новый проект: File/new/New Quartus II Project. Откроется окно New Project Wizard. Укажите расположение проекта (используйте личную папку, созданную на рабочем столе) и название проекта.
3) Необходимо выбрать используемую схему ПЛИС Altera. Для этого два раза щелкните по выбранной ПЛИС в поле Hierarchy(рисунок 1).

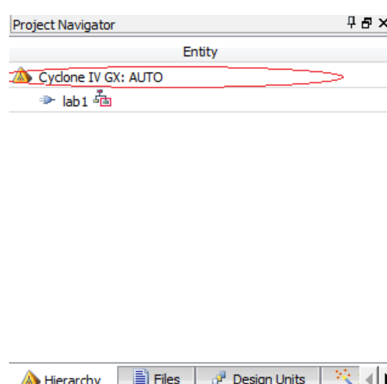


Рисунок 1- Поле навигации проекта

4)Выберите устройство EP4CE22F17C6 из семейства Cyclone IV E. Эта ПЛИС используется в отладочной плате Deo-Nano, которую на которой вы будете выполнять лабораторные работы.

5)К проекту необходимо подключить файл описания вашей схемы. Для этого выберите File/new/Block Deagram/Schematic File. В Квартусе откроется поле, в котором вы должны собрать схемы «исключающее ИЛИ» на основе компонент AND2(и), OR2(или) и NOT(не). Компоненты можно получить, нажав на кнопку Symbol Tool на панели инструментов и выбрав в библиотеке primitives/logic, см. рисунок 2.

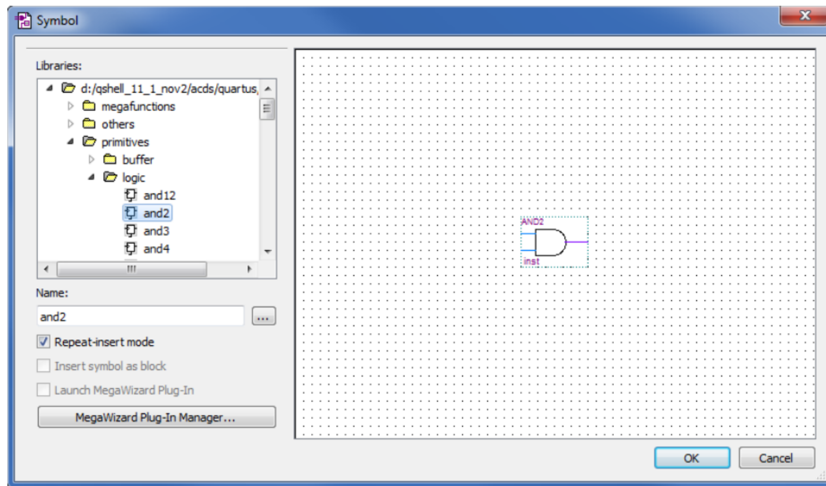


Рисунок 2 – Выбор логических элементов

6) Соберите схему «исключающее ИЛИ» , изображенную на рисунке 3

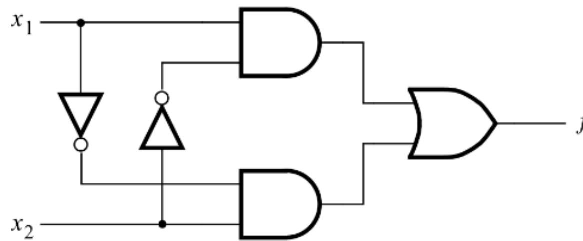


Рисунок 3 Исключающее ИЛИ

Ко входам и выходам схемы подключите элементы Input и Output, расположенные на панели инструментов правее кнопке Symbol Tool (кнопка Pin Tool).

7) Откомпилируйте проект (кнопка Start Compilation)

8) Привяжите проект к реальной плате. Нажмите кнопку Pin Planner на панели инструментов Quartus. Присвойте входам значения PIN_M1 и PIN_T8(первый и второй переключатели), выходу значение Pin_A15(первый светодиод). Обозначения остальных светодиодов, кнопок и переключателей можно найти в файле DE0_Nano_User_Manual_v1.7.pdf

9)Прошейте плату. Для этого войдите в меню Programmer ,нажав на кнопку Programmer на панели инструментов Quartus. Далее нажмите кнопку Start. Протестируйте ваш проект, переключая рычажки на плате. Отразите результаты в отчете

Симуляция проекта в Quartus ll

Проект можно просимулировать в Quartus, прежде чем прошивать ПЛИС. Как правило, симуляция-обязательный этап при разработке любых проектов. Quartus позволяет симулировать несложные проекты, для более обьемных используется отдельная программа ModelSim.

1)Создайте файл симуляции. Для этого выберите File/new/University Program VWF.

2)Вставте в симулятор временные диаграммы ваших входов и выходов. Для этого нажмите Edit/Insert/Insert Node or Bas. В появившемся окне выберите Node Finder и нажмите кнопку List. Выберете все пины.

3) Установите значения входных сигналов. Для этого выделите временной интервал, нажмите правую кнопку мышки и установите необходимое значение в разделе Value.

4)Нажмите кнопку Run Timing Simulation, сохраните результаты симуляции.

5)Нажмите кнопку Run Functional Simulation, сохраните результаты симуляции. Объясните разницу результатов с предыдущим пунктом.

Создание полного сумматора

На рисунке 3 изображена схема полусумматора

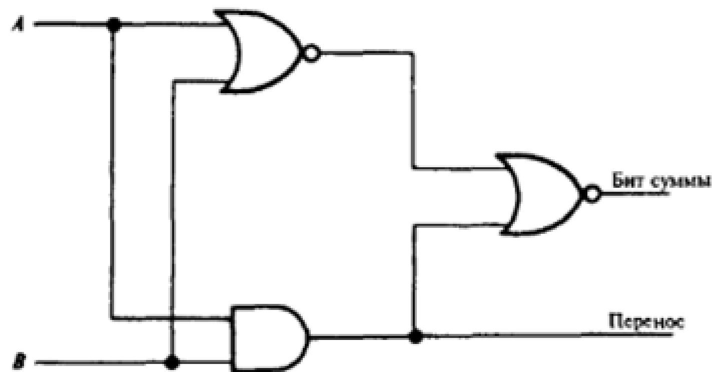


Рисунок 3- Полусумматор

На рисунке 4 на основе полусумматора собран полный сумматор.

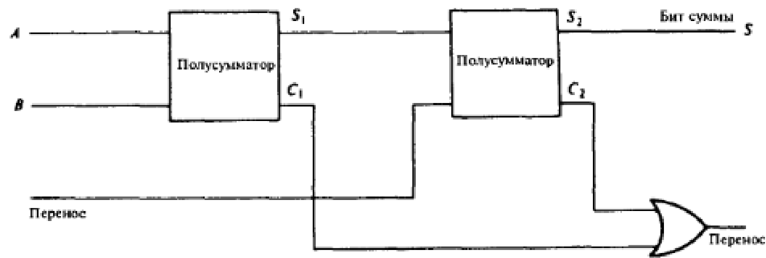


Рисунок 4 – Полный сумматор

Соберите на элементах Qurtus полный сумматор, и проведите симуляцию согласно разделу 2 . Результаты симуляции вставьте в отчет.

Создание полного сумматора на языке Verilog

Создайте новый проект (назовите проект как full_adder) и вставьте новый файл «Verilog HDL» и скопируйте код, который находится ниже (сохраните файл как half_adder).

Схема полусумматора, представленная на рисунке 3 можно представить на языке Verilog в следующем виде:

```
module half_adder (
input a,
input b,
output s,
output c
);

assign s = !((!(a || b)) || (a && b));
assign c = a && b;

endmodule
```

Вставьте новый файл «Verilog HDL» и скопируйте код, который находится ниже (сохраните файл как full_adder).

Схема полного сумматора, представленная на рисунке 4 можно представить на языке Verilog в следующем виде.

```
module full_adder(
input a,
```

```
input b,
input c_in,
output s,
output c_out
);

wire s1;
wire c1;
wire c2;

half_adder my_adder_1 (
                    .a(a),
                    .b(b),
                    .c(c1),
                    .s(s1)
                    );

half_adder my_adder_2 (
                    .a(s1),
                    .b(c_in),
                    .c(c2),
                    .s(s)
                    );

assign c_out = c1 || c2;

endmodule
```

Сделайте симуляцию во времени. Сравните результаты

Лабораторная работа №2.

«Счетчики и делители частоты»

Цель работы: Знакомство с основами языка Verilog, типовой реализацией счетчика и делителя частоты

Счетчик

Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Иными словами счетчик считает, сколько раз произошло событие, на которое настроен счетчик.

Лабораторное задание

Реализовать модуль счетчика тактовых импульсов на плате DE0 NANO. Реализовать делитель частоты. Вывести сигнал с пониженной частотой на светодиоды платы.

Порядок выполнения работы

Создание счетчика

1)Создайте проект Quartus II

2) Добавьте в проект файл Verilog HDL, в котором реализуйте синхронный счетчик тактовых импульсов (clk).

Сам счетчик описывает следующий код:

```
always @(posedge clk)
    if(reset)
        counter <= 0;
    else if(en)
        counter <= counter + 1'b1;
```

Этот код нужно понимать следующим образом. При каждом положительном (posedge) фронте тактового импульса(clk), если нет сигнала сброса(reset), и есть разрешающий сигнал(en), к значению счетчика добавляется 1 бит. Если есть сигнал сброса(reset)

Вам необходимо добавит этот код в модуль счетчика. Самостоятельно необходимо обозначит входы и выходы модуля согласно правилам синтаксиса языка Verilog. Разрядность счетчика (регистр counter) 8.

3) Выполните симуляцию проекта с помощью инструмента University Program VWF. Посмотрите, как изменяются младшие разряды регистра counter(для этого разверните список разрядов).

Делитель частоты

Обратите внимание на изменение значений в разрядах регистра счетчика counter. Младший разряд(counter[0]) изменяет свое значение по положительному фронту тактового сигнала clk, при этом его частота в два раза меньше частоты сигнала clk. Частота изменения значения разряда counter[1] в два раза меньше частоты counter[0] и т.д.

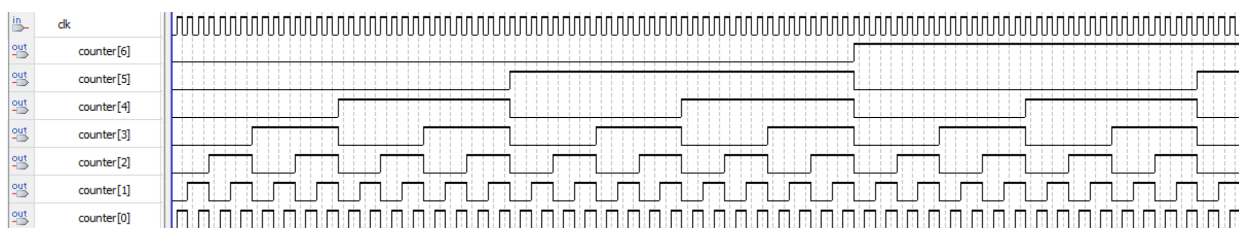


Рисунок 1- Временные диаграммы разрядов регистра counter.

Применяя это свойство счетчиков, может быть получено устройство, называющееся делителем частоты. На выходе этого устройства получается тактовый сигнал с частотой в N раз меньшей, чем на входе. Плата Deo-Nano имеет встроенный генератор тактовых импульсов с частотой 50 МГц. Таким образом, с помощью делителя частоты можно получить частоты, в 2^N меньшие.

Создание делителя частоты.

1) Создайте новый проект Quartus II.

2) Модифицируйте код, написанный вами в пункте 1, чтобы реализовать делитель частоты. Необходимая частота – спросить преподавателя.

3) Реализовать 8-разрядный счетчик с частотой заданной преподавателем. Создайте 8-разрядную шину LEDS, привязанную к разрядам 8-разрядного счетчика.

4) Привяжите проект к вашей ПЛИС(EP4CE22F17C6 из семейства Cyclone IV E) с помощью инструмента PIN_Planner. Соедините ваш тактовый сигнал (clk) с PIN_R8.

5) Выведите 8-разрядную шину LEDS на светодиоды платы Deo-nano. Светодиоды на ПЛИС EP4CE22F17C6 имеют пины PIN_A15, PIN_A13, PIN_B13, PIN_A11, PIN_D1, PIN_F3, PIN_B1, PIN_L3.

6)Назначьте входам reset и en пины PIN_M1 и PIN_T8 (первый и второй переключатели)

7) Прошейте проект в плату. Включите переключатель, соответствующий входу en и выключите соответствующий входу reset.

8)Убедитесь, что светодиоды моргают с необходимой частотой.

По результатам работы напишите отчет, с результатами симуляции из первого пункта и листингом файлов Verilog.

Лабораторная работа №3.

«Реализация широтно-импульсной модуляции на ПЛИС»

Цель работы: Знакомство с широтно-импульсной модуляцией (ШИМ) и реализация алгоритма ШИМ на языке Verilog в пакете Quartus II. Реализация алгоритма на ПЛИС.

Широтно – импульсная модуляция.

Широтно-импульсная модуляция применяется в технике, как способ преобразования переменного напряжения в постоянное, с изменением его среднего значения. Управление средним значением напряжения происходит путем изменения скважности импульсов. Скважность импульса (S) – это отношение периода импульса к его длительности (рис. 1). Скважность описывается формулой:

$$S = \frac{T}{\tau},$$

где T – период импульса и τ – длительность импульса.

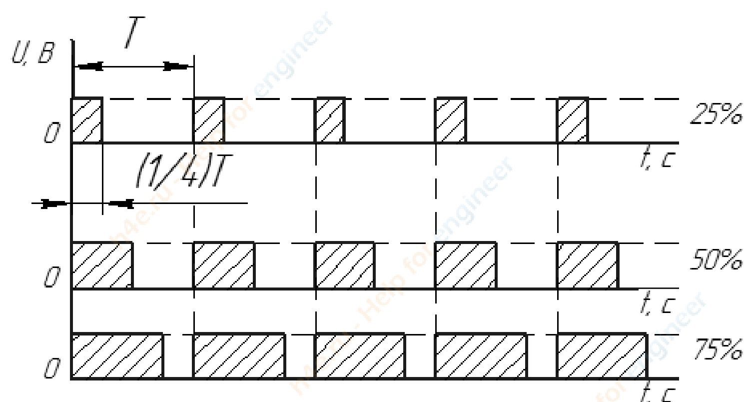


Рисунок 1 – Скважность импульсов

На рис. 1 изображены импульсы, которые возникают с определенной периодичностью. Рассмотрим первый случай. Длительность импульса равна ¼ периода T, значит скважность равна 4. Аналогично выполнение для остальных случаев.

С помощью ШИМ можно управлять яркостью свечения диодов на ПЛИС. Для этого на светодиод нужно подать сигнал в виде последовательности импульсов с ШИМ. Соотношение между скважностью импульсов и яркостью диода изображено на рис. 2.

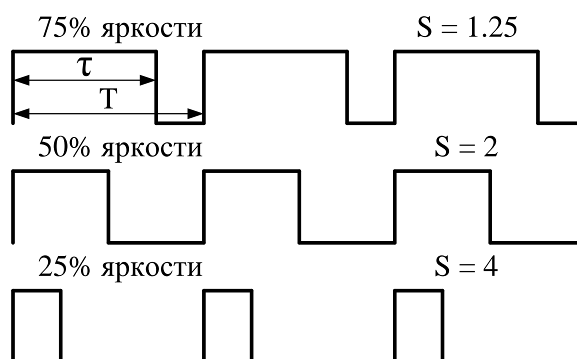


Рисунок 2 – Соотношение между скважностью сигнала и яркостью диода

Лабораторное задание

Написать на языке Verilog алгоритм, который позволит управлять яркостью свечения диодов, расположенных на ПЛИС. Яркость диода должна изменяться от крайнего наименьшего значения (диод не горит) до крайнего наибольшего значения (максимальная яркость) и наоборот. Период времени между крайними положениями должен быть равен одной секунде.

Порядок выполнения работы

1) Создать новый проект Quartus II.
2) Обозначить вход генератора тактовых импульсов (ГТИ) (clk) и выход светодиодов (СД) (wire [7:0] LED).
3) Привяжите проект к ПЛИС (EP4CE22F17C6 из семейства CycloneIVE) с помощью инструмента PIN_Planner. Привяжите переменные ГТИ и СД вашего проекта к портам (табл. 1) ПЛИС, т.е. поставьте в соответствие конкретной переменной конкретную железку.

Таблица 1 – Порты ПЛИС

Назнач.	ГТИ	СД 1	СД 2	СД 3	СД 4	СД 5	СД 6	СД 7	СД 8
Порт	PIN_R8	PIN_A15	PIN_A13	PIN_B13	PIN_A11	PIN_D1	PIN_F3	PIN_B1	PIN_L3

4) Обозначить регистры (диоды (LED_REG [???]), счетчик (counter [???]), ШИМ (PWM_width [6:0]), изменение ШИМ (PWM_add [5:0])).
5) Написать алгоритм счетчика.

Нужно учесть, что частота ГТИ = 50 МГц. В связи с этим нужно корректно рассчитать условие и разрядность счетчика, при котором будет происходить обнуление.

6) Применить алгоритм широтно - импульсной модуляции.

```
PWM_width <= PWM_width [5:0] + PWM_add;
        if (counter[???]==1)
                PWM_add <= ~counter[5:0];
        else
                PWM_add <= counter[5:0];
```

Просимулируйте алгоритм в Quartus, посмотрите как изменяется управляющий импульс, содержащийся в 6-ом разряде регистра PWM_width. Этот импульс является ШИМ-импульсом.

7) Уменьшите частоту ШИМ, для этого регистру PWM_add присвойте регистры счетчика counter с 25 по 20. Назначьте первой лампочки управляющий импульс ШИМ:

   LED_REG[???] <= PWM_width[6];

   Прошейте получившийся код в кристалл ПЛИС. Обьясните, как управляющий импульс ШИМ (PWM_width[6] влияет на яркость светодиода.

8) Поэкспериментируйте с регистром изменения ШИМ (PWM_add), поставте вместо него регистр единиц(6’b111111), регистр 6’b101010. Проделайте с обоими

вариантами пункты 6 и 7. Результаты симуляции отразите в отчете, обратите внимание на яркость светодиодов при разном значении регистра изменения ШИМ, объясните получившийся результат.

Содержание отчета

В отчете необходимо отразить:

- цель работы;
- привести эпюры ШИМ с разной скважности (скважность задает преподаватель);
- объяснить выбор разрядности счетчика;
- кратко объяснить работу алгоритма ШИМ.

Лабораторная работа №4

«Машина конечных состояний»

Цель работы: Изучение алгоритма работы машины с конечным числом состояний на языке Verilog. Реализация алгоритма на основе известной диаграммы состояний.

Машина конечных состояний

Машина конечных состояний или конечный автомат - это математическая абстракция, используемая при проектировании алгоритмов. Говоря простым языком, машина с конечным числом состояний умеет считывать последовательности входных данных. Когда она считывает входной сигнал, то переключается в новое состояние. Куда именно переключится, получив данный сигнал, - заложено в её текущем состоянии.

Представим устройство, которое читает длинную бумажную ленту. На каждом дюйме этой ленты напечатана буква – *a* и *b*.

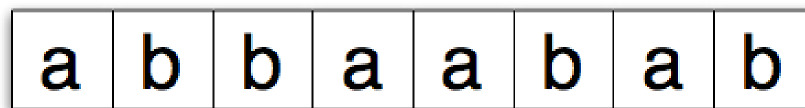


Рисунок 1 – Бумажная лента с символами *a* и *b*

как только устройство считывает букву, оно меняет своё состояние. Вот очень простой граф переходов для такой машины:

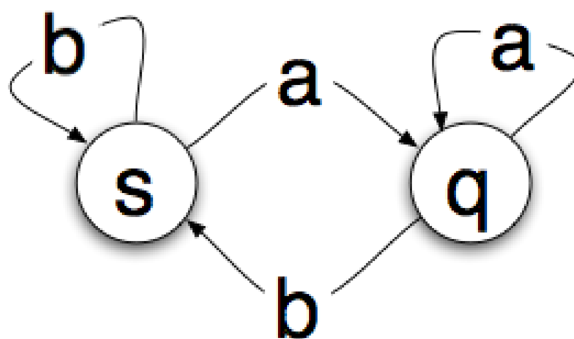


Рисунок 2 – Диаграмма переходов состояний.

Кружки – это состояния, в которых машина может быть. Стрелочки – переходы между ними. Так что, если вы находитесь в состоянии *s* и считываете *a*, то вам необходимо перейти в состояние *q*. А если *b*, то просто остаётесь на месте.

Итак, если изначально мы находимся с состоянии *s* и начинаем читать ленту из первого рисунка слева направо, то сначала будет прочитана *a*, и мы переместимся в состояние *q*, затем *b* — вернёмся обратно в *s*. Следующая *b* оставит нас на месте, а *a* вновь переместит в *q*.

Оказывается, если пропустить ленту с буквами через FSM, то по её итоговому состоянию можно сделать некоторые выводы о последовательности букв. Для приведённого выше простого конечного автомата финальное состояние *s* означает, что лента закончилась буквой *b*. Если же мы закончили в состоянии *q*, то последней на ленте была буква *a*.

Рассмотрим пример реализации конечного автомата на языке Verilog. Предположим, что стоит задача реализовать преобразователь уровня в импульс (Level-to-Pulse), который генерирует одиночный импульс каждый раз, когда вход из «0» переходит в «1». Другими словами, это синхронный обнаружитель нарастающего фронта (рисунок 3).

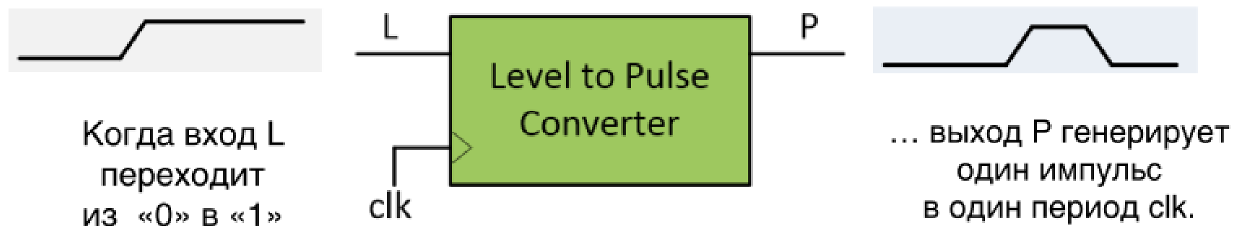


Рисунок 3 – Принцип работы преобразователя Level-to-Pulse

Диаграмма состояний для данного алгоритма будет выглядеть следующим образом:

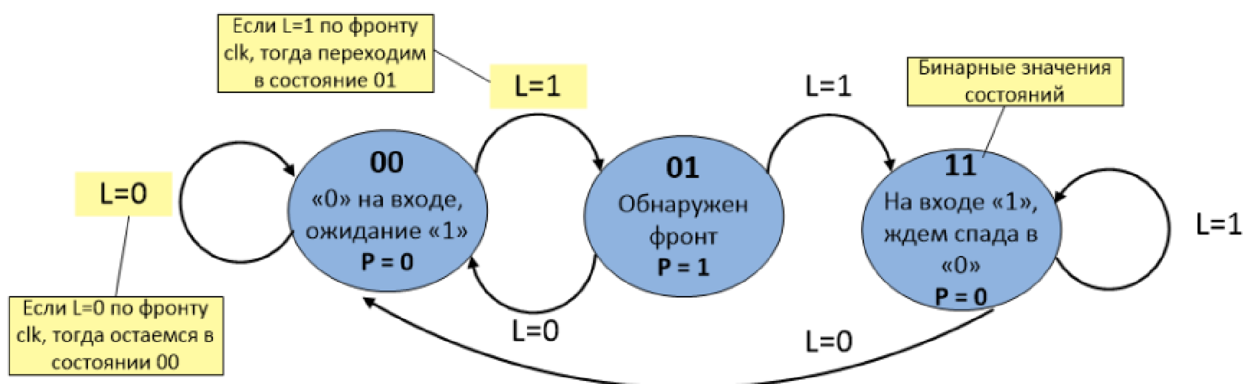


Рисунок 4 – Диаграмма состояний

На рисунке 4 представлена диаграмма состояний для реализации преобразователя уровня в импульс. Как видно по состоянию «01» появляется импульс на регистре “P” после чего следующий импульс появится только после из состояния «00», когда снова произойдет перепад с 0 на 1 на входе «L».

Ниже представлен пример кода на языке Verilog, соответствующий данному алгоритму.

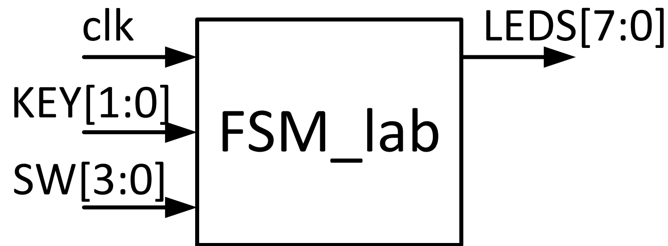
```
module FSM_lab(input reset, input clk, input L, output reg P);
reg [1:0] state;
always @(posedge clk)
    if (~reset)
        state <= 0;
    else
        case(state)
            2'b00: if (L==1) state <= 2'b01;
            2'b01: if (L==1) state <= 2'b11;
                else state <=2'b00;
            2'b11: if (L==0) state <= 2'b00;
            default: state <=0;
        endcase
always @(posedge clk)
    if (~reset)
        P<=0;
    else
        if (state==2'b01) P <= 1;
        else P <= 0;
endmodule
```

Лабораторное задание

Реализовать программу с блоком FSM на основе заданной диаграммы состояний. В качестве входных сигналов для смены состояний необходимо использовать переключатели и кнопки. Проверить результат в инструменте State Machine Viewer. Сигналы на выходе вывести на светодиоды для проверки правильности реализации алгоритма в соответствии с вариантом.

Порядок выполнения работы

1) Создать проект в Quartus II.
2) Обозначить следующие входы и выходы в модуле программы.



SW[3:0] – 4-х разрядная шина от 4-х переключателей SWITCH на плате DE0-NANO,

KEY[1:0] – 2-х разрядная шина от 2-х кнопок на плате DE0-NANO,

clk – тактовый генератор на 50 МГц на плате DE0-NANO,

LEDS[7:0] – 8-и разрядная шина, подключенная к 8 светодиодам на плате DE0-NANO.

3) Написать код на языке Verilog с блоком FSM, соответствующий вашему варианту (вариант указывает преподаватель).

Варианты заданий:

1-й ВАРИАНТ:

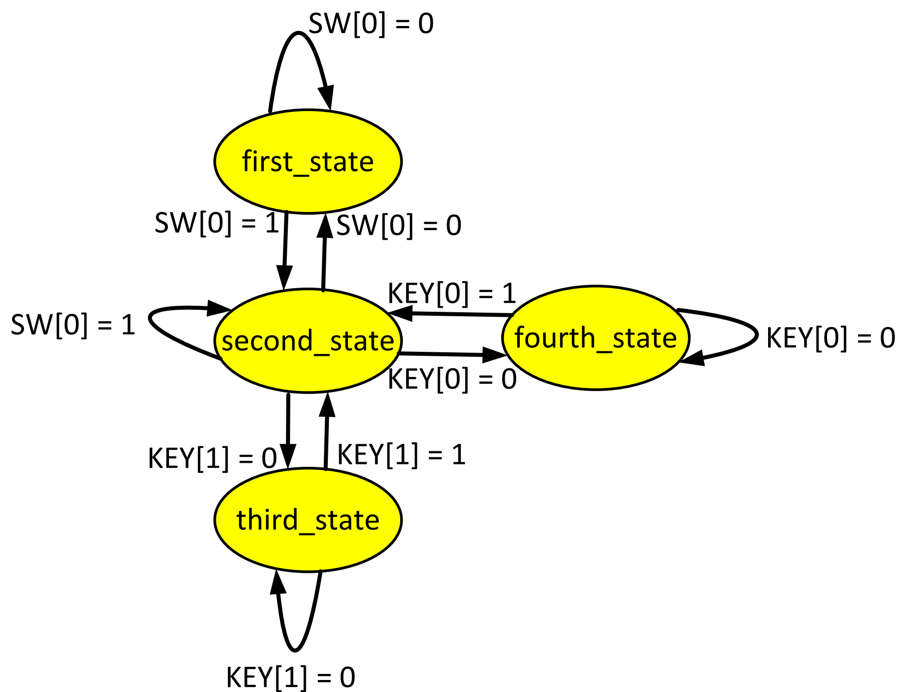


Таблица переходов состояний:

Исходное состояние	Конечное состояние	Условие
first_state	first_state	SW[0] = 0
first_state	second_state	SW[0] = 1
second_state	second_state	SW[0] = 1
second_state	first_state	SW[0] = 0
second_state	third_state	KEY[1] = 0
second_state	fourth_state	KEY[0] = 0
third_state	third_state	KEY[1] = 0
third_state	second_state	KEY[1] = 1
fourth_state	fourth_state	KEY[0] = 0
fourth_state	second_state	KEY[0] = 1

При каждом состоянии должно выполняться следующее присвоение:

first_state: LEDS <= 0;

second_state: LEDS <= 8'b11111111;

third_state: LEDS <= 8'b10101010;

fourth_state: LEDS <= 8'b01010101;

2-й ВАРИАНТ:

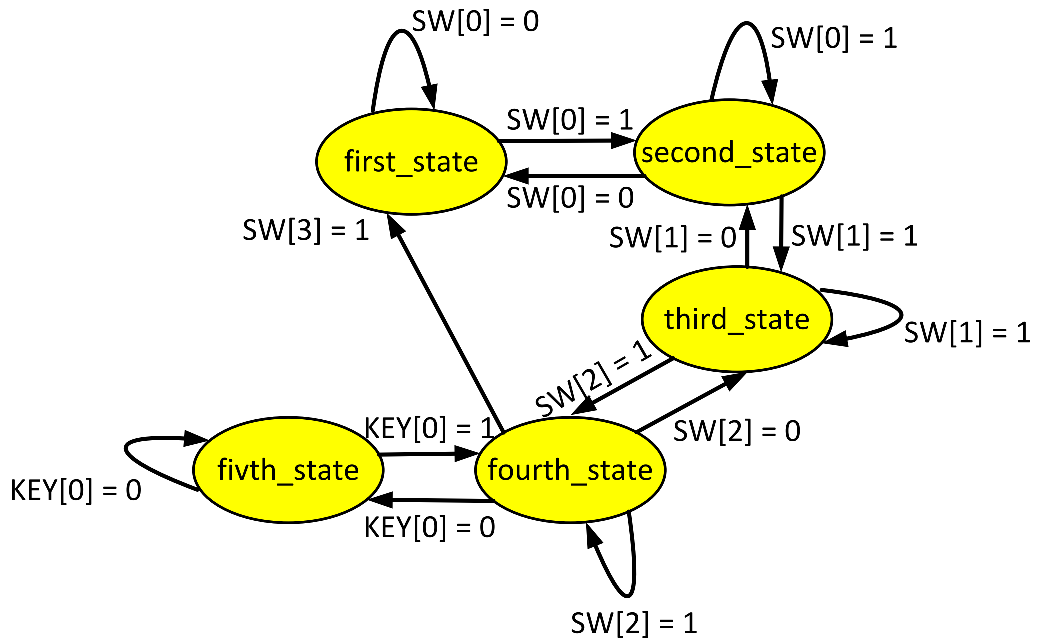


Таблица переходов состояний:

Исходное состояние	Конечное состояние	Условие
first_state	first_state	SW[0] = 0
first_state	second_state	SW[0] = 1
second_state	second_state	SW[0] = 1
second_state	first_state	SW[0] = 0
second_state	third_state	SW[1] = 1
third_state	third_state	SW[1] = 1
third_state	second_state	SW[1] = 0
third_state	fourth_state	SW[2] = 1
fourth_state	fourth_state	SW[2] = 1
fourth_state	third_state	SW[2] = 0
fourth_state	fivth_state	KEY[0] = 0
fourth_state	first_state	SW[3] = 1
fivth_state	fourth_state	KEY[0] = 1
fivth_state	fivth_state	KEY[0] = 0

При каждом состоянии должно выполняться следующее присвоение:

first_state: LEDS <= 0;

second_state: LEDS <= 8'b11111111;

third_state: LEDS <= 8'b10101010;

fourth_state: LEDS <= 8'b01010101;

fivth_state: LEDS <= 8'b11110000;

3-й ВАРИАНТ:

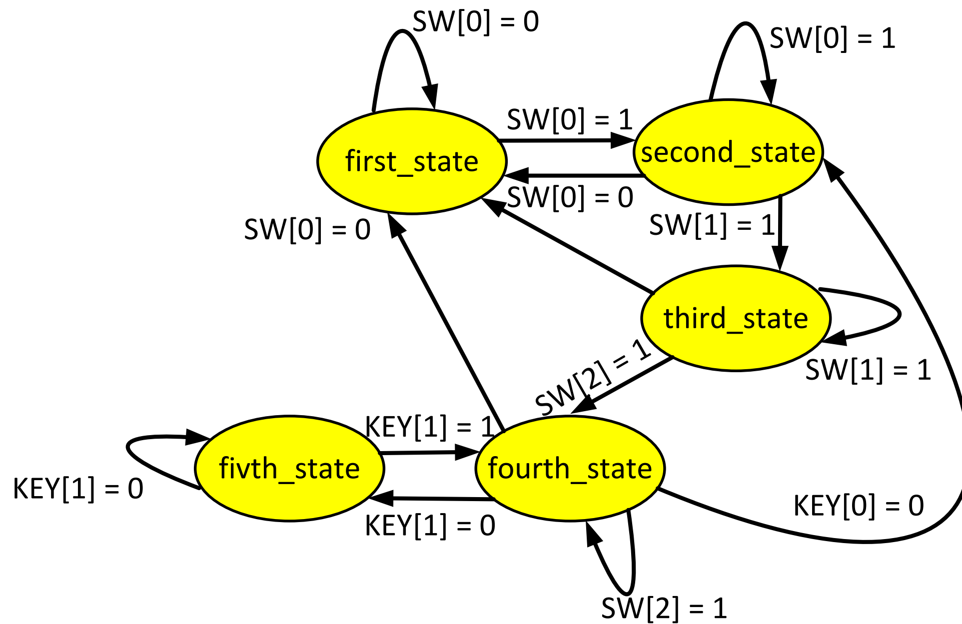


Таблица переходов состояний:

Исходное состояние	Конечное состояние	Условие
first_state	first_state	SW[0] = 0
first_state	second_state	SW[0] = 1
second_state	second_state	SW[0] = 1
second_state	first_state	SW[0] = 0
second_state	third_state	SW[1] = 1
third_state	third_state	SW[1] = 1
third_state	first_state	SW[0] = 0
third_state	fourth_state	SW[2] = 1
fourth_state	fourth_state	SW[2] = 1
fourth_state	second _state	KEY[0] = 0
fourth_state	fivth_state	KEY[1] = 0
fourth_state	first_state	SW[0] = 0
fivth_state	fourth_state	KEY[1] = 1
fivth_state	fivth_state	KEY[1] = 0

При каждом состоянии должно выполняться следующее присвоение:

first_state: LEDS <= 0;

second_state: LEDS <= 8'b11111111;

third_state: LEDS <= 8'b10101010;

fourth_state: LEDS <= 8'b01010101;

fivth_state: LEDS <= 8'b11110000;

4-й ВАРИАНТ:

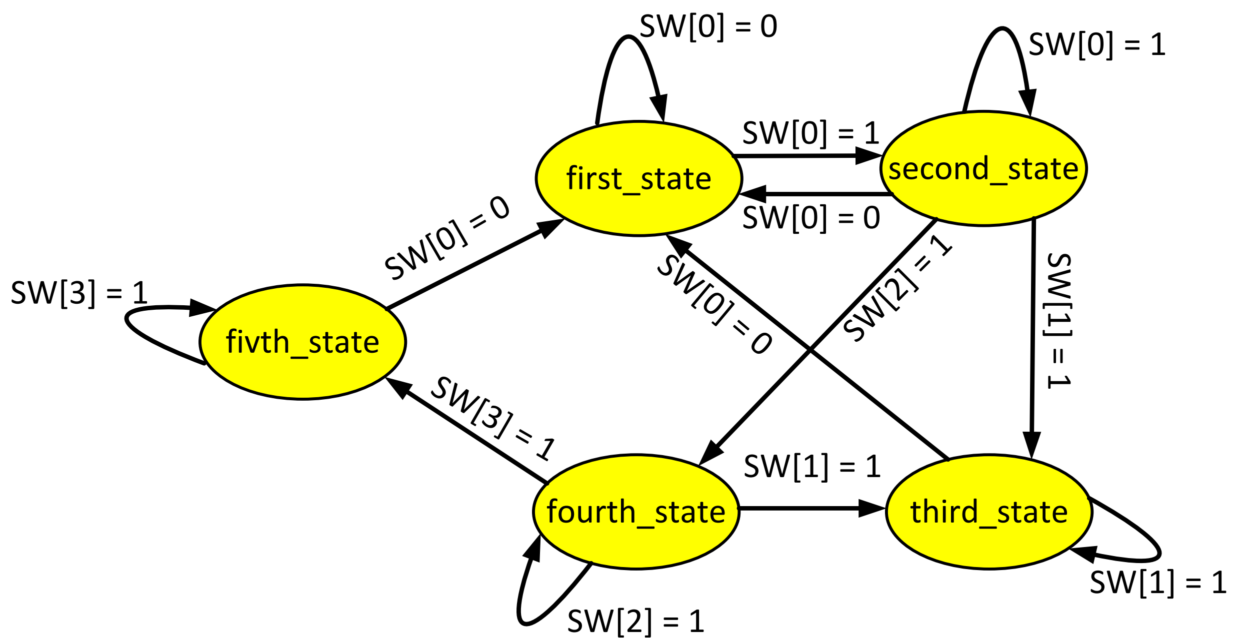


Таблица переходов состояний:

Исходное состояние	Конечное состояние	Условие
first_state	first_state	SW[0] = 0
first_state	second_state	SW[0] = 1
second_state	second_state	SW[0] = 1
second_state	first_state	SW[0] = 0
second_state	third_state	SW[1] = 1
second_state	fourth_state	SW[2] = 1
third_state	third_state	SW[1] = 1
third_state	first_state	SW[0] = 0
fourth_state	fourth_state	SW[2] = 1
fourth_state	fivth_state	SW[3] = 1
fivth_state	fivth_state	SW[3] = 1
fivth_state	first_state	SW[0] = 0

При каждом состоянии должно выполняться следующее присвоение:

first_state: LEDS <= 0;

second_state: LEDS <= 8'b11111111;

third_state: LEDS <= 8'b10101010;

fourth_state: LEDS <= 8'b01010101;

fivth_state: LEDS <= 8'b11110000;

4) После завершения программы, проверить результат в инструменте State Machine Viewer. Убедиться, что результат совпадает с вашим вариантом.
5) Провести временную симуляцию с демонстрацией переходов всех состояний.
6) Прошить плату. Убедиться, что светодиоды горят в соответствии с вашим вариантом.
7) Сделать выводы по проделанной работе.

1. Содержание отчета:

- цель работы
- вариант задания
- листинг программы
- временная симуляция
- результат синтеза в инструменте State Machine Viewer
- выводы по проделанной работе

Лабораторная работа №5

«Интерфейс передачи данных SPI»

Цель работы: Изучение интерфейса последовательной передачи данных SPI, включая основные принципы приёма и передачи.

Интерфейс SPI.

Последовательный периферийный интерфейс SPI (Serial Peripheral Interface) представляет собой последовательный синхронный протокол передачи данных в режиме полного дуплекса, разработанный компанией Motorola для обеспечения простого и недорогого сопряжения микроконтроллеров и периферии. SPI также иногда называют четырёхпроводным (англ. four-wire) интерфейсом.

Главным составным блоком интерфейса SPI является обычный сдвиговый регистр, сигналы синхронизации и ввода/вывода битового потока которого и образуют интерфейсные сигналы. Таким образом, протокол SPI правильнее назвать не протоколом передачи данных, а протоколом обмена данными между двумя сдвиговыми регистрами, каждый из которых одновременно выполняет и функцию приемника, и функцию передатчика. Непременным условием передачи данных по шине SPI является генерация сигнала синхронизации шины. Этот сигнал имеет право генерировать только ведущий шины и от этого сигнала полностью зависит работа подчиненного шины.

На рисунке 1 представлена структурная схема интерфейса SPI. В данном примере интерфейс состоит из ведущего(master) и ведомого (slave), где на каждой стороне реализован сдвиговый регистр. В схеме всегда может быть только один ведущий, в то время как ведомых может быть несколько. Интерфейс состоит из следующих линий:

- MOSI — выход ведущего, вход ведомого (англ. Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.
- MISO — вход ведущего, выход ведомого (англ. Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.

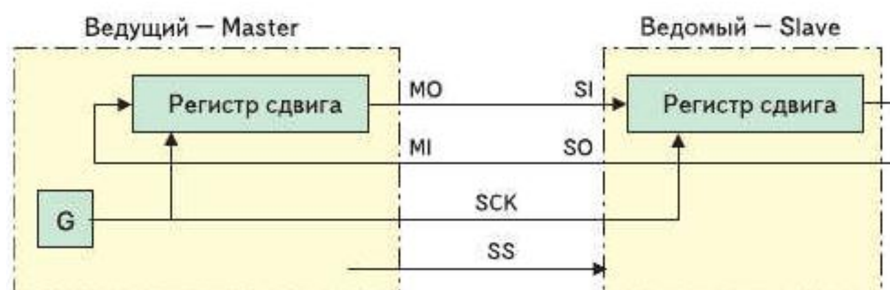


Рисунок 5.1 – Структурная схема

- SCLK — последовательный тактовый сигнал (англ. Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
- SS — выбор микросхемы, выбор ведомого (англ. Slave Select, Chip Select). С помощью данного сигнала происходит активация ведомого устройства. Обычно он является инверсным, то есть низкий уровень считается активным.

Существует четыре режима работы SPI устройств. Как правило, именно они вызывают больше всего путаницы у новичков. Данные четыре режима представляют собой комбинацию двух бит:

- CPOL (Clock Polarity) — определяет начальный уровень (полярность) сигнала синхронизации. CPOL=0 показывает, что сигнал синхронизации (SCK) начинается с низкого уровня, так что передний фронт является нарастающим, а задний — падающим. CPOL=1, сигнал синхронизации начинается с высокого уровня, таким образом передний фронт является падающим, а задний — нарастающим.
- CPHA (Clock Phase) — фаза синхронизации, определяет по какому из фронтов синхронизирующего сигнала производить выборку данных. CPHA=0 показывает, что необходимо производить выборку по переднему фронту, а CPHA=1 показывает, что выборку данных необходимо производить по заднему фронту.[1]

Рассмотрим временные диаграммы всех четырех режимов в зависимости от значений CPOL и CPHA, представленные на рисунке 2.

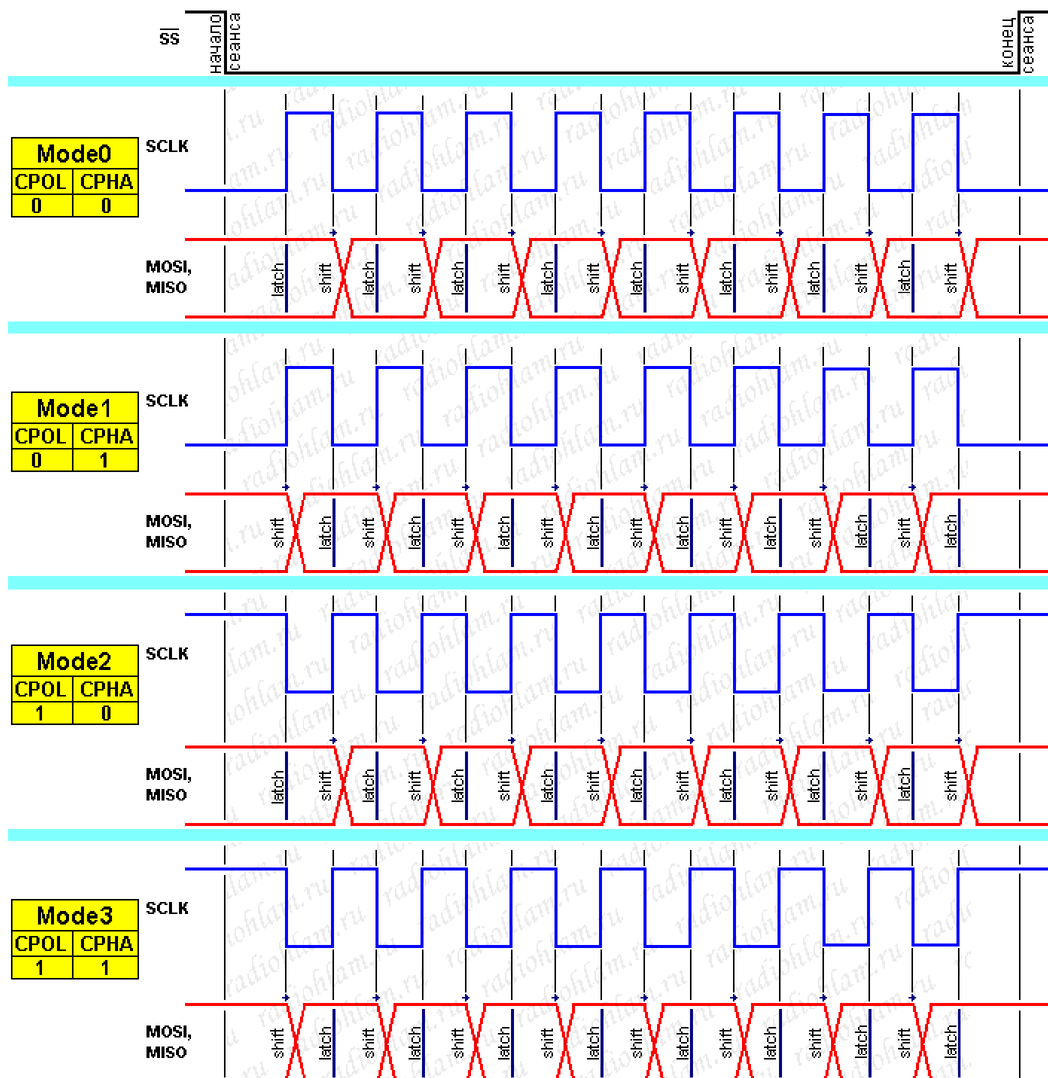


Рисунок 5.2 – Режимы работы интерфейса SPI

Как видно на рисунке 2, начало обмена для всех режимов работы определяется установкой сигнала выбора ведомого SS в активное состояние SS = 0. Обозначения «Latch» и «Shift» обозначают захват и сдвиг данных. Соответственно, в случае mode0 во время захвата данных при направлении передачи от ведущего к ведомому либо от ведомого к ведущему в первый перепад сигнала синхронизации SCK используется ведомым или ведущим устройством для запоминания очередного бита во внутреннем сдвиговом регистре контроллера SPI. Во время сдвига данных очередной бит посылки на линии MOSI сдвигается по каждому нисходящему фронту сигнала SCK.

Лабораторное задание

Реализовать на языке Verilog интерфейс передачи данных со стороны ведомого (slave) с односторонней передачей данных в сторону ведущего, т.е. от ведомого к

ведущему. Проверить программу в среде Modelsim, задать сигналы управления SS, SLK и данные идущие на с выхода ведомого.

Порядок выполнения работы

1) Создать проект в среде ModelSim

2) Создать два файла в вашем проекте. Первый файл будет вашей программой, где реализован сам интерфейс, например, назовем «spi_lab». Второй файл будет выполнять роль тестбенча (testbench), т.е. программы для симуляции работы, например, назовем «spi_lab_tb».

3) Для программы spi_lab обозначьте следующие входы и выходы:

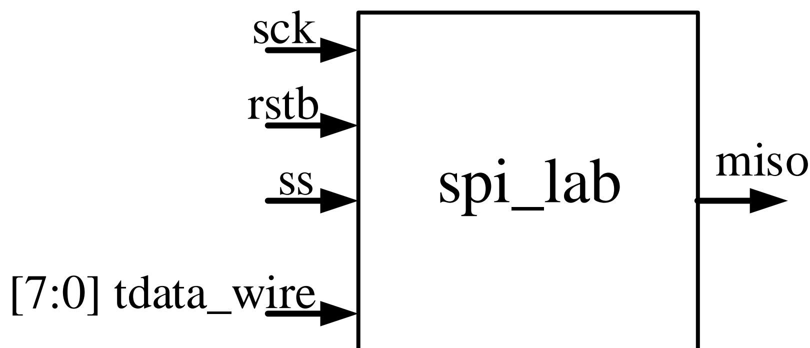


Рисунок 3 – Входы и выходы модуля spi_lab

sck – входная шина синхронизации от ведущего,

rstb – асинхронный сброс,

ss – выбор ведомого,

tdata_wire – 8-разрядная шина, идущая на вход сдвигового регистра.

miso – выход идущий на вход ведущего.

4) Реализовать 3-разрядный счетчик (например, reg [2:0] counter) с асинхронным сбросом и входом разрешения на запись. Тактовую частоту завести из шины sck с условием posedge sck. В качестве сброса использовать инверсную шину «!rstb». В качестве входа разрешения на запись использовать инверсную шину «!ss».

5) Использовать код, представленный ниже, со сдвиговым регистром для передачи информации в сторону ведущего:

```
assign miso = shift_reg[7];

always @(negedge sck or negedge rstb)
```

```
begin
    if (!rstb)
  shift_reg <= 0;
    else
        begin
         if(!ss)
                begin
                 if(counter==0)
                     shift_reg <= tdata_wire;
                 else
           shift_reg <= {shift_reg[6:0],1'b1};
                end //!ss
        end //rstb
    end //always
```

Данный код выполняет следующую операцию: Передача информации выполняется при условии отсутствия сигнала на «!rstb» и после того, как сигнал ss опустится в «0». В это же время начинает работать счетчик counter и вся информация с шины tdata_wire сбрасывается в сдвиговый регистр shift_reg при условии counter==0. Новая информация поступит на регистр, когда счетчик снова сбросится в ноль т.е. после восьми тактов sck. Далее в течении восьми тактов сдвиговый регистр будет посылать данные на шину miso, подключенную к восьмому регистру shift_reg, пока счетчик не досчитает до восьми и не загрузит новую информацию с шины tdata_wire. Так как сдвиг происходит по negedge sck,то данный код предназначен для режимов mode0 и mode3. Для большей наглядности проведем симуляцию.

Примечание: НЕ забудьте объявить 8-разрядный регистр shift_reg.

6) Теперь приступим к написанию тестбенча. В новом файле в начале нужно задать следующие параметры:

```
`timescale 1 ns / 1 ns //шаг задержки//шаг симуляции

module spi_lab_tb;
   reg rstb = 1'b0;
   reg ss = 1'b0;
   reg [7:0] tdata_wire = 8'b00000000;
```

```
reg sck = 1'b0;
wire miso;

spi_lab spi_lab_in_tb( // если файл называется по
    .rstb(rstb),       // другому то следует поменять
        .ss(ss),       // spi_lab на свое название
        .sck(sck),
        .miso(miso),
        .tdata_wire(tdata_wire)
);
```

Далее после команды initial begin нужно прописать все значения для каждого момента времени. Всего нужно послать два байта информации на шину tdata_wire. Значение обоих байт выбрать самостоятельно. Подробная временная диаграмма, которую нужно просимулировать представлена на рисунке 4. Как видно исходное положение у SCK равна единицы и так как сдвиг данных происходит по нисходящему фронту, то в данном случае реализован режим mode3.

В качестве примера ниже описан кусок кода до 120 нс.

```
initial begin
    // 20ns
    #20;
    ss = 1'b1;
    sck = 1'b1;
    // -------------------------------------
    // 40ns
    #20;
    rstb = 1'b1;
    // -------------------------------------
    // 60ns
    #20;
    tdata_wire = 8'b01111100;
    // -------------------------------------
    // 80ns
    #20;
```

```
ss = 1'b0;
// ------------------------------------
// 100ns
#20;
sck = 1'b0;
// ------------------------------------
// 120ns
#20;
sck = 1'b1;
    end
```

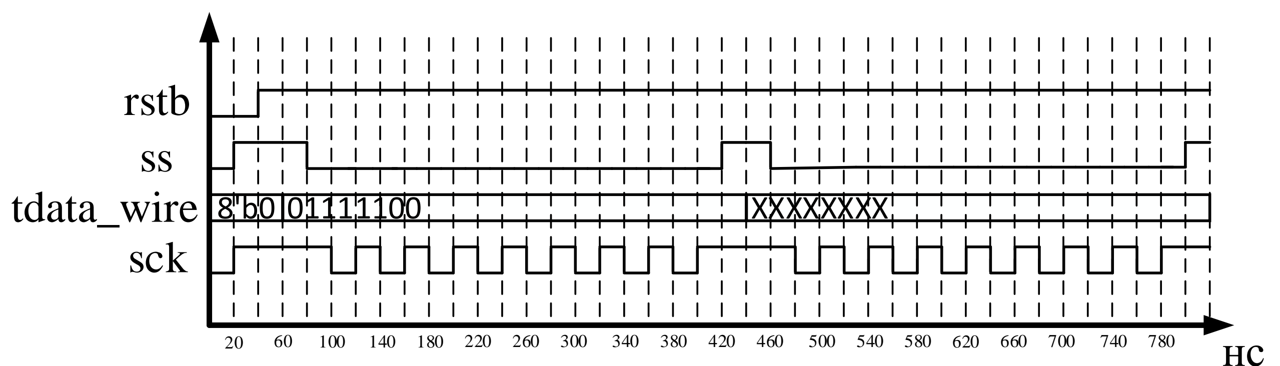


Рисунок 5.4 – Временная симуляция, для реализации в ModelSim

7) Сохранить результат симуляции. Сделать выводы по проделанной работе

Содержание отчета:

- цель работы
- листинг программы реализации интерфейса SPI
- листинг программы для симуляции в Modelsim
- результат симуляции
- выводы по проделанной работе

Лабораторная работа №6.

Реализация КИХ-фильтра на ПЛИС

Цель работы: Создание КИХ фильтра на основе ПЛИС на языке Verilog с заданными параметрами. Знакомство со средой симуляции Modelsim

Ких фильтр:

Фильтр с конечной импульсной характеристикой (КИХ, FIR- finite impulse response, нерекурсивный фильтр) – один из видов цифровых фильтров с ограниченной по времени импульсной характеристикой.

Фильтр может быть задан разностным уравнением:

$$y(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) + \cdots + a_k x(n-k) \qquad (1)$$

, где y(n)-выходной сигнал, a_i-коэффициенты фильтра, x(n-i)-входной сигнал, задержанный на i отсчетов, k-порядок КИХ фильтра. Можно сказать, что значение выхода фильтра есть значение отклика на входное значение фильтра и сумма постепенно затухающий откликов k предыдущих отсчетов. На рисунке 6.1 изображена реализация КИХ - фильтра четвертого порядка.

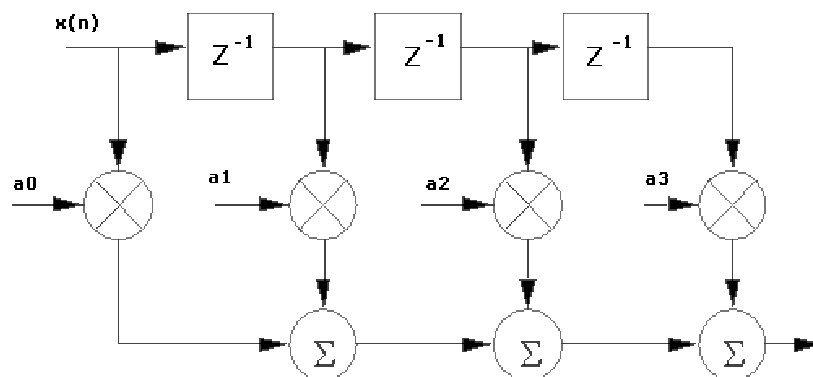


Рисунок 6.1- Реализация КИХ фильтра четвертого порядка

На рисунке 1 x(n)-входной сигнал, Z^{-1}- задержка, a_0-коэффициенты фильтра, ×-умножитель, ∑-сумматор.

Цель лабораторной работы - реализовать средствами проектирования ПЛИС КИХ-фильтр нижних частот с частотой среза и порядком, заданными преподавателем (вариант табл. 1).ъ

Таблица 6.1. Варианты заданий.

	1-ый вариант	2-ой вариант	3-ий вариант	4-ый вариант	5-ый вариант
Тип фильтра	ФНЧ	ПФ	ФНЧ	ПФ	ФВЧ
Порядок фильтра	12	10	13	12	11
Частота среза 1	10 МГц	10МГц	15МГц	5МГц	-
Частота среза 2	-	20МГц	-	15МГц	15МГц

Лабораторное задание

Рассчитать коэффициенты фильтра. Реализовать модуль КИХ-фильтра на языке Verilog. Изучить пакет ModelSim, освоить методику написания тестбенчей. Провести симуляцию КИХ-фильтра.

Порядок выполнения работы

Расчет коэффициентов фильтра средствами Scilab

Scilab-пакет прикладным математических программ, представляет собой бесплатный аналог пакета Matlab.

1) Откройте Scilab, создайте в нем новый файл.
2) Для расчета коэффициентов КИХ фильтра используйте функцию ffilt.

[x]=ffilt(ft,n,fl,fh)

- x-массив коэффициентов фильтра
- ft-тип фильтра. "lp"-фильтр нижних частот, "hp"-фильтр верхних частот, "bp"-полосовой фильтр, "sb"-полосно заграждающий фильтр.
- n-порядок фильтра
- fl-нижняя частота среза фильтра
- fh – верхняя частота среза фильтра

При расчете коэффициентов ФНЧ и ФВЧ (заданный параметром ft тип фильтра) задается только одна частота среза.

Частота среза задается в виде коэффициента, рассчитанного относительно частоты дискретизации:

$$f_{l(h)} = \frac{f_{\text{ср}}}{f_{\text{д}}} \quad (2)$$

Исходя из теоремы Котельникова следует, что коэффициент лежит в диапазоне [0:0.5]. При расчете $f_{l(h)}$ возьмите $f_{\text{д}}$ равную 50 МГц.

Предположим, что необходимо рассчитать коэффициенты КИХ фильтра нижних частот 10-го порядка с частотой среза 10Мгц.Тогда функция ffilt будет иметь следующий вид:

x=ffilt("lp",10,0.2)

3) При реализации фильтра вам будет нужно умножить входные отсчеты сигнала на коэффициенты фильтра. Отсчеты сигнала имеют размерность 12 бит. Необходимо обеспечить такую же разрядность коэффициентов. Для этого отнормируйте полученные коэффициенты относительно модуля максимального. После этой операции максимальный коэффициент станет равный единице, модуль остальных будет лежать в интервал [0:1]. Приведите коэффициенты к нужной разрядности. Для этого умножьте все отсчеты на 2^{n-1} -1 (n=12), при этом максимальный коэффициент будет занимать все n разрядов. Показатель степени 2 равен n-1 в силу того, что старший разряд числа знаковый.

   Эти операции описывает формула:

$$koeff_{\text{норм}} = \frac{koeff}{max(|(koeff)|)} * (2^{n-1} - 1) \quad (3)$$

   После этой операции отбросьте дробную часть чисел с помощью функции round().

4) Сохраните рассчитанные коэффициенты.

Реализация КИХ-фильтра на ПЛИС

Напишите программу на языке Verilog, описывающую работу КИХ-фильтра.

Разрядность входных данных - 12 бит. Коэффициенты фильтра (a) и его задержки (delay_pipe) задайте в виде массива знаковых регистров длиной n (порядок фильтра), с размерностью в 12 бит. Значения регистров коэффициентов фильтра задайте в блоке initial. Пример синтаксиса:

```
reg signed [11:0] a [10:0];
initial
    begin
            a[0]=488;
    ....
            a[10]=488;
    end
```

В вышеуказанных примерах слово signed означает знаковый тип переменной, т.е. старший разряд отведен под обозначение знака, «1» соответствует отрицательному числу, все регистры, которые потенциально могут содержать отрицательные числа, должны быть знаковыми.[2]

Задержка отсчетов сигнала реализуется с помощью сдвигового регистра. Такую конструкцию удобнее всего реализовать через цикл for. Пример синтаксиса:

```
for (index = 0; index <n; index=index+1)
                        «тело цикла»;
```

index-переменная типа integer;

Конструкция:

```
always@(posedge clk)
for (index = 0; index <3; index=index+1)
a[index+1]<=a[index];
```

равносильна коду:

```
always@(posedge clk)
begin
        a[1]<=a[0];
        a[2]<=a[1];
        a[3]<=a[2];
end
```

Операции в регистрах задержки происходят синхронно с тактовой частотой(always блок). Младшему регистру массива delay_pipe (массив регистров задержки) присваивается входной сигнал, остальные регистры массива передают свои значения друг другу по цепочке. Каждая задержка умножается на свой коэффициент, результат n произведений суммируется. Таким образом система использует n умножителей 12*12. Обратите внимание, что входной сигнал без задержки также умножается на свой коэффициент (a_0), и суммируется с остальными.

Результат произведения и суммирования будет иметь разрядность, не равную разрядности множителей. Самостоятельно выберите разрядность, обоснуйте выбор в отчете.

Реализуйте синхронный сброс массива регистров задержки сигнала и регистра суммы по сигналу reset.

Симуляция фильтра в Modelsim

Modelsim – среда для симуляции и отладки программ на HDL языках. По сравнению с встроенным симулятором Quartus II, она имеет больше возможностей, не требует долгой компиляции проекта в Quartus.

Для работы в Modelsim пишется отдельный модуль, называющиеся testbench. В нем задаются сигналы для входов тестируемого модуля, производятся операции над данными, полученными в результате работы модуля (сохранение в файл, экспорт, пост-обработка).

1)Создайте новый проект в среде Modelsim: File/New/Project . Откроется окно создания проекта, приведенное на рисунке 6.2.

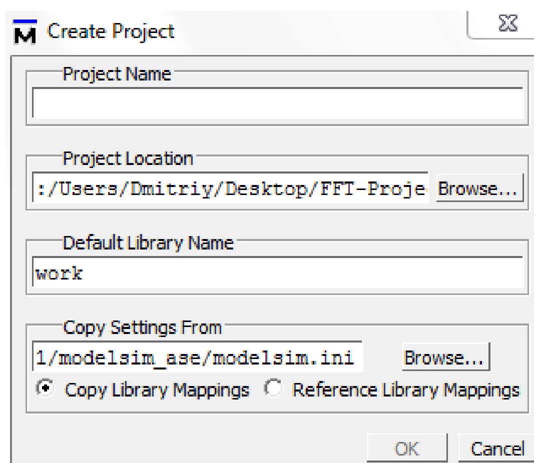


Рисунок 6.2 – Окно создания проекта Modelsim

Первая строка-название проекта, второе-путь к папке с проектом. Как и в Quartus, в них не должно содержаться русских букв и пробелов. должно содержать пробелов.

Далее появится окно, где вам предлагают создать новый файл, или добавить готовый. Выбираем «Add Existing File», предварительно скопировав файл fir_tb.v из папки с методичкой в папку с вашим проектом Modelsim. Добавьте в проект файл testbench и ваш файл, реализующий КИХ-фильтр. Добавьте в папку файл m_test.txt с записанным сигналом. В файле частотно модулированный сигнал, приведенный к нужной разрядности. Начальная частота сигнала 1 МГц, полоса-25 МГц. На рисунке 6.3 изображена временная форма сигнала, сгенерированная в MatLab.

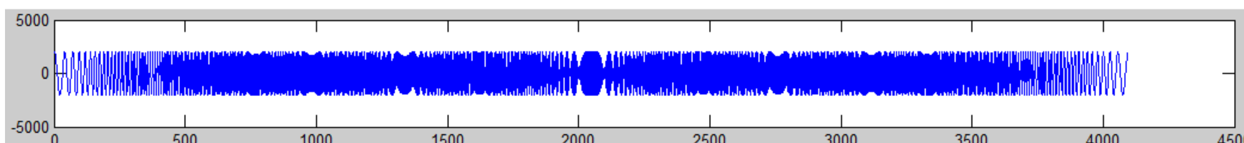


Рисунок 6.3 – Временная форма сигнала с частотной модуляцией

Откройте файл fir_tb.v. Это файл-testbench к реализованному вами фильтру. Testbench имеет синтаксис языка Verilog, но использует несинтизируемые конструкции, которые нельзя прошить в ПЛИС, они используются только при симуляции. Каждый testbench имеет примерно следующую структуру:

- *`timescale 10 ns / 10 ns* //директива шкалы времени.
- *module fir_tb;* // название модуля тестбенча-обратите внимание на отсутствие входов и выходов.
- *reg clk;* и т.д. //обьявление внутренних регистров и шин, соединенных с портами модуля, или используемых только в тестбенче.
- Задаем внутреннюю тактовую частоту

always // Отутствует список чувствительности, блок выполняется всегда.

begin

clk <= 1'b1;

 # 1; // задержка на один такт, длительность такта задается в первой строке модуля (timescale)

clk <= 1'b0;

 # 1; //после осуществления задержки, снова выполняется первая операция блока always (clk <= 1'b1;) и т.д.

```
end
```

- Задаем сброс (reset)

```
initial                               // операции в блоке initial выполняются один раз

 begin
 reset <=1'b1;
 # 30;                                //30 тактов reset равен «1»
 reset <=1'b0;
 end
```

- Прописываем путь к файлу(проверьте!) с тестовым сигналом и открываем его в режиме чтения

```
initial

signal_in_tb = $fopen("C:/Users/Dmitriy/Desktop/tb_fir/m_test.txt","r"); //переменная типа integer
```

- Каждый такт считываем отсчет тестового сигнала и записываем его в переменную input_signal.

```
always  @(posedge clk)
        if(!reset)
                data_read<=$fscanf(signal_in_tb,"%d",input_signal);
        else
                input_signal <=1'b0;
```

- Подключаем написанный вами модуль

```
fir fir (        // Первое слово-имя вашего модуля, второе- его название в тестбенче,
                 если вы назвали модуль не fir, измените первое слово.
        .clk(clk),
        .reset(reset),
        .filter_in(input_signal),
        .filter_out(filter_out)
        );
```

Если в вашем модуле КИХ фильтра порты называются по-иному, переименуйте их в тестбенче.

2) Откомпилируйте файлы, для этого нажмите на кнопку Compile All на панели инструментов

3)Проведите симуляцию, для этого нажмите на кнопку Simulate на панели инструментов и выберите файл тестбенча, расположенный в списке work. Расположение кнопок приведено на рисунке 6.4.

Рисунок 6.4 – Расположение кнопок компиляции и симуляции

В открывшимся окне раскройте список work и выберите fir_tb

4) Добавьте на временные диаграммы внутренние регистры модуля КИХ-фильтра и тестбенча, рисунок 6.5.

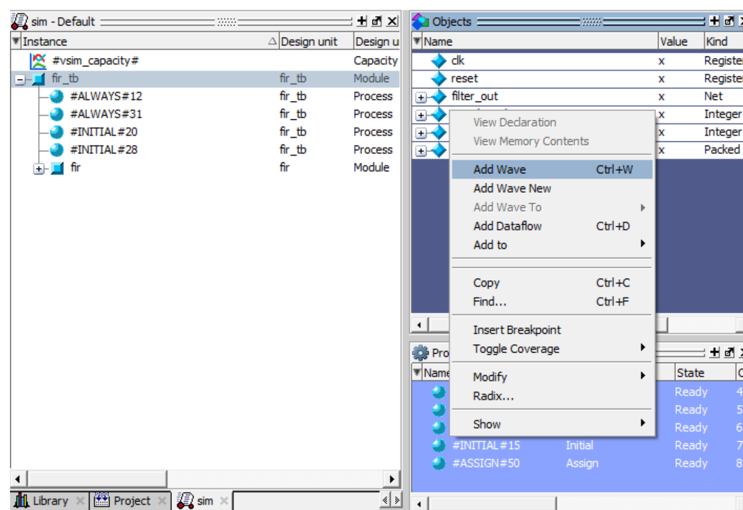


Рисунок 6.5 – Добавление временных диаграмм

5) Нажмите на кнопку Zoom Full на панели инструментов для уменьшения масштаба времени.

6) Измените формат представления диаграмм многоразрядных регистров с цифрового на аналоговый, для этого нажмите на название правой кнопкой и выберите Format/Analog(automatic), см. рисунок . На вход модуля подается сигнал с частотной модуляцией(полоса 25 МГц). При этом на выходе фильтра, огибающая сигнала будет соответствовать АЧХ фильтра.

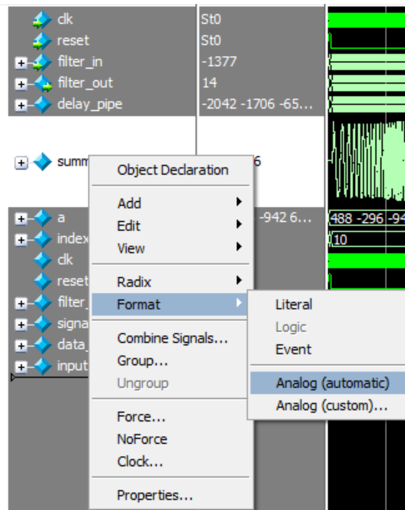


Рисунок 6.6 - Представление временных диаграмм в аналоговом виде

7) Разрядность регистра, в котором хранится результат суммирования произведения задержек на коэффициенты высока, разрядность выхода фильтра должна быть меньше, самостоятельно примите решение о том, насколько разрядов можно обрезать регистр суммы. Проведите симуляцию с разной разрядностью, посмотрите как искажается форма выходного сигнала. Результаты симуляции вставьте в отчет.

   Пример реализации уменьшения разрядности:

   assign filter_out=summ[46:18]; //17-разрядной шине выхода присваивается 17 старших разрядов регистра сумм.

При изменении разрядности портов вашего модуля, необходимо установить такую же разрядность шин, подключенных к портам, в тестбенче.

8) Получите импульсную характеристику фильтра. ИХ-реакция фильтра на дельта функцию. Вам необходимо изменить код файла fit_tb.v так, чтобы на вход фильтра подавалась дельта функция.

Содержание отчета:

- цель работы
- вариант задания
- листинг модуля Quartus и тестбенча Modelsim
- обоснование выбора разрядности регистров и шин
- результаты симуляции в Modelsim
- выводы по проделанной работе

Лабораторная работа №7.

Управление АЦП на отладочной плате DE0-NANO

Цель работы: Реализация алгоритма для управления АЦП на отладочной плате DE0-NANO. Оцифровка и запись сигнала с внешнего источника.

Аналогово-цифровой преобразователь

Аналогово-цифровой преобразователь - устройство, выполняющий преобразование входной физической величины в ее числовое представление. Формально входной величиной АЦП может быть любая физическая величина – напряжение, ток, сопротивление, емкость, частота следования импульсов, угол поворота вала и т.п. Однако, для определенности, в дальнейшем под АЦП мы будем понимать исключительно преобразователи напряжение-код. На рисунке 7.1 показан пример передаточной характеристики для 4-х разрядного АЦП.

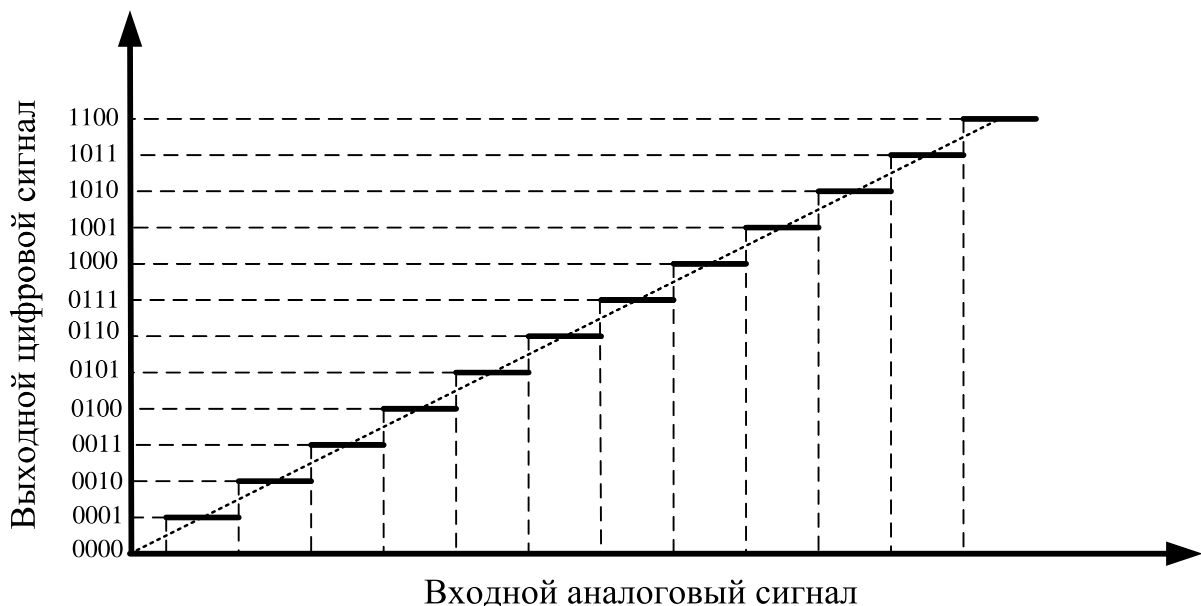


Рисунок 7.1 – Передаточная характеристика для 4-хразрядного АЦП

АЦП имеет множество характеристик, из которых основными можно назвать частоту преобразования и разрядность. Частота преобразования обычно выражается в отсчетах в секунду (samples per second, SPS), разрядность – в битах. Современные АЦП могут иметь разрядность до 24 бит и скорость преобразования до единиц GSPS (конечно, не одновременно). Чем выше скорость и разрядность, тем труднее получить требуемые характеристики, тем дороже и сложнее преобразователь. Скорость преобразования и

разрядность связаны друг с другом определенным образом, и мы можем повысить эффективную разрядность преобразования, пожертвовав скоростью.

Рассмотрим подробнее АЦП, который используется в отладочной плате DE0-NANO. Плата DE0-NANO содержит в себе маломощный 8-канальный АЦП серии ADC128SO22 с разрядностью 12 бит, который позволяет оцифровывать данные со скоростью от 50 до 200 килосэмплов в секунду. Входы 8 каналов АЦП подключены к разъему GPIO 2×13, который показан на рисунке 7.2. На рисунке 7.3 показана схема соединения кристалла ПЛИС между АЦП и разъемом GPIO 2×13.

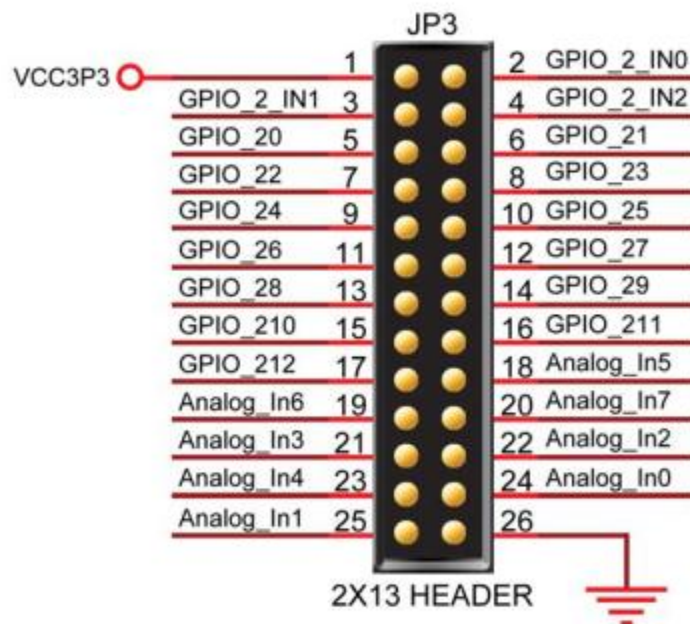


Рисунок 7.2 – Распределение пинов на GPIO 2×13

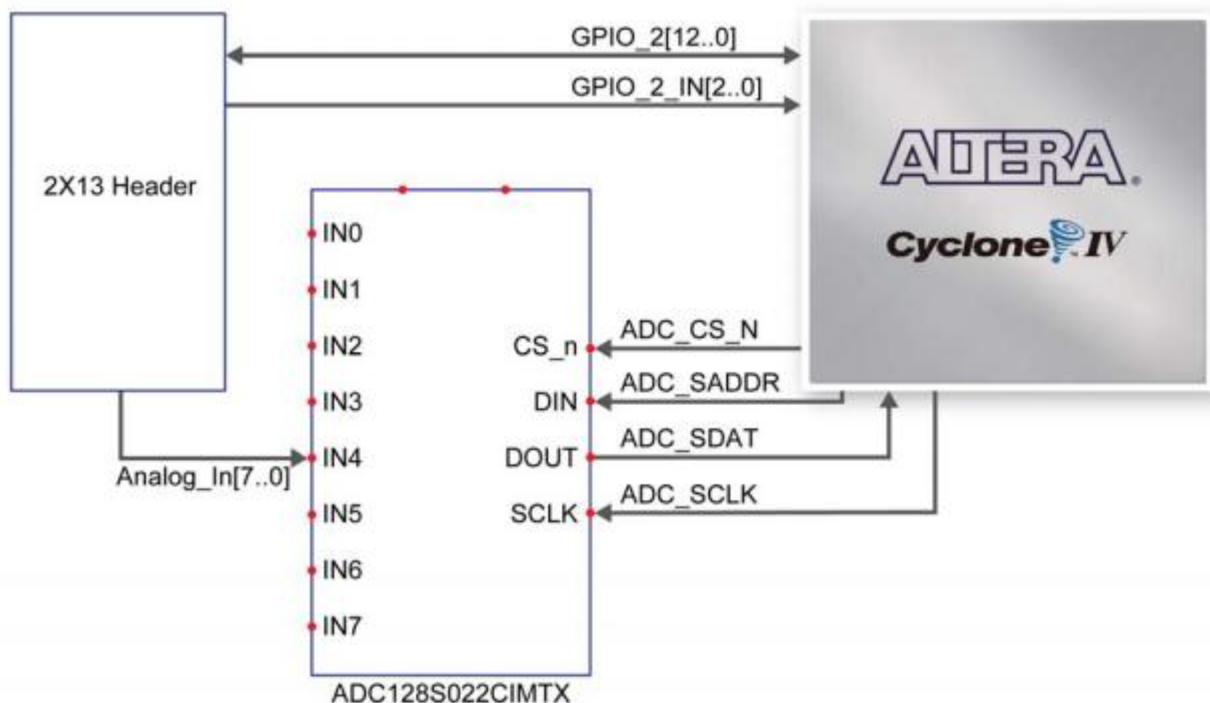


Рисунок 7.3 – Схема соединения АЦП на плате DE0-NANO

Как видно на рисунке 7.3 для управления АЦП используется три входа (CS_N, SADDR, SCLK) для управления и один выход (SDAT) из которого идет оцифрованный сигнал. Рассмотрим временные диаграммы этих сигналов, представленные на рисунке 7.4.

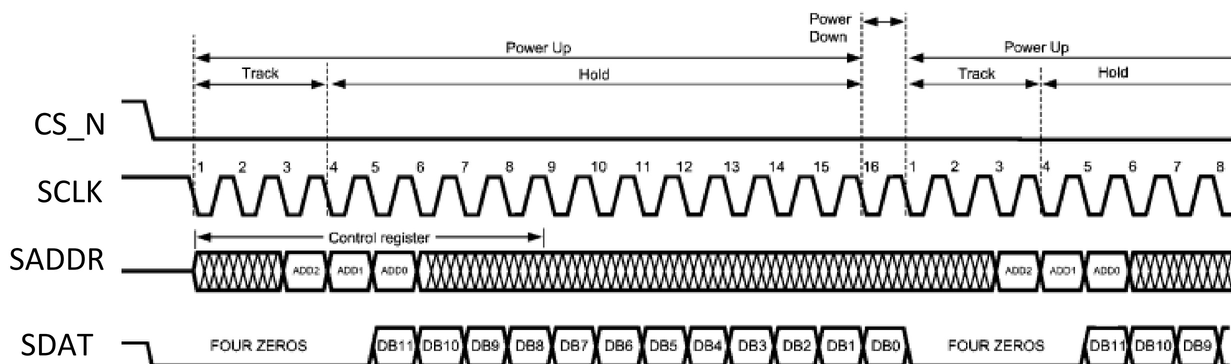


Рисунок 7.4 –Временные диаграммы сигналов с АЦП

Как видно на рисунке 7.4, для работы АЦП требуются тактовые импульсы на вход SCLK, который управляет процессом преобразования и считывания. Оптимальный диапазон тактовых частот для входа SCLK от 0.8 до 3.2 МГц.

Запуск процесса оцифровки АЦП управляется входом Chip Select (CS). После появления отрицательного фронта (из 1 в 0) АЦП будет работать до тех пор, пока на вход подается ноль.

На вход SADDR подается управляющий сигнал для переключения каналов. При этом для правильной работы АЦП нужно, чтобы сигналы шли в определённый момент времени как показано на рисунке 7.4. В таблице 7.1 указаны значения SADDR для всех восьми каналов АЦП. Соответственно, если на вход SADDR идут одни нули, то выходе АЦП будут идти данные со 1-ого канала.

Таблица 7.1 – Значения SADDR для каждого канала.

ADD2	ADD1	ADD0	Input Channel
0	0	0	IN0 (Default)
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

Выходные данные на выходе АЦП с момента включения АЦП, а точнее с момента сброса сигнала CS с 1 на ноль, как видно на рисунке 4, появляются после 5 такта SCLK, и при этом каждое следующее значение идет с такой же задержкой в 4 нуля.

Лабораторное задание

Реализовать программу для управления аналогово-цифровым преобразователем на отладочной плате DE0_NANO на языке Verilog. Изучить инструменты MegaWizard Plug-In Manager и SignalTap II Logic Analyzer. Записать аналоговый сигнал с внешнего источника.

Порядок выполнения работы

Таблица 7.2 - Варианты заданий:

Частота с выхода ФАПЧ	Количество оцифрованных точек	Тип сигнала с источника
1) 0.8 МГц 2) 1 МГц 3) 1.2 МГц 4) 1.4 МГц 5) 1.6 МГц 6) 1.8 МГц 7) 2.0 МГц 8) 2.2 МГц	1) 31 2) 63 3) 127 4) 255 5) 511 6) 1023	

9) 2.4 МГц 10) 2.6 МГц 11) 2.8 МГц 12) 3.0 МГц 13) 3.2 МГц		

1) Создать проект в Quartus II
2) Обозначить следующие входы и входы для программы, рисунок 7.5:

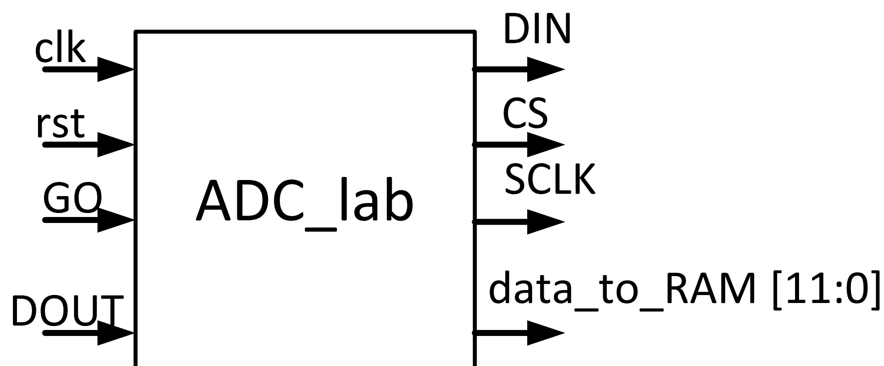


Рисунок 7.5 – Входы и выходы для управления АЦП

clk – вход тактовой частоты 50 МГЦ (PIN_R8),

rst – сброс (выбрать один из 4-х переключателей на плате switch[3:0]),

GO – вход управляющего сигнала для запуска АЦП (выбрать одну из двух кнопок на плате KEY[1:0]),

DOUT – оцифрованные данные с выхода АЦП,

DIN – выход для управляющего сигнала для переключения каналов,

CS – выход для запуска АЦП,

SCLK – выход для тактовой частоты АЦП.

data_to_RAM – оцифрованные данные с выхода контроллера АЦП.

3) Так как нужно задать частоту тактового от 0.8 до 3.2 МГц, то нужно реализовать ФАПЧ с помощью инструмента MegaWizard Plug-In Manager. Делается это в следующим образом:
   - Заходим в MegaWizard Plug-In Manager через Tools → MegaWizard Plug-In Manager;
   - В появившемся окне выбираем: Create a new custom version;
   - Далее выбираем мегафункцию из библиотеки слева на рисунке 6 через: I/O → ALTPLL, выбираем семейство кристаллов сверху Cyclone IV, тип файла, название и расположение;

- В новом окне (рисунок 7.7) выставляем необходимые параметры (рисунок 7.7): класс скорости устройства – 6, входная частота кварцевого генератора – 50 МГц
- Далее (рисунок 8) следует убрать вход для асинхронного сброса (areset) и выход для индикации запирания (locked) снятием всех галочек.
- После этого выставляется тактовая частота на выходе (рисунок 9). Частота соответствует варианту вашего задания. Стоит заметить, что здесь можно реализовать 5 выходов с различными частотами. Задействуем только один.
- Указываем необходимые файлы, которые будут лежать в вашем проекте (рисунок 10). Среди них обязательно должен быть instantiation template file.
- Открываем тот самый inst-файл, копируем содержимое и вставляем в ваш топовый модуль. Содержимое примерно должно выглядеть следующим образом: adc_pll adc_pll_inst (.inclk0 (inclk0_sig), .c0 (c0_sig)). Так как вы обозначили clk под вход кварцевого генератора, то меняем inclk0_sig на clk, и не забудьте обозначить новую шину c0_sig, написав ниже:

  wire c0_sig;

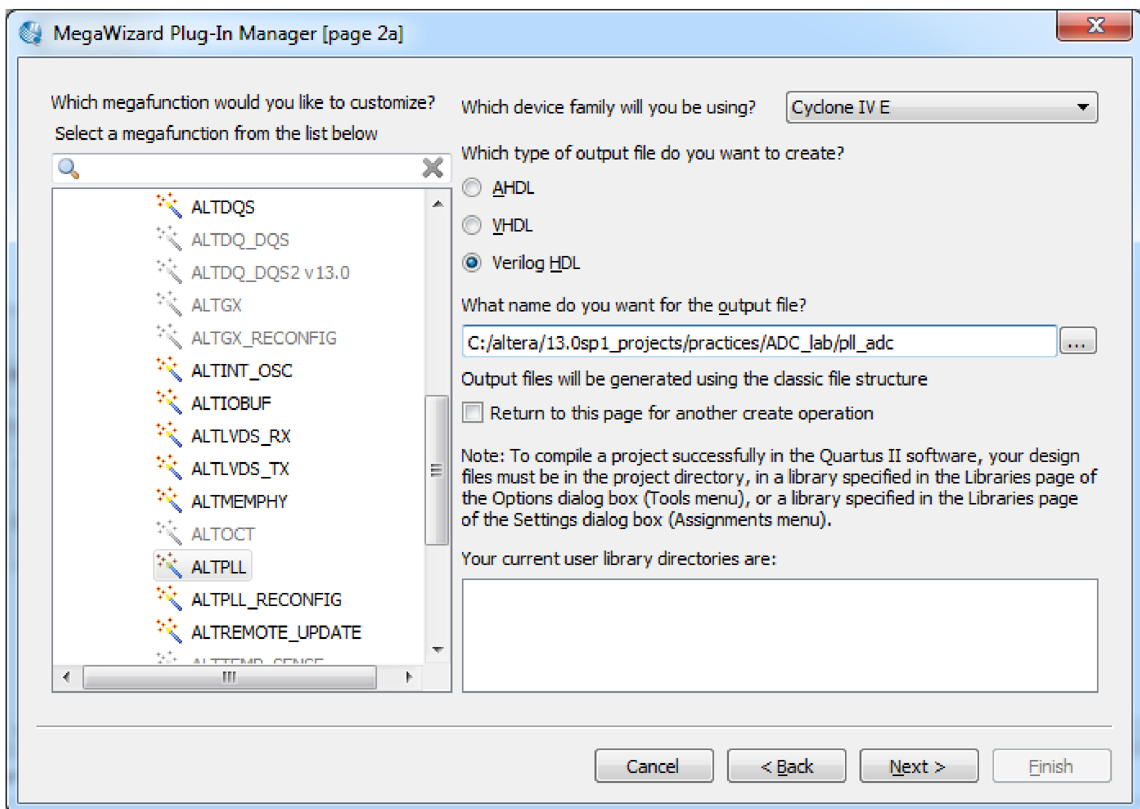


Рисунок 7.6 – Выбор мегафункции из библиотеки

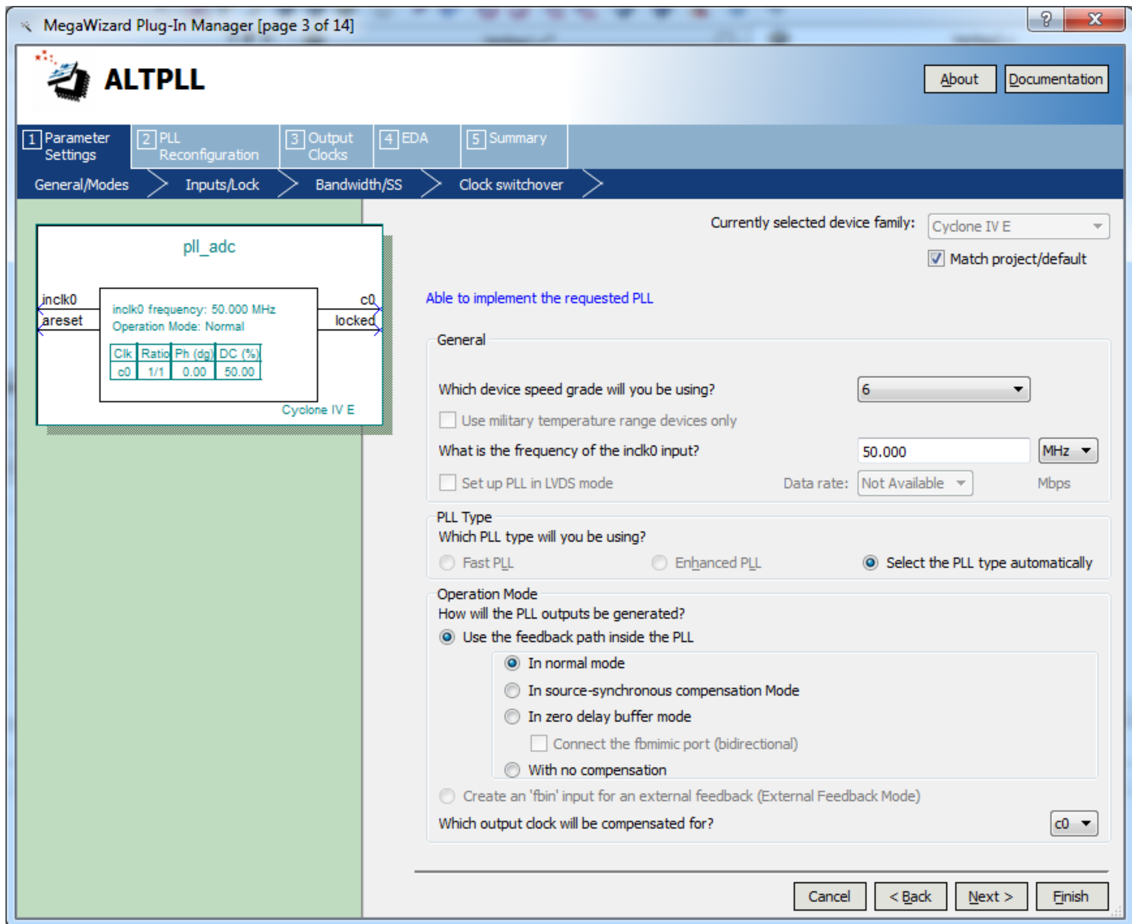


Рисунок 7.7 – Выбор параметров

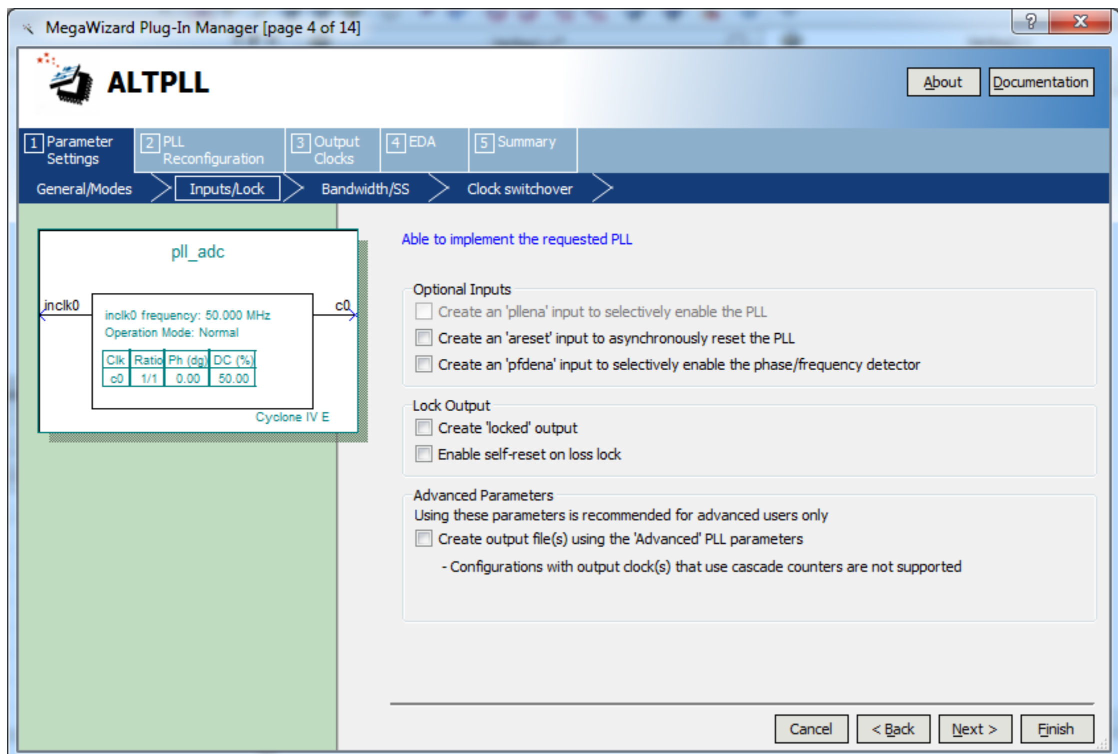


Рисунок 7.8 - Настройка дополнительных входов и выходов

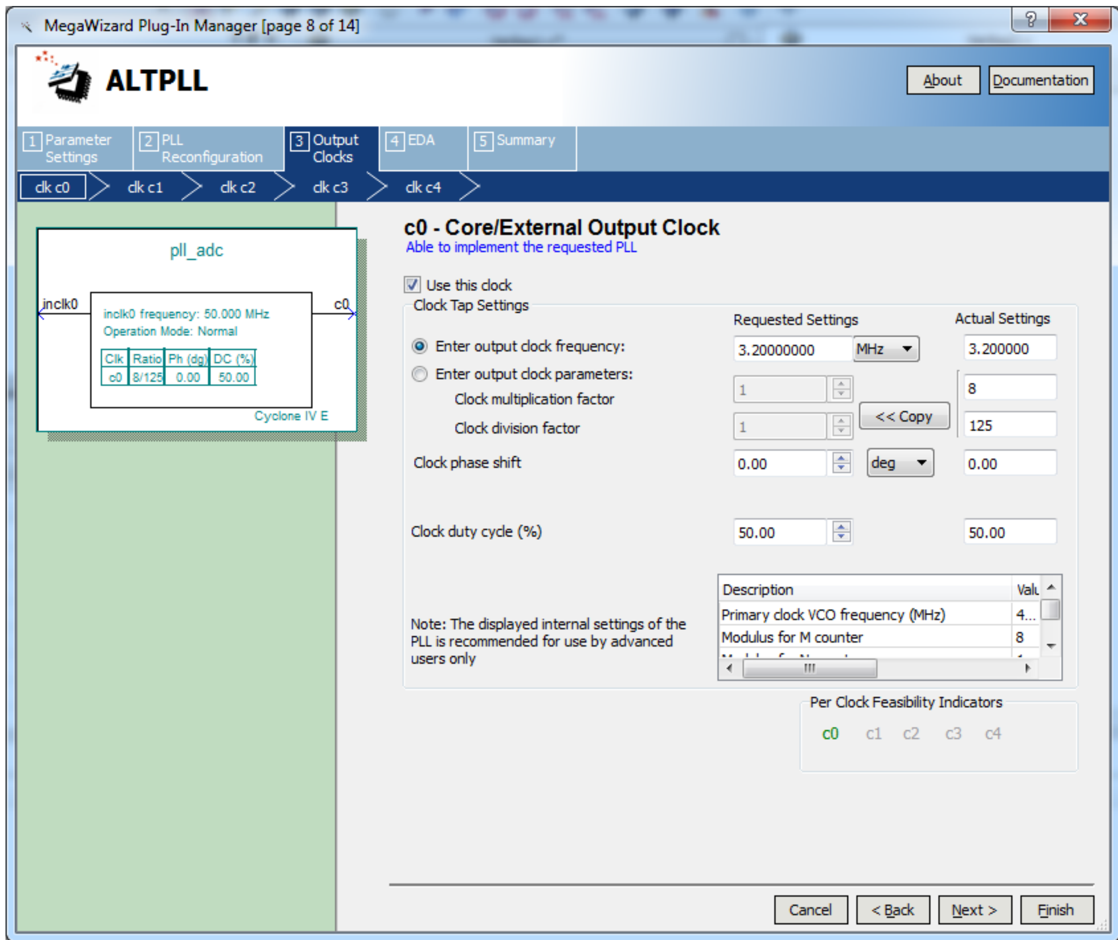


Рисунок 7.9 – Выставление частоты на выходе

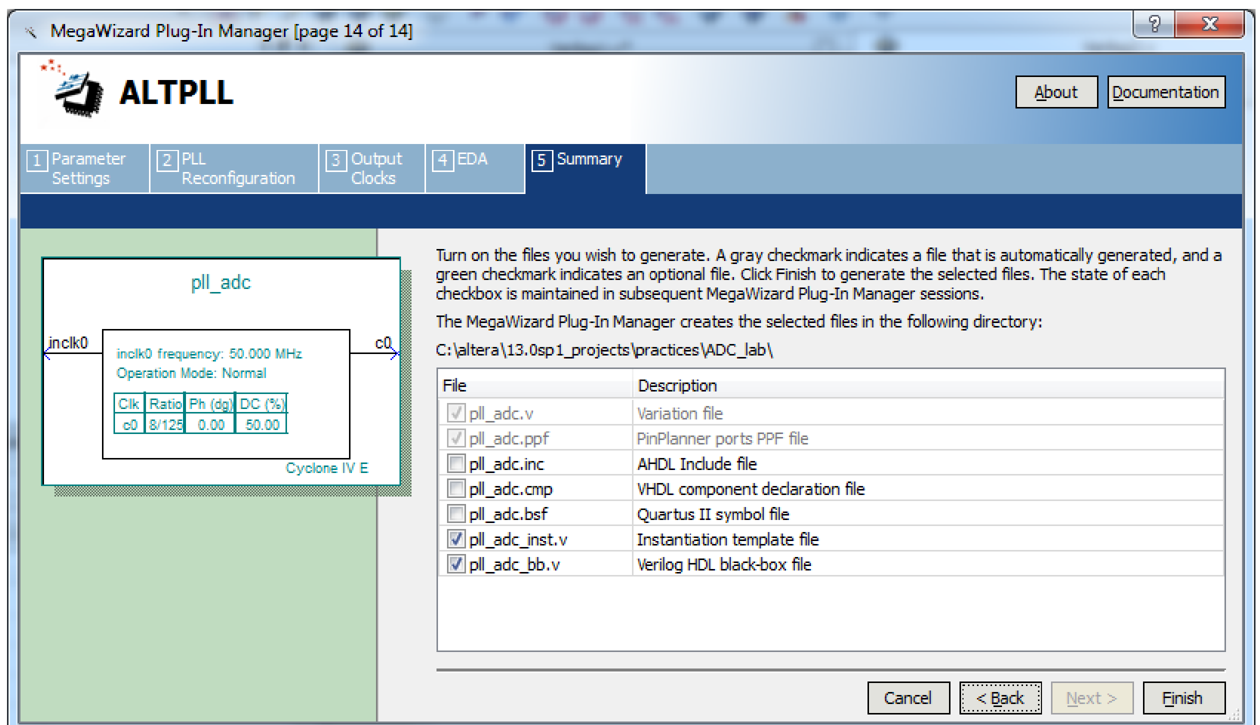
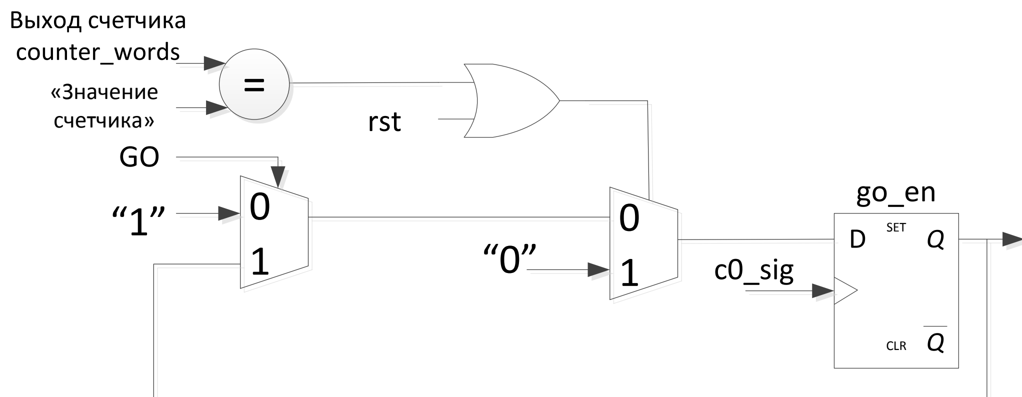


Рисунок 7.10 – Выбор необходимых файлов для синтеза

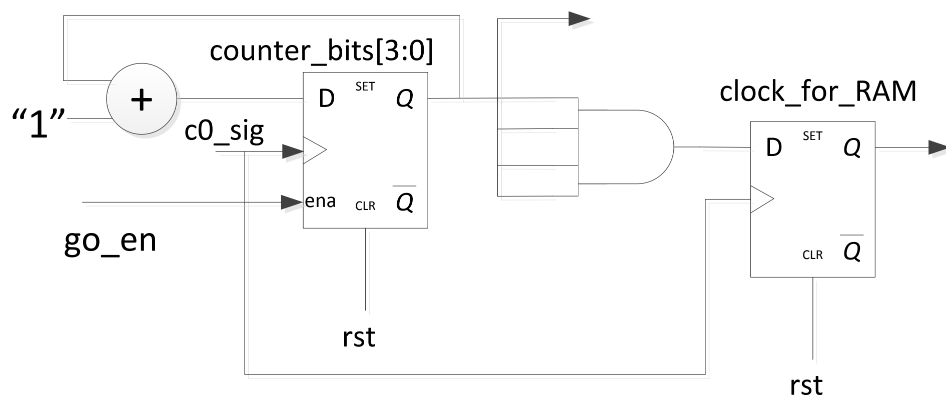
4) Настройте запуск АЦП через кнопку KEY на плате, которая уже обозначена в начале как “GO”. Создайте одноразрядный регистр (например, под именем go_en) и регистр для счетчика значений (далее counter_words, разрядность указывается преподавателем), задайте условие сброса в ноль по двум следующим событиям: счетчик counter_words досчитал до максимального значения (например counter_words == 1024) либо rst сбросили на единицу. Затем задайте условие, при котором go_en будет равен единице по событию GO == 0, так как в при нажатая кнопка соответствует нулю. Синхронизация по шине c0_sig. Схема алгоритма работы:



5) Реализовать 4-разрядный счетчик (далее counter_bits) с асинхронным сбросом (rst), с входом разрешения на запуск (go_en) и синхронизацией через c0_sig. Добавить внутрь процедурного блока того счетчика еще один одноразрядный регистр (далее clock_for_RAM) который равен единице только если все разряды счетчика counter_bits равны единицы. В упрощенном виде это пишется следующим образом:

clock_for_RAM = &counter_bits[3:0];

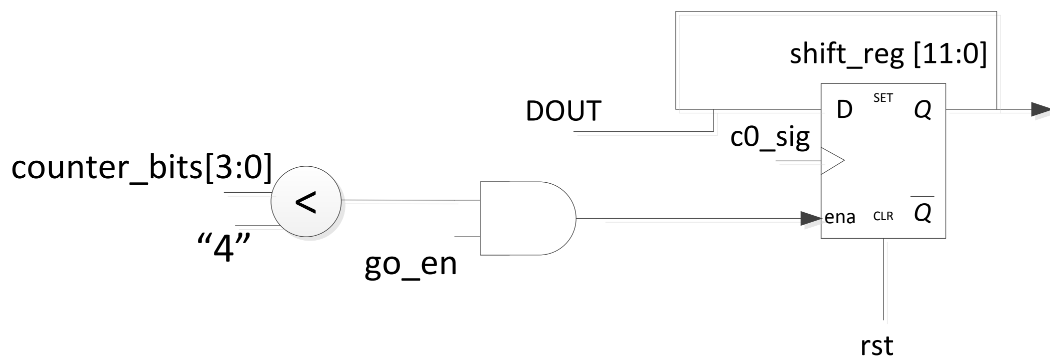
На схеме это выглядит следующим образом:



6) Реализовать счетчик *counter_words* с асинхронным сбросом *(rst)* синхронизацией через *clock_for_RAM*(обратите внимание здесь уже не *c0_sig*). Разрядность регистра зависит от варианта вашего задания.
7) Реализовать 12-ти разрядный сдвиговый регистр (далее *shift_reg*) с асинхронным сбросом (*rst*), с синхронизацией через c0_sig и с входом разрешения на запуск с выходов регистров go_en и counter_bits через условие: *go_en && counter_bits >= 4*. Таким образом, сдвиговый регистр начнет работать только после того как счетчик *counter_bits* будет больше либо равен 4 и go_en будет равен 1. Подключить к первому разряду выход АЦП DOUT следующим образом:

shift_reg[11:0] <= {shift_reg[10:0],DOUT};

Примерная схема работы:



8) Теперь нужен 12-разрядный регистр *data_to_RAM*, который бы сбрасывал значения сдвигового регистра *shift_reg* по мере его заполнения. Подключите к нему синхронизацию по шине *clock_for_RAM*,добавьте асинхронный сброс (rst) и добавьте неблокирующее присвоение *data_to_RAM* к *shift_reg*:

data_to_RAM <= shift_reg;

9) Подать на вход DIN постоянный ноль «0» (Таким образом, будет работать только первый канал АЦП). Подключить к CS выход с регистра go_en с инверсией. Напрямую подсоедините шину *SCLK* с шиной *c0_sig*. Удостоверьтесь, что вы указали все регистры и шины в вашей программе. Закончите программу, написав в конце *endmodule*. Произведите компиляцию проекта. Обозначить все пины в инструменте PIN planner. Снова прокомпилировать.
10) Теперь нужно рассмотреть сигнал на выходе с помощью инструмента SignalTap II Logic Analyzer. Для этого следуйте инструкции:
    - Запустить SignalTap II Logic Analyzer через Tools → SignalTap II Logic Analyzer

- В окне Signal configuration выберете источник тактовой частоты. Из списка Pins:all выберете SCLK.
- Выберете необходимое количество точек в окне Sample depth (примерно 2 K)
- Задать триггер по которому будут выводиться данные. Выберем для этого ту же кнопку, по которому запускается АЦП, которая обозначена вначале как GO. В окне Node из того же списка выбрать GO. Выбрать следующие настройки:

  Trigger flow control: Sequental;

  Trigger position: Pre trigger position

  Trigger conditions: 1;

  Node: GO;

  Pattern: Falling Edge
- Теперь нужно выбрать данные которые нужно рассмотреть. В окне Setup после двойного нажатия в появившимся окне из списков Pins: all и Registers: Pre-synthesis добавить следующие данные: go_en, DOUT, counter_bits, counter_words, clock_for_RAM, shift_reg, data_to_RAM.
- Сохранить и закрыть SignalTap II Logic Analyzer файл. Произвести компиляцию. Снова открыть тот файл. В окне JTAG Chain configuration выбрать файл с прошивкой и здесь же прошить плату. Подключить к ножке платы соответствующему первому каналу сигнал с генератора. Задать сигнал, соответствующий вашему варианту. После того как плата прошита нажать Run Analysis после чего он будет ждать до тех пор пока вы не нажмете на кнопку, соответствующую GO. После нажатия кнопки должны появится данные.
- Нажать правой кнопкой мыши на название data_to_RAM и задать указать отображение по уровню через Bus Display Format → Unsigned Line Chart.

11) Сделать необходимые скриншоты из инструментов SignalTap II Logic Analyzer, RTL Viewer. Сделать выводы по работе.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Ю.П.Кондратенко, В.В.Мохор, С.А.Сидоренко. Verilog-HDL для моделирования и синтеза цифровых электронных схем – Н: НГТУ, 2002

2. И.Г. Каршенбойм Краткий курс HDL, Компоненты и Технологии №1, 2008 - М: Компоненты и Технологии, 2008