Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по лабораторной работе № 8**

Дисциплина: Языки описания аппаратных средств вычислительных систем.

Выполнил студент гр. 5130901/10101 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Т. Непомнящий

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Федотов

(подпись)

Санкт-Петербург

2023

**Оглавление**

[1. Задание lab8\_1: 3](#_Toc150113798)

[1.1. Цель работы: 3](#_Toc150113799)

[1.2. Ход решения: 3](#_Toc150113800)

[Листинг кода конечного автомата из lab 5\_1 3](#_Toc150113801)

[1.3. Моделирование: 7](#_Toc150113802)

[1.4. Выводы: 7](#_Toc150113803)

[2. Задание lab8\_2: 8](#_Toc150113804)

[2.1. Задача: 8](#_Toc150113806)

[2.2. Решение: 8](#_Toc150113807)

[Код устройства сортировки на языке Verilog. Lab6\_3 8](#_Toc150113808)

[Проверка результатов сортировки 10](#_Toc150113809)

[2.3. Моделирование: 11](#_Toc150113810)

[2.4. Выводы: 12](#_Toc150113811)

[3. Задание lab8\_3: 13](#_Toc150113812)

[3.1. Задача: 13](#_Toc150113814)

[3.2. Решение: 13](#_Toc150113815)

[Преобразователь в код 1 из N трехразрядного числа 13](#_Toc150113816)

[3.3. Моделирование: 14](#_Toc150113817)

[Модуль тестирования lab8\_3 14](#_Toc150113818)

[3.4. Моделирование на ПЛИС: 14](#_Toc150113819)

[3.5. Выводы: 15](#_Toc150113820)

# Задание lab8\_1:

## Цель работы:

На языке Verilog, опишите тест 1 класса для проверки конечного автомата, созданного в lab5\_1:

Изображение выглядит как текст, диаграмма, линия, Шрифт

Автоматически созданное описание

*Рис. 1.1 - Конечный автомат. Lab5\_1*

## Ход решения:

Для тестирования используется конечный автомат, разработанный в lab5\_1:

### Листинг кода конечного автомата из lab 5\_1

**module** lab5\_1

**(**

**input** clk**,** srst\_in**,**

**input** **[**3**:**0**]** din**,**

**output** **reg** **[**7**:**0**]** q

**);**

**reg** temp\_srst**,** srst**;**

**reg** **[**3**:**0**]** temp\_din**,** d**;**

**always** **@** **(posedge** clk**)**

**begin**

temp\_din **<=** din**;**

d **<=** temp\_din**;**

**end**

**always** **@** **(posedge** clk**)**

**begin**

temp\_srst **<=** srst\_in**;**

srst **<=** temp\_srst**;**

**end**

**reg** **[**1**:**0**]** state**;**

**parameter** s0 **=** 0**,** s1 **=** 1**,** s2 **=** 2**;**

**always** **@** **(posedge** clk **or** **negedge** srst**)**

**begin**

**if** **(!**srst**)** state **<=** s0**;**

**else**

**case** **(**state**)**

s0**:**

**if** **(**d **==** 4'h1**)** state **<=** s1**;**

s1**:**

**if** **(**d **==** 4'h2**)** state **<=** s2**;**

s2**:**

**if** **(**d **==** 4'h4**)** state **<=** s1**;**

**else** **if** **(**d **==** 4'h8**)** state **<=** s0**;**

**default** state **<=** s0**;**

**endcase**

**end**

**always** **@** **(**state**)**

**case** **(**state**)**

s0**:** q **=** 8'h00**;**

s1**:** q **=** 8'h55**;**

s2**:** q **=** 8'hff**;**

**endcase**

**endmodule**

Разрабатываемый тестовый модуль назовем lab5\_1\_tb, вызовем в нем тестируемый модуль, создав все необходимые входы и выходы:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 1.2 - Вызов тестируемого модуля.

Создадим clk т. к. исследуемое устройство – синхронное. Поскольку в тестируемом модуле необходимо несколько раз перебирать числа от 0 до 15, за исключением 1-2 штук (для того, чтоб проверить, что переход между состояниями происходит в соответствии с схемой только) напишем отдельный task с необходимым циклом:

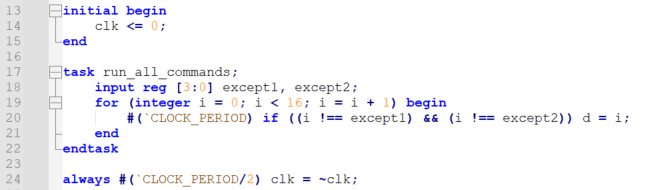


Рис. 1.3 - Task с циклом для тестирования переходов между состояниями.

Этот Task имеет 2 входа:

* two\_exceptions – флаг. Если равен нулю, то второе число в массиве except будет игнорироваться, в противном случае будут пропускаться оба числа из массива except.
* [3:0] except [0:1] – массив двух четырехбитных чисел. Первое число в массиве всегда пропускается в цикле, второе – в зависимости от флага two\_exceptions.

Реализуем сами тестовые воздействия:

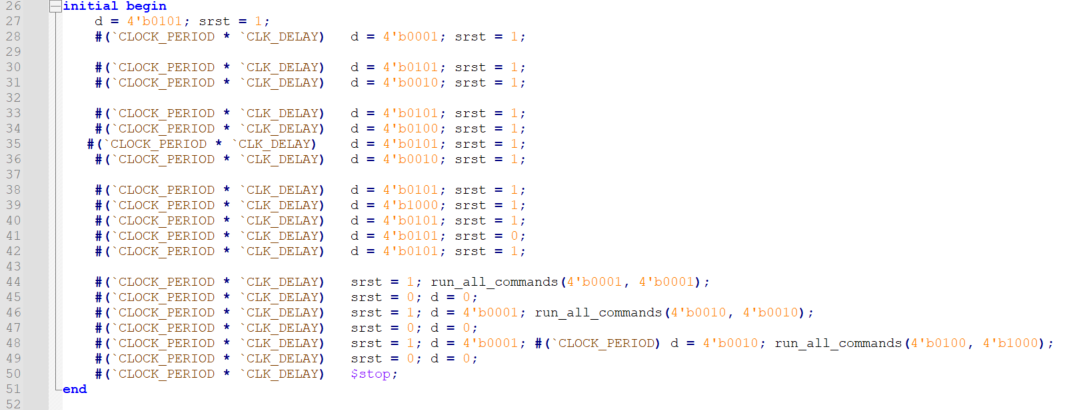


Рис. 1.4. Тестовые воздействия на исследуемое устройство.

Подготовим файлы к последующему моделированию. Последовательность шагов представлена на рисунках 1.5 - :

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

Автоматически созданное описание

Рис. 1.5 – Создание библиотеки проекта

Изображение выглядит как текст, снимок экрана, число, дисплей

Автоматически созданное описание

Рис. 1.6 – Создание библиотеки проекта

Изображение выглядит как текст, Шрифт, линия, программное обеспечение

Автоматически созданное описание

Рис. 1.7 – Добавление файлов

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

Автоматически созданное описание

Рис. 1.8 – Компиляция файлов

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

Автоматически созданное описание

Рис. 1.9 – Симуляция файлов

Данные тесты проверяют все возможные переходы в устройстве, а также корректность работы синхронного сброса.

## Моделирование:

Запустим полученный тестовый файл, добавив на wave помимо clk, srst\_in, din и q некоторые переменные из тестируемого модуля, а именно текущее состояние, d, srst:

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

Автоматически созданное описание

Рис. 1.10 - Результат выполнения тестового файла.

Как видно, устройство работает корректно:

* Выполняет переходы только от требуемых значений.
* Выполняет синхронный сброс в соответствии с ТЗ.

## Выводы:

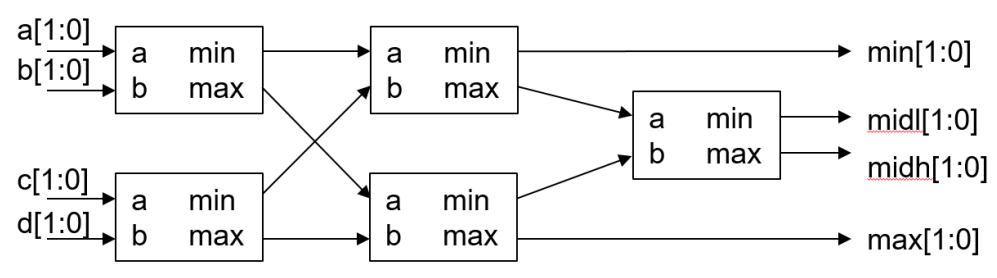
В заключении, созданный тест первого класса на языке Verilog для проверки конечного автомата из лабораторной работы lab5\_1 представляет собой важный инструмент для начальной проверки функциональности устройства. Этот тест позволяет убедиться, что конечный автомат правильно реагирует на основные входные сигналы и выполняет ожидаемые действия. Он служит важным этапом в разработке и верификации устройства, позволяя выявить и устранить ошибки на ранних этапах. Однако для более глубокой проверки рекомендуется также разработать тесты более высоких классов, чтобы охватить более широкий спектр возможных сценариев работы устройства.

# Задание lab8\_2:



## Задача:

На языке Verilog, опишите тест 2 класса для проверки устройства сортировки, созданного в lab6\_3:



*Рис. 2.1 - Устройство сортировки. Lab6\_3*

*Дополнение:*

* Подаваемые на вход устройства данные считываются из файла с помощью $readmemh (или $readmemb)
* Ожидаемые данные считываются из файла, используя расширения стандарта 2001 (т.е. без $readmemh (или $readmemb))

## Решение:

Для тестирования используется устройство сортировки, разработанное в lab6\_3:

### Код устройства сортировки на языке Verilog. Lab6\_3

**module** lab6\_3

**#(parameter** W**=**2**,** TYPE**=**"lab6\_1"**)**

**(**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"3.3-V LVTTL\""**,** chip\_pin **=** "88, 89" **\*)**

**input** **[**W**-**1**:**0**]** a**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"3.3-V LVTTL\""**,** chip\_pin **=** "90, 91" **\*)**

**input** **[**W**-**1**:**0**]** b**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"3.3-V LVTTL\""**,** chip\_pin **=** "49, 46" **\*)**

**input** **[**W**-**1**:**0**]** c**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"3.3-V LVTTL\""**,** chip\_pin **=** "25, 24" **\*)**

**input** **[**W**-**1**:**0**]** d**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"3.3-V LVTTL\""**,** chip\_pin **=** "23" **\*)**

**input** clk**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"2.5 V\""**,** chip\_pin **=** "64" **\*)**

**input** rst**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"2.5 V\""**,** chip\_pin **=** "65, 66" **\*)**

**output** **reg** **[**W**-**1**:**0**]** min**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"2.5 V\""**,** chip\_pin **=** "67, 68" **\*)**

**output** **reg** **[**W**-**1**:**0**]** midl**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"2.5 V\""**,** chip\_pin **=** "69, 70" **\*)**

**output** **reg** **[**W**-**1**:**0**]** midh**,**

**(\*** altera\_attribute **=** "-name IO\_STANDARD \"2.5 V\""**,** chip\_pin **=** "71, 72" **\*)**

**output** **reg** **[**W**-**1**:**0**]** max

**);**

**wire** **[**W**-**1**:**0**]** min\_1**,** max\_1**,**

min\_2**,** max\_2**,**

min\_3**,** max\_3**,**

min\_**,** midl\_**,** midh\_**,** max\_**;**

**reg** **[**W**-**1**:**0**]** a\_**,** b\_**,** c\_**,** d\_**;**

**always** **@(posedge** clk**)**

**begin**

**if** **(!**rst**)**

**begin**

a\_ **<=** 0**;**

b\_ **<=** 0**;**

c\_ **<=** 0**;**

d\_ **<=** 0**;**

min **<=** 0**;**

midl **<=** 0**;**

midh **<=** 0**;**

max **<=** 0**;**

**end**

**else**

**begin**

a\_ **<=** a**;**

b\_ **<=** b**;**

c\_ **<=** c**;**

d\_ **<=** d**;**

min **<=** min\_**;**

midl **<=** midl\_**;**

midh **<=** midh\_**;**

max **<=** max\_**;**

**end**

**end**

**generate**

**if** **(**TYPE **==** "lab6\_1"**)**

**begin** **:**lab6\_1

lab6\_1 **#(.**W**(**W**))** lab6\_1\_1 **(.**a**(**a\_**),** **.**b**(**b\_**),** **.**min**(**min\_1**),** **.**max**(**max\_1**));**

lab6\_1 **#(.**W**(**W**))** lab6\_1\_2 **(.**a**(**c\_**),** **.**b**(**d\_**),** **.**min**(**min\_2**),** **.**max**(**max\_2**));**

lab6\_1 **#(.**W**(**W**))** lab6\_1\_3 **(.**a**(**min\_1**),** **.**b**(**min\_2**),** **.**min**(**min\_**),** **.**max**(**max\_3**));**

lab6\_1 **#(.**W**(**W**))** lab6\_1\_4 **(.**a**(**max\_1**),** **.**b**(**max\_2**),** **.**min**(**min\_3**),** **.**max**(**max\_**));**

lab6\_1 **#(.**W**(**W**))** lab6\_1\_5 **(.**a**(**min\_3**),** **.**b**(**max\_3**),** **.**min**(**midl\_**),** **.**max**(**midh\_**));**

**end**

**else**

**begin** **:**lab6\_2

lab6\_2 **#(.**W**(**W**))** lab6\_2\_1 **(.**a**(**a\_**),** **.**b**(**b\_**),** **.**min**(**min\_1**),** **.**max**(**max\_1**));**

lab6\_2 **#(.**W**(**W**))** lab6\_2\_2 **(.**a**(**c\_**),** **.**b**(**d\_**),** **.**min**(**min\_2**),** **.**max**(**max\_2**));**

lab6\_2 **#(.**W**(**W**))** lab6\_2\_3 **(.**a**(**min\_1**),** **.**b**(**min\_2**),** **.**min**(**min\_**),** **.**max**(**max\_3**));**

lab6\_2 **#(.**W**(**W**))** lab6\_2\_4 **(.**a**(**max\_1**),** **.**b**(**max\_2**),** **.**min**(**min\_3**),** **.**max**(**max\_**));**

lab6\_2 **#(.**W**(**W**))** lab6\_2\_5 **(.**a**(**min\_3**),** **.**b**(**max\_3**),** **.**min**(**midl\_**),** **.**max**(**midh\_**));**

**end**

**endgenerate**

**endmodule**

Разрабатываемый тестовый модуль назовем lab6\_3\_tb, вызовем в нем тестируемый модуль, создав все необходимые входы и выходы:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 2.2 - Вызов тестируемого модуля.

В тестируемом устройстве, используя параметр TYPE задается итоговая архитектура устройства. Выполним тестирование обеих архитектур, для этого вызываем модуль дважды, задав другое значение TYPE.

Создадим clk т.к. исследуемое устройство – синхронное:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 2.3 - Clk для исследуемого устройства.

Чтоб не запутаться в ходе тестирования, в качестве значения периода зададим локальный параметр CLK\_PERIOD.

Основная логика проверки устройства приведена ниже:

### Проверка результатов сортировки

**wire** **[**1**:**0**]** min\_2**,** midl\_2**,** midh\_2**,** max\_2**;**

lab6\_3 **#(.**TYPE**(**"lab6\_1"**))** uut\_lab\_6\_3\_1 **(**a**,** b**,** c**,** d**,** clk**,** rst**,** min\_1**,** midl\_1**,** midh\_1**,** max\_1**);**

lab6\_3 **#(.**TYPE**(**"lab6\_2"**))** uut\_lab\_6\_3\_2 **(**a**,** b**,** c**,** d**,** clk**,** rst**,** min\_2**,** midl\_2**,** midh\_2**,** max\_2**);**

**initial** **begin**

clk **=** 1'b0**;**

**forever** **#(**CLK\_PERIOD **/** 2**)** clk **=** **!**clk**;**

**end**

**localparam** COUNT\_TESTS **=** 100**;**

**reg** **[**1**:**0**]** A **[**0**:**4 **\*** COUNT\_TESTS **-** 1**];**

**integer** answers\_file**,** char**;**

**reg** **[**1**:**0**]** min\_ans**,** midl\_ans**,** midh\_ans**,** max\_ans**;**

**initial** **begin**

rst **=** 1'b1**;**

// read tests

$readmemb**(**"data\_in.txt"**,** A**);**

answers\_file **=** $fopen**(**"data\_out.txt"**,** "r"**);**

`define eof 32'hffff\_ffff

char **=** $fgetc**(**answers\_file**);**

**for** **(int** i **=** 0**;** i **<** COUNT\_TESTS **&&** char **!=** `eof**;** i**++)** **begin**

$ungetc**(**char**,** answers\_file**);**

$fscanf**(**answers\_file**,** "%b"**,** min\_ans**);**

$fscanf**(**answers\_file**,** "%b"**,** midl\_ans**);**

$fscanf**(**answers\_file**,** "%b"**,** midh\_ans**);**

$fscanf**(**answers\_file**,** "%b"**,** max\_ans**);**

a **=** A**[**i **\*** 4**];**

b **=** A**[**i **\*** 4 **+** 1**];**

c **=** A**[**i **\*** 4 **+** 2**];**

d **=** A**[**i **\*** 4 **+** 3**];**

**#(**CLK\_PERIOD **\*** 2**);**

**if** **(**min\_ans **!=** min\_1 **||** midl\_ans **!=** midl\_1 **||** midh\_ans **!=** midh\_1 **||** max\_ans **!=** max\_1**)** **begin**

$display**(**"Error test lab6\_3\_1; id: %0d. \nExcepted:\t\t%b, %b, %b, %b\nReal:\t\t%b, %b, %b, %b"**,** i**,**

min\_ans**,** midl\_ans**,** midh\_ans**,** max\_ans**,**

min\_1**,** midl\_1**,** midh\_1**,** max\_1**);**

$stop**;**

**end**

**if** **(**min\_ans **!=** min\_2 **||** midl\_ans **!=** midl\_2 **||** midh\_ans **!=** midh\_2 **||** max\_ans **!=** max\_2**)** **begin**

$display**(**"Error test lab6\_3\_2; id: %0d. \nExcepted: %b, %b, %b, %b\nReal: %b, %b, %b, %b"**,** i**,**

min\_ans**,** midl\_ans**,** midh\_ans**,** max\_ans**,**

min\_2**,** midl\_2**,** midh\_2**,** max\_2**);**

$stop**;**

**end**

char **=** $fgetc**(**answers\_file**);**

**end**

$display**(**"All test have been passed!"**);**

$fclose**(**answers\_file**);**

$stop**;**

**end**

**endmodule**

Здесь локальный параметр COUNT\_TESTS задает число тестов во входном файле. Именно используя этот параметр выбирается размер массива А, в который записываются входные данные из файла data\_in. Так же этот параметр влияет на цикл, в котором происходит считывание ответа и сравнение его с реальным результатом.

Однако важно, что если увеличивать значение COUNT\_TESTS, но не добавить значения в файл answers\_file.txt цикл остановится по eof.

Стоит отметить, что из-за дополнительных регистров на входе и выходе тестируемого устройства, было принято решение добавить дополнительную задержку, чтоб дождаться интересующего результата.

Естественно, можно было поменять реализацию и каждый такт подавать новые данные, а сравнение реализовать другим образом, однако в силу простоты текущей реализации, было принято решение её оставить.

## Моделирование:

Создадим 4 тестовых последовательности, входные данные выглядят таким образом:

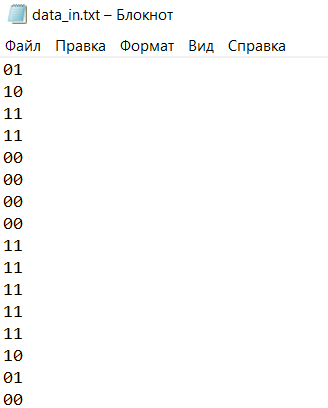


Рис. 2.4 - data\_in.txt

Данные, с которыми сравниваем выходные такие:

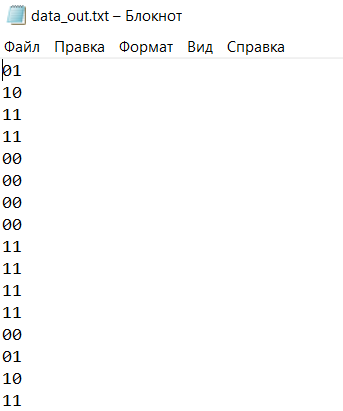


Рис. 2.5 - data\_out.txt

Запустим полученный тестовый файл, добавив на wave все интересующие нас значения:

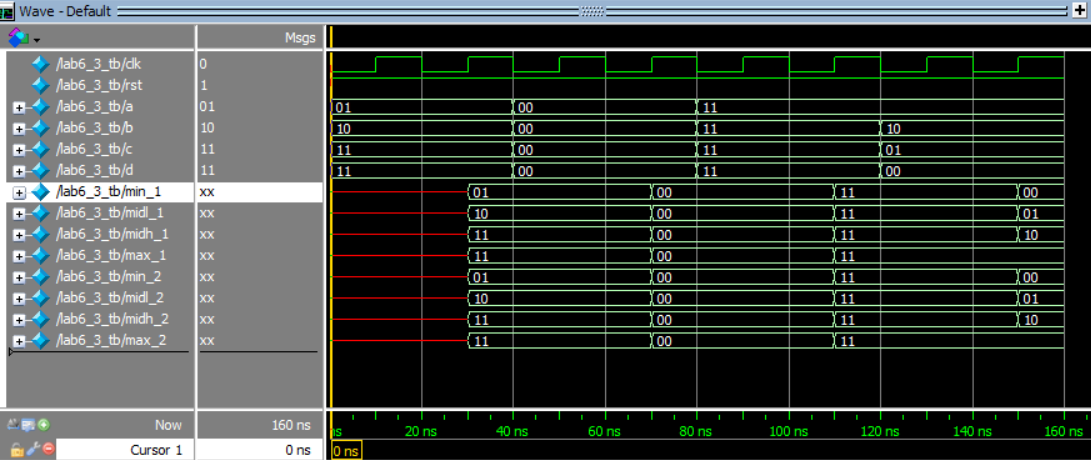


Рис. 2.6 - Результат выполнения тестового файла.

Как видно, устройство работает корректно, стоит отметить, что тесты не являются полными т. к. rst всегда остается = 1, а также в приведенных файлах всего 4 разных теста.

## Выводы:

В рамках лабораторной работы lab8\_2 были приобретены важные навыки по разработке тестов второго типа, которые опираются на данные, полученные из файлов. Эти навыки предоставляют возможность удобно создавать тестовые сценарии, а также проверять корректность функционирования устройства.

Работа с файлами в языке Verilog оказывается эффективной операцией, которая позволяет быстро создавать тестовые случаи, основанные на реальных данных или заданных шаблонах. Этот подход существенно упрощает процесс верификации и анализа результатов тестирования, а также обнаружение потенциальных ошибок и несоответствий в работе цифровых устройств.

Способность взаимодействовать с файлами в Verilog не только облегчает разработку тестов, но также усиливает способность анализа и контроля процесса верификации. Эти навыки содействуют более надежному и эффективному развитию цифровых систем и устройств.

# Задание lab8\_3:



## Задача:

На языке Verilog разработать преобразователь двоичных чисел разрядностью 3 в код 1 из N. Устройство должно работать на ROM памяти, инициализацию которой выполнить из файла.

*Входы:*

* [2:0] da – вход множимого, на входе следует использовать регистр.
* clk – вход тактового сигнала.

*Выходы:*

* [N-1:0] q – выходы.

## Решение:

Реализуем требуемое устройство на языке SystemVerilog:

### Преобразователь в код 1 из N трехразрядного числа

**module** lab8\_3**(**

**input** **[**2**:**0**]** da**,**

**input** clk**,**

**output** **reg** **[**7**:**0**]** q

**);**

**reg** **[**2**:**0**]** address**;**

**reg** **[**7**:**0**]** rom **[**0**:**7**];**

**initial** **begin**

$readmemb**(**"rom\_init\_values.txt"**,** rom**);**

**end**

**always** **@(posedge** clk**)** **begin**

address **<=** da**;**

q **<=** rom**[**address**];**

**end**

**endmodule**

Файл mem.txt имеет следующий вид:

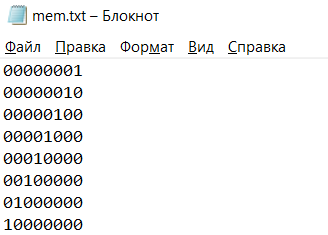


Рис. 3.2 - Содержимое файла mem.txt

## Моделирование:

Для моделирования разработанного устройства, напишем тестовый модуль 1 класса:

### Модуль тестирования lab8\_3

**module** lab8\_3\_tb**;**

**reg** **[**2**:**0**]** da**;**

**reg** clk**;**

**wire** **[**7**:**0**]** q**;**

lab8\_3 dut**(.**da**(**da**),** **.**clk**(**clk**),** **.**q**(**q**));**

**initial** **begin**

clk **=** 1'b0**;**

**forever** **#**5 clk **=** **!**clk**;**

**end**

**initial** **begin**

**#**5**;**

da **=** 0**;**

**#**100**;**

da **=** 1**;**

**#**100

da **=** 2**;**

**#**100**;**

da **=** 3**;**

**#**100**;**

da **=** 4**;**

**#**100**;**

da **=** 5**;**

**#**100**;**

da **=** 6**;**

**#**100**;**

da **=** 7**;**

**#**100**;**

$stop**;**

**end**

**endmodule**

Этот модуль перебирает все возможные входные данные на da, тем самым проводя полный тест разработанного модуля.

Добавим на waveform интересующие входы и выходы и получим:

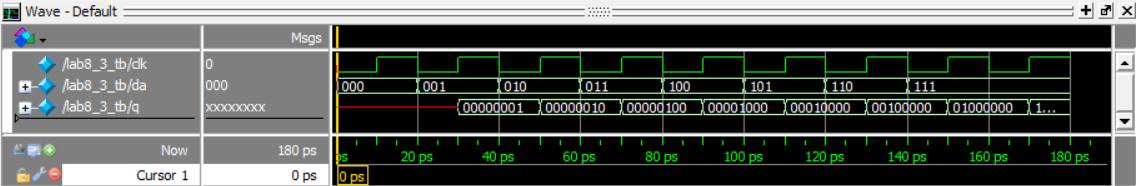


Рис. 3.4 - Результат моделирования в ModelSim.

Полученный вывод соответствует ожиданиям, что говорит о корректной работе устройства.

## Моделирование на ПЛИС:

Выполним моделирование на ПЛИС, входы и выводы платы заданы через altera\_attribute. Выполним компиляцию и проверим, что входы и выходы действительно получили требуемые пины:

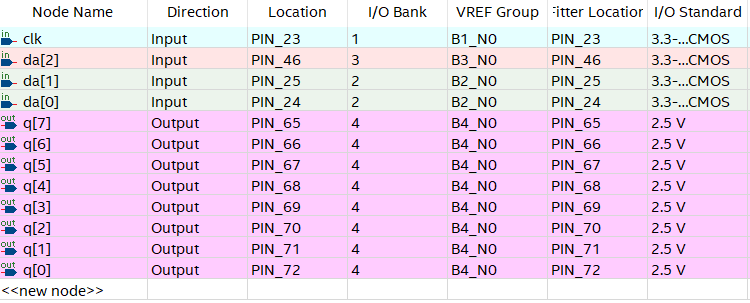


Рис. 3.5. Pin Planner.

В ходе моделирования на ПЛИС видно, что устройство работает корректно.

## Выводы:

В ходе лабораторной работы lab8\_3 было разработано устройство, которое способно преобразовывать двоичные числа в код 1 из N. Это устройство использует RAM для хранения данных и инициализируется с использованием $readmemb. Процесс разработки и отладки этого устройства значительно упрощается при использовании инструмента Model Sim и директив для чтения файлов.

Разработанное устройство может найти применение в различных цифровых приложениях, где требуется перекодирование данных для облегчения их обработки и управления. Примеры включают цифровую коммутацию, управление периферийными устройствами, адресацию памяти, кодирование событий и сигналов, а также цифровую обработку сигналов. Такие устройства полезны для оптимизации обработки данных и управления информацией в различных цифровых системах.