Лабораторная работа №2
Исследование комбинационных схем.

1. Цель работы:

Познакомиться с особенностями HDL языков и отличием Verilog от VHDL. Изучить основные элементы Verilog – типы данных, идентификаторы, комментарии, описание шин, структура модулей. Научиться создавать описание работы комбинационных схем с помощью Verilog и моделировать их работу в ModelSim.

2. Теоретические сведения

Языки описания аппаратуры (**HDL**, **H**ardware **D**escription **L**anguage) могут быть использованы на всех этапах разработки цифровых электронных систем. Они применяются на этапах проектирования, верификации, синтеза и тестирования аппаратуры, а также для передачи данных о проекте, его модификации и сопровождении.
Языки описания аппаратуры, в основном, используются для проектирования программируемых логических устройств (PLD – Programmable Logic Devices) различного уровня сложности, вентильных программируемых матриц (FPGA – Field Programmable Gate Array). На сегодняшний день нашли применение несколько таких языков. Наиболее популярные из них – Абель, Palasm и Cupl (используются для устройств малой степени сложности), Verilog и VHDL (для сложных PLD и FPGA устройств).
Преимущества **HDL**:

- По имеющемуся **HDL** описанию можно синтезировать принципиальную схему устройства – генерация **RTL** описания (**RTL** – **R**egister **T**ransfers **L**evel).
- Схему, описанную с помощью **HDL,** проще документировать.
- **HDL** позволяют реализовать как структурное описание узлов и цепей, так и поведенческое описание. Они имеют возможность вызывать функции, написанные на других языках программирования (например, на “C”).
- **HDL** позволяют описывать такие специфические для цифровых схем понятия, как временные задержки и параллельное выполнение операций.

HDL позволяют создавать принципиальные схемы, иерархические блоки, алгоритмы функционирования устройств, а также тестовые файлы (test-bench) для проверки созданных модулей.

2.1. Язык Verilog

Verilog — это язык описания аппаратуры, используемый для разработки и моделирования электронных систем. Этот язык (также известный как ***Verilog HDL***) позволяет осуществить проектирование, верификацию и реализацию (например, в виде **СБИС**) аналоговых, цифровых и смешанных электронных систем на различных уровнях абстракции.

Verilog был разработан фирмой **Gateway Design Automaton** для использования внутри компании. Затем, в 1989 г., **Verilog** был открыт для общего использования. Стандарт для данного языка был принят в 1995 году (IEEE1364-1995).
Verilog имеет простой синтаксис, сходный с языком программирования “C”. Малое количество служебных слов и простота основных конструкций упрощают изучение и позволяют создавать эффективные приложения. На описание одной и той же конструкции в **Verilog** потребуется в 3–4 раза меньше символов, чем в **VHDL**.
Интересной особенностью **Verilog** является наличие стандарта **PLI** (Program Language Interface), который позволяет включать функции, написанные пользователем (например, на C), в код симулятора.

Краткие теоретические сведения по языку Verilog

Поддерживаемые типы данных:

- **integer** – 32-х разрядное целое число со знаком;

- **real** – 64-х разрядное число с плавающей точкой. Синтезирующими САПР **не поддерживается**;
- **time** – 64-х разрядное целое беззнаковое число, которое применяется встроенными функциями для моделирования времени;

Создавать свои типы данных в Verilog **нельзя**.

Кроме основных типов данных, которые присущи всем языкам программирования, в HDL вводится новое понятие – **сигнал**.

Сигналы бывают двух основных типов:

- **wire** – цепи;
- **reg** – регистры.

Отличие сигналов **wire** от сигналов **reg** состоит в том, что **reg** способен сохранять присвоенное значение (работает как переменная в языках программирования или как устройство последовательного типа). К сигналам типа **wire** требуется прилагать непрерывное воздействие (как устройство комбинационного типа). То есть **wire** моделирует связь (или устройство), которая переходит в неопределенное состояние при отключении входного воздействия. Существуют также типы **wand**, **wor**, **tri0**, **tri1**, **triand**, **trior**, **trireg** для моделирования различных типов связей (**wand** – wired and или «монтажное И», **tri0** – подтягивающий резистор к нулевому уровню, **trireg** – накопительная емкость, и т.п.), но такие цепи встречаются редко и используются только для моделирования.

Идентификаторы:

Идентификаторы в Verilog **являются чувствительными к прописным и строчным символам** и подчиняются обычным правилам: **не могут** начинаться с цифры или знака $ и **могут** содержать буквы, цифры, $, и символ подчеркивания.

Комментарии:

Комментарии в языке Verilog бывают двух типов и полностью соответствуют комментариям языков программирования высокого уровня (C++, Java, C#):

```
1  // - это комментарий (однострочный)
2  /*
3  и это комментарий (групповой)
4  */
```

Простейший пример кода на Verilog:

```
1  integer i, j, k; // объявление переменных
2  time start, duration;
3  /* далее следуют объявления
4  однобитовых сигналов */
5  wire a, b, c;
6  reg store, ff, A; // reg A не совпадает с wire a
7
8  Описание шин (групп сигналов):
9  Для описания шин или регистров неединичной разрядности используются
10 диапазоны вида [n:m], где m и n целые числа.
11 wire [7:0] data_bus;
12 reg [3:0] high_nibble, low_nibble; // два 4-х разрядных регистра
13
14 Массивы в Verilog не поддерживаются, но существует понятие "памяти".
15 Память соответствует двумерному массиву.
16
17 reg [8:0] Fifo [31:0]; // память, состоящая из 9 32-х разрядных ячеек
18
```

Допустимые значения для сигналов:
Всего существует четыре типа значений, которые могут принимать сигналы (`wire` и `reg`): **0, 1, z, x**. Первые три соответствуют двум логическим уровням и состоянию с высоким импедансом. Четвертый (**x**) означает неопределенное состояние и используется при моделировании неинициализированных сигналов, при возникновении конфликтов (два выхода с противоположными состояниями соединены вместе), указания нестабильных состояний триггеров (при нарушении временных соотношений между входами данных и тактовым входом) и т.п. Другими словами – во всех случаях, когда моделирующая программа не может определить значение для данного сигнала.
Для указания значения многоразрядных сигналов (констант, переменных) используются следующие конструкции:

1. **1'bz** – одноразрядный высокоимпедансный сигнал;
2. **10'd1_000** – десятиразрядное число 1000 записанное в десятичном виде (символ подчеркивания игнорируется);
3. **4'bx01z** – четырехразрядный двоичный сигнал с неопределенным старшим битом, высокоимпедансным младшим, вторым и третьим в состоянии логических «1» и «0», соответственно.

В общем виде – вначале указывается разрядность сигнала, потом одинарная кавычка ' (не путать с апострофом), далее основание системы счисления (**`b,o,d,h`**) и цифры, использующиеся в данной системе счисления, задающие значение сигнала.

Для двоичной системы допускается использование символов z и x.
Символ подчеркивания служит для улучшения восприятия записи и **игнорируется** при синтезе и моделировании. Использование констант без указания разрядности не желательно, так как по умолчанию константа воспринимается с разрядностью, равной 32 бита.
Данные типа **`integer`** могут присваиваться регистрам.

Файлы языка Verilog
Имеют расширение "`.v`". Например, "`module1.v`", "`FPU.v`", "`generator.v`".

Структурное описание:
Основной структурной единицей программы на языке Verilog является **module**. Модуль описывается ключевыми словами **module** – **endmodule**. В одном программном файле может присутствовать несколько модулей. Модули не могут быть вложенными. Другие модули могут подключаться к входным и выходным портам модуля, образуя иерархическую структуру. При запуске компилятора языка Verilog, он формирует иерархическое дерево проекта из всех входящих модулей и находит модуль верхнего уровня иерархии. **Важно помнить**, что имя модуля верхнего уровня иерархии должно совпадать с именем файла, в котором этот модуль описан!

Общая структура модуля в языке Verilog:

```
1   module SomeModule( Param1, Param2, Param3,…,ParamN );
2   input Param1; // входной порт с именем Param1
3   output Param2; // выходной порт с именем Param2
4   inout Param3; // двунаправленный порт (вход/выход) с именем Param3
5   ……….
6   `timescale 1ns/10ps
7   ……….
8   `include "somefile.v"
9   `define nc @( negedge clk )
10  /*
11
12  Тут может быть ваш код!!!
13
14  */
15
16  endmodule
```

Итак, сразу же после директивы **module**, следует имя модуля, а за ним – в круглых скобках, перечислены имена **портов** модуля (интерфейс модуля с внешним миром). Порты могут быть входными – **input**, выходными – **output** или двунаправленными – **inout**. После описания портов следует описание директив препроцессора '**include**, **'define**, и др. которые совершенно аналогичны директивам препроцессора языка С (**Один важный момент** – вышеописанные директивы начинаются с символа апострофа, а не одинарной кавычки).
Еще один нюанс связан с использованием директивы '**include**. Как известно, эта директива применяется для включения текста одного файла в другой. **Однако** в Verilog модули не могут объявляться внутри другого модуля (не могут быть вложенными). Поэтому, директиву '**include** можно использовать либо вне модуля, либо включать в модуль код, не содержащий описания модулей.
Директиву **'timescale** мы рассмотрим в следующей лабораторной работе.

Ключевое слово assign
Ключевое слово **assign** используется для присвоения значения сигналу типа **wire**. Данный оператор не используется в процедурных блоках. Его синтаксис следующий:

```
1   assign <wire_name> = <expression>;
```

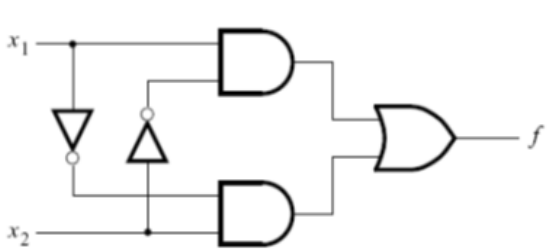
Параметр <expression> может быть представлен логическим выражением. Параметр <wire_name> представляет собой имя сигнала типа **wire**. Как только значение <expression> изменяется, полученное новое значение <expression> присваивается сигналу с именем <wire_name> Подробнее оператор **assign** будет рассмотрен в лабораторных работах.

3. Порядок выполнения работы

Опишем на языке Verilog схему исключающего «ИЛИ» (XOR).
Таблица истинности для XOR:

x1	x2	f
0	0	0
0	1	1
1	0	1
1	1	0

Схема, реализующая данную функцию:



3.1. Работа в среде ModelSim.

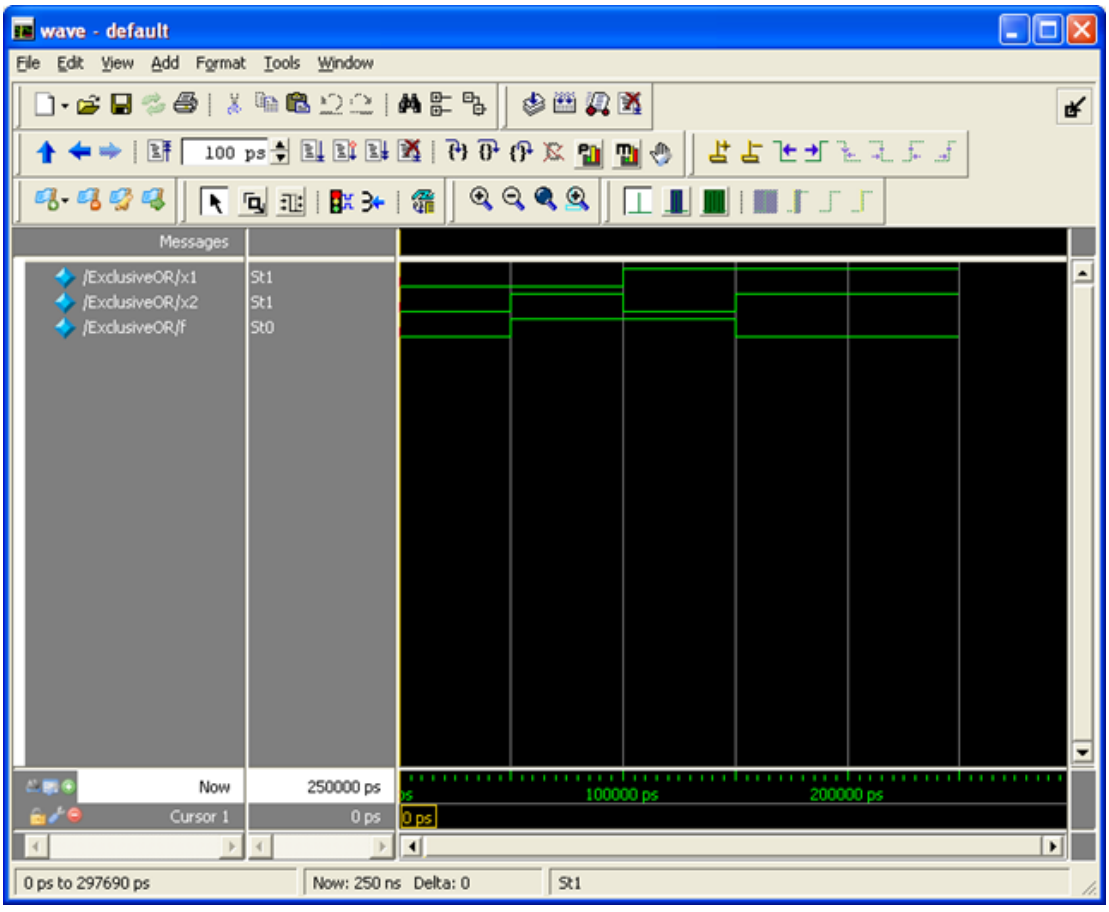
1. Создайте новый проект в среде ModelSim.
2. Создайте новый файл на языке Verilog с описанием работы устройства (команда **Create New File** диалогового окна **Add items to the Project**). Укажите имя файла «ExclusiveOR» и язык описания файла Verilog в открывшемся диалоговом окне **Create Project File**. Закройте диалоговое окно **Add items to the Project**
3. Откройте текстовый редактор (двойное нажатие левой кнопки мыши на имени файла). Наберите следующий текст:

```
1  module ExclusiveOR(x1,x2,f);
2  input x1,x2; // входные порты с именами x1 и x2
3  output f; // выходной порт с именем f
4
5  assign f = (x1&~x2)|(~x1&x2); // реализуемая функция
6
7  endmodule
```

4. Сохраните файл.
5. Скомпилируйте проект.
6. Создайте файл со входными тестовыми сигналами. В меню **File** выберите команду **New – Source – Do**. В открывшемся новом окне наберите следующий код:

```
1  force x1 2#0 0ns, 2#1 100ns;
2
3  force x2 2#0 0ns, 2#1 50ns, 2#0 100ns, 2#1 150ns;
```

7. Сохраните его под именем **Stimul.do**.
8. Перейдите в режим моделирования. Откройте графическое окно и добавьте в него проверяемые сигналы. Подключите к проекту файл с тестовыми сигналами. Запустите проект на моделирование. Проверьте полученные результаты:

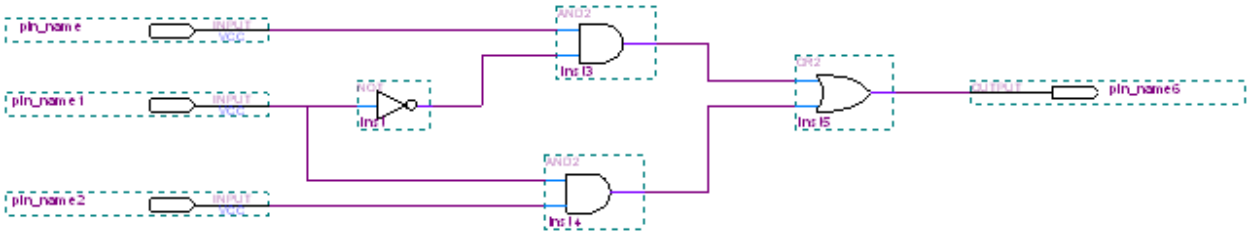


9. Выйдите из режима моделирования.

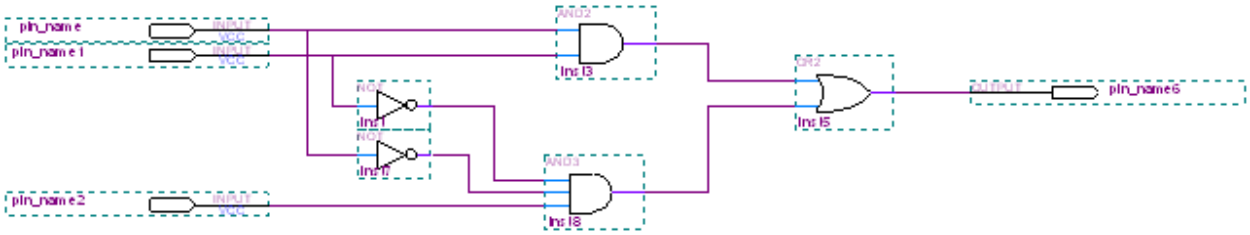
4. Задания

4.1. Задание 1. Создайте описание и промоделируйте работу следующих комбинационных схем.

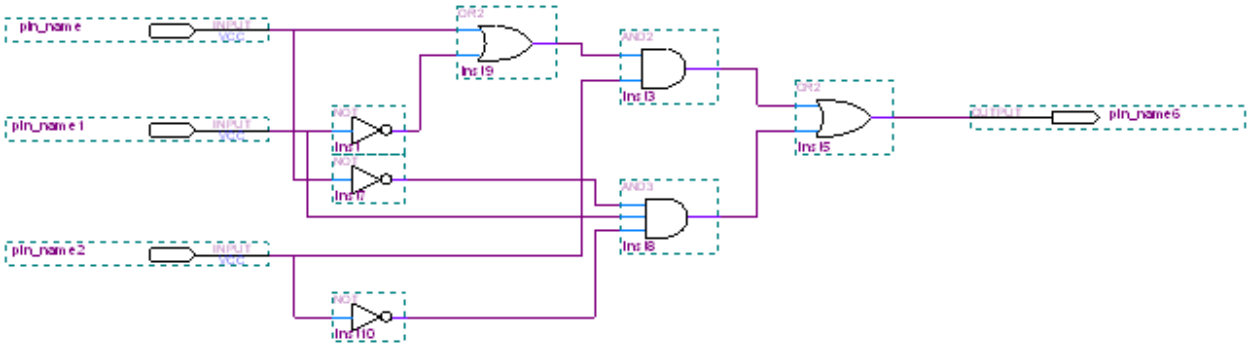
Вариант 1



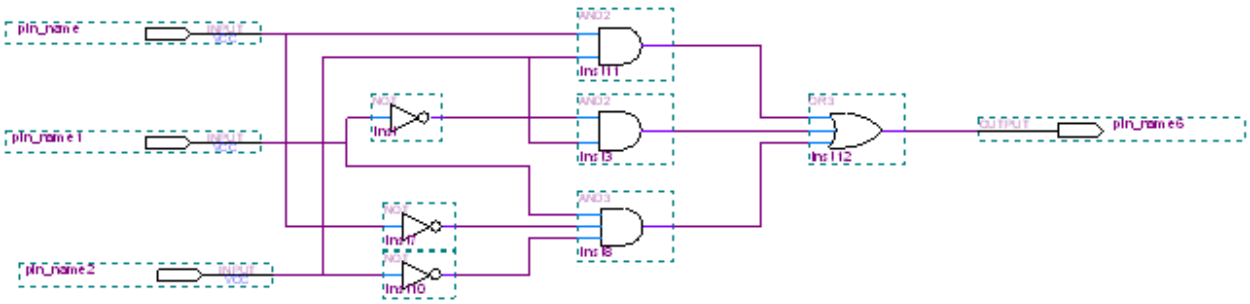
Вариант 2



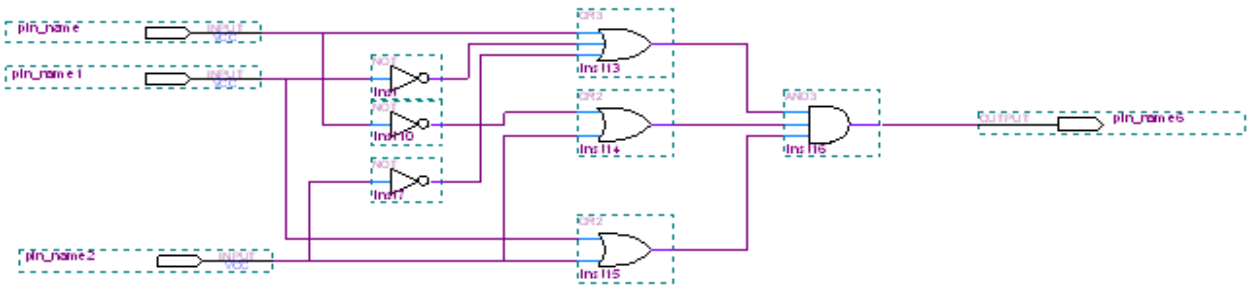
Вариант 3



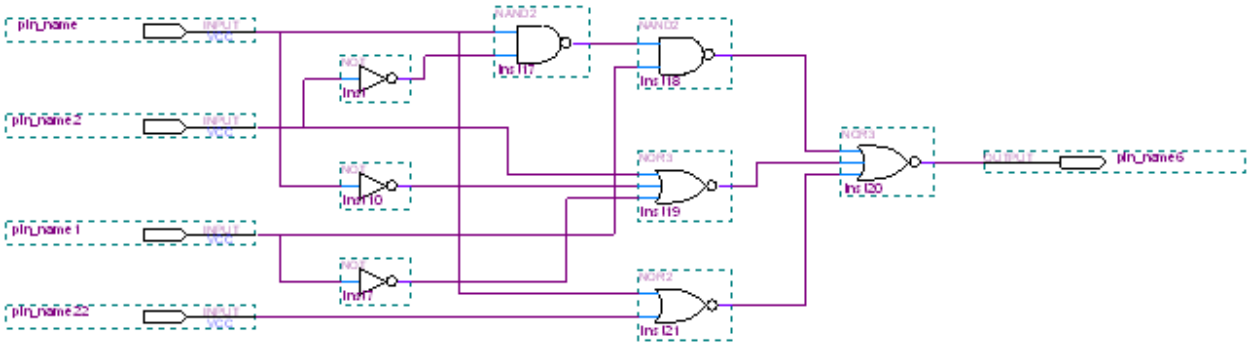
Вариант 4



Вариант 5



Вариант 6



4.2. **Задание 2.** Создайте описание (если возможно – минимизируйте функцию) и промоделируйте работу устройств, заданных следующими логическими функциями:

Вариант	Функция
I	`F = X'*Y'*Z' + X*Y*Z + X*Y'*Z`
II	`F = A*B + A*B'*C' + A'*B*C`
III	`F = A'*B*(C*B*A' + B*C')`
IV	`F = X*Y*(X'*Y*Z + X*Y'*Z + X*Y*Z' + X'*Y'*Z)`
V	`F = (A + A')*B + B*A*C' + C*(A + B')*(A' + B)`
VI	`F = X*Y' + Y*Z + Z'*X`

5. Подготовить и представить отчет о выполненной лабораторной работе в электронном виде.