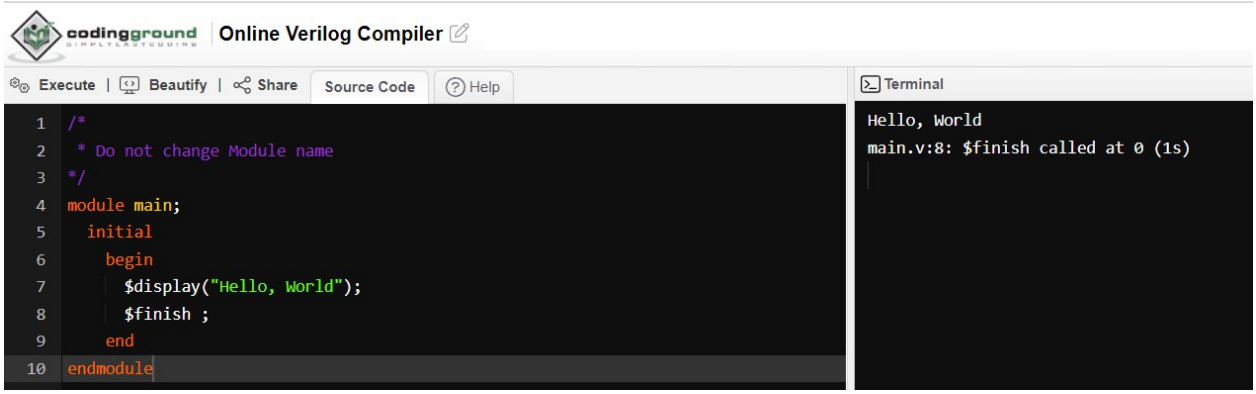
При разработке данных заданий предполагалось, что студент не обладает знаниями языка verilog. Однако, поощряется первичное ознакомление с языком и даже попытки его применения для решения задач. Большое количество примеров кода можно найти в интернете в целом и на сайте github в частности.

Предлагаю перед выполнением заданий ознакомиться с небольшим гайдом, который поможет понять, как именно можно применить verilog для выполнения заданий.

Попробовать применить verilog в текстовом режиме может каждый очень быстро. Для этого надо зайти на сайт:

https://www.tutorialspoint.com/compile_verilog_online.php



Для вас сразу доступна программа hello world которую вы можете выполнить.

Сделаем более сложный пример:

```
// Пример схемы: D-триггер
module dff (q, d, clk);
  output q;
  input d, clk;
  reg q;
  always @(posedge clk)
    q <= d;
endmodule

// Пример тестбенча: проверка работы D-триггера
module testbench;
  reg d, clk;
  wire q;

  // Создаем экземпляр схемы
  dff uut (q, d, clk);

  // Генерируем тактовыи сигнал
  initial begin
    clk = 0;
    forever #5 clk = ~clk; // Период тактового сигнала - 10 единиц времени
  end

  // Генерируем входнои сигнал и выводим результат на консоль
  initial begin
   $monitor("time = %5d, d = %d, q=%d", $time, d, q); // Формат вывода значении в каждыи момент
времени
    #0 d = 0; // Начальное значение входного сигнала - ноль
   #10 d = ~d; // Меняем значение входного сигнала на противоположное после первого отрицательного
фронта тактового сигнала
    #25 $finish; // Завершаем тестбенч после третьего положительного фронта тактового сигнала
  end

endmodule
```

Результат:

codingground | Online Verilog Compiler

Execute | Beautify | Share | Source Code | Help

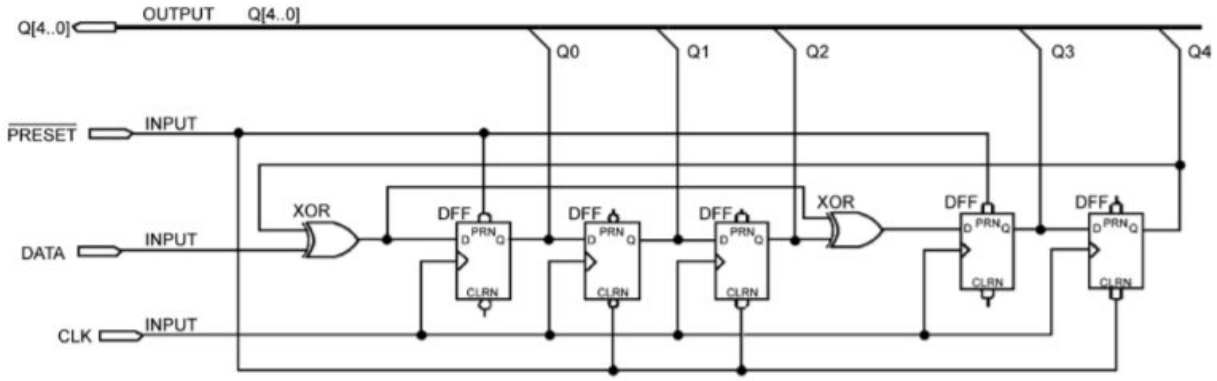
```
1  // Пример схемы: D-триггер
2  module dff (q, d, clk);
3    output q;
4    input d, clk;
5    reg q;
6    always @(posedge clk)
7      q <= d;
8  endmodule
9
10 // Пример тестбенча: проверка работы D-триггера
11 module testbench;
12   reg d, clk;
13   wire q;
14
15   // Создаем экземпляр схемы
16   dff uut (q, d, clk);
17
18   // Генерируем тактовый сигнал
19   initial begin
20     clk = 0;
21     forever #5 clk = ~clk; // Период тактового сигнала - 10 единиц времени
22   end
23
24   // Генерируем входной сигнал и выводим результат на консоль
25   initial begin
26     $monitor("time = %5d, d = %d, q=%d", $time, d, q); // Формат вывода
           значений в каждый момент времени
27     #0 d = 0; // Начальное значение входного сигнала - ноль
```

Terminal

```
time =     0, d = 0, q=x
time =     5, d = 0, q=0
time =    10, d = 1, q=0
time =    15, d = 1, q=1
main.v:29: $finish called at 35 (1s)
```

Задание 1

Дана схема для вычисления CRC (cyclic redundancy check), представленная на рисунке:



DFF – это триггеры типа flip-flop.

Данная схема инициализирована значениями, показанными в таблице:

Register	Preload value
Q0	1
Q1	0
Q2	0
Q3	1
Q4	0

В схему через вход DATA последовательно выполняется загрузка следующего двоичного кода:

1 0 1 0 1

Необходимо определить, каким будет значение на выходах Q[4:0] после загрузки данного двоичного кода.

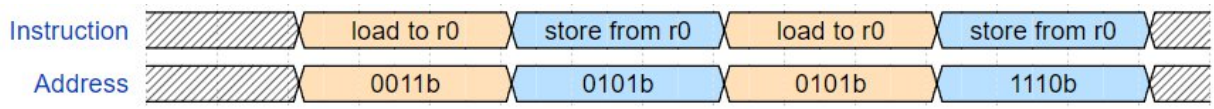
Задание 2

Дана память, заполненная данными. WL0-WL7 – это строки памяти, BL0-BL7 – это столбцы памяти. Память является однократно программируемой, т.е. каждая ячейка может изменить свое состояние из 0 в 1, но не может изменить его обратно, т.е. из 1 в 0.

	BL0	BL1	BL2	BL3	BL4	BL5	BL6	BL7
WL0	0	1	1	0	1	0	1	0
WL1	1	1	0	0	1	0	1	1
WL2	0	0	1	1	1	0	0	1
WL3	1	0	0	1	0	1	1	1
WL4	0	1	1	1	1	0	0	0
WL5	1	1	0	0	0	1	0	1
WL6	0	1	1	1	0	0	0	1
WL7	1	0	0	0	1	1	1	1

Блок памяти имеет организацию 16x4, т.е. считывание осуществляется по 4 бита. Адрес [3:0] имеет разрядность 4 бита, при этом старшие 3 бита [3:1] служат для выбора строки, а младший бит [0] служит для выбора группы столбцов (0 – BL3-BL0, 1 – BL4-BL7).

Над данным массивом ЦПУ выполняет следующую последовательность операций:



load to r0 – это чтение информации по установленному адресу в регистр r0.

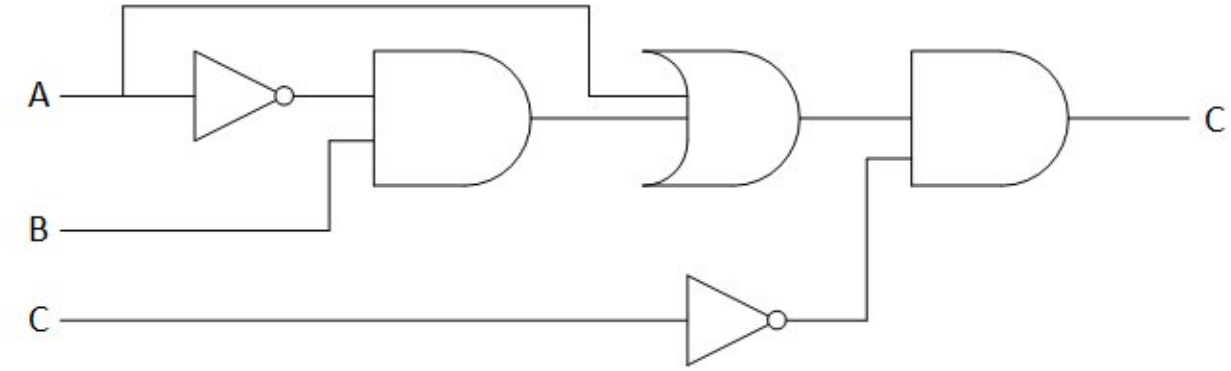
store from r0 – это сохранение информации, хранимой в регистре r0, по установленному адресу.

Необходимо вычислить результат, который будет в результате сформирован по адресу, являющемуся последним в приведенной последовательности.

Ответ дать в виде двоичного 4-разрядного числа, например: 0101

Задание 3

Составить таблицу истинности для представленной схемы



Задание 4

Разрабатывается автомат состояний для турникета.

Состояния: открыт/закрыт

Действия пользователя: вставить монету/повернуть турникет

Необходимо описать на C++ автомат состояний следующим образом. Состояние автомата записано в переменной. Имеется функция change_state, которая принимает на вход текущее состояние автомата и действие, выполняемое с автоматом и возвращает новое состояние автомата. Функция должна печатать на экран значения входных данных и полученное в результате состояние. В функции main описывается тест, который производит несколько раз вызов change_state с разными значениями, подаваемыми на вход.

Для компиляции рекомендуется использовать сайт https://cpp.sh/