Лабораторная работа № 3

Мультиплексор и декодер. Автоматизация тестирования с помощью файла с набором тестовых векторов.

1. Цель работы

Целью работы является изучение способов реализации мультиплексора и декодера на языке Verilog (SystemVerilog). Научиться тестировать схемы при помощи файла с набором тестовых векторов.

2. Краткие теоретические сведения

Языки описания аппаратуры HDL позволяют описывать параметризованные модули с портами переменной ширины.

Для этого перед списком входов и выходов для определения параметров модуля ставится конструкция #(parameter …). В примере оператор parameter состоит из параметра по имени WIDTH со значением по умолчанию, равным 8.

```
module my_module #(parameter WIDTH = 8)(
input logic [WIDTH-1:0] data;
output logic [WIDTH-1:0] outdata;
);
assign outdata = data;
endmodule
```

При создании экземпляра такого модуля понадобится указать значение параметра (если этого не сделать, то создастся модуль с параметром, заданным по умолчанию). В примере ниже создаётся экземпляр приведённого выше модуля с шириной входов, равной 12.

```
my_module #(12) outmodule (data, outdata);
```

Не нужно путать использование знака # для обозначения задержек с использованием #(...) при объявлении и переопределении параметров.

3. Описание работы

Мультиплексор

Мультиплексор осуществляет подключение одного из входных каналов к выходному под управлением управляющего (адресующего) слова. Разрядности каналов могут быть различными.

Пример реализации простейшего мультиплексора на языке SystemVerilog:

```
module mux2 (
input logic c0, c1,
input logic a,
output logic out
);

assign out = (a) ? c0 : c1;

endmodule

```

Декодер

Двоичный декодер (дешифратор) преобразует двоичный код в код «1 из N». В кодовой комбинации этого кода только одна позиция занята единицей, остальные – нулевые.

Для описания дешифратора на языке Verilog используется оператор case. Ниже приведён прототип дешифратора 3 на 8.

```
module decoder3_8 (input logic [2:0] a,
                   output logic [7:0] y);
   always_comb
      case (a)
         3'b000: y = 8'b00000001;
         //здесь пропущено несколько строк
         default: y = 8'bxxxxxxxx;
      endcase

endmodule
```

4. Автоматизация тестирования

Для тестирования модуля, требующего большого количества входных значений можно использовать файлы с набором векторов.

Тестовый вектор – совокупность входного варианта значений и соответствующего ожидаемого результата.

Среда тестирования будет читать значения из файла, подавать входной вектор на входы тестируемого модуля, проверять, что значения входов совпадают с выходным вектором, и повторять, пока не будет достигнут конец файла.

Пример тестирующего модуля для однобитного мультиплексора приведён ниже.

```
module testbench_testvectors_mux2_01();

   logic clk, reset;
   logic operand0, operand1, yexpected, y;
   logic sel;
   logic [31:0] vectornum, errors;
   logic [3:0]  testvectors[10000:0];

   mux2 #(1) dut(operand0, operand1, sel, y);

   always
      begin
         clk = 1; #5; clk = 0; #5;
      end

```

```
16      initial
17         begin
18            $readmemb("testcase_mux2_01.dat", testvectors);
19            vectornum = 0; errors = 0;
20            reset = 1; #27; reset = 0;
21         end
22
23      always @(posedge clk)
24         begin
25            #1; {operand0, operand1, sel, yexpected} = testvectors[vectornum];
26         end
27
28      always @(negedge clk)
29         if (~reset)
30            begin
31               if (y !== yexpected)
32                  begin
33                     $display("Error: inputs = %b %b %b", operand0, operand1, sel);
34                     $display(" outputs = %b (%b expected)", y, yexpected);
35                     errors = errors + 1;
36                  end
37
38               vectornum = vectornum + 1;
39
40               if (testvectors[vectornum] === 4'bx)
41                  begin
42                     $display("%d tests completed with %d errors", vectornum, errors);
43                     $stop;
44                  end
45            end
46
47   endmodule
```

Первый блок **always** необходим для генерации тактового сигнала. Так как тестируемый модуль является комбинационным, то для его работы тактовый сигнал не нужен. В данном примере тактовый сигнал используется только для организации работы самого тестирующего модуля.

Блок **initial** используется инициализации переменных тестирующего модуля и для чтения тестовых векторов из файла testcase_mux2_01.dat

Остальные блоки используются для инкремента номера тестового вектора, последовательной подачи на вход тестируемого модуля всех входных вариантов из набора, сравнения выхода тестируемого модуля с ожидаемым результатом и выдачи сообщений об ошибках.

Формат файла: текстовый файл с набором строчек из нулей и единиц:

```
0000
0010
0100
```

Каждая строка – это один тестовый вектор. Для удобства можно использовать символ подчеркивания с целью разделения разрядов.

```
000_0
001_0
010_0
```

В вышеприведенном случае символ подчеркивания был использован для разделения входного варианта и ожидаемого результата. При загрузке файлов символ подчеркивания игнорируется.

5. Задание

1. Спроектируйте на языке System Verilog модуль двухвходового мультиплексора. Сделайте этот модуль параметризуемым. Пусть параметром будет количество разрядов в мультиплексируемых шинах. Для реализации комбинационной логики в таком мультиплексоре достаточно использовать оператор **assign**.

2. Незаконченный прототип такого мультиплексора:

```
1   module mux2 //???
2      input logic [WIDTH-1:0] d0,d1,
3      input logic s,
4      output logic //???
5   );
6
7      //???
8   endmodule
9
```

Закончите строки со знаками вопроса.

3. Напишите на языке System Verilog модуль, тестирующий работу мультиплексора (тестбенч). Для простоты ограничьте разрядность мультиплексируемых входов до 1-го бита.

6. Составьте файл, в котором будут храниться тестовые вектора. Так как тестируется однобитный двухвходовый мультиплексор, следовательно у него немного вариантов входных значений. В файле тестовых векторов перечислите их все.

6. Проведите симуляцию и убедитесь, что ошибок нет.

7. Временно добавьте к тестовым векторам один заведомо неверный вариант, чтобы убедиться, что тестбенч правильно обрабатывает такую ситуацию.

8. Увеличьте разрядность мультиплексора до 8-и бит, внесите необходимые изменения в тестбенч и создайте другой файл с тестовыми векторами, который будет соответствовать новому мультиплексору.

Подсказка: необходимо внести изменения в строки с номерами 4, 7, 9, и 40 примера.

Так как возможных вариантов входных значений теперь становится слишком много, то выберите те из них, которые в достаточной мере протестируют мультиплексор.

Проведите симуляцию и убедитесь, что ошибок нет.

9. Спроектируйте модуль декодера с трёхбитным входом. Для реализации комбинационной логики используйте блок **always_comb**

11. Протестируйте его, внеся необходимые изменения в тестбенч, и создав новый файл с тестовыми векторами.