1. ЛАБОРАТОРНАЯ РАБОТА №1

Тема работы: Реализация конечных автоматов, заданных автоматным графом.

Лабораторная работа посвящена проектированию конечных автоматов на языке Verilog средствами САПР Vivado.

1.1. Задание на практическую работу

Реализовать автомат Мили, заданный графом, в соответствии с вариантом, который определяет выходные значения для каждого из состояний автомата на языке Verilog. Автомат должен иметь два входа: a,b,c (помимо входа синхросигнала). На графе над дугой указаны три символа, их следует трактовать как значения для a, b, c соответственно, необходимые для совершения обозначенного перехода. Варианты заданий присутствуют в СДО или могут быть изменены преподавателем самостоятельно.

1.2. Теоретическое введение

Одним из возможных вариантов классификации синхронных автоматов может служить их разделение на автоматы Мура и автоматы Мили.

1.2.1. Автоматы Мура и Мили

Автомат Мура — синхронный автомат, у которого выходные значения определяются только состоянием автомата в тот же дискретный момент времени. При этом комбинационная схема, вычисляющая выходные значения, не связана непосредственно с входными сигналами.

Схематично внутреннее устройство автомата Мура может быть представлено, как показано на Рисунок. 1.1.

Рисунок 1.1. Структура автомата Мура

Как видно из рисунка, входы влияют только на изменение состояния, при этом выходные значения полностью определяются в зависимости от текущего состояния автомата.

Таким образом, функции G и F такого автомата можно определить следующим образом (Формулы 1.1 – 1.2):

$$G = G(A, Q) \quad (1.1)$$

$$F = F(Q) \quad (1.2)$$

Граф автомата Мура будет содержать состояния в качестве вершин графа, помеченные символами выходного алфавита, дуги помечаются символами входного алфавита.

Автомат Мили — синхронный автомат, у которого вход и выход не развязаны во времени, т.е. хотя бы один выход зависит от текущего значения на входе.

Схематично внутреннее устройство автомата Мили может быть представлено, как показано на Рисунок. 1.2.

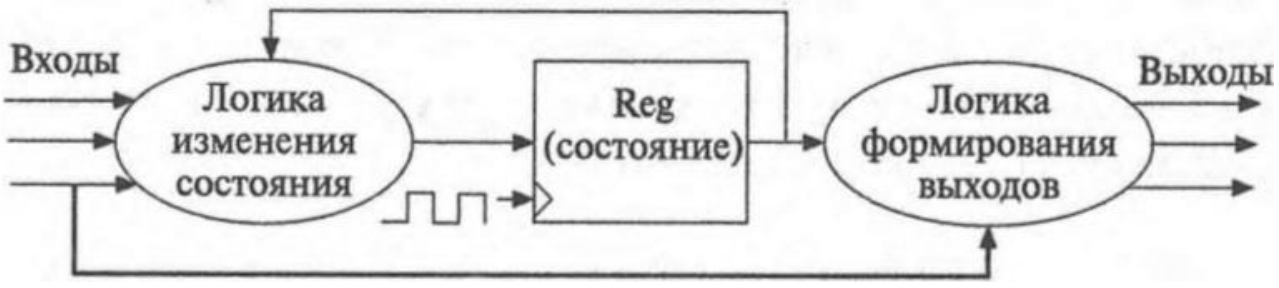

Рисунок 1.2. Структура автомата Мили

В данном случае стоит отметить, что если у автомата присутствует более одного выхода, но при этом есть хотя бы один выход, в логике формирования

значения, на котором непосредственно участвует хотя бы один вход, то такой автомат всё равно будет причислен к автоматам Мили.

Таким образом, функции G и F такого автомата можно определить следующим образом (Формулы 1.3 – 1.4):

$$G = G(A, Q) \tag{1.3}$$

$$F = F(A, Q) \tag{1.4}$$

Граф автомата Мили будет содержать состояния в качестве вершин графа, дуги помечаются символами входного и выходного алфавитов.

Таким образом говорят, что автомат Мура развязывает по времени входы и выходы автомата. При этом автомат Мили зачастую имеет меньшее число состояний. Полезным свойством автоматов Мура и Мили является эквивалентность: из автомата Мили может быть получен эквивалентный автомат Мура, что может быть полезно, если необходима вышеупомянутая развязка для снижения возможности появления паразитных значений вследствие прямой зависимости выходов автомата Мили от входов. Под эквивалентными автоматами в данном случае понимают такие два инициальных автомата, которые перерабатывают одну и ту же входную последовательность в одну и ту же выходную, однако автомат Мура выдаёт те же значения со сдвигом в такт. Подробнее эти вопросы будут изучены на дисциплине «Теория автоматов».

1.2.2. Автоматная таблица

Автоматная таблица является одним из способов задания функций определения нового состояния, а также определения выходного значения для автоматов.

Каждая строка таблицы соответствует состоянию автомата, а каждый столбец — символу входного алфавита. На пересечении строки с индексом i и столбцом с индексом j располагается новое состояние автомата, в которое переходит автомат, находясь в состоянии i при приходе входного слова j, а также значение выхода автомата (для автомата Мили — в текущий дискретный момент времени, для автомата Мура — в следующий дискретный момент времени).

1.2.3. Операторы выбора

Для проектирования автомата средствами языка Verilog, следует остановить своё внимание на операторе «case».

Оператор «case» является оператором выбора. Имеется три формы данного оператора: «case», «casex», «casez».

1.2.3.1. Оператор «case»

Синтаксис оператора «case» приведён в листинге 1.1.

Листинг 1.1. Синтаксис оператора «case»

```
case (выражение)
    конст_значение_1: begin
        <набор инструкций>
    end
    конст_значение_2: begin
        <набор инструкций>
    end
    default: begin
        <набор инструкций>
    end
endcase
```

Результат вычисления выражения, стоящего в скобках у ключевого слова «case», сравнивается с константными значениями. В случае, если было обнаружено совпадение, то выполняется набор инструкций, соответствующих этому константному значению.

В случае, если не было найдено ни одного совпадения, то выполняется набор инструкций, указанный в блоке «default».

Например, пусть необходимо описать схему дешифратора, используя оператор case. Фрагмент исходного кода такого модуля приведён в листинге 1.2.

Листинг 1.2. Пример описания дешифратора с помощью оператора case

```
…
input [3:0] DATA;
output reg [7:0] DOUT;
…
case (DATA)
    4'h1: DOUT <= 8'b0000_0001;
    4'h3: DOUT <= 8'b0000_0010;
    4'h5: DOUT <= 8'b0000_0100;
    4'h7: DOUT <= 8'b0000_1000;
    4'h9: DOUT <= 8'b0001_0000;
```

Продолжение листинга 1.2

```
        4'hB: DOUT <= 8'b0010_0000;
        4'hD: DOUT <= 8'b0100_0000;
        4'hF: DOUT <= 8'b1000_0000;
        default: DOUT <= 8'b0000_0000;
endcase
```

Стоит обратить внимание, что в таком примере при отсутствии «default» схема будет содержать защёлку «DOUT», поскольку не описано поведение «DOUT» во всех ситуациях в зависимости от «DATA». Поскольку в примере указан блок «default», все возможные значения определены, а значит описанная схема – комбинационная.

1.2.3.2. Операторы «casex» и «casez»

Синтаксис этих операторов не отличается от оператора «case» за исключением ключевого слова. Используются операторы в том случае, если часть битов результата вычисления выражения должны быть проигнорированы. Оператор «casex» игнорирует биты, отмеченные как «**x**» или как «**z**», в то время как оператор «casez» игнорирует только биты, отмеченные как «**z**». Оба оператора позволяют использовать символ «**?**» вместо «**x**» и «**z**», чтобы наглядно подчеркнуть тот факт, что значение битов должны быть проигнорированы.

При использовании операторов «casex» и «casez» следует внимательно следить за тем, чтобы в константных значениях отсутствовал дубляж, поскольку реакция на такое описание синтезатора не может быть предугадано.

Примеры использования операторов «casex» и «casez» приведены в листингах 1.3 и 1.4 соответственно.

Листинг 1.3. Пример использования оператора casex

```
casex (DATA)
    8'b1100_xx00: DOUT <= 8'b0000_0001;
    8'b0110_0xx0: DOUT <= 8'b0000_0010;
    8'b0011_00xx: DOUT <= 8'b0000_0100;
    8'b1001_x00x: DOUT <= 8'b0000_1000;
    8'b1010_x0x0: DOUT <= 8'b0001_0000;
    8'b0101_0x0x: DOUT <= 8'b0010_0000;
    8'b1111_xxxx: DOUT <= 8'b0100_0000;
    8'b0000_0000: DOUT <= 8'b1000_0000;
    default: DOUT <= 8'b0000_0000;
endcase
```

Листинг 1.4. Пример использования оператора casez

```
casez (DATA)
    8'b1???_????: DOUT <= 8'b0000_0001;
    8'b01??_????: DOUT <= 8'b0000_0010;
    8'b001?_????: DOUT <= 8'b0000_0100;
    8'b0001_????: DOUT <= 8'b0000_1000;
    8'b0000_1???: DOUT <= 8'b0001_0000;
    8'b0000_01??: DOUT <= 8'b0010_0000;
    8'b0000_001?: DOUT <= 8'b0100_0000;
    8'b0000_0001: DOUT <= 8'b1000_0000;
    default: DOUT <= 8'b0000_0000;
endcase
```

Данные операторы удобны при описании реакции перехода автоматов. Далее будет рассмотрен пример описания автомата на языке Verilog.

1.2.4. Принцип описания конечного автомата

Пусть необходимо описать автомат, имеющий вход «a», а также выход, который полностью совпадает с номером состояния автомата. Граф перехода автомата приведён на Рисунок 1.3.

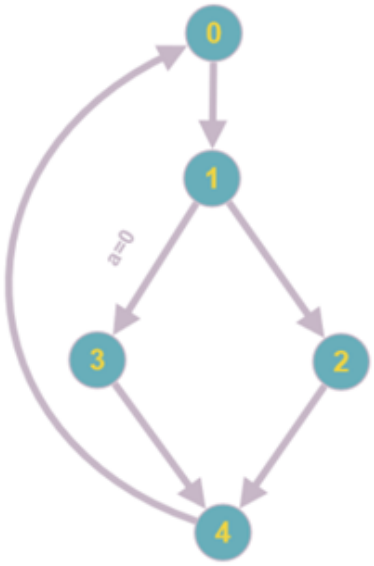

Рисунок 1.3. Граф автомата на пять состояний

Переход из нулевого состояния в первое является безусловным. Находясь в первом состоянии, автомат должен проанализировать значение на входе «a» и перейти в третье состояние в случае, если на входе «a» присутствует значение логического нуля, во всех остальных случаях должен состояться переход во второе состояние. Из второго и третьего состояний автомат безусловно переходит в четвертое состояние, а из четвертого безусловно в нулевое. Также пусть нулевое состояние будет зафиксировано как начальное (инициальный автомат). Исходный код модуля автомата приведён в листинге 1.5.

Листинг 1.5. Описание модуля автомата Мили для заданного графа

```
module fsm (input clk, a, output reg [2:0] state);
    initial
        state = 0;
    always@(posedge clk)
    begin
        case (state)
            3'd0: state <= 3'd1;
            3'd1: begin
                if(a)
state <= 3'd2;
```

Продолжение листинга 1.5

```
                    else
                        state <= 3'd3;
                end
                3'd2: state <= 3'd4;
                3'd3: state <= 3'd4;
                3'd4: state <= 3'd0;
                default: state <= state;
            endcase
        end
    endmodule
```

Описанная схема представлена на Рисунке 1.4, временная диаграмма — Рисунок 1.5. Стоит обратить особое внимание при тестировании автомата на необходимость получить на временной диаграмме все возможные переходы, соответствующие графу автомата, хотя бы один раз.

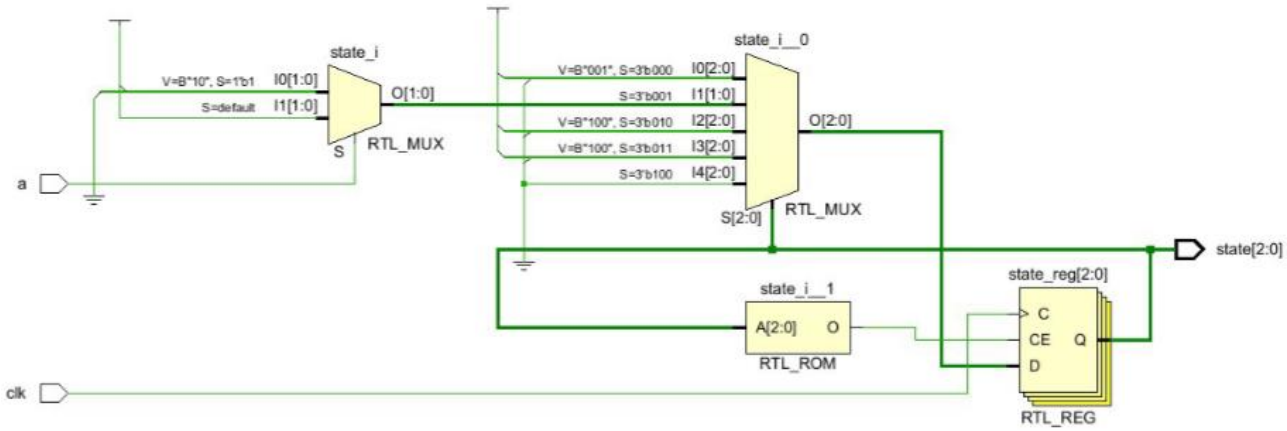

Рисунок 1.4. RTL-представление автомата Мили

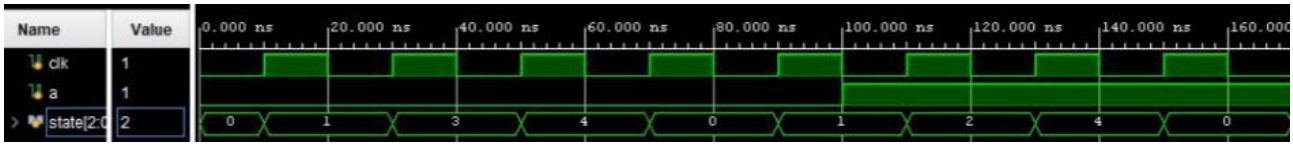

Рисунок 1.5. Временная диаграмма работы автомата Мили

1.2.5. Пример описания конечного автомата Мили на языке Verilog

Пусть ставится задача разработать автомат Мили, имеющий два входа «a», «b» и вход синхросигнала, а также выход «d». Входы и выходы являются однобитными. Автомат должен работать согласно следующему принципу: если на входе «b» присутствует значение логической единицы, то автомат должен перейти в состояние, номер которого указан на входе «a», в противном случае номер состояния инкрементируется, а максимальное значение номера состояния ограничивается единицей. На выход «d» поступает результат сложения по модуль два номера состояния автомата и значения на входе «a».

Соответственно, для такого автомата число состояний будет определено как два, а значит потребуется один D-триггер для фиксации состояния. Поскольку для автомата Мили верно, что выход не развязан по времени со входами, то значения со входов будут записываться на соответствующие триггеры, а лишь затем проанализированы автоматом. Логика перехода в новое состояние и логика определения выходного значения будет комбинационной.

Исходный код модуля, реализующего такой автомат как автомат Мили на языке Verilog приведён в листинге 1.6.

Листинг 1.6. Реализация автомата Мили на Verilog HDL

```
`timescale 1ns / 1ps


module Mealy_FSM(
    input a, b, clk,
    output d);
wire next_state;
reg state = 0;
reg a_=0, b_=0;
always @(posedge clk)
begin
    a_ = a;
    b_ = b;
```

Продолжение листинга 1.6

```
    end
    assign next_state = b_ ? a_ : state + 1;
    assign d = state ^ a_;
    always @(posedge clk)
    begin
        state = next_state;
    end
endmodule
```

Схема, соответствующая описанию модуля представлена на рис. 7.6. Стоит обратить внимание, что выход непосредственно связан со входным значением, взятым с соответствующего триггера (входное значение относительно автомата).

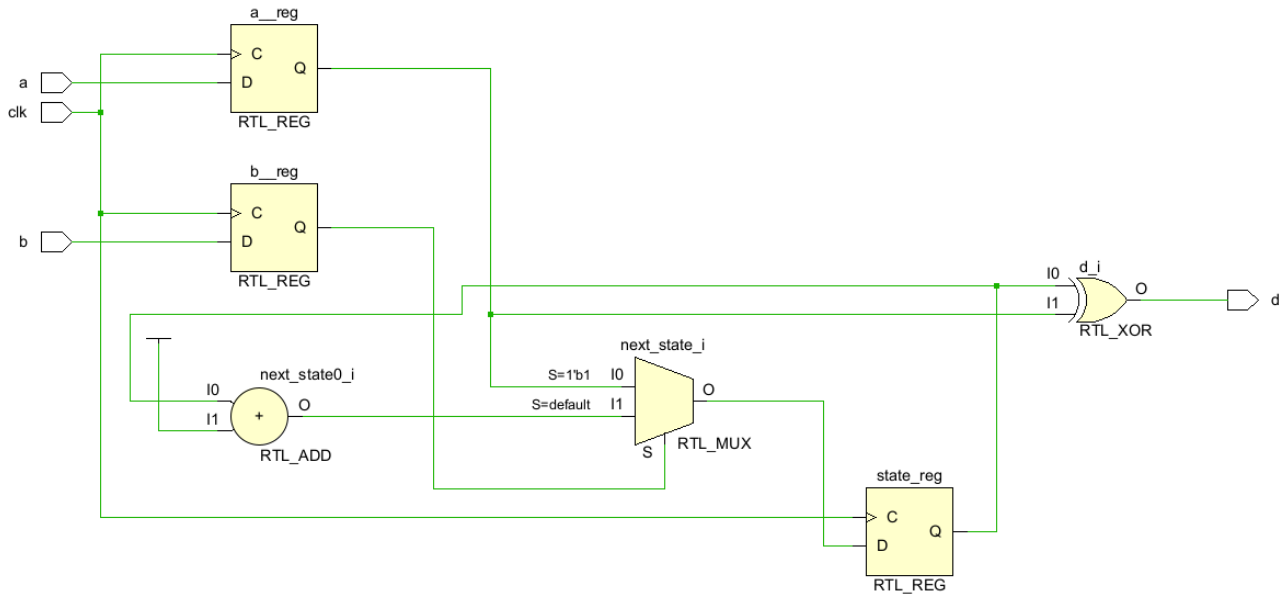

Рисунок 1.6. RTL-представление автомата Мили

Фрагмент временной диаграмма работы автомата приведён на Рисунок 1.7.

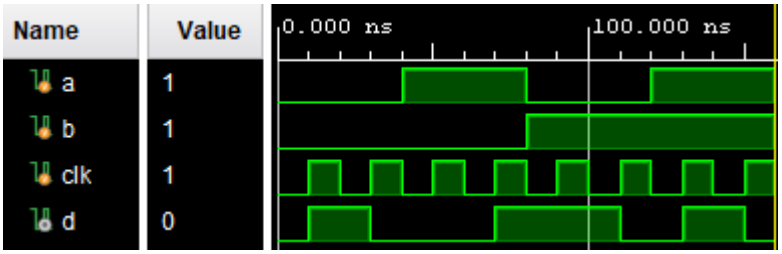

Рисунок 1.7. Временная диаграмма работы автомата Мили

1.2.6. Пример описания конечного автомата Мура на языке Verilog

Пусть стоит задача построить автомат Мура, эквивалентный автомату Мили, рассматриваемому в пункте 1.2.5.

Для синтеза такого автомата будет построена автоматная таблица автомата Мили, в которой последовательность входов зафиксирована от «a» к «b» (табл. 1.1).

Таблица 1.1. Автоматная таблица Мили

Состояние\входы	00	01	10	11
S_0	S_1, 0	S_0, 0	S_1, 1	S_1, 1
S_1	S_0, 1	S_0, 1	S_0, 0	S_1, 0

Далее таблица будет перестроена в эквивалентную для автомата Мура (табл. 1.2).

Таблица 1.2. Автоматная таблица Мура

Состояние\входы	00	01	10	11
S_0 0	S_1 0	S_0 0	S_1 1	S_1, 1
S_0 1	S_1 0	S_0 0	S_1 1	S_1 1
S_1 0	S_0 1	S_0 1	S_0 0	S_1 0
S_1 1	S_0 1	S_0 1	S_0 0	S_1 0

В результате эквивалентный автомат Мура будет иметь четыре состояния и функционировать согласно разработанной таблице переходов. Автомат будет синтезирован, как автомат с регистром (двумя триггерами) на входах, как и в варианте для автомата Мили. Исходный код модуля для автомата Мура представлен в листинге 1.7.

Листинг 1.7. Реализация автомата Мура на Verilog HDL

```
`define X 2'b00
`define Y 2'b01
`define Z 2'b10
`define G 2'b11

module Moore_FSM (
    input a, b, clk,
    output reg d
);

reg [1:0] next_state = 0;
reg [1:0] state = 0;

reg a_=0, b_=0;

always @(posedge clk)
begin
    a_ = a;
    b_ = b;
```

```
    end
    // x00, y01, z10, g11
    always @(a_, b_, state) begin
        casex({state,a,b})
            {`X, 2'b00}, {`Y, 2'b00}: next_state <= `Z;
            {`X, 2'b01}, {`Y, 2'b01}: next_state <= `X;
            {`X, 2'b10}, {`Y, 2'b10}: next_state <= `G;
            {`X, 2'b11}, {`Y, 2'b11}: next_state <= `G;
            {`Z, 2'b00}, {`G, 2'b00}: next_state <= `Y;
            {`Z, 2'b01}, {`G, 2'b01}: next_state <= `Y;
            {`Z, 2'b10}, {`G, 2'b10}: next_state <= `X;
            {`Z, 2'b11}, {`G, 2'b11}: next_state <= `Z;
        endcase
    end
    always @(state) begin
        casex(state)
            2'b?1: d = 1'b1;
            default: d = 0;
        endcase
    end

    always @(posedge clk) begin
        state = next_state;
    end
    endmodule
```

Схема, соответствующая описанию модуля представлена на рис. 7.8.

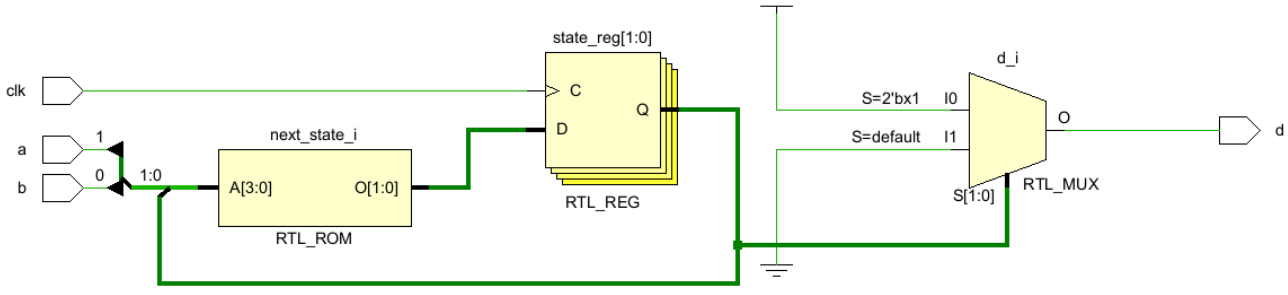

Рисунок 1.8. RTL-представление автомата Мура

Временная диаграмма работы схемы приведена на Рисунок 1.9. Если сравнить временные диаграммы работы автоматов Мура и Мили можно заметить, что выходное значение появляется с запозданием в один такт в случае автомата Мура.

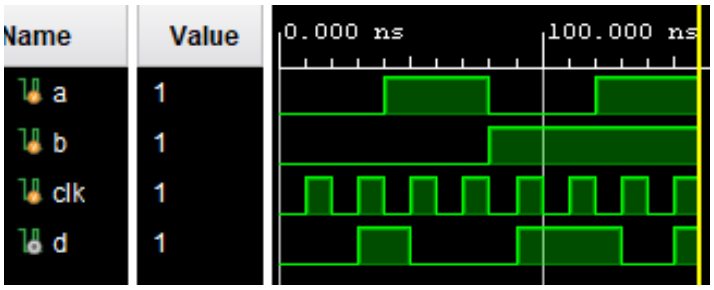

Рисунок 1.9. Временная диаграмма работы автомата Мура

1.3 Порядок выполнения работы

1. Создать проект в САПР Vivado.
2. Создать модуль, описывающий автомат Мили и Мура согласно варианту.
3. Создать тестовый модуль на языке Verilog HDL.
4. Произвести верификацию модуля посредством временной диаграммы. На временной диаграмме должны быть представлены все возможные переходы между состояниями, при этом каждый переход должен быть совершён хотя бы один раз.
5. Составить отчёт.
6. Представить работу и отчёт на проверку.
7. Ответить на вопросы преподавателя.

В отчёте в соответствующих разделах должны присутствовать:

1. Код описанных на Verilog HDL модулей с подробным описанием построения каждого модуля.
2. Скриншоты временных диаграмм.