МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине
‘‘Функциональная схемотехника’’

Студенты:

Алексеев Даниил
Сабитов Данил

Группа P33302

Преподаватель:

Васильев С.Е.

Санкт-Петербург
2023

Задание

Вариант 9

Вариант	Функция 1	FSM	Функция 2	Разрядности	Делитель частоты
9	COUNT_FREE	FSM_4	LRU	32 бита	5

FSM_4:

((A + B)*4 + B)/2 + (B/2 + A*4)

Функция COUNT_FREE:

Необходимо после прошествия определенного количества времени сформировать однобитный сигнал запроса. Информацию о количестве тактов, после которого необходимо формировать сигнал, устройство получает по однобитному последовательностному порту запроса. Необходимые сигналы в интерфейсе вашего модуля:

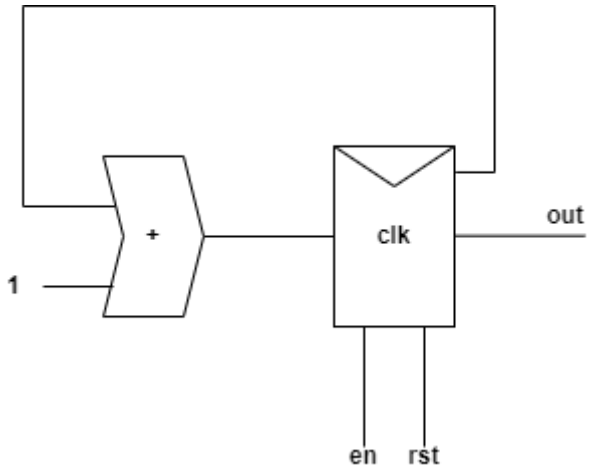
Порт	Тип	Описание
clk	Вх	Сигнал тактовой частоты.
rst	Вх	Асинхронный сигнал сброса.
start_req_i	Вх	Сигнал запроса. Представляет собой сигнал валидности для однобитного сигнала входных данных.
start_data_i	Вх	Однобитный сигнал данных, представляющий собой один разряд исходного числа.
ready_i	Вх	Сигнал готовности внешнего устройства принять результат.
result_rsp_o	Вых	Сигнал готовности результата. Выставляется в высокий уровень тогда и только тогда, когда на шине данных установлен корректный результат. Держится в высоком состоянии ровно один период тактового сигнала, когда внешнее устройство готово принять результат; в противном случае держится в высоком состоянии до тех пор, пока устройство не будет готово принять результат.
busy_o	Вых	Сигнал занятости устройства.

Модуль “BUFFER_LRU”:

Модуль-буффер из восьми элементов, в котором вытеснение происходит по алгоритму LRU. Интерфейс подачи транзакции должен включать в себя шину данных и соответствующий ей сигнал валидности данных. Выходной интерфейс представляет собой текущее состояние всех восьми элементов буффера.

Часть 1:

Счетчик:



Модуль счётчика на 32 бита. По каждому фронту тактового сигнала clk инкрементирует значение внутреннего регистра и на шину out выдаёт значение. Сигнал rst обнуляет счётчик, а сигнал en отвечает за включение счётчика.

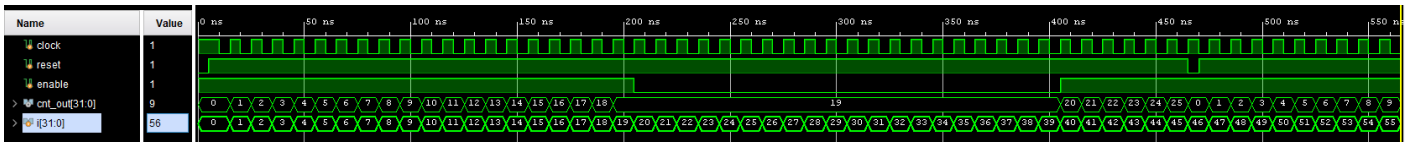
Код:

```
module counter(clk, rst, en, out);
    input clk, rst, en;
    output reg [31:0] out;


    always @ (posedge clk or negedge rst) begin
        if(! rst)
            out <= 0;
        else if (en)
            out <= out+1;
    end
endmodule
```

Тестовый план:

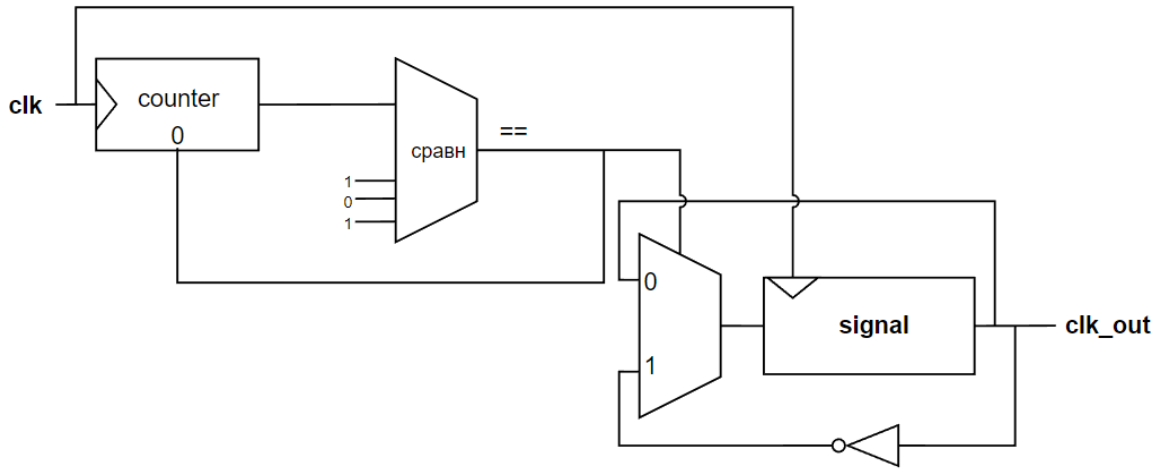
1) Проверка работоспособности на промежутке времени
2) Проверка выходного значения при enable = 1

3) Проверка сброса значения
4) Проверка работы после сброса значения
5) Снова проверка при enable = 1



временная диаграмма работы тестового окружения модуля counter

Делитель частоты:



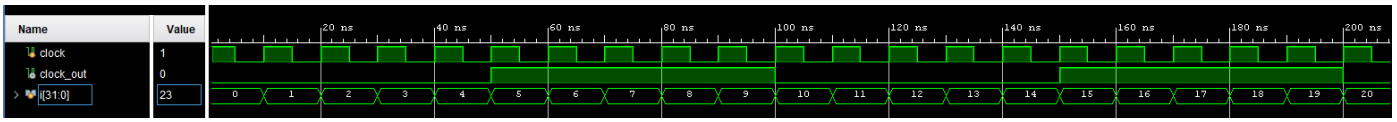
Делитель частоты на 2 представляет собой последовательную архитектуру, состоящую из сумматора и регистра, у которой есть вход тактового сигнала CLK, а также выход clk_out.

Код:

```
module freq_div(
    input clk,
    output reg clk_out
);

reg [2:0] counter = 3'b111;
reg signal = 0;

always @(posedge clk) begin
    counter = counter + 1;
    if (counter == 3'b101) begin
        counter = 0;
        signal = ~signal;
    end
    clk_out = signal;
end
endmodule
```

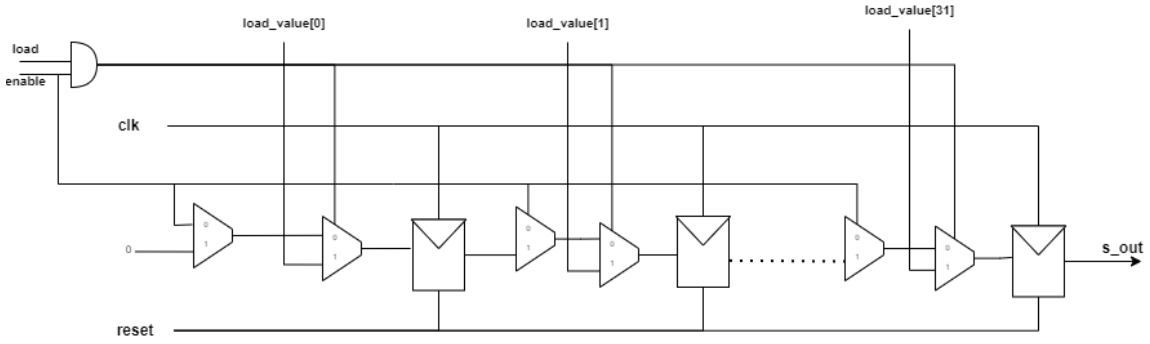
Тестовый план:

1) Проверка деления подаваемой на вход частоты на 2



временная диаграмма работы тестового окружения модуля freq_divider

Сдвиговый регистр:



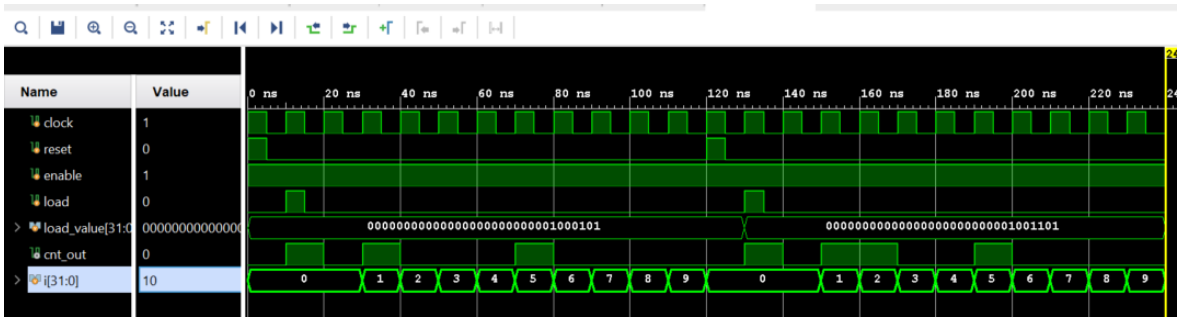
Сдвиговый регистр с параллельным входом и последовательным
выходом. Каждый фронт тактового сигнала, при наличии активного сигнала разрешения,
выполняется операция сдвига вправо. Крайний правый выдвинутый разряд должен подаваться на выход схемы. Необходимо также предусмотреть возможность асинхронного
сброса. Разрядность регистра устанавливается в соответствии с вариантом.

Код:

```
module shift_register(
    input clk,
    input reset,
    input enable,
    input load,
    input [31:0] load_value,
    output s_out
);
reg [31:0] shiftreg_value;
wire [31:0] shifted_val_next;
assign shifted_val_next = (load) ? load_value : shiftreg_value >> 1;
always @(posedge clk, posedge reset) begin
    if (reset)
        shiftreg_value <= 31'b0;
    else if (enable) begin
        shiftreg_value <= shifted_val_next;
    end
end
assign s_out = shiftreg_value[0];
endmodule
```

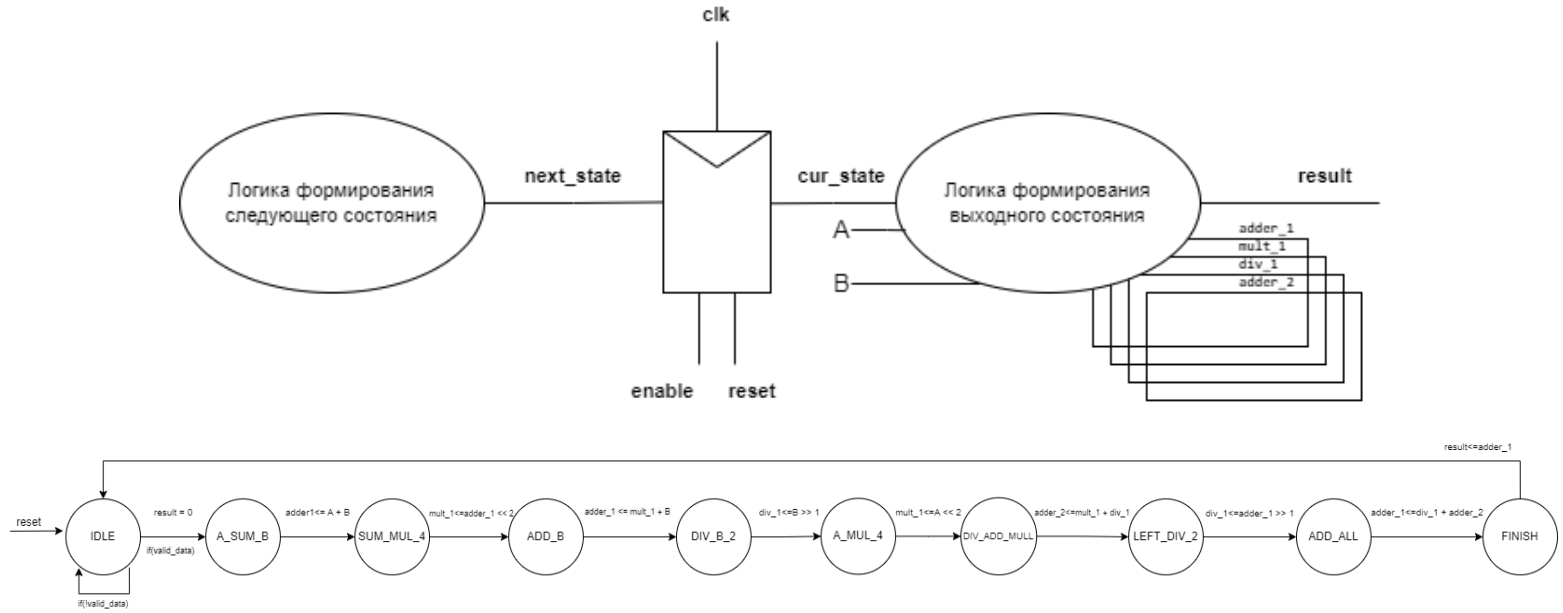
Тестовый план:

1) Проверка работы сброса
2) Проверка загруженного значения



временная диаграмма работы тестового окружения модуля shift_register

FSM_4:



Код:

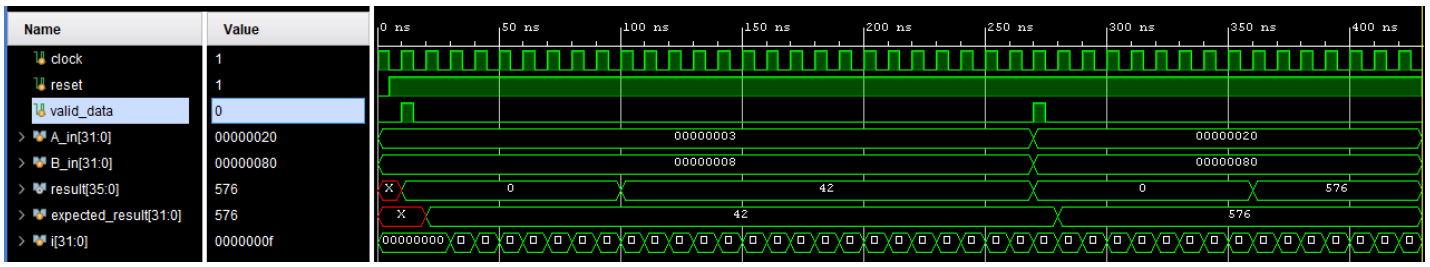
```
module fsm_4(
    input clk,
    input reset,
    input valid_data,
    input [31:0] A,
    input [31:0] B,
    output reg [35:0] result
);

parameter IDLE = 0;
parameter A_SUM_B = 1;
parameter SUM_MUL_4 = 2;
parameter ADD_B= 3;
parameter DIV_B_2 = 4;
parameter A_MUL_4 = 5;
parameter DIV_ADD_MULL = 6;
parameter LEFT_DIV_2 = 7;
parameter ADD_ALL = 8;
parameter FINISH = 9;

reg [3:0] cur_state, next_state;
reg [35:0] adder_1, mult_1, div_1, adder_2;


always @(posedge clk, negedge reset) begin
    if (!reset)
        cur_state <= IDLE;
    else if(cur_state != IDLE)
```

```
        cur_state <= next_state;
    else if(valid_data) begin
        result <= 0;
        cur_state <= next_state;
    end

end

always @(*) begin
case (cur_state)
        IDLE:
            next_state = A_SUM_B;
        A_SUM_B:
            next_state = SUM_MUL_4;
        SUM_MUL_4:
            next_state = ADD_B;
        ADD_B:
            next_state = DIV_B_2;
        DIV_B_2:
            next_state = A_MUL_4;
        A_MUL_4:
            next_state = DIV_ADD_MULL;
        DIV_ADD_MULL:
            next_state = LEFT_DIV_2;
        LEFT_DIV_2:
            next_state = ADD_ALL;
        ADD_ALL:
            next_state = FINISH;
        FINISH:
            next_state = IDLE;
        default:
            next_state = IDLE;
    endcase
end

always @(posedge clk) begin
    case (cur_state)
        IDLE:
            ;
        A_SUM_B:
            adder_1 <= A + B;
        SUM_MUL_4:
            mult_1 <= adder_1 << 2;
        ADD_B:
            adder_1 <= mult_1 + B;
        DIV_B_2:
            div_1 <= B >> 1;
        A_MUL_4:
            mult_1 <= A << 2;
        DIV_ADD_MULL:
            adder_2 <= mult_1 + div_1;
```

```
        LEFT_DIV_2:
            div_1 <= adder_1 >> 1;
        ADD_ALL:
            adder_1 <= div_1 + adder_2;
        FINISH:
            result <= adder_1;
        default:
            next_state <= IDLE;
    endcase
end
endmodule
```

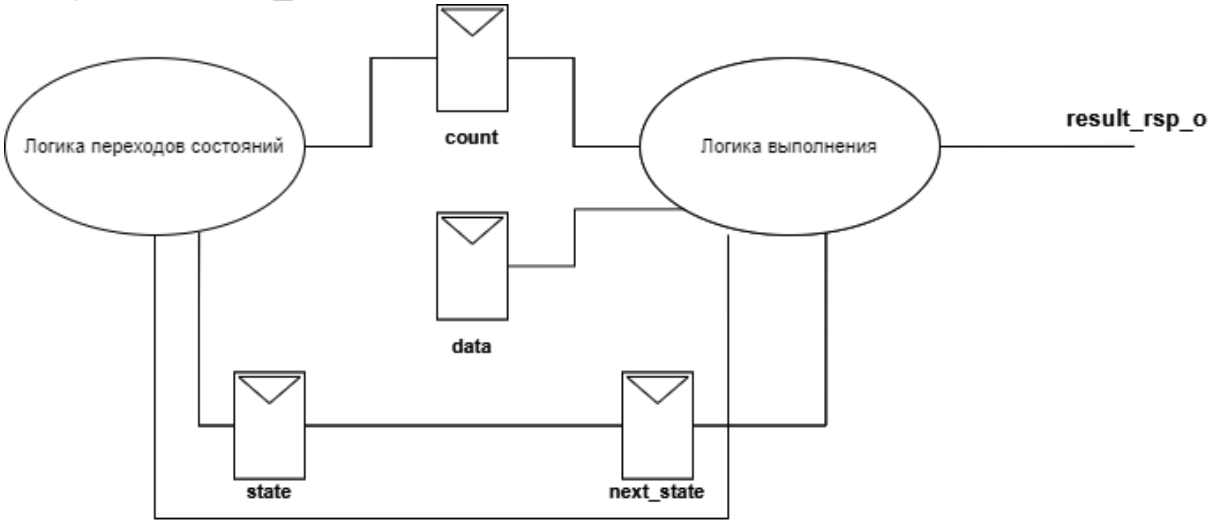
Тестовый план:

1) Проверка работы сброса значения
2) Подача сигнала валидности входных данных. Проверка вычисления на одной паре чисел (A =3; B = 8)
3) Подача сигнала валидности входных данных. Проверка вычисления на второй паре чисел (A=32; B=128)



временная диаграмма работы тестового окружения модуля fsm_4

Модуль COUNT_FREE:



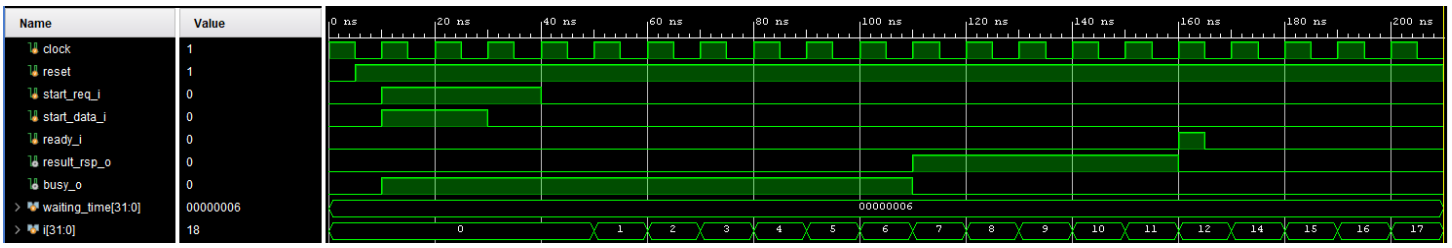
Код:

```
module count_free(
    input clk,
    input rst,
    input start_req_i,
    input start_data_i,
    input ready_i,
    output reg result_rsp_o,
    output reg busy_o
);

parameter IDLE = 0, COUNTING = 1, WAITING = 2, DONE = 3;
reg [31:0] count;
reg [31:0] data = 31'd0;
reg [1:0] state;
reg [1:0] next_state = IDLE;

always @(*) begin
    case (state)
        IDLE: begin
            next_state = COUNTING;
            result_rsp_o = 0;
            busy_o = 0;
        end
        COUNTING: begin
            next_state = WAITING;
            busy_o = 1;
        end
        WAITING: begin
            next_state = DONE;
            busy_o = 1;
        end
        DONE: begin
            next_state = IDLE;
            result_rsp_o = 1;
```

```
            busy_o = 0;
        end
    endcase
end
always @ (posedge clk, negedge rst) begin
    if (!rst) begin
        state <= IDLE;
        count <= 0;
        data <= 0;
    end else begin
        case (state)
            IDLE: begin
                if (start_req_i) begin
                    data <= start_data_i;
                    state <= next_state;
                end
            end
            COUNTING: begin
                if (start_req_i) begin
                    data <= (data << 1) + start_data_i;
                end else begin
                    count <= 0;
                    state <= next_state;
                end
            end
            WAITING: begin
                if (count == data) state <= next_state;
                else count <= count + 1;
            end
            DONE: begin
                if (ready_i) state <= next_state;
            end
        endcase
    end
end
endmodule
```

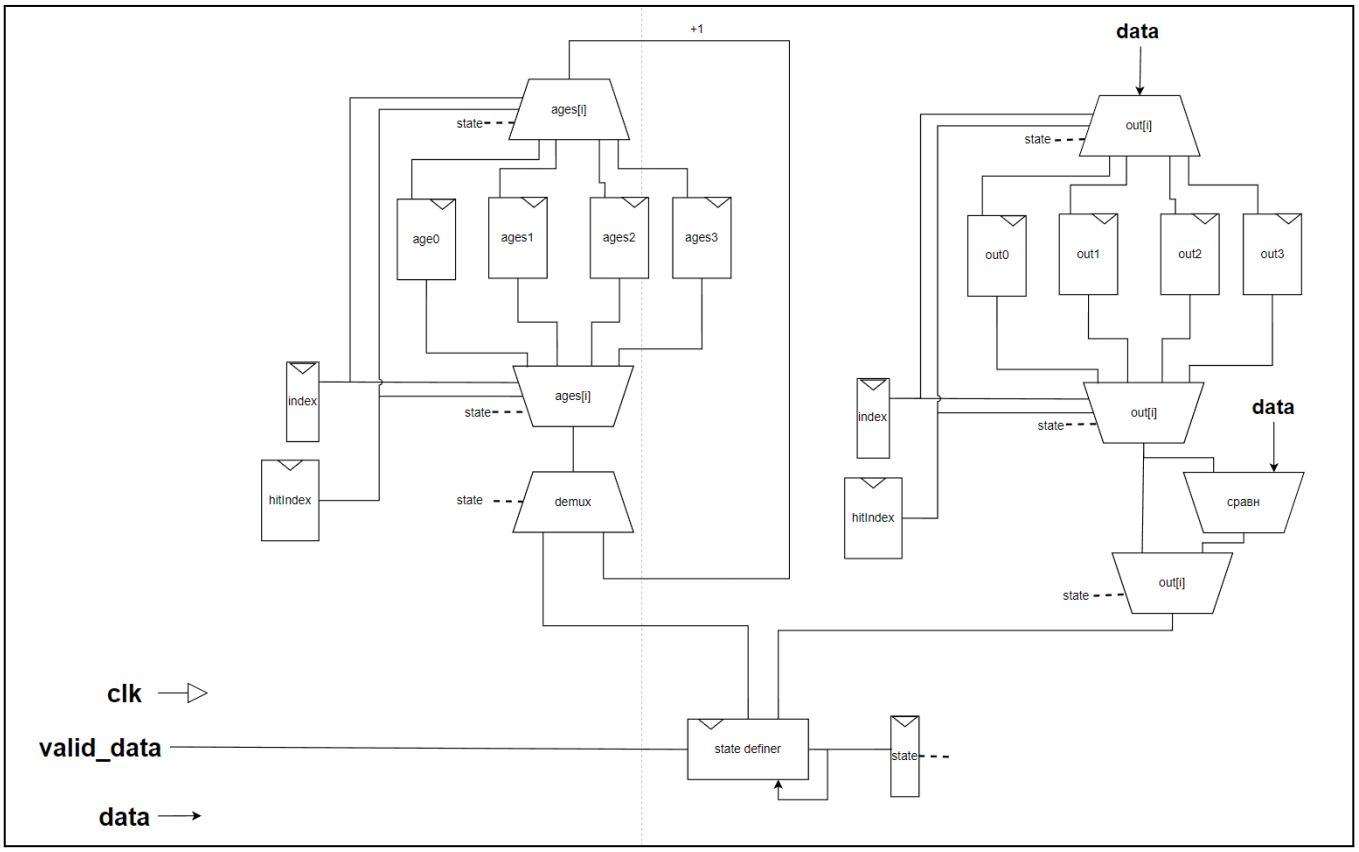
Тестовый план:

1) Произвести сброс.
2) Загрузить в модуль число 6.
3) Подать сигнал готовности внешнего источника спустя 12 тактов после окончания записи числа в модуль.



временная диаграмма работы тестового окружения модуля `count_free`

Модуль BUFFER_LRU:



Код:

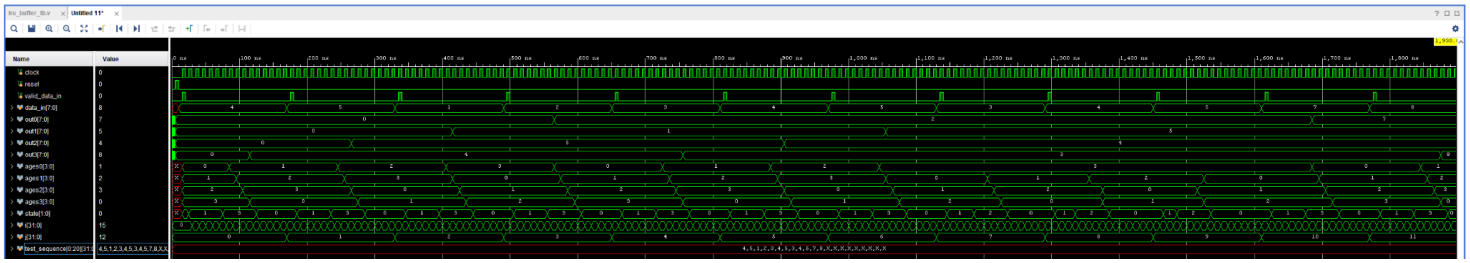
```
`timescale 1ns / 1ps

module lru_buffer(clk, rst, valid_data, data,
                  out0, out1, out2, out3);
    input clk, rst, valid_data;
    input [7:0] data;
    output reg [7:0] out0, out1, out2, out3;;

    reg[2:0] state;

    reg [2:0] ages[0:3];
    reg [2:0] hitIndex;
    reg [2:0] index;

    integer j;

    parameter IDLE = 0, CHECKING_HIT = 1, HIT_UPDATING = 2, nHIT_UPDATING = 3;

    always @ (posedge rst) begin
        out0 <= 7'd0;
        out1 <= 7'd0;
        out2 <= 7'd0;
```

```
        out3 <= 7'd0;
        hitIndex <= 4'd0;
        for(j = 0; j < 4; j = j + 1) ages[j] <= j;
        state <= IDLE;
    end

    always @ (posedge clk) begin
        case(state)
            IDLE: begin
                if(valid_data) begin
                    state <= CHECKING_HIT;
                    hitIndex <= 0;
                    index <= 0;
                end
            end
            CHECKING_HIT: begin
                if(hitIndex > 3) state <= nHIT_UPDATING;
                else begin
                    case(hitIndex)
                            0:  begin
                                if(out0 == data) state <= HIT_UPDATING;
                                else hitIndex <= hitIndex + 1;
                            end
                            1:  begin
                                if(out1 == data) state <= HIT_UPDATING;
                                else hitIndex <= hitIndex + 1;
                            end
                            2:  begin
                                if(out2 == data) state ⩽ HIT_UPDATING;
                                else hitIndex <= hitIndex + 1;
                            end
                            3:  begin
                                if(out3 == data) state ⩽ HIT_UPDATING;
                                else hitIndex <= hitIndex + 1;
                            end
                    endcase

                end
            end
            HIT_UPDATING: begin
                if(index > 3) begin
                    state <= IDLE;
                    ages[hitIndex] = 0;
                end else begin
                    if(ages[index] < ages[hitIndex]) ages[index] <= ages[index] + 1;
                    index <= index + 1;
                end
            end
            nHIT_UPDATING: begin
                if(index > 3) state <= IDLE;
                else begin
```

```
                    if(ages[index] == 3) begin
                        ages[index] <= 0;
                        case(index)
                            0:  out0 <= data;
                            1:  out1 <= data;
                            2:  out2 <= data;
                            3:  out3 <= data;
                        endcase
                    end else ages[index] <= ages[index] + 1;
                    index <= index + 1;
                end
            end
        endcase
    end
endmodule
```

Тестовый план:

последовательность: 4 5 1 2 3 4 5 3 4 5 7 8



временная диаграмма работы тестового окружения модуля buffer_lru

```
inserted           4, out = [  0,   0,   0,   4]
inserted           5, out = [  0,   0,   5,   4]
inserted           1, out = [  0,   1,   5,   4]
inserted           2, out = [  2,   1,   5,   4]
inserted           3, out = [  2,   1,   5,   3]
inserted           4, out = [  2,   1,   4,   3]
inserted           5, out = [  2,   5,   4,   3]
inserted           3, out = [  2,   5,   4,   3]
inserted           4, out = [  2,   5,   4,   3]
inserted           5, out = [  2,   5,   4,   3]
inserted           7, out = [  7,   5,   4,   3]
inserted           8, out = [  7,   5,   4,   8]
```

состояние буфера во время работы модуля

Вывод:

В результате проделанной лабораторной работы мы получили представления о принципах работы с последовательной логикой**.**

В ходе работы были реализованы модули на языке

описание аппаратуры verilog, также были написаны тесты для каждого из них. Построены схемы и описаны тестовые схемы.