Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

# Лабораторная работа №2
# Дисциплина «Функциональная схемотехника»
## Разработка и верификация цифровых последовательностных схем

## Вариант: Буфер LRU

Выполнили:
Кузнецов Д. А.
Баянов Р. Д.

Группа P3334

Преподаватели:
Кустарев Павел Валерьевич
Васильев Сергей Михайлович

Санкт-Петербург
2024

## Цели работы:

Разработать цифровой сложно-функциональный блок, который обрабатывает и хранит данные в соответствии с требованиями, указанными в варианте задания. Разработка включает в себя следующие этапы:

1. Определение и согласование интерфейсов;
2. Выбор алгоритма и согласование микроархитектуры;
3. Описание устройства средствами HDL;
4. Функциональная верификация;
5. Прототипирование разработанного блока на FPGA.

## Задание:

**Вариант.** Буфер LRU.
Это память особого вида, в которой значения вытесняются согласно политике "Least Recently Used". У памяти должно быть два порта для записи данных и один порт для чтения. Количество ячеек в памяти должно быть равно 8. При записи элементов в переполненный буфер некоторые элементы должны быть вытеснены согласно указанной ранее политике и новый элемент должен быть беспрепятственно размещён в буфере. При чтении элемент из буфера не вытесняется, но вы должны регистрировать обращение к элементу. Именно на основе данных, запрашиваемых на чтение, вы и должны формировать кандидата на вытеснение.

**Объем буфера:** 8 ячеек.
**Формат данных:** 8-битное беззнаковое целое число.
**Протокол:** Ready-Valid.
**Особенность:** Обработка ситуации одновременной записи двух чисел.

1. **Согласованный с преподавателем интерфейс:**
   Тактовый сигнал (**clk**): подается от тактового генератора платы. Частота работы — 100 МГц.
   Сброс (**reset**): Сброс всей очереди в начальное состояние.

   Добавление элемента (**in_valid**): Активный сигнал записи данных.
   Чтение элемента (**out_ready**): Активный сигнал чтения данных.
   Индекс доступа при чтении (**access_index**): Индекс, значение которого мы хотим получить.
   Входные данные (**in_data**): Данные, которые необходимо записать.
   Выходные данные (**out_data**): Массив, куда будут записаны выходные данные.

```
module LRU_Buffer #(
    parameter CACHE_SIZE = 8,
    parameter DATA_SIZE = 8,
    parameter MAX_SIZE = 2000
)(
    input wire clk,
    input wire reset,

    input wire in_valid,      // источник готов отдать данные
    input wire [DATA_SIZE - 1:0] in_data,
    output reg in_ready,      // мы готовы принять данные с источника

    output reg [DATA_SIZE - 1:0] out_data,
    input wire out_ready,     // получатель готов принять данные
    output reg out_valid,     // мы готовы отдать данные

    input [$clog2(CACHE_SIZE)-1:0] access_index
);
```

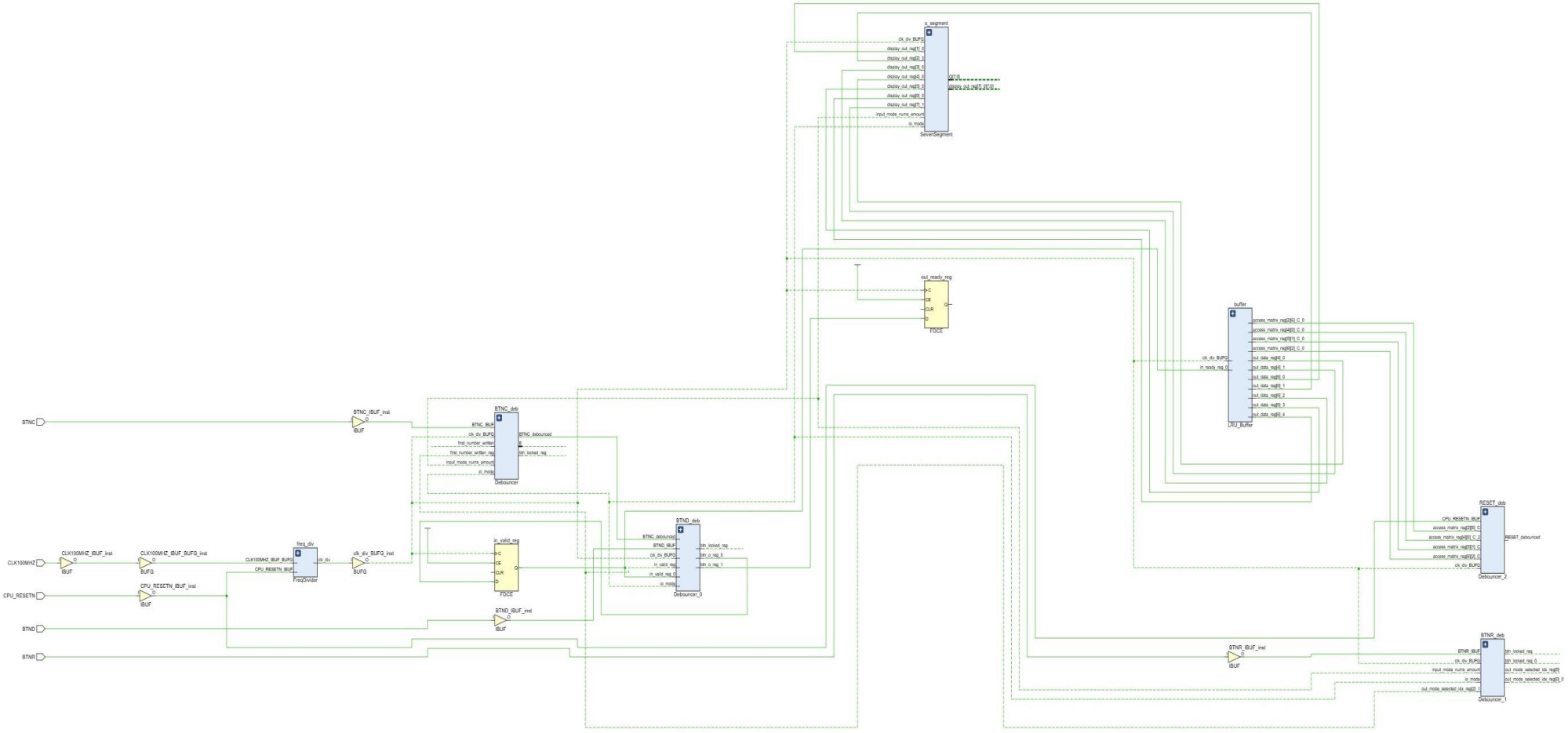
## 2. Выбор алгоритма и согласование микроархитектуры

Алгоритм работы

Структура буфера: Кеш представлен в виде обычного массива. При этом реализована матрица приоритетов, для корректного вытеснения. Объем кеша — 8 элементов.

Логика чтения и записи: Запись производится либо по порядку, либо на позицию наиболее давно используемого элемента. Приоритет вставленного элемента обновляется.

При чтении данные извлекаются из позиции, которую указал пользователь, при этом обновляется приоритет элемента, к которому было обращение.




## Диаграмма микроархитектур

# 3. Описание устройства средствами HDL.

Debouncer.v

```
1. timescale 1ns / 1ps
2.
3. module Debouncer
4.     #(parameter WAIT_CLOCKS = 100_000)
5.     (
6.         input clk_i,
7.         input btn_i,
8.         output reg btn_o
9.     );
10.
11.     reg [$clog2(WAIT_CLOCKS)-1:0] cnt;
12.
13.     always @(posedge clk_i) begin
14.         if (btn_i) begin
15.             if (cnt == WAIT_CLOCKS) begin
16.                 btn_o <= 1;
17.             end else begin
18.                 cnt <= cnt + 1;
19.             end
20.         end else begin
21.             cnt <= 0;
22.             btn_o <= 0;
23.         end
24.     end
25. endmodule
```

```
26.
```

FreqDivider.v

```
1. timescale 1ns / 1ps
2.
3. module FreqDivider(
4.     input wire clk_in,
5.     input wire reset,
6.     output reg clk_out
7. );
8.     reg toggle;
9.
10.     always @(posedge clk_in or posedge reset) begin
11.         if (reset) begin
12.             toggle <= 1'b0;
13.             clk_out <= 1'b0;
14.         end else begin
15.             toggle <= ~toggle;
16.             clk_out <= toggle;
17.         end
18.     end
19. endmodule
20.
```

LRU_Buffer.v

```
1. timescale 1ns / 1ps
2.
3. module LRU_Buffer #(
4.     parameter CACHE_SIZE = 8,
5.     parameter DATA_SIZE = 8,
6.     parameter MAX_SIZE = 2000
7. )(
8.     input wire clk,
9.     input wire reset,
10.
11.     input wire in_valid,     // источник готов отдать данные
12.     input wire [DATA_SIZE - 1:0] in_data,
13.     output reg in_ready,     // мы готовы принять данные с источника
```

```
14.
15.     output reg [DATA_SIZE - 1:0] out_data,
16.     input wire out_ready,    // получатель готов принять данные
17.     output reg out_valid,    // мы готовы отдать данные
18.
19.     input [$clog2(CACHE_SIZE)-1:0] access_index
20. );
21.
22.     reg [DATA_SIZE - 1:0] cache_data [0:CACHE_SIZE - 1];
23.     reg [CACHE_SIZE-1:0] access_matrix [0:CACHE_SIZE-1];
24.
25.     integer i;
26.     integer j;
27.
28.     reg [3:0] zeroes_count [7:0];
29.     reg [3:0] max_zeroes;
30.
31.     reg [$clog2(CACHE_SIZE)-1:0] replace_index;
32.
33.     always @(posedge clk or posedge reset) begin
34.         if (reset) begin
35.             out_valid = 0;   // мы не готовы отдать данные
36.             in_ready = 1'b0; // мы не готовы принять данные с источника
37.
38.             zeroes_count[0] = 4'b0;
39.             zeroes_count[1] = 4'b0;
40.             zeroes_count[2] = 4'b0;
41.             zeroes_count[3] = 4'b0;
42.             zeroes_count[4] = 4'b0;
43.             zeroes_count[5] = 4'b0;
44.             zeroes_count[6] = 4'b0;
45.             zeroes_count[7] = 4'b0;
46.
47.             cache_data[0] = 8'b0;
48.             cache_data[1] = 8'b0;
49.             cache_data[2] = 8'b0;
50.             cache_data[3] = 8'b0;
51.             cache_data[4] = 8'b0;
52.             cache_data[5] = 8'b0;
53.             cache_data[6] = 8'b0;
54.             cache_data[7] = 8'b0;
55.
56.             access_matrix[0] = 8'b0;
57.             access_matrix[1] = 8'b0;
58.             access_matrix[2] = 8'b0;
59.             access_matrix[3] = 8'b0;
60.             access_matrix[4] = 8'b0;
61.             access_matrix[5] = 8'b0;
62.             access_matrix[6] = 8'b0;
63.             access_matrix[7] = 8'b0;
64.
65.             max_zeroes = 4'b0;
66.             replace_index = 3'b0;
67.
68.             in_ready = 1'b1; // мы готовы принять данные с источника
69.
70.         end else begin
71.
72.             if (in_valid && in_ready) begin
73.
74.                 // Ищем наименее приоритетный элемент
75.                 replace_index = 3'b0;
76.                 max_zeroes = 4'b0;
77.
78.                 for (i = 0; i < 8; i = i + 1) begin
79.                     zeroes_count[i] =
80.                         (access_matrix[i][0] == 0) +
81.                         (access_matrix[i][1] == 0) +
82.                         (access_matrix[i][2] == 0) +
83.                         (access_matrix[i][3] == 0) +
84.                         (access_matrix[i][4] == 0) +
85.                         (access_matrix[i][5] == 0) +
86.                         (access_matrix[i][6] == 0) +
87.                         (access_matrix[i][7] == 0);
```

```
            end

            for (i = 0; i < 8; i = i + 1) begin
                if (zeroes_count[i] > max_zeroes) begin
                    max_zeroes = zeroes_count[i];
                    replace_index = i;
                end
            end

            // Обновляем приоритет
            for (i = 0; i < 8; i = i + 1) begin
                access_matrix[replace_index][i] <= 1'b1;
                access_matrix[i][replace_index] <= 1'b0;
            end
            access_matrix[replace_index][replace_index] <= 1'b0;

            // Обновляем данные в кэше
            cache_data[replace_index] <= in_data;
            in_ready <= 1'b0; // Мы не готовы принять данные с источника

        end
        if (!in_valid) begin
            in_ready <= 1'b1; // мы готовы принять данные с источника
        end
        if (out_ready) begin

            out_data <= cache_data[access_index];

            // обновляем приоритет
            for (i = 0; i < 8; i = i + 1) begin
                access_matrix[access_index][i] <= 1'b1;
                access_matrix[i][access_index] <= 1'b0;
            end
            access_matrix[access_index][access_index] <= 1'b0;

            out_valid <= 1'b1;
        end else begin
            out_valid <= 0; // если источник считал прошлые данные, мы не готовы отдавать новые
        end
    end
  end

endmodule
```

SevenSegment.v

```
module SevenSegment(
    input clk,

    input wire [7:0] input_number,
    input wire io_mode,                 // 0 - output, 1 - input
    input wire [2:0] out_mode_selected_idx,
    input wire input_mode_nums_amount,      // 0 - one number, 1 - two numbers input

    output reg [7:0] AN,
    output reg CA,
    output reg CB,
    output reg CC,
    output reg CD,
    output reg CE,
    output reg CF,
    output reg CG,
    output reg DP
);

    reg [7:0]display_out;
    reg [31:0] counter;

    always @* begin
        {CA, CB, CC, CD, CE, CF, CG, DP} = display_out;
```

```
25.  end
26.
27.  always @(posedge clk) begin
28.     counter <= counter + 1;
29.     if(counter <= 40000) begin                    AN = 8'b11111110;
30.        case (input_number[3:0])
31.           4'b0000: display_out = 8'b00000011;
32.           4'b0001: display_out = 8'b10011111;  //one
33.           4'b0010: display_out = 8'b00100101;  //two
34.           4'b0011: display_out = 8'b00001101;  //three
35.           4'b0100: display_out = 8'b10011001;  //four
36.           4'b0101: display_out = 8'b01001001;  //five
37.           4'b0110: display_out = 8'b01000001;  //six
38.           4'b0111: display_out = 8'b00011111;  //seven
39.           4'b1000: display_out = 8'b00000001;  //eight
40.           4'b1001: display_out = 8'b00001001;  //nine
41.           4'b1010: display_out = 8'b00010001;  //A
42.           4'b1011: display_out = 8'b11000001;  //b
43.           4'b1100: display_out = 8'b01100011;  //C
44.           4'b1101: display_out = 8'b10000101;  //d
45.           4'b1110: display_out = 8'b01100001;  //E
46.           4'b1111: display_out = 8'b01110001;  //F
47.        endcase
48.     end if(counter > 40000 && counter <= 80000) begin      AN = 8'b11111101;
49.        case (input_number[7:4])
50.           4'b0000:          display_out = 8'b00000011;
51.           4'b0001:          display_out = 8'b10011111;  //one
52.           4'b0010:          display_out = 8'b00100101;  //two
53.           4'b0011:          display_out = 8'b00001101;  //three
54.           4'b0100:          display_out = 8'b10011001;  //four
55.           4'b0101:          display_out = 8'b01001001;  //five
56.           4'b0110:          display_out = 8'b01000001;  //six
57.           4'b0111:          display_out = 8'b00011111;  //seven
58.           4'b1000:          display_out = 8'b00000001;  //eight
59.           4'b1001:          display_out = 8'b00001001;  //nine
60.           4'b1010:          display_out = 8'b00010001;  //A
61.           4'b1011:          display_out = 8'b11000001;  //b
62.           4'b1100:          display_out = 8'b01100011;  //C
63.           4'b1101:          display_out = 8'b10000101;  //d
64.           4'b1110:          display_out = 8'b01100001;  //E
65.           4'b1111:          display_out = 8'b01110001;  //F
66.        endcase
67.     end
68.     if(counter > 80000 && counter <= 120000) begin        AN = 8'b11111011;
69.        display_out = 8'b00000011;
70.     end
71.     if(counter > 120000 && counter <= 160000) begin       AN = 8'b11110111;
72.        display_out = 8'b00000011;
73.     end
74.     if(counter > 160000 && counter <= 200000) begin       AN = 8'b11101111;
75.        if (io_mode) begin
76.           case (input_mode_nums_amount)
77.              1'b0:          display_out = 8'b10011111;  //one
78.              1'b1:          display_out = 8'b00100101;  //two
79.           endcase
80.        end
81.        else begin
82.           case (out_mode_selected_idx)
83.              3'b000:          display_out = 8'b00000011;
84.              3'b001:          display_out = 8'b10011111;  //one
85.              3'b010:          display_out = 8'b00100101;  //two
86.              3'b011:          display_out = 8'b00001101;  //three
87.              3'b100:          display_out = 8'b10011001;  //four
88.              3'b101:          display_out = 8'b01001001;  //five
89.              3'b110:          display_out = 8'b01000001;  //six
90.              3'b111:          display_out = 8'b00011111;  //seven
91.           endcase
92.        end
93.     end
94.     if(counter > 200000 && counter <= 240000) begin       AN = 8'b11011111;
95.        if (io_mode == 1) begin
96.           display_out = 8'b00110000; // p
97.        end
98.        if (io_mode == 0) begin
```

```
            display_out = 8'b11100000; // t
        end
    end
    if(counter > 240000 && counter <= 280000) begin        AN = 8'b10111111;
        if (io_mode == 1) begin
            display_out = 8'b11010101; // n
        end
        if (io_mode == 0) begin
            display_out = 8'b11000111; // u
        end
    end
    if(counter > 280000 && counter <= 320000) begin        AN = 8'b01111111;
        if (io_mode == 1) begin
            display_out = 8'b10011111; // I
        end
        if (io_mode == 0) begin
            display_out = 8'b00000011; // O
        end
    end
    if (counter > 320000) begin
        counter <= 0;
    end
  end
endmodule

```

ModuleWrapper.v

```
timescale 1ns / 1ps

module Module_Wrapper (
    input wire CLK100MHZ,
    input wire CPU_RESETN,

    input wire BTND, // change mode (Input, Output)
    input wire BTNR, // increase number (Input mode - one or two numbers input, Output mode - out index)
    input wire BTNC, // Input / Output

     input wire [15:0] SW,

    output wire CA,
    output wire CB,
    output wire CC,
    output wire CD,
    output wire CE,
    output wire CF,
    output wire CG,
    output wire DP,
    output wire [7:0] AN
);

    wire clk = CLK100MHZ;
    wire reset = ~CPU_RESETN;

    wire clk_div;

    FreqDivider freq_div (
        .clk_in(clk),
        .reset(reset),
        .clk_out(clk_div)
    );

    wire BTND_debounced;
    wire BTNR_debounced;
    wire BTNC_debounced;
    wire RESET_debounced;

    Debouncer BTND_deb (.clk_i(clk_div), .btn_i(BTND), .btn_o(BTND_debounced));
    Debouncer BTNR_deb (.clk_i(clk_div), .btn_i(BTNR), .btn_o(BTNR_debounced));
    Debouncer BTNC_deb (.clk_i(clk_div), .btn_i(BTNC), .btn_o(BTNC_debounced));
    Debouncer RESET_deb (.clk_i(clk_div), .btn_i(reset), .btn_o(RESET_debounced));

```

```
45.   reg io_mode; // 1 - input, 0 - output
46.   wire [7:0] first_number = SW[7:0];
47.   wire [7:0] second_number = SW[15:8];
48.
49.   reg out_ready; // мы готовы принять данные
50.   reg in_valid;  // мы готовы отдать данные
51.
52.   wire out_valid; // подмодуль готов отдать данные
53.   wire in_ready;  // подмодуль готов принять данные
54.
55.   wire [$clog2(8)-1:0] access_index;
56.
57.   wire [7:0] buffer_out_data;
58.   reg [7:0] buffer_in_data;
59.
60.   // Seven Segment bus
61.   reg [2:0] out_mode_selected_idx;
62.
63.   reg input_mode_nums_amount;      // 0 - one number, 1 - two numbers input
64.   reg first_number_written;
65.
66.   LRU_Buffer buffer (
67.       .clk(clk_div),
68.       .reset(RESET_debounced),
69.       .in_valid(in_valid),
70.       .in_data(buffer_in_data),
71.       .in_ready(in_ready),
72.       .out_data(buffer_out_data),
73.       .out_ready(out_ready),
74.       .out_valid(out_valid),
75.       .access_index(out_mode_selected_idx)
76.   );
77.
78.   SevenSegment s_segment (
79.       .clk(clk_div),
80.       .input_number(buffer_out_data),
81.       .io_mode(io_mode),                              // 1 - input, 0 - output
82.       .out_mode_selected_idx(out_mode_selected_idx),
83.       .input_mode_nums_amount(input_mode_nums_amount),      // 0 - one number, 1 - two numbers input
84.       .AN(AN),
85.       .CA(CA),
86.       .CB(CB),
87.       .CC(CC),
88.       .CD(CD),
89.       .CE(CE),
90.       .CF(CF),
91.       .CG(CG),
92.       .DP(DP)
93.   );
94.
95.   reg [31:0] block_cnt;       // Счётчик блокировки (например, на 1 секунду)
96.   reg btn_locked;             // Флаг блокировки всех кнопок
97.
98.   always @(posedge clk_div or posedge RESET_debounced) begin
99.       if (RESET_debounced) begin
100.          in_valid <= 0;
101.          out_ready <= 0;
102.          io_mode <= 1'b1;
103.          first_number_written <= 0;
104.          input_mode_nums_amount <= 0;
105.          out_mode_selected_idx <= 0;
106.          buffer_in_data <= 0;
107.          block_cnt <= 0;
108.          btn_locked <= 0;
109.      end else begin
110.          // Если кнопки заблокированы, увеличиваем счётчик
111.          if (btn_locked) begin
112.              if (block_cnt == 10_000_000) begin  // 1 секунда при 50 MHz
113.                  btn_locked <= 0;  // Разблокируем кнопки после 1 секунды
114.                  block_cnt <= 0;
115.              end else begin
116.                  block_cnt <= block_cnt + 1;
117.              end
118.          end else begin
```

```
            // Если кнопки не заблокированы
            if (BTND_debounced) begin
                io_mode <= ~io_mode;
                out_ready <= 0;
                in_valid <= 0;
            end

            if (BTNR_debounced) begin
                if (!io_mode) begin                 // Output
                    if (out_mode_selected_idx == 7) begin
                        out_mode_selected_idx <= 0;
                    end else begin
                        out_mode_selected_idx <= out_mode_selected_idx + 1;
                    end
                end else begin                      // Input
                    input_mode_nums_amount <= ~input_mode_nums_amount;
                end
            end

            // Main logic processing
            if (BTNC_debounced) begin
                if (io_mode) begin                                  // Input mode
                    if (!in_ready) begin
                        in_valid <= 0;
                    end
                    if (!input_mode_nums_amount) begin // Input one number
                        buffer_in_data <= first_number;
                        in_valid <= 1;
                    end else begin                                  // Input two numbers
                        if (!first_number_written) begin
                            buffer_in_data <= first_number;
                            in_valid <= 1;
                            first_number_written <= 1'b1;
                        end else begin
                            buffer_in_data <= second_number;
                            in_valid <= 1;
                        end
                    end
                end else begin                                      // Output mode
                    out_ready <= 1;
                end
            end

            btn_locked <= 1;  // Блокируем кнопки после нажатия
            block_cnt <= 0;  // Сбрасываем счётчик блокировки
        end
    end
  end

endmodule

```

# 4. Функциональная верификация;

```
timescale 1ns / 1ps

ifndef SYNTHESIS
module Test_Environment();

    parameter CACHE_SIZE = 8;
    parameter DATA_SIZE = 8;

    reg clk;
    reg reset;
    reg in_valid;
    reg [DATA_SIZE - 1:0] in_data;
    reg out_ready;
```

```
14.  reg [$clog2(CACHE_SIZE)-1:0] access_index;
15.
16.  wire in_ready;
17.  wire [DATA_SIZE - 1:0] out_data;
18.  wire out_valid;
19.
20.  LRU_Buffer #(
21.      .CACHE_SIZE(CACHE_SIZE),
22.      .DATA_SIZE(DATA_SIZE)
23.  ) uut (
24.      .clk(clk),
25.      .reset(reset),
26.      .in_valid(in_valid),
27.      .in_data(in_data),
28.      .in_ready(in_ready),
29.      .out_data(out_data),
30.      .out_ready(out_ready),
31.      .out_valid(out_valid),
32.      .access_index(access_index)
33.  );
34.
35.  initial begin
36.      clk = 0;
37.      forever #5 clk = ~clk; // Переключение через 5 нс
38.  end
39.
40.  integer i;
41.
42.  initial begin
43.      reset = 1;
44.      in_valid = 0;
45.      out_ready = 0;
46.      access_index = 0;
47.      #10;
48.      reset = 0;
49.
50.      // Test buffer load sequentially
51.      $display("Test 1: buffer loading sequentially");
52.      #10;
53.
54.      for (i = 1; i <= 8; i = i + 1) begin
55.          in_data = 8'h0 + i;
56.          in_valid = 1;
57.          #10;
58.          in_valid = 0;
59.          #10;
60.      end
61.
62.      for (i = 0; i < 8; i = i + 1) begin
63.          access_index = i;
64.          out_ready = 1;
65.          #10;
66.
67.          if (out_valid && out_data == (8'h0 + i + 1)) begin
68.              $display("Test 1: buffer[%0d] = %d - passed", i, out_data);
69.          end else begin
70.              $display("Test 1: buffer[%0d] = %d - failed!", i, out_data);
71.          end
72.          out_ready = 0;
73.      end
74.
75.
76.
77.      // Test buffer correct displacement
78.      $display("Test 2: correct displacement");
79.      #10;
80.      reset = 1;
81.      in_valid = 0;
82.      out_ready = 0;
83.      access_index = 0;
84.      #10;
85.      reset = 0;
86.
87.      for (i = 1; i <= 8; i = i + 1) begin
```

```
88.             in_data = 8'h0 + i;
89.             in_valid = 1;
90.             #10;
91.             in_valid = 0;
92.             #10;
93.         end
94.
95.         #10;
96.         in_data = 8'd52;
97.         in_valid = 1;
98.         #10;
99.         in_valid = 0;
100.        #10;
101.
102.        access_index = 0;
103.        out_ready = 1;
104.        #10;
105.        if (out_valid && out_data == 8'd52) begin
106.            $display("Test 2: buffer[%0d] = %d - passed", access_index, out_data);
107.        end else begin
108.            $display("Test 2: buffer[%0d] = %d - failed!", access_index, out_data);
109.        end
110.
111.        // 3x access to 2'nd element
112.        for (i = 0; i < 3; i = i + 1) begin
113.            access_index = 1;
114.            out_ready = 1;
115.            #10;
116.        end
117.
118.        // now buffer must replace 3'rd element
119.        in_data = 8'd62;
120.        in_valid = 1;
121.        #10;
122.        in_valid = 0;
123.        #10;
124.
125.        access_index = 2;
126.        out_ready = 1;
127.        #10;
128.        if (out_valid && out_data == 8'd62) begin
129.            $display("Test 2: buffer[%0d] = %d - passed", access_index, out_data);
130.        end else begin
131.            $display("Test 2: buffer[%0d] = %d - failed!", access_index, out_data);
132.        end
133.        out_ready = 0;
134.
135.
136.
137.        // Test 3: further displacement test
138.        $display("Test 3: another displacement test");
139.        #10;
140.        reset = 1;
141.        in_valid = 0;
142.        out_ready = 0;
143.        access_index = 0;
144.        #10;
145.        reset = 0;
146.
147.        for (i = 0; i < 8; i = i + 1) begin
148.            in_data = 8'h0 + i;
149.            in_valid = 1;
150.            #10;
151.            in_valid = 0;
152.            #10;
153.        end
154.
155.
156.        access_index = 0;
157.        out_ready = 1;
158.        #10;
159.        access_index = 1;
160.        #10;
161.        access_index = 2;
```

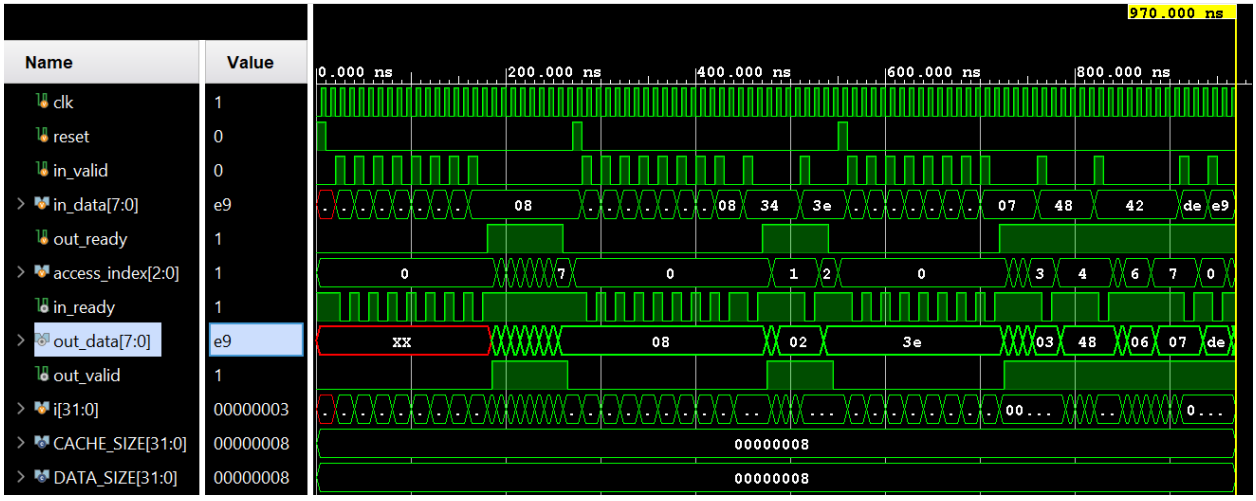
```
162.        #10;
163.        access_index = 3;
164.        #10;
165.
166.        in_data = 8'd72;
167.        in_valid = 1;
168.        #10;
169.        in_valid = 0;
170.        #10;
171.
172.        access_index = 4;
173.        out_ready = 1;
174.        #10;
175.        if (out_valid && out_data == 8'd72) begin
176.            $display("Test 3: buffer[%0d] = %d - passed", access_index, out_data);
177.        end else begin
178.            $display("Test 3: buffer[%0d] = %d - failed!", access_index, out_data);
179.        end
180.        out_ready = 0;
181.
182.        // 3x access to 5'th element
183.        for (i = 0; i < 3; i = i + 1) begin
184.            access_index = 4;
185.            out_ready = 1;
186.            #10;
187.        end
188.
189.        in_data = 8'd66;
190.        in_valid = 1;
191.        #10;
192.        in_valid = 0;
193.        #10;
194.
195.        access_index = 5;
196.        out_ready = 1;
197.        #10;
198.        if (out_valid && out_data == 8'd66) begin
199.            $display("Test 3: buffer[%0d] = %d - passed", access_index, out_data);
200.        end else begin
201.            $display("Test 3: buffer[%0d] = %d - failed!", access_index, out_data);
202.        end
203.
204.        // 4x access to 7'th element
205.        for (i = 0; i < 3; i = i + 1) begin
206.            access_index = 6;
207.            out_ready = 1;
208.            #10;
209.        end
210.
211.        // 4x access to 8'st element
212.        for (i = 0; i < 3; i = i + 1) begin
213.            access_index = 7;
214.            out_ready = 1;
215.            #10;
216.        end
217.
218.        in_data = 8'd222;
219.        in_valid = 1;
220.        #10;
221.        in_valid = 0;
222.        #10;
223.
224.        access_index = 0;
225.        out_ready = 1;
226.        #10;
227.        if (out_valid && out_data == 8'd222) begin
228.            $display("Test 3: buffer[%0d] = %d - passed", access_index, out_data);
229.        end else begin
230.            $display("Test 3: buffer[%0d] = %d - failed!", access_index, out_data);
231.        end
232.
233.        in_data = 8'd233;
234.        in_valid = 1;
235.        #10;
```

```
236.         in_valid = 0;
237.         #10;
238.
239.         access_index = 1;
240.         out_ready = 1;
241.         #10;
242.         if (out_valid && out_data == 8'd233) begin
243.             $display("Test 3: buffer[%0d] = %d - passed", access_index, out_data);
244.         end else begin
245.             $display("Test 3: buffer[%0d] = %d - failed!", access_index, out_data);
246.         end
247.
248.         $stop;
249.     end
250. endmodule
251. endif
```

Код тестирует:

- Заполнение буфера
- Чтение из буфера
- Корректное вытеснение из буфера в случае переполнения

Результаты симуляции:



## Выводы по работе

Достоинства и недостатки решения

**Достоинства:** Точный алгоритм вытеснения LRU, который исключает любые помехи и погрешности.

**Недостатки:**

Изначальная схема не влезала во временные ограничения, пришлось делить частоту в 2 раза.

**Альтернативные способы решения задачи**

- Использование FIFO с обновлением приоритетов – была бы очень сложная логика и схема занимала бы бОльшую площадь.
- Использование временных меток – переполнение времени. При нормализации из-за проблем представления вещественных чисел в ЭВМ мы не могли бы гарантировать точность работы кеша – были бы неверные записи. Но такой метод меньше по площади.

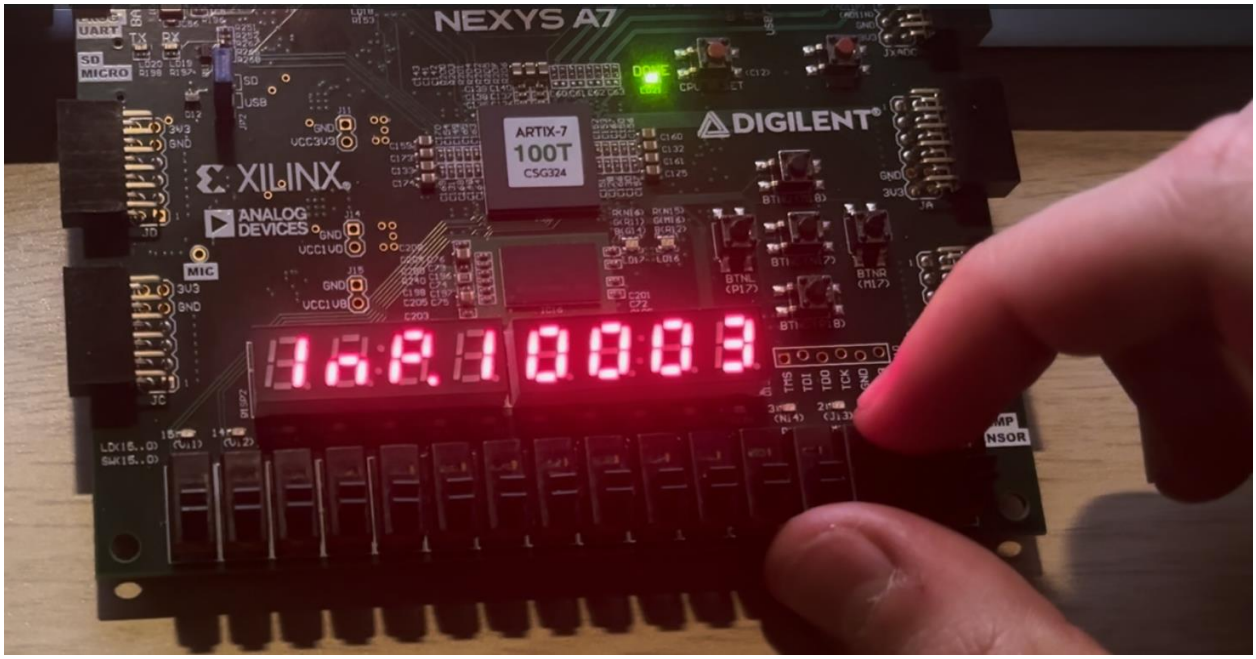
Преимущества выбранного решения

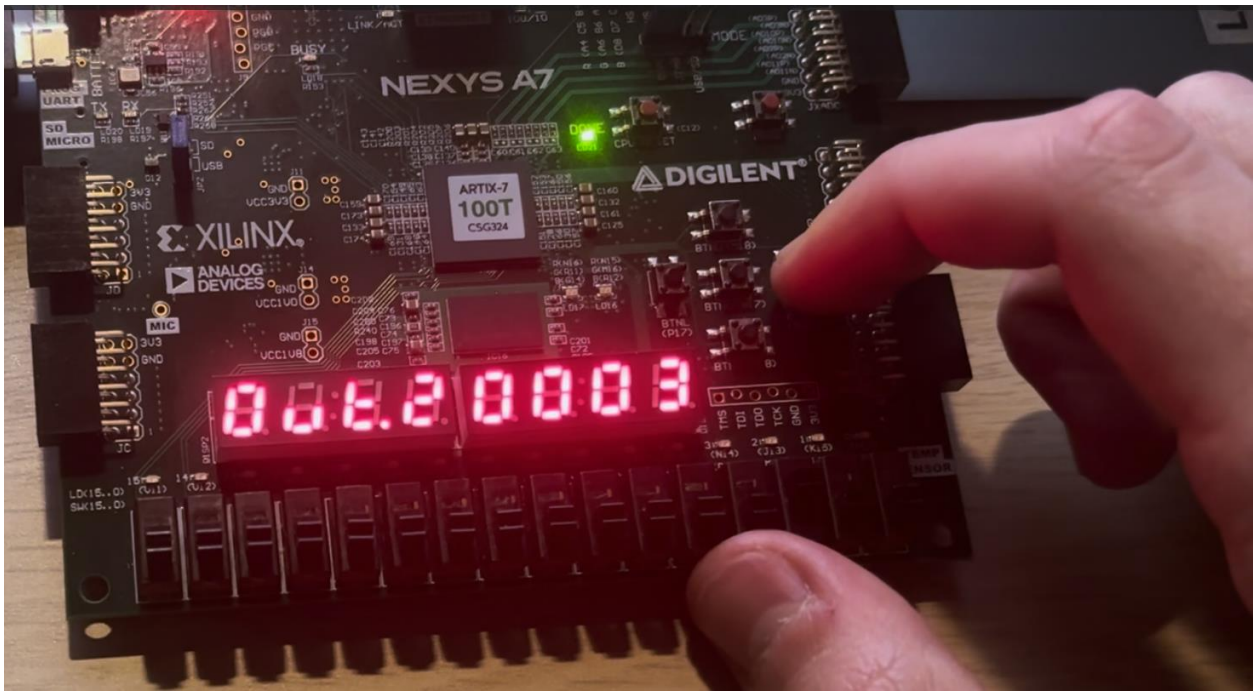
- Гарантия работоспособности. Мы покрыли все основные режимы работы: запись, чтение, одновременные операции, сброс. Проверены пограничные случаи: вытеснение при переполнении, а также последовательное заполнение при равных приоритетах.

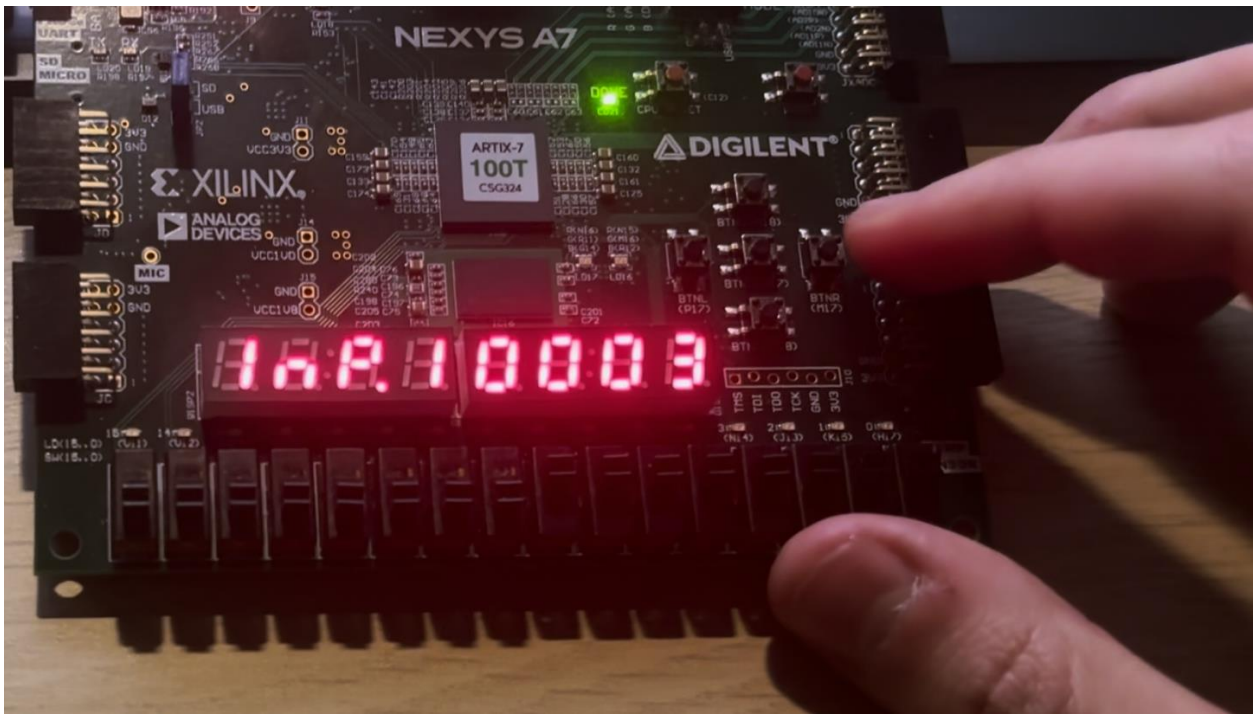
- Решение помещается на плату при синтезе (требование по площади).

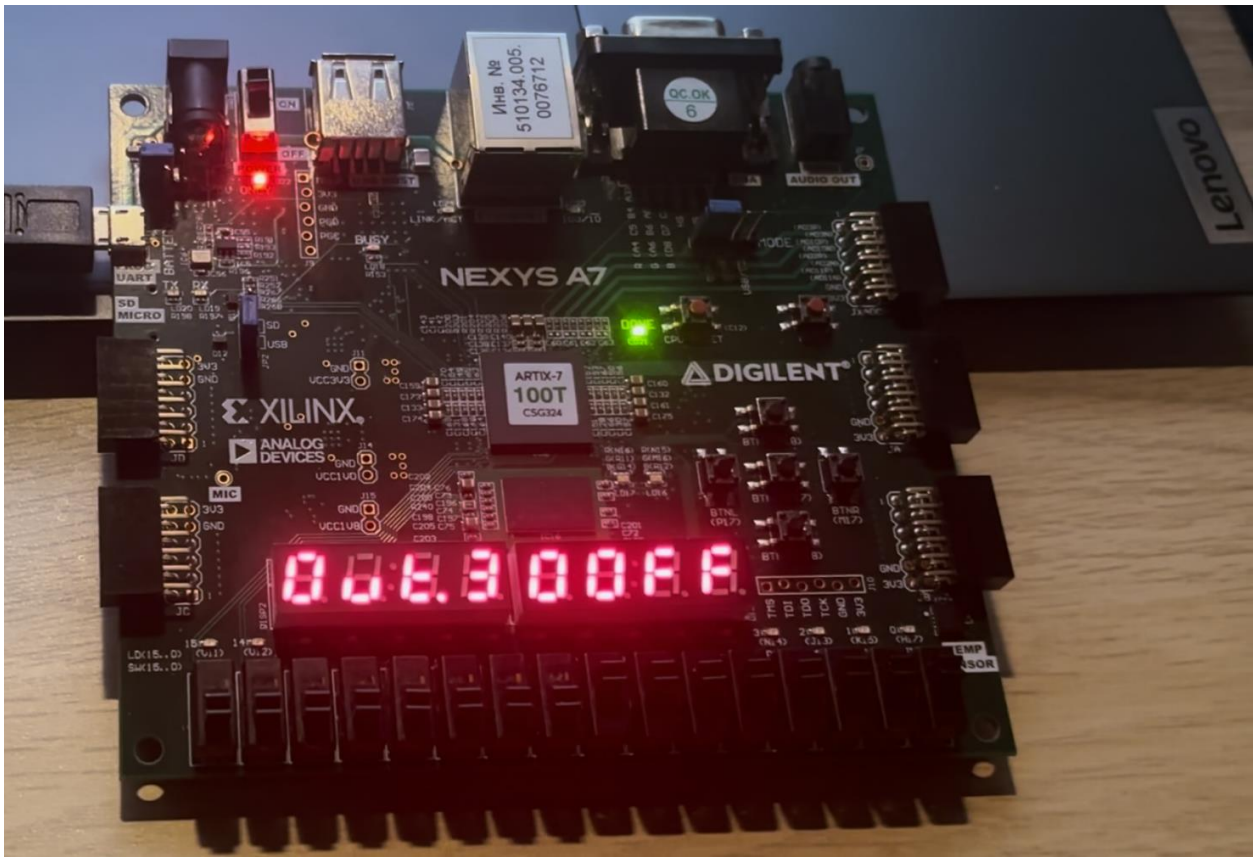
- Использованы способы проектирования схем, позволяющие корректно работать при заданной частоте (100 Mhz).
- Обработан дребезг сигналов.
- Блокировка кнопок на заданное время для избежания ложных срабатываний.

Демонстрация работы платы.




NEXYS A7
DIGILENT
XILINX
ANALOG DEVICES
ARTIX-7
100T
CSG324


NEXYS A7
DIGILENT
XILINX
ANALOG DEVICES
ARTIX-7
100T
CSG324
Lenovo

**Заключение**

Почему выбранное решение лучше альтернатив и почему оно было выбрано:

Отсутствие перезаписи и простая структура. Данный подход очень важен для встроенных систем. Также наша реализация хранит данные во внутренней памяти, что увеличивает скорость работы алгоритма, и его эффективность.

Проблемы, возникшие при реализации:

- Сложно было организовать in_ready, in_valid, out_ready и out_valid. – ломался мозг.
- Долго не могли понять в чем проблема выхода за временные рамки – другие реализации LRU не помогли, пришлось делить частоту.
- Очень долго не могли разобраться почему плата выдаёт случайные числа в индексах и кнопки работают через – оказалось, что одно нажатие без отпускания регистрирует очень много ложных срабатываний, что логично, следовательно ввод и вывод прокручивались много раз подряд, ломая логику. Было принято решение блокировать ввод кнопок после успешного срабатывания на 0.1 секунды.