Факультет Программной инженерии и компьютерной техники

Лабораторная работа №4

Архитектура гетерогенной СнК

по дисциплине «Информационная безопасность»

Вариант - 4

Группа: P3432

Выполнили:

Глотов Егор Дмитриевич

Чмурова Мария Владиславовна

Ефимов Арслан Альбертович

Преподаватель:

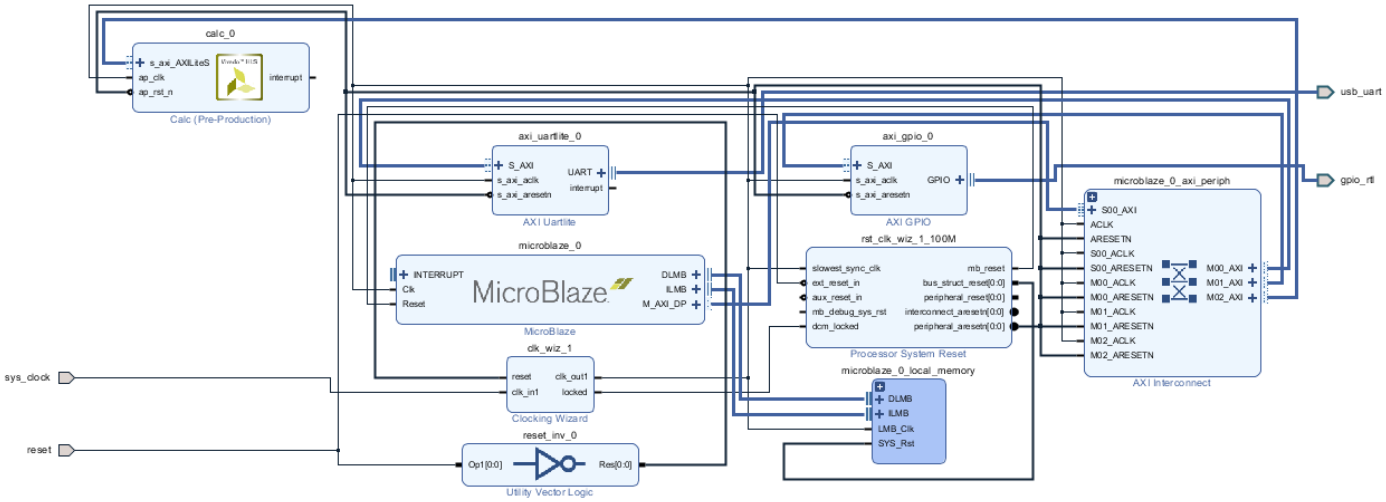
Быковский Сергей Вячеславович

г. Санкт-Петербург

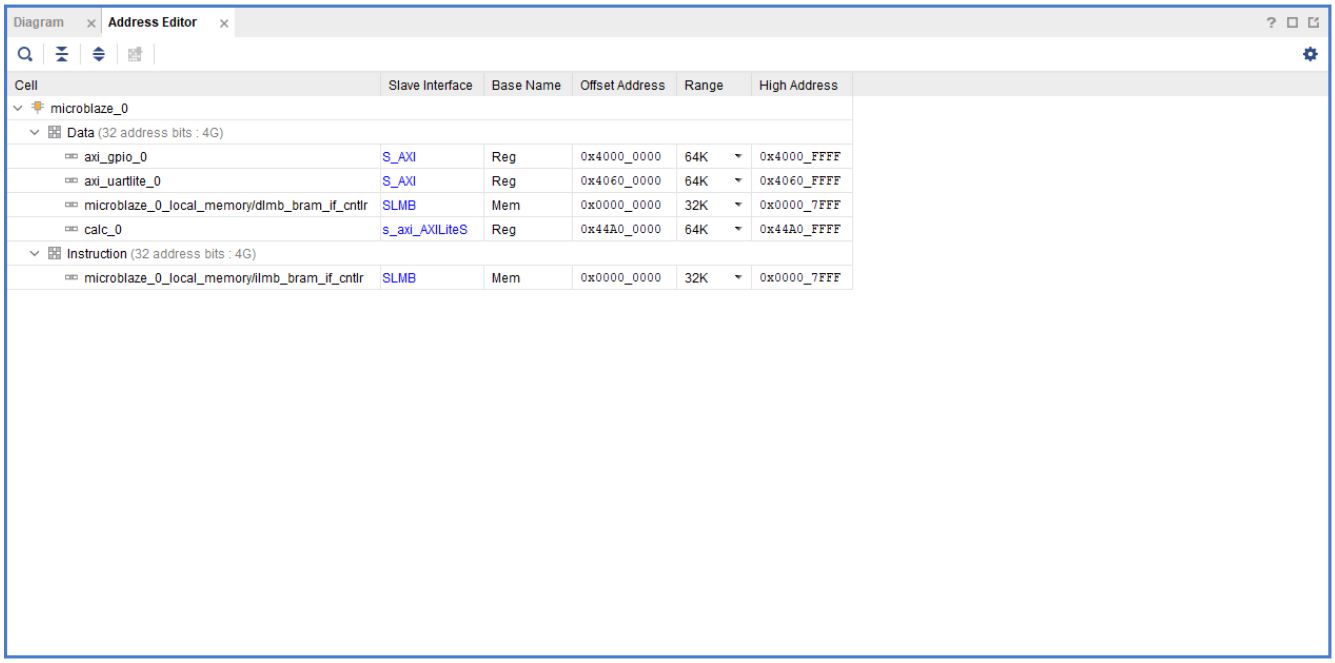
2025 г.

## Содержание

# Изображение SoC с аппаратным ускорителем



# Карта регистров аппаратного ускорителя




| Cell | Slave Interface | Base Name | Offset Address | Range | High Address |
|---|---|---|---|---|---|
| microblaze_0 | | | | | |
| Data (32 address bits : 4G) | | | | | |
| axi_gpio_0 | S_AXI | Reg | 0x4000_0000 | 64K | 0x4000_FFFF |
| axi_uartlite_0 | S_AXI | Reg | 0x4060_0000 | 64K | 0x4060_FFFF |
| microblaze_0_local_memory/dlmb_bram_if_cntlr | SLMB | Mem | 0x0000_0000 | 32K | 0x0000_7FFF |
| calc_0 | s_axi_AXILiteS | Reg | 0x44A0_0000 | 64K | 0x44A0_FFFF |
| Instruction (32 address bits : 4G) | | | | | |
| microblaze_0_local_memory/ilmb_bram_if_cntlr | SLMB | Mem | 0x0000_0000 | 32K | 0x0000_7FFF |

# Сравнение утилизации ресурсов для системы из 1й лабораторной работы и текущей

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 1311 | 63400 | 2.07 |
| LUTRAM | 160 | 19000 | 0.84 |
| FF | 1639 | 126800 | 1.29 |
| BRAM | 8 | 135 | 5.93 |
| IO | 20 | 210 | 9.52 |
| MMCM | 1 | 6 | 16.67 |

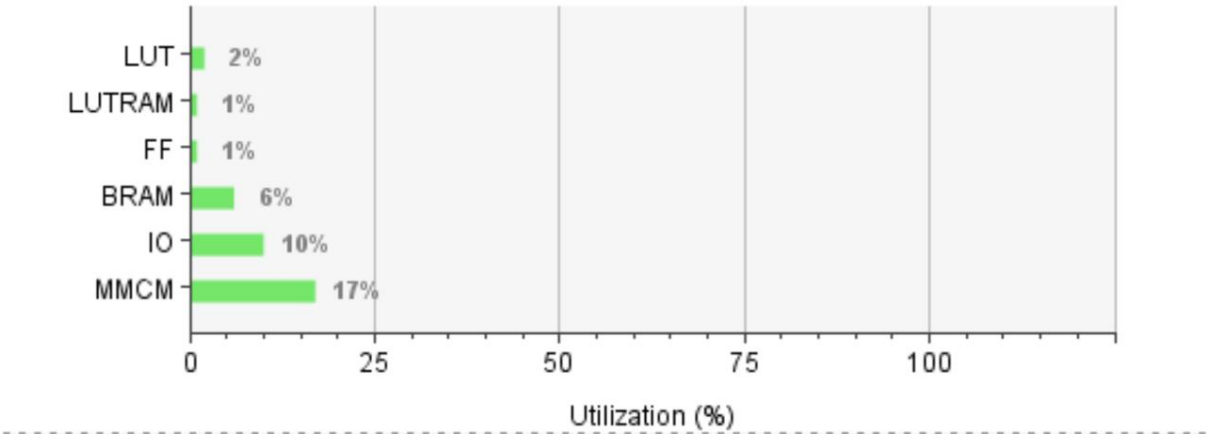



Рисунок 1 – утилизация ресурсов 1 лабораторной работы

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 5984 | 63400 | 9.44 |
| LUTRAM | 160 | 19000 | 0.84 |
| FF | 4652 | 126800 | 3.67 |
| BRAM | 12 | 135 | 8.89 |
| DSP | 184 | 240 | 76.67 |
| IO | 36 | 210 | 17.14 |
| MMCM | 1 | 6 | 16.67 |

Рисунок 2 – утилизация ресурсов 2 лабораторной работы

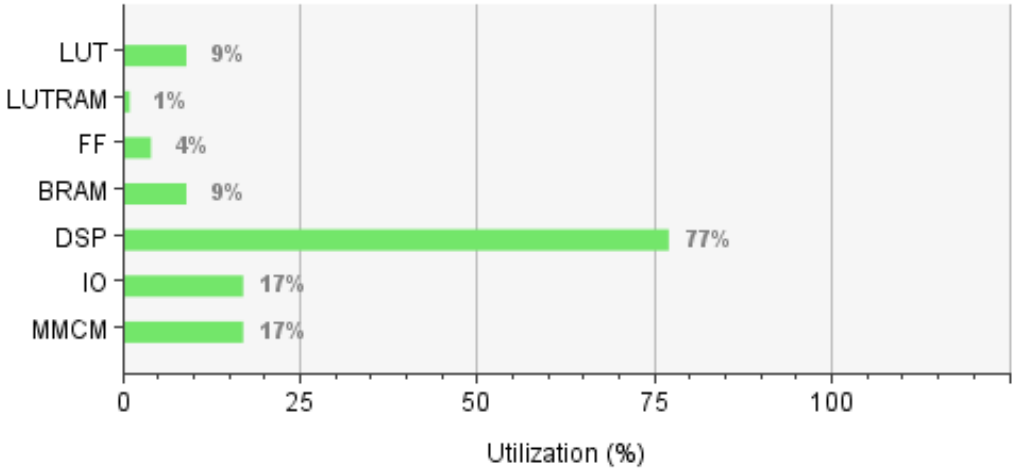



Рисунок 3 – утилизация ресурсов 2 лабораторной работы

**Время выполнения алгоритма для реализации только программного обеспечения и с использованием аппаратного ускорителя в виде таблицы**

| Наименование | Время выполнения мкс |
|---|---|
| IP (с аппаратным ускорителем) | $941.055 - 126.585 = 814.47$ |
| NO-IP (без аппаратного ускорителя) | $325.265 - 124.135 = 201.31$ |

# Набор временных диаграмм с результатами моделирования


Рисунок 2 – без аппаратного ускорителя


Рисунок 2 – с аппаратным ускорителем

# Исходный код программы для IP-ядра Microblaze

```
static void write_matrix_to_ip(int offset, int **M) {
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            Xil_Out32(CALC_BASE + offset + 4*(i*matrix_size + j),
M[i][j]);
}
static void read_matrix_from_ip(int offset, int **M) {
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            M[i][j] = Xil_In32(CALC_BASE + offset + 4*(i*matrix_size +
j));
}
int** calculate_function_hw(int **A, int **B)
{
    int **R = create_matrix();
    write_matrix_to_ip(A_OFFSET, A);
    write_matrix_to_ip(B_OFFSET, B);
    Xil_Out32(CALC_BASE + 0x00, 1);
    while ((Xil_In32(CALC_BASE + 0x00) & 0x2) == 0) {}
    read_matrix_from_ip(R_OFFSET, R);
    return R;
}
```

# Исходный код программного обеспечения

```
int main()
{
    init_platform();
    int **A = create_ones_matrix();
    int **B = create_sequence_matrix();
    int is_started = 1;
    uint32_t prev_sw = Xil_In32(GPIO_BASE_ADDR + GPIO_DATA2) & 0x1;
    while (1) {
        uint32_t cur_sw = Xil_In32(GPIO_BASE_ADDR + GPIO_DATA2) & 0x1;
        if ((cur_sw != prev_sw) || is_started) {
            if (cur_sw == 0) {
                int **R = calculate_function_sw(A, B);
                show_matrix_on_led(R, 1);
                free_matrix(R);
            } else {
                int **R = calculate_function_hw(A, B);
                show_matrix_on_led(R, 0);
                free_matrix(R);
            }
            prev_sw = cur_sw;
            is_started = 0;
        }
    }
    free_matrix(A);
    free_matrix(B);
    cleanup_platform();
    return 0;
}
```

```c
#include "matrix.h"
int** create_ones_matrix(void) {
    int** matrix = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        matrix[i] = (int*)malloc(matrix_size * sizeof(int));
        for (int j = 0; j < matrix_size; j++) {
            matrix[i][j] = 1;
        }
    }
    return matrix;
}
int** create_sequence_matrix(void) {
    int** matrix = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        matrix[i] = (int*)malloc(matrix_size * sizeof(int));
        for (int j = 0; j < matrix_size; j++) {
            matrix[i][j] = j + 1;
        }
    }
    return matrix;
}
```

```c
int** create_matrix(void) {
    int **matrix = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        matrix[i] = (int*)malloc(matrix_size * sizeof(int));
    }
    return matrix;
}
void free_matrix(int** matrix) {
    if (!matrix) return;
    for (int i = 0; i < matrix_size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}
```

```c
int** multiply_matrices(int **A, int **B) {
    int **result = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        result[i] = (int*)malloc(matrix_size * sizeof(int));
    }
    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            result[i][j] = 0;
            for (int k = 0; k < matrix_size; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return result;
}
```

```
int** subtract_matrices(int **A, int **B) {
    int **result = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        result[i] = (int*)malloc(matrix_size * sizeof(int));
    }
    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }
    return result;
}
int** calculate_function_sw(int **A, int **B)
{
    int** A_multiply_B = multiply_matrices(A, B);
    int** result       = subtract_matrices(A_multiply_B, B);
    free_matrix(A_multiply_B);
    return result;
}
```

```
static void write_matrix_to_ip(int offset, int **M) {
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            Xil_Out32(CALC_BASE + offset + 4*(i*matrix_size + j),
M[i][j]);
}
static void read_matrix_from_ip(int offset, int **M) {
    for (int i = 0; i < matrix_size; i++)
        for (int j = 0; j < matrix_size; j++)
            M[i][j] = Xil_In32(CALC_BASE + offset + 4*(i*matrix_size +
j));
}
```

```
int** calculate_function_hw(int **A, int **B)
{
    int **R = create_matrix();
    write_matrix_to_ip(A_OFFSET, A);
    write_matrix_to_ip(B_OFFSET, B);
    Xil_Out32(CALC_BASE + 0x00, 1);
    while ((Xil_In32(CALC_BASE + 0x00) & 0x2) == 0) {}
    read_matrix_from_ip(R_OFFSET, R);
    return R;
}
```

```
void show_matrix_on_led(int **M, int stop_on_sw_value)
{
    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            uint32_t sw = Xil_In32(GPIO_BASE_ADDR + GPIO_DATA2) & 0x1;
            if (sw == (uint32_t)stop_on_sw_value) {
                return;
            }
            uint16_t led_value = (uint16_t)(M[i][j] & 0xFFFF);
            Xil_Out16(GPIO_BASE_ADDR, led_value);
            sleep(1);
            Xil_Out16(GPIO_BASE_ADDR, 0x0000);
            sleep(1);
        }
    }
}
```

```
int get_value()
{
        int value = Xil_In16(GPIO_IN);
        while ((value & 0x4000) == 0)
                value = Xil_In16(GPIO_IN);
        value = value & 0x00FF;
        Xil_Out16(GPIO_OUT, 0x4000);
        Xil_Out16(GPIO_OUT, 0x0000);
        return value;
}
void send_value(int value)
{
        value = value | 0x2000;
        Xil_Out16(GPIO_OUT, value);
        Xil_Out16(GPIO_OUT, 0x0000);
}
```

# Исходный код верификационного окружения

```
int main() {
    init_platform();
    int **C;
    int **A = create_matrix();
    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            A[i][j] = get_value();
        }
    }
    int **B = create_matrix();
    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            B[i][j] = get_value();
        }
    }
    uint32_t cur_sw = (Xil_In32(GPIO_BASE_ADDR + GPIO_DATA2) >> 15) &
0x1;
    if (cur_sw == 0) {
      C = calculate_function_hw(A, B);
    } else {
      C = calculate_function_sw(A, B);
    }
      for (int i = 0; i < matrix_size; i++) {
            for (int j = 0; j < matrix_size; j++) {
                  send_value(C[i][j]);
            }
      }
      free_matrix(A);
      free_matrix(B);
      free_matrix(C);
    cleanup_platform();
    return 0;
}
```

```
module dev_tb;
reg clk;
wire [15:0] gpio_switch;
wire [14:0] gpio_switch_bus;
wire [15:0] gpio_led;
wire test_done;
reg rst_n;
reg mode_sw;
agent agent_impl (
    .clk_i(clk),
    .rst_i(!rst_n),
    .gpio_switch(gpio_switch_bus),
    .gpio_led(gpio_led),
    .test_done(test_done)
);
assign gpio_switch = {mode_sw, gpio_switch_bus};
design_1_wrapper DUT (
    .dip_switches_16bits_tri_i(gpio_switch),
    .sys_clock(clk),
    .gpio_rtl_tri_o(gpio_led),
    .reset(rst_n)
);
```

```
initial begin
    rst_n = 0;
    mode_sw= 1'b1;
    #200
    rst_n = 1;

    @(posedge test_done);
    $display("=== FIRST RUN (mode_sw = 1) DONE ===");

    #100
    rst_n = 0;
    mode_sw = 1'b0;
    #200
    rst_n = 1;

    @(posedge test_done);
    $display("=== SECOND RUN (mode_sw = 0) DONE ===");

    #100
    $finish;

end
```

```
initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

endmodule
module agent#(
    parameter int SIZE = 64
)(
    input clk_i,
    input rst_i,
    output [14:0] gpio_switch,
    output [14:0] gpio_led,
    output logic  test_done
    );

    logic [14:0] expected_matrix[SIZE];
    logic [14:0] tmp_sequence[SIZE];
    logic [14:0] result_sequence[SIZE];
    logic sequence_valid, sequence_send, result_valid;

    sequencer sequencer_impl (
        .clk_i(clk_i),
        .rst_i(rst_i),
        .sequence_o(tmp_sequence),
        .sequence_valid_o(sequence_valid),
        .sequence_send_i(sequence_send)
    );

    driver driver_impl (
        .clk(clk_i),
        .rst(rst_i),

        .sequence_i(tmp_sequence),
        .sequence_valid(sequence_valid),
        .sequence_send(sequence_send),

        .gpio_switch(gpio_switch),
        .gpio_led(gpio_led)
    );

    monitor monitor_impl (
        .clk(clk_i),
        .rst(rst_i),
        .gpio_led(gpio_led),

        .input_matrix_i(tmp_sequence),
        .input_valid_i(sequence_valid),

        .expected_matrix_i(expected_matrix),

        .result_matrix_o(result_sequence),
        .result_valid(result_valid)
    );
```

```
 scoreboard scoreboard_impl (
        .clk_i(clk_i),
        .rst_i(rst_i),

        .matrix_input(tmp_sequence),
        .matrix_vld(sequence_valid),

        .result_matrix(result_sequence),
        .result_vld(result_valid),
        .expected_matrix(expected_matrix),
        .test_done(test_done)
    );

endmodule
module sequencer#(
    parameter int SIZE = 64
)(
    input  logic clk_i,
    input  logic rst_i,
    output logic [14:0] sequence_o[SIZE],
    output logic sequence_valid_o,
    input  logic sequence_send_i
);
    localparam int SEND_A = 0;
    localparam int WAIT   = 1;
    localparam int SEND_B = 2;
    logic [1:0] state;
    function automatic logic [14:0] pick_weighted();
        int sel;
        int rand_val;
        sel = $urandom_range(1, 5);
        if (sel <= 3) begin
            rand_val = $urandom_range(10, 70);
        end else begin
            rand_val = $urandom_range(80, 170);
        end
        return rand_val[14:0];
    endfunction
```

```
always_ff @(posedge clk_i) begin
        if (rst_i) begin
            state <= SEND_A;
            for (int i = 0; i < SIZE; i++) begin
                sequence_o[i] <= '0;
            end
            sequence_valid_o <= 1'b0;
        end
        else begin
            case (state)
                SEND_A: begin
                    for (int i = 0; i < SIZE; i++) begin
                        sequence_o[i] <= pick_weighted();
                    end
                    sequence_valid_o <= 1'b1;
                    state <= WAIT;
                end
                WAIT: begin
                    if (sequence_send_i) begin
                        state <= SEND_B;
                    end
                    sequence_valid_o <= 1'b0;
                end
                SEND_B: begin
                    for (int i = 0; i < SIZE; i++) begin
                        sequence_o[i] <= pick_weighted();
                    end
                    sequence_valid_o <= 1'b1;
                    state <= WAIT;
                end
                default: begin
                    state <= SEND_A;
                end
            endcase
        end
    end
endmodule
```

```
module driver#(
    parameter int SIZE = 64
)(
    input clk,
    input rst,

    input [14:0] sequence_i[SIZE],
    input sequence_valid,
    output logic sequence_send,

    output logic [14:0] gpio_switch,
    input [14:0] gpio_led
    );

    localparam WAIT_SEQ = 0;
    localparam SEND_SEQ = 1;
```

```
    logic state;
    logic recieve;

    logic [14:0] temp_matrix [SIZE];
    logic [6:0] temp_idx;

    int i;

    wire send_vld = gpio_switch[14];
    wire recieve_vld = gpio_led[14];

    always_ff @(posedge clk) begin
        if (rst) begin
            for (int i = 0; i < SIZE; i++) begin
                temp_matrix[i] <= '0;
            end
            temp_idx <= '0;
            state <= WAIT_SEQ;
            sequence_send <= '0;
            recieve <= '0;
        end else begin
            case (state)
                WAIT_SEQ: begin
                    if (sequence_valid) begin
                        temp_matrix <= sequence_i;
                        state <= SEND_SEQ;
                    end
                    recieve <= '0;
                    temp_idx <= '0;
                    sequence_send <= '0;
                end
                SEND_SEQ: begin
                    if (temp_idx == SIZE) begin
                        state <= WAIT_SEQ;
                        sequence_send <= '1;
                    end else begin
                        if (send_vld) begin
                            if (recieve && !recieve_vld) begin
                                temp_idx <= temp_idx + 1'b1;
                                gpio_switch <= '0;
                                recieve <= '0;
                            end else if (recieve_vld) begin
                                recieve <= 1;
                            end
                        end else begin
                                gpio_switch <= temp_matrix[temp_idx] |
15'h4000;
                        end
                    end
                end
            endcase
        end
    end
endmodule
```

```
module monitor#(
    parameter int SIZE = 64
)(
    input clk,
    input rst,
    input [14:0] gpio_led,
    input [14:0] input_matrix_i[SIZE],
    input        input_valid_i,
    input logic [14:0] expected_matrix_i[SIZE],
    output logic [14:0] result_matrix_o[SIZE],
    output logic result_valid
    );
    integer file;
    time start_time, end_time;
    logic [14:0] saved_matrix_a[SIZE];
    logic [14:0] saved_matrix_b[SIZE];
    logic [6:0] temp_idx;
    logic [1:0] state;
    logic [1:0] phase;
    localparam RECIEVE = 0;
    localparam DELAY   = 1;

    int i;
    initial begin
        file = $fopen("monitor_log.txt", "w");
    end
    always_ff @(posedge clk) begin
        if (rst) begin
            temp_idx <= 0;
            state <= RECIEVE;
            phase <= 0;
            start_time <= $time;
        end
```

```
else begin
            if (input_valid_i) begin
                if (phase == 0) begin
                    saved_matrix_a <= input_matrix_i;
                    $fwrite(file, "\n=== Matrix A captured at %0t ns
===\n", $time);
                    for (int i = 0; i < SIZE; i++)
                        $fwrite(file, "A[%0d] = %h\n", i,
input_matrix_i[i]);
                    phase <= 1;
                end else if (phase == 1) begin
                    saved_matrix_b <= input_matrix_i;
                    $fwrite(file, "\n=== Matrix B captured at %0t ns
===\n", $time);
                    for (int i = 0; i < SIZE; i++)
                        $fwrite(file, "B[%0d] = %h\n", i,
input_matrix_i[i]);
```

```
phase <= 2;
                end
            end
            if (state == RECIEVE) begin
                if (gpio_led[13]) begin
                    result_matrix_o[temp_idx] <= gpio_led[12:0];
                    $fwrite(file, "Time %0t ns | Expected[%0d] = %h |
Result[%0d]=%h\n",
                            $time, temp_idx, expected_matrix_i[temp_idx],
temp_idx, gpio_led[12:0]);
                    temp_idx <= temp_idx + 1'b1;
                    state <= DELAY;
                end
            end else begin
                if (!gpio_led[13]) begin
                    state <= RECIEVE;
                end
            end
            if (temp_idx == SIZE) begin
                result_valid <= 1;
                end_time = $time;
            end else begin
                result_valid <= 0;
            end
        end
    end

    final begin
        $fwrite(file, "\n=== Algorithm finished ===\n");
        $fwrite(file, "Execution time: %0t ns\n", end_time - start_time);
        $fclose(file);
    end

endmodule
```

```
module scoreboard#(
    parameter int SIZE = 64,
    parameter int ROW = 8
)(
    input  logic        clk_i,
    input  logic        rst_i,
    input  logic [14:0] matrix_input [SIZE],
    input  logic        matrix_vld,
    input  logic [14:0] result_matrix [SIZE],
    input  logic        result_vld,
    output logic [14:0] expected_matrix[SIZE],
    output logic        test_done
);
```

```
  typedef enum logic [1:0] {
        WAIT_A      = 2'd0,
        WAIT_B      = 2'd1,
        WAIT_RESULT = 2'd2,
        CHECK       = 2'd3
    } state_t;
    state_t state;
    logic [14:0] matrix_a [SIZE];
    logic [14:0] matrix_b [SIZE];
    logic [14:0] product_matrix [SIZE];
    int i, j, k;
    int errors;
    int sum;

    always_ff @(posedge clk_i) begin
        if (rst_i) begin
            state <= WAIT_A;
            errors <= 0;
            test_done <= 1'b0;
            for (i = 0; i < SIZE; i++) begin
                matrix_a[i]       <= '0;
                matrix_b[i]       <= '0;
                product_matrix[i] <= '0;
            end
        end
        else begin
            case (state)
                WAIT_A: begin
                    if (matrix_vld) begin
                        for (i = 0; i < SIZE; i++)
                            matrix_a[i] <= matrix_input[i];
                        state <= WAIT_B;
                    end
                end
                WAIT_B: begin
                    if (matrix_vld) begin
                        for (i = 0; i < SIZE; i++)
                            matrix_b[i] <= matrix_input[i];
                        state <= WAIT_RESULT;
                    end
                end
                WAIT_RESULT: begin
                    for (i = 0; i < ROW; i++) begin
                        for (j = 0; j < ROW; j++) begin
                            sum = 0;
                            for (k = 0; k < ROW; k++) begin
                                sum += matrix_a[i*ROW + k] *
matrix_b[k*ROW + j];
                            end
```

```
product_matrix[i*ROW + j] = (sum - matrix_b[i*ROW + j]) & 15'h1FFF;
                        end
                    end
                    for (i = 0; i < SIZE; i++) begin
                        expected_matrix[i] = product_matrix[i];
                    end
                    if (result_vld)
                        state <= CHECK;
                end
                CHECK: begin
                    errors = 0;
                    for (i = 0; i < SIZE; i++) begin
                        if (product_matrix[i] !== result_matrix[i]) begin
                            $display("Mismatch [%0d]: expected=%0h,
got=%0h",
                                     i, product_matrix[i],
result_matrix[i]);
                            errors++;
                        end else begin
                            $display("Match [%0d]: %0h", i,
result_matrix[i]);
                        end
                    end
                    if (errors == 0)
                        $display("RESULT CORRECT: (A*B - B) mod 16
bits");
                    else
                        $display("Total mismatches: %0d", errors);
                    test_done <= 1'b1;
//                    $finish;
                end
            endcase
        end
    end
endmodule
```

# Вывод

В данной лабораторной работе мы получили знания и навыки использования аппаратных и программных реализаций вычислительных блоков в одной системе на кристалле.

В итоге был интегрирован аппаратный ускоритель, разработанный в Vivado HLS, в SoC на базе MicroBlaze с использованием интерфейса AXI4-Lite. Для системы выбрана оптимальная конвейерная версия ускорителя, обеспечивающая лучшую производительность при меньших затратах ресурсов.

Было разработано программное обеспечение, включающее программную реализацию вычислений и взаимодействие с аппаратным ускорителем. Создан тестовый стенд, который сравнил оба варианта вычислений и подтвердил преимущество аппаратной реализации по скорости.

Проект был синтезирован, загружен на FPGA, и реализована возможность переключения режима вычислений в реальном времени.