EHB436E Digital System Design Applications

Final Project Report

Mehmet Yasir Bağcı

040200037

Salih Ömer Ongün

040220780

Mehmet Yasir Bağcı
040200037

Salih Ömer Ongün
040220780

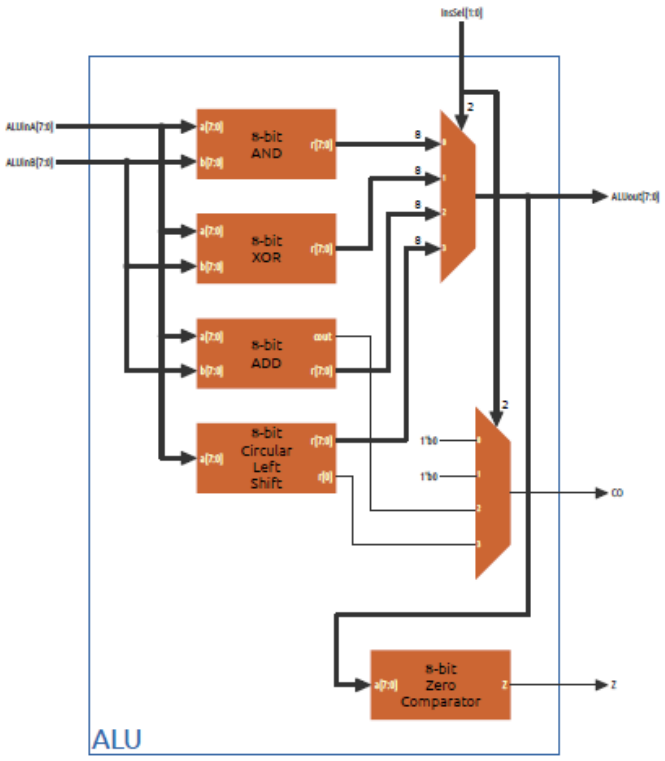
1. Problem Statement

The problem that was assigned to us is calculating the A^B by using the blocks that are given to us. The blocks that we utilized are:

- ALU
- Register Block
- Control Unit
- Top module

We implement our algorithm by inserting the relative code to control unit block.

2. ALU



Şekil 1 ALU Block Diagram

```
module ALU

(
    input [1:0] insel,
    output [7:0] alu_out,
    input [7:0] alu_in_a,
    input [7:0] alu_in_b,
    output co,
    output z
);

    wire cout;
    wire [7:0]
add_sum,shift_out,and_out,xor_out;
    wire shift_out_0;

    // ALU_OUT_MUX modülünün örneklemesi
    alu_out_mux alu_mux_inst (
        .D0(and_out),
        .D1(xor_out),
        .D2(add_sum),
        .D3(shift_out),
        .insel(insel),
        .O(alu_out)
    );

    // CO_MUX modülünün örneklemesi
    co_mux co_mux_inst (
        .D0(1'b0),
        .D1(1'b0),
        .D2(cout),
        .D3(shift_out_0),
        .insel(insel),
        .O(co)
    );

    AND and_8
    (
        .a(alu_in_a),
        .b(alu_in_b),
        .r(and_out)
    );

    XOR xor_8
    (
        .a(alu_in_a),
        .b(alu_in_b),
        .r(xor_out)
    );
```

```
ADD add_8

    (
        .x(alu_in_a),
        .y(alu_in_b),
        .cin(1'b0),
        .cout(cout),
        .sum(add_sum)
    );

    circular_shift shift8
    (
        .a(alu_in_a),
        .r(shift_out),
        .r_0(shift_out_0)
    );

    zero_comp comparator8
    (
        .a(alu_out),
        .z(z)
    );


endmodule



module alu_out_mux (
    input [7:0] D0,
    input [7:0] D1,
    input [7:0] D2,
    input [7:0] D3,
    input [1:0] insel, // 2-bit seçim girişi
    output reg [7:0] O      // 4-bit çıkış
);
always @(*)

    begin

        case (insel)

            2'b00: O = D0;

            2'b01: O = D1;

            2'b10: O = D2;

            2'b11: O = D3;

            default: O = 8'b0; // Güvenlik için bir default durumu ekledik

        endcase

    end

endmodule
```

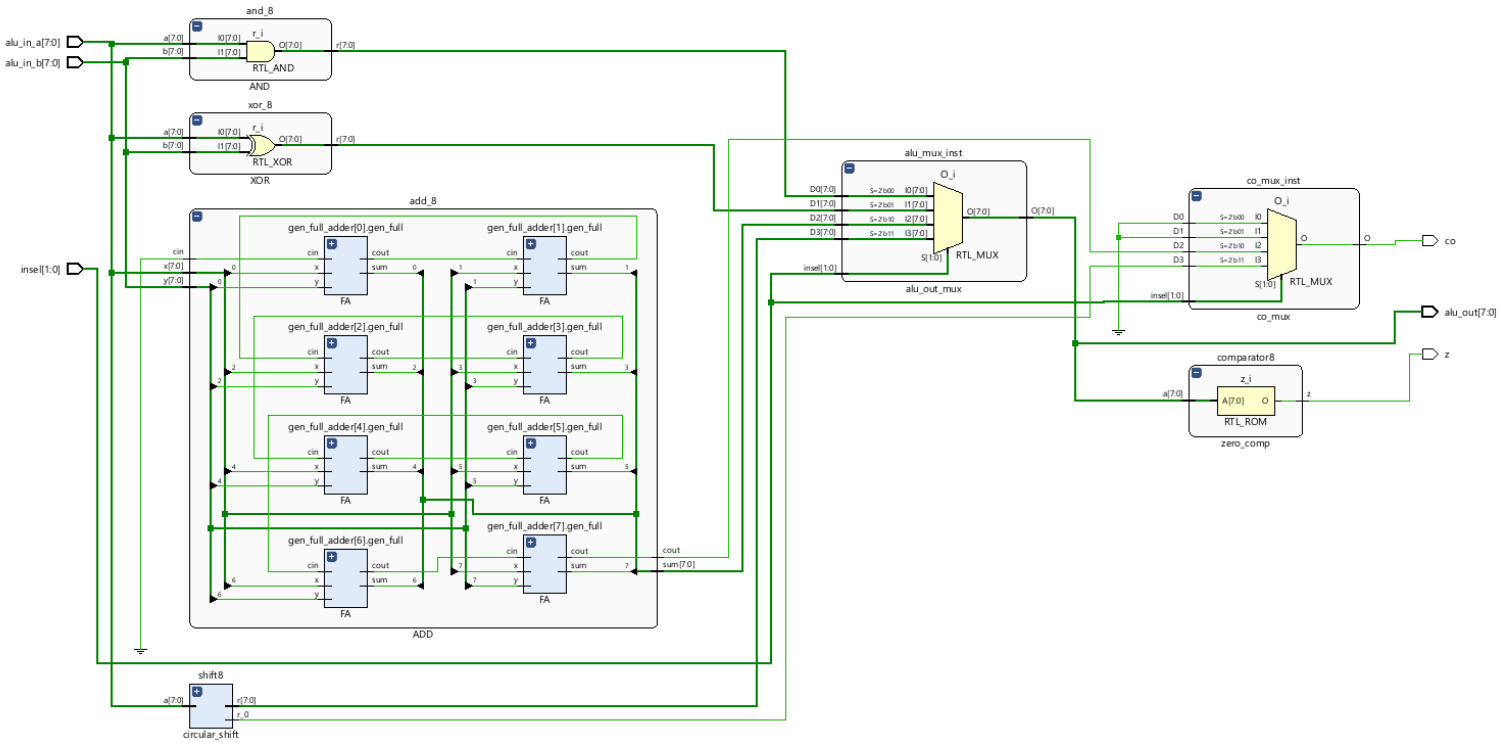
```
module co_mux (

    input D0,
    input D1,
    input D2,
    input D3,
    input [1:0] insel, // 2-bit seçim girişi
    output reg O       // 4-bit çıkış
);

    always @(*)
    begin
        case (insel)
            2'b00: O = D0;
            2'b01: O = D1;
            2'b10: O = D2;
            2'b11: O = D3;
            default: O = 0; // Güvenlik için bir default durumu ekledik
        endcase
    end

endmodule


module circular_shift
(
    input [7:0] a ,
    output [7:0] r,
    output r_0
);

    assign r = {a[6:0],a[7]};
    assign r_0 = r[0];

endmodule


module XOR(
    input [7:0] a,
    input [7:0] b,
    output [7:0] r);

    assign r = a ^ b;
endmodule

module AND(
    input [7:0] a,
    input [7:0] b,
    output [7:0] r
    );
    assign r = a & b;
endmodule
```

```
module ADD

(
    input [7:0] x,
    input [7:0] y,
    input cin,
    output cout,
    output [7:0] sum
);
    wire [8:0] cout_gen;

    assign cout_gen[0] = cin;

    genvar i;
    generate
        for(i = 0; i<8; i = i+1) begin :
gen_full_adder
            FA gen_full
            (
                x[i],
                y[i],
                cout_gen[i],
                cout_gen[i+1],
                sum[i]
            );
        end
    endgenerate
    assign cout = cout_gen[8];

endmodule


module HA
(
    input x,
    input y,
    output cout,
    output sum
);
    reg sum_reg,cout_reg;

    always@(x,y) begin
        {cout_reg,sum_reg} = x + y;
    end

    assign sum = sum_reg;
    assign cout = cout_reg;

endmodule
```

```
module FA

(
    input x,
    input y,
    input cin,
    output cout,
    output sum
);
    wire ha1_sum, ha1_cout, ha2_cout;

    HA half1
    (
      .x(x),
      .y(y),
      .sum(ha1_sum),
      .cout(ha1_cout)
    );

    HA half2
    (
        .x(ha1_sum),
        .y(cin),
        .sum(sum),
        .cout(ha2_cout)
    );

    OR or1
    (
    .l1(ha1_cout),
    .l2(ha2_cout),
    .O(cout)
    );

endmodule



module OR
(
    input l1,
    input l2,
    output O
);

    assign O = l1 | l2;

endmodule
```

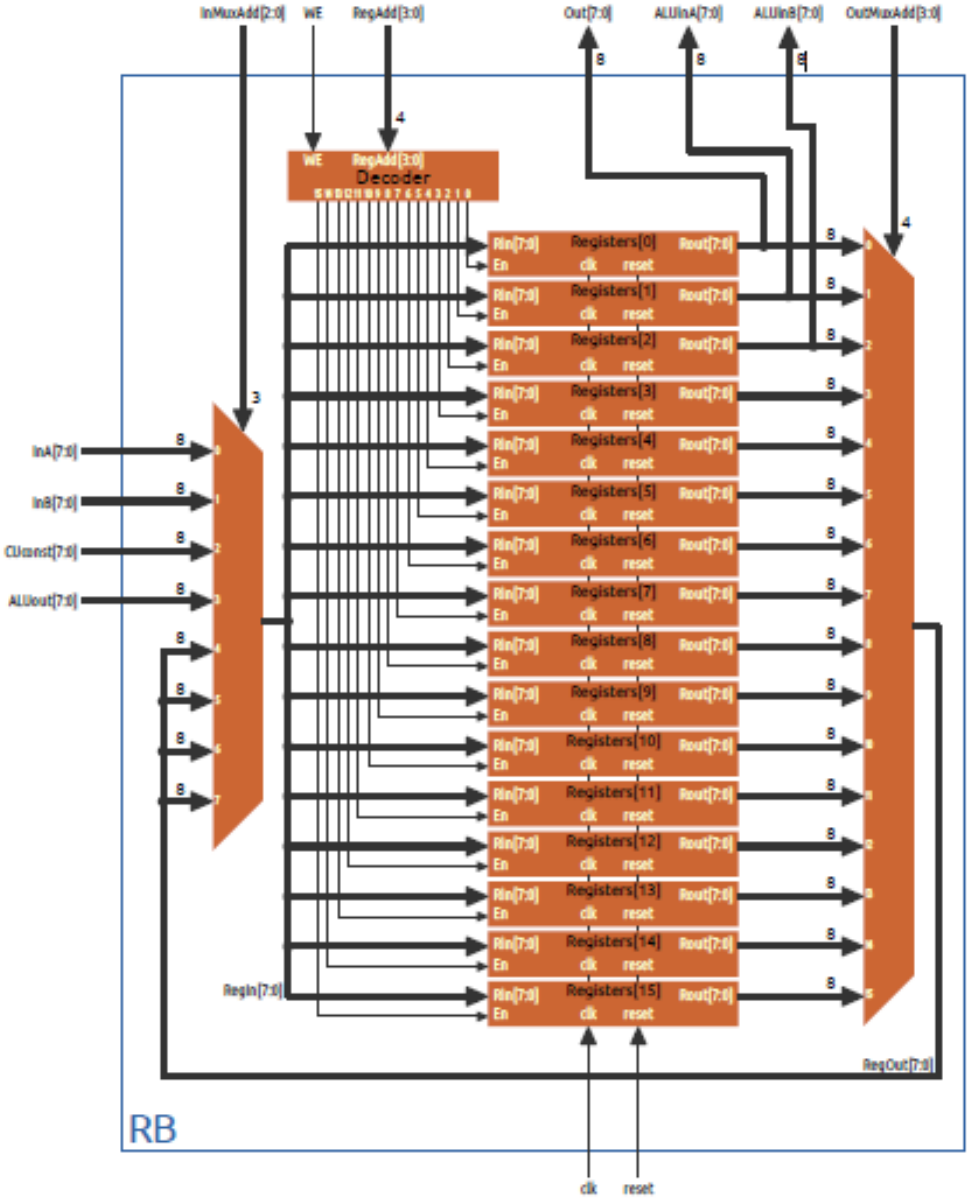
```
module zero_comp

(
    input [7:0] a,
    output reg z
);
    always @(*) begin
        if (a == 8'b0) begin
            z <= 1;
        end
        else begin
            z <= 0;
        end
    end
endmodule
```

Şekil 2 Verilog Code of ALU Block and Inner modules



Şekil 3 RTL Schematic

3. Register Block



Şekil 4 RB Block Diagram

```
module REGISTER_BLOCK

(
    input clk,
    input rst,
    input [7:0] inA,
    input [7:0] inB,
    input [7:0] cu_const,
    input [2:0] in_mux_add,
    input [3:0] out_mux_add,
    input [3:0] reg_add,
    input we,
    input [7:0] alu_out,
    output [7:0] alu_in_a,
    output [7:0] alu_in_b,
    output [7:0] out
);

    wire [15:0] decoder_out;
    wire [7:0] out_mux_out;
    wire [7:0] in_mux_out;
    wire [7:0] re_out0, re_out1, re_out2, re_out3, re_out4, re_out5,re_out6;

    // DECODER instantiation
    DECODER decoder
    (
        .reg_add(reg_add),
        .we(we),
        .OUT(decoder_out)
    );


      // IN_MUX instantiation

    IN_MUX u_in_mux
    (
        .D0(inA),
        .D1(inB),
        .D2(cu_const),
        .D3(alu_out),
        .D4(out_mux_out),
        .D5(out_mux_out),
        .D6(out_mux_out),
        .D7(out_mux_out),
        .in_mux_add(in_mux_add),
        .O(in_mux_out)
    );
```

```
// OUT_MUX instantiation

    OUT_MUX u_out_mux
    (
        .D0(re_out0),
        .D1(re_out1),
        .D2(re_out2),
        .D3(re_out3),
        .D4(re_out4),
        .D5(re_out5),
        .D6(re_out6),
        .D7(8'b0),
        .D8(8'b0),
        .D9(8'b0),
        .D10(8'b0),
        .D11(8'b0),
        .D12(8'b0),
        .D13(8'b0),
        .D14(8'b0),
        .D15(8'b0),
        .out_mux_add(out_mux_add),
        .O(out_mux_out)
    );

    // REGISTER instantiations
    REGISTER reg0
    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[0]),
        .rout(re_out0)
    );


      REGISTER reg1

    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[1]),
        .rout(re_out1)
    );

    REGISTER reg2
    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[2]),
        .rout(re_out2)
    );
```

```
REGISTER reg3
    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[3]),
        .rout(re_out3)
    );
    REGISTER reg4
    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[4]),
        .rout(re_out4)
    );
    REGISTER reg5
    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[5]),
        .rout(re_out5)
    );
```

```
REGISTER reg6

    (
        .clk(clk),
        .rst(rst),
        .rin(in_mux_out),
        .en(decoder_out[6]),
        .rout(re_out6)
    );

    // Output assignment
    assign alu_in_a = re_out1;
    assign alu_in_b = re_out2;
    assign out = re_out0;

endmodule


module DECODER
(
    input [3:0] reg_add,
    input we,                // write enable sinyali
    output [15:0] OUT
);


reg [15:0] out_reg;



    always @(*)

    begin

        if (we) begin          // we sinyali 1 ise çalışır

            case(reg_add)

                4'b0000: out_reg = 16'b0000_0000_0000_0001;

                4'b0001: out_reg = 16'b0000_0000_0000_0010;

                4'b0010: out_reg = 16'b0000_0000_0000_0100;

                4'b0011: out_reg = 16'b0000_0000_0000_1000;

                4'b0100: out_reg = 16'b0000_0000_0001_0000;

                4'b0101: out_reg = 16'b0000_0000_0010_0000;

                4'b0110: out_reg = 16'b0000_0000_0100_0000;
```

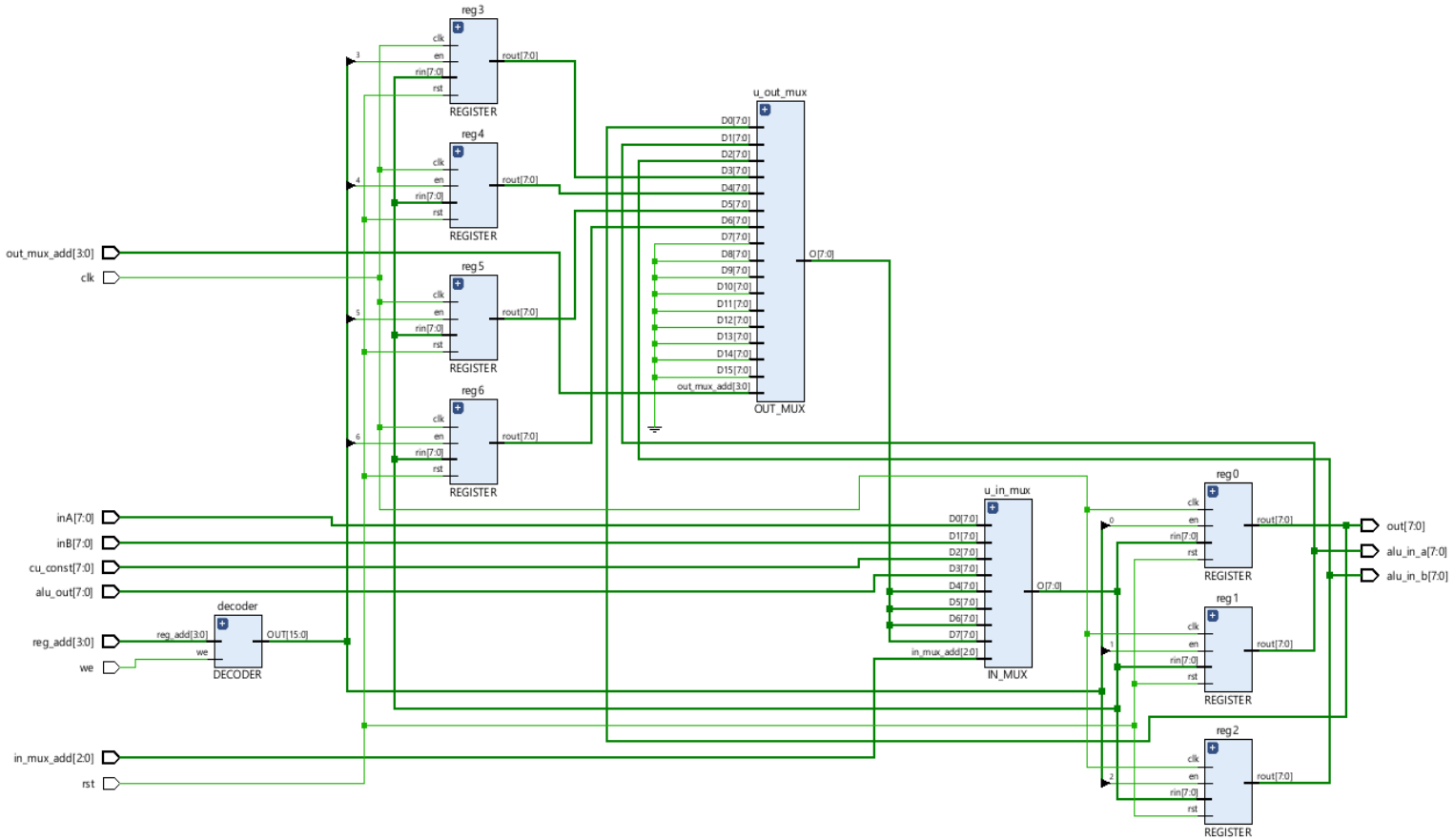
```
                4'b0111: out_reg = 16'b0000_0000_1000_0000;

                 4'b1000: out_reg = 16'b0000_0001_0000_0000;
                 4'b1001: out_reg = 16'b0000_0010_0000_0000;
                 4'b1010: out_reg = 16'b0000_0100_0000_0000;
                 4'b1011: out_reg = 16'b0000_1000_0000_0000;
                 4'b1100: out_reg = 16'b0001_0000_0000_0000;
                 4'b1101: out_reg = 16'b0010_0000_0000_0000;
                 4'b1110: out_reg = 16'b0100_0000_0000_0000;
                 4'b1111: out_reg = 16'b1000_0000_0000_0000;
                default: out_reg = 16'b0000_0000_0000_0000;
            endcase
        end else begin          // we sinyali 0 ise çıkış sıfırlanır
            out_reg = 16'b0000_0000_0000_0000;
        end
    end
assign OUT = out_reg;


endmodule




module IN_MUX
(
    input [7:0] D0,
    input [7:0] D1,
    input [7:0] D2,
    input [7:0] D3,
    input [7:0] D4,
    input [7:0] D5,
    input [7:0] D6,
    input [7:0] D7,
    input [2:0] in_mux_add, // 3-bit seçim girişi
    output reg [7:0] O      // 8-bit çıkış
);
```

```
always @(*)

    begin

        case (in_mux_add)

            3'b000: O = D0;

            3'b001: O = D1;

            3'b010: O = D2;

            3'b011: O = D3;

            3'b100: O = D4;

            3'b101: O = D5;

            3'b110: O = D6;

            3'b111: O = D7;

            default: O = 8'b0; // için bir default durumu ekledik

        endcase Güvenlik

    end

endmodule

module OUT_MUX
(
    input [7:0] D0,
    input [7:0] D1,
    input [7:0] D2,
    input [7:0] D3,
    input [7:0] D4,
    input [7:0] D5,
    input [7:0] D6,
    input [7:0] D7,
    input [7:0] D8,
    input [7:0] D9,
    input [7:0] D10,
    input [7:0] D11,
    input [7:0] D12,
    input [7:0] D13,
    input [7:0] D14,
    input [7:0] D15,
    input [3:0] out_mux_add, // 4-bit seçim girişi
    output reg [7:0] O       // 8-bit çıkış
);
```

```
always @(*)

    begin

        case (out_mux_add)

            4'b0000: O = D0;

            4'b0001: O = D1;

            4'b0010: O = D2;

            4'b0011: O = D3;

            4'b0100: O = D4;

            4'b0101: O = D5;

            4'b0110: O = D6;

            4'b0111: O = D7;

            4'b1000: O = D8;

            4'b1001: O = D9;

            4'b1010: O = D10;

            4'b1011: O = D11;

            4'b1100: O = D12;

            4'b1101: O = D13;

            4'b1110: O = D14;

            4'b1111: O = D15;

            default: O = 8'b0; // Default durumu, güvenlik için

        endcase

    end


endmodule

module REGISTER
(
    input clk,
    input rst,
    input [7:0] rin,
    input en,
    output reg [7:0] rout
);
```

```
        if (rst == 1'b1) begin
                rout <= 8'b0;
        end

        else begin
            if (en == 1'b1) begin
                rout <= rin;
            end
        end
    end

endmodule
```

Şekil 5 Verilog Code of Register Block



Şekil 6 Register Block RTL Schematic

4. Control Unit

EXPONENTIAL CALCULATING

Register4 holds the sum value of the result of each operation. When decreasing the upper, we assign the total value (Register4) to Base (Register6).

EXAMPLE

$$4^3 = 64$$

Firstly

$$REGISTER4 = 4$$

FIRST LOOP

$$REGISTER3 = 3 - 1 = 2$$

$$BASE\ (REGISTER6) = 4$$

4 + 4 = 8

$$REGISTER4 = 8$$

8 + 4 = 12

$$REGISTER4 = 12$$

12+ 4 = 16

$$REGISTER4 = 16$$

$$REGISTER4 = 16$$

$$BASE(REGISTER6) = REGISTER4\ = 16$$

SECOND LOOP

$$REGISTER3 = 2 - 1 = 1$$

$$BASE\ (REGISTER6) = 16$$

16 + 16 = 32

$$REGISTER4 = 32$$

32 + 16 = 48

$$REGISTER4 = 48$$

48 + 16 = 64

$$REGISTER4 = 64$$

$$REGISTER4 = 64$$

$$BASE(REGISTER6) = REGISTER4 = 64$$

THIRD LOOP

$$REGISTER3 = 1 - 1 = 0$$

$$z = 1$$

Go to DONE state

OUT <= $REGISTER4\ (64)$

State goes to DONE and Register4 assigned to OUT.

```
module control_unit
(
    input clk,
    input rst,
    input start,
    input co,
    input z,
    output reg busy,
    output reg [1:0] insel,
    output [7:0] cu_const,
    output reg [2:0] in_mux_add,
    output reg [3:0] out_mux_add,
    output reg [3:0] reg_add,
    output reg we
);
    localparam IDLE        = 5'b00000;
    localparam START       = 5'b00001;
    localparam ZERO_A      = 5'b00010;
    localparam ZERO_B      = 5'b00011;
    localparam OUT1        = 5'b00100;
    localparam OUT1_1      = 5'b00101;
    localparam OUT1_2      = 5'b00110;
    localparam EXP_SUB     = 5'b00111;
    localparam SUB_REC     = 5'b01000;
    localparam SEL_BASE    = 5'b01001;
    localparam EXP_TO_BASE = 5'b01010;
    localparam MULT_SUB    = 5'b01011;
    localparam ADD_REC     = 5'b01100;
    localparam MULT_ADD    = 5'b01101;
    localparam SEL_SUM_A   = 5'b01110;
    localparam ADD_RES     = 5'b01111;
    localparam SEL_A       = 5'b10000;
    localparam DONE        = 5'b10001;
    localparam ARA1        = 5'b10010;
    localparam ARA2        = 5'b10011;
    reg [4:0] state;

    assign cu_const = 8'b11111111;

    always @(posedge clk or posedge rst) begin

        if (rst == 1'b1) begin
            state <= IDLE;
            we <= 0;

        end

        else begin
            case(state)
                IDLE: begin
                    if(start == 1'b1) begin
                        state <= START;
                        we <= 1'b1;
                              busy <= 1;
                              in_mux_add <= 3'd0;
                        reg_add <= 4'd4; // A degerini registira atadık hem B = 1 hem de sonraki işlemde ekleme
yakmak icin
                    end
                end
```

```
START: begin
    in_mux_add <= 3'd0; //  A secildi
    reg_add <= 4'd1; // A inA ya verildi
    state <= ARA1;
end

ARA1: begin
    insel <= 2'd3; // shift yapildi
    state <= ZERO_A;

end

ZERO_A: begin // A sıfır mı
    if( z == 1) begin
        state <= DONE;
    end
    else begin
        state <= ARA2;
        in_mux_add <= 3'd1; // B seciyorum
        reg_add <= 4'd1; // B yi inA veriyorum

    end
end

ARA2: begin
    insel <= 2'd3; // shift yapildi
    state <= ZERO_B;
end

ZERO_B: begin
    if( z == 1) begin
        state <= OUT1;
        in_mux_add <= 3'd2; // cuconst secildi
        reg_add <= 4'd1; // inA ya verildi
    end
    else begin
        state <= EXP_SUB;
        in_mux_add <= 3'd1; // B seciyorum
        reg_add <= 4'd1; // B yi inA veriyorum
    end
end
OUT1: begin
    state <= OUT1_1;
    in_mux_add <= 3'd2; // cuconst secildi
    reg_add <= 4'd2; // inB ye verildi
end

OUT1_1: begin
    state <= OUT1_2;
    insel <= 2'd2; // 255 + 255 yapilip 510 yani 1_1111_1110 bulundu
    in_mux_add <= 3'd3; // ALUout (1_1111_1110) secildi
    reg_add <= 4'd1; // inA ye verildi
end


OUT1_2: begin
    state <= DONE;
    insel <= 2'd1; // 255 ile 510 exor oldu 0000_0001 yani decimal 1 bulundu
    in_mux_add <= 3'd3; // ALUout (0000_0001) secildi
    reg_add <= 4'd4; // DONE da outa verilen register4 e verildi
end

EXP_SUB : begin
    in_mux_add <= 3'd2; // constant sec
    reg_add <= 4'd2; // constant inB ye gonder
    insel <= 2'd2; // B dan 1 cikar
    state <= SUB_REC;
end
```

```
SUB_REC : begin
    in_mux_add <= 3'd3; // Eksilen B yi (alu out) SEC
    reg_add <= 4'd3;    // Register a at

    if ( z == 1) begin // ustte ifade kalmadı B = 0 oldu
        state <= DONE;
        //out_mux_add <= 4'd4; // toplam degeri feedback olarak verildi
    end
    else begin
        state <= SEL_BASE;
        out_mux_add <= 4'd4; // toplam degeri feedback olarak verildi (base olarak kullanilacak)
    end

end

SEL_BASE: begin
    out_mux_add <= 4'd4; // toplam degeri feedback olarak verildi (base olarak kullanilacak)
    in_mux_add <= 3'd4; //  base'i sec
    reg_add <= 4'd6; // base i registira atadik
    state <= EXP_TO_BASE;
end

EXP_TO_BASE : begin
    in_mux_add <= 3'd0; // A sec
    reg_add <= 4'd1; // A yi inA ya gonder
    state <= MULT_SUB;
end

MULT_SUB: begin
    in_mux_add <= 3'd2; // constant sec
    reg_add <= 4'd2; // constan i inB ya gonder
    insel <= 2'd2; // A dan 1 cikar
    state <= ADD_REC;

end

ADD_REC: begin
    in_mux_add <= 3'd3; // Eksilen A yi (alu out) SEC
    reg_add <= 4'd5;    // Register a at
    state <= MULT_ADD;
end
MULT_ADD : begin
    if (z == 1) begin  // toplama bitti
        state <= EXP_SUB;
        out_mux_add <= 4'd3; // register da tutulan B yi seciyorum
        in_mux_add <= 3'd4; // feedback olan değeri seçtim
        reg_add <= 4'd1; // inA ya atiyorum
    end

    else begin
        out_mux_add <= 4'd6; // register da tutulan base degerini sec
        in_mux_add <= 3'd4; // base degerini sec
        reg_add <= 4'd1;   // inA ya ver
        state <= SEL_SUM_A;
    end
end


SEL_SUM_A: begin
    out_mux_add <= 4'd4; // register da tutulan degeri feedback yap
    in_mux_add <= 3'd4; // registerda tutulan degeri sec
    reg_add <= 4'd2; // degeri inB ye ver
    insel <= 2'd2; // base ile tutulan degeri topla
    state <= ADD_RES;
end

ADD_RES : begin
    in_mux_add <= 3'd3; // (toplanan sayiyi yani ALUout sec)
    reg_add <= 4'd4; // toplamı baska registire ata
    state <= SEL_A;
end
```

```
                SEL_A: begin
                    out_mux_add <= 4'd5; // Eksilen A yi sec
                    in_mux_add <= 3'd4;
                    reg_add <= 4'd1; // eksilen A yi inA ya ver
                    state <= MULT_SUB;


                end

                DONE: begin
                    out_mux_add <= 4'd4; // toplam degeri feedback olarak verildi
                    in_mux_add <= 4'd4; // son deger secildi
                    reg_add <= 4'd0;    // cikisa verildi
                    busy <= 0;  // we yi sıfır yapınca cikisa deger gitmiyor
                    state <= IDLE;
                end

                default: begin
                    state <= IDLE;
                    we <= 0;
                end
            endcase
        end
    end
endmodule
```

Şekil 7 Control Unit Verilog Code

We enable the registers in IDLE state. We give Input A to register4 to use it when collecting in the next states.

In START state we give input A to ALUinA. To check if A = 0.

Since the assigned value goes to ALUinA 1 clk later due to the register, we created the ARA1 state. When input A value comes to ALUinA, we make a circular shift. We check whether the value is zero at the output. If the value is zero, z = 1 and we understand that the input value is 0.

In the case of ZERO_A, if z = 1, it goes to the DONE state (In the DONE state, the value in register4 is selected and given to OUT. We initially assigned the input A value to register4). If z = 0, this time input B value is assigned to ALUinA to check if B = 0.

In ARA2, B value is shifted and z output is obtained.

In ZERO_B, if z = 1, i.e. B = 0, it goes to OUT1 state. If B != 0, it will continue for the addition.

We are trying to give 1 to OUT output at OUT1, OUT1_1 and OUT1_2. Because if the exponent of the number is 0, the result is 1. Cuconstant = 255 (8'b111111_111111). To get the result 1, both ALUinA and ALUinB are given CUconstant value and summed. 9'b1_111111_1110 is obtained. Since we can get 8 bits, we get 8'b111111_1110. When we XOR this number with cuconstant

(8‘b111111_1111) value, we get 8’b0000_0001 (decimal 1) value in ALUout. OUT1_2 is given to register4 for output and goes to DONE state.

If A!=0 or B!=0, it continues with EXP_SUB status for exponentiation.

Firstly, we subtract 1 from the exponent number (B), in the EXP_SUB state. For the subtraction process, I add the input B with the cuconst value of 8‘b1111_111111. So I subtract 1 from B. I assigned the cuconst value 8’b111111_1111 to make the subtraction process. When the subtraction process is done, we switch to SUB_REC state.

In the SUB_REC state, I assigned the decreasing result of B to register3 for the process to continue. The purpose here is to check whether z = 1 in SUB_REC to check whether the result is zero when I subtract 1 from B. If it is zero, I switch to DONE state and output the result stored in the register. If Input B = 1, input A, which I assigned to register4 in START state, will be output when I subtract 1 from B. If B is greater than 1, the process will continue, register4 will be updated, and when the last B is zero, the updated result will be given to OUT in the DONE state. If it is greater than 1, it will switch to SEL_BASE state and the sum value held in register4 (initially input A value) will be feedback.

In SEL_BASE state, we assigned the total value to Register6 to use it as a base.

Register4 keeps the updated sum result at each operation in the exponentiation process. If the exponent is over, the sum value is given as base.

In EXP_TO_BASE state, input A, which is the base number, is given to ALUinA in order to perform the addition operation as A number.

In the MULT_SUB state , we subtract 1 from input A as we did in case of EXP_SUB.

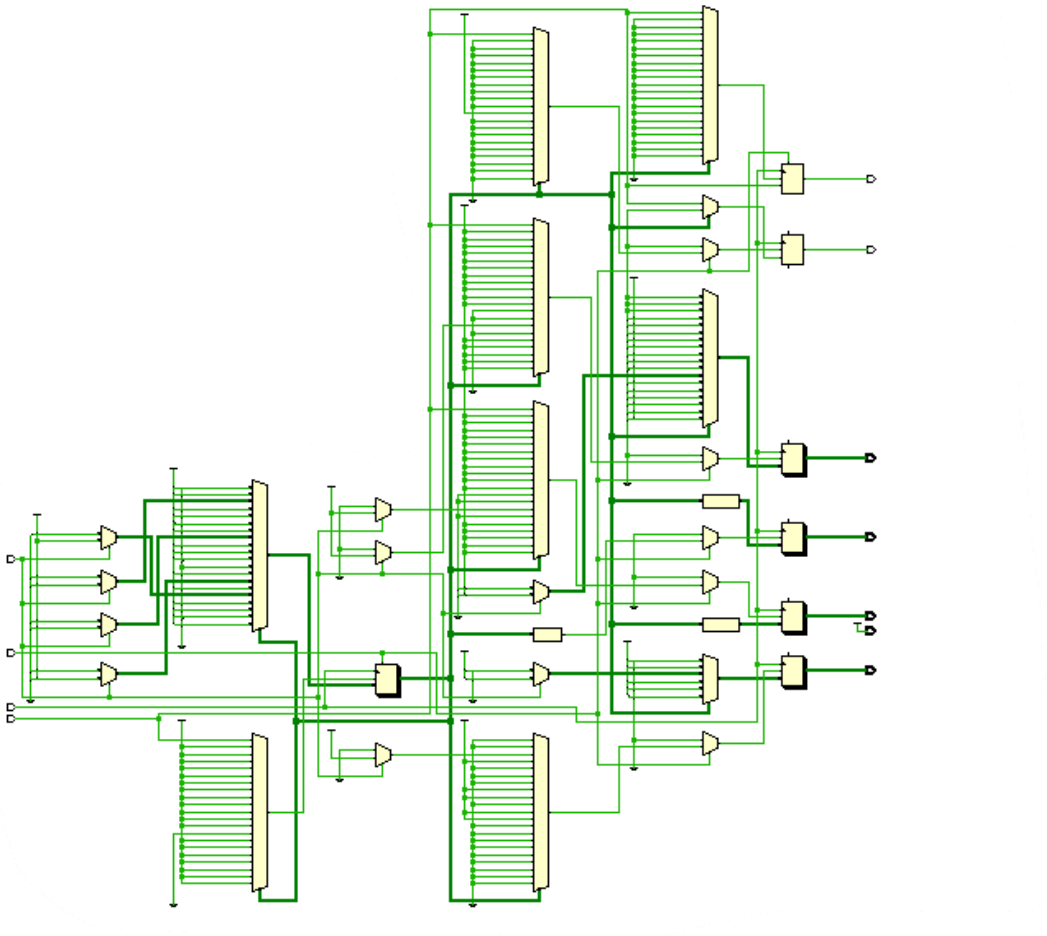
In ADD_REC, we assign the subtracted number A to register5. We decide how many times the addition operation will be performed on this number. The subtracted A number is still kept in the ALUout wire.

In the MULT_ADD state, the subtracted A value in ALUout is checked to see if it is zero. If z = 1, the addition is finished. Go to EXP_SUB state and subtract one from the exponential number (B) and continue. If not, the base value held in register6 is selected and given to ALUinA.

In SEL_SUM_A, the sum value held in register4 is given as feedback and transferred to ALUinB. We have given the base value to ALUinA. ADD block is selected with insel and the base value and total value are added.

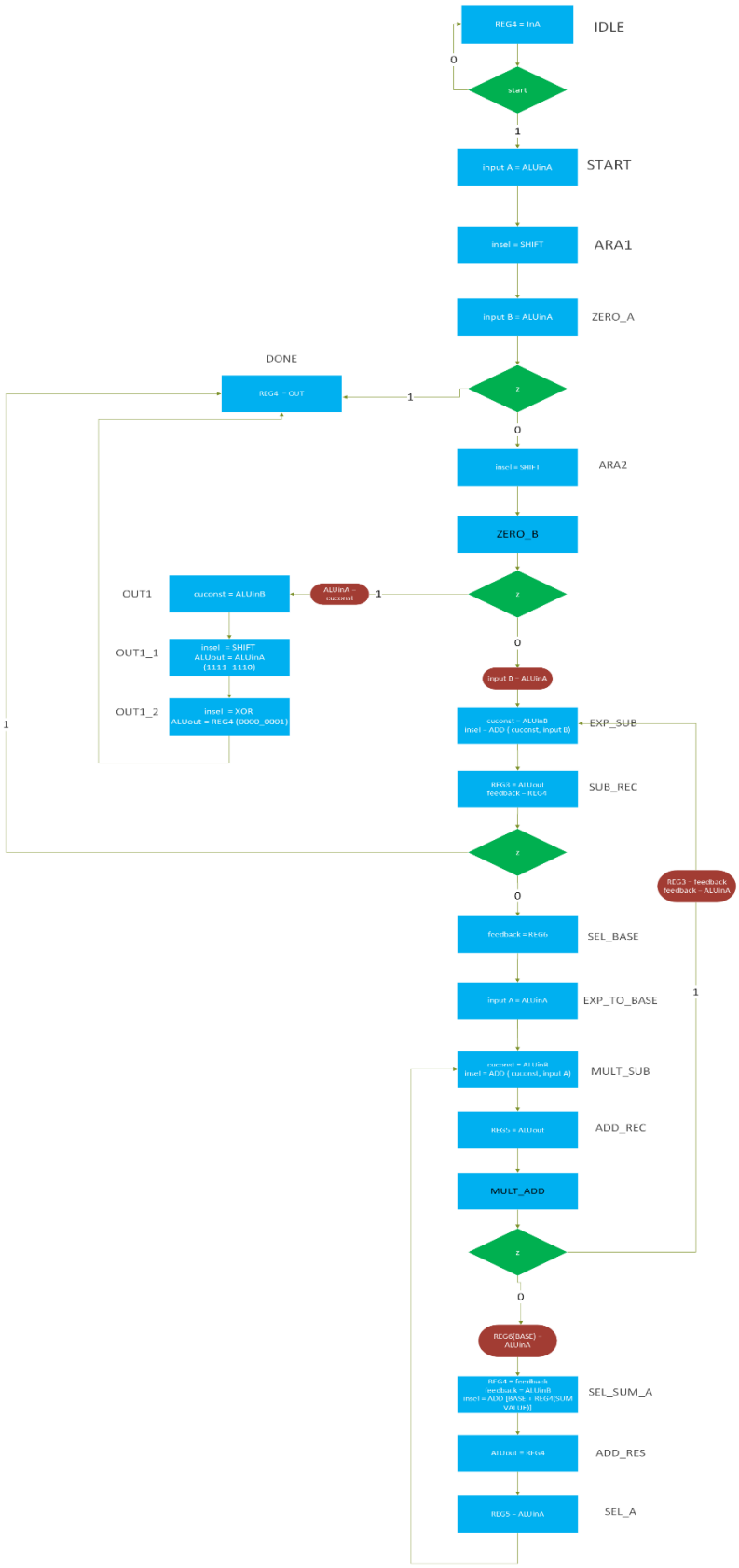
In the ADD_RES state , the sum result is assigned to register4.

In the SEL_A state, we give the value which is calculated in the ADD_REC state and held in register5 to ALUinA. We return to the MULT_SUB state.



Şekil 8 RTL Schematic of Control Unit

5. Algorithmic State Machine



6. Top Module

```
module top_module
(
    input clk,
    input rst,
    input [7:0] inA,
    input [7:0] inB,
    input start,
    output busy,
    output [7:0] out
);
    wire co;
    wire z;
    wire [1:0] insel;
    wire [7:0] cu_const;
    wire [2:0] in_mux_add;
    wire [3:0] out_mux_add;
    wire [3:0] reg_add;
    wire we;
    wire [7:0] alu_out;
    wire [7:0] alu_in_a;
    wire [7:0] alu_in_b;

    wire [4:0] cu_state = cu.state ;
    control_unit cu
    (
        .clk(clk),
        .rst(rst),
        .start(start),
        .co(co),
        .z(z),
        .busy(busy),
        .insel(insel),
        .cu_const(cu_const),
        .in_mux_add(in_mux_add),
        .out_mux_add(out_mux_add),
        .reg_add(reg_add),
        .we(we)
    );
    wire [7:0] re_out0 = rb.re_out0;
    wire [7:0] re_out1 = rb.re_out1;
    wire [7:0] re_out2 = rb.re_out2;
    wire [7:0] re_out3 = rb.re_out3;
    wire [7:0] re_out4 = rb.re_out4;
    wire [7:0] re_out5 = rb.re_out5;
    wire [7:0] re_out6 = rb.re_out6;
    wire [7:0] out_mux_out = rb.out_mux_out;
    wire [7:0] in_mux_out = rb.in_mux_out;
        REGISTER_BLOCK rb
        (
        .clk(clk),
        .rst(rst),
        .inA(inA),
        .inB(inB),
        .cu_const(cu_const),
        .in_mux_add(in_mux_add),
        .out_mux_add(out_mux_add),
        .reg_add(reg_add),
        .we(we),
        .alu_out(alu_out),
        .alu_in_a(alu_in_a),
        .alu_in_b(alu_in_b),
        .out(out)
    );
    wire [7:0] add_sum = alu.add_sum;
    ALU alu
    (
        .insel(insel),
        .alu_in_a(alu_in_a),
        .alu_in_b(alu_in_b),
        .alu_out(alu_out),
        .co(co),
        .z(z)
    );
endmodule
```

Şekil 9 Verilog Code of Top Module

Mehmet Yasir Bağcı
040200037

Salih Ömer Ongün
040220780

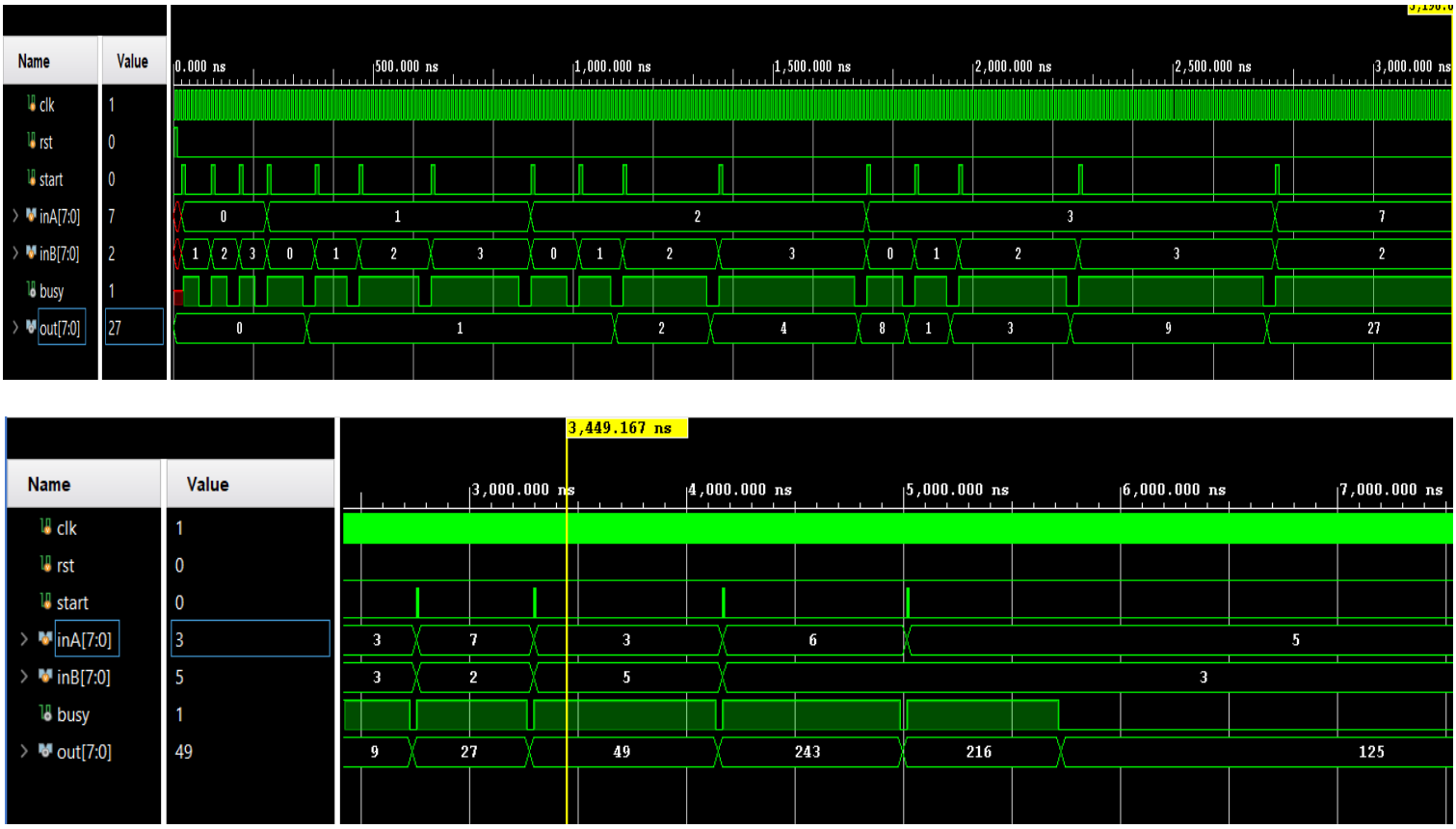
We combined ALU, REGISTER BLOCK, CONTROL UNIT modules in the Top Module. I put additional wires to see the data in the Register Block and the states of the control unit.

```
module top_module_tb;

    // Testbench sinyalleri
    reg clk;
    reg rst;
    reg start;
    reg [7:0] inA;
    reg [7:0] inB;
    wire busy;
    wire [7:0] out;

    // Test edilen modülün örneklenmesi
    top_module uut (
        .clk(clk),
        .rst(rst),
        .inA(inA),
        .inB(inB),
        .start(start),
        .busy(busy),
        .out(out)
    );

    // Clock üretimi
    always #5 clk = ~clk;

    // Test senaryoları
    initial begin

        clk = 0;
        rst = 1;    // Reset aktif
        start = 0;

        // Reset işleminden çık
        #10 rst = 0;

        #10;
        inA = 8'd0;
        inB = 8'd1;
        start = 1;
        #10 start = 0;
        wait(busy == 0); // İşlem bitene kadar bekle
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd0;
        inB = 8'd2;
        start = 1;
        #10 start = 0;
        wait(busy == 0); // İşlem bitene kadar bekle
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd0;
         inB = 8'd3;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd1;
         inB = 8'd0;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);
```

```
         #10;
         inA = 8'd1;
         inB = 8'd1;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd1;
         inB = 8'd2;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd1;
         inB = 8'd3;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd2;
         inB = 8'd0;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd2;
         inB = 8'd1;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd2;
         inB = 8'd2;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);

         #10;
         inA = 8'd2;
         inB = 8'd3;
         start = 1;
         #10 start = 0;
         wait(busy == 0); // İşlem bitene kadar bekle
         #20
         $display("A=%d, B=%d, Out=%d", inA, inB, out);
```

```
        #10;
        inA = 8'd3;
        inB = 8'd0;
        start = 1;
        #10 start = 0;
        wait(busy == 0); // İşlem bitene kadar bekle
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd3;
        inB = 8'd1;
        start = 1;
        #10 start = 0;
        wait(busy == 0); // İşlem bitene kadar bekle
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd3;
        inB = 8'd2;
        start = 1;
        #10 start = 0;
        wait(busy == 0); // İşlem bitene kadar bekle
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd3;
        inB = 8'd3;
        start = 1;
        #10 start = 0;
        wait(busy == 0); // İşlem bitene kadar bekle
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd7;
        inB = 8'd2;
        start = 1;
        #10 start = 0;
        wait(busy == 0);
        #20
        $display( "A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd3;
        inB = 8'd5;
        start = 1;
        #10 start = 0;
        wait(busy == 0);
        #20
        $display(" A=%d, B=%d, Out=%d", inA, inB, out);
        #10;
        inA = 8'd6;
        inB = 8'd3;
        start = 1;
        #10 start = 0;
        wait(busy == 0);
        #20
        $display( "A=%d, B=%d, Out=%d", inA, inB, out);

        #10;
        inA = 8'd5;
        inB = 8'd3;
        start = 1;
        #10 start = 0;
        wait(busy == 0);
        #20
        $display("A=%d, B=%d, Out=%d", inA, inB, out);
        // Testlerin sonu
        #10;
        $stop; // Simülasyonu durdur
    end

endmodule
```

Şekil 10 Simulation Codes of Top Module



Şekil 11 Simulation Waveforms

In the job description, it was written that input A and input B would be 2 bits. Therefore, we tried all 2-bit inputs. In addition, we tried some values where the result of the operation would be at most 8 bits. Since the ports and wires are 8 bits, we could not try values that would produce more than 8 bits. When the busy signal is 0, the code gives the result to OUT. We got the correct result in all operations.

```
A=  0, B=  1, Out=  0
A=  0, B=  2, Out=  0
A=  0, B=  3, Out=  0
A=  1, B=  0, Out=  1
A=  1, B=  1, Out=  1
A=  1, B=  2, Out=  1
A=  1, B=  3, Out=  1
```

```
A=  2, B=  0, Out=  1
A=  2, B=  1, Out=  2
A=  2, B=  2, Out=  4
A=  2, B=  3, Out=  8
A=  3, B=  0, Out=  1
A=  3, B=  1, Out=  3
A=  3, B=  2, Out=  9
A=  3, B=  3, Out= 27
A=  7, B=  2, Out= 49
 A=  3, B=  5, Out=243
A=  6, B=  3, Out=216
A=  5, B=  3, Out=125
```

Şekil 12 Console Outputs

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	9	8	4	rb/reg2/rout_reg[2]/C	cu/out_mux_add_reg[0]/D	7.166	1.676	5.490	∞
Path 2	∞	8	8	10	rb/reg2/rout_reg[2]/C	cu/in_mux_add_reg[2]/CE	7.118	1.324	5.794	∞
Path 3	∞	8	8	10	rb/reg2/rout_reg[2]/C	cu/in_mux_add_reg[0]/CE	7.116	1.324	5.792	∞
Path 4	∞	8	8	10	rb/reg2/rout_reg[2]/C	cu/in_mux_add_reg[1]/CE	6.824	1.324	5.500	∞
Path 5	∞	8	8	10	rb/reg2/rout_reg[2]/C	cu/reg_add_reg[0]/CE	6.824	1.324	5.500	∞
Path 6	∞	8	8	10	rb/reg2/rout_reg[2]/C	cu/reg_add_reg[1]/CE	6.824	1.324	5.500	∞
Path 7	∞	8	8	10	rb/reg2/rout_reg[2]/C	cu/reg_add_reg[2]/CE	6.824	1.324	5.500	∞
Path 8	∞	8	7	10	rb/reg2/rout_reg[2]/C	cu/state_reg[1]/D	6.782	1.324	5.458	∞
Path 9	∞	7	7	8	rb/reg2/rout_reg[2]/C	rb/reg0/rout_reg[7]_lopt_replica/D	6.773	1.200	5.573	∞
Path 10	∞	8	7	10	rb/reg2/rout_reg[2]/C	cu/state_reg[3]/D	6.584	1.324	5.260	∞

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	S
Path 11	∞	2	1	21	cu/state_reg[0]/C	cu/in_mux_add_reg[1]/D	0.346	0.186	0.160	-∞	
Path 12	∞	2	1	21	cu/state_reg[0]/C	cu/reg_add_reg[0]/D	0.350	0.186	0.164	-∞	
Path 13	∞	2	1	21	cu/state_reg[3]/C	cu/insel_reg[0]/D	0.371	0.186	0.185	-∞	
Path 14	∞	2	1	19	cu/state_reg[4]/C	cu/busy_reg/D	0.383	0.186	0.197	-∞	
Path 15	∞	2	1	21	cu/state_reg[0]/C	cu/state_reg[1]/D	0.388	0.186	0.202	-∞	
Path 16	∞	2	1	25	cu/state_reg[2]/C	cu/state_reg[2]/D	0.400	0.186	0.214	-∞	
Path 17	∞	2	1	19	cu/out_mux_add_reg[0]/C	cu/out_mux_add_reg[0]/D	0.407	0.186	0.221	-∞	
Path 18	∞	2	1	19	cu/state_reg[4]/C	cu/reg_add_reg[2]/D	0.415	0.186	0.229	-∞	
Path 19	∞	2	1	19	cu/state_reg[4]/C	cu/reg_add_reg[1]/D	0.417	0.186	0.231	-∞	
Path 20	∞	2	1	19	cu/state_reg[1]/C	cu/state_reg[3]/D	0.420	0.186	0.234	-∞	

Şekil 13 Setup and Hold Delays

As can be seen in the tables, the maximum delay is 7.166 ns. Max clock frequency is 139.53 MHz.

$$f = \frac{1}{T}$$

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	94	32600	0.29
FF	82	65200	0.13
IO	28	210	13.33
BUFG	1	32	3.13

Şekil 14 Utilization Report

94 LUT, 82 FF, 28 IO, 1 BUFG are used in the project

Name	Code	Report
Mehmet Yasir Bağcı	ALU CONTROL UNIT	• Problem Statement • ALU • Register Block • ASM • Utilization Report
Salih Ömer Ongün	REGISTER BLOCK CONTROL UNIT TOP MODULE	• Control Unit • ASM • Top Module • Timing • Utilization Report