DIGITAL SYSTEM DESIGN APPLICATIONS

Final Project

ALGORITHMIC PROBLEM SOLVING USING A SYSTEM CONSISTING OF AN ALU, A REGISTER BLOCK AND A STATE MACHINE

GROUP-2

Elif ÇATIKKAŞ 040190094

Berfin DUMAN 040190108

ABSOLUTE VALUE (A-B) ALGHORITMIC STATE MACHINE

Based on the group number, our goal is to calculate $C = abs(A - B)$; where A and B are 8-bit signed integers represented by two's complement and C is an 8-bit positive integer.

First of all, the algorithm for solving the problem was determined and a state diagram was prepared.

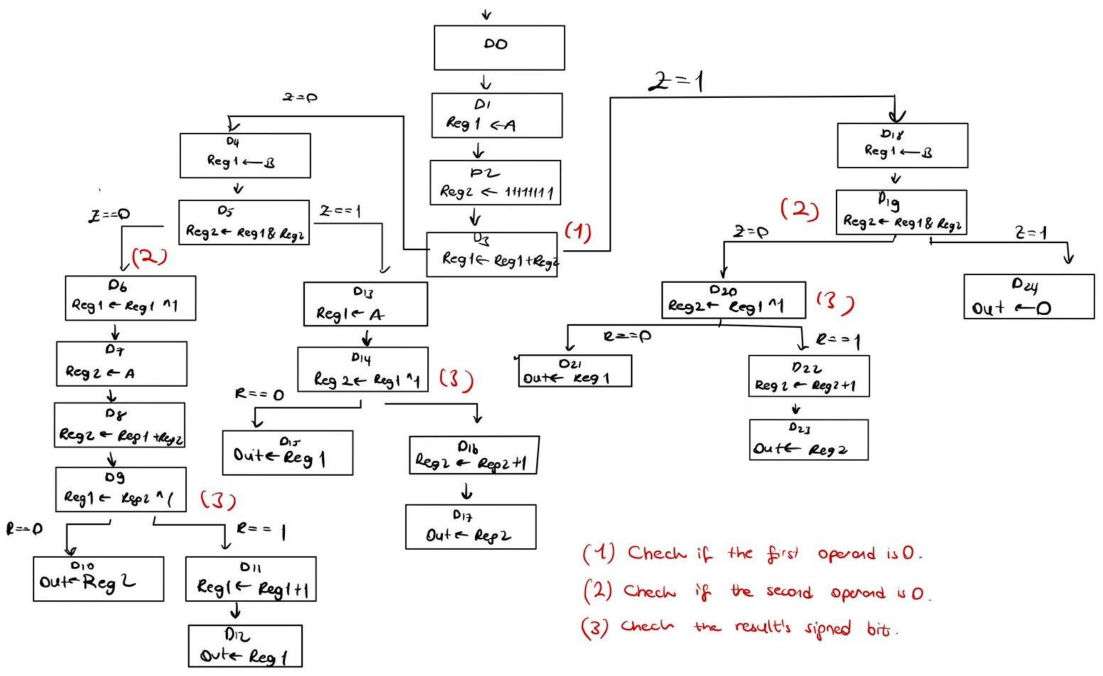
Algorithmic State Machine (ASM)

To implement our algorithm using the modules in the proposed ALU, we first created our Moore-type state diagram, using only the ALU outputs as mentioned in the assignment.

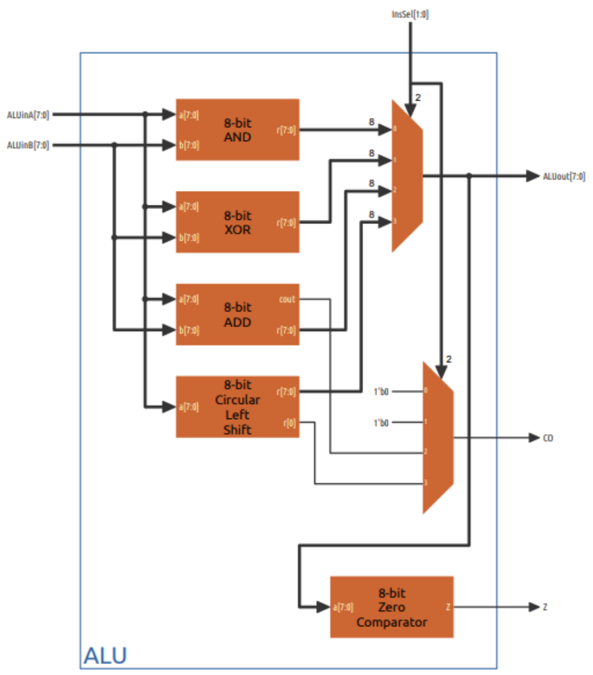
We first started by determining whether the operand A, which is the first of our inputs, is 0 or not with 8'b11111111 and evaluating the Z value coming from the Zero bit comparator output. This created two different states, the states where A is 0 and the states where A is not 0, in both states. Afterwards, we checked the B operand in the same way to see if it was 0 or not. If both were 0, the answer was directly 0 (state 24).

If A is not 0, we checked B. If B is not 0, we add A and B by taking the 2's complement of the next B. In this way, we have completed the A-B operation. By looking at the signed bit of the result, if it was negative, we converted it to positive by taking the 2's complement again.

If A is 0, we checked B and if B is not zero, we directly look at the signed bit, if it is negative, 2's complement, otherwise we print the result directly to out.



ALU (Arithmetic Logic Unit)



The Arithmetic Logic Unit (ALU) is an integral part of a computer's central processing unit (CPU), tasked with carrying out both arithmetic and logical functions. This digital circuit is pivotal in executing the instructions retrieved from the computer's memory. It is equipped with two input channels to receive operands and is capable of performing a variety of operations, including but not limited to, addition, bitwise shifting, as well as AND and XOR operations. Additionally, the ALU maintains a collection of status or flag registers. These registers are crucial as they reflect the outcome of the most recent operation conducted by the ALU. The information from these flags plays a significant role in guiding the CPU to make informed decisions and effectively manage the execution flow of the program.

ALU Verilog Code

While writing the modules, we added the "dont_touch" or “keep = true” constraint as requested, so we saw the expected modules in our rtl schematics as stated in the figures.

```
module ALU(
    input [7:0] ALUinA, ALUinB,
    input [1:0] InsSel,
    output [7:0] ALUout,
    output CO,Z
    );

    wire [7:0] AND_r, XOR_r, ADD_r, CLS_r;
    wire  ADDcout, CLSrout;
    wire [7:0] ALU_out;
    (* dont_touch="true" *)AND and_op(.a(ALUinA), .b(ALUinB), .r(AND_r));
    (* dont_touch="true" *)XOR xor_op(.a(ALUinA), .b(ALUinB), .r(XOR_r));
    (* dont_touch="true" *)ADD add_op(.a(ALUinA), .b(ALUinB), .r(ADD_r),
.cout(ADDcout));
    (* dont_touch="true" *)CLS cls_op(.a(ALUinA), .r(CLS_r));
    (* dont_touch="true" *)ZC   zc_op(.a(ALU_out), .z(Z));

    (* dont_touch="true" *)MUX_R mux_r(.sel(InsSel), .a(AND_r), .b(XOR_r),
.c(ADD_r), .d(CLS_r), .out(ALU_out));
    (* dont_touch="true" *)MUX_OUT mux_out(.sel(InsSel), .a(1'b0), .b(1'b0),
.c(ADDcout), .d(CLSrout), .out(CO));

    assign ALUout = ALU_out;
    assign CLSrout = CLS_r[0];

endmodule
```

ALU's Submodules Verilog Codes

- **AND**

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

- **XOR**

```
module XOR(
    input  [7:0] a,b,
    output [7:0] r
    );
    assign r[7] = a[7] ^ b[7];
    assign r[6] = a[6] ^ b[6];
    assign r[5] = a[5] ^ b[5];
    assign r[4] = a[4] ^ b[4];
    assign r[3] = a[3] ^ b[3];
    assign r[2] = a[2] ^ b[2];
    assign r[1] = a[1] ^ b[1];
    assign r[0] = a[0] ^ b[0];
endmodule
```

- **ADD**

```
module ADD(
    input  [7:0] a,b,
    output [7:0] r,
    output cout
    );
    assign {cout, r} = a + b;
endmodule
```

- **Circular Left Shift**

```
module ADD(
    input  [7:0] a,b,
    output [7:0] r,
    output cout
    );
    assign {cout, r} = a + b;
endmodule
```

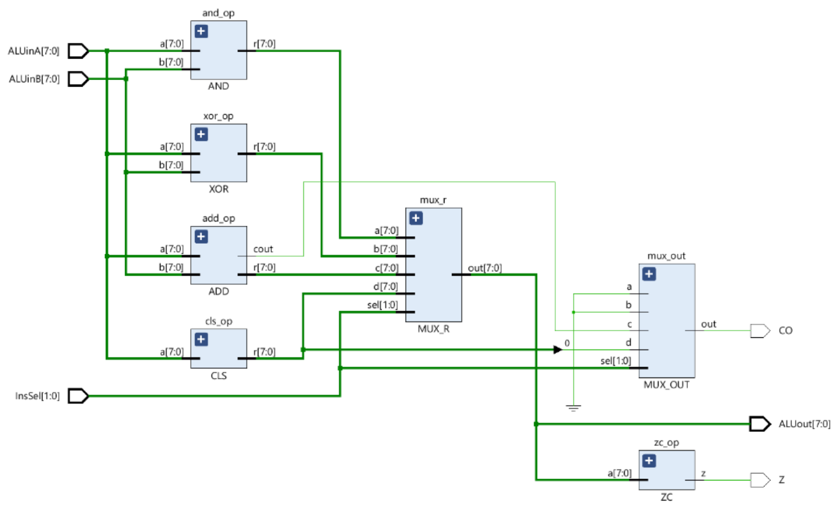
- **Zero Comparator**

```
module ZC(
    input  [7:0] a,
    output z
    );
    assign z = ~(a[0]|a[1]|a[2]|a[3]|a[4]|a[5]|a[6]|a[7]);
endmodule
```

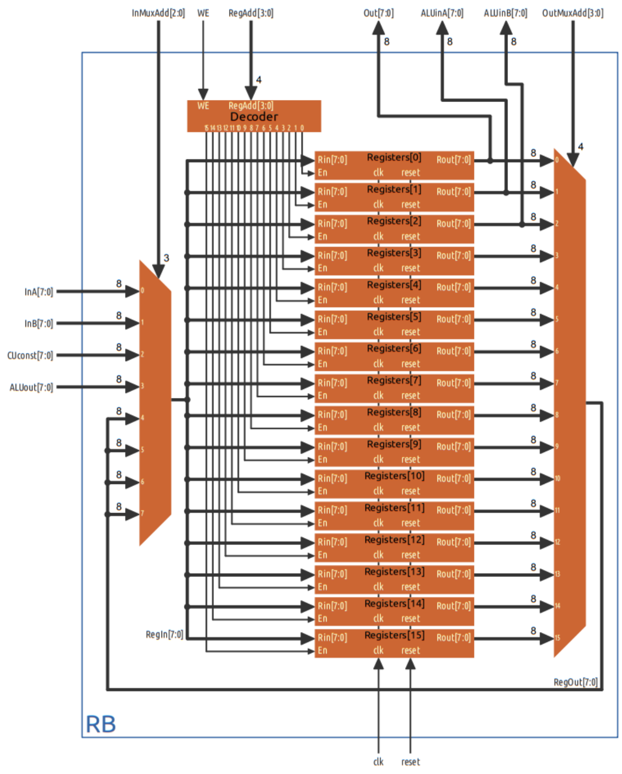
- **MUX**

```
module MUX_R(
    input  [7:0] a,b,c,d,
    input  [1:0] sel,
    output reg [7:0] out
    );
    always@* begin
        case (sel)
            2'b00: out = a;
            2'b01: out = b;
            2'b10: out = c;
            2'b11: out = d;
            default: out = 8'bZZZZZZZZ;
        endcase
```

ALU RTL Schematic



RB (Register Block)



A Register Block is a vital component in a computer processor, serving as a collection of registers that temporarily hold data and instructions while a program is being executed. These registers are essentially small, rapid storage locations integrated into the processor. The configuration of a Register Block varies according to the processor's design – some are designed with fewer but larger registers, while others might contain a greater number of smaller registers. Additionally, the Register Block is equipped with a control unit. This unit is responsible for managing data transfers between the registers, the Arithmetic Logic Unit (ALU), and overseeing the progression of the program. The design of the Register Block is tailored to effectively support these functions.

RB Verilog Code

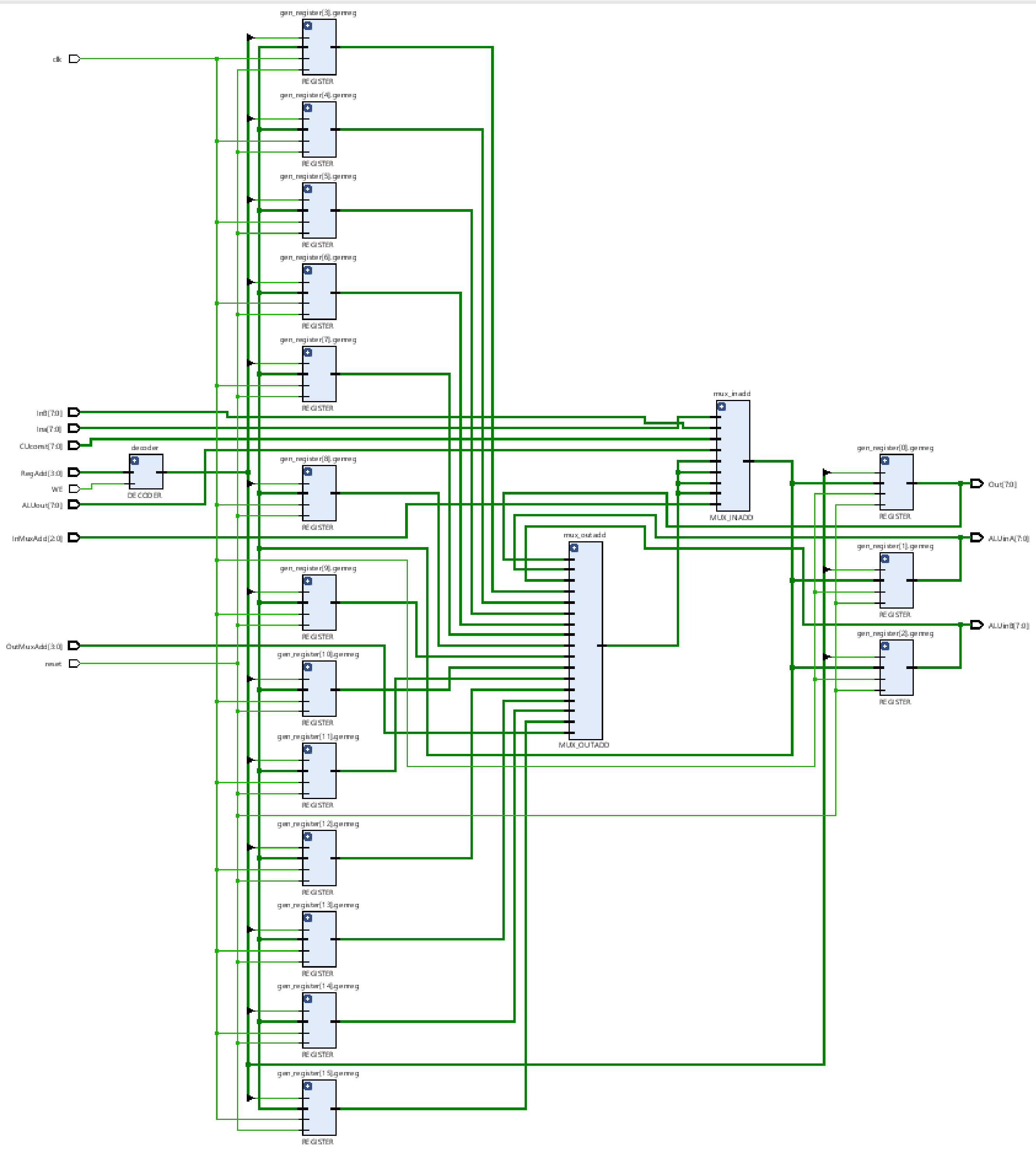
While writing the modules, we added the "dont_touch" or “keep = true” constraint as requested, so we saw the expected modules in our rtl schematics as stated in the figures.

```
`timescale 1ns / 1ps

module RB(
    input clk, reset, WE,
    input [7:0] Ina, InB, CUconst, ALUout,
    input [2:0] InMuxAdd,
    input [3:0] RegAdd, OutMuxAdd,
    output [7:0] Out, ALUinA, ALUinB
    );

    wire [15:0] decoder_out;
    wire [7:0] mux_inadd_out;
    wire [7:0] mux_outadd_out;
    wire [127:0] reg_out;

    (* dont_touch="true" *)DECODER decoder(.in(RegAdd), .out(decoder_out),
.sel(WE));
    (* dont_touch="true" *)MUX_INADD mux_inadd (.a0(Ina),
                         .a1(InB),
                         .a2(CUconst),
                         .a3(ALUout),
                         .a4(mux_outadd_out),
                         .a5(mux_outadd_out),
                         .a6(mux_outadd_out),
                         .a7(mux_outadd_out),
                         .sel(InMuxAdd),
                         .out(mux_inadd_out));

    genvar i;
    generate
        for (i = 0; i < 16; i = i + 1) begin : gen_register
            (* dont_touch="true" *)REGISTER genreg (
                .En(decoder_out[i]),
                .clk(clk),
                .reset(reset),
                .Rin(mux_inadd_out),
                .Rout(reg_out[(8*i+7):(8*i)])
            );
        end
    endgenerate
    assign Out = reg_out[7:0];
    assign ALUinA = reg_out[15:8];
    assign ALUinB = reg_out[23:16];

    (* dont_touch="true" *)MUX_OUTADD mux_outadd (.a0(reg_out[7:0]),
                           .a1(reg_out[15:8]),
                           .a2(reg_out[23:16]),
                           .a3(reg_out[31:24]),
                           .a4(reg_out[39:32]),
                           .a5(reg_out[47:40]),
                           .a6(reg_out[55:48]),
                           .a7(reg_out[63:56]),
                           .a8(reg_out[71:64]),
                           .a9(reg_out[79:72]),
                           .a10(reg_out[87:80]),
                           .a11(reg_out[95:88]),
                           .a12(reg_out[103:96]),
                           .a13(reg_out[111:104]),
                           .a14(reg_out[119:112]),
                           .a15(reg_out[127:120]),
                           .sel(OutMuxAdd),
                           .out(mux_outadd_out));
endmodule
```

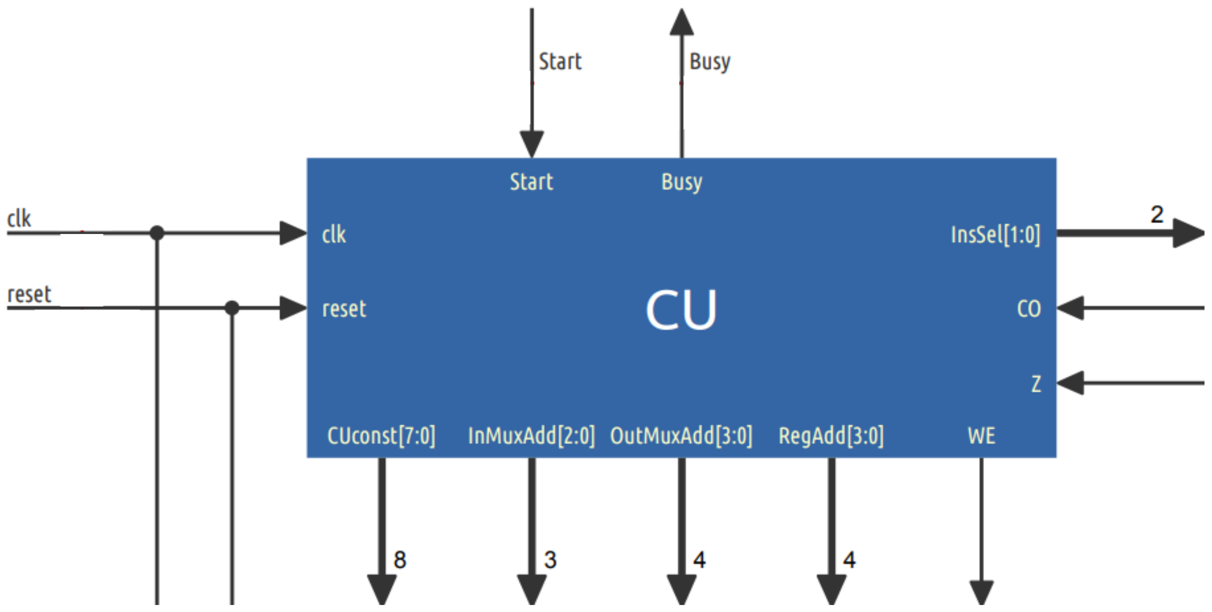
Register Generate Module Verilog Code

```
  module REGISTER(

    input En, clk, reset,
    input [7:0] Rin,
    output reg [7:0] Rout
    );
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            Rout <= 8'b00000000;
        end else if (En ==1'b1) begin
            Rout <= Rin;
        end
    end

endmodule
```

RB RTL Schematic



CU (Control Unit)



The Control Unit (CU) within a computer's processor plays a pivotal role in orchestrating the flow of data and instructions. It functions by fetching instructions from the memory, interpreting them to ascertain the required action, and subsequently dispatches control signals to various components such as the Arithmetic Logic Unit (ALU) and the register block, thereby facilitating the execution of these instructions. Additionally, the CU is instrumental in updating the program counter, which is key in controlling the sequence of the program. It also keeps tabs on the processor's status by interpreting flags provided by the ALU, using this information to make informed decisions. Essentially, the Control Unit acts as the central coordinator in a processor, decoding instructions, directing other parts of the system, managing the flow of the program, and continuously monitoring the status of the processor to ensure efficient operation.

CU Verilog Code

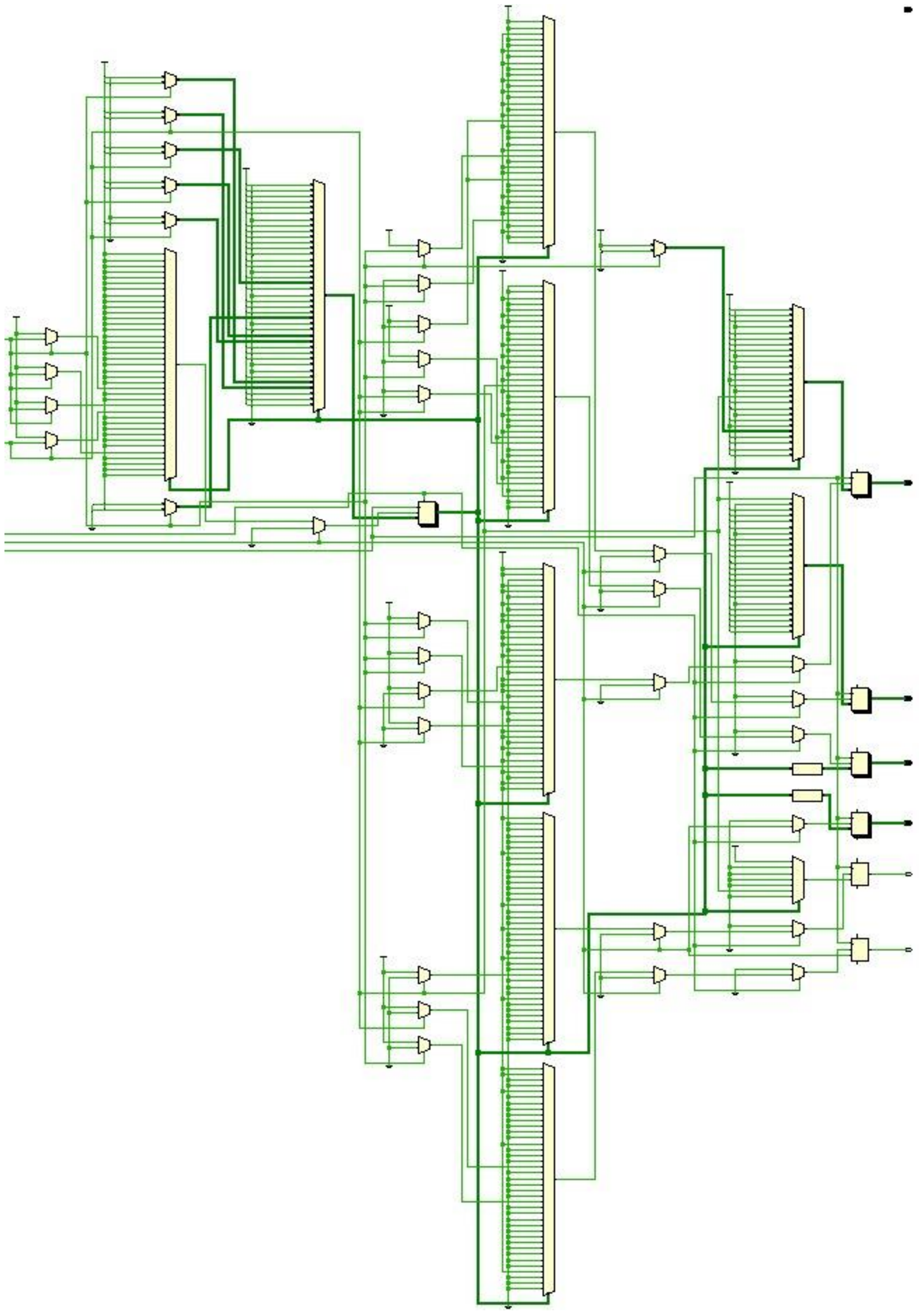
```
`timescale 1ns / 1ps

module CU(
    input Start, clk, reset, CO, Z,
    output reg [7:0] CUconst,
    output reg [2:0] InMuxAdd,
    output reg [3:0] OutMuxAdd, RegAdd,
    output reg WE, Busy,
    output reg [1:0] InsSel
    );

    reg [5:0]state;
    always @(posedge clk or posedge reset)begin
        if (reset)begin
            state <= 4'b0000;
        end
        else if (Start)begin
        CUconst <= 8'b00000001;
            case(state)
                6'd0: begin //a -> reg1
                    Busy <= 1;
                    InMuxAdd <= 3'b000;
                    RegAdd <= 4'b0001;
                    state <= 6'd1;
                    WE <= 1'b1;
                end
                6'd1: begin //1 -> reg2
                    InMuxAdd <= 3'b010;
                    RegAdd <= 4'b0010;
                    CUconst <= 8'b11111111;
                    state <= 6'd2;
                    WE <= 1'b1;
                end
                6'd2: begin //reg1 &reg2
                    state <= 6'd3;
                    InsSel <= 2'b00;
                    RegAdd <= 4'b1111;
                end
                6'd3: begin
                    if(Z==0)begin
                        InMuxAdd <= 3'b001;
                        RegAdd <= 4'b0001;
                        state <= 6'd4;
                        WE <= 1'b1;
                    end
                    else if(Z==1) begin
                        state <= 6'd32;
                        InMuxAdd <= 3'b001;
                        RegAdd <= 4'b0001;
                    end
                end
                6'd4: begin //reg1 &reg2
                    state <= 6'd5;
                    InsSel <= 2'b00;
                    RegAdd <= 4'b1111;
                end
                6'd5: begin
                    if(Z==0)begin
                        state <= 6'd7;
                        InsSel <= 2'b11;
                        RegAdd <= 4'b1111;
                    end
                    else if (Z==1)begin
                        state <= 6'd24;
                        InMuxAdd <= 3'b000;
                        RegAdd <= 4'b0001;
                    end
                end
                6'd7: begin
                    if(CO==0)begin
                        state <= 6'd8;
                        InMuxAdd <= 3'b000;
                        RegAdd <= 4'b0001;
                        WE <= 1'b1;
                    end
                    else begin
                        state <= 6'd16;
                        InMuxAdd <= 3'b001;
                        RegAdd <= 4'b0001;
                        WE <= 1'b1;
                    end
                end
                6'd8: begin //reg1 ^reg2
                    state <= 6'd9;
                    InsSel <= 2'b01;
                end
                6'd9: begin //reg1 ^reg2
                    state <= 6'd10;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0001;
                end
                6'd10: begin
                    state <= 6'd11;
                    InMuxAdd <= 3'b010;
                    RegAdd <= 4'b0010;
                end
                6'd11: begin //reg1 ^reg2
                    state <= 6'd12;
                    InsSel <= 2'b010;
                end
                6'd12: begin //reg1 ^reg2
                    state <= 6'd13;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0001;
                end
                6'd13: begin //1 -> reg2
                    InMuxAdd <= 3'b0001;
                    RegAdd <= 4'b0010;
                    state <= 6'd14;
                    WE <= 1'b1;
                end
                6'd14: begin //reg1 ^reg2
                    state <= 6'd15;
                    InsSel <= 2'b010;
                end
                6'd15: begin //reg1 ^reg2
                    state <= 6'd0;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0000;
                    Busy <= 0;
                end
                6'd16: begin //reg1 ^reg2
                    state <= 6'd17;
                    InsSel <= 2'b01;
                end
                6'd17: begin //reg1 ^reg2
                    state <= 6'd18;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0001;
                end
                6'd18: begin
                    state <= 6'd19;
                    InMuxAdd <= 3'b010;
                    RegAdd <= 4'b0010;
                end
                6'd19: begin //reg1 ^reg2
                    state <= 6'd20;
                    InsSel <= 2'b010;
                end
                6'd20: begin //reg1 ^reg2
                    state <= 6'd21;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0001;
                end
```

```
6'd21: begin //1 -> reg2
    InMuxAdd <= 3'b000;
    RegAdd <= 4'b0010;
    state <= 6'd22;
    WE <= 1'b1;
end
6'd22: begin //reg1 ^reg2
    state <= 6'd23;
    InsSel <= 2'b010;
end
6'd23: begin //reg1 ^reg2
    state <= 6'd0;
    Busy <= 0;
    InMuxAdd <= 3'b011;
    RegAdd <= 4'b0000;
end
6'd24: begin //reg1 ^reg2
    state <= 6'd25;
    InsSel <= 2'b11;
end
6'd25: begin
    if(CO==0)begin
        state <= 6'd26;
        InMuxAdd <= 3'b001;
        RegAdd <= 4'b0010;
        WE <= 1'b1;
    end
    else begin
        state <= 6'd28;
        InsSel <= 2'b01;
    end
end
6'd26: begin //reg1 ^reg2
    state <= 6'd27;
    InsSel <= 2'b10;
end
6'd27: begin //reg1 ^reg2
    state <= 6'd0;
    Busy <= 0;
    InMuxAdd <= 3'b011;
    RegAdd <= 4'b0000;
end
```

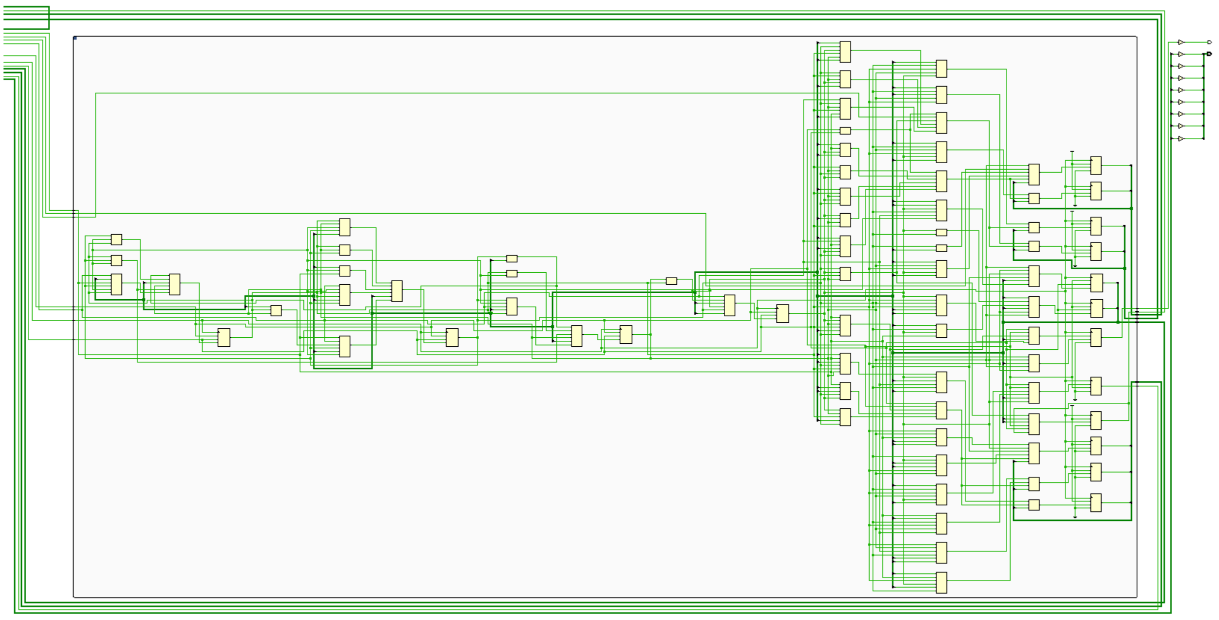
```
                6'd28: begin //reg1 ^reg2
                    state <= 6'd29;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0001;
                end
                6'd29: begin //reg1 ^reg2
                    state <= 6'd30;
                    InMuxAdd <= 3'b010;
                    RegAdd <= 4'b0010;
                end
                6'd30: begin //reg1 ^reg2
                    state <= 6'd31;
                    InsSel <= 2'b10;
                end
                6'd31: begin //reg1 ^reg2
                    state <= 6'd0;
                    Busy <= 0;
                    InMuxAdd <= 3'b011;
                    RegAdd <= 4'b0000;
                end
                6'd32: begin //reg1 ^reg2
                    state <= 6'd33;
                    InsSel <= 2'b01;
                end
                6'd33: begin //reg1 ^reg2
                    if(Z==0)begin
                        state <=6'd34;
                        InsSel <= 2'b11;
                    end
                    else if(Z==1) begin
                        state<= 6'd39;
                        CUconst <= 8'b00000001;
                    end
                end
                6'd34:begin
                    if(CO==0)begin
                        InMuxAdd <= 4'b0001;
                        RegAdd <= 4'b0000;
                        state <= 6'd0;
                        Busy <= 0;
                    end
                    else if(CO==1) begin
                        state <= 6'd35;
                        InsSel <= 2'b01;
                    end
```

```
                 end
                 6'd35:begin
                     InMuxAdd <= 3'b011;
                     RegAdd <= 4'b0001;
                     state <= 6'd36;
                 end
                 6'd36:begin
                     InMuxAdd <= 3'b010;
                     RegAdd <= 4'b0010;
                     state <= 6'd37;
                 end
                 6'd37:begin
                     InsSel <= 2'b10;
                     state <= 6'd38;
                 end
                 6'd38:begin
                     InMuxAdd <= 3'b011;
                     RegAdd <= 4'b0000;
                     state <= 6'd0;
                     Busy <= 0;
                 end
                 6'd38:begin
                     InMuxAdd <= 3'b010;
                     RegAdd <= 4'b0000;
                     state <= 6'd0;
                     Busy <= 0;
                 end

                 default: begin state <= 6'b0000; end
             endcase
         end
     end
endmodule
```

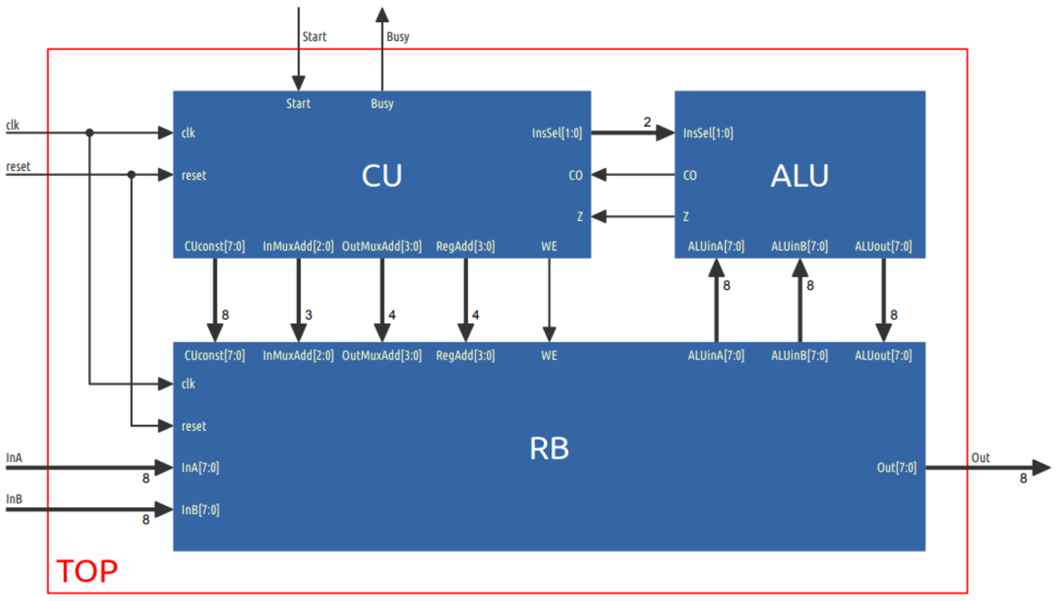
RTL Schematic



Technology Schematic



Top Module



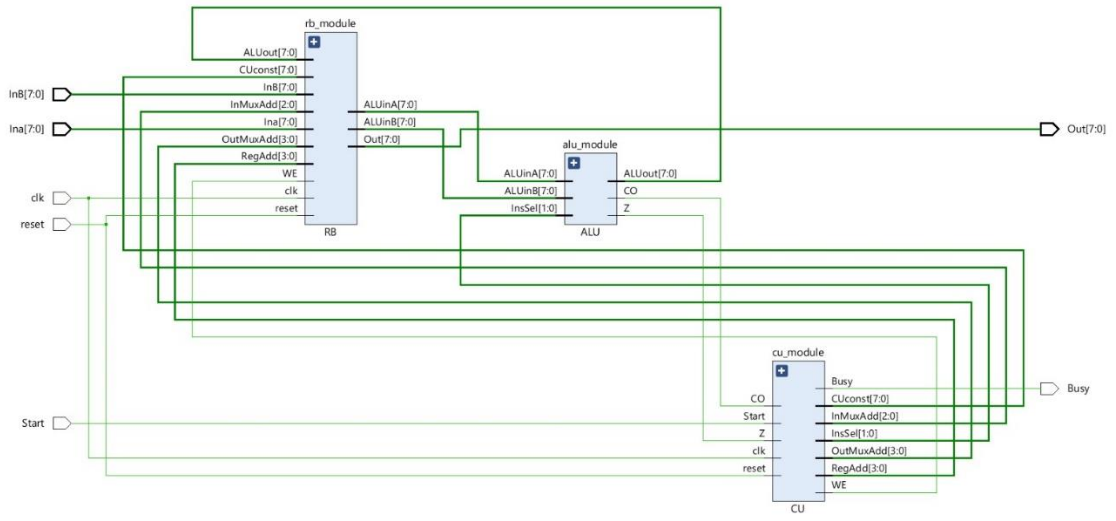
The module described encompasses all the previously mentioned blocks, including the Control Unit, Arithmetic Logic Unit (ALU), and the Register Block. The interconnection of these blocks within the module has been verified through simulation results, confirming the correct functionality of our circuit. Detailed insights and outcomes of these simulations are provided in the following sections, offering a comprehensive analysis of the circuit's performance and effectiveness.

Top Module Verilog Code

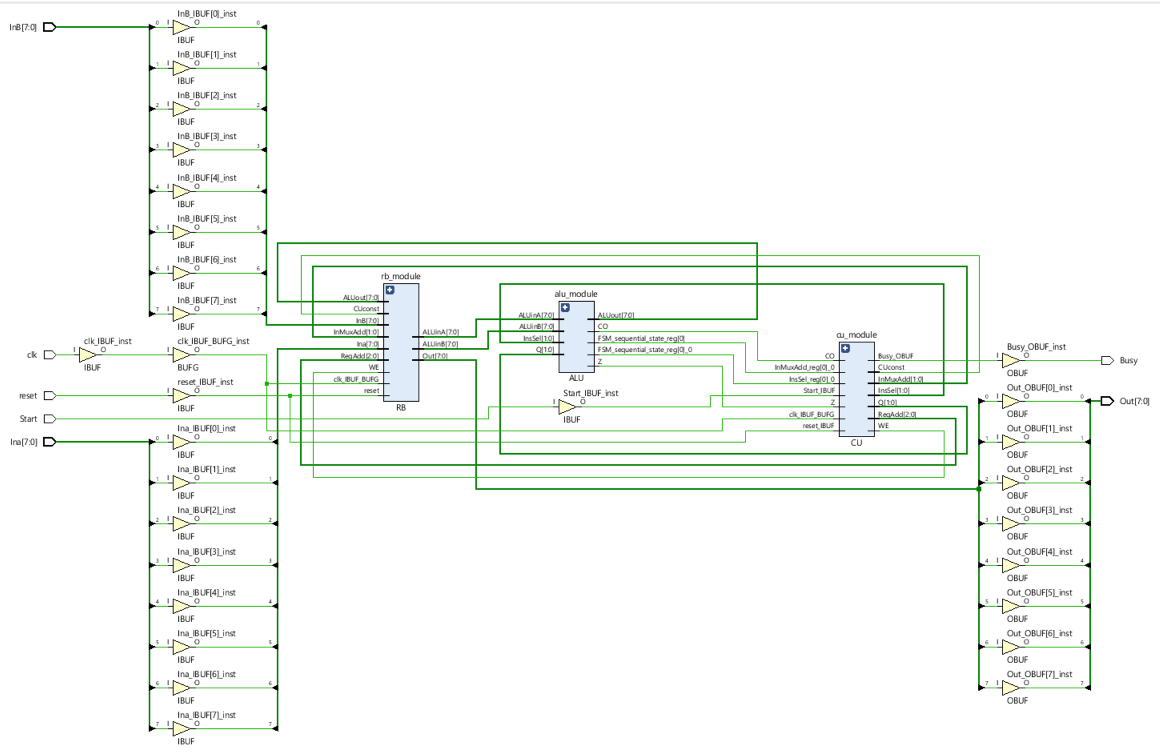
```
`timescale 1ns / 1ps

module TOP(
    input clk, reset, Start,
    input [7:0] Ina, InB,
    output Busy,
    output [7:0] Out
    );
    wire CO, Z, WE;
    wire [1:0] InsSel;
    wire [2:0] InMuxAdd;
    wire [3:0] OutMuxAdd, RegAdd;
    wire [7:0] CUconst, ALUinA, ALUinB, ALUout;

    CU cu_module(.clk(clk),
                 .reset(reset),
                 .Start(Start),
                 .CO(CO),
                 .Z(Z),
                 .Busy(Busy),
                 .InsSel(InsSel),
                 .CUconst(CUconst),
                 .InMuxAdd(InMuxAdd),
                 .OutMuxAdd(OutMuxAdd),
                 .RegAdd(RegAdd),
                 .WE(WE));
    ALU alu_module(.ALUinA(ALUinA),
                   .ALUinB(ALUinB),
                   .InsSel(InsSel),
                   .CO(CO),
                   .Z(Z),
                   .ALUout(ALUout));
    RB rb_module(.clk(clk),
                 .reset(reset),
                 .Ina(Ina),
                 .InB(InB),
                 .CUconst(CUconst),
                 .InMuxAdd(InMuxAdd),
                 .OutMuxAdd(OutMuxAdd),
                 .RegAdd(RegAdd),
                 .WE(WE),
                 .ALUinA(ALUinA),
                 .ALUinB(ALUinB),
                 .ALUout(ALUout),
                 .Out(Out));
endmodule
```

Top Module RTL Schematic

The blocks within this design are structured intentionally to achieve a specific architectural layout. This deliberate arrangement has resulted in a circuit diagram that exhibits a notable similarity to the RTL (Register Transfer Level) Schematic. This resemblance between the two indicates a cohesive and well-planned structural design in the circuit's architecture.

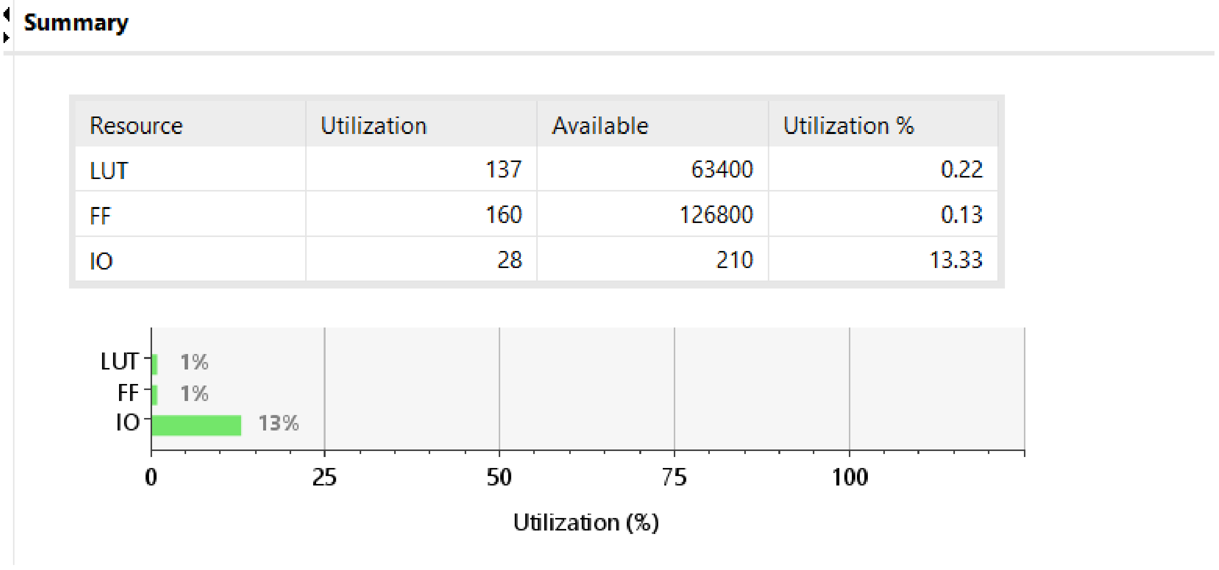


Top Module Technology Schematic



Utilization Summary

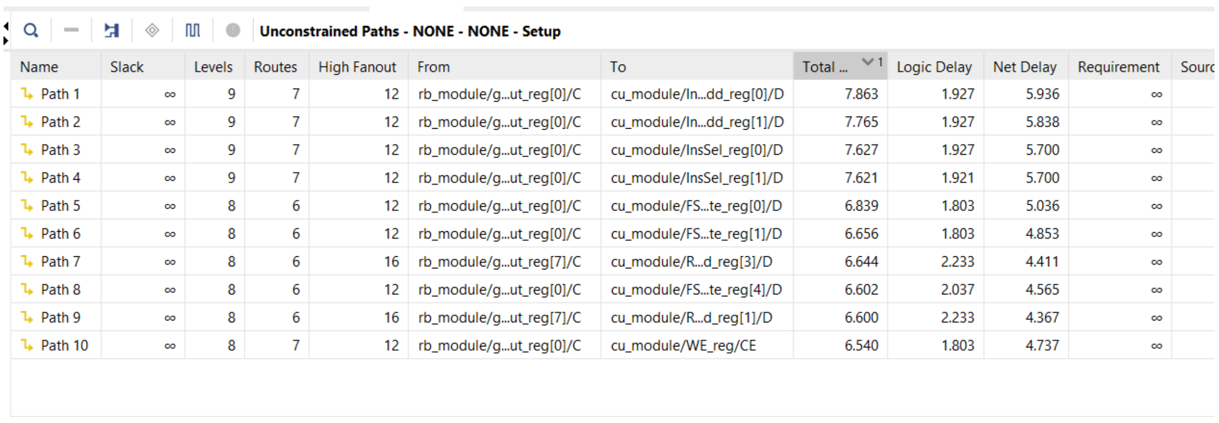
As seen in the circuit, 137 LUTs, 1160 FF’s and 28 IO are used.


Summary

Resource	Utilization	Available	Utilization %
LUT	137	63400	0.22
FF	160	126800	0.13
IO	28	210	13.33

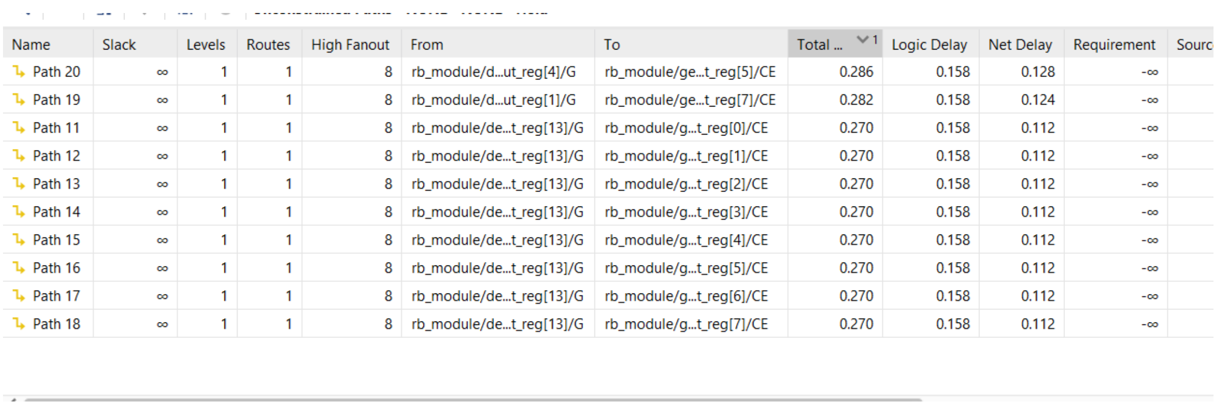
Timing Summary

SETUP


Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Sourc
Path 1	∞	9	7	12	rb_module/g...ut_reg[0]/C	cu_module/In...dd_reg[0]/D	7.863	1.927	5.936	∞	
Path 2	∞	9	7	12	rb_module/g...ut_reg[0]/C	cu_module/In...dd_reg[1]/D	7.765	1.927	5.838	∞	
Path 3	∞	9	7	12	rb_module/g...ut_reg[0]/C	cu_module/InsSel_reg[0]/D	7.627	1.927	5.700	∞	
Path 4	∞	9	7	12	rb_module/g...ut_reg[0]/C	cu_module/InsSel_reg[1]/D	7.621	1.921	5.700	∞	
Path 5	∞	8	6	12	rb_module/g...ut_reg[0]/C	cu_module/FS...te_reg[0]/D	6.839	1.803	5.036	∞	
Path 6	∞	8	6	12	rb_module/g...ut_reg[0]/C	cu_module/FS...te_reg[1]/D	6.656	1.803	4.853	∞	
Path 7	∞	8	6	16	rb_module/g...ut_reg[7]/C	cu_module/R...d_reg[3]/D	6.644	2.233	4.411	∞	
Path 8	∞	8	6	12	rb_module/g...ut_reg[0]/C	cu_module/FS...te_reg[4]/D	6.602	2.037	4.565	∞	
Path 9	∞	8	6	16	rb_module/g...ut_reg[7]/C	cu_module/R...d_reg[1]/D	6.600	2.233	4.367	∞	
Path 10	∞	8	7	12	rb_module/g...ut_reg[0]/C	cu_module/WE_reg/CE	6.540	1.803	4.737	∞	

HOLD

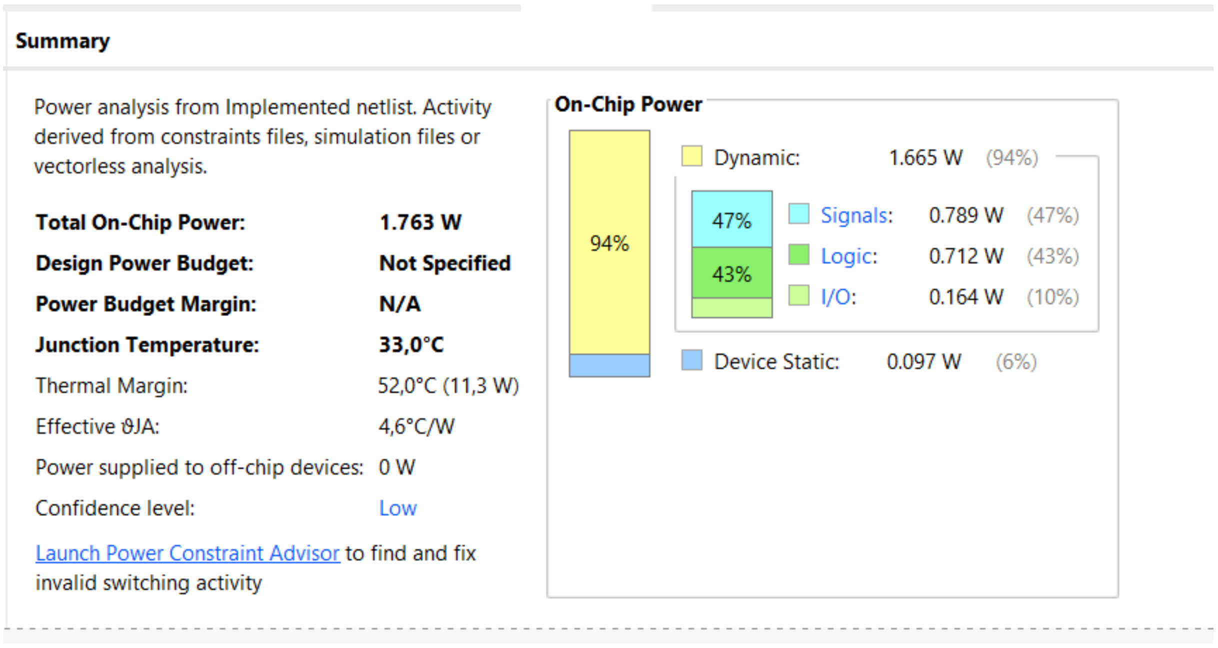


Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Sourc
Path 20	∞	1	1	8	rb_module/d...ut_reg[4]/G	rb_module/ge...t_reg[5]/CE	0.286	0.158	0.128	-∞	
Path 19	∞	1	1	8	rb_module/d...ut_reg[1]/G	rb_module/ge...t_reg[7]/CE	0.282	0.158	0.124	-∞	
Path 11	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[0]/CE	0.270	0.158	0.112	-∞	
Path 12	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[1]/CE	0.270	0.158	0.112	-∞	
Path 13	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[2]/CE	0.270	0.158	0.112	-∞	
Path 14	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[3]/CE	0.270	0.158	0.112	-∞	
Path 15	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[4]/CE	0.270	0.158	0.112	-∞	
Path 16	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[5]/CE	0.270	0.158	0.112	-∞	
Path 17	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[6]/CE	0.270	0.158	0.112	-∞	
Path 18	∞	1	1	8	rb_module/de...t_reg[13]/G	rb_module/g...t_reg[7]/CE	0.270	0.158	0.112	-∞	

As observed in the previous analysis, the maximum delay encountered in our circuit is 5.854 nanoseconds. To determine the maximum operational frequency of the circuit, we utilize the formula $f = \frac{1}{T}$ (where f is the frequency in hertz and is the T time period in seconds). Applying this formula, we calculate the maximum frequency to be approximately 127.18 MHz.

Power Consumption

The total power consumption of the chip is 1.763W, as shown below. The breakdown of individual power consumptions is provided on the right. A significant portion of this consumption is attributed to Dynamic Power. While the combined power usage of signal and logic is about 90%, the I/O value notably contributes to the remaining 10% of the total power consumption.



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.763 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**33,0°C**
Thermal Margin:	52,0°C (11,3 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

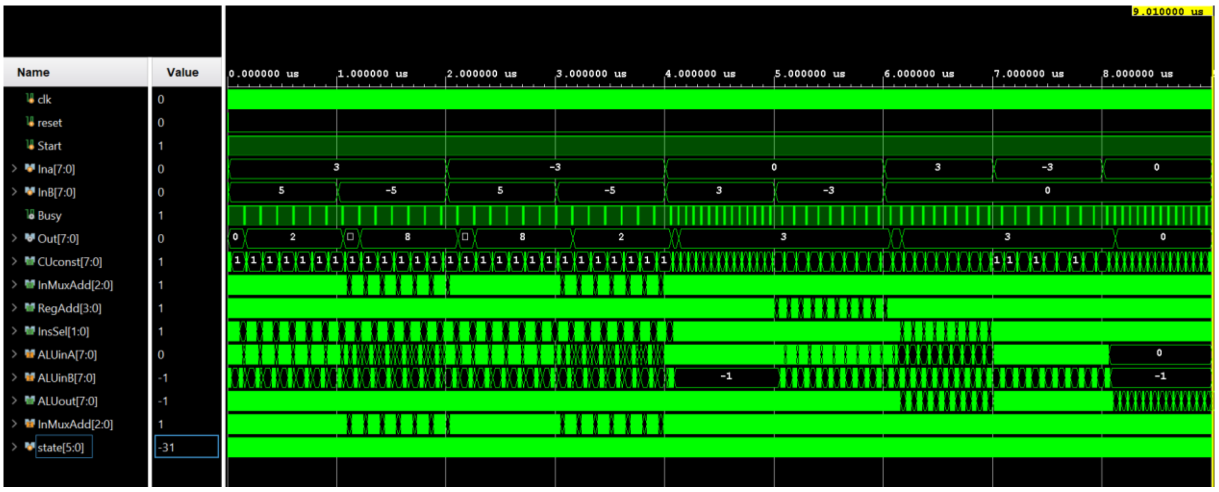
🟨	Dynamic:	1.665 W	(94%)
🟦	Signals:	0.789 W	(47%)
🟩	Logic:	0.712 W	(43%)
🟩	I/O:	0.164 W	(10%)
🟦	Device Static:	0.097 W	(6%)

Testbench Code

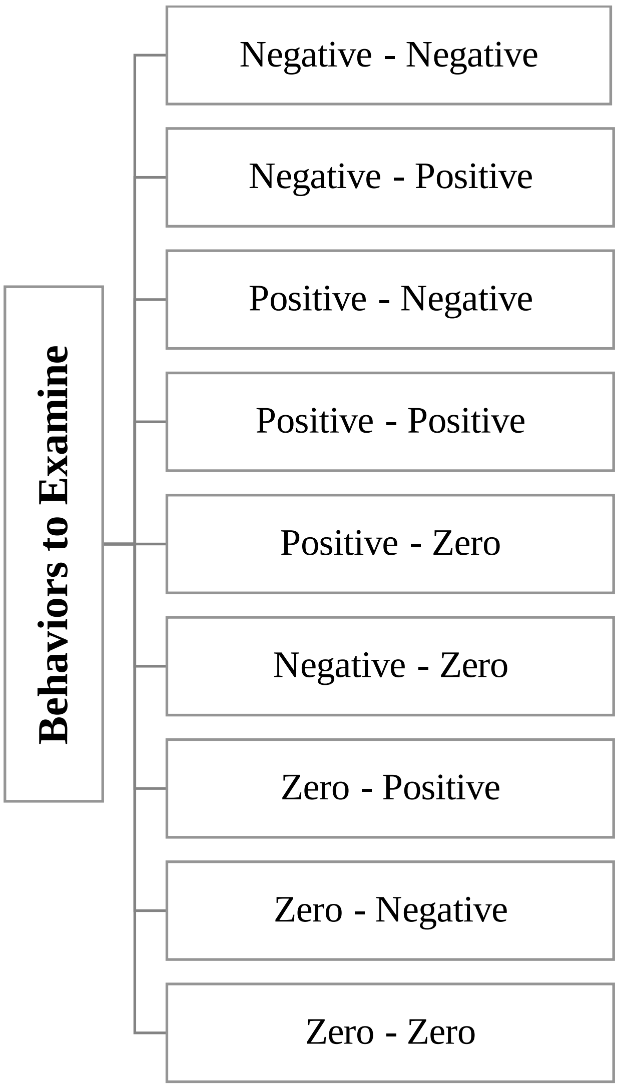
```
`timescale 1ns / 1ps

module TOP_tb();
    reg clk, reset, Start;
    reg signed [7:0] Ina, InB;
    wire Busy;
    wire [7:0] Out;

    TOP uut(.clk(clk),
            .reset(reset),
            .Start(Start),
            .Ina(Ina),
            .InB(InB),
            .Busy(Busy),
            .Out(Out)
            );

    initial begin
        clk=1'b0;
        reset=1'b1;
        #5;
    end
    always begin
        clk =~clk;
        #5;
    end

    initial begin
        reset=1'b1;
        #5;
        reset=1'b0;
        #5;
        Ina = 8'b00000011;InB = 8'b00000101;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b00000011;InB = 8'b11111011;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b11111101;InB = 8'b00000101;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b11111101;InB = 8'b11111011;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b00000000;InB = 8'b00000011;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b00000000;InB = 8'b11111101;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b00000011;InB = 8'b00000000;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b11111101;InB = 8'b00000000;Start = 1'b1;#1000;Start = 0;
        Ina = 8'b00000000;InB = 8'b00000000;Start = 1'b1;#1000;Start = 0;

        $finish;
    end
endmodule
```

Behavioral Simulation

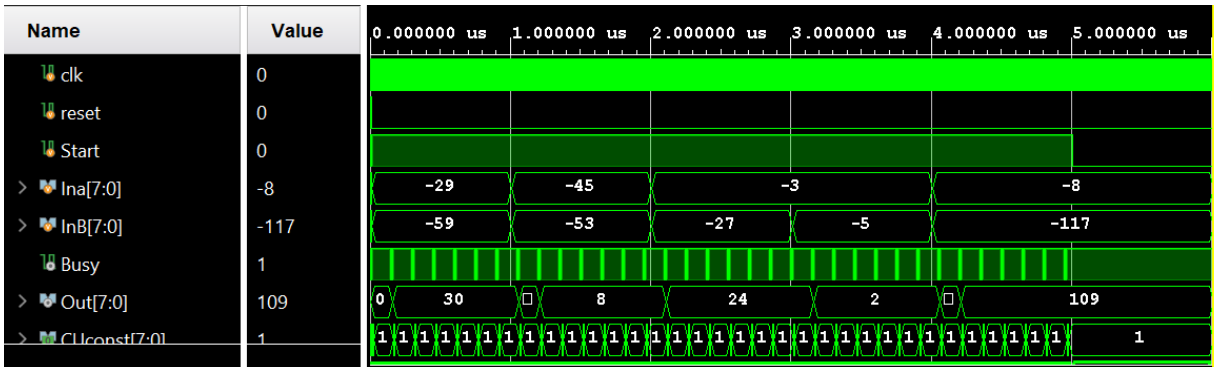
First of all, a behavioral simulation in which all situations are presented was created and added. Detailed analyzes are made in the following section.



All the situations to be examined are grouped under 9 subheadings.



1. Negative – Negative



Ex: $C = abs(A - B)$

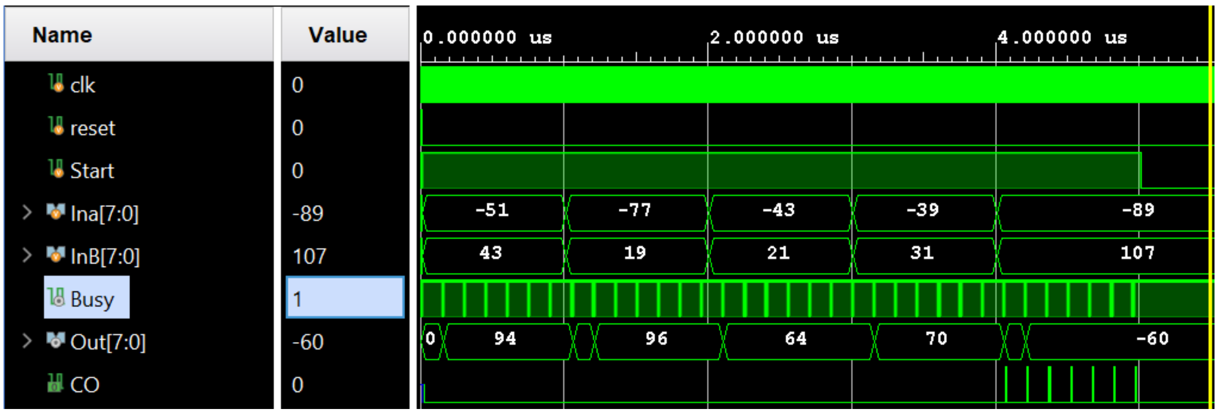
$= abs((-45) - (-53))$

$= abs(-8)$

$= 8$

In the second column, it can be observed that the output is as desired.

2. Negative – Positive



Ex: $C = abs(A - B)$

$= abs((-77) - (19))$

$= abs(-96)$

$= 96$

In the second column, it can be observed that the output is as desired.

Ex: $C = abs(A - B)$

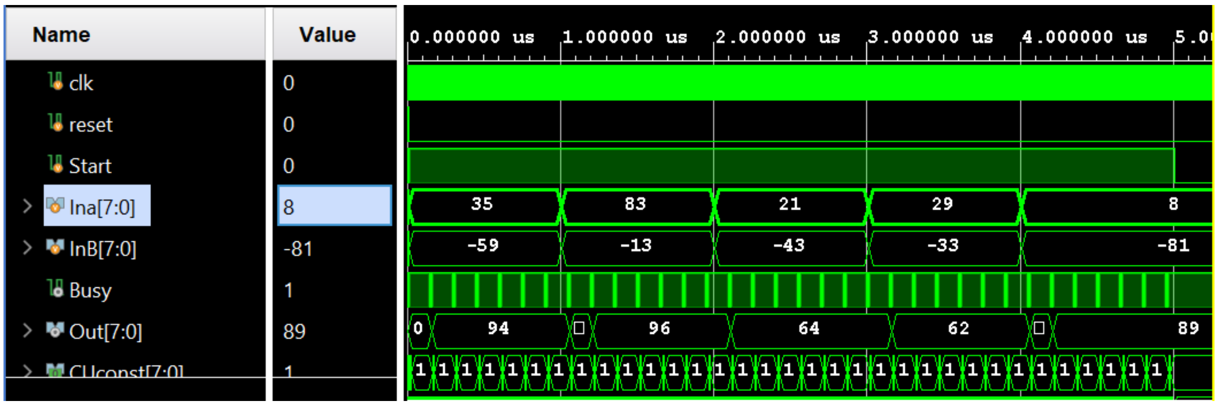
$= abs((-89) - (107))$

$= abs(-196)$

$= 196$

In the last column, the result is observed as -60, and the CO (Carry Out) output is present. The reason for this is that the result exceeds 8 bits. In binary, 100000000 is equal to 256, and subtracting 60 from 256 results in 196. This overflow in the result is reflected in the Carry Out.

3. Positive – Negative



Ex: $C = abs(A - B)$

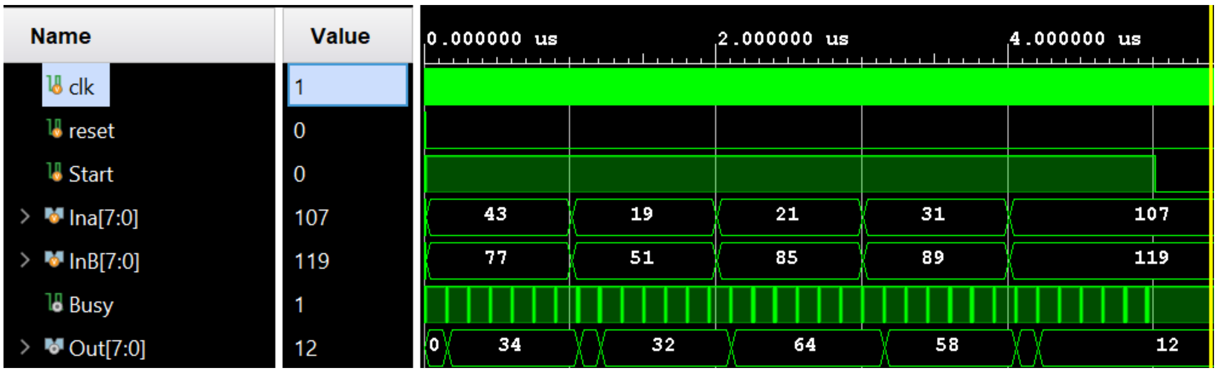
$= abs((83) - (-13))$

$= abs(96)$

$= 96$

In the second column, it can be observed that the output is as desired.

4. Positive – Positive



Ex: $C = abs(A - B)$

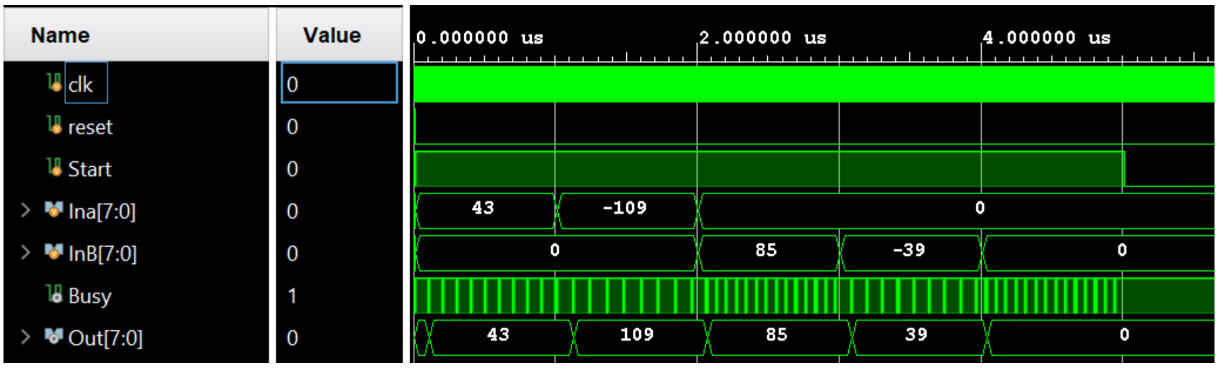
$= abs((19) - (51))$

$= abs(-32)$

$= 32$

In the second column, it can be observed that the output is as desired.

5. With 0 Conditions



Ex: $C = abs(A - B)$

$= abs((43) – (0)) =\ 43$

$= abs((-109) – (0)) =\ 109$

$= abs((0) – (85)) =\ 85$

$= abs((0) – (-39)) =\ 39$

$= abs((0) – (0)) =\ 0$

These results can be observed that the output is as desired.

Work Package Table

	Berfin DUMAN	Elif ÇATIKKAŞ
Literature Review	x	x
Research	x	x
Preparation	x	x
Design/Verilog	x	x
Report	x	x