Digital System Design Applications

FINAL PROJECT

EHB436E

CRN: 10345

Group:1

Nurgül Gürler

040190251

Hatice Nur Andı
040200203

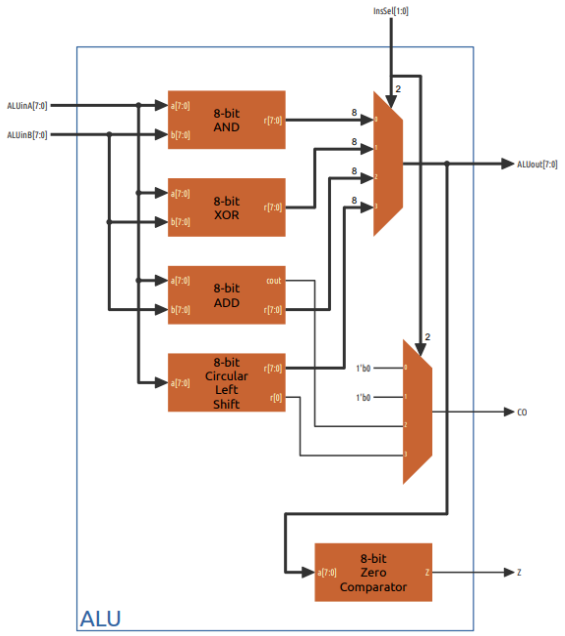
Hatice Nur Andı 040200203
Nurgül Gürler 040190251

Problem

The assigned problem to us is: “Calculate $C = A - 3B$, where A and B are 6-bit positive integers and C is an 8-bit sign integer.”

Arithmetic Logic Unit

The Arithmetic Logic Unit (ALU) performs four operations on input A and input B, and based on the InsSel (Instruction Select) bits, selects one of the four operations as the output. Simultaneously, it provides the Carry Out and Zero outputs. The four operations are designed as separate submodules.



- AND

```
module AND_8bit(
    input [7:0] a,
    input [7:0] b,
    output [7:0] r
    );

    assign r = a & b;

endmodule
```

- ADD

```
module ADD_8bit(

    input [7:0] a, b,
    output reg [7:0] r,
    output reg cout
);

reg [8:0] result;
reg carry_out;

always @(a or b) begin
    result = a + b;
    carry_out = result[8];
end

always @(result or carry_out) begin
    r = result[7:0];
    cout = carry_out;
end
```

- XOR

```
module XOR_8bit(
    input [7:0] a,
    input [7:0] b,
    output [7:0] r
    );

    assign r = a ^ b;

endmodule
```

- Circular Left Shift

```
module CLS_8bit(
    input [7:0] a,
    input [2:0] b,
    output reg [7:0] r,
```

```
    output Z
    );
    integer i;
    always@(*) begin
        for ( i = 0; i < 8; i = i + 1) begin
            if (i < b)
                r[i] = a[7 - (b - 1 - i)];
            else
                r[i] = a[i - b];
        end
    end

assign Z= r[0];

endmodule
```

- Zero Comparator

```
module ZC_8bit(
input [7:0] a,
output Z
);

assign Z = (a ==8'b0 )? 1: 0;

endmodule
```

- 4 to 1 MUX (8-bit)

```
module MUX(
    input [7:0] A,B,C,D,
    //input [3:0] D,
    input [1:0] S,
    output [7:0] O
    );
    wire [7:0] w0,w1,w2,w3;
    assign w0=A & {8{~S[0]}} & {8{~S[1]}};
    assign w1=B& {8{S[0]}}& {8{~S[1]}};
    assign w2=C& {8{~S[0]}}& {8{S[1]}};
    assign w3=D& {8{S[0]}}& {8{S[1]}};
    assign O= w0|w1|w2|w3;
endmodule
```

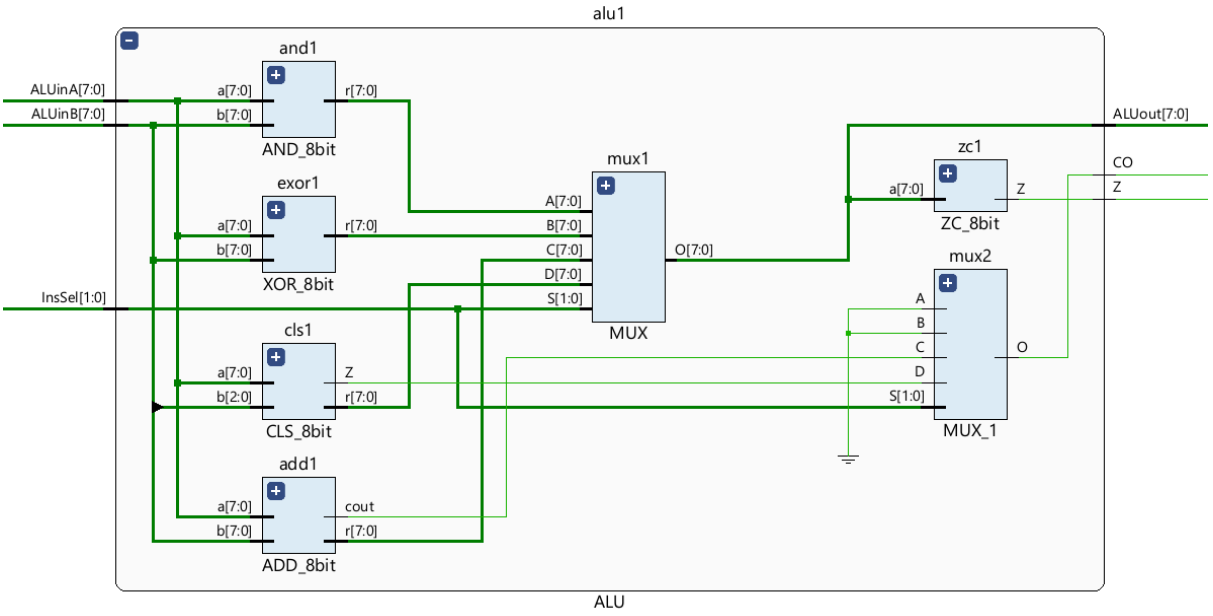
- 4 to 1 MUX (1 - bit)

```
    module MUX_1(
    input  A,B,C,D,
    //input [3:0] D,
    input [1:0] S,
    output O
    );
    wire w0,w1,w2,w3;
    assign w0=A&~S[0]&~S[1];
    assign w1=B&S[0]&~S[1];
    assign w2=C&~S[0]&S[1];
    assign w3=D&S[0]&S[1];
    assign O= w0|w1|w2|w3;
endmodule
```

- **ALU - MODULE**

```
module ALU(
input [7:0] ALUinA, ALUinB,
input [1:0] InsSel,
output [7:0] ALUout,
output CO,Z
);
wire [7:0] r0,r1,r3;
wire [7:0] r2;
wire carry_out,zero;

AND_8bit and1(.a(ALUinA), .b(ALUinB), .r(r0));
XOR_8bit exor1(.a(ALUinA), .b(ALUinB), .r(r1));
ADD_8bit add1(.a(ALUinA), .b(ALUinB), .r(r2), .cout(carry_out));
CLS_8bit cls1(.a(ALUinA), .b(ALUinB), .r(r3), .Z(zero));

MUX mux1(.A(r0), .B(r1), .C(r2), .D(r3), .O(ALUout), .S(InsSel));
MUX_1 mux2(.A(0), .B(0), .C(carry_out), .D(zero), .O(CO), .S(InsSel));

ZC_8bit zc1(.a(ALUout), .Z(Z));

endmodule
```

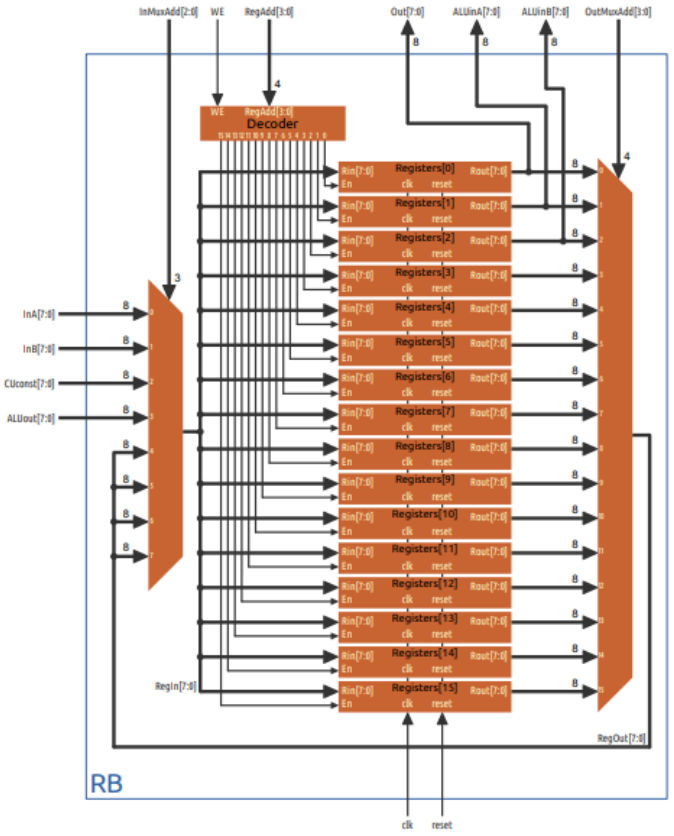
- **RTL SCHEMA of ALU**



Register Block

Register block consists of 16 registers, each utilizing flip-flops with enable and reset bits. In this setup, based on the InMUXAdd select bits, input A, input B, the Carry Out from the ALU module, the result of the operation from the ALU unit, or data from any selected register can be written. The address to write is determined by the RegAdd information after entering the RegAdd decoder. If WE (Write Enable) is 1, the enable bit of the specified register is set to 1, and the desired data is written to the register.

When writing data from one register to another, the OutMuxAdd select bits are used to determine from which register to write.



- Decoder

```
module decoder_4to16 (
  input [3:0] address,
  input write_enable,
  output reg [15:0] select_output
);

always @(address or write_enable) begin
  if (write_enable) begin
    case (address)
      4'b0000: select_output = 16'b0000000000000001;
      4'b0001: select_output = 16'b0000000000000010;
      4'b0010: select_output = 16'b0000000000000100;
      4'b0011: select_output = 16'b0000000000001000;
      4'b0100: select_output = 16'b0000000000010000;
      4'b0101: select_output = 16'b0000000000100000;
      4'b0110: select_output = 16'b0000000001000000;
      4'b0111: select_output = 16'b0000000010000000;
```

```
      4'b1000: select_output = 16'b0000000100000000;
      4'b1001: select_output = 16'b0000001000000000;
      4'b1010: select_output = 16'b0000010000000000;
      4'b1011: select_output = 16'b0000100000000000;
      4'b1100: select_output = 16'b0001000000000000;
      4'b1101: select_output = 16'b0010000000000000;
      4'b1110: select_output = 16'b0100000000000000;
      4'b1111: select_output = 16'b1000000000000000;
      default: select_output = 16'b0000000000000000;
    endcase
  end else begin
    select_output = 16'b0000000000000000;
  end
end

endmodule
```

- 16 to 1 MUX

```
module multiplexer_16to1 (
  input [3:0] select,
  input [7:0] data_in0, data_in1, data_in2, data_in3, data_in4,
data_in5, data_in6, data_in7, data_in8, data_in9, data_in10,
data_in11,data_in12,data_in13,data_in14,data_in15,
  output reg [7:0] out
);

always @(*) begin
  case (select)
    4'b0000: out = data_in0;
    4'b0001: out = data_in1;
    4'b0010: out = data_in2;
    4'b0011: out = data_in3;
    4'b0100: out = data_in4;
    4'b0101: out = data_in5;
    4'b0110: out = data_in6;
    4'b0111: out = data_in7;
    4'b1000: out = data_in8;
    4'b1001: out = data_in9;
    4'b1010: out = data_in10;
    4'b1011: out = data_in11;
    4'b1100: out = data_in12;
    4'b1101: out = data_in13;
```

```
    4'b1110: out = data_in14;
    4'b1111: out = data_in15;
    default: out = 8'b00000000; // Varsayılan durum
  endcase
end

endmodule
```

- 8 to 1 MUX

```
module MUX_2 (
  input [2:0] select,
  input [7:0] A,B,C,D,E,F,G,H,
  output reg [7:0] out
);

always @(*) begin
  case (select)
    3'b000: out = A;
    3'b001: out = B;
    3'b010: out = C;
    3'b011: out = D;
    3'b100: out = E;
    3'b101: out = F;
    3'b110: out = G;
    3'b111: out = H;
    default: out = 8'b00000000; // Varsayılan durum
  endcase
end

endmodule
```

- Register

```
module register (
    input clk, reset, En,
    input [7:0] Rin,
    output reg [7:0] Rout
);

    always @(posedge clk or posedge reset) begin
        if (reset)
```

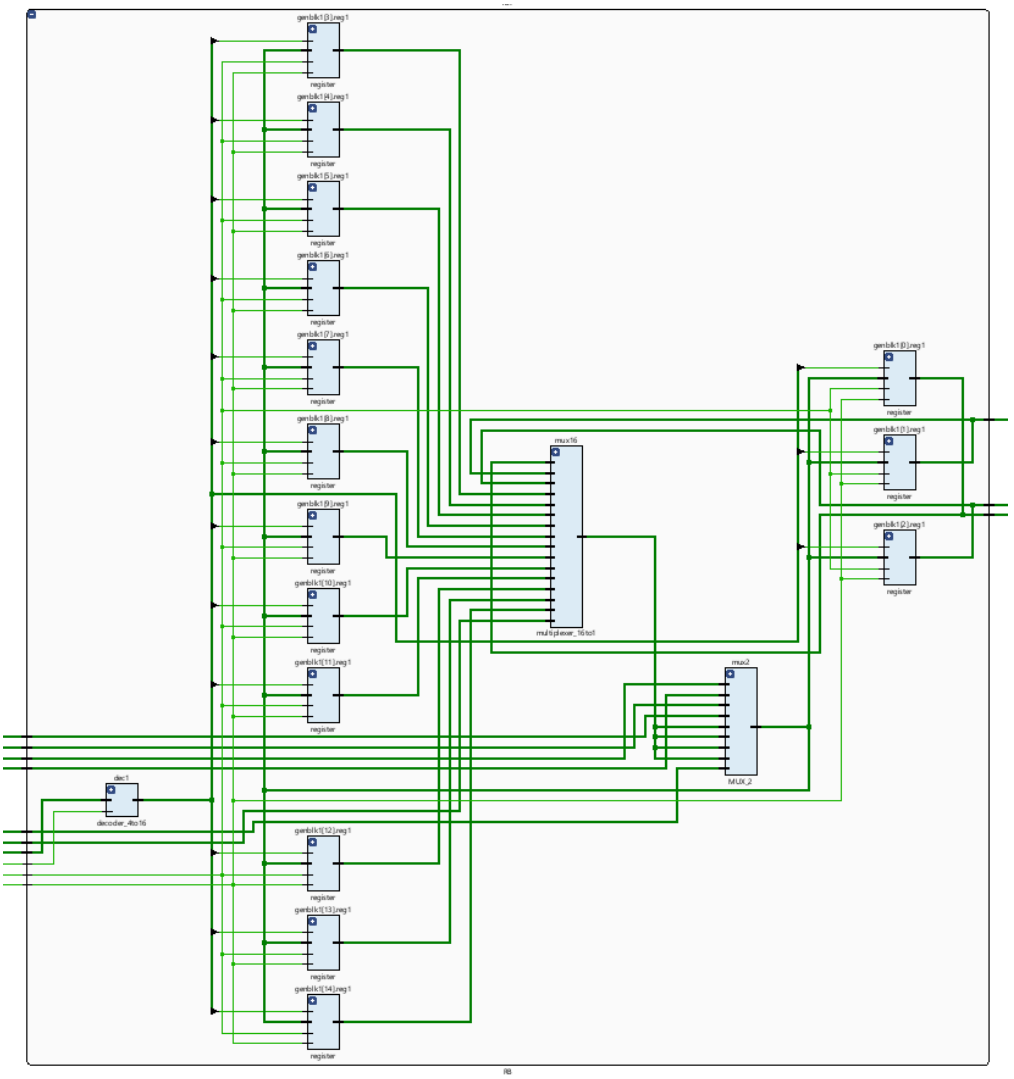
```
            Rout <= 0;
        else if (En)
            Rout <= Rin;
    end

endmodule
```

- **RB - MODULE**

```
module RB(
input [7:0] InA, InB,CUconst, ALUout,
input  [2:0] InMuxAdd,
input WE,clk,rst,
input [3:0] RegAdd,OutMuxAdd,
output [7:0] Out,ALUinA, ALUinB
);
wire [7:0] regout,regin;
wire [15:0] dec_out;
wire [7:0] register [15:0];
wire data;

MUX_2 mux2(.select(InMuxAdd), .A(InA), .B(InB), .C(CUconst),
.D(ALUout), .E(regout), .F(regout),.G(regout),.H(regout), .out(regin));
decoder_4to16 dec1(.address(RegAdd), .write_enable(WE),
.select_output(dec_out));

generate
genvar i;

for(i=0;i<15;i=i+1) begin

register reg1(.clk(clk), .reset(rst), .En(dec_out[i]), .Rin(regin),
.Rout(register[i]));

end
endgenerate

assign Out=register[0] ;
assign  ALUinA= register[1];
assign ALUinB= register[2];

multiplexer_16to1 mux16(.data_in0(register[0]),
```

```
                        .data_in1(register[1]),
                        .data_in2(register[2]),
                        .data_in3(register[3]),
                        .data_in4(register[4]),
                        .data_in5(register[5]),
                        .data_in6(register[6]),
                        .data_in7(register[7]),
                        .data_in8(register[8]),
                        .data_in9(register[9]),
                        .data_in10(register[10]),
                        .data_in11(register[11]),
                        .data_in12(register[12]),
                        .data_in13(register[13]),
                        .data_in14(register[14]),
                        .data_in15(register[15]),

.select(OutMuxAdd), .out(regout));

endmodule
```

- **RTL SCHEMA of RB**



Control Unit

The control unit of a Moore-type machine consists of 10 states, ranging from 0 to 9, implemented using a case statement. When a reset signal is received and in the last state, the busy bit is set to 0, while in other states, the busy bit remains 1. Additionally, the transition from state 0 to state 1 is not allowed unless the start bit is received. The operations performed in each state are as follows:

- State 0: The value of 'b' is selected as InMuxAdd 1, and, accordingly, it is placed in the 1st register after proper configuration in the decoder.

- State 1: The value of 'b' is selected as InMuxAdd 2, and, accordingly, it is placed in the 2nd register after proper configuration in the decoder.

- State 2: The ALU InsSel bit is set to 2, and the 'b' values in register 1 and register 2 are added. Subsequently, with InMuxAdd 3 selected, the result is stored in the 2nd address, obtaining the 2B value in the 2nd register.

- State 3: The 'b' values in register 1 and register 2 are added, and the result is stored in register 2, resulting in the value 3B in register 2.

- State 4: CuConsta is set to 11111111, and with InMuxAdd 2 selected, the necessary RegAdd is chosen to write 11111111 into the 1st register.

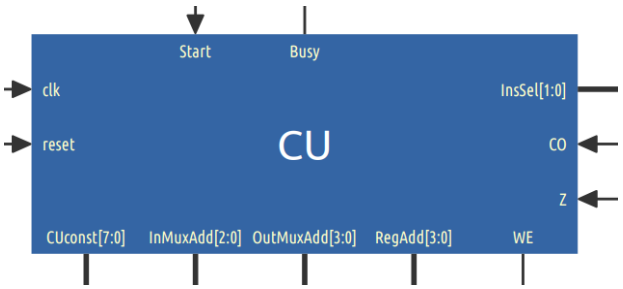
- State 5: InsSel is set to 01, and the bitwise XOR operation is performed between 8'b11111111 and 3B. The one's complement of the result is then written to the 1st register.

- State 6: CuConsta is set to 00000001, and with InMuxAdd 2 selected, the necessary RegAdd is chosen to write 00000001 into the 2nd register.

- State 7: The ALU InsSel bit is set to 2, and the 'b' values in register 1 and register 2 are added. Subsequently, with InMuxAdd 3 selected, the result is stored in the 2nd address, obtaining the -3B value in the 2nd register.

- State 8: The value of 'A' is selected as InMuxAdd 0, and, accordingly, it is placed in the 2nd register after proper configuration in the decoder.

- State 9: The InsSel bits are set to 2, and the 'A' and -3B values are added. The result is then written to the 0th register.



- **Control Unit - Module**

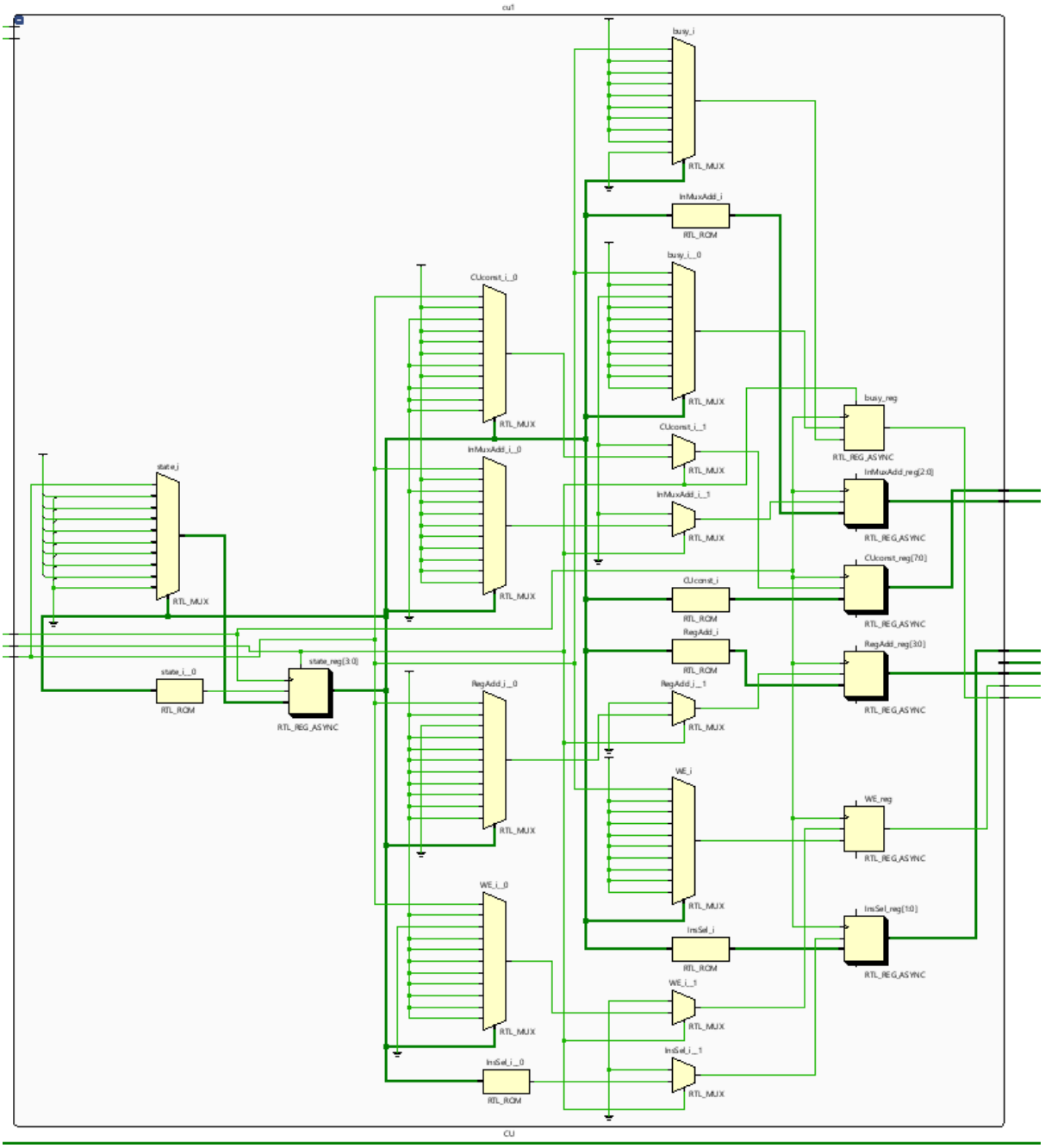
```
module CU(
input clk,reset,start,CO,Z,
output reg busy,WE,
output reg [1:0] InsSel,
output reg [2:0] InMuxAdd,
output reg [3:0] RegAdd, OutMuxAdd,
output reg [7:0] CUconst
```

```
    );
reg [3:0] state;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        state<=4'b0;
        busy<=1'b0;
        end
    else begin
    case (state)

         4'b0000: begin
                 if (start) begin
                 state <= 4'b0001;
                 InMuxAdd <= 3'b1;
                 WE<=1;
                 RegAdd<=4'b1;
                 CUconst <= 0;
                 busy<=1'b1;
                 end
                 else state<=4'b0000;
                 end

         4'b0001: begin
                 state <= 4'b0010;
                 InMuxAdd <= 3'b1;
                 WE<=1;
                 RegAdd<=4'b0010;
                 CUconst <= 0;
                 busy<=1'b1;
                 end

         4'b0010: begin
                 state <= 4'b0011;
                 InsSel<=2'b10;
                 InMuxAdd <= 3'b011;
                 WE<=1;
                 RegAdd<=4'b0010;
                 CUconst <= 0;
                 busy<=1'b1;
                 end

         4'b0011: begin
```

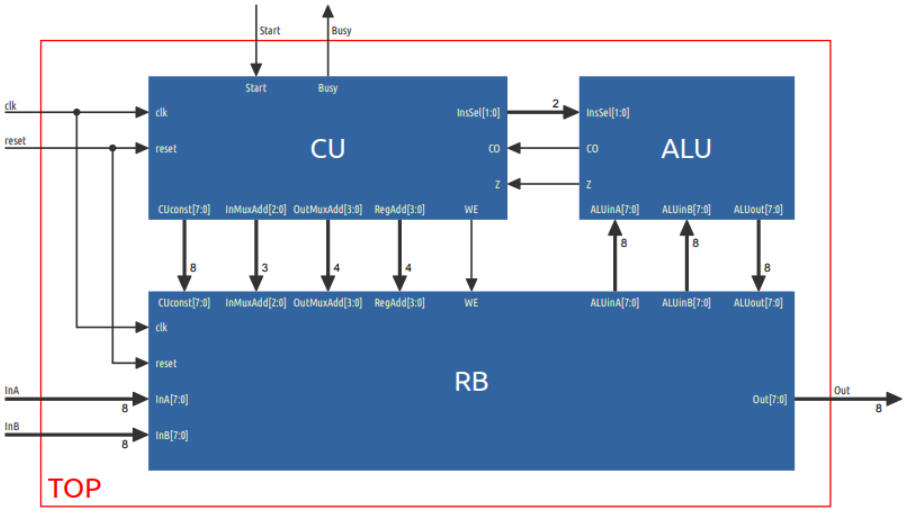
```
                state <= 4'b0100;
                InsSel<=2'b10;
                InMuxAdd <= 3'b011;
                WE<=1;
                RegAdd<=4'b0010;
                CUconst <= 0;
                busy<=1'b1;
                end

        4'b0100: begin
                state <= 4'b0101;
                CUconst <=8'b1111_1111;
                WE<=1;
                RegAdd<=4'b0001;
                InMuxAdd <= 3'b010;
                busy<=1'b1;
                end

        4'b0101: begin
                state <= 4'b0110;
                WE<=1;
                RegAdd<=4'b0001;
                InsSel<=2'b01;
                InMuxAdd <= 3'b011;
                busy<=1'b1;
                end

        4'b0110: begin
                state <= 4'b0111;
                WE<=1;
                RegAdd<=4'b0010;
                CUconst<=8'b0000_0001;
                InMuxAdd <= 3'b010;
                busy<=1'b1;
                end

        4'b0111: begin
                state <= 4'b1000;
                WE<=1;
                RegAdd<=4'b0001;
                InMuxAdd <= 3'b011;
                InsSel<=2'b10;
                busy<=1'b1;
```

```
                end

        4'b1000: begin
                state <= 4'b1001;
                WE<=1;
                RegAdd<=4'b0010;
                InMuxAdd <= 3'b000;
                busy<=1'b1;
                end

         4'b1001: begin
                state <= 4'b0000;
                WE<=1;
                RegAdd<=4'b0000;
                InMuxAdd <= 3'b011;
                InsSel<= 2'b10;
                busy<=1'b0;
                end


endcase
end
 end
endmodule
```

- **RTL SCHEMA of CU**



Top Module



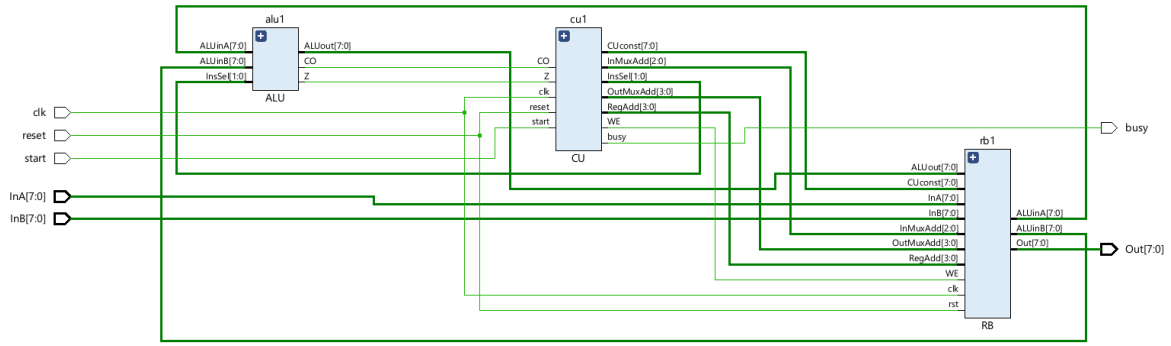
- **Top Module- Module**

In Top Module the submodules connected like in the figure TOP

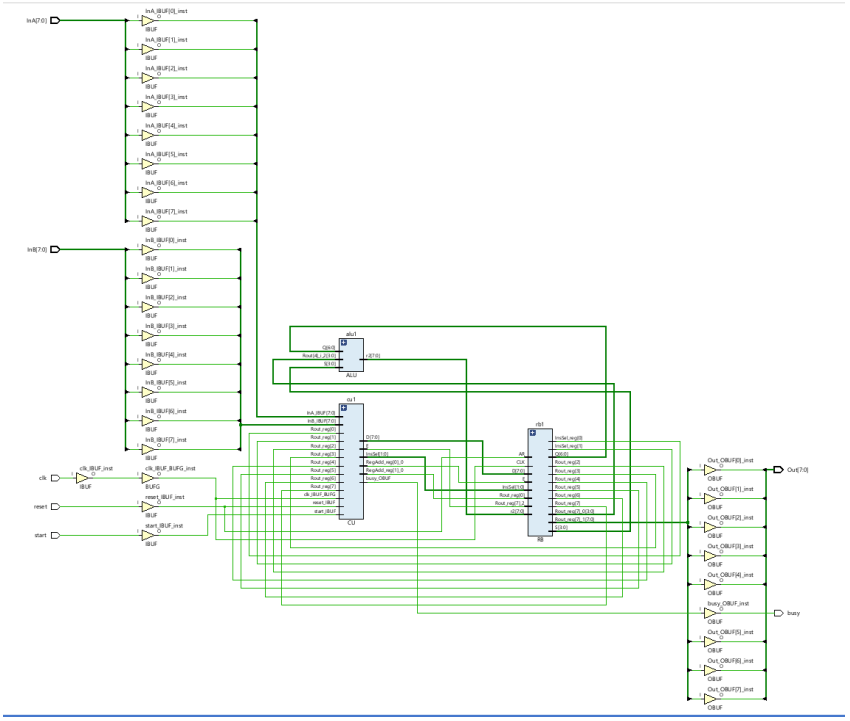
```
module TOP(
input clk,reset,start,
input [7:0] InA,InB,
output  busy,
output [7:0] Out
    );
wire [1:0] InsSel;
wire CO,Z,WE;
wire [7:0] CUconst;
wire [2:0] InMuxAdd;
wire [3:0] OutMuxAdd, RegAdd;

    CU cu1(.clk(clk), .reset(reset), .start(start), .busy(busy),
.InsSel(InsSel), .CO(CO),
    .Z(Z), .CUconst(CUconst), .InMuxAdd(InMuxAdd),
.OutMuxAdd(OutMuxAdd), .RegAdd(RegAdd),
    .WE(WE));

wire [7:0] ALUinA, ALUinB, ALUout;
```

```
    ALU alu1(.InsSel(InsSel), .CO(CO), .Z(Z), .ALUinA(ALUinA),
.ALUinB(ALUinB),
    .ALUout(ALUout));

    RB rb1(.clk(clk), .rst(reset), .InA(InA), .InB(InB),
.CUconst(CUconst),
    .InMuxAdd(InMuxAdd), .OutMuxAdd(OutMuxAdd), .RegAdd(RegAdd),
.WE(WE),
    .ALUinA(ALUinA), .ALUinB(ALUinB), .ALUout(ALUout), .Out(Out));

endmodule
```

- **RTL SCHEMA of the TOP MODULE**



- **TECHNOLOGY SCHEMA of the TOP MODULE**



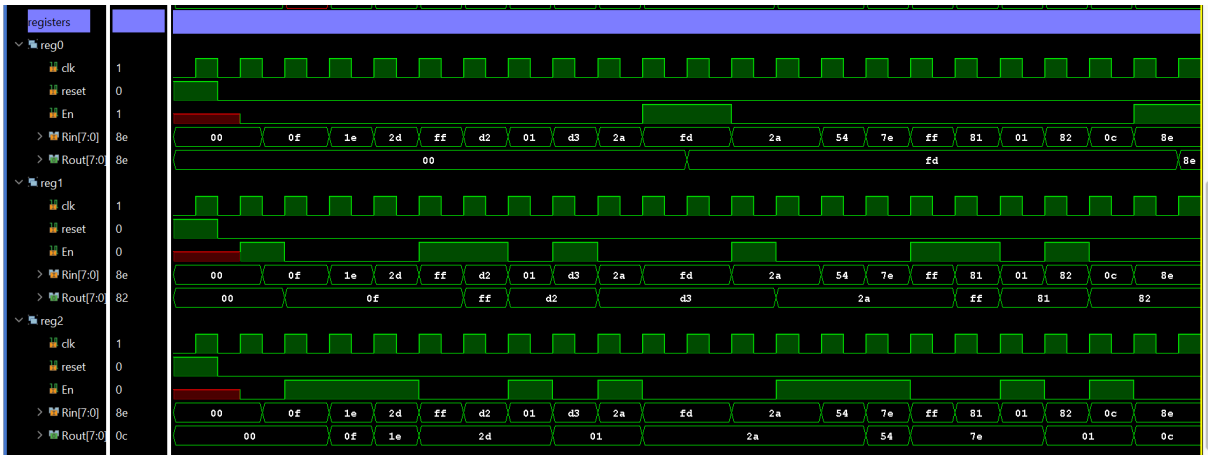
Testbench - Simulation

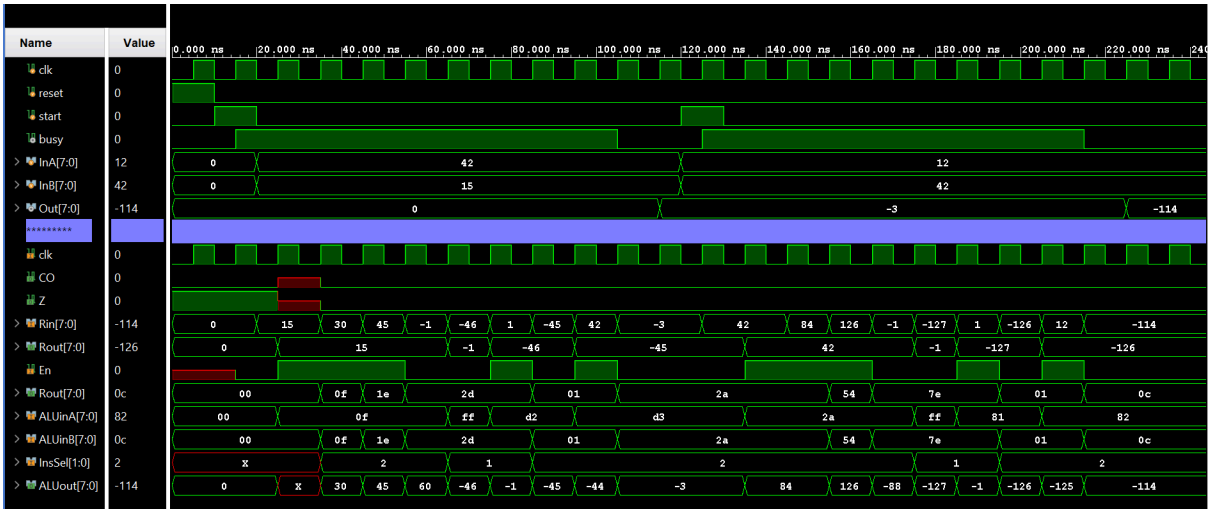
```
module TOP_tb;

  reg clk, reset, start;
  reg [7:0] InA, InB;
  wire busy;
  wire [7:0] Out;

  // Instantiate the TOP module
  TOP top_inst(
    .clk(clk),
    .reset(reset),
    .start(start),
    .InA(InA),
    .InB(InB),
    .busy(busy),
    .Out(Out)
  );

  // Clock generation
```

```
  always begin
    #5 clk = ~clk;
  end

  // Initial block
  initial begin
    // Initialize inputs
    clk = 0;
    reset = 0;
    start = 0;
    InA = 8'b00000000;
    InB = 8'b00000000;

    // Apply reset
    reset = 1;
    #10 reset = 0;

    // Apply start
    start = 1;
    #10 start = 0;

    // Test case 1
    InA = 8'b00101010;
    InB = 8'b00001111;
    #100;

    // Test case 2
    InA = 8'b00001100;
    InB = 8'b00101010;
    start=1'b1;
    #10 start=1'b0;
    #100;


    // Add more test cases as needed

    // End simulation
    $stop;
  end

endmodule
```

In this simulation, A was chosen as 42 and B as 15, resulting in an output of -3 observed after 9 clock cycles. Similarly, later on, A was set to 12 and B to 42, with the output of -114 being observed in the output after some clock cycles. It can be observed that the operations performed in each clock cycle are consistent with the states. The sequential changes in the registers after each clock cycle are as follows:

- 1. clock : register 1 = B
- 2. clock : register 2 = B
- 3. clock : register 2 = 2B
- 4. clock : register 2 = 3B
- 5. clock : register1 = 8'b1111_1111
- 6. clock : register 1 = 1's Complement of 3B
- 7. clock : register 2 = 1
- 8. clock : register 1 = -3B
- 9. clock : register 2 = A
- 10. clock : register 0 = A - 3B





```
// Test case 3
    InA = 8'b00101010;
    InB = 8'b10101111;
    #100;

    // Test case 4
    InA = 8'b00000011;
    InB = 8'b00000001;
    start=1'b1;
    #10 start=1'b0;
    #100;
```

In the first case of this simulation, only overflow was detected, which can be observed by the Co bit being set to 1. In the subsequent case, the focus was on the zero output, and it was observed that the result was 1 for the 3 - 1 * 3 case.