8-bit Arithmetic Logic Unit Simulation

2nd Semester

Digital Computers

Coordinator teacher: Bozdog Alexandru

Team: Bociort Norica-Diana
Jinar Denis-Raul
Plesa Diana-Ioana
Popa Maria-Denisa

Contents

1. Overview

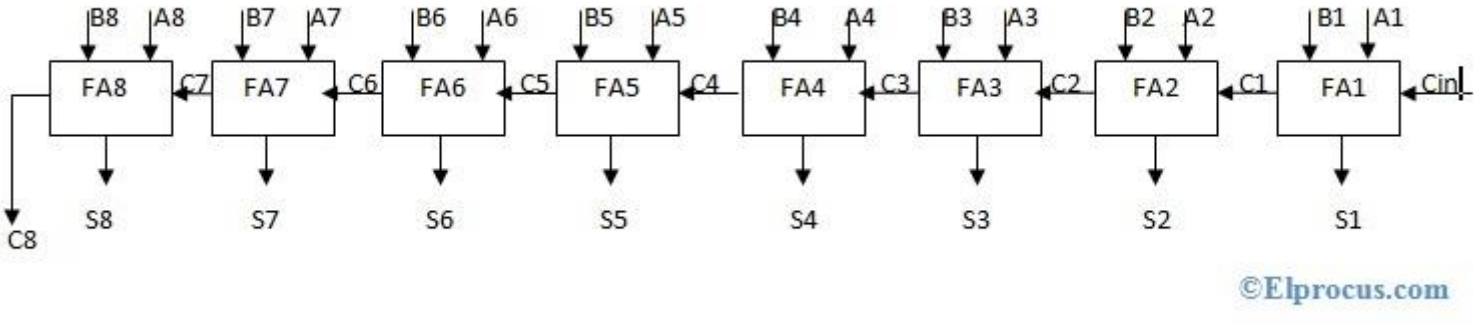
This project involves the structural design and simulation of an 8-bit Arithmetic Logic Unit (ALU) using a hardware description language (Verilog). The ALU performs core arithmetic operations: addition, subtraction, multiplication and division. The goal is to understand and demonstrate how arithmetic functions are built and controlled at hardware level.

2. Project Phases

2.1. Design Phase

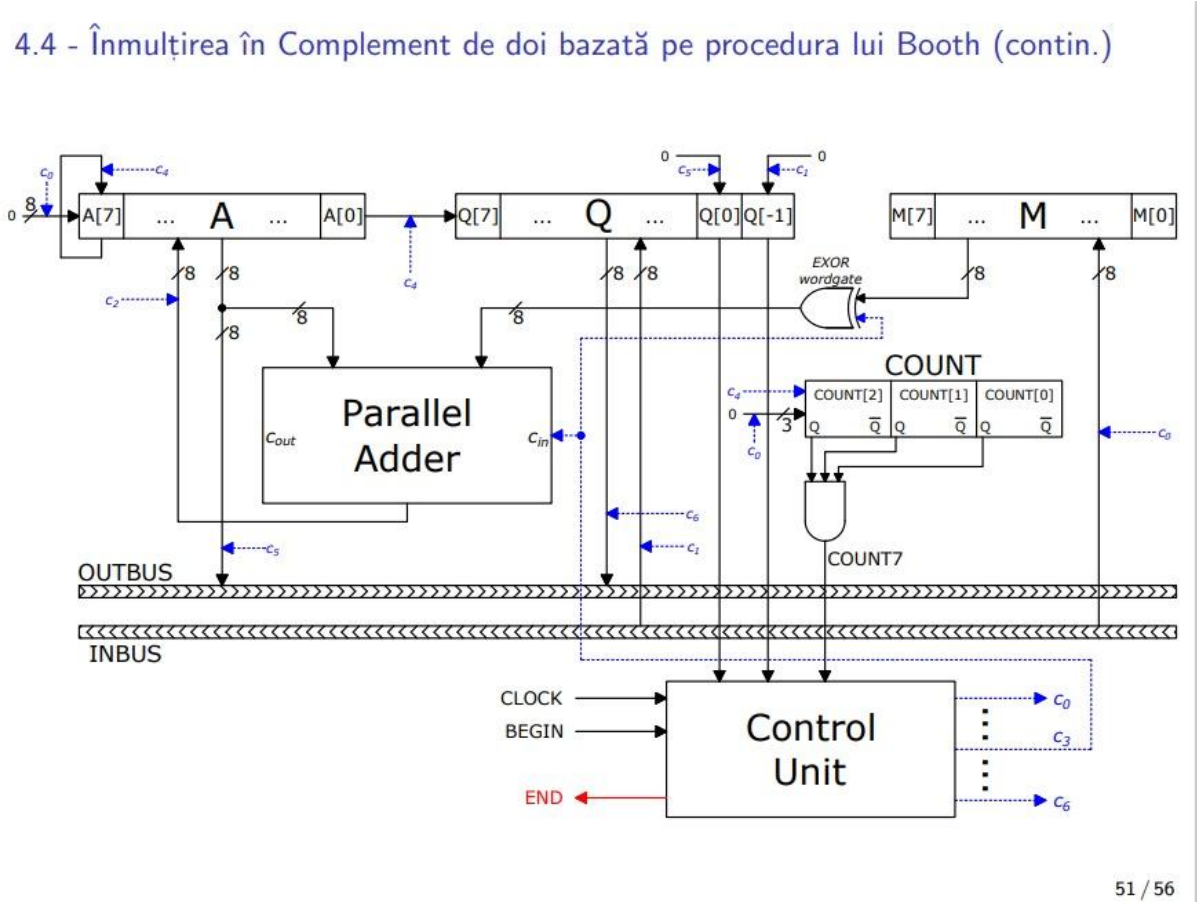
The components' modules that we used were the following: ALU, Control Unit, Flip-Flop Data, Flip-Flop Toggle, Flip-Flop Set-Reset, FAC and Ripple Carry Adder, Register (for A, Q, M), Counter, Decoder one-hot, Modulo 5 Sequence Counter, MUX.

For the 8-bit **addition** and **subtraction** operations in our ALU, we used a Ripple Carry Adder (RCA) due to its simplicity and ease of implementation. Although it is not the fastest adder, it is well-suited for small bit-widths like 8 bits, where propagation delay is minimal.



To implement subtraction, we reused the same RCA by applying 2's complement to the subtrahend, allowing us to perform subtraction as addition of a negative number, which simplifies the hardware design.

For **multiplication**, we implemented Booth's Radix-2 Algorithm. The algorithm examines two bits at a time (Q[0] and Q[-1]) to decide whether to add, subtract or do nothing with the multiplicand before shifting right. This technique reduces the number of required arithmetic operations by compressing sequences of 1s in the multiplier.



For **division**, we chose the Restoring Division Algorithm, a step-based method suitable for unsigned or signed binary division. The algorithm works by performing a left shift followed by a subtraction of the divisor. If the result is negative, the original value is restored, and a 0 is recorded in the quotient. If the result is positive, a 1 is recorded instead. It was well-suited for our 8-bit Verilog implementation and allowed us to maintain clarity in both logic and simulation.

For the control unit logic, we also used a Modulo5 Sequence Counter in order to count through the cycles. We have implemented controls for 5 cycles in total, so we used a 5-bit variable.

```
module Modulo5_Sequence_Counter(
   input clk, reset, begin_sig, end_sig,
   output [4:0] cycle
);

    wire q;
    wire [2:0] count;

    SR_FF sr_ff (.s(begin_sig), .r(end_sig), .clk(clk), .q(q));
    Modulo5_Counter counter (.clk(clk), .reset(reset), .count_up(q), .count(count));
    Decoder_one_hot decoder (.count(count), .cycle(cycle));

endmodule
```

To implement this sequence counter, we designed:

- a Set-Reset Flip Flop - starts the count whenever it receives the begin signal and stops the count for an end signal, the output is q
- a Modulo5 Counter - counts the cycles, from 0 to 4, and is activated by q
- a one-hot Decoder – it transforms the count value in an output signal, cycle that will be one-hot coded

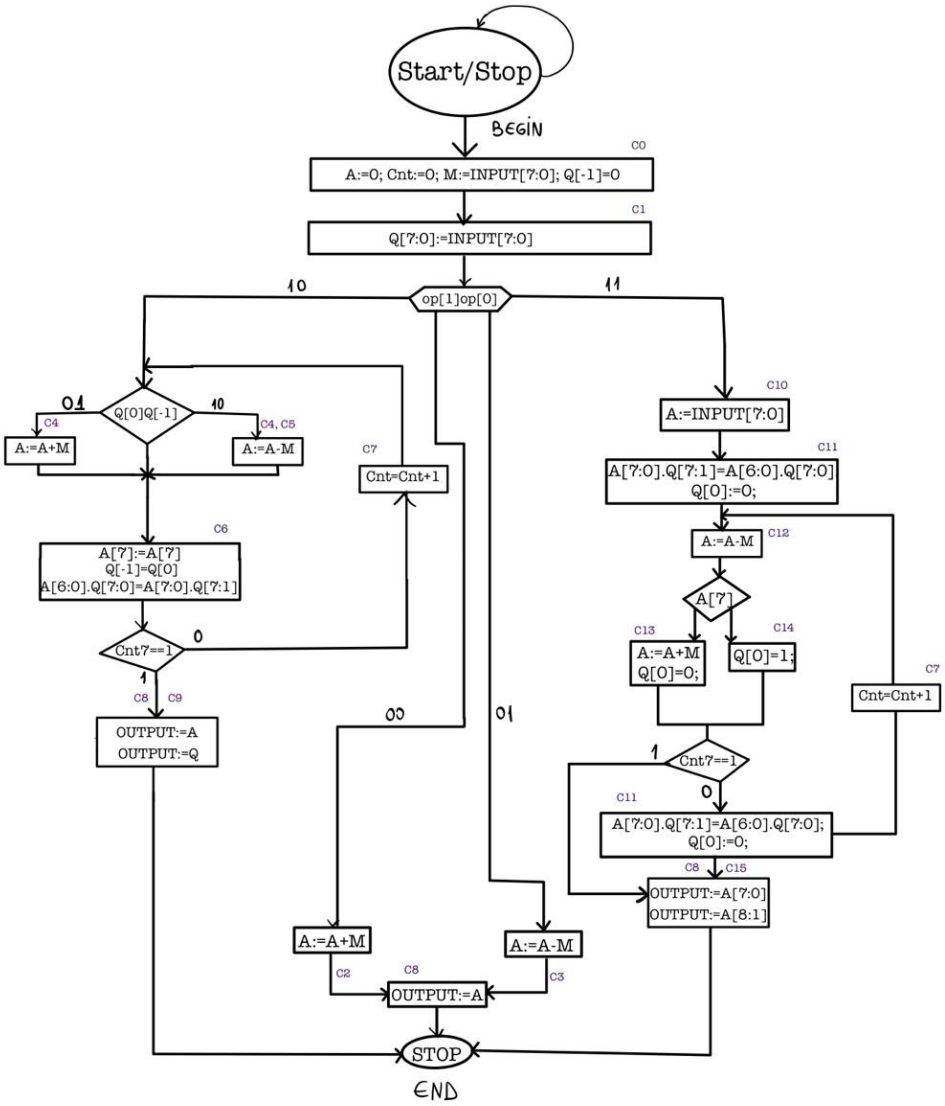
Example:

count=010 (=2) => cycle=00100 (the bit 2 will be 1 and the rest 0 so we know cycle 2 is currently active)

The arithmetic operations are selected by a 2-bit input that goes into the Control Unit.

SEL	OPERATION
00	Addition
01	Subtraction
10	Multiplication
11	Division

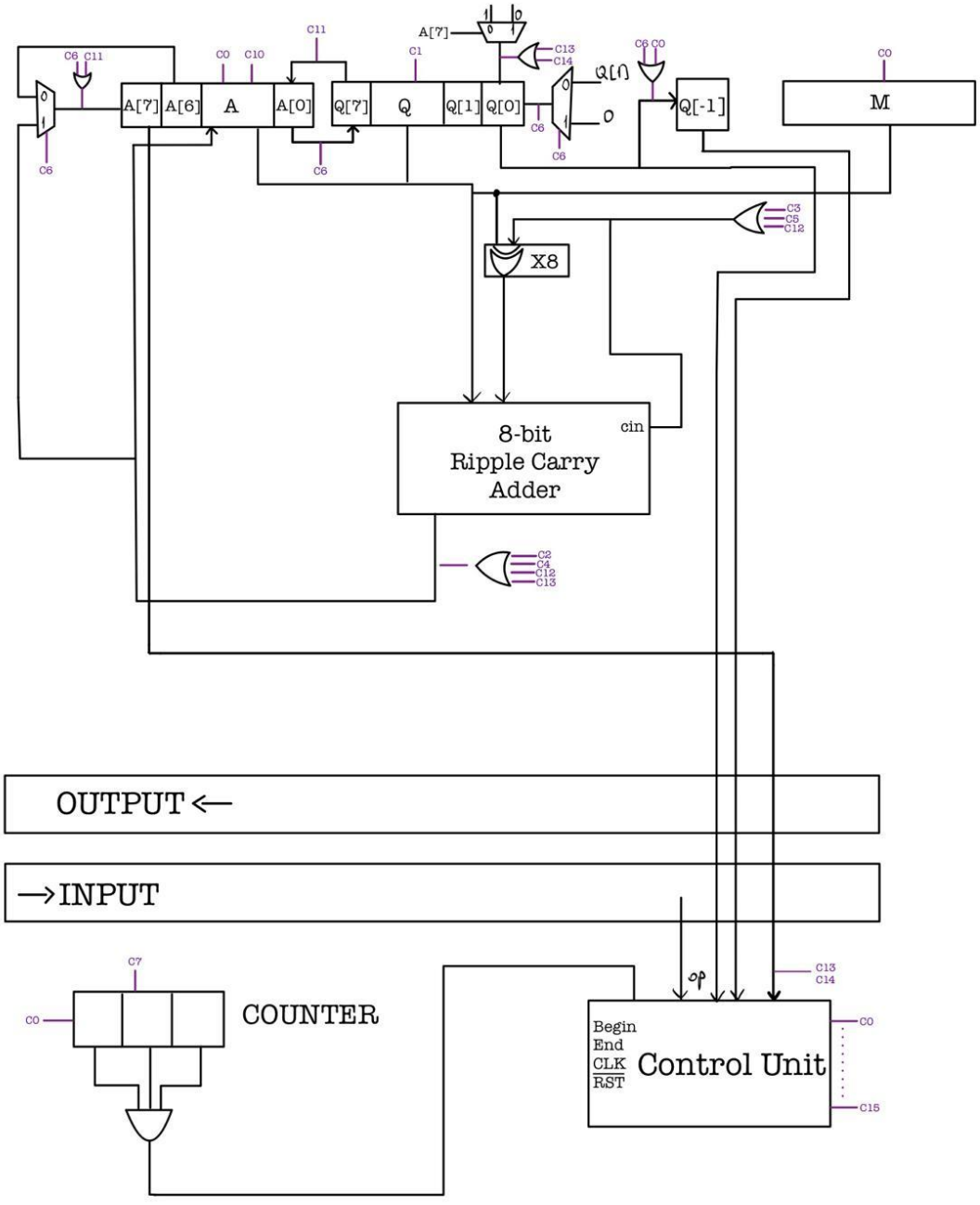
Finite State Diagram



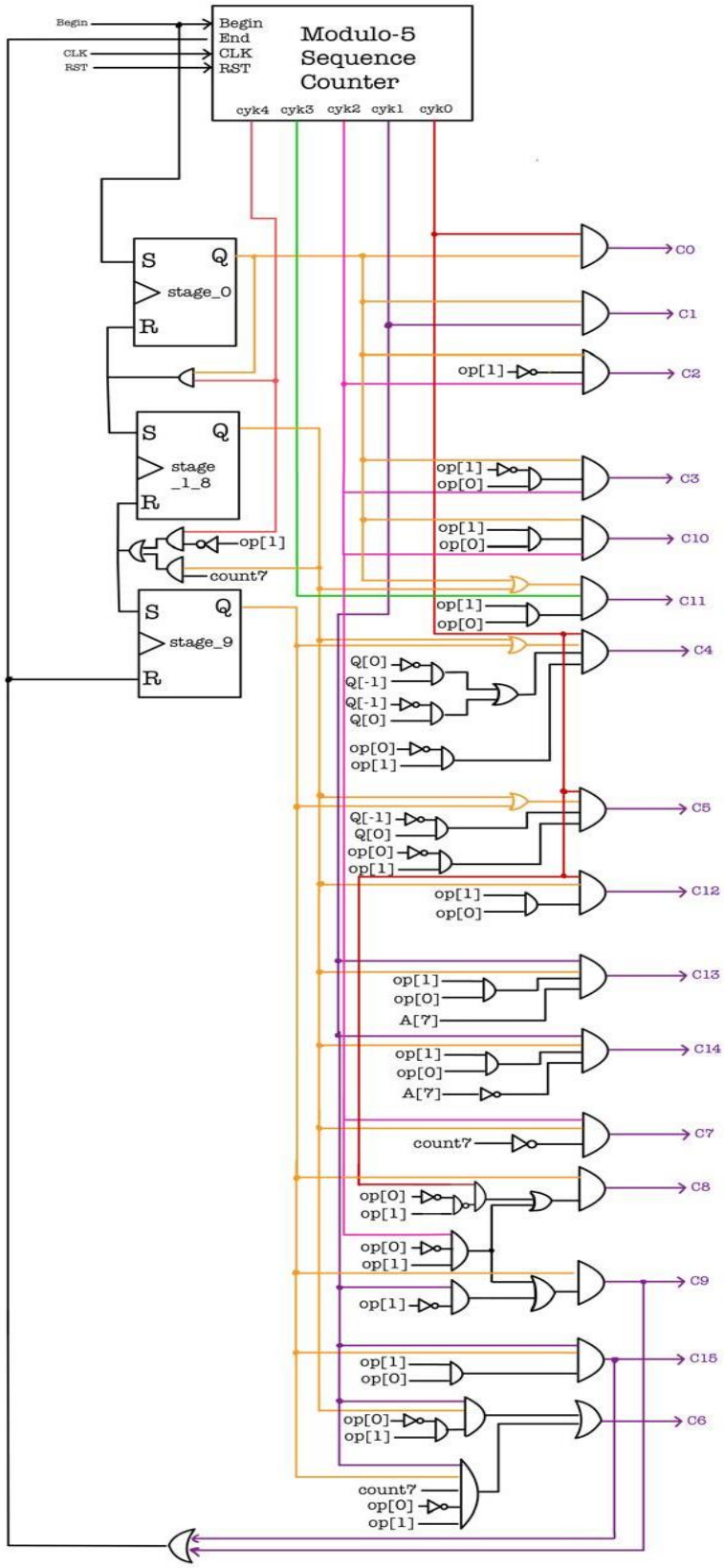
The diagram illustrates the ALU's overall operation, outlining the steps involved in processing data and selecting the appropriate operation based on the provided operation code.

The **control logic** decodes the operation code to determine which operation to execute: addition, subtraction, multiplication, division. Each operation progresses through a defined sequence of stages, with conditional branches directing the flow of execution.

ALU Hardware Design



Control Unit



2.2 Implementation Phase

Code (Control Unit and ALU)

```
module Control_Unit(

   input clk, reset, begin_sig, A7, Q0, Q_1, count7,
   input [1:0] op,
   output reg end_sig,
   output [17:0] c
);

   wire [4:0]cycle;
   wire cyk0, cyk1, cyk2, cyk3, cyk4;
   assign cyk0 = cycle[0];
   assign cyk1 = cycle[1];
   assign cyk2 = cycle[2];
   assign cyk3 = cycle[3];
   assign cyk4 = cycle[4];

   wire stg0, stg1_8, stg9; //stages
   wire reset_stage_0, reset_stage_1_8;
   assign reset_stage_0 = cyk4 & stg0; //suntem in stage0 si se ajunge in ciclul 4=> se da reset la stage 0 (reset se pune pe 1)
   assign reset_stage_1_8 = (stg1_8 & count7 & cyk4) | ((~op[1]) & cyk4);
//suntem in stage 1-8 si count7 este 1 si suntem in ciclul 4 SAU avem op de adunare/scadere si suntem in ciclul 4 => se da reset la stage 1-8

   Modulo5_Sequence_Counter sc(.clk(clk), .reset(reset), .begin_sig(begin_sig), .end_sig(end_sig), .cycle(cycle));

//in cycle vom primi ciclul activ codificat de genul 00100, asadar cyk2 va fi activ, iar restul inactive si tot asa pana se trece prin toate
//si primeste semnalul de eng_sig pentru a opri numaratoarea din sequence counter

   SR_FF stage_0 (.s(begin_sig), .r(reset_stage_0), .clk(clk), .q(stg0)); //stg0=1 cand primim begin signal
   SR_FF stage_1_8 (.s(reset_stage_0), .r(reset_stage_1_8), .clk(clk), .q(stg1_8)); //daca s-a trecut prin stage0 (<=> reset stage0 e activ), stg1_8=1
   SR_FF stage_9 (.s(reset_stage_1_8), .r(end_sig), .clk(clk), .q(stg9)); //daca s-a trecut prin stage 1-8 (reset stage1-8 e activ), stg9=1

   assign c[0] = stg0 & cyk0;
   assign c[1] = stg0 & cyk1;
   assign c[2] = stg0 & cyk2 & (~op[1]);        // se activeaza la addition, subtraction
   assign c[3] = stg0 & cyk2 & (~op[1] & op[0]);        // subtraction
   assign c[10] = stg0 & cyk2 & (op[1] & op[0]);        // division
   assign c[11] = (stg0 | stg1_8) & cyk3 & (op[1] & op[0]);     // division
   assign c[4] = (stg1_8 | stg9) & cyk0 & (~Q0 & Q_1 | Q0 & ~Q_1) & (op[1] & ~op[0]);   // 01 sau 10 la multiplication
   assign c[5] = (stg1_8 | stg9) & cyk0 & (Q0 & ~Q_1) & (op[1] & ~op[0]);       // 10 la multiplication
   assign c[12] = stg1_8 & cyk0 & (op[1] & op[0]);      // division
   assign c[13] = stg1_8 & cyk1 & A7 & (op[1] & op[0]); // division
   assign c[14] = stg1_8 & cyk1 & ~A7 & (op[1] & op[0]);        // division
   assign c[7] = stg1_8 & cyk2 & ~count7;       // division, multiplication
   assign c[8] = stg9 & ( (cyk2 & op[1] & ~op[0]) | (cyk0 & ~(op[1] & ~op[0])) ); //load
   assign c[9] = stg9 & ( (cyk2 & op[1] & ~op[0]) | (cyk1 & ~op[1]) ); //load
   assign c[15] = stg9 & cyk1 & (op[1] & op[0]); //load la division
   assign c[6] = ( stg1_8 & cyk1 & (op[1] & ~op[0]) ) | ( stg9 & cyk1 & count7 & (op[1] & ~op[0]));     // multiplication

   always @(posedge clk) begin
      if (!reset)
        end_sig <= 1'b0;
      else
        end_sig <= c[9] | c[15];
```

```
module ALU(

   input clk,
   input resetn,
   input [7:0] X, Y,
   input [7:0] A_divide,
   input [1:0] op,
   input BEGIN,

   output [15:0] OUT,
   output wire END,
   output [7:0] Q,
   output [7:0] A,
   output [2:0]  count,
   output q_1,
   output [7:0] m,
   output [7:0] sum_out,
   output [17:0] control
);

   wire A_Din;
   wire Q_1;
   wire Q_Din;
   wire [7:0] sum;
   wire cout;
   wire [2:0] count7;
   wire [17:0] c;
   wire [7:0] M;

   reg [1:0] shift_A;
   reg [1:0] shift_Q;
   reg sum_in;

   assign count = count7;
   assign q_1 = Q_1;
   assign m = M;
   assign sum_out = sum;
   assign control = c;

   always @(*) begin
      sum_in = (c[2] | c[4] | c[12] | c[13]);
   shift_A = { c[0] | (c[11]) | sum_in | c[10], c[0] | c[6] | sum_in | c[10] };
   shift_Q = { c[0] | (c[11]), c[0] | c[6] };
end


MUX_4_to_1 A_Din_mux( //pentru shiftare, ce valoare va lua noul A[7] (daca e shiftare dreapta) sau A[0] (daca e shiftare stanga)
                     .in0(1'bx),
                     .in1(A[7]),
                     .in2(Q[7]),
                     .in3(1'bx),
                     .select(shift_A), //pentru 01 shiftare dreapta / pentru 10 shiftare stanga
                     .out(A_Din)
                     );

Register A_Register(
               .clk(clk),
               .resetn(resetn),
               .load_data((c[0]) ? {8{1'b0}} : (c[10] ? A_divide : sum)),
               .shift(shift_A),
               .load_Din(1'b0),
               .Din(A_Din),
               .Q(A)
               );

D_FF Q_1_FlipFlop(
                     .clk(clk),
                     .resetn(resetn),
                     .enable(c[0] | c[6]),
                     .D(c[0] ? 1'b0 : Q[0]),
                     .Q(Q_1)
                     );

MUX_4_to_1 Q_Din_mux(
                     .in0(1'bx),
                     .in1(A[0]),//01=shift dreapta
                     .in2(1'b0),//10=shift stanga
                     .in3(1'bx),
                     .select(shift_Q),
                     .out(Q_Din)
                     );

reg load_S;
```

```
  always @(*) begin
     load_S = c[13] | c[14]; //pentru division
  end


  Register Q_Register(
                .clk(clk),
                .resetn(resetn),
                .load_data(X),
                .shift(shift_Q),
                .load_Din(load_S),
                .Din(load_S ? ~A[7] : Q_Din),
                .Q(Q)
                );

  Register M_Register(
                .clk(clk),
                .resetn(resetn),
                .load_data(Y),
                .shift({ c[0], c[0] }),
                .load_Din(1'b0),
                .Din(1'b0),
                .Q(M)
                );

  RippleCarryAdder RCA(
                    .a(c[2] ? Q : A),
                    .b(M),
                    .cin(c[3] | c[12] | c[5]),
                    .enable(c[2] | c[4] | c[12] | c[13]),
                    .cout(cout),
                    .sum(sum)
                    );


  Counter counter(
                 .clk(clk),
                 .resetn(resetn | c[0]),
                 .count_up(c[7]),
                 .count(count7)
                 );
```

```
   Control_Unit CU(
                    .clk(clk),
                    .reset(resetn),
                    .begin_sig(BEGIN),
                    .Q0(Q[0]),
                    .Q_1(Q_1),
                    .A7(A[7]),
                    .count7(count7[0] & count7[1] & count7[2]),
                    .op(op),
                    .c(c),
                    .end_sig(END)
                    );

   reg [7:0] OUT_1;
   reg [7:0] OUT_2;

   always @(*) begin
      OUT_1 = c[8] ? A : OUT_1;
      OUT_2 = c[9] | c[15] ? Q[7:0] : Q[7:0];
   end

   assign OUT = { OUT_1, OUT_2 };
endmodule
```

Control Signals

ADDITION / SUBTRACTION

Operation codes: ADDITION (00), SUBTRACTION (01)
Stage 0:
• Cycle 0: c0 -> Load operand M
• Cycle 1: c1 -> Load operand Q
• Cycle 2: c2 -> Triggers addition, the result is stored in A
• Cycle 2: c3 -> Triggers subtraction, the result is stored in A
Stage 9:
• Cycle 0: c8 -> Load result A
• Cycle 1: c9 -> Load result Q
END signal

MULTIPLICATION (Booth Radix-2)

Operation codes: MULTIPLICATION (10)
Stage 0:
• Cycle 0: c0 -> Load operand M and initialize A, COUNT, Q[-1]=0
• Cycle 1: c1 -> Load operand Q
Stage 1 - 8:
• Cycle 0: c4 -> Triggers addition for Q[0]Q[-1] in 01, the result is stored in A (A=A+M)
• Cycle 0: c5 -> Additional signal for subtraction for Q[0]Q[-1] in 10, the result is stored in A (A=A-M)
• Cycle 1: c6 -> Triggers right shift
• Cycle 2: c7 -> COUNT is updated
Stage 9:
• Cycle 0: c4 -> Triggers addition for Q[0]Q[-1] in 01, the result is stored in A (A=A+M)
• Cycle 0: c5 -> Additional signal for subtraction for Q[0]Q[-1] in 10, the result is stored in A (A=A-M)
• Cycle 1: c6 -> Triggers right shift
• Cycle 2: c8 -> Load result A
• Cycle 2: c9 -> Load result Q
END signal

DIVISION (Restoring Division)

Operation codes: DIVISION (11)
Stage 0:
• Cycle 0: c0 -> Load operand M
• Cycle 1: c1 -> Load operand Q
• Cycle 2: c10 -> Load operand A (first half of the dividend)
• Cycle 3: c11 -> Triggers initial left shift
Stage 1 - 8:
• Cycle 0: c12 -> Triggers addition + subtraction
• Cycle 1: c13 -> Triggers addition when A[7] = 1, updates Q[0] = 0
• Cycle 1: c14 -> Updates Q[0] = 1
• Cycle 2: c7 -> COUNT is updated
• Cycle 3: c11 -> Triggers left shift
Stage 9:
• Cycle 0: c8 -> Load result A
• Cycle 0: c15 -> Load result Q
END signal

2.3 Testing Phase

For testing, in the testbench module, uncomment the chosen operation, the X and Y values for the respective operation and make sure the other ones are commented so they do not interfere with the result.
If you have chosen division you also must uncomment the A_divide value, which represents the first 8 bits for A.

```
initial begin
   $dumpfile("ALU_tb.vcd");
   $dumpvars(0, ALU_tb);

   // 31000 : 123 = 252 R 4
   //A_divide = 8'b01111001;

   // 4876 : 71 = 68 R 38
   A_divide = 8'b00010011;

   clk = 0;
   resetn = 0;
   #10
   resetn = 1;
   BEGIN = 1;
   #10
   BEGIN = 0;
```

```
   // 67 +- 16
   //X = 8'b01000011; //for add/sub
   //Y = 8'b00010000; //for add/sub

   // 85 +- 23
   //X = 8'b01010101; //for add/sub
   //Y = 8'b00010111; //for add/sub

   // 53 * 19 = 1007
   //X = 8'b00110101; //for multiplication
   //Y = 8'b00010011; //for multiplication

   // (-100) * (-3) = 300
   //X = 8'b10011100; //for multiplication
   //Y = 8'b11111101; //for multiplication

   // 31000 : 123 = 252 R 4
   //X = 8'b00011000; //for division
   //Y = 8'b01111011; //for division

   // 4876 : 71 = 68 R 48
   //X = 8'b00001100; //for division
   //Y = 8'b01000111; //for division

   //operation selection
   //op = 2'b00; // ADD
   //op = 2'b01; // SUB
   //op = 2'b10; // MUL
   //op = 2'b11; // DIV

   #1000;
   $finish;
end
```

Simulation

We have incorporated multiple variables into the testing display to clearly illustrate each step of the selected algorithm. The **COUNT** variable indicates the current iteration, **C_ACTIVE** shows the active control signal, and we also display the values of **A**, **Q**, **M**, and **Q[-1]** during multiplication. Different numbers were used for each operation to verify the code's accuracy across various inputs and to conform to register size constraints. We have also included waveform simulations for each operation.

- **DIVISION**

  X=4876
  Y=71

```
COUNT=000 | A=0000 0000 | Q=0000 0000 | M=0000 0000 | C_ACTIVE=
COUNT=000 | A=0000 0000 | Q=0000 0000 | M=0000 0000 | C_ACTIVE=                c0
COUNT=000 | A=0000 0000 | Q=0000 1100 | M=0100 0111 | C_ACTIVE=                c1
COUNT=000 | A=0000 0000 | Q=0000 1100 | M=0100 0111 | C_ACTIVE=               c10
COUNT=000 | A=0001 0011 | Q=0000 1100 | M=0100 0111 | C_ACTIVE=               c11
COUNT=000 | A=0010 0110 | Q=0001 1000 | M=0100 0111 | C_ACTIVE=
COUNT=000 | A=0010 0110 | Q=0001 1000 | M=0100 0111 | C_ACTIVE=               c12
COUNT=000 | A=1101 1111 | Q=0001 1000 | M=0100 0111 | C_ACTIVE=               c13
COUNT=000 | A=0010 0110 | Q=0001 1000 | M=0100 0111 | C_ACTIVE=                c7

COUNT=001 | A=0010 0110 | Q=0001 1000 | M=0100 0111 | C_ACTIVE=               c11
COUNT=001 | A=0100 1100 | Q=0011 0000 | M=0100 0111 | C_ACTIVE=
COUNT=001 | A=0100 1100 | Q=0011 0000 | M=0100 0111 | C_ACTIVE=               c12
COUNT=001 | A=0000 0101 | Q=0011 0000 | M=0100 0111 | C_ACTIVE=               c14
COUNT=001 | A=0000 0101 | Q=0011 0001 | M=0100 0111 | C_ACTIVE=                c7

COUNT=010 | A=0000 0101 | Q=0011 0001 | M=0100 0111 | C_ACTIVE=               c11
COUNT=010 | A=0000 1010 | Q=0110 0010 | M=0100 0111 | C_ACTIVE=
COUNT=010 | A=0000 1010 | Q=0110 0010 | M=0100 0111 | C_ACTIVE=               c12
COUNT=010 | A=1100 0011 | Q=0110 0010 | M=0100 0111 | C_ACTIVE=               c13
COUNT=010 | A=0000 1010 | Q=0110 0010 | M=0100 0111 | C_ACTIVE=                c7

COUNT=011 | A=0000 1010 | Q=0110 0010 | M=0100 0111 | C_ACTIVE=               c11
COUNT=011 | A=0001 0100 | Q=1100 0100 | M=0100 0111 | C_ACTIVE=
COUNT=011 | A=0001 0100 | Q=1100 0100 | M=0100 0111 | C_ACTIVE=               c12
COUNT=011 | A=1100 1101 | Q=1100 0100 | M=0100 0111 | C_ACTIVE=               c13
COUNT=011 | A=0001 0100 | Q=1100 0100 | M=0100 0111 | C_ACTIVE=                c7

COUNT=100 | A=0001 0100 | Q=1100 0100 | M=0100 0111 | C_ACTIVE=               c11
COUNT=100 | A=0010 1001 | Q=1000 1000 | M=0100 0111 | C_ACTIVE=
COUNT=100 | A=0010 1001 | Q=1000 1000 | M=0100 0111 | C_ACTIVE=               c12
COUNT=100 | A=1110 0010 | Q=1000 1000 | M=0100 0111 | C_ACTIVE=               c13
COUNT=100 | A=0010 1001 | Q=1000 1000 | M=0100 0111 | C_ACTIVE=                c7

COUNT=101 | A=0010 1001 | Q=1000 1000 | M=0100 0111 | C_ACTIVE=               c11
COUNT=101 | A=0101 0011 | Q=0001 0000 | M=0100 0111 | C_ACTIVE=
COUNT=101 | A=0101 0011 | Q=0001 0000 | M=0100 0111 | C_ACTIVE=               c12
COUNT=101 | A=0000 1100 | Q=0001 0000 | M=0100 0111 | C_ACTIVE=               c14
COUNT=101 | A=0000 1100 | Q=0001 0001 | M=0100 0111 | C_ACTIVE=                c7

COUNT=110 | A=0000 1100 | Q=0001 0001 | M=0100 0111 | C_ACTIVE=               c11
COUNT=110 | A=0001 1000 | Q=0010 0010 | M=0100 0111 | C_ACTIVE=
COUNT=110 | A=0001 1000 | Q=0010 0010 | M=0100 0111 | C_ACTIVE=               c12
COUNT=110 | A=1101 0001 | Q=0010 0010 | M=0100 0111 | C_ACTIVE=               c13
COUNT=110 | A=0001 1000 | Q=0010 0010 | M=0100 0111 | C_ACTIVE=                c7

COUNT=111 | A=0001 1000 | Q=0010 0010 | M=0100 0111 | C_ACTIVE=               c11
COUNT=111 | A=0011 0000 | Q=0100 0100 | M=0100 0111 | C_ACTIVE=
COUNT=111 | A=0011 0000 | Q=0100 0100 | M=0100 0111 | C_ACTIVE=                c8
COUNT=111 | A=0011 0000 | Q=0100 0100 | M=0100 0111 | C_ACTIVE=               c15

[1;32m

X =    4876 (0001001100001100)   Y =   71 (0100 0111)   | REST =   48 (0011 0000)   QUOTIENT =   68 (0100 0100)[0m

[1;31m>>> END signal activated at T=435000. Simulation ends.[0m
** Note: $finish    : ALU_tb.v(159)
   Time: 435 ns  Iteration: 2  Instance: /ALU_tb
```

- **MULTIPLICATION**

  X=-100
  Y=-3

```
COUNT=000 | A=0000 0000 | Q=1001 1100 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=
COUNT=000 | A=0000 0000 | Q=1001 1100 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c6
COUNT=000 | A=0000 0000 | Q=0100 1110 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c7


COUNT=001 | A=0000 0000 | Q=0100 1110 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=
COUNT=001 | A=0000 0000 | Q=0100 1110 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c6
COUNT=001 | A=0000 0000 | Q=0010 0111 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c7


COUNT=010 | A=0000 0000 | Q=0010 0111 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=
COUNT=010 | A=0000 0000 | Q=0010 0111 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=          c4    c5
COUNT=010 | A=0000 0011 | Q=0010 0111 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c6
COUNT=010 | A=0000 0001 | Q=1001 0011 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c7


COUNT=011 | A=0000 0001 | Q=1001 0011 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=
COUNT=011 | A=0000 0001 | Q=1001 0011 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c6
COUNT=011 | A=0000 0000 | Q=1100 1001 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c7


COUNT=100 | A=0000 0000 | Q=1100 1001 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=
COUNT=100 | A=0000 0000 | Q=1100 1001 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c6
COUNT=100 | A=0000 0000 | Q=0110 0100 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c7


COUNT=101 | A=0000 0000 | Q=0110 0100 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=
COUNT=101 | A=0000 0000 | Q=0110 0100 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c4
COUNT=101 | A=1111 1101 | Q=0110 0100 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=                c6
COUNT=101 | A=1111 1110 | Q=1011 0010 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c7


COUNT=110 | A=1111 1110 | Q=1011 0010 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=
COUNT=110 | A=1111 1110 | Q=1011 0010 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c6
COUNT=110 | A=1111 1111 | Q=0101 1001 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c7


COUNT=111 | A=1111 1111 | Q=0101 1001 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=
COUNT=111 | A=1111 1111 | Q=0101 1001 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=          c4    c5
COUNT=111 | A=0000 0010 | Q=0101 1001 | Q[-1]=0 | M=1111 1101 | C_ACTIVE=                c6
COUNT=111 | A=0000 0001 | Q=0010 1100 | Q[-1]=1 | M=1111 1101 | C_ACTIVE=          c8    c9


[1;32m

X =   -100 (1001 1100)   Y =    -3 (1111 1101)   | OUT =    300 (0000000100101100)[0m


[1;31m>>> END signal activated at T=445000. Simulation ends.[0m
** Note: $finish    : ALU_tb.v(159)
   Time: 445 ns  Iteration: 2  Instance: /ALU_tb
```

- **SUBTRACTION**

  X=85
  Y=23

```
COUNT=000 | A=0000 0000 | Q=0000 0000 | M=0000 0000 | C_ACTIVE=
COUNT=000 | A=0000 0000 | Q=0000 0000 | M=0000 0000 | C_ACTIVE=                c0
COUNT=000 | A=0000 0000 | Q=0101 0101 | M=0001 0111 | C_ACTIVE=                c1
COUNT=000 | A=0000 0000 | Q=0101 0101 | M=0001 0111 | C_ACTIVE=           c2   c3
COUNT=000 | A=0011 1110 | Q=0101 0101 | M=0001 0111 | C_ACTIVE=
COUNT=000 | A=0011 1110 | Q=0101 0101 | M=0001 0111 | C_ACTIVE=                c8
COUNT=000 | A=0011 1110 | Q=0101 0101 | M=0001 0111 | C_ACTIVE=                c9


[1;32m

X =     85 (0101 0101)   Y =    23 (0001 0111)   | OUT_A =   62 (0011 1110)   [0m


[1;31m>>> END signal activated at T=85000. Simulation ends.[0m
** Note: $finish    : ALU_tb.v(159)
   Time: 85 ns  Iteration: 2  Instance: /ALU_tb
1
```

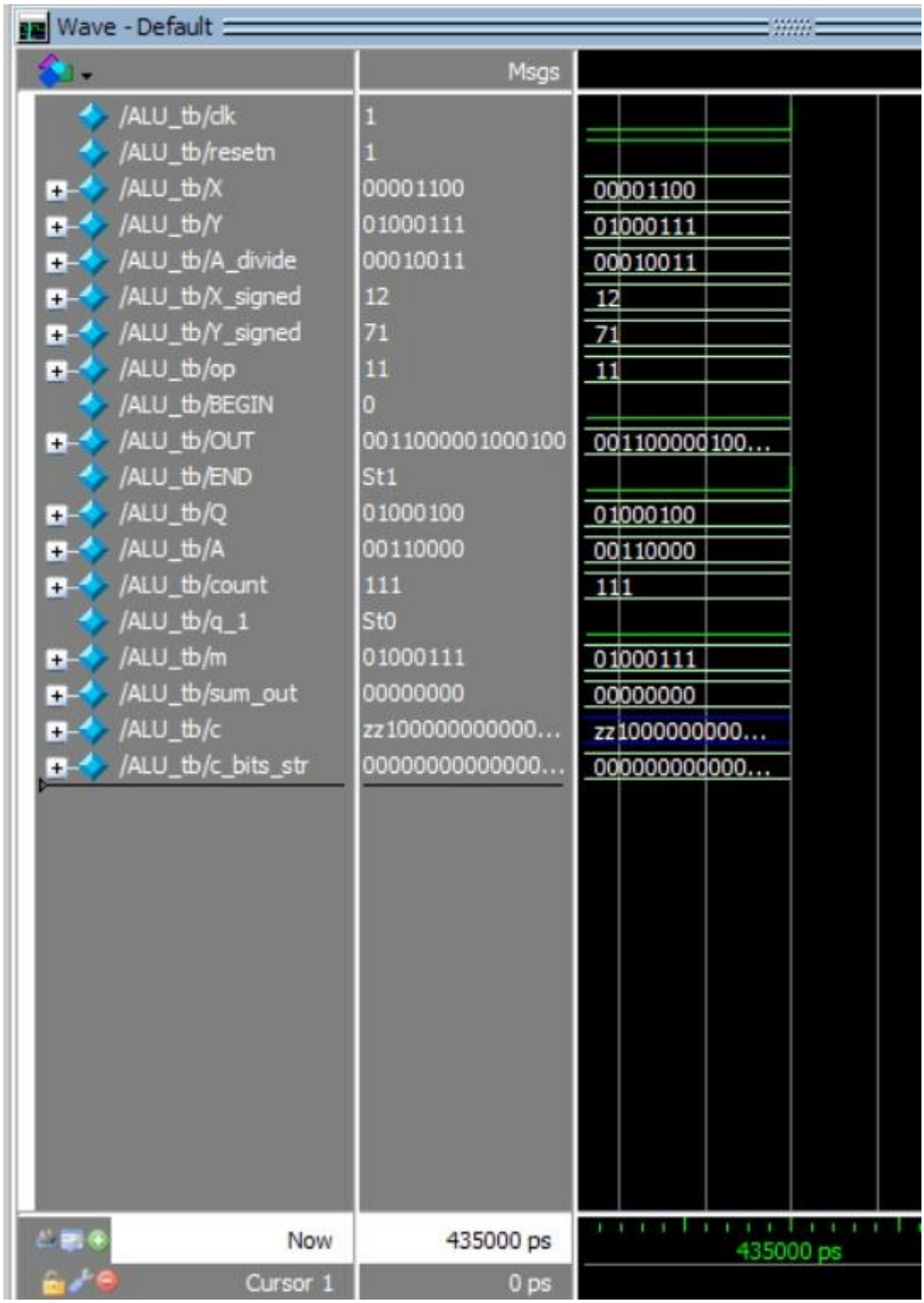
- **ADDITION**

  X=67
  Y=16

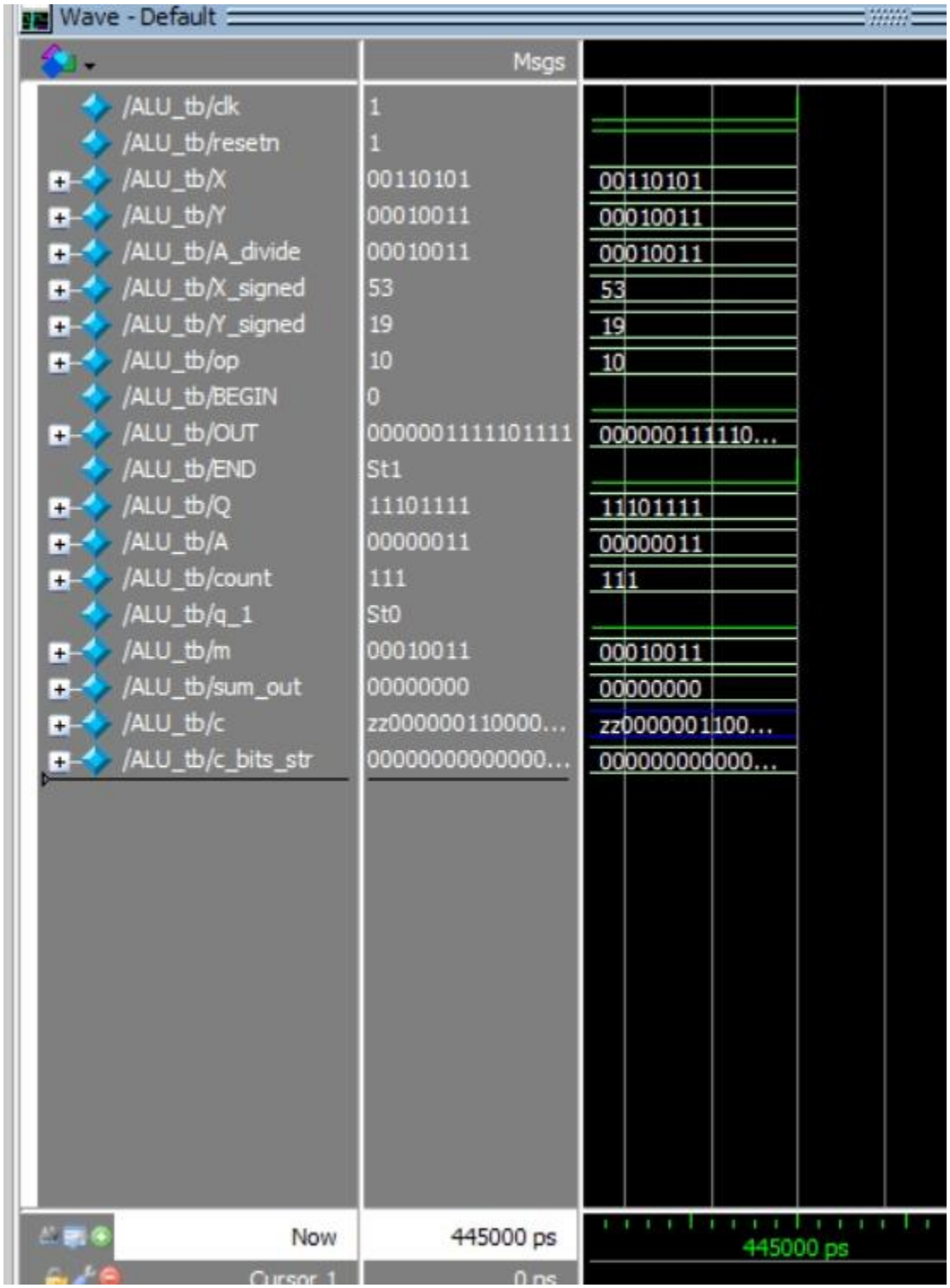
```
COUNT=000 | A=0000 0000 | Q=0000 0000 | M=0000 0000 | C_ACTIVE=
COUNT=000 | A=0000 0000 | Q=0000 0000 | M=0000 0000 | C_ACTIVE=                c0
COUNT=000 | A=0000 0000 | Q=0100 0011 | M=0001 0000 | C_ACTIVE=                c1
COUNT=000 | A=0000 0000 | Q=0100 0011 | M=0001 0000 | C_ACTIVE=                c2
COUNT=000 | A=0101 0011 | Q=0100 0011 | M=0001 0000 | C_ACTIVE=
COUNT=000 | A=0101 0011 | Q=0100 0011 | M=0001 0000 | C_ACTIVE=                c8
COUNT=000 | A=0101 0011 | Q=0100 0011 | M=0001 0000 | C_ACTIVE=                c9


[1;32m

X =     67 (0100 0011)   Y =    16 (0001 0000)   | OUT_A =   83 (0101 0011)   [0m


[1;31m>>> END signal activated at T=85000. Simulation ends.[0m
** Note: $finish    : ALU_tb.v(159)
   Time: 85 ns  Iteration: 2  Instance: /ALU_tb
```

Waveform Simulation

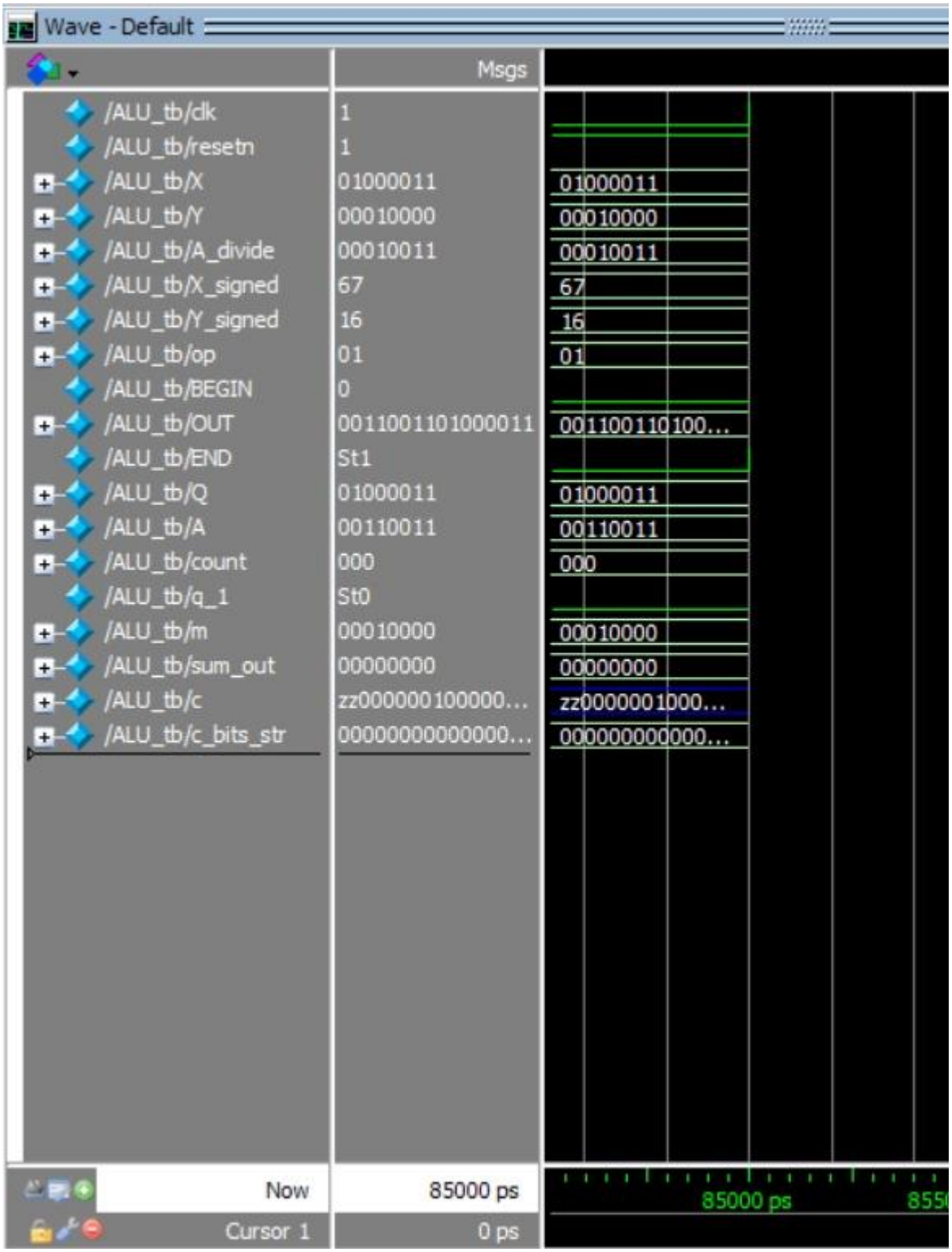
- **DIVISION**



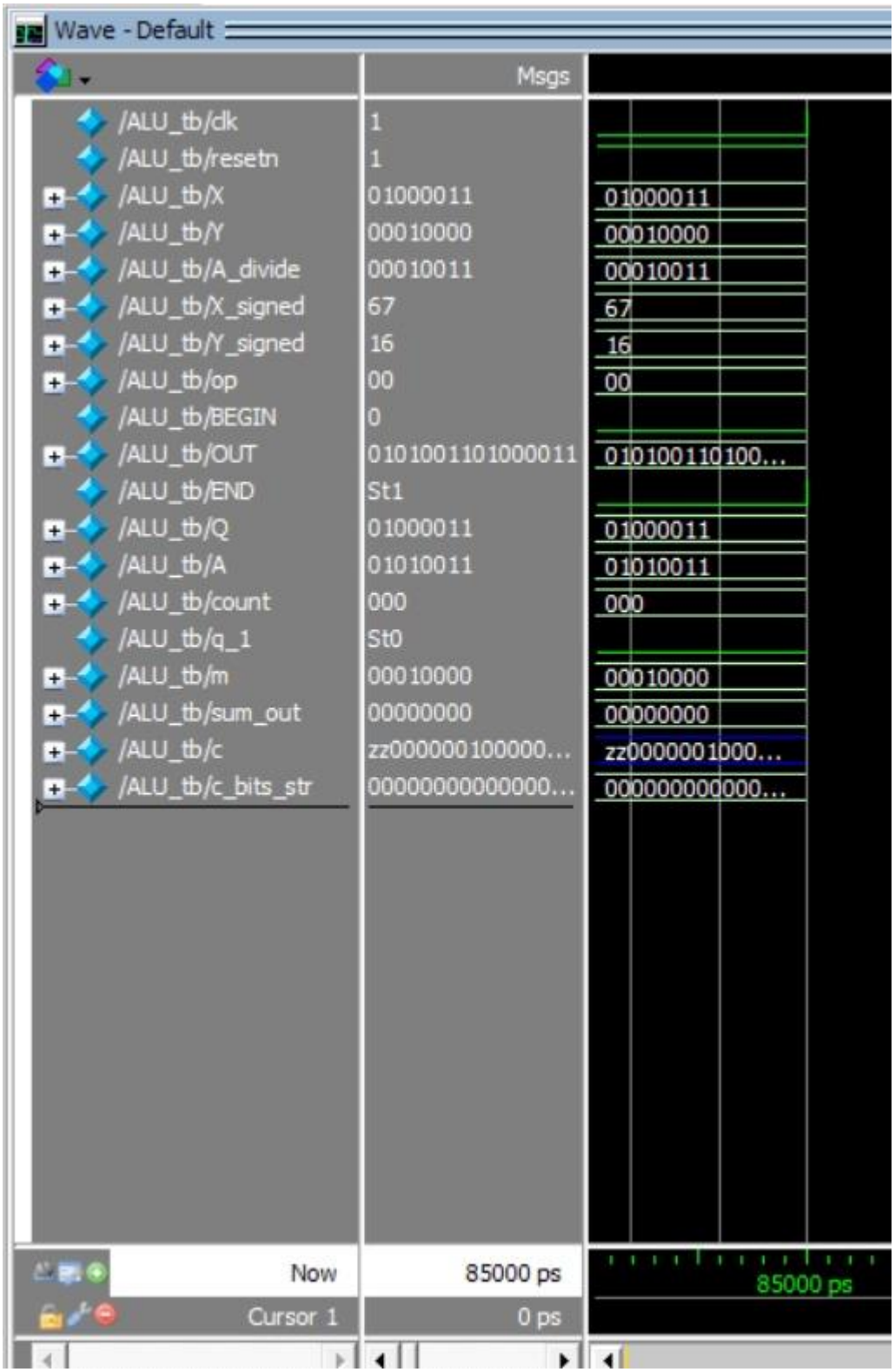
- **MULTIPLICATION**



- **SUBTRACTION**



- **ADDITION**


Wave - Default
Msgs
/ALU_tb/clk
/ALU_tb/resetn
/ALU_tb/X
/ALU_tb/Y
/ALU_tb/A_divide
/ALU_tb/X_signed
/ALU_tb/Y_signed
/ALU_tb/op
/ALU_tb/BEGIN
/ALU_tb/OUT
/ALU_tb/END
/ALU_tb/Q
/ALU_tb/A
/ALU_tb/count
/ALU_tb/q_1
/ALU_tb/m
/ALU_tb/sum_out
/ALU_tb/c
/ALU_tb/c_bits_str
1
1
01000011
00010000
00010011
67
16
00
0
0101001101000011
St1
01000011
01010011
000
St0
00010000
00000000
zz000000100000...
000000000000000...
01000011
00010000
00010011
67
16
00
01010011001100...
01000011
01010011
000
00010000
00000000
zz0000001000...
000000000000...
Now
85000 ps
Cursor 1
0 ps
85000 ps

3. Bibliography

- Opritoiu Flavius, (2025) Computer Arhitecture, Polytechnic University of Timisoara – course
- Mihai Udrescu, Digital Computers – course
- Hennessy & Patterson - Computer Organization and Design, 5th ed
- Hennessy & Patterson - Computer Organization and Design, RISC-V Edition: The Hardware-Software Interface

Tools used:

ModelSim – used for running Verilog code and underlying simulation for compiling

EDAPlayground – used for writing and testing the code

NotePad – used for writing the code to run on ModelSim

ChatGPT – used in refining the written content to ensure clarity