NMJ31804 – Principles of Computer Architecture
SEMESTER 2, 2021/22

LAB 2

Name: TAN YI JIE
Matric Number: 191020976
Program code: RK20 - Computer Engineering

Coding

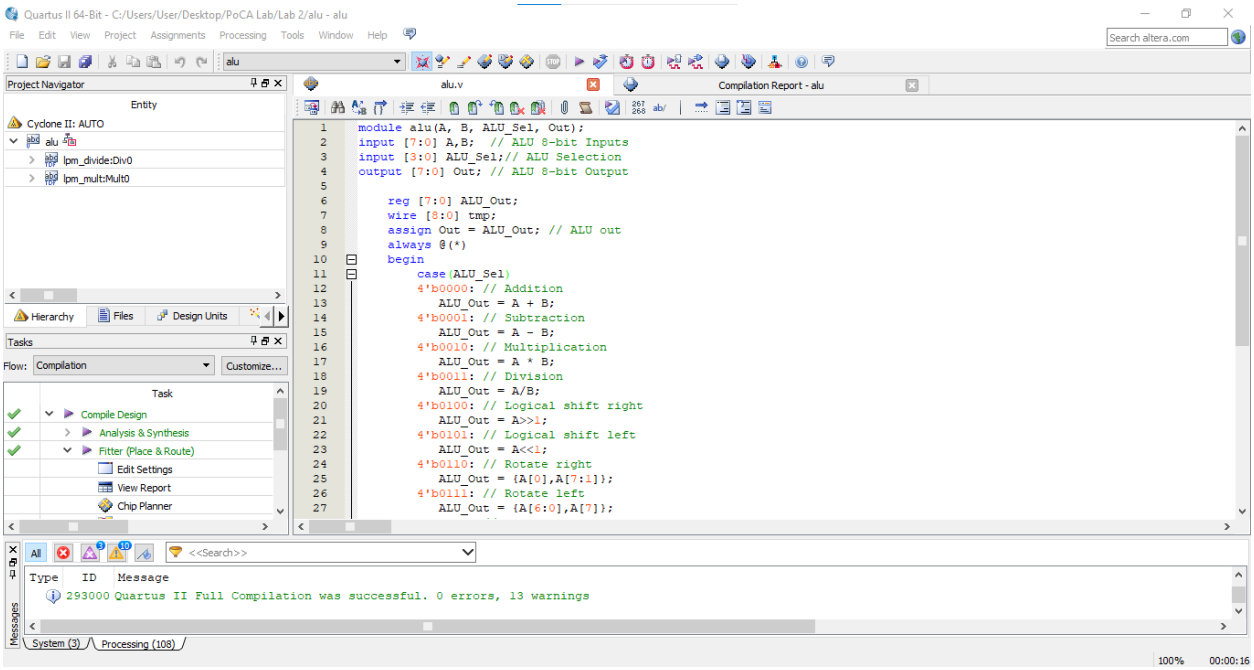

Figure. 1

Figure. 1 shows that the compilation of coding is successful, no errors. The full coding is listed at the Appendix.

Output (Waveform)

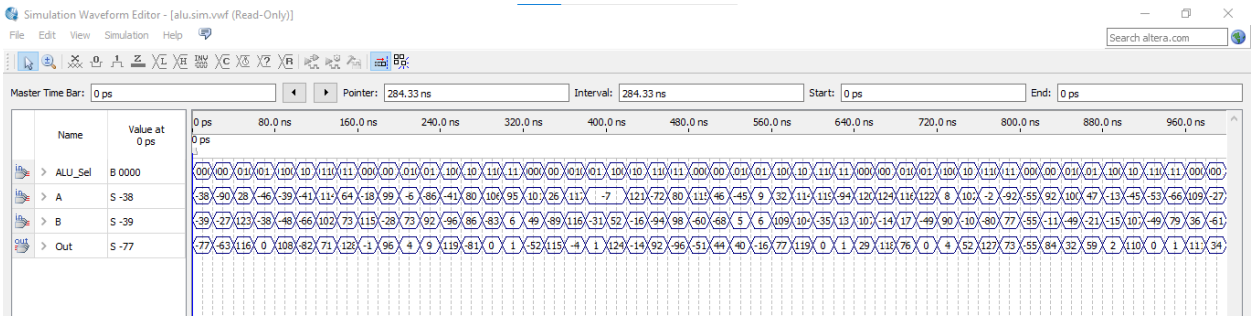

Figure. 2

Figure. 2 shows that ALU_Sel been set as count value for every interval of 20.0ns while A and B been set as random value for every interval of 20.0ns. The radix of A, B, and Out been assigned as signed decimal.

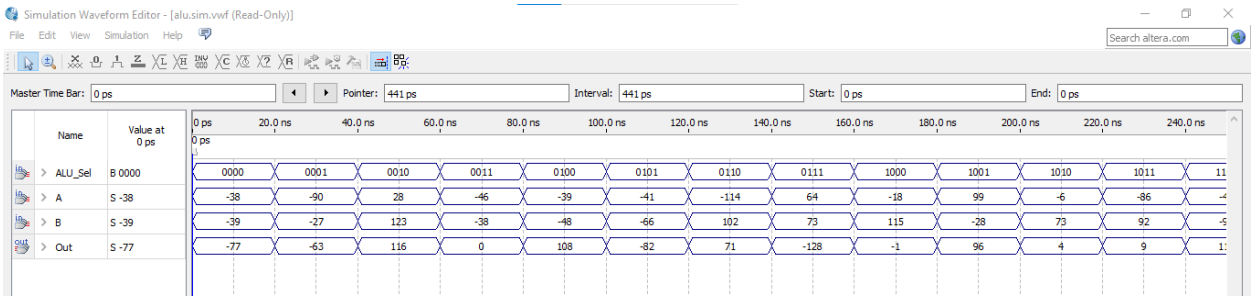

Figure. 3

Figure. 3 shows that the radix been set as signed decimal to let the user easier to read the result of arithmetic operation. The first interval result, Out = -38 + (-39) = -77.

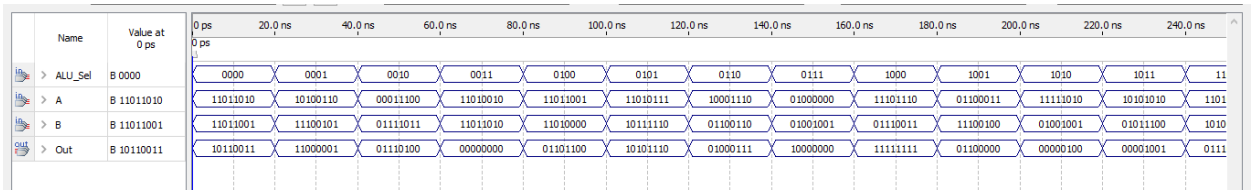

Figure. 4

Figure. 4 shows that The radix been set as binary to let the user easier to read the result of logic operation and comparison. The 7th interval result of 0110, Out = 01000111, which is the right rotation of A by 1.

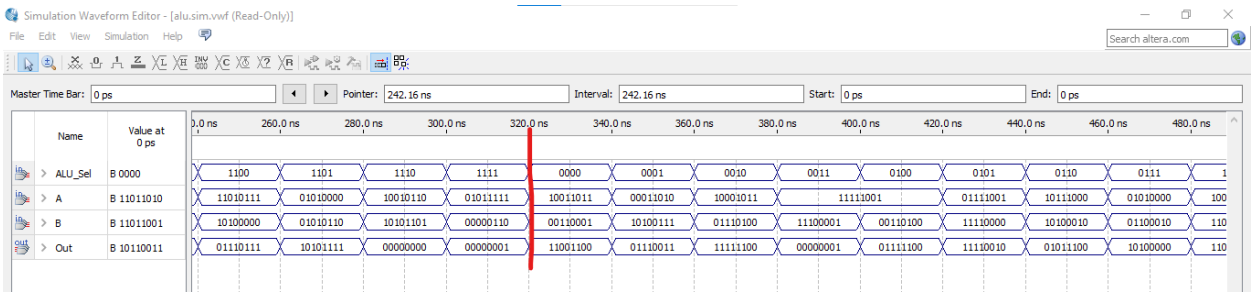

Figure. 5

Figure. 5 shows the remaining four result.

Appendix

```
module alu(A, B, ALU_Sel, Out);
input [7:0] A,B;  // ALU 8-bit Inputs
input [3:0] ALU_Sel;// ALU Selection
output [7:0] Out; // ALU 8-bit Output

    reg [7:0] ALU_Out;
    wire [8:0] tmp;
    assign Out = ALU_Out; // ALU out
    always @(*)
    begin
        case(ALU_Sel)
        4'b0000: // Addition
            ALU_Out = A + B;
        4'b0001: // Subtraction
            ALU_Out = A - B;
        4'b0010: // Multiplication
            ALU_Out = A * B;
        4'b0011: // Division
            ALU_Out = A/B;
        4'b0100: // Logical shift right
            ALU_Out = A>>1;
        4'b0101: // Logical shift left
            ALU_Out = A<<1;
        4'b0110: // Rotate right
```

```
        ALU_Out = {A[0],A[7:1]};
      4'b0111: // Rotate left
        ALU_Out = {A[6:0],A[7]};
      4'b1000: //  Logical OR
        ALU_Out = A | B;
      4'b1001: //  Logical AND
        ALU_Out = A & B;
      4'b1010: //  Logical NOR
        ALU_Out = ~(A | B);
      4'b1011: // Logical XNOR
        ALU_Out = ~(A ^ B);
      4'b1100: //  Logical XOR
        ALU_Out = A ^ B;
      4'b1101: // Logical NAND
        ALU_Out = ~(A & B);
      4'b1110: // Equal comparison
        ALU_Out = (A==B)?8'd1:8'd0;
      4'b1111: // Greater comparison
        ALU_Out = (A>B)?8'd1:8'd0;
      default: ALU_Out = A + B;
      endcase
  end
endmodule
```