**CSE4117 – Microprocessors**



**THE DESIGN OF ALU**

**NAME SURNAME** **STUDENT NUMBER**

Aydın DUYGU 150118981

Tugay SARICI 150119829

**INTRODUCTION**

In this assignment, we designed an 8-bit ALU that can perform addition, subtraction, and, or, shift-left, shift-right, move and xor operations. As a result of these operations, we obtained the result, and we ensured that the carry, zero, overflow and negative flags were set. On this road, we first needed an eight-by-one multiplexer. Because we had eight operations to perform. The zero and negative flags were determined for all operations, but the carry and overflow flags were just determined at the addition and the substraction operations.

We designed a full subtractor to implement the substraction operation. However, we first built a single-bit full subtractor. Then we were able to obtain a two-bits subtractor using single-bits, four-bits using two-bits, and an eight-bits subtractor using four-bits, respectively.

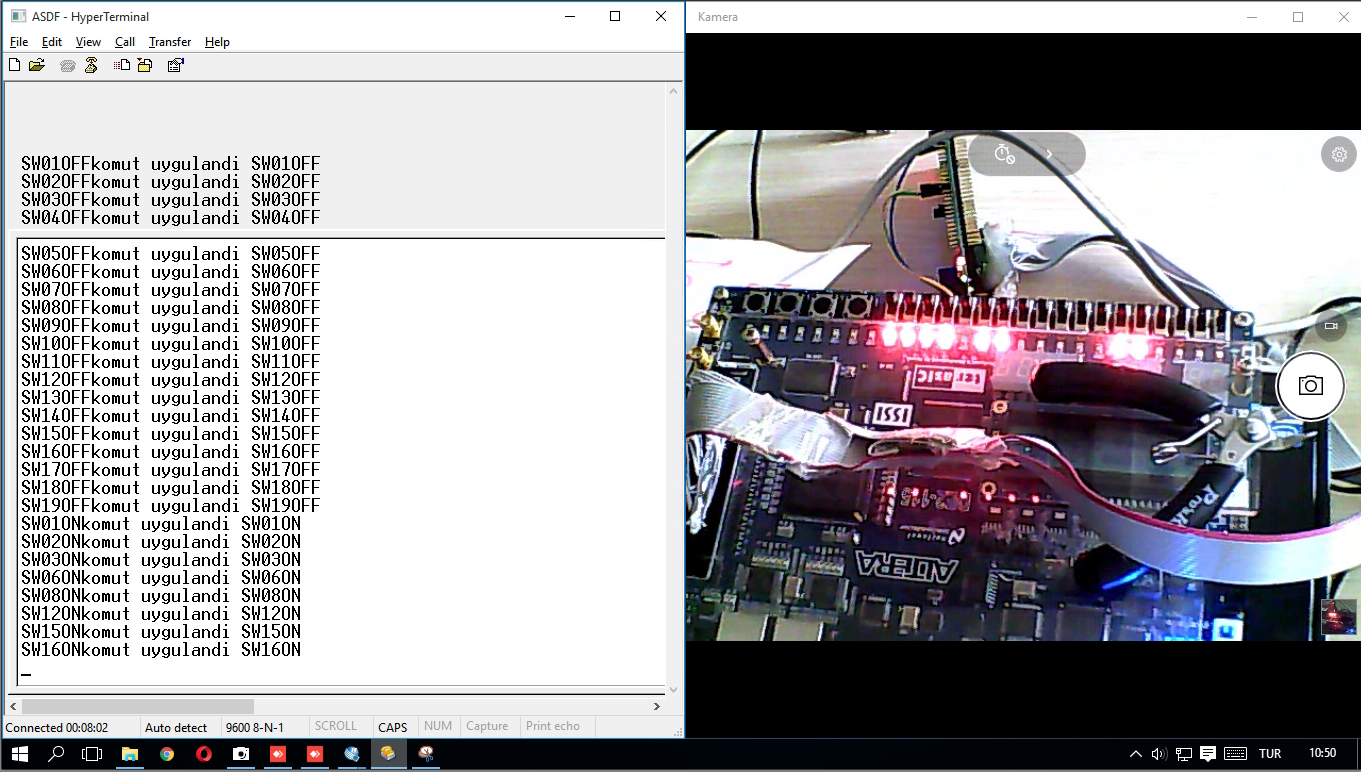
We had to check the most significant bits of the operands and the result in order to detect whether there was an overflow. Because in an addition operation, if the operands are negative and the result is positive, or vice versa, it means overflow. We proceeded in the subtraction process by checking the most important bits in a similar way for overflow detection. We have seen that our design yielded successful results both in our simulations and in our tests on the fpga device. We attach the simulation results and photos of our tests on the fpga device below.

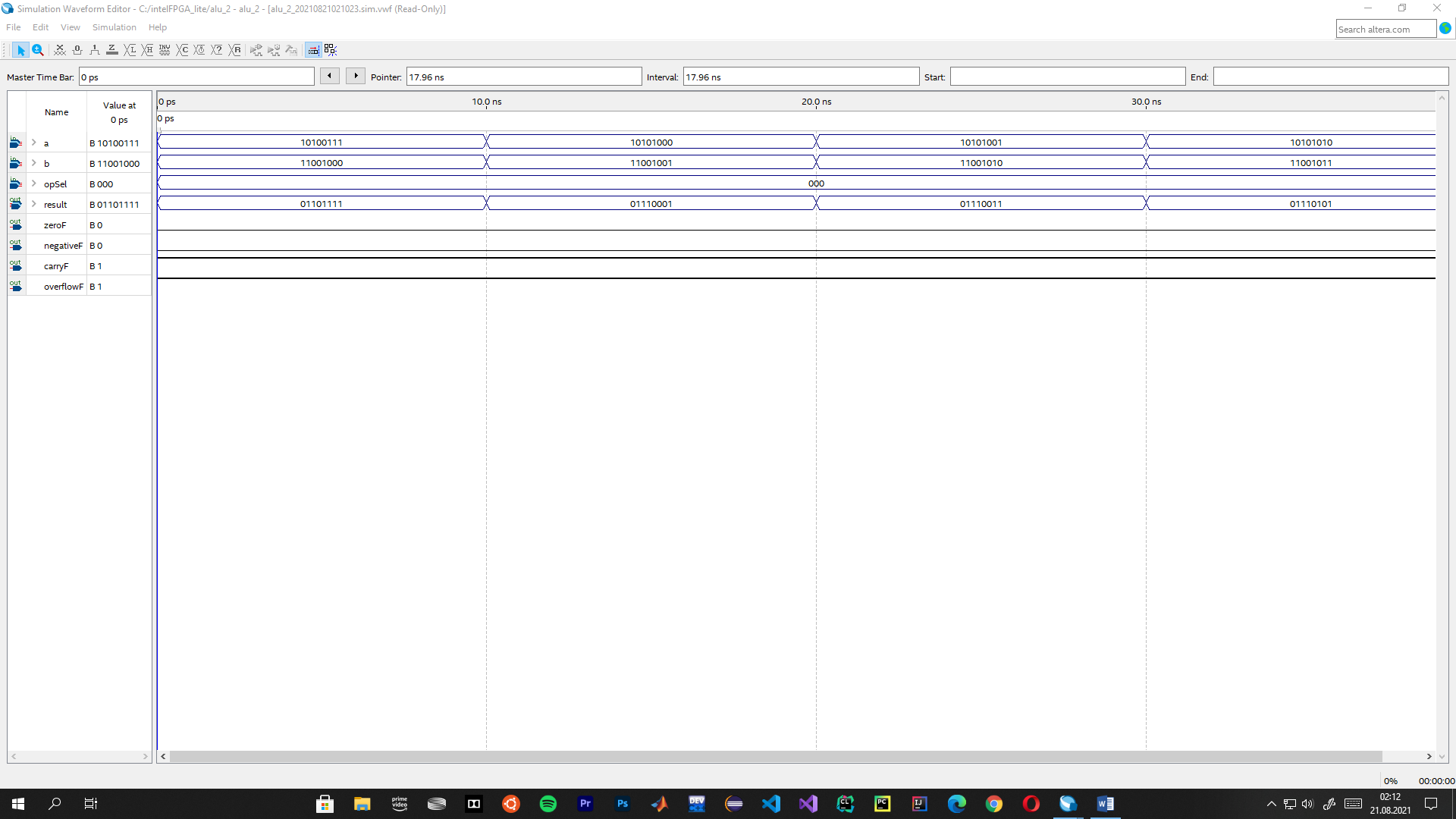
**The RTL View was attached as separate from the word document.**

1010 0111 (-89) and 1100 1000 (-56) operands were used in the eight operations.

**Addition Operation**

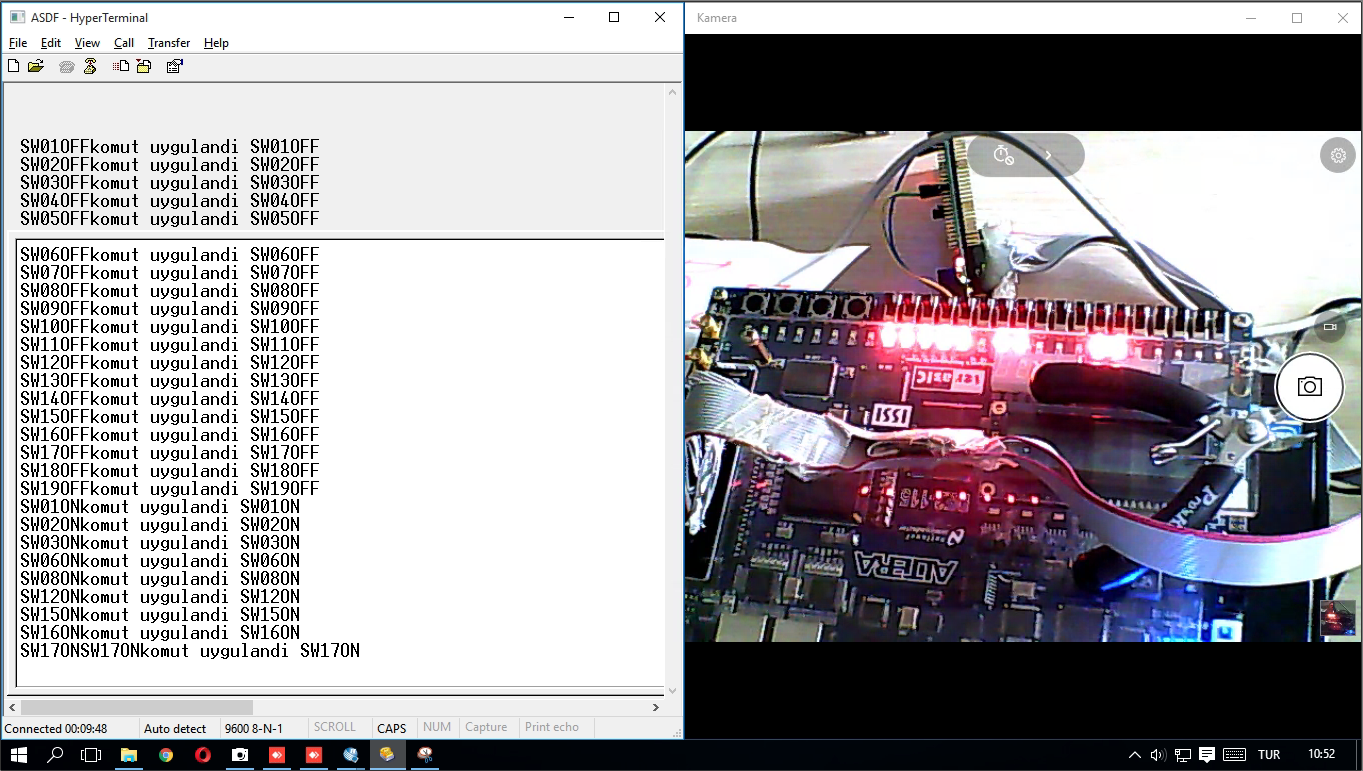
According to the operands; the result is 0110 1111 (111), the zero, negative, carry and overflow flags were found as 0, 0, 1, 1 respectively.

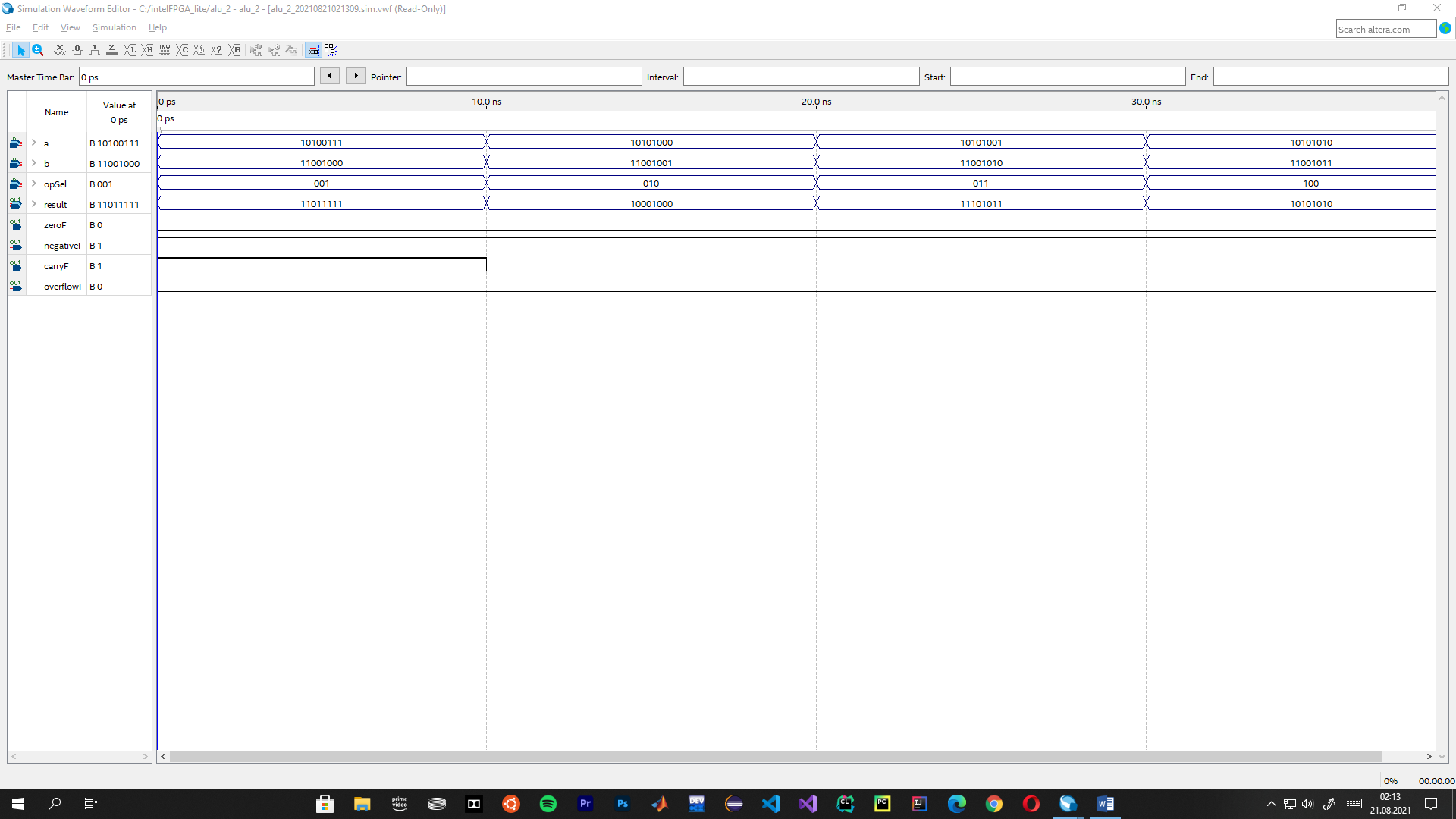




**Substraction Operation**

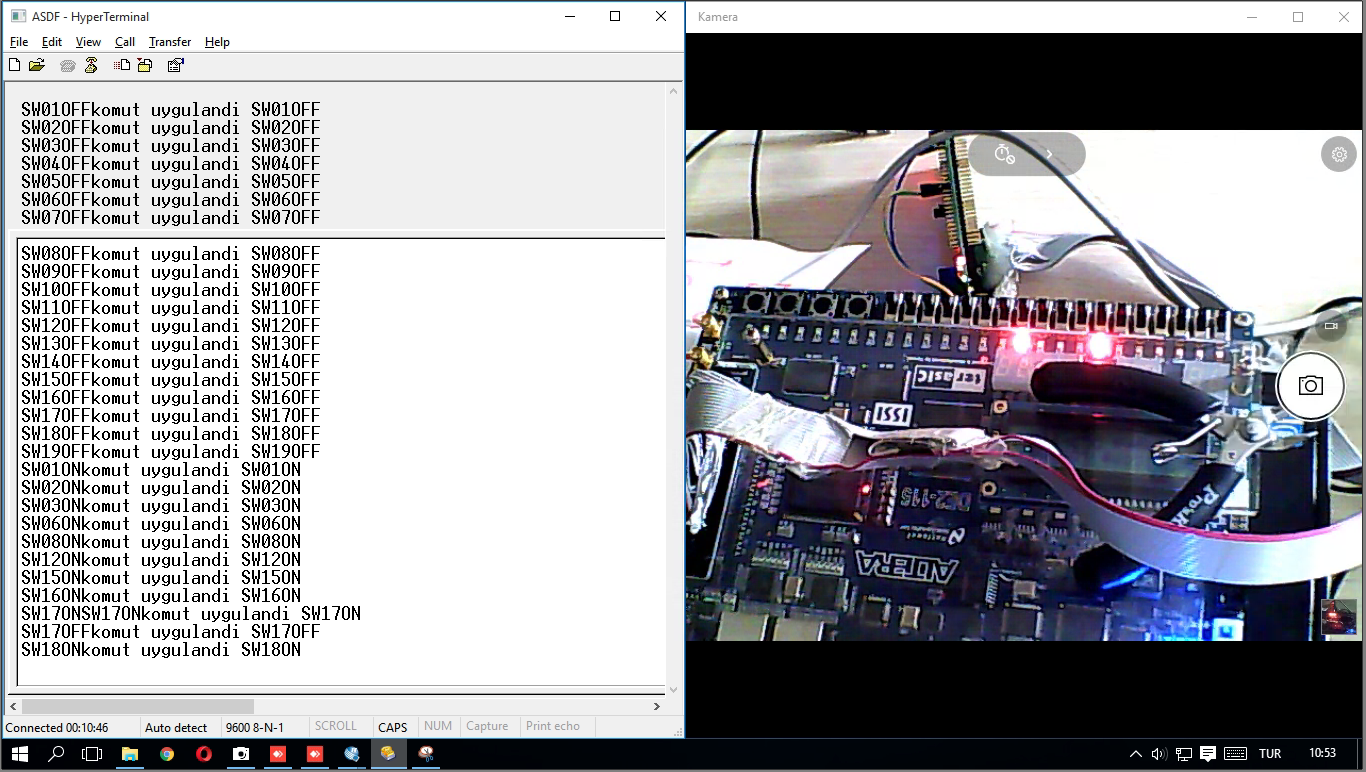
According to the operands; the result is 1110 1111 (-17), the zero, negative, carry and overflow flags were found as 0, 1, 1, 0 respectively.

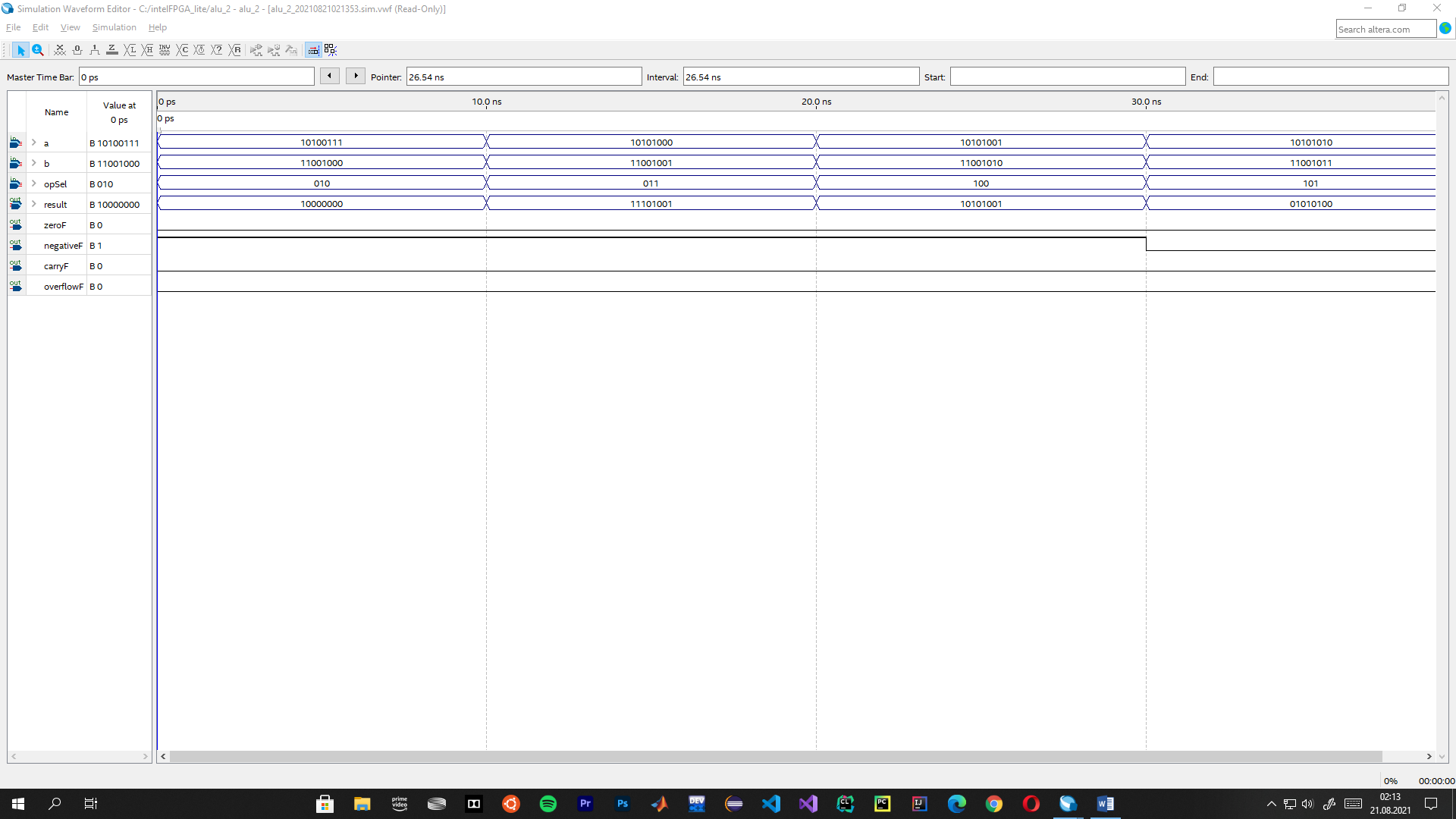




**AND Operation**

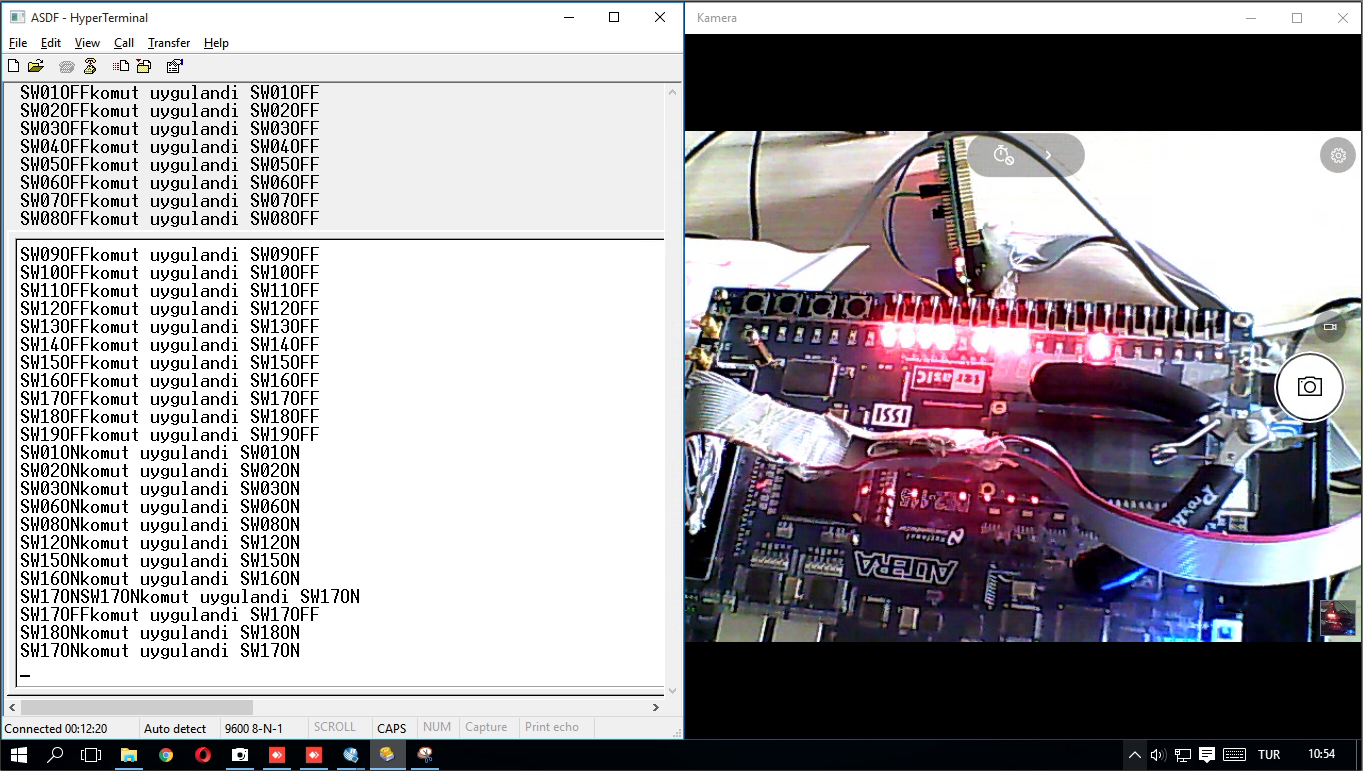
According to the operands; the result is 1000 0000 (-128), the zero, negative, carry and overflow flags were found as 0, 1, 0, 0 respectively.

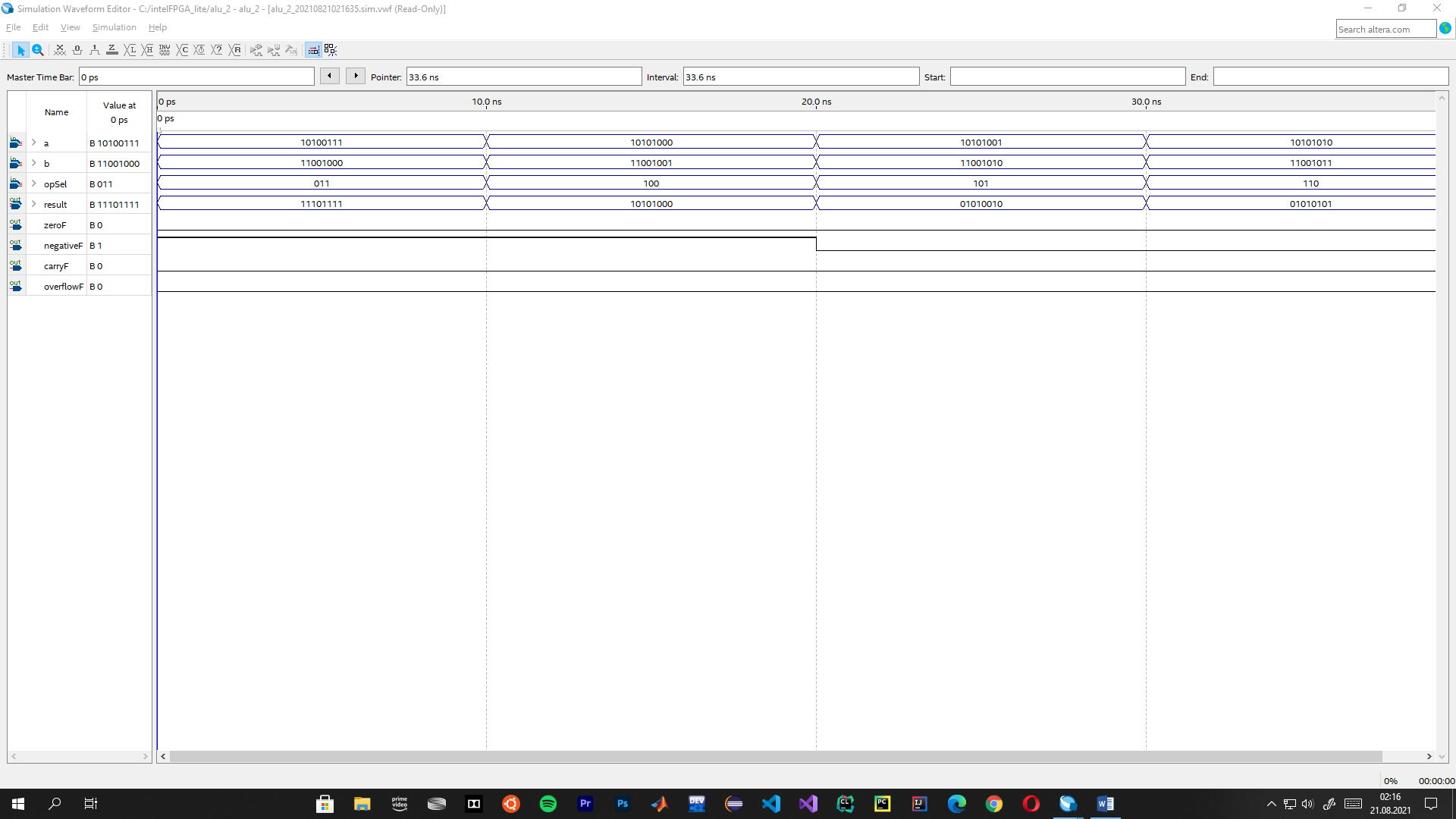




**OR Operation**

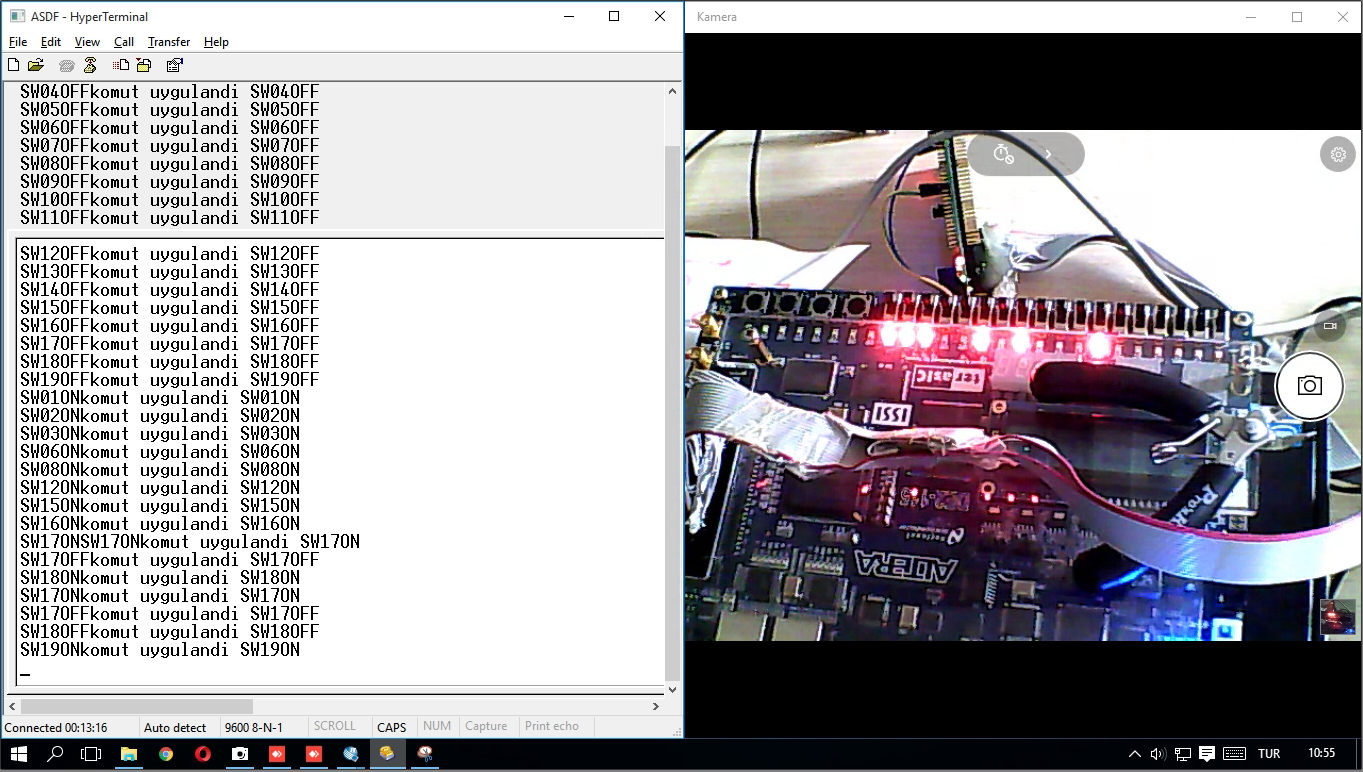
According to the operands; the result is 1110 1111 (-17), the zero, negative, carry and overflow flags were found as 0, 1, 0, 0 respectively.

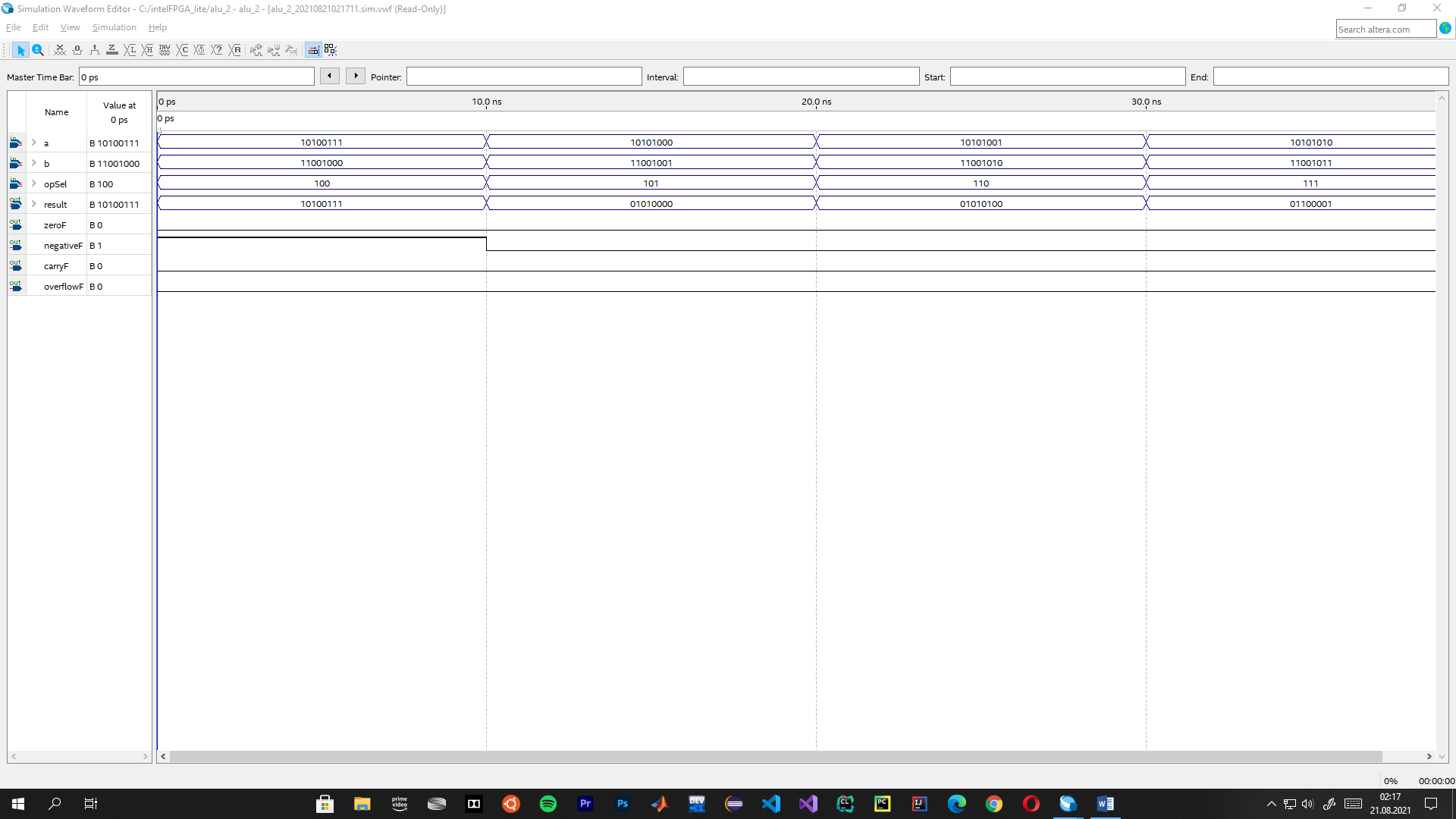




**Mov A Operation**

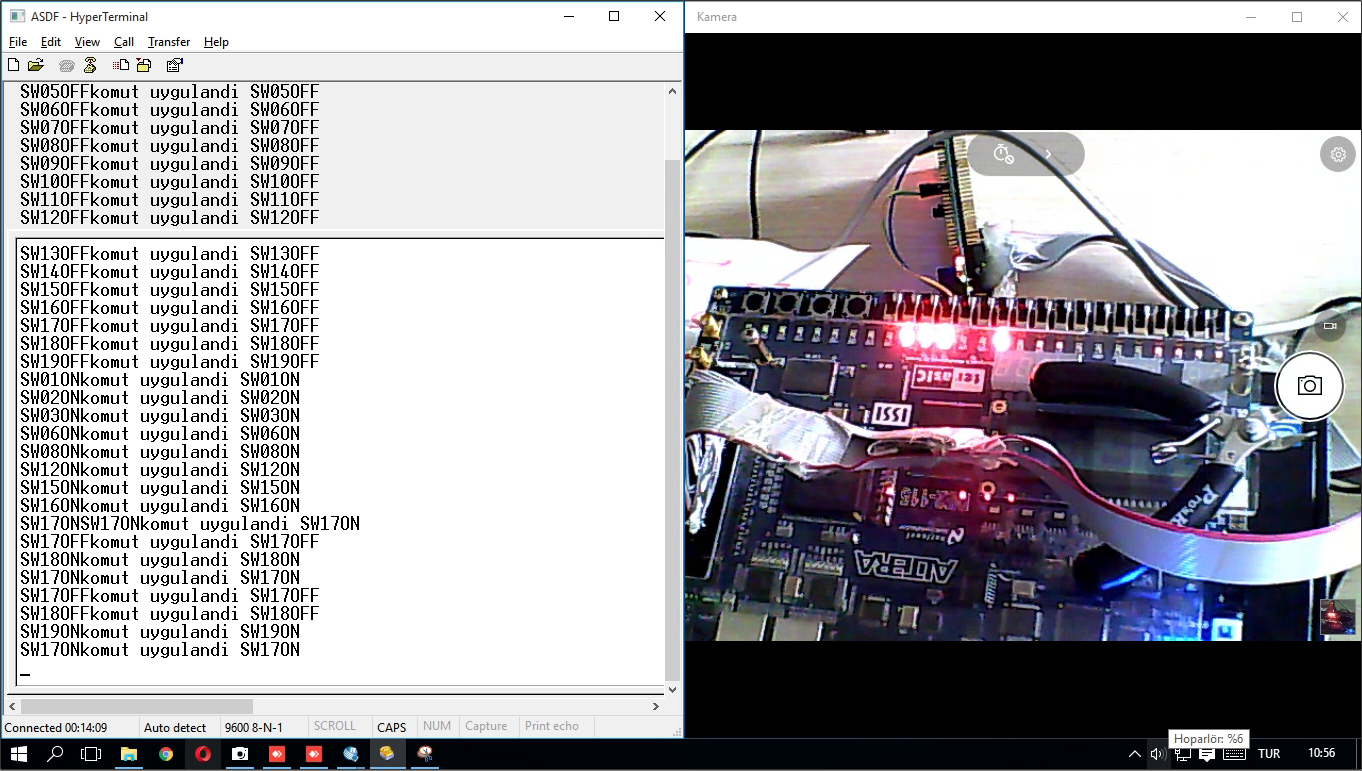
According to the operands; the result is 1010 0111 (-89), the zero, negative, carry and overflow flags were found as 0, 1, 0, 0 respectively.

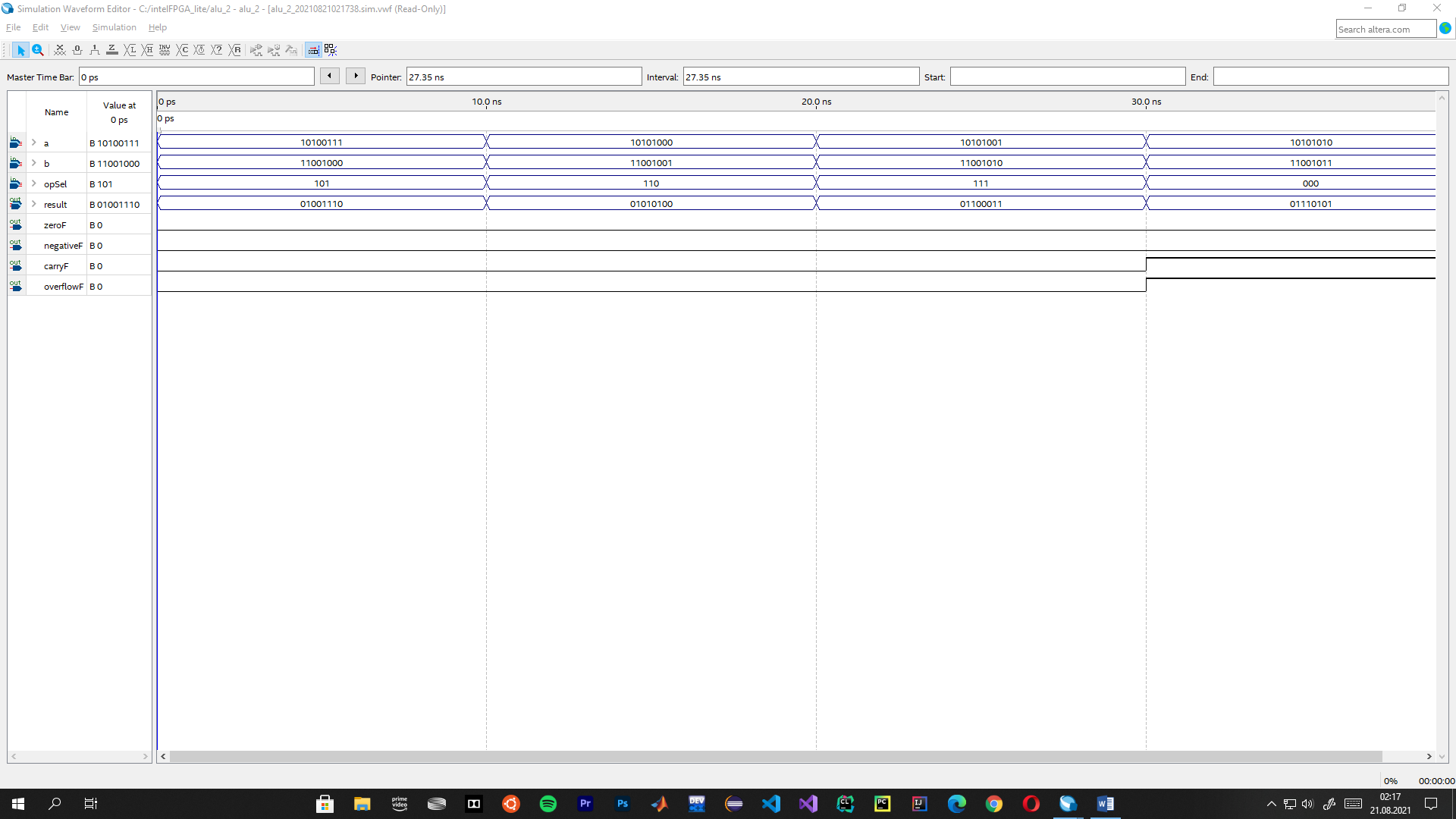




**Shift Left Operation**

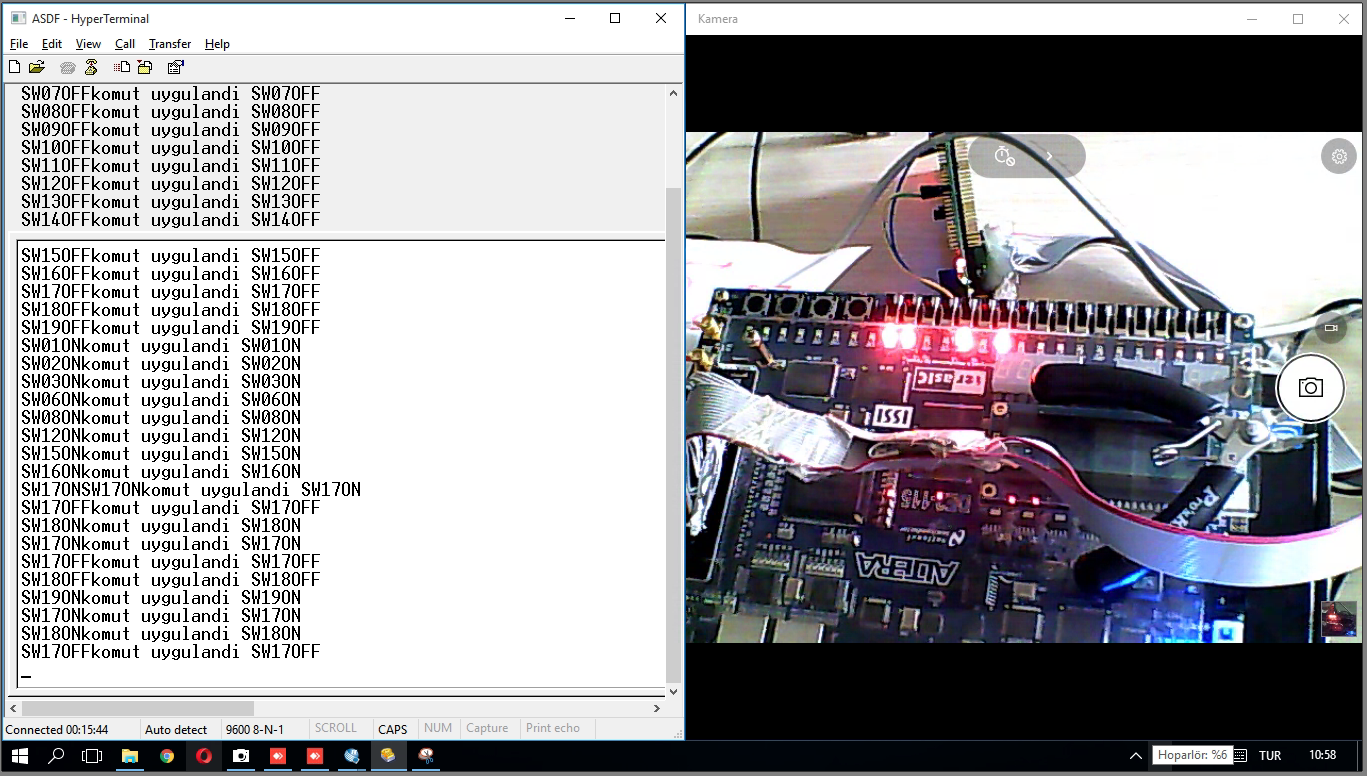
According to the operands; the result is 0100 1110 (78), the zero, negative, carry and overflow flags were found as 0, 0, 0, 0 respectively.

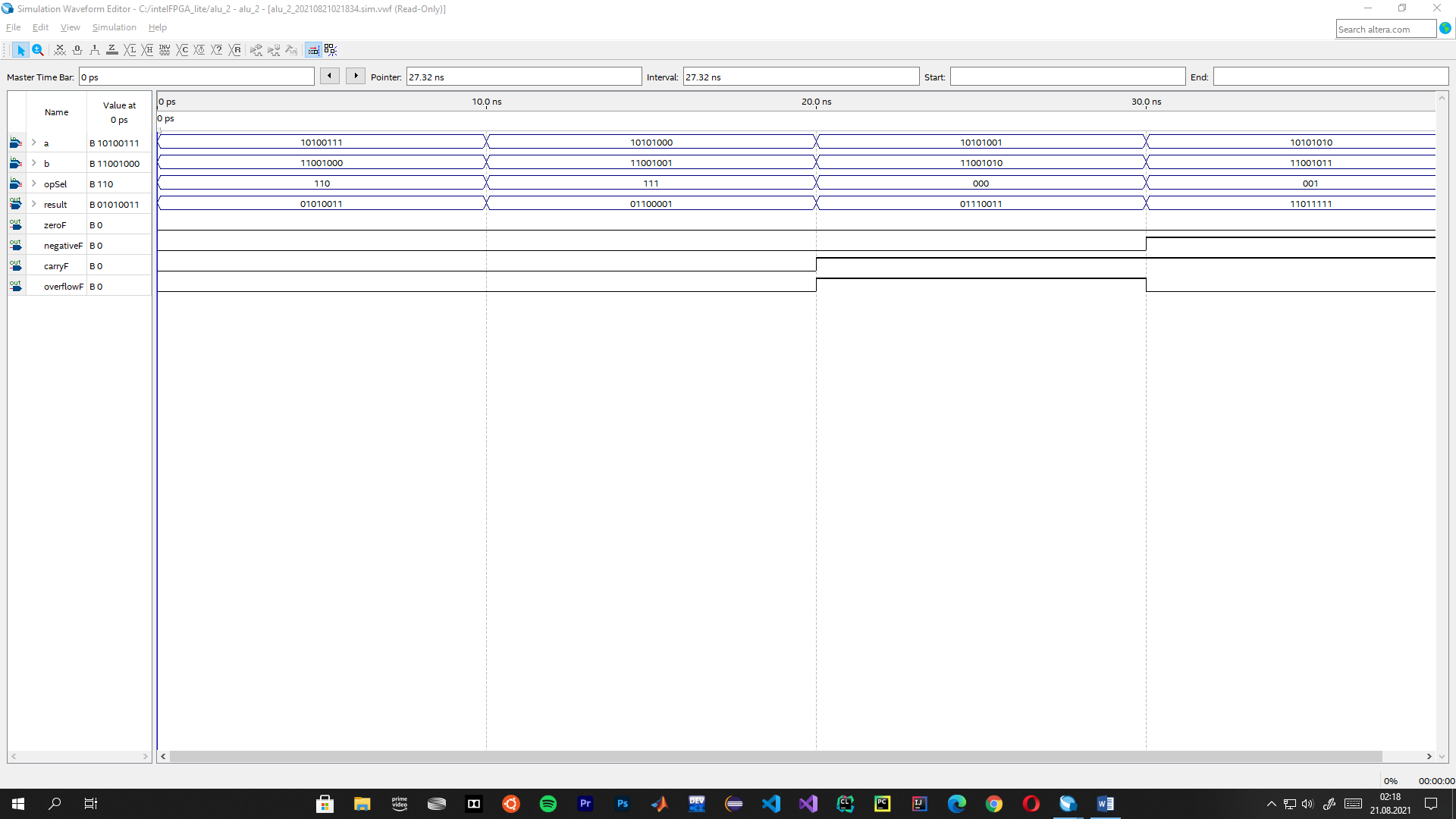




**Shift Right Operation**

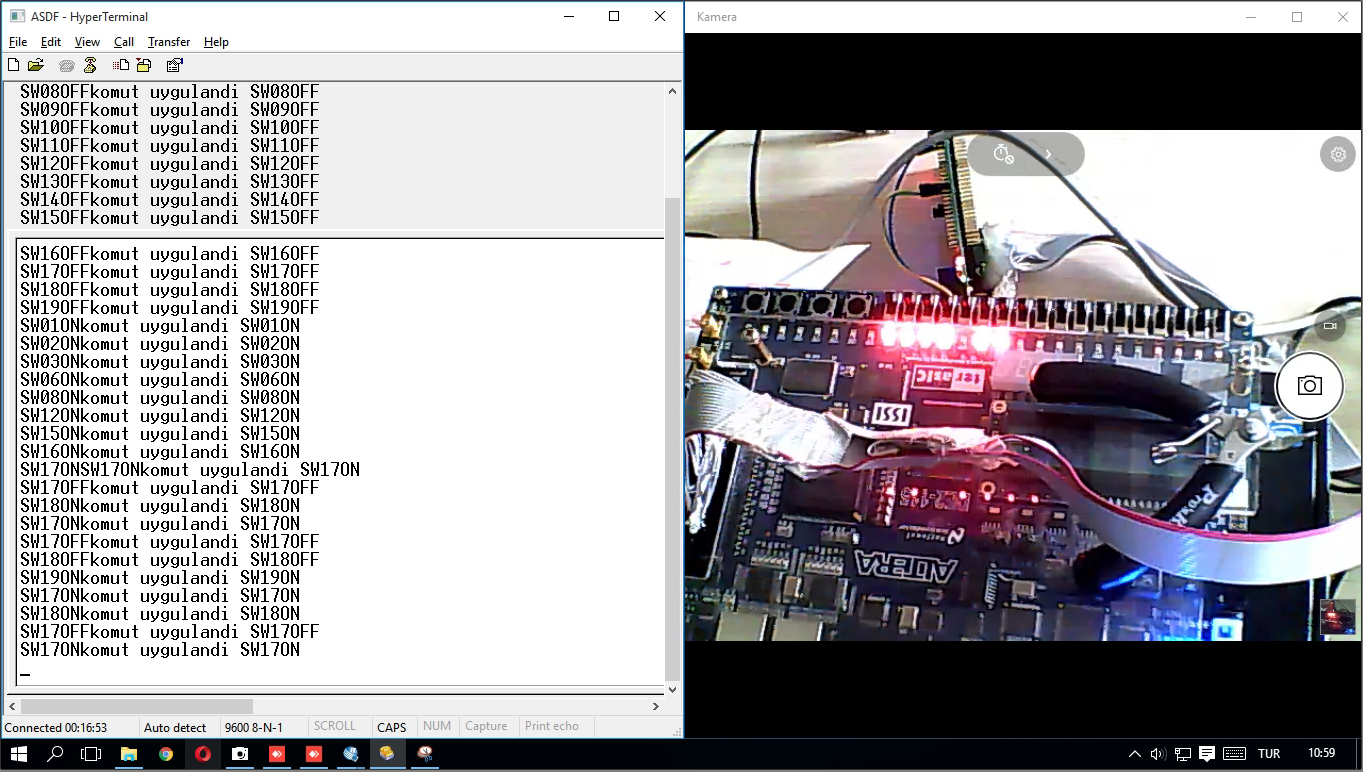
According to the operands; the result is 0101 0011 (83), the zero, negative, carry and overflow flags were found as 0, 0, 0, 0 respectively.

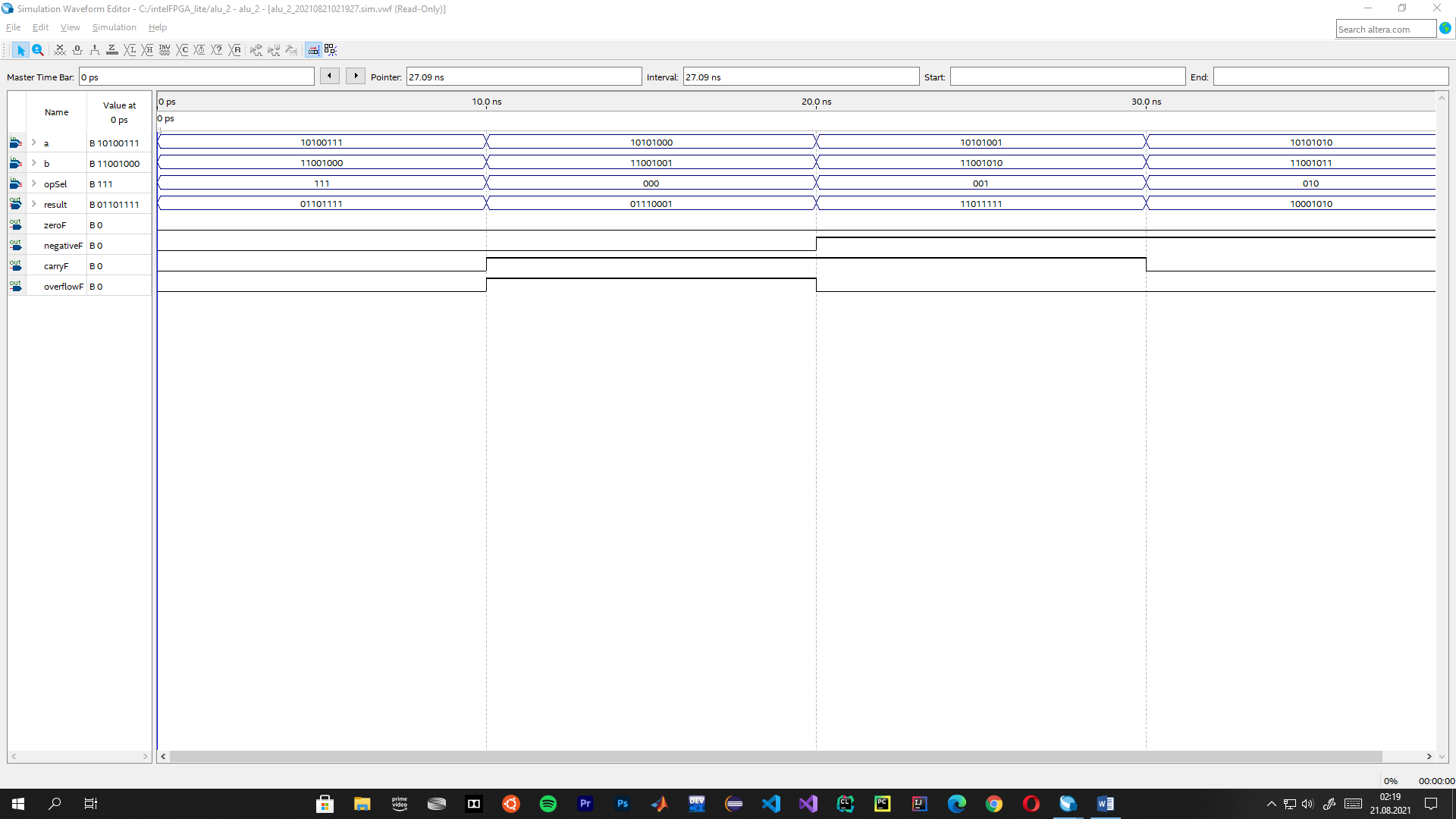




**XOR Operation**

According to the operands; the result is 0110 1111 (111), the zero, negative, carry and overflow flags were found as 0, 0, 0, 0 respectively.





**Source Code**

|  |
| --- |
| module onebitsubstractor(input a, input b, input c\_in, output c\_out, output result);  assign c\_out = (~a&b)|(b&c\_in)|(~a&c\_in);  assign result = (a ^ b) ^ c\_in;  endmodule  //-------------------------------------------------------------------  module twobitsubstractor(input [1:0]a,input [1:0]b, input c\_in, output c\_out, output [1:0] result);  wire cout;  onebitsubstractor s0(a[0],b[0],c\_in,cout,result[0]);  onebitsubstractor s1(a[1],b[1],cout,c\_out,result[1]);  endmodule  //-------------------------------------------------------------------  module fourbitsubstractor(input [3:0]a,input [3:0]b, input c\_in, output c\_out, output [3:0] result);  wire cout;  twobitsubstractor s0(a[1:0],b[1:0],c\_in,cout,result[1:0]);  twobitsubstractor s1(a[3:2],b[3:2],cout,c\_out,result[3:2]);  endmodule  //-------------------------------------------------------------------  module eightbitsubstractor(input [7:0]a,input [7:0]b, input c\_in, output c\_out, output [7:0] result);  wire cout;  fourbitsubstractor s0(a[3:0],b[3:0],c\_in,cout,result[3:0]);  fourbitsubstractor s1(a[7:4],b[7:4],cout,c\_out,result[7:4]);  endmodule  module fulladder(  input [7:0] a,  input [7:0] b,  input c\_in,  output reg c\_out,  output reg [7:0] resultFullAdder);  always @(a or b or c\_in)  begin  {c\_out, resultFullAdder} = a+b+c\_in;  end  endmodule  //-------------------------------------------------------------------  module mux2to1(in0,in1,sel0,out);  input[7:0] in0,in1;  input sel0;  output[7:0] out;  assign out[7] = (in0[7] & ~sel0)|(in1[7] & sel0);  assign out[6] = (in0[6] & ~sel0)|(in1[6] & sel0);  assign out[5] = (in0[5] & ~sel0)|(in1[5] & sel0);  assign out[4] = (in0[4] & ~sel0)|(in1[4] & sel0);  assign out[3] = (in0[3] & ~sel0)|(in1[3] & sel0);  assign out[2] = (in0[2] & ~sel0)|(in1[2] & sel0);  assign out[1] = (in0[1] & ~sel0)|(in1[1] & sel0);  assign out[0] = (in0[0] & ~sel0)|(in1[0] & sel0);  endmodule  //-------------------------------------------------------------------  module mux4to1(in0,in1,in2,in3,sel0,sel1,out);  input[7:0] in0,in1,in2,in3;  input sel0,sel1;  output[7:0] out;  wire[7:0] w1,w2;  mux2to1 m1(in0,in1,sel0,w1);  mux2to1 m2(in2,in3,sel0,w2);  mux2to1 m3(w1,w2,sel1,out);  endmodule  //-------------------------------------------------------------------  module mux\_8to1(in0,in1,in2,in3,in4,in5,in6,in7,sel0,sel1,sel2,out);  input[7:0] in0,in1,in2,in3,in4,in5,in6,in7;  input sel0,sel1,sel2;  output[7:0] out;  wire[7:0] w1,w2;  mux4to1 m1(in0,in1,in2,in3,sel0,sel1,w1);  mux4to1 m2(in4,in5,in6,in7,sel0,sel1,w2);  mux2to1 m3(w1,w2,sel2,out);  endmodule  //-------------------------------------------------------------------  module alu\_2(a, b, opSel, zeroF, carryF, negativeF, overflowF, carryFS, carryFA, result);  input [7:0] a, b;  input [2:0] opSel;  output zeroF, negativeF, carryF, overflowF, carryFS, carryFA;  output [7:0] result;  wire [7:0] result0, result1, result2, result3, result4, result5, result6, result7;  fulladder(a, b, 0, carryFA, result0);  eightbitsubstractor(a, b, 0, carryFS, result1);  assign result2 = a & b;  assign result3 = a | b;  assign result4 = a;  assign result5 = a << 1;  assign result6 = a >> 1;  assign result7 = a ^ b;  mux\_8to1 mux1(result0, result1,result2, result3, result4, result5, result6, result7, opSel[0], opSel[1], opSel[2], result);  assign zeroF = ~(result[7] | result[6] | result[5] | result[4] | result[3] | result[2] | result[1] | result[0]);  assign negativeF = result[7] ;  assign overflowF = (((~a[7] & ~b[7] & result[7]) | (a[7] & b[7] & ~result[7]))  & (~opSel[0] & ~opSel[1] & ~opSel[2])) | (((~a[7] & b[7] & result[7]) | (a[7] & ~b[7] & ~result[7]))  & (opSel[0] & ~opSel[1] & ~opSel[2]));    assign carryF = (~opSel[0] & ~opSel[1] & ~opSel[2] & carryFA) | (opSel[0] & ~opSel[1] & ~opSel[2] & carryFS);    endmodule  //------------------------------------------------------------------- |

**The RTL View**