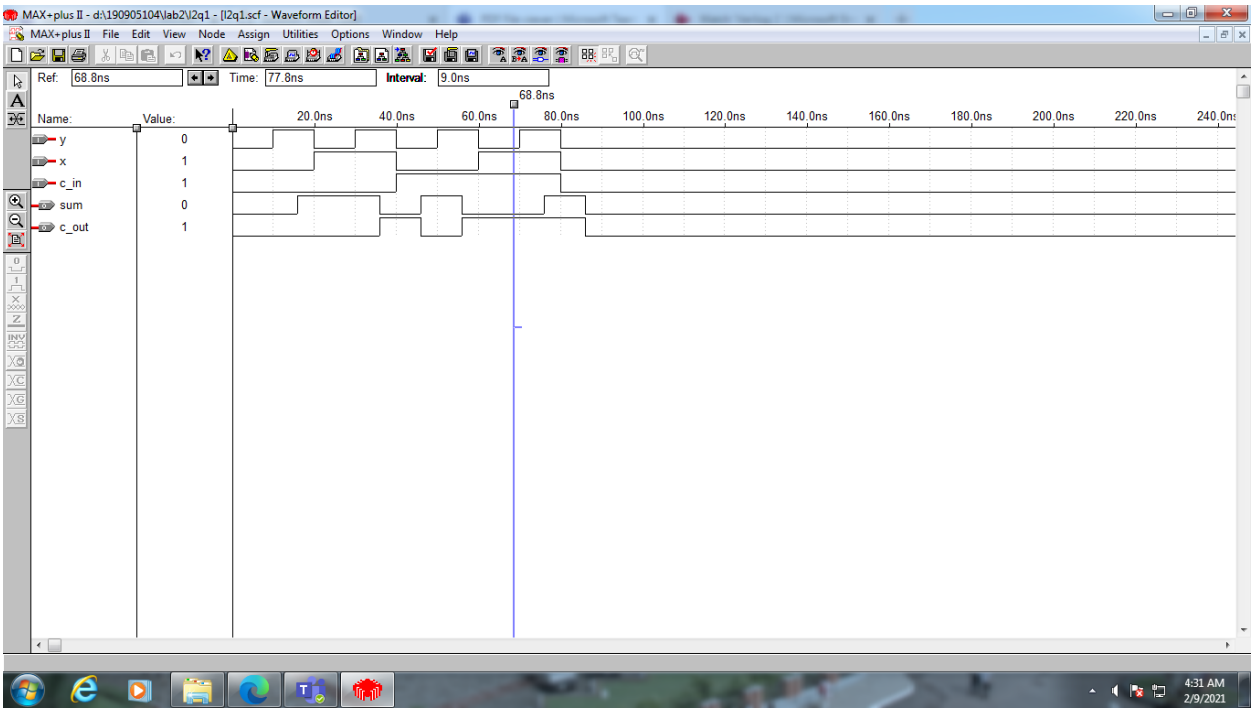
1) Write behavioral Verilog code to implement the following and simulate

// Full adder

```
module l2q1(c_in, x, y, sum, c_out);
input c_in, x, y;
output sum, c_out;
// Calculating sum
assign sum = c_in ^ x ^ y;
// calculating carry out = (x.y) + (y.c_in) + (c_in.x)
assign c_out = (x & y) | (y & c_in) | (c_in & x);
endmodule
```



2) Write behavioral Verilog code to implement the following and simulate

Four bit adder/subtractor

```
module full_adder(c_in, x, y, sum, c_out);
```

```
input c_in, x, y;
output sum, c_out;
// Calculating sum
assign sum = c_in ^ x ^ y;
// calculating carry out = (x.y) + (y.c_in) + (c_in.x)
assign c_out = (x & y) | (y & c_in) | (c_in & x);
endmodule

module l2q2(S, C, V, A, B, Op);
output [3:0] S; // The 4-bit sum/difference.
output C; // The 1-bit carry/borrow status.
output V; // The 1-bit overflow status.
input [3:0] A; // The 4-bit augend/minuend.
input [3:0] B; // The 4-bit addend/subtrahend.
input Op; // The operation: 0 => Add, 1=>Subtract.

wire C0; // The carry out bit of fa0, the carry in bit of fa1.
wire C1; // The carry out bit of fa1, the carry in bit of fa2.
wire C2; // The carry out bit of fa2, the carry in bit of fa3.
wire C3; // The carry out bit of fa2, used to generate final carry/borrrow.
wire B0;
wire B1;
wire B2;
wire B3;
xor(B0, B[0], Op);
xor(B1, B[1], Op);
xor(B2, B[2], Op);
xor(B3, B[3], Op);
xor(C, C3, Op); // Carry = C3 for addition, Carry = not(C3) for subtraction.
```

xor(V, C3, C2); // If the two most significant carry output bits differ, then we have an overflow.

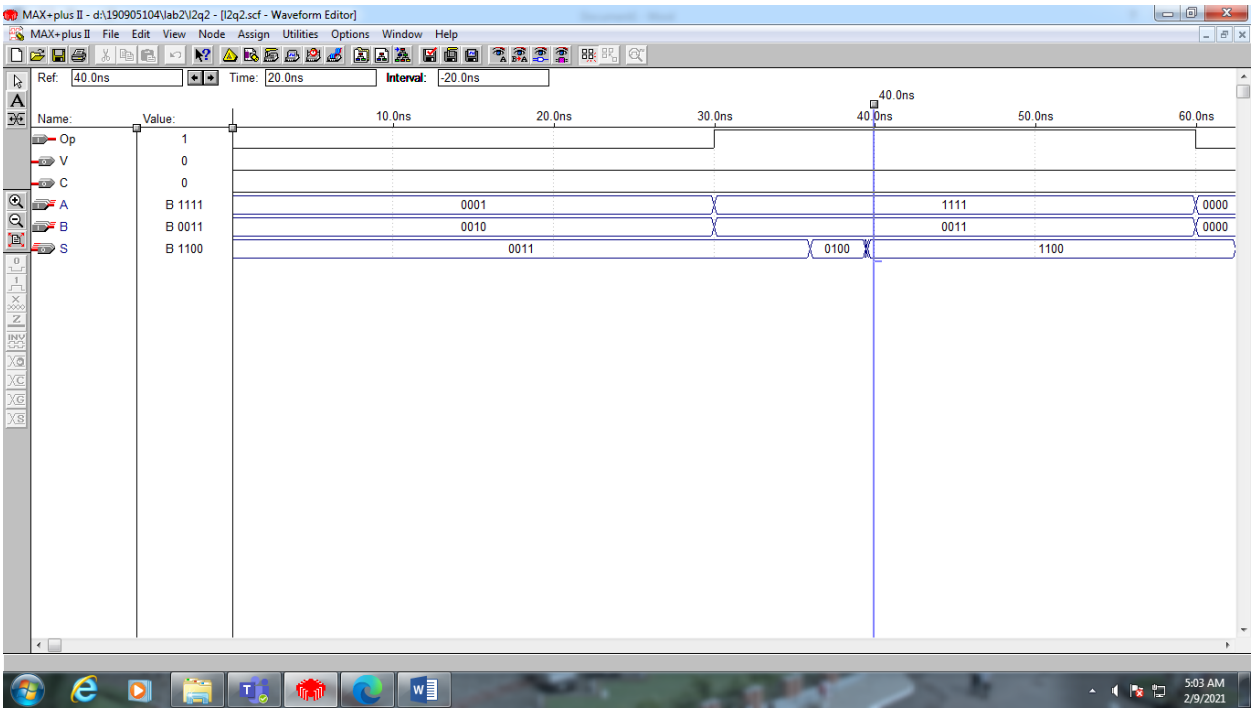
full_adder fa0(Op, A[0], B0, S[0], C0);

full_adder fa1(C0, A[1], B1, S[1], C1);

full_adder fa2(C1, A[2], B2, S[2], C2);

full_adder fa3(C2, A[3], B3, S[3], C3);

endmodule



3) Write behavioral Verilog code to implement the following and simulate Single-digit BCD adder using a four-bit adder(s).

// Single-digit BCD adder using a four-bit adder(s).

module halfadd(a, b, s, c);

```
input a, b;
output s, c;
assign s = a ^ b;
assign c = a & b;
endmodule

module fulladd(a, b, c_in, s, c_out);
input a, b, c_in;
output s, c_out;
wire w1,w2,w3;
halfadd ha1(a, b, w1, w2);
halfadd ha2(w1, c_in, s, w3);
assign c_out = w3 | w2;
endmodule

module l2q3(a, b, c_in, s, c_out);
input [3:0]a;
input [3:0]b;
input c_in;
output[3:0]s;
output c_out;

wire [3:0]w;
wire y, c0, c1, c2, c3, c4, c5;

fulladd fa1(a[0], b[0], c_in, w[0], c0);
fulladd fa2(a[1], b[1], c0, w[1], c1);
fulladd fa3(a[2], b[2], c1, w[2], c2);
fulladd fa4(a[3], b[3], c2, w[3], c3);
```

```
assign y = c3 | (w[3] & (w[2] | w[1]));

halfadd a1(w[1], y, s[1], c4);

halfadd a2(w[2], y, c4, s[2], c5);

halfadd a3(w[3], c5, s[3], c_out);

assign s[0] = w[0];

endmodule
```