ELC 2137 Lab 05: Verilog Intro

Abigail Joseph

October 1, 2020

Summary

In this lab, we made a half adder, full adder, and 2-bit adder subtractor in using Verilog.

Q&A

Q:Did the simulations match the expected output values?

A:Yes, the simulation values were the same as the expected outputs, as seen in the tables.

Q:What is one thing you still don't understand about Verilog?"

A:To be honest, I think all the things I don't know are things I don't know I don't know. I suppose the least clear thing for me is the difference between regs and wires in practice. I know how to use them, just not exactly how they're different.

Code

Listing 1: Half Adder

```
module halfadder(
    input a,
    input b,
    output c,
    output s
    );
    wire c, s;

   assign c= a & b;
   assign s= a ^ b;


endmodule
```

Listing 2: Half Adder Test

```
module halfadder_test();

reg a, b;
wire c, s;
```

```
halfadder dut(
    .a(a),
    .b(b),
    .c(c),
    .s(s)
    );

    initial begin
        a=0; b=0; #10;
        a=0; b=1; #10;
        a=1; b=0; #10;
        a=1; b=1; #10;

    $finish;
     end


endmodule
```

Listing 3: Full Adder

```
module fulladder(
    input a_in,
    input b_in,
    input c_in,
    output c_out,
    output s_out
    );
    wire c1, c2, s1;

    halfadder ha0(
        .a(a_in),
        .b(b_in),
        .c(c1),
        .s(s1)
        );

    halfadder ha1(
        .a(s1),
        .b(c_in),
        .c(c2),
        .s(s_out)
        );


    assign c_out = c1 | c2;



endmodule
```

Listing 4: Full Adder Test

```
module fulladder_test();
```

```
reg c_in, a_in, b_in;
wire c_out, s_out;

fulladder dut(
    .a_in(a_in),
    .b_in(b_in),
    .c_in(c_in),
    .c_out(c_out),
    .s_out(s_out)
    );

    initial begin
        c_in=0; a_in=0; b_in=0; #10;
        c_in=0; a_in=0; b_in=1; #10;
        c_in=0; a_in=1; b_in=0; #10;
        c_in=0; a_in=1; b_in=1; #10;
        c_in=1; a_in=0; b_in=0; #10;
        c_in=1; a_in=0; b_in=1; #10;
        c_in=1; a_in=1; b_in=0; #10;
        c_in=1; a_in=1; b_in=1; #10;

        $finish;
    end




endmodule
```

Listing 5: Adder-Subtractor

```
module addsub(
    input [1:0] a,b,
    input m,
    output c_out,
    output [1:0] sum
    );

    wire c1, c2;
    wire [1:0] b_i;

    assign b_i[0] = b[0] ^ m;
    assign b_i[1] = b[1] ^ m;


    fulladder fa0(
    .a_in(a[0]),
    .b_in(b_i[0]),
    .c_in(m),
    .c_out(c1),
    .s_out(sum[0])
    );
```

```
    fulladder fa1(
    .a_in(a[1]),
    .b_in(b_i[1]),
    .c_in(c1),
    .c_out(c2),
    .s_out(sum[1])
    );

    assign c_out = m ^ c2;


endmodule
```

Listing 6: Half Adder

```
module addsub_test();

reg [1:0] a,b;
reg m;
wire [1:0] sum;
wire c_out;

addsub dut(
    .a(a),
    .b(b),
    .m(m),
    .c_out(c_out),
    .sum(sum)

    );



initial begin
   m=0; a[1]= 0; a[0]= 0; b[1]= 0; b[0]= 0; #10
   m=0; a[1]= 0; a[0]= 0; b[1]= 0; b[0]= 1; #10
   m=0; a[1]= 0; a[0]= 0; b[1]= 1; b[0]= 0; #10
   m=0; a[1]= 0; a[0]= 0; b[1]= 1; b[0]= 1; #10
   m=0; a[1]= 0; a[0]= 1; b[1]= 0; b[0]= 1; #10
   m=0; a[1]= 1; a[0]= 0; b[1]= 0; b[0]= 1; #10
   m=0; a[1]= 1; a[0]= 0; b[1]= 0; b[0]= 0; #10
   m=1; a[1]= 0; a[0]= 0; b[1]= 0; b[0]= 0; #10
   m=1; a[1]= 0; a[0]= 0; b[1]= 0; b[0]= 1; #10
   m=1; a[1]= 0; a[0]= 0; b[1]= 1; b[0]= 0; #10
   m=1; a[1]= 0; a[0]= 0; b[1]= 1; b[0]= 1; #10
   m=1; a[1]= 0; a[0]= 1; b[1]= 0; b[0]= 1; #10
   m=1; a[1]= 1; a[0]= 0; b[1]= 0; b[0]= 1; #10
   m=1; a[1]= 1; a[0]= 0; b[1]= 0; b[0]= 0; #10

  $finish;

end
```

```
endmodule
```

Results

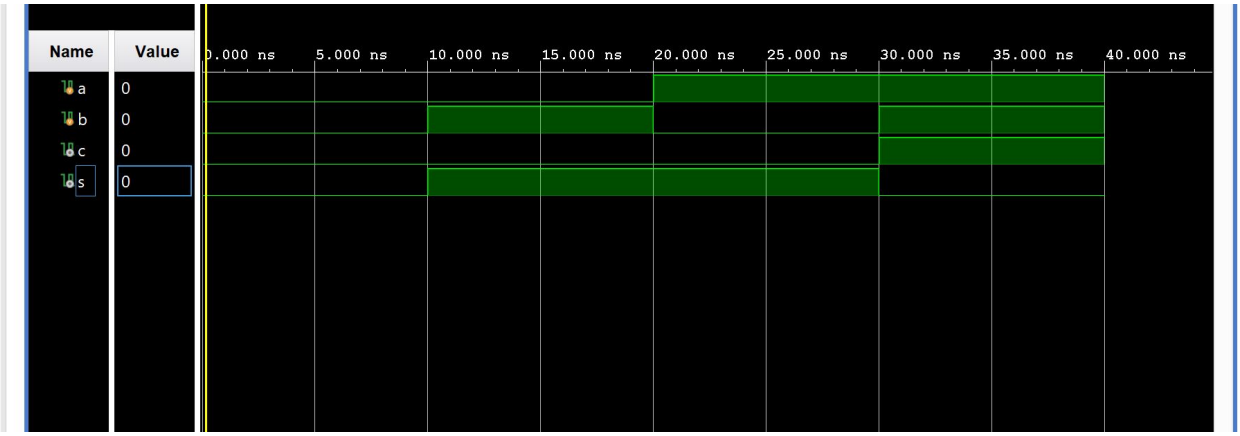

Figure 1: Half Adder Waveform

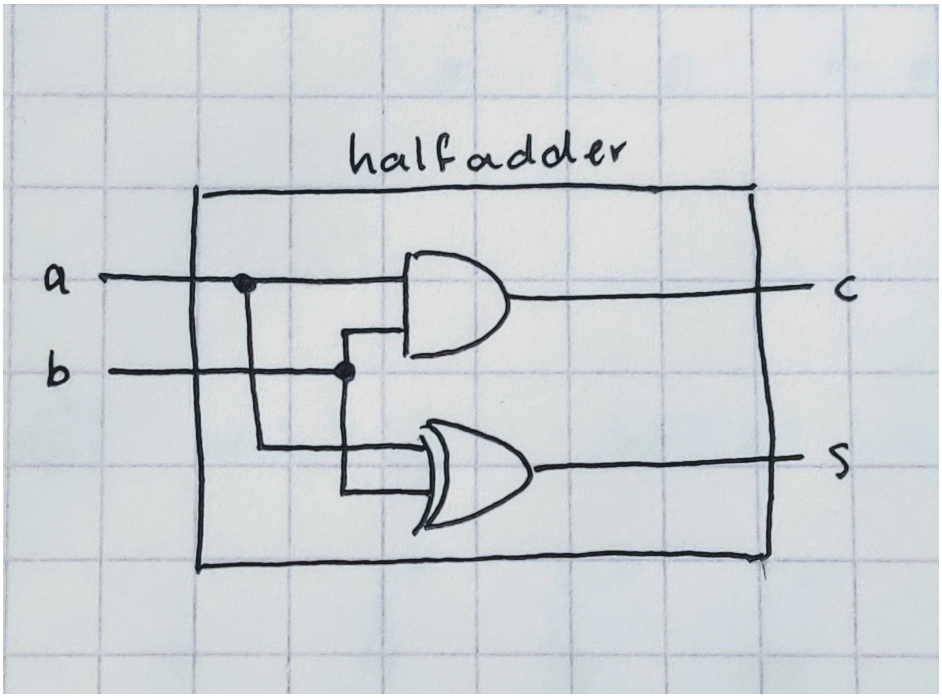

Figure 2: Half Adder Diagram

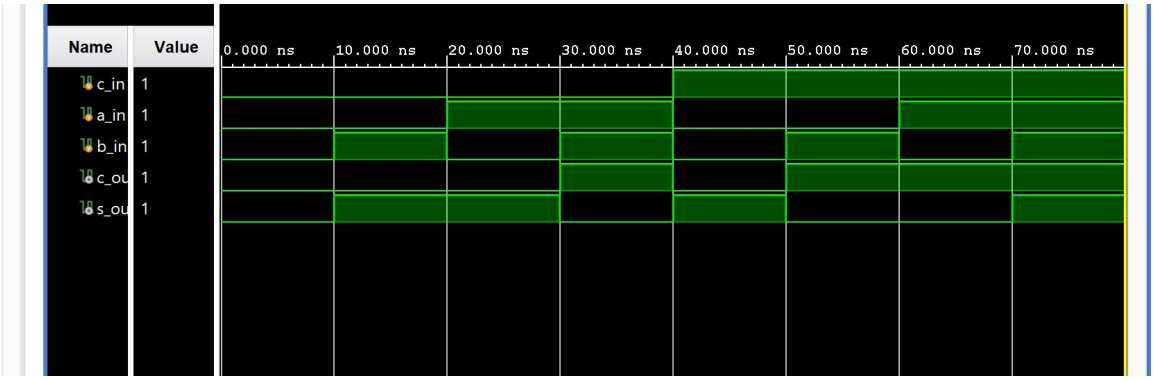

Figure 3: Full Adder Waveform

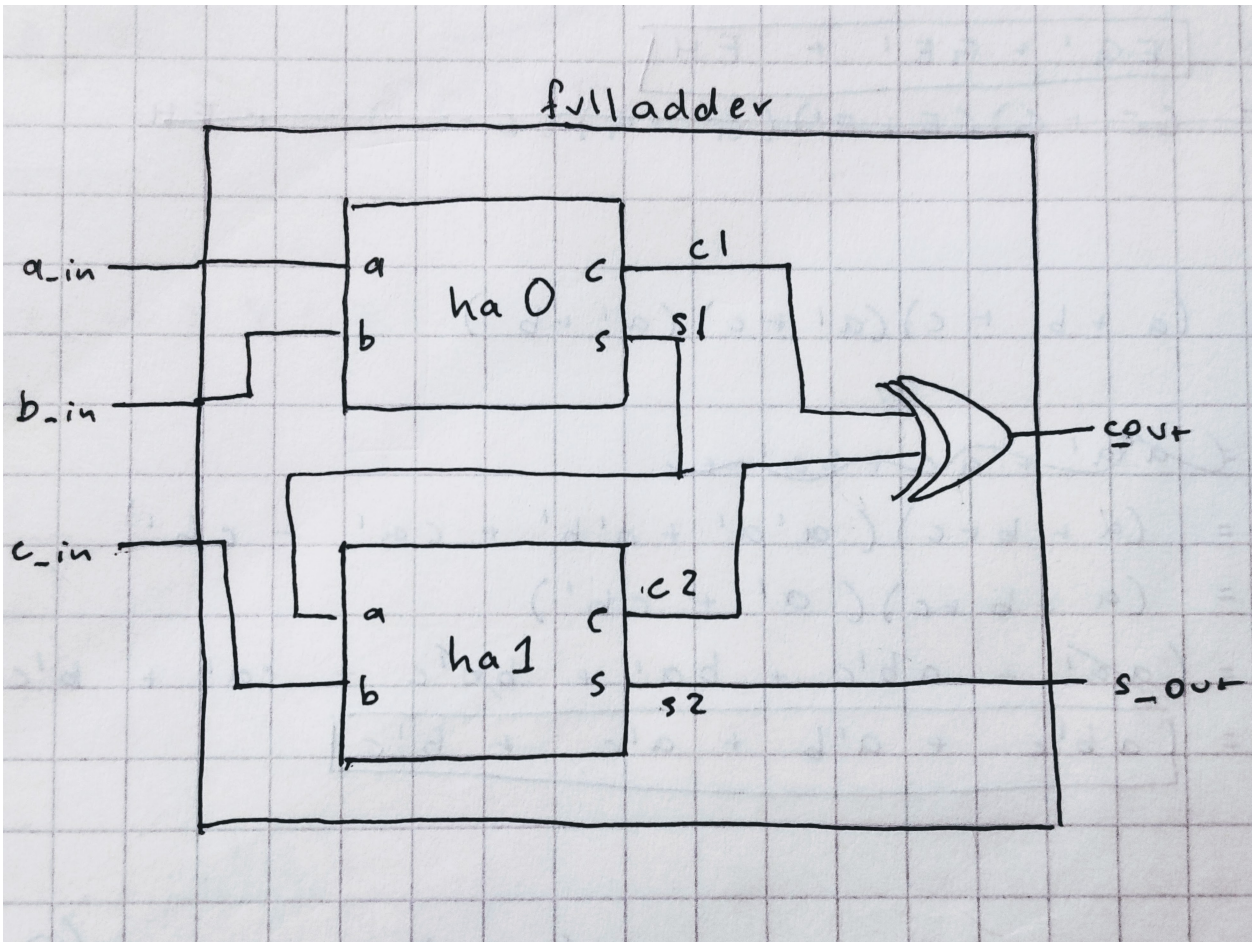

Figure 4: Full Adder Diagram

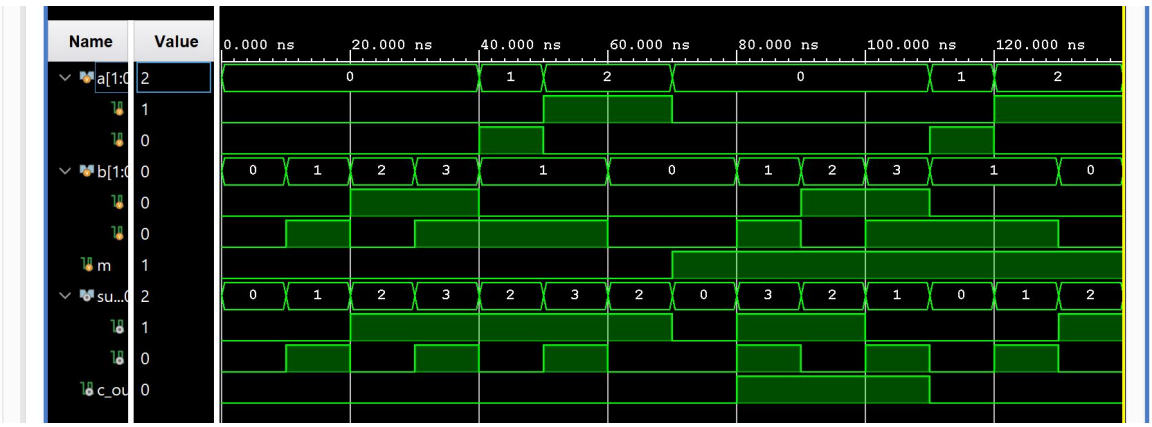

Figure 5: Adder/Subtractor Waveform

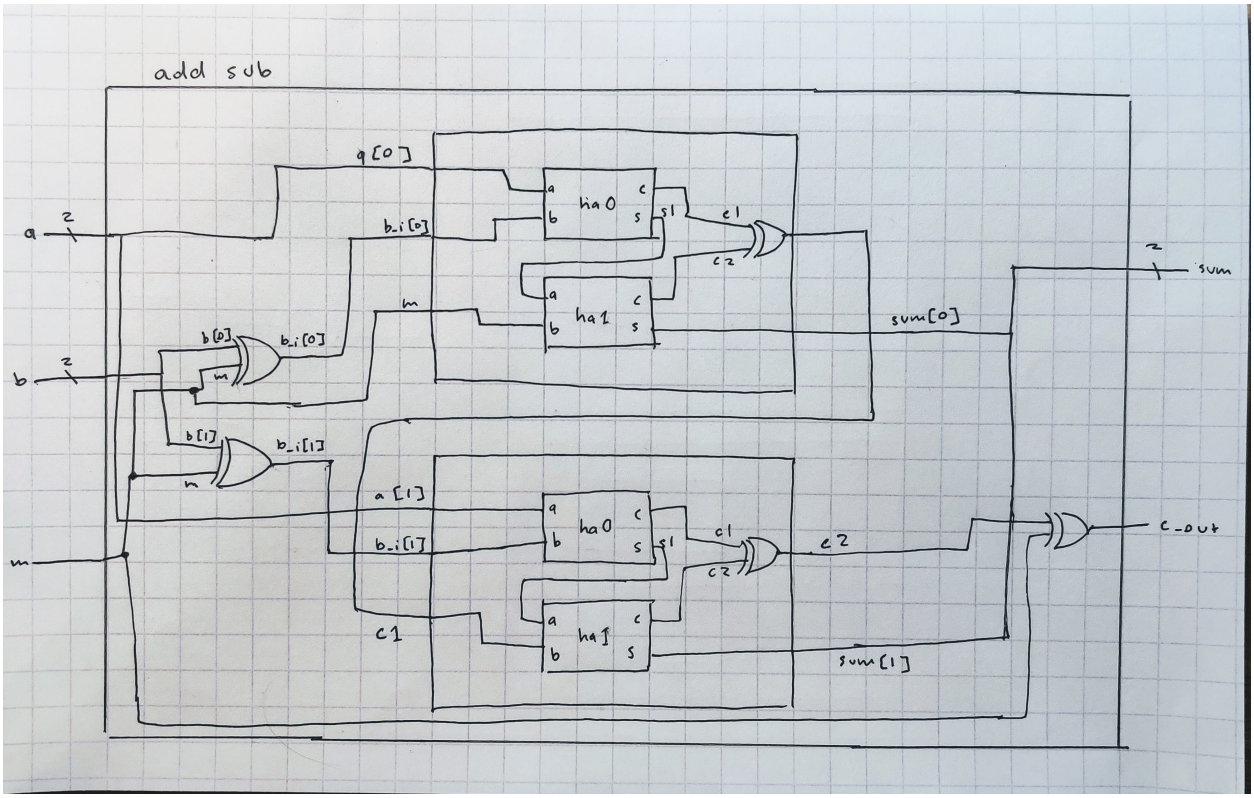

Figure 6: Adder Subtractor Diagram