ELC 2137 Lab 5: Intro to Verilog

Aaron Mendoza

September 27, 2020

Summary

The purpose of this lab is to familiarize myself with Verilog coding by building a half adder, full adder, and 2 bit adder/subtractor on Vivado.

There are three different types of coding styles: gate-level, functional/behavioral, and structural. Gate-level coding utilizes the actual gates within the code. This is done by literally typing out the exact gate like "xor" within the code. Behavioral coding is different from gate-level because the gate is not actually specified, but rather the behavior of the gate is achieved through the code that is written. This utilizes symbols like the ampersand. Structural coding is code that works based off of other code that has already been previously written. For example, the full adder code uses the half adder code within it.

I used behavorial coding when building my half adder, full adder, and 2 bit adder/subtractor. I used structural coding to build my full adder and my 2 bit adder/subtractor.

Block Diagrams

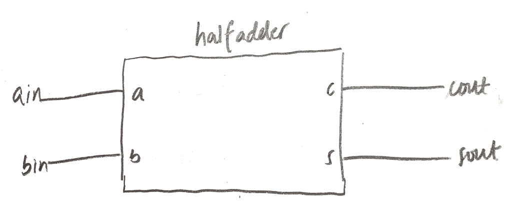

Figure 1: This is the block diagram for a half adder.

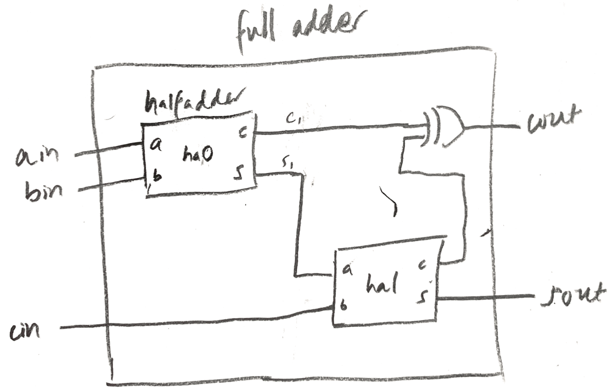

Figure 2: This is the block diagram for a full adder.

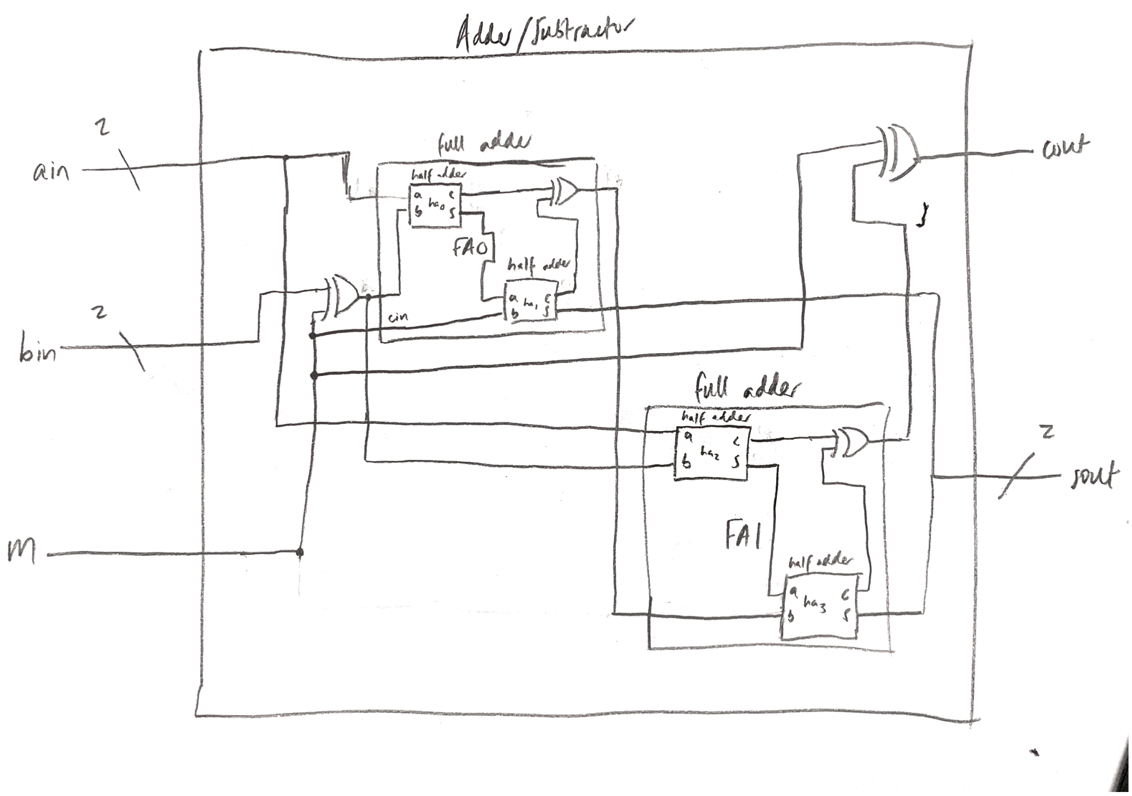

Figure 3: This is the block diagram for an adder/subtractor.

Q&A

1. Do the simulations match the expected output values?

   Yes the simulations match the expected output values.

2. What is one thing that you still don't understand about Verilog?

   I still don't completely understand the "always" block.

Results

Time (ns):	0	10	20	30
a	0	0	1	1
b	0	1	0	1
c	0	0	0	1
s	0	1	1	0

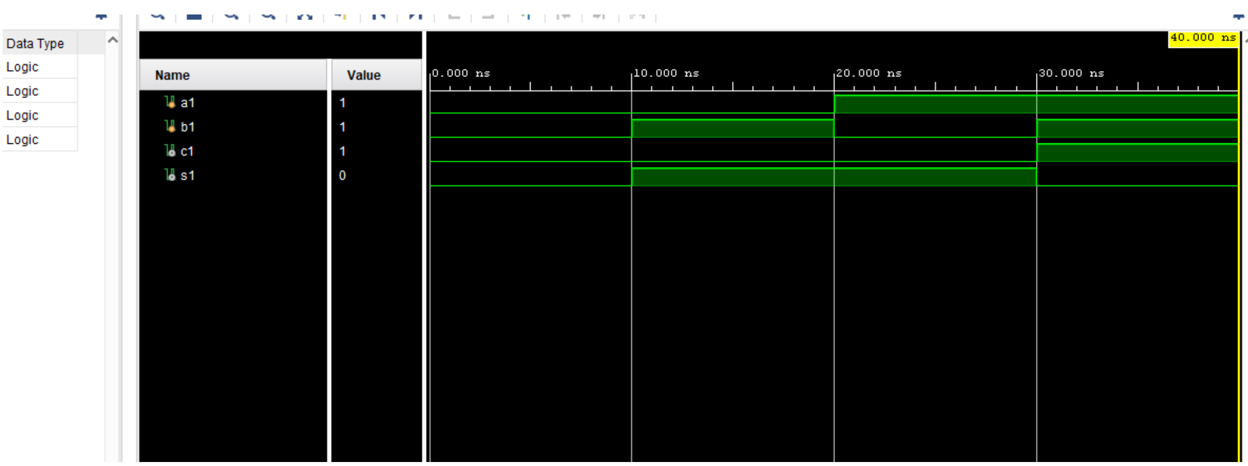

Figure 4: Half adder simulation waveform and ERT

Time (ns):	0	10	20	30	40	50	60	70
a	0	0	0	0	1	1	1	1
b	0	0	1	1	0	0	1	1
cin	0	1	0	1	0	1	0	1
cout	0	0	0	1	0	1	1	1
sout	0	1	1	0	1	0	0	1

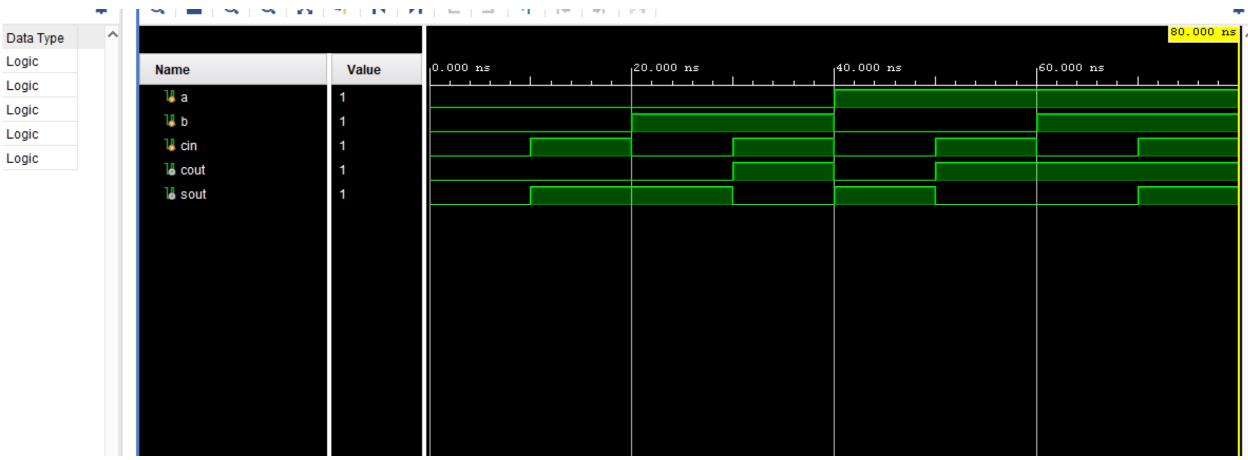

Figure 5: Full adder simulation waveform and ERT

Time (ns):	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180
a	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0
b	0	0	1	1	0	0	1	1											
m	0	1	0	1	0	1	0	1											
cout	0	0	0	1	0	1	1	1											
sout	0	1	1	0	1	0	0	1											

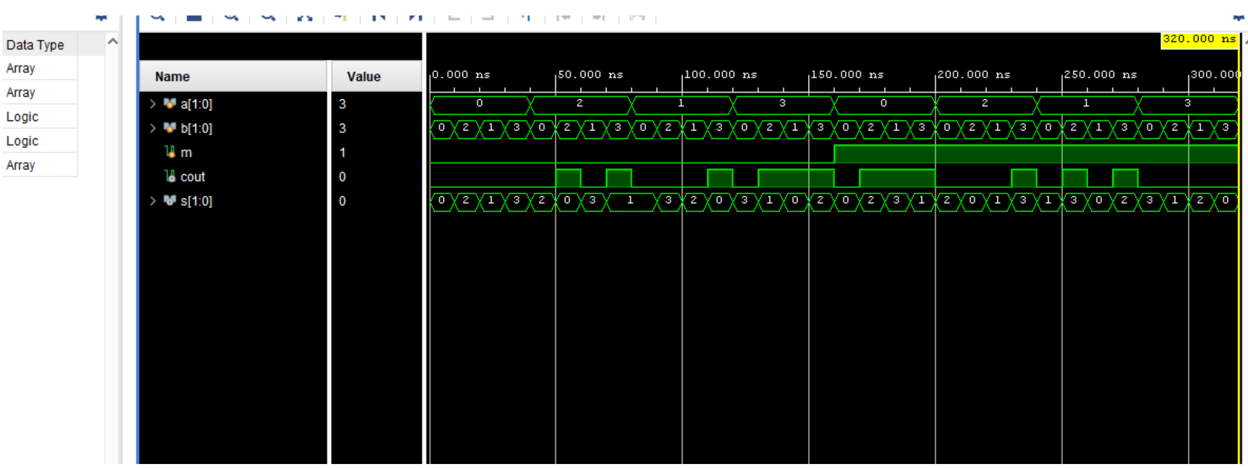

Figure 6: Adder/Subtractor simulation waveform and ERT

Code

Listing 1: Half Adder Verilog code

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

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

—

Listing 2: Half Adder Test Bench Verilog code

```
module halfadder_test();
    reg a1;
    reg b1;
    wire c1;
    wire s1;


halfadder dut(
    .a(a1),
    .b(b1),
    .c(c1),
    .s(s1)
    );

initial begin
a1=0;b1=0;#10;
a1=0;b1=1;#10;
a1=1;b1=0;#10;
a1=1;b1=1;#10;
$finish;
end

endmodule
```

—

Listing 3: Full Adder Verilog code

```
module fulladder(
    input a,
    input b,
    input cin,
    output cout,
```

```
    output sout
    );

wire s1, c1, c2;

halfadder HA1(
    .a(a),
    .b(b),
    .c(c1),
    .s(s1)
    );

halfadder HA2(
    .a(cin),
    .b(s1),
    .c(c2),
    .s(sout)
    );

assign cout = c1 ^ c2;

endmodule
```

—

Listing 4: Full Adder Test Bench Verilog code

```
module fulladder_test();
    reg a;
    reg b;
    reg cin;
    wire cout;
    wire sout;

fulladder dut(
    .a(a),
    .b(b),
    .cin(cin),
    .cout(cout),
    .sout(sout)
    );

initial begin
a = 0; b = 0; cin = 0; #10;
a = 0; b = 0; cin = 1; #10;
a = 0; b = 1; cin = 0; #10;
a = 0; b = 1; cin = 1; #10;
a = 1; b = 0; cin = 0; #10;
a = 1; b = 0; cin = 1; #10;
a = 1; b = 1; cin = 0; #10;
a = 1; b = 1; cin = 1; #10;
$finish;
end
```

```
endmodule
```

—

Listing 5: Adder/Subtractor Verilog code

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

wire [1:0]B;
wire [1:0]c;

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

fulladder FA1(
    .a(a[0]),
    .b(B[0]),
    .cin(m),
    .cout(c[0]),
    .sout(s[0])
    );

fulladder FA2(
    .a(a[1]),
    .b(B[1]),
    .cin(c[0]),
    .cout(c[1]),
    .sout(s[1])
    );

assign cout = c[1] ^ m;

endmodule
```

—

Listing 6: Adder/Subtractor Test Bench Verilog code

```
module addsub_test();

reg [1:0]a;
reg [1:0]b;
reg m;
wire cout;
wire [1:0]s;

addsub dut(
```

```
    .a(a),
    .b(b),
    .m(m),
    .s(s),
    .cout(cout)
    );

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

endmodule
```