ELC 2137 Lab 05: Verilog Intro

My Nguyen

September 30, 2020

Summary

This lab purpose is to learn the basic Verilog syntax, organize files and folder structure to recreate half-adder, full-adder, and 2-bit adder/subtractor. First, familiarize by creating a RTL project using Basys3 board. From this create a half-adder file and populate it with code and logic for a half-adder, then create a test file to exhaustively test it. Repeat this process for full-adder and 2-bit adder/subtractor.

Table and Figure

Code

Listing 1: Half Adder Implementation

```
module half_adder(
    input a,
    input b,
    output c,
    output s
    );
    assign c = a & b; // can't use "s" to caulculate "c
    assign s = a ^ b; // everything is one directional
endmodule
```

Listing 2: Half Adder Test Bench

```
module half_adder_test();
    //input -> reg , ouput -> wire
    reg a;
    reg b;
    wire c;
    wire s;

    half_adder dut(
        .a(a),
        .b(b),
        .c(c),
        .s(s)
        );
    initial begin
        a=0; b=0; #10;
        a=1; b=0; #10;
        a=0; b=1; #10;
```

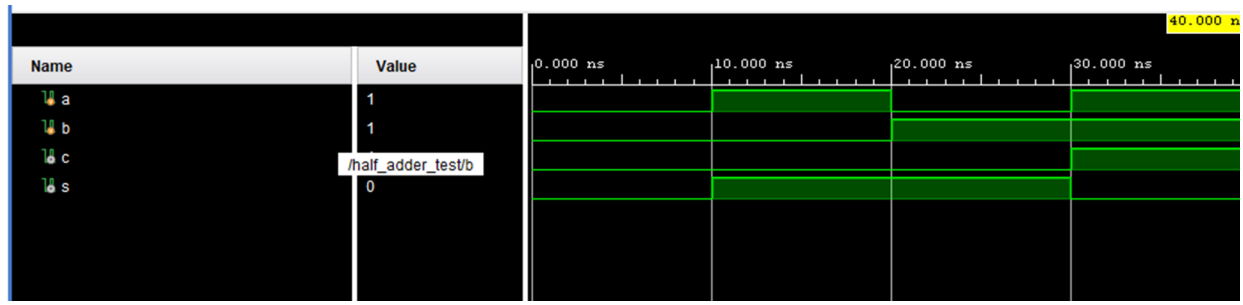

Figure 1: Half Adder ERT

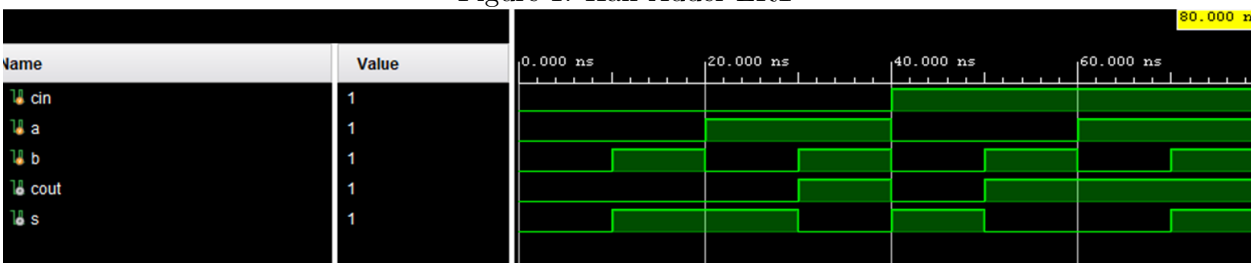

Figure 2: Full Adder ERT

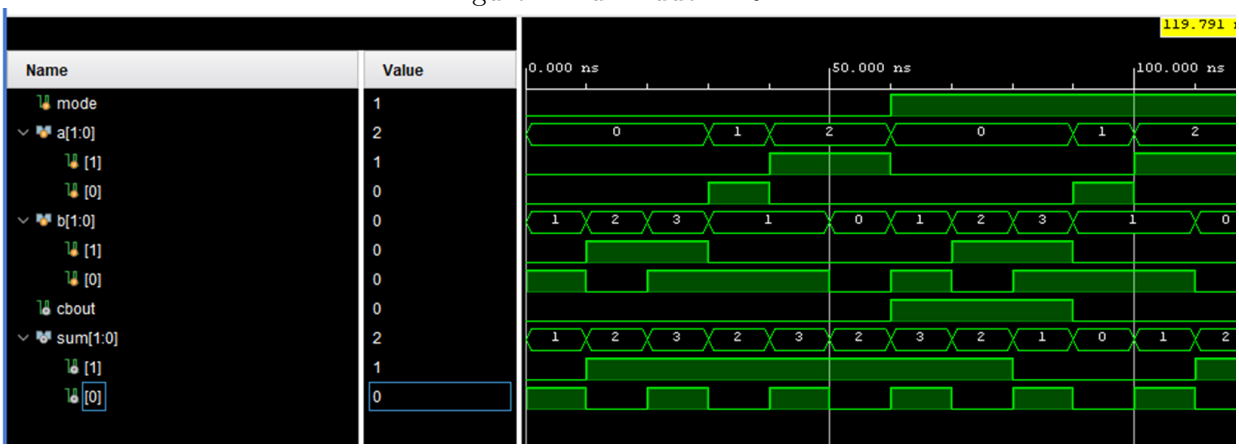

Figure 3: 2-bit Adder/Subtractor ERT

```
        a=1; b=1; #10;

        $finish;
    end
endmodule
```

Listing 3: Full Adder Implementation

```
module full_adder(
    input a,
    input b,
    input cin,
    output cout,
    output s
    );
```

```
    //internal signals
    wire c1, c2, s1;

    //first half adder
    half_adder ha0(
        .a(a),
        .b(b),
        .c(c1),
        .s(s1)
    );

    //second half adder
    half_adder ha1(
        .a(s1),
        .b(cin),
        .c(c2),
        .s(s)
    );

    //the last xor gate connecting cin and the second carry
    assign cout = c1 ^ c2;

endmodule
```

Listing 4: Full Adder Test Bench

```
module full_adder_test();
    reg cin;
    reg a;
    reg b;
    wire cout;
    wire s;

    //initilize a full adder for testing
    full_adder dut(
        .a(a),
        .b(b),
        .cin(cin),
        .cout(cout),
        .s(s)
    );

    //test cases
    initial begin
        cin = 0; a = 0; b = 0; #10;
        cin = 0; a = 0; b = 1; #10;
        cin = 0; a = 1; b = 0; #10;
        cin = 0; a = 1; b = 1; #10;
        cin = 1; a = 0; b = 0; #10;
        cin = 1; a = 0; b = 1; #10;
        cin = 1; a = 1; b = 0; #10;
        cin = 1; a = 1; b = 1; #10;
        $finish;
```

```
    end
endmodule
```

Listing 5: 2-bit Adder/Subtractor Implementation

```
module adder_subtractor(
    input [1:0] a, b,
    input mode,
    output cbout,
    output [1:0] sum
    );

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

    //Invert b input for subtraction
    assign b_add[0] = b[0] ^ mode;
    assign b_add[1] = b[1] ^ mode;

    //the first full adder
    full_adder fa0(
        .a(a[0]),
        .b(b_add[0]),
        .cin(mode),
        .cout(c1),
        .s(sum[0])
    );

    //the second full adder
    full_adder fa1(
        .a(a[1]),
        .b(b_add[1]),
        .cin(c1),
        .cout(c2),
        .s(sum[1])
    );

    //Convert carry to borrow when subtracting
    assign cbout =  c2 ^ mode;
endmodule
```

Listing 6: 2-bit Adder/Subtractor Test Bench

```
module adder_subtractor_test();
    reg [1:0] a, b;
    reg mode;
    wire [1:0] sum;
    wire cbout;

    adder_subtractor dut(
        .a(a),
        .b(b),
        .mode(mode),
        .cbout(cbout),
```

```
        .sum(sum)
        );

    //test cases for addition and subtraction
    initial begin
       mode = 0; a[1] = 0; a[0] = 0; b[1] = 0; b[0] = 1; #10;
       mode = 0; a[1] = 0; a[0] = 0; b[1] = 1; b[0] = 0; #10;
       mode = 0; a[1] = 0; a[0] = 0; b[1] = 1; b[0] = 1; #10;
       mode = 0; a[1] = 0; a[0] = 1; b[1] = 0; b[0] = 1; #10;
       mode = 0; a[1] = 1; a[0] = 0; b[1] = 0; b[0] = 1; #10;
       mode = 0; a[1] = 1; a[0] = 0; b[1] = 0; b[0] = 0; #10;

       mode = 1; a[1] = 0; a[0] = 0; b[1] = 0; b[0] = 1; #10;
       mode = 1; a[1] = 0; a[0] = 0; b[1] = 1; b[0] = 0; #10;
       mode = 1; a[1] = 0; a[0] = 0; b[1] = 1; b[0] = 1; #10;
       mode = 1; a[1] = 0; a[0] = 1; b[1] = 0; b[0] = 1; #10;
       mode = 1; a[1] = 1; a[0] = 0; b[1] = 0; b[0] = 1; #10;
       mode = 1; a[1] = 1; a[0] = 0; b[1] = 0; b[0] = 0; #10;

       $finish;
    end
endmodule
```

Screenshot

Questions

4. The simulations matches exact the result from lab 3 and lab 4. 5. One thing I still cannot figure out is how to assign value to a multi-bit variable using another multi-bit variable.

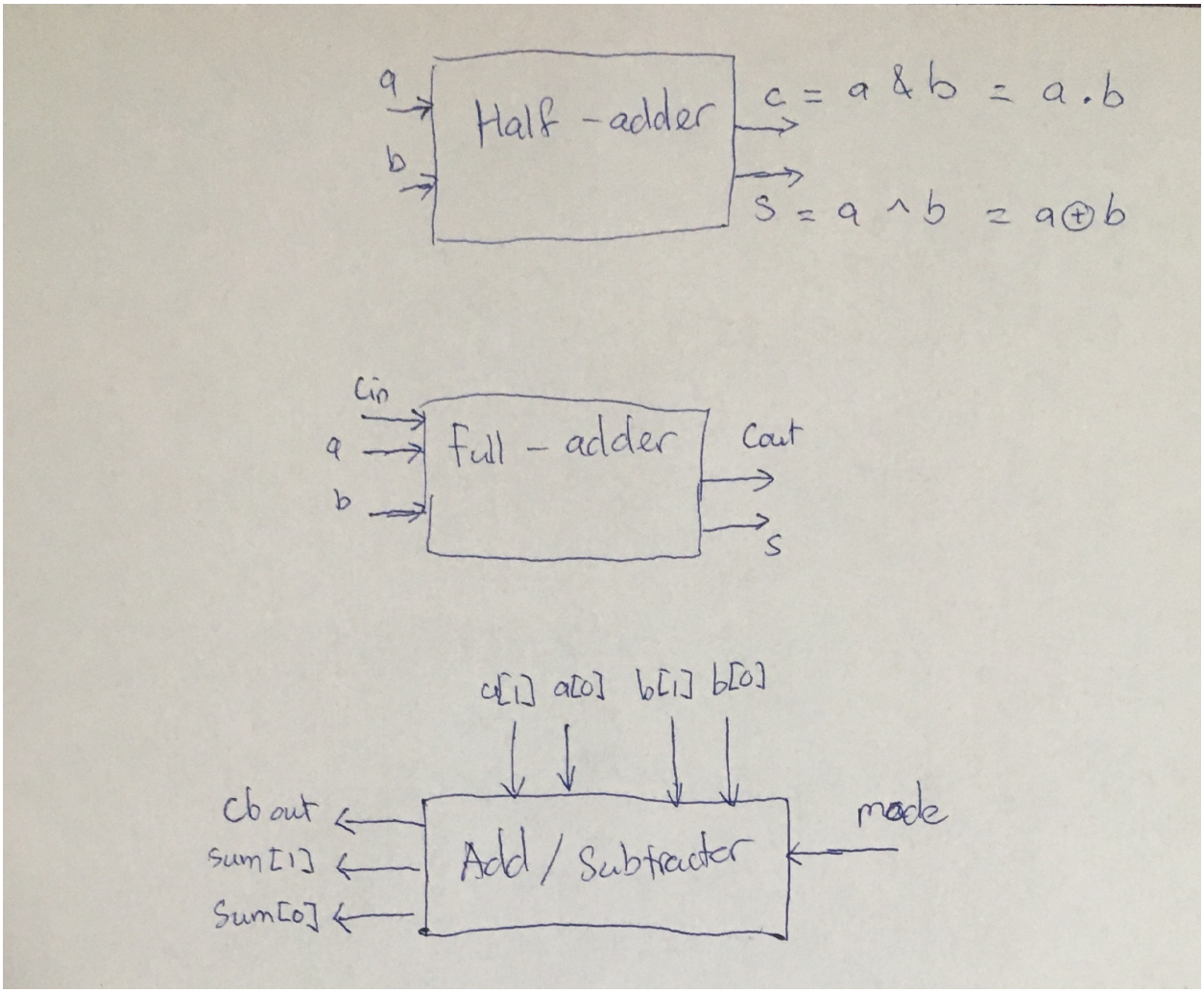

Figure 4: Block Diagrams