Experiment 2 Lab Report

EEE3342C - 00012

Yousef Awad

January 2025

Contents

Equipment

For this experiment a computer running Linux 6.12.13 was used alongside the Xilinx Vivado 2024.2 software, alongside an FPGA board, the BASYS 3 development board. The board specifically only used to ensure the simulation by the Vivado software was accurate in the real world, as well as to verify the simulation software wasn't incorrect.

Objective

Part 1: Half/Full Adder

Half Adder

The circuit given for this part was the following:

$$\neg[(A \wedge B) \vee \neg(C \vee D)]$$

Of which has the following given schematic: And when compiled into a truth

Figure 1: Schematic for Half Adder

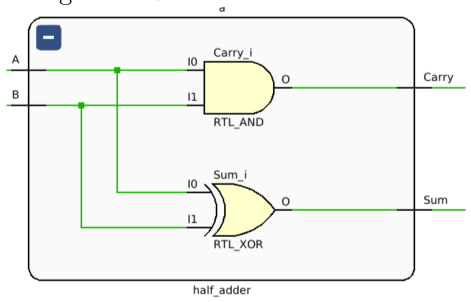


table with the inputs of A, B, and an output of Carry, would be the following:

Truth Table for Half Adder

A	B	$Output$	$Carry$
F	F	F	F
F	T	T	F
T	F	T	F
T	T	F	T

And when written up in verilog, has the following text: And when tested, used the following testbench, after reading through part 1 of the experiment manual:

```
module testbench();
    parameter numin = 2;
```

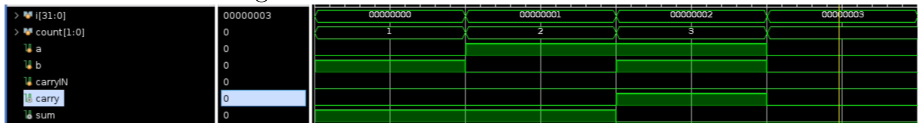
```
module half_adder(
      input A, B,
      output Sum, Carry
  );

  assign Sum = A ^ B;
  assign Carry = A & B;

endmodule
```

```
    integer i;
    reg[numin - 1:0] count;

    reg a, b;
    wire carry, sum;

    half_adder UUT(.A(a), .B(b), .Carry(carry), .Sum(sum));


    initial begin
        count = 0;
        for ( i = 0; i < 2**numin; i = i + 1 ) begin
            assign a = count[1];
            assign b = count[0];

            count = count + 1;
            #10;
        end
    end

endmodule
```

Listing 1: test

And, when simulated to confirm the truth table above to be true or false, it gave out the following waveform: Of which perfectly shows that the truth table

Figure 2: Waveform for the Half-Adder

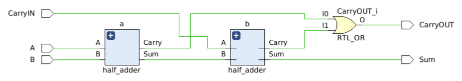


compiled above for the circuit is accurately shown in the simulation on Vivado. To ensure, even further, I then pushed the bitstream generated onto the BASYS board to manually enter and double check the simulation/truth table proper by flicking every possible combination.

Full Adder

The circuit schematic given for this part was the following: And when written

Figure 3: Schematic for Full Adder



up in verilog, has the following text:

```
module full_adder(
        input A, B, CarryIN,
        output Sum, CarryOUT
    );

    wire carry1, carry2, sum1;

    half_adder a(.A(A), .B(B), .Carry(carry1), .Sum(sum1));
    half_adder b(.A(CarryIN), .B(sum1), .Sum(Sum), .Carry(
        carry2));

    assign CarryOUT = carry1 | carry2;

endmodule
```

Listing 2: test

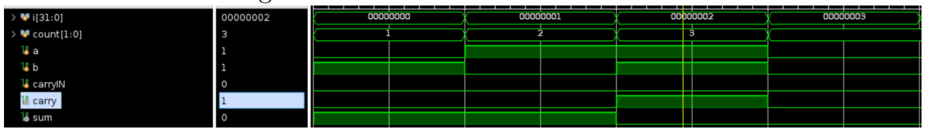
And when tested, used the following testbench, after reading through part 1 of the experiment manual:

```
module testbench();
    parameter numin = 2;
    integer i;
    reg[numin - 1:0] count;

    reg a, b, carryIN;
    wire carry, sum;

    full_adder UUT(.A(a), .B(b), .CarryIN(carryIN), .
        CarryOUT(carry), .Sum(sum));


    initial begin
        count = 0;
        for ( i = 0; i < 2**numin; i = i + 1 ) begin
            assign a = count[1];
            assign b = count[0];
            assign carryIN = 0;

            count = count + 1;
            #10;
        end
    end

endmodule
```

Listing 3: test

And, when simulated, it gave out the following correct waveform: To ensure,

Figure 4: Waveform for the Half-Adder

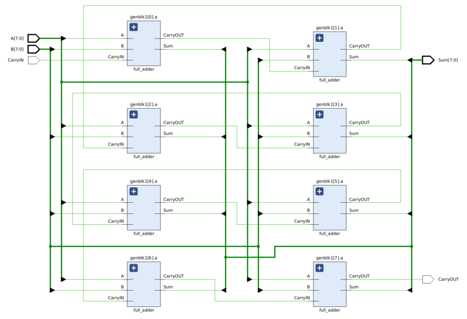


even further past the given truth table in lab report, I then pushed the bit-stream generated onto the BASYS board to manually enter and double check the simulation proper by flicking every possible combination.

Part 1: 8-Bit Ripple Carry Adder

The schematic was designed as following: And when written up in verilog, has

Figure 5: Schematic for 8 Bit Ripple Carry Adder



the following text:

```
module eight_bit_ripple_adder(
        input[7:0] A, B,
        input CarryIN,
        output[7:0] Sum,
        output CarryOUT
    );

    wire[8:0] carry;

    assign carry[0] = CarryIN;

    genvar i;

    generate
        for ( i = 0; i < 8; i = i + 1 ) begin
            full_adder a(.A(A[i]), .B(B[i]), .CarryIN(carry[
                i]), .CarryOUT(carry[i + 1]), .Sum(Sum[i]));
        end
    endgenerate

    assign CarryOUT = carry[8];

endmodule
```

Listing 4: test

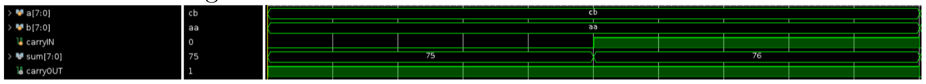
And when tested, used the following testbench, after reading through part 1 of the experiment manual:

```
module testbench_part2();
    parameter numin = 8;

    reg[numin - 1:0] a, b;
    reg carryIN;
    wire[numin - 1:0] sum;
    wire carryOUT;

    eight_bit_ripple_adder UUT(.A(a), .B(b), .CarryIN(
        carryIN), .CarryOUT(carryOUT), .Sum(sum));

    initial begin
        assign a = 8'b11001011;
        assign b = 8'b10101010;
        assign carryIN = 0;
        #10;
        assign carryIN = 1;
        #10
        assign a = 0;
        assign b = 0;
        assign carryIN = 0;
    end

endmodule
```

Listing 5: test

And, when simulated to confirm the addition of 170 and 203 (assuming it is unsigned) above to be 75 Carry 1 as it overflows and switches sign, it gave out the following waveform: Of which perfectly shows the correct result to be 75 or

Figure 6: Waveform for Addition of 170 and 203



76 of which is in binary either 01001011 or 01001100, depending on the carry in value with a carry out of 1 meaning it overflowed.

Conclusion