EE2016: Experiment 1

Sparsh Gupta
EE23B117

Charan Srikkanth
EE23B127

Kaushik Iyer
EE23B135

August 07, 2024

Abstract

In this experiment we were asked to get familiar with the software ‘Xilinx Vivado‘ and use it along with an FPGA to realize the circuits defined below.

1 Half Adder

A half adder is a simple circuit that adds 2 1-bit numbers and outputs the sum and carry bits. In the module defined below, ‘a‘ and ‘b‘ are the input bits while ‘sum‘ and ‘carry‘ are the outputs to the addition. The relations are given by:

```
sum = a XOR b;
carry = a AND b;
```

1.1 Verilog Implementation

```verilog
// half_adder.v
module half_adder(
    a, b, sum, carry
    );
        input a, b;
        output sum, carry;
        xor(sum, a, b);
        and(carry, a, b);
endmodule
```

1.2 Testbench

```verilog
// half_adder_tb.v
module half_adder_tb();
    reg a, b;
    wire sum, carry;
    half_adder ha(a, b, sum, carry );

    initial begin
        a = 0; b = 0; // Expected 0, 0
        #10
        a = 1; // Expected 1, 0
        #10
        b = 1; // Expected 0, 1
        #10
        a = 0; // Expected 1, 0
        #10
        $stop;
    end
endmodule
```

2 Full Adder

A full adder is similar to the half adder. It adds 3 1-bit numbers and outputs the sum and carry bits. In the module defined below, ‘a‘, ‘b‘ and ‘cin‘ are the input bits while ‘sum‘ and ‘carry‘ are the outputs to the addition. The relations are given by:

```
sum = a XOR b XOR cin;
carry = (a AND b) OR (a AND cin) OR (b AND cin);
```

2.1 Verilog Implementation

```
// full_adder.v
module full_adder(
    a, b, cin, sum, cout
    );
    input a, b, cin;
    output sum, cout;
    wire w1, w2, w3, w4, w5; // Intermediate wires used for connecting the circuit

    // NOTE: The gate names are optional :)
    xor x1(w1, a, b);
    xor x2(sum, w1, cin);
    and x3(w2, a, b);
    and x4(w3, a, cin);
    and x5(w5, b, cin);
    or x6(w4, w2, w3);
    or x7(cout, w4, w5);
endmodule
```

2.2 Testbench

```
// full_adder_tb.v
module full_adder_tb();
    reg a, b, cin;
    wire sum, cout;
    full_adder fa(a, b, cin, sum, cout );

    initial begin
        a = 0; b = 0; cin = 0; // Expected 0, 0
        #10
        a = 1; // Expected 1, 0
        #10
        b = 1; // Expected 0, 1
        #10
        cin = 1; // Expected 1, 1
        #10
        b = 0; // Expected 0, 1
        #10
        a = 0; // Expected 1, 0
        #10
        $stop;
    end
endmodule
```

3 Ripple Carry Adder

A ripple carry adder is simply a chain of full adders used together (With the cout of one adder connected to the cin of the next), in order to add number consisting of multiple bits.

In this case, 4-bit addition was required, therefore 4 full adders were used together to form the required circuit. In the verilog code below ‘a‘ and ‘b‘ are the 4 bit numbers to be added and cin is an initial carry bit that the 4 bit adder takes.

3.1 Verilog Implementation

```verilog
// ripple_adder.v
module ripple_adder(
    sum, a, b, cin, cout
    );
    input[3:0] a;
    input[3:0] b;
    input cin;
    output[3:0] sum;
    output cout;
    wire c1, c2, c3, c4;
    full_adder BIT0(a[0], b[0], cin, sum[0], c1);
    full_adder BIT1(a[1], b[1], c1, sum[1], c2);
    full_adder BIT2(a[2], b[2], c2, sum[2], c3);
    full_adder BIT3(a[3], b[3], c3, sum[3], c4);
    assign cout = c4;
endmodule
```

3.2 Xilinx Design Constraints

9 Switches were used to set the inputs a[4], b[4] and cin.

5 LEDs were used to show the output of the addition sum[4] and carry.

```
# constraints.xdc
## Switches
set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { a[0] }];#LSB
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports { a[3] }];
set_property -dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports { b[0] }];
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports { b[1] }];
set_property -dict { PACKAGE_PIN N2    IOSTANDARD LVCMOS33 } [get_ports { b[2] }];
set_property -dict { PACKAGE_PIN N1    IOSTANDARD LVCMOS33 } [get_ports { b[3] }];
set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports { cin }];


## LEDs
set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { sum[0] }];#LSB
set_property -dict { PACKAGE_PIN H3    IOSTANDARD LVCMOS33 } [get_ports { sum[1] }];
set_property -dict { PACKAGE_PIN J1    IOSTANDARD LVCMOS33 } [get_ports { sum[2] }];
set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { sum[3] }];
set_property -dict { PACKAGE_PIN L3    IOSTANDARD LVCMOS33 } [get_ports { cout }];
```

4 Comments

4.1 Debugging

4.1.1 Simulation Problems

Initially we were facing trouble getting the testbench to run. We soon realized that this was because our testbench file was not loaded as a simulation source. Once we rectified this we were able to view the simulations properly.

4.1.2 FPGA Testing

We downloaded the wrong xdc template (One that doesn't work for the FPGA we had). This caused issues when we tried to load the verilog logic into the FPGA. After going through the error logs (which were hard to read), we realized the issue and fixed the xdc file. After this was done, everything worked as expected.