Lab 1 Report: Two-Bit Computer

ECE 3700

September 12, 2024

Nathaniel Fargo

Lab Objectives

The objective of this lab is to design and implement a simple 2-bit computer using three different methods: schematic entry, structural Verilog, and functional Verilog. The design will be tested using a Verilog testbench, simulated to confirm correctness, and then synthesized onto an FPGA for real-world validation. This lab introduces key techniques for digital logic design, simulation, and synthesis using Quartus and Verilog.

Introduction to Assignment

The challenge involves designing this circuit using three approaches:

1. Block design using basic logic gates (AND/OR/NOT/XOR/XNOR)
2. Structural Verilog which instantiates logic gates as components.
3. Functional Verilog which describes the circuit using assignments and bitwise operators.

Approach

Boolean Functions

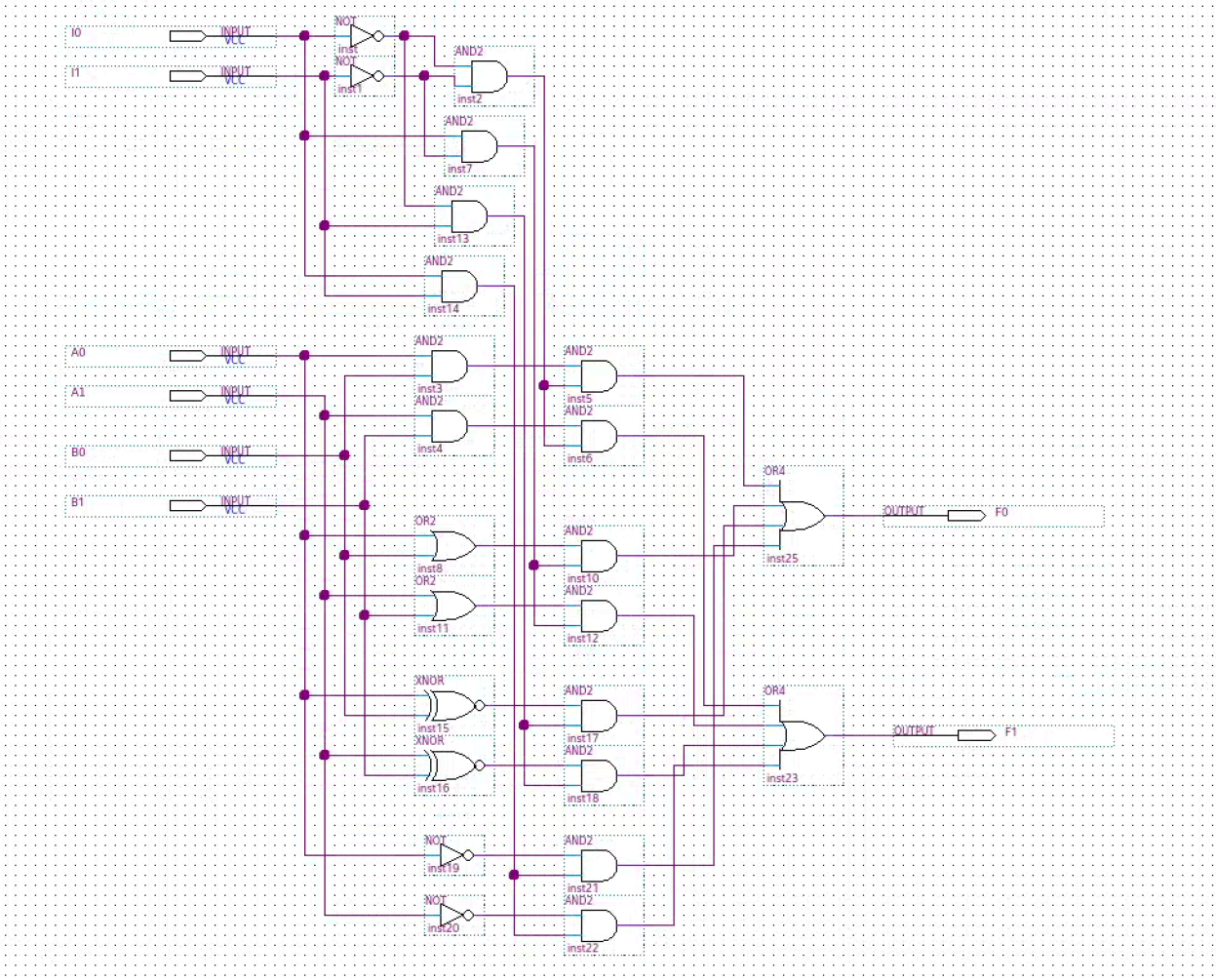
The logic functions for F[1:0] were derived based on the input I[1:0]. Here's the breakdown of the Boolean functions:

- For I = 00, the output is F[1] = A[1] & B[1], F[0] = A[0] & B[0].
- For I = 01, the output is F[1] = A[1] | B[1], F[0] = A[0] | B[0].
- For I = 10, the output is F[1] = A[1] == B[1], F[0] = A[0] == B[0].
- For I = 11, the output is the complement of A: F[1] = ~A[1], F[0] = ~A[0].

Implementation (Verilog Code)

Schematic Entry

The schematic was created using Quartus' Block Diagram/Schematic File. Using basic logic gates (AND, OR, NOT, XOR), I manually designed the circuit to implement the functions described above. A screenshot of the schematic design is shown below:



The top section represents decoding the instructions from 2 wires into 4 wires, which get sent down into the and gates. The other input of those AND gates is a juxtaposition of A[1:0] and B[1:0]. Either “and”ing, “or"ing, “xnor”ing, or “not”ing the inputs. Because only one of the four instruction lines will be active at once, only one pair of the 8 AND gates will be allowed to propagate their signals to the or gate and subsequently the output F.

The exact same structure is applied in the structural verilog code, which is shown here. This time basic gates are written as lines of code to combine inputs.

```
// Create versions of I
and (I00, _I[1], _I[0]);
and (I01, _I[1], I[0]);
and (I10, I[1], _I[0]);
and (I11, I[1], I[0]);

// AND gates
and (andAB[0], A[0], B[0]);
and (andAB[1], A[1], B[1]);

// OR gates
or (orAB[0], A[0], B[0]);
or (orAB[1], A[1], B[1]);

// XNOR gates
xnor (xnorAB[0], A[0], B[0]);
xnor (xnorAB[1], A[1], B[1]);

// NOT gates
not (notA[0], A[0]);
not (notA[1], A[1]);

// Define outputs based on I inputs and previous logic
and (out0[0], I00, andAB[0]);
and (out0[1], I00, andAB[1]);

and (out1[0], I01, orAB[0]);
and (out1[1], I01, orAB[1]);

and (out2[0], I10, xnorAB[0]);
and (out2[1], I10, xnorAB[1]);

and (out3[0], I11, notA[0]);
and (out3[1], I11, notA[1]);

// Combine results into F
or (F[0], out0[0], out1[0], out2[0], out3[0]);
or (F[1], out0[1], out1[1], out2[1], out3[1]);
```

Finally, a functional version was generated which consolidated the 50 lines of structural code down to just one line of clear and concise verilog. This module is also presented below.

Functional.v

```
1  module Functional(
2      input [1:0] A,
3      input [1:0] B,
4      input [1:0] I,
5      output [1:0] F
6  );
7
8      assign F = (I==2'b00 && (A & B)) | (I==2'b01 && (A | B)) | (I==2'b10 && (A ~^ B)) | (I==2'b11 && (~A));
9
10 endmodule
```

This version uses bitwise operators (&, |, ^, and ~) to combine the inputs A and B and then used equality checks to select which pairs are added together. This is essentially the exact same method of implementation used by the other two methods, but more more compact.

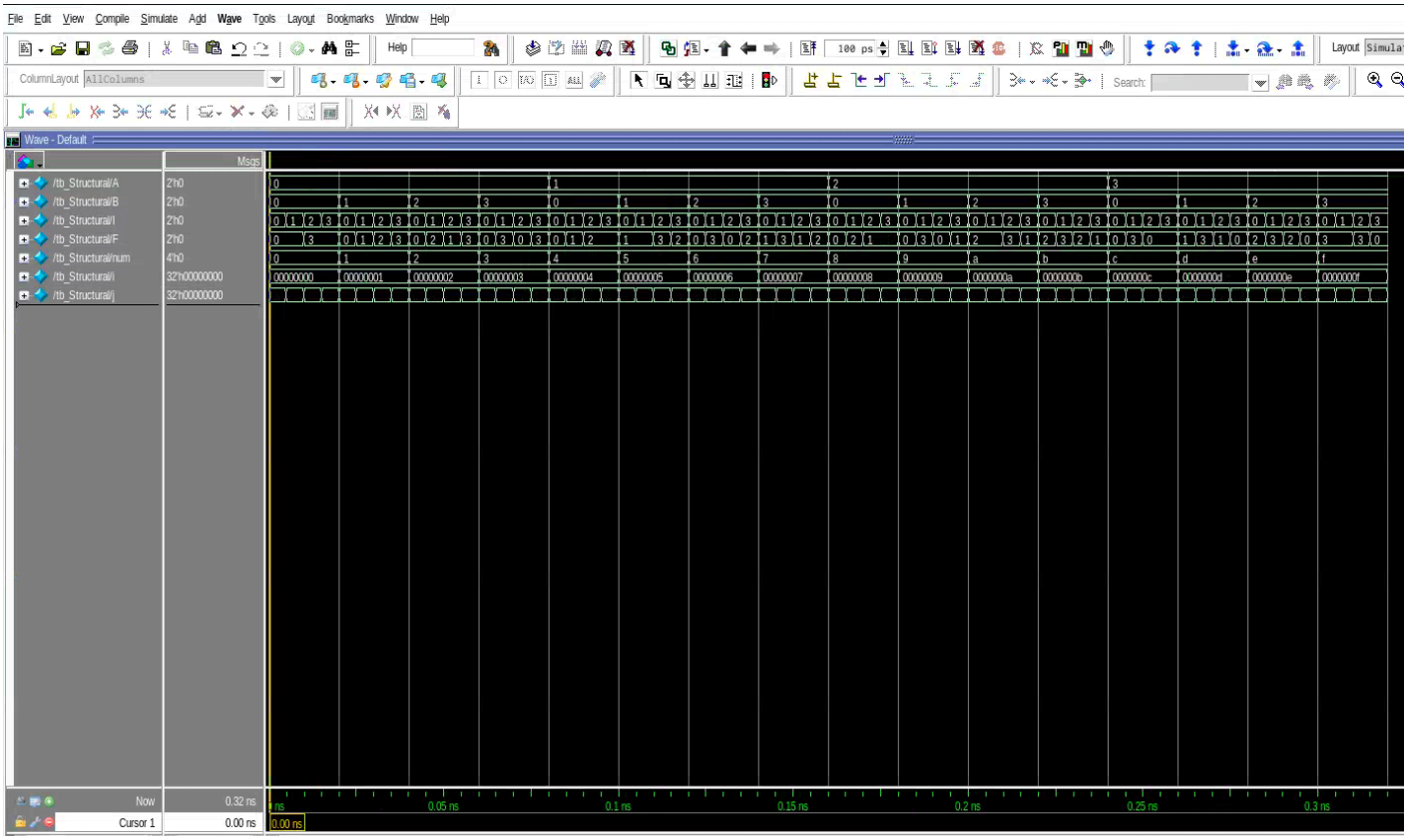
Simulation

Now that code has been created, it's time to test them. For this the simulation software Questa is used to generate waveforms and console outputs. First I wrote a testbench to simulate different signals and states to measure the output. Here's one of the testbenches used to probe every condition.

tb_Functional.v

```
1  module tb_Functional;
2     reg [1:0] A;
3     reg [1:0] B;
4     reg [1:0] I;
5     wire [1:0] F;
6
7     Functional uut (
8        .A(A),
9        .B(B),
10       .I(I),
11       .F(F)
12    );
13
14    reg [3:0] num;
15    integer i, j;
16    initial begin
17       // loop all 16 possible A/B combos
18       for (i = 0; i < 16; i = i + 1) begin
19          num = i; // set to a 4 bit register
20          A = num[3:2]; // break it down by 2 bits
21          B = num[1:0];
22
23          // display A & B inputs
24          $display("A is %x, B is %b", A, B);
25
26          for (j = 0; j < 4; j = j + 1) begin
27             // change I using same method
28             I = j;
29             $display("Instruction is %x output is %B", I, F);
30             // delay
31             #5;
32          end
33       end
34    end
35 endmodule
```

Next we ran this code to view the simulation output. Here's an example waveform from the functional test. "A" increments least frequently, then "B", then "I". As far as they were checked, all the outputs seemed to be working correctly.



The console output was also collected and a section is presented here.

```
VSIM 26> run
# A is 00, B is 00
# Instruction is 00 output is 00
# Instruction is 01 output is 00
# Instruction is 10 output is 11
# Instruction is 11 output is 11
# A is 00, B is 01
# Instruction is 00 output is 00
# Instruction is 01 output is 01
# Instruction is 10 output is 10
# Instruction is 11 output is 11
# A is 00, B is 10
# Instruction is 00 output is 00
# Instruction is 01 output is 10
# Instruction is 10 output is 01
# Instruction is 11 output is 11
# A is 00, B is 11
# Instruction is 00 output is 00
```

Programming

The device was also programmed for this lab and was shown to be working correctly. The TA checked this functionality off. The switches and LEDs were used to control and output signals. Shown here is the top module to implement the submodules and also the pin planner which physically connects them to the I/O.

```
module Top (
    input [9:0] switches,
    output [9:0] leds
);

    wire [1:0] A, B, I, F;

    assign A = switches[9:8];
    assign B = switches[7:6];
    assign I = switches[1:0];

    Structural struct_inst(
        .A(A),
        .B(B),
        .I(I),
        .F(F)
    );

    assign leds[9:5] = { 1'b0, F[1], F[1], F[1], 1'b0 };
    assign leds[4:0] = { 1'b0, F[0], F[0], F[0], 1'b0 };

endmodule
```