Project: Building an Integer ALU - Report 1

John Yamamoto Alan Zacatula Samuel Wait

Due September 21, 2023

0.1 Introduction

The first step of this project required the group to install Verilog, the hardware descriptive language that models electronic systems, and use it to define the mathematics of an Integer ALU according to the project requirements. We then compile and run the code against test cases also coded in Verilog. Lastly, we use GTK Wave to generate simulation circuit waveforms based on the compiled and tested Verilog code, all done in a Ubuntu virtual environment. Our group goal is to write and compile great Verilog code, test it against scripts that properly evaluate the code, and generate circuit waveforms that represent an Integer ALU according to the project specifications.

Chapter 1

Verilog and Virtual Environment

Since this the first status check and report of the project, we had to start by setting up the virtual environment. We are all using an Ubuntu Virtual Machine (VM) on top of different Operating Systems and processor architectures, but this should not affect how the Verilog code is compiled or tested, or how the GTK Wave circuit waveforms are generated. Once the VM's were set up, we were able to install Verilog and GTK Wave on the virtual Ubuntu environments through the Ubuntu terminals.

For the first status check of the project, we were were required to code 1-bit Not, Nand, and Nor Circuits, and a 4-bit shift circuit in Verilog.

1.1 Not Circuit

The Not circuit gate takes an input, a, and gives an output, y, for the module not_gate. The module then sets the output y equal to ã, so whatever the module takes as input for a, it will output the opposite (or not a). The test bench Verilog code was a little bit more complicated, as it included coding a clock that pulses every 5 nanoseconds, changing to either 0 or 1 with every other pulse, or every 10 nanoseconds. We then coded the input (a) initially be 1 for the first 10 nanoseconds, then when the clock switched back to 0 after 10 nanoseconds, the input changed to 0. The output (y) was what we expected for the Not circuit, which was 0 for the first 10 nanoseconds and 1 for the last 10. The output is then put into a .vcd file that GTK Wave uses to generate the waveform. The Not Circuit Verilog source code and test bench is shown below:

```
1 module not_gate(a,y);
2
3 input a;
4 output y;
5
6 assign y = ~a;
7
8 endmodule
```

Figure 1.1: Not Circuit Source Code

```
1 module not_gate_test;                    //module name
2
3   /* Make a regular pulsing clock. */    //tracks every 5 ns
4   reg clk = 0;
5   always #5 clk = !clk;                  //every 5 ns clk either 1 or 0S
6
7   reg a = 0;                             //inputs
8   initial begin                          //load new values
9     # 10 a = 1;                          //every 10 nano
10    # 10 $finish;                        //seconds
11  end
12
13  wire y;                                //output
14  not_gate a0(a, y);                     //call for gate module
15
16  initial begin                          //code for gtkwave
17    $dumpfile("not_gate.vcd");
18    $dumpvars(0,clk);
19    $dumpvars(1,a);
20    $dumpvars(2,y);
21  end
22
23  initial
24    $monitor("At time %t, a(%b)= y(%b)",        //prints to screen
25                                $time, a, y);   //when any value changes
26 endmodule //test
```

Figure 1.2: Not Circuit Test Bench

1.2 Nor and Nand Circuits

The next two circuits, Nor and Nand follow a similar pattern but required two inputs rather than one. The Verilog source code took an input a and b into the nor_gate module, and gives the output y. The module sets y equal to not a or b then ends the module, returning y as the output. The test bench for the Nor circuit works in a similar manner as the Not circuit, setting the clock equal to 0 and pulsing every 5 nanoseconds, changing the input to the module every time the clock is set to 0, which is every 10 nanoseconds. The difference here is that there are two inputs, a and b, so the clock has to run for 40 nanoseconds to change the input a total of 4 times. The output is then put into a .vcd file that GTK Wave uses to generate the waveform. The Nor Circuit Verilog source code and test bench is shown below:

```
1 module nor_gate(a,b,y);
2
3 input a,b;
4 output y;
5
6 assign y = ~(a | b);
7
8 endmodule
```

Figure 1.3: Nor Circuit Source Code

```
1  module nor_gate_test;                    //module name
2
3    /* Make a regular pulsing clock. */    //tracks every 5 ns
4    reg clk = 0;
5    always #5 clk = !clk;                  //every 5 ns clk either 1 or 0S
6
7    reg a = 0;                             //inputs
8    reg b = 0;
9    initial begin                          //load new values
10     # 10 a = 1;                          //every 10 nano
11     # 10 b = 1;                          //seconds
12     # 10 a = 0;
13     # 10 $finish;
14   end
15
16   wire y;                                //output
17   nor_gate a0(a, b, y);                  //call for gate module
18
19   initial begin                          //code for gtkwave
20     $dumpfile("nor_gate.vcd");
21     $dumpvars(0,clk);
22     $dumpvars(1,a);
23     $dumpvars(2,b);
24     $dumpvars(3,y);
25   end
26
27   initial                                //prints to screen when values change
28     $monitor("At time %t, a(%b), b(%b) = y(%b)",
29                 $time, a, b, y);
30
31 endmodule //test
```

Figure 1.4: Nor Circuit Test Bench

The Nand circuit is most like the Nor circuit in both coding and testing, since they both require two inputs. For the Verilog source code, the Nand circuit is coded the exact same way as the Nor circuit, with one slight change. the nand_gate module takes both inputs a and b, but y equal to not a and b, then ends the module and outputs y. The test bench works the exact same way as the Nor test bench, where the clock alternates between 0 and 1 every 5 nanoseconds and between changing a and b every time the clock hits 0, or every 10 nanoseconds. The output is then put into a .vcd file that GTK Wave uses to generate the waveform.The Nand Circuit Verilog source code and test bench is shown below:

```
1 module nand_gate(a,b,y);
2
3 input a,b;
4 output y;
5
6 assign y = ~(a & b);
7
8 endmodule
```

Figure 1.5: Nand Circuit Source Code

```
1 module nand_gate_test;                    //module name
2
3   /* Make a regular pulsing clock. */     //tracks every 5 ns
4   reg clk = 0;
5   always #5 clk = !clk;                   //every 5 ns clk either 1 or 0S
6
7   reg a = 0;                              //inputs
8   reg b = 0;
9   initial begin                           //load new values
10    # 10 a = 1;                           //every 10 nano
11    # 10 b = 1;                           //seconds
12    # 10 a = 0;
13    # 10 $finish;
14  end
15
16  wire y;                                 //output
17  nand_gate a0(a, b, y);                  //call for gate module
18
19  initial begin                           //code for gtkwave
20    $dumpfile("nand_gate.vcd");
21    $dumpvars(0,clk);
22    $dumpvars(1,a);
23    $dumpvars(2,b);
24    $dumpvars(3,y);
25  end
26
27  initial                                 //prints to screen when values change
28    $monitor("At time %t, a(%b), b(%b) = y(%b)",
29                $time, a, b, y);
30 endmodule //test
```

Figure 1.6: Nand Circuit Test Bench

1.3 4-bit Shift Circuit

The last circuit, the 4-bit Shift circuit, is coded completely different from the other three circuits, but the test bench works in a similar manner. The Verilog source code for the shift_4bit module takes a hexidecimal input, a, which is the original value to be shifted 4-bits. The output of the module is hexidecimal values l_shift and r_shift, which are assigned to the input a shifted left and right by input shift, respectively. The test bench is similar to the other three test benches as it uses a clock that pulses every 5 nanoseconds and changes the input every 10 nanoseconds. The output is then put into a .vcd file that GTK Wave uses to generate the waveform. The Shift Circuit Verilog source code and test bench is shown below:

```
1 module shift_4bit (
2     input [3:0] input_4bit,
3     output [3:0] left_shift, right_shift
4 );
5
6 // 4-bit Left Shift (multiplication by 2) with zero-fill
7 assign left_shift = input_4bit << 1;
8
9 // 4-bit Right Shift (division by 2) with zero-fill
10 assign right_shift = input_4bit >> 1;
11
12 endmodule
```

Figure 1.7: shift Circuit Source Code

```
1 module shift_circuit_tb;                  //module name
2
3   /* Make a regular pulsing clock. */     //tracks every 5 ns
4   reg clk = 0;
5   always #5 clk = !clk;                   //every 5 ns clk either 1 or 0S
6
7   reg [3:0] a;                            //input
8
9   initial begin                                   //load new values
10          a = 4'b0010;          //every 10 nano
11    # 10 a = 4'b1000;           //seconds
12    # 10 a = 4'b1111;
13    # 10 a = 4'b1001;
14    # 10 $finish;
15  end
16
17  wire [3:0]l_shift;                          //outputs
18  wire [3:0]r_shift;
19
20  shift_4bit a0(a, l_shift, r_shift);  //call for gate module
21
22  initial begin                           //code for gtkwave
23    $dumpfile("shift_circuit.vcd");
24    $dumpvars(0,clk);
25    $dumpvars(1,a);
26    $dumpvars(2,l_shift);
27    $dumpvars(3,r_shift);
28  end
29
30  initial                                 //prints to screen when values change
31    $monitor("At time %t, a(%b) << 1 = (%b), a(%b) >> 1 = (%b)",
32                $time, a, l_shift, a, r_shift);
33
34 endmodule //test
```

Figure 1.8: Shift Circuit Test Bench

Chapter 2

Waveforms

2.1 Not Circuit

The waveform for the Not Circuit is shown below:

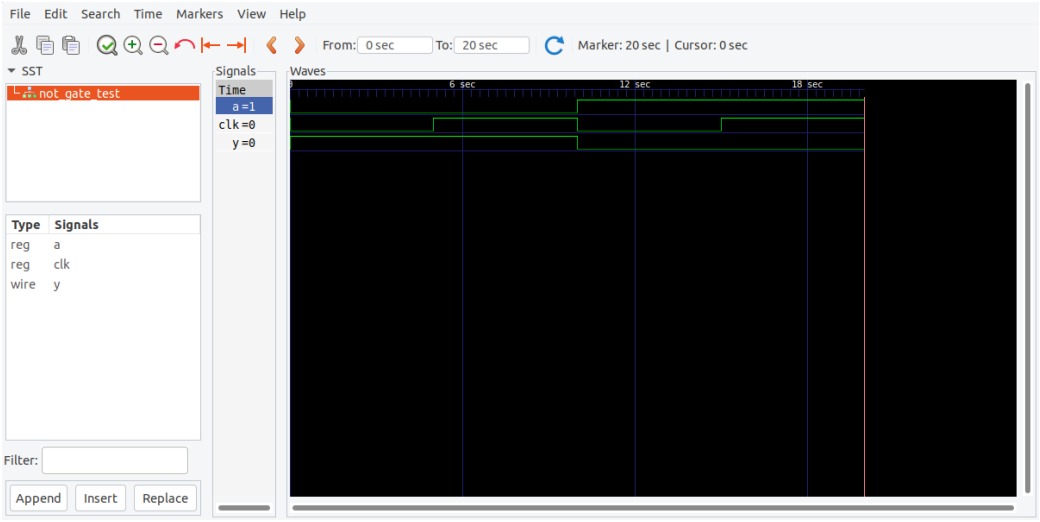

Figure 2.1: Not Circuit Waveform

Here, both a and b are initialized to 0, so the output of the module, y, is 1 for the first 10 nanoseconds. Input a is then changed to 1 for the second 10 nanoseconds, then b is changed to 1 for the third. Finally, a is changed to 0 for the last 10 nanoseconds, For the last 30 nanoseconds, the output is always

0. This shows that for a Not Circuit, the output is always the opposite of the input, so if a is on, the output is 0, and it a is off, the output is 1.

2.2 Nor Circuit

The waveform for the Nor Circuit is shown below:

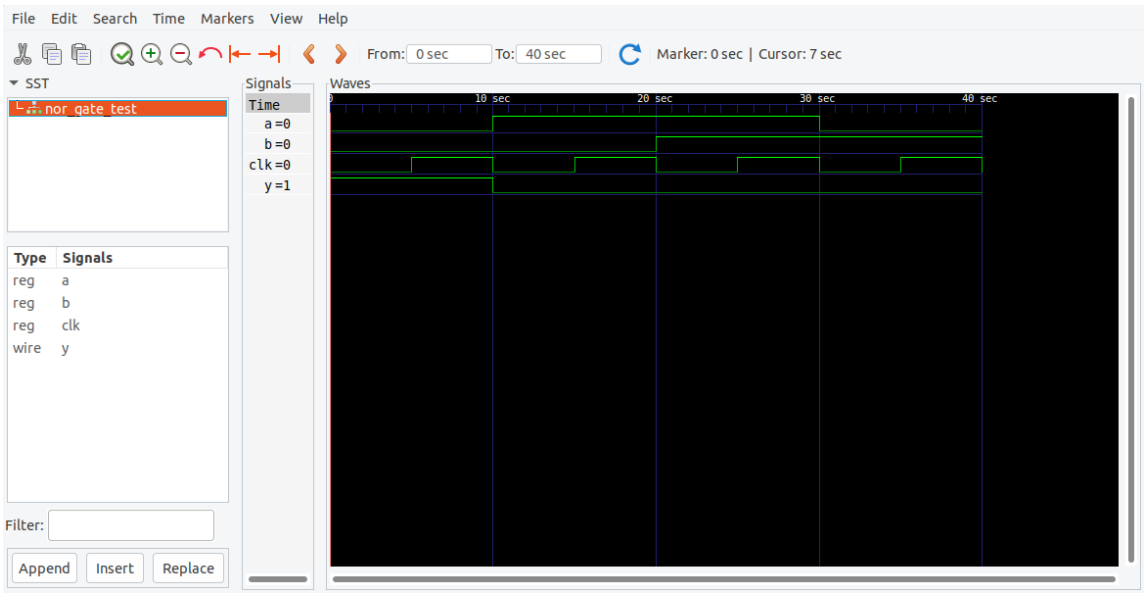

Figure 2.2: Nor Circuit Waveform

Here, inputs a and b are both initialized to 0, which outputs 1, then a is changed to 0 for the second 10 nanoseconds and outputs 0. For the third 10 nanoseconds b is also set to 1, then a is changed back to 0 for the last 10 nanoseconds, which both of which output 0 again. This shows that for a Nor Circuit, the only time the output is 1 is when both a and b are off, otherwise the output is 0.

2.3 Nand Circuit

The waveform for the Nand Circuit is shown below:

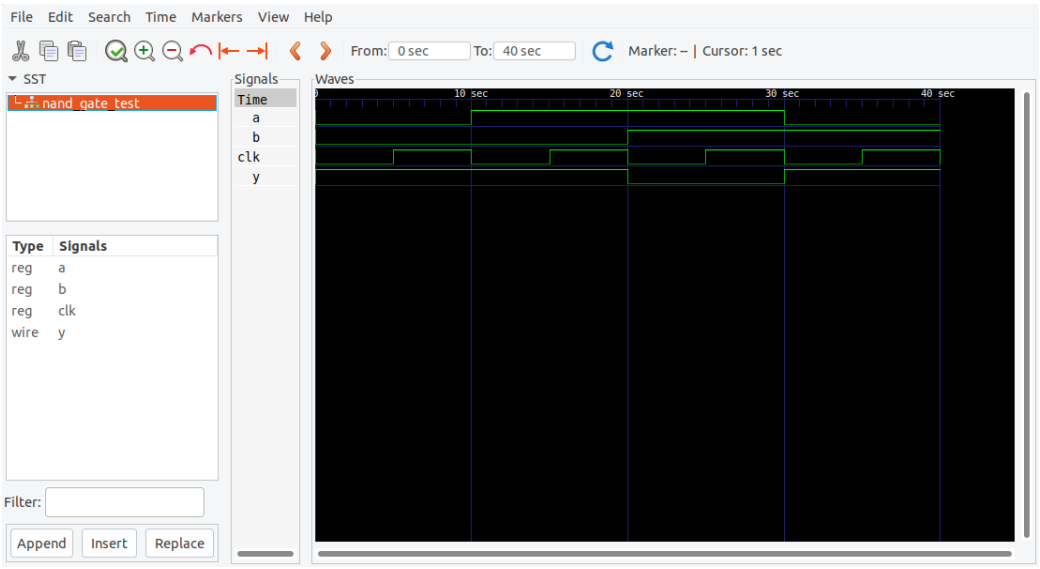

Figure 2.3: Nand Circuit Waveform

Here, inputs a and b are both initialized to 0, then a is changed to 1 for the second 10 nanoseconds, both of which output 1. For the third 10 nanoseconds b is set to 1 and outputs 0, and finally a is changed back to 0 for the last 10 nanoseconds, which outputs 1. This shows that for a Nand Circuit, the only time that the output is 0 is when both inputs a and b are on, otherwise the output is 1.

2.4 4-bit Shift

The waveform for the 4-bit shift is shown below:

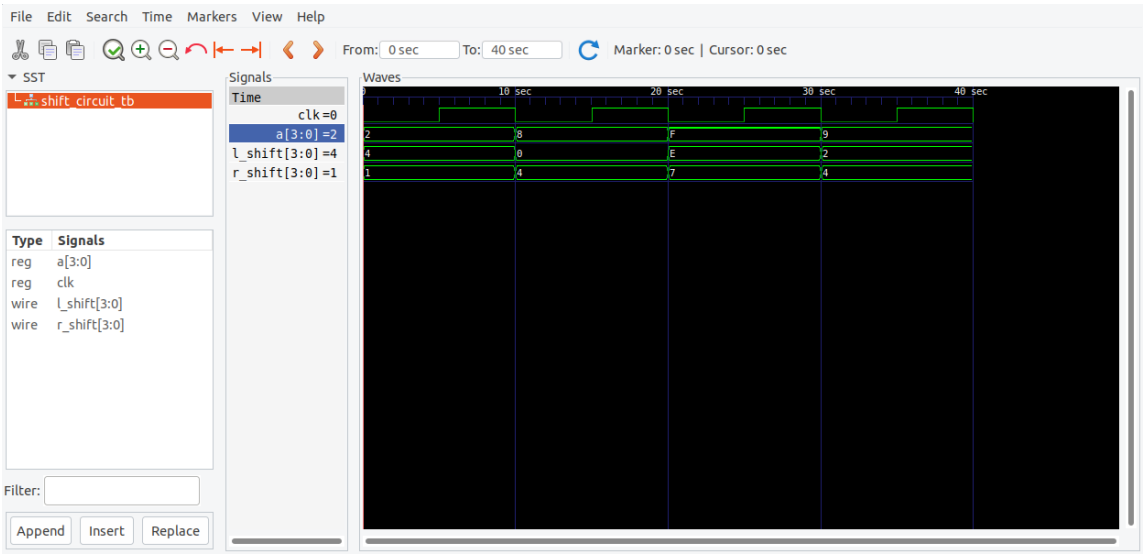

Figure 2.4: Shift Circuit Waveform

Here, the input starts at 2 for the first 10 nanoseconds which outputs 4 for the left shit and 1 for the right shift, then moves to 8 for the second 10 nanoseconds, which outputs 0 for the left shift and 4 for the right shift. The third 10 nanoseconds has an input of F and a left shift output of E and a right shift output of 7. Finally, the input is 9 for the last 10 nanoseconds and outputs 2 for the left shift and 4 for the right shift.

Chapter 3

Conclusion

Given the requirements for this first project status report, I believe our group got the correct results. The code, test benches, and waveforms for the Not, Nor, Nand, and 4-bit Shift circuits seem to follow the truth tables for their corresponding logical and bitwise operations and inputs, so it stands to reason that they are correct and our project is successful so far. There was some difficulty in the beginning stages of the project, mostly with properly setting up the tools and understanding how the Verilog syntax and compilation and waveform generation works. Sam had some initial difficulty setting up his Ubuntu environment because he uses an apple laptop that has a 64-bit ARM processor that is fairly new, so finding a version of Ubuntu and VM software compatible with his hardware took a little bit of time. Once the environments were set up, the only real difficulty was understanding how the Verilog syntax works and how the compiled Verilog code works with GTK Wave to generate the waveforms we needed. The actual operations themselves were straightforward, so once we understood how to accomplish this it just took some time to meet the requirements.