Building an Integer ALU Step 2
CS 3339
Group: The Architects

Aaron Reed & Carlo Marroquin & Ryan Bieker

November 1, 2023

Instructor: Professor Klepetko

1 Objective

To introduce the process and methodology of designing a new computer circuit from scratch by coding 4-bit Binary Functions And, Nand, Or, Nor, Xor, Xnor, Not, and a Shifter. 4-bit arithmetic operations Addition, Subtraction, Multiplication all including carry in and carry out circuits

2 Verilog Code

3 AND Design

```
module and_4b(x, y, out);

  output [3:0] out;
  input [3:0] x, y;

  wire x, y, out;

  assign out = x & y;

endmodule
```

4 AND Testbench

```
module and_1b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 $finish;
  end

  wire [3:0] out;
  and_4b a0 (x, y, out);

  initial begin
     $dumpfile("and_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b) = out(%b)",
              $time, x, y, out);
endmodule // test
```

5 NAND Design

```
module nand_4b(x, y, out);

  output [3:0] out;
  input [3:0] x, y;

  wire x, y, out;

  assign out = ~(x & y);

endmodule
```

6 NAND Testbench

```
module nand_1b_test;
```

```
  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 $finish;
  end

  wire [3:0] out;
  and_4b a0 (x, y, out);

  initial begin
     $dumpfile("nand_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b) = out(%b)",
              $time, x, y, out);
endmodule // test
```

7 OR Design

```
module or_4b(x, y, out);

  output [3:0] out;
  input [3:0] x, y;

  wire x, y, out;

  assign out = x | y;

endmodule
```

8 OR Testbench

```
module or_1b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 $finish;
  end

  wire [3:0] out;
  or_4b a0 (x, y, out);

  initial begin
     $dumpfile("or_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b) = out(%b)",
              $time, x, y, out);
endmodule // test
```

9 NOR Design

```
module nor_4b(x, y, out);

  output [3:0] out;
  input [3:0] x, y;

  wire x, y, out;

  assign out = ~(x | y);

endmodule
```

10 NOR Testbench

```
module nor_1b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 $finish;
  end

  wire [3:0] out;
  nor_4b a0 (x, y, out);

  initial begin
     $dumpfile("nor_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b) = out(%b)",
              $time, x, y, out);
endmodule // test
```

11 XOR Design

```
module xor_4b(x, y, out);

  output [3:0] out;
  input [3:0] x, y;

  wire x, y, out;

  assign out = x ^ y;

endmodule
```

12 XOR Testbench

```
module xor_1b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 $finish;
  end

  wire [3:0] out;
  xor_4b a0 (x, y, out);

  initial begin
     $dumpfile("xor_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b) = out(%b)",
              $time, x, y, out);
endmodule // test
```

13 XNOR Design

```
module xnor_4b(x, y, out);

  output [3:0] out;
  input [3:0] x, y;

  wire x, y, out;

  assign out = ~(x ^ y);

endmodule
```

14 XNOR Testbench

```
module xnor_1b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 $finish;
  end

  wire [3:0] out;
  xnor_4b a0 (x, y, out);

  initial begin
     $dumpfile("xnor_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b) = out(%b)",
              $time, x, y, out);
endmodule // test
```

15 NOT Design

```
module not_4b(x, out);

  output [3:0] out, o;
  input [3:0] x;

  wire x, out;

  assign out = ~x;

endmodule
```

16 NOT Testbench

```
module not_1b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 x = 4'b1101;
     # 10 x = 4'b0101;
     # 10 $finish;
  end

  wire [3:0] out;
  not_4b a0 (x, out);

  initial begin
     $dumpfile("not_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(3,out);
  end

  initial
     $monitor("At time %t, x(%b) = out(%b)",
              $time, x, out);
endmodule // test
```

17 SHIFT Design

```
module shift_4b(x, y);

  output [3:0] xOut, yOut;
  input [3:0] x, y;

  wire [3:0] x, y, xOut, yOut;

  assign xOut = x;
  assign yOut = y;

endmodule
```

18 SHIFT Testbench

```
module shift_4b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #5 clk = !clk;

  reg [3:0] x;
  reg [3:0] y;
  initial begin

      x = 4'b1000;
      y = 4'b0001;

      //shift bit in x by 1 to right, y by 1 left every 10 seconds 4 times
      for (integer i = 0; i < 3; i = i + 1) begin
         # 10 x = ( x >> 1);
         y = ( y << 1);
      end
     # 10 $finish;
  end

  wire o;
  wire out;
  shift_4b a0 (x, y);

  initial begin
     $dumpfile("shift_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
  end

  initial
     $monitor("At time %t, x(%b) + y(%b)",
              $time, x, y);
endmodule // test
```

19 ADDITION Design

```
module add_4b(x, y, out, Cin, Cout);

  output [3:0] out;
  output Cout;
  input [3:0] x, y;
  input Cin;

  wire x, y, out;

  assign {Cout, out} = x + y + Cin;


endmodule
```

20 ADDITION Testbench

```
module add_4b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  reg Cin = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 Cin = 1;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 Cin = 0;
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 $finish;
  end

  wire [3:0] out;
  wire Cout;
  add_4b a0 (x, y, out, Cin, Cout);

  initial begin
     $dumpfile("add_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
    $dumpvars (4, Cout);
    $dumpvars (5, Cin);
  end
  initial
     $monitor("At time %t, x(%b) + y(%b) + Cin(%b)= out(%b) with Cout(%b)",
              $time, x, y, Cin, out, Cout);
endmodule // test
```

21 SUBTRACTION Design

```
module sub_4b(x, y, out, Cin, Cout);

  output [3:0] out;
  output Cout;
  input [3:0] x, y;
  input Cin;

  wire x, y, out;

  assign {Cout, out} = x - y - Cin;

endmodule
```

22 SUBTRACTION Testbench

```
module sub_4b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  reg Cin = 0;
  initial begin
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 Cin = 1;
     # 10 x = 4'b1101;
     # 10 y = 4'b0110;
     # 10 Cin = 0;
     # 10 x = 4'b1000;
     # 10 y = 4'b1001;
     # 10 $finish;
  end

  wire [3:0] out;
  wire Cout;
  sub_4b a0 (x, y, out, Cin, Cout);

  initial begin
     $dumpfile("sub_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
    $dumpvars (4, Cout);
    $dumpvars (5, Cin);
  end

  initial
     $monitor("At time %t, x(%b) - y(%b) - Cin(%b)= out(%b) with Cout(%b)",
              $time, x, y, Cin, out, Cout);
endmodule // test
```

23 MULTIPLICATION Design

```
module mult_4b(x, y, out, Cin, Cout);

  output [7:0] out;
  output Cout;
  input [3:0] x, y;
  input Cin;

  wire x, y, out;

  assign {Cout, out} = (x * y) * Cin;


endmodule
```

24 MULTIPLICATION Testbench

```
module mult_4b_test;

  /* Make a regular pulsing clock. */
  reg clk = 0;
  always #10 clk = !clk;

  reg [3:0] x = 0;
  reg [3:0] y = 0;
  reg Cin = 1;
  initial begin
```

```
      # 10 x = 4'b1000;
      # 10 y = 4'b1001;
      # 10 Cin = 0;
      # 10 x = 4'b1101;
      # 10 y = 4'b0110;
      # 10 Cin = 1;
      # 10 x = 4'b1000;
      # 10 y = 4'b1001;
      # 10 $finish;
  end

  wire [7:0] out;
  wire Cout;
  mult_4b a0 (x, y, out, Cin, Cout);

  initial begin
    $dumpfile("mult_4b.vcd");
     $dumpvars(0,clk);
     $dumpvars(1,x);
     $dumpvars(2,y);
     $dumpvars(3,out);
    $dumpvars (4, Cout);
    $dumpvars (5, Cin);
  end

  initial
     $monitor("At time %t, x(%b) * y(%b) *Cin(%b) = out(%b) with Cout(%b)",
              $time, x, y, out, Cout, Cin);
endmodule // test
```

25 Waveforms

26 AND

The AND function took only 4 bit inputs X and Y then used them in a logical AND function.

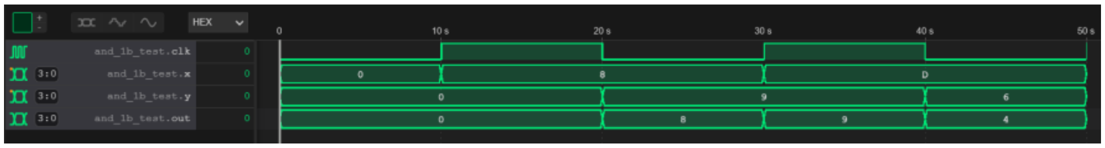

Figure 1: The AND function's waveform.

27 NAND

The NAND function took 4 bit inputs X and Y as well, then preformed the logical NAND. An out is displayed on our waveform.

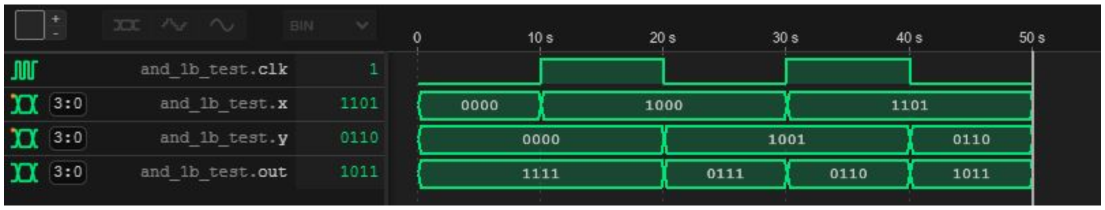

Figure 2: The NAND function’s waveform.

28 OR

The OR function took inputs x and y, each 4 bits. Out is also displayed on the waveform. Then it preformed the logical OR on both 4 bit numbers.

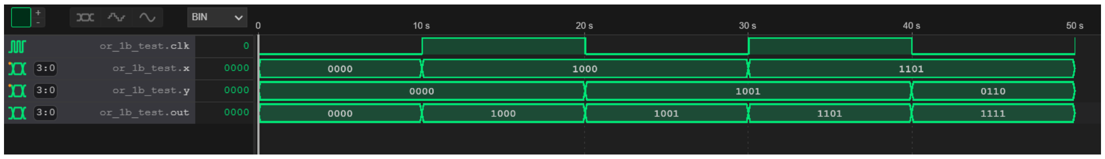

Figure 3: The OR functions waveform.

29 NOR

The NOR function took the 4 bit inputs X and Y, out is displayed on the waveform. Then the logical NOR was preformed and waveform produced.

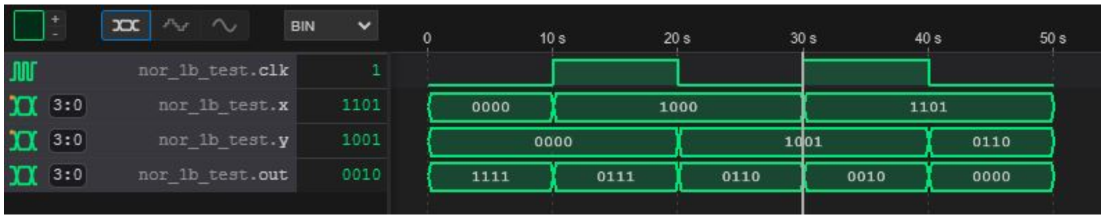

Figure 4: The NOR function Waveform.

30 XOR

The XOR function took two 4 bit inputs X and Y. Then the logical XOR is preformed, an out is also displayed on the waveform.

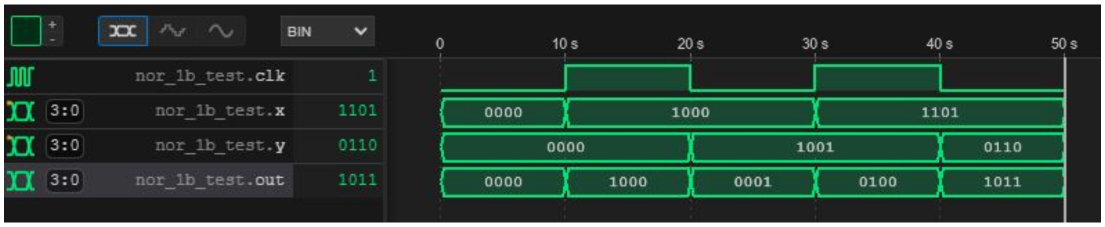

Figure 5: The XOR waveform.

31 XNOR

The XNOR function receives two 4-bit inputs X and Y. When the individual bits are the same, the output bit-place is 1.

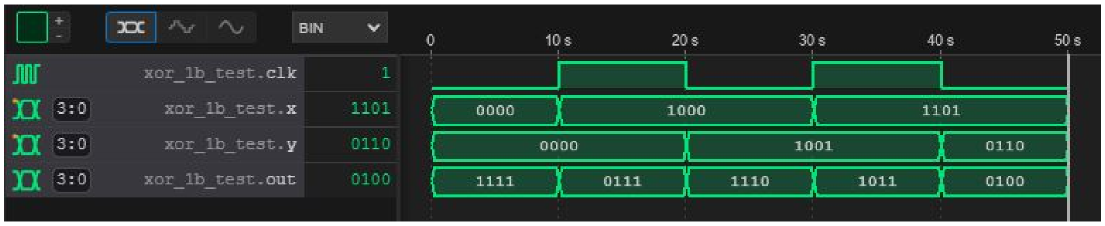

Figure 6: The XNOR wave form.

32 NOT

The NOT module receives one 4-bit input and flips the value of each bit.

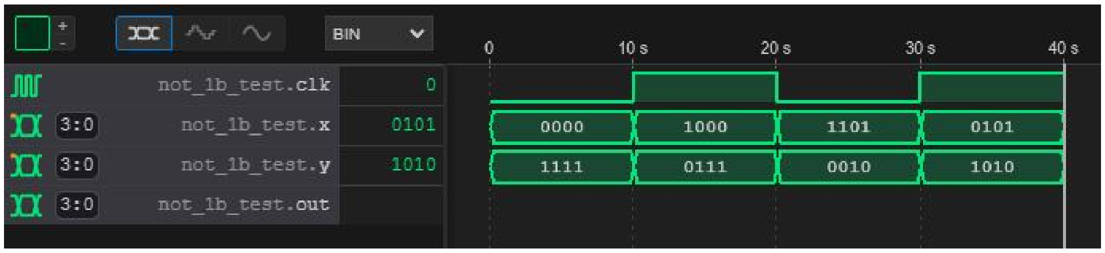

Figure 7: The 4-bit NOT waveform.

33 SHIFT

The SHIFT module receives one 4-bit input and shifts the bits right or left and.

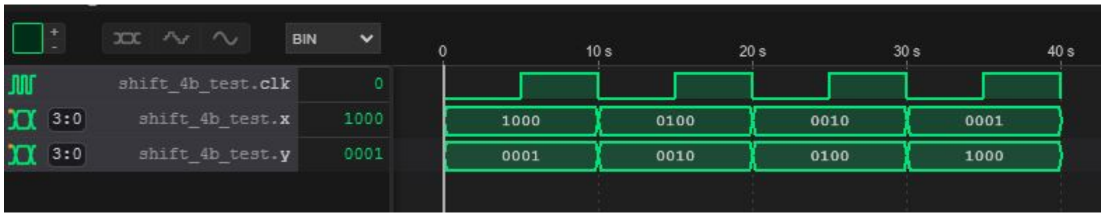

Figure 8: The 4-bit SHIFT waveform.

34 ADDITION

The ADD module receives two 4-bit inputs and adds the two values together. If the value exceeds 4-bits a Carry-out is assigned to 1.

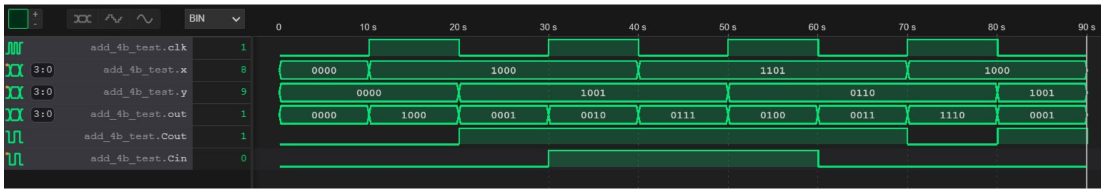

Figure 9: The 4-bit ADDITION waveform.

35 SUBTRACTION

The SUBTRACTION module receives two 4-bit inputs and subtracts the second 4-bit value from the first with an output using two's complement.

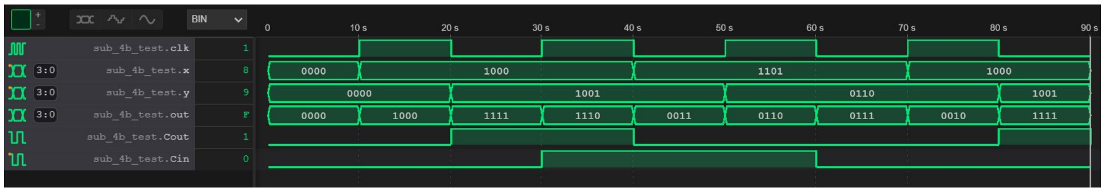

Figure 10: The 4-bit SUBTRACTION waveform.

36 MULTIPLICATION

The MULTIPLICATION module receives two 4-bit values, multiplies the two together, and produces an 8-bit output.

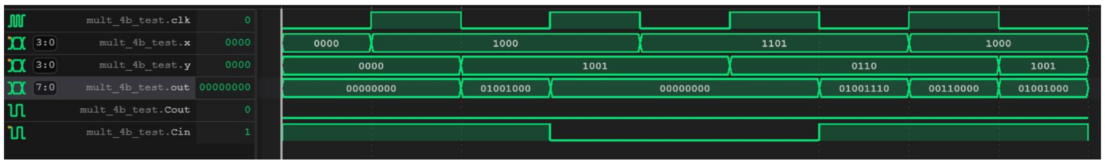

Figure 11: The 4-bit MULTIPLICATION waveform.

37 Conclusion

In order to create our 4-bit modules in Verilog, we had to determine how to store 4-bit numbers in registers and wires. Addition was the most straightforward to code with Carry-In and Carry-Out values. In our multiplication module, we found that the product of two 4-bit numbers needed to be 8-bits in order to properly implement the Carry-Out of the module. The multiplication module gave us the most trouble as we couldn't decide what size the output product should be. The logic gates were quick to get through using 4-bit values as they are built into the Verilog code as operators.