EECS 151/251A Homework 2

Due Monday, Feb 11th, 2019

Suggestions for this HW Assignment

You will be asked to write several Verilog modules as part of this HW assignment. You are highly encouraged to test your modules by running them through a simulator. Several simulator options were discussed in Discussion 2 including `https://www.edaplayground.com` which is a free, online, Verilog simulator.

Problem 1: Interpreting Verilog [4,2 pts.]

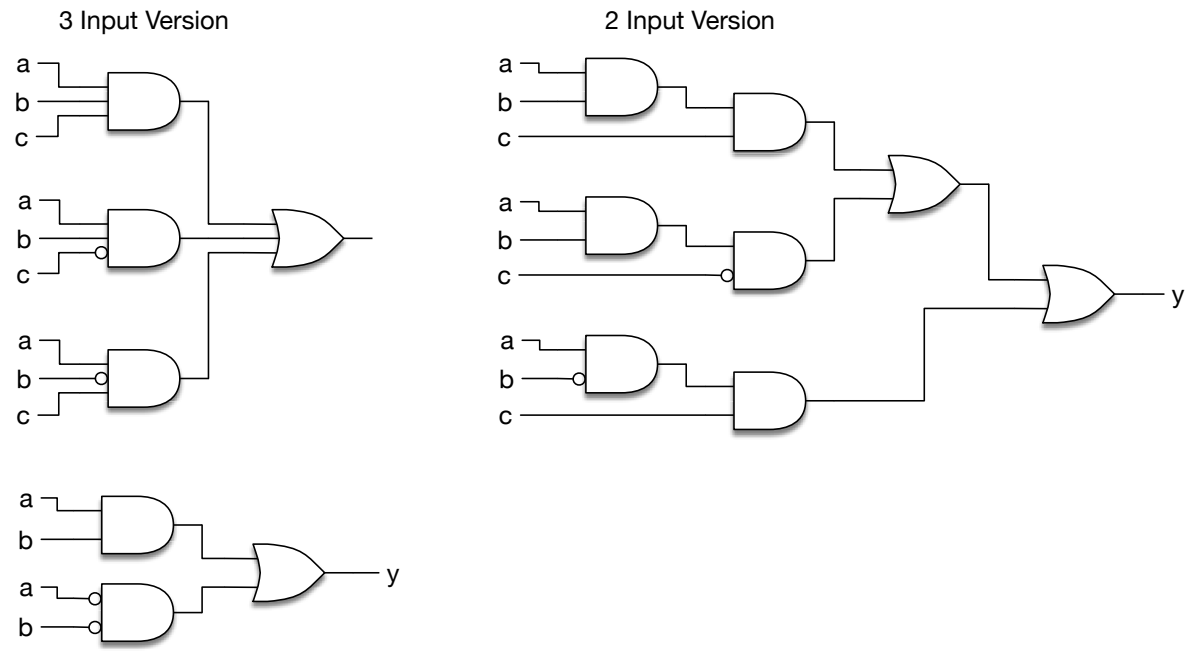
(a) Sketch a gate-level schematic of the circuit described by the Verilog code below. [4 pts]

(b) Try to simplify the schematic by sharing gates when possible. [2 pts]

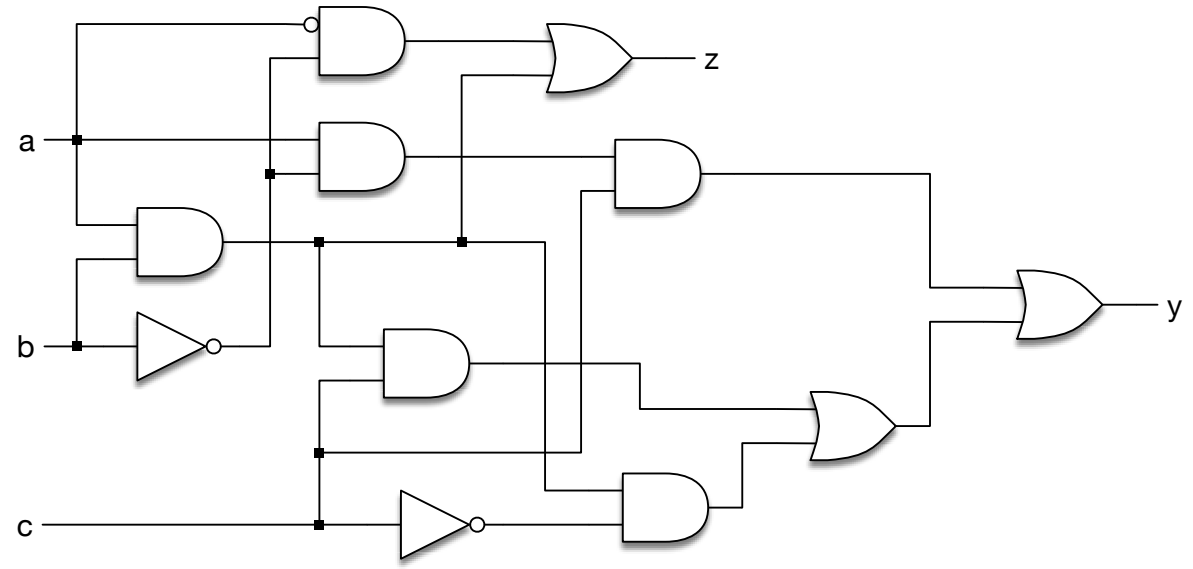
```
module circuit1(a, b, c, y z)
  input a, b, c;
  output y,z;

  assign y = a & b & c | a & b & ~c | a & ~b & c;
  assign z = a & b | ~a & ~b;
endmodule
```

Solution:

Part (a):



Part (b):



Problem 2: Interpreting and Re-Implementing Verilog [4,3,4 pts.]

(a) Sketch a gate-level schematic of the circuit described by the Verilog code below (using only simple logic gates)

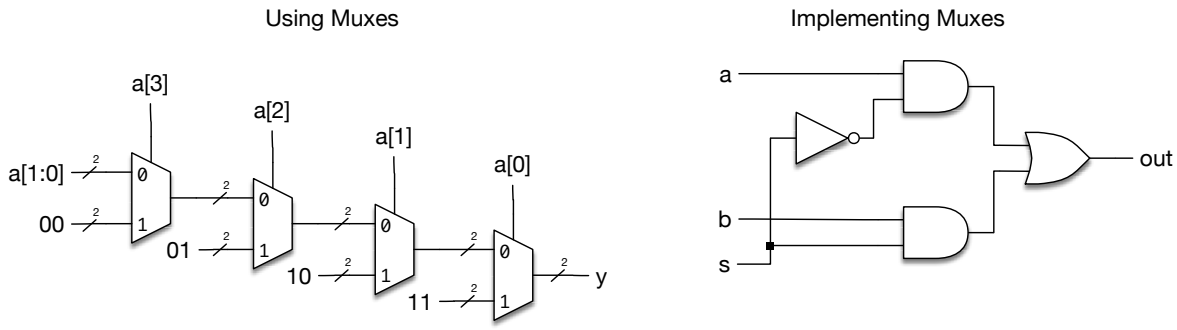
(b) Try to simplify the schematic by sharing gates and eliminating unnecessary gates.

(c) Write an equivalent implementation of circuit2 that uses the Verilog "case" construct.

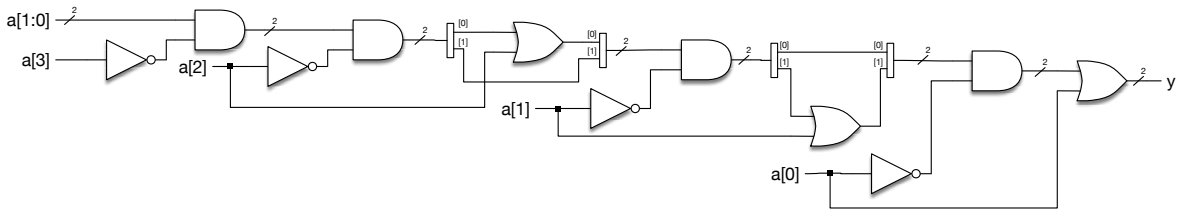
```
module circuit2(a, y)
  input [3:0] a;
  output [1:0] y;

  always @(*)
    if (a[0]) y = 2'b11;
    else if (a[1]) y=2'b10;
    else if (a[2]) y=2'b01;
    else if (a[3]) y=2'b00;
    else           y=a[1:0];
endmodule
```

Solution:

Part (a):



Part (b):



Part (c): Verilog Modules:

```
1  module circuit2_case(a, y);
2    input [3:0] a;
3    output reg [1:0] y;
4
5    always @(*)
6      //Need to handle the precidence in the if/else
7      case(a)
8        //First if
```

```
        4'b0001: y = 2'b11; //The actual greycode
        //4'b0011: y = 2'b11; Handled by default
        4'b0101: y = 2'b11;
        //4'b0111: y = 2'b11; Handled by default
        4'b1001: y = 2'b11;
        //4'b1011: y = 2'b11; Handled by default
        4'b1101: y = 2'b11;
        //4'b1111: y = 2'b11; Handled by default

        //Second if
        4'b0010: y = 2'b10; //The actual greycode
        //4'b0110: y = 2'b10; Handled by default
        //4'b1010: y = 2'b10; Handled by default
        //4'b1110: y = 2'b10; Handled by default

        //Third if
        4'b0100: y = 2'b01; //The actual greycode
        4'b1100: y = 2'b01;

        //Fourth if
        4'b1000: y = 2'b00; //The actual greycode
        default: y = a[1:0];
      endcase
endmodule
```

There is a more consise way of describing this circuit using a casez statement. The casez allows you to specify don't cares with '?'s in case statements.

```
module circuit2_casez(a, y);
  input [3:0] a;
  output reg [1:0] y;

  always @(*)
    //Need to handle the precidence in the if/else
    casez(a)
      //First if
      4'b???1: y = 2'b11;
      //Second if
      4'b??10: y = 2'b10;
      //Third if
      4'b?100: y = 2'b01;
      //Fourth if
      4'b1000: y = 2'b00;
      default: y = a[1:0];
    endcase
endmodule
```

Problem 3: Translating to Verilog [3,3 pts.]

Consider the decoder examples presented in lecture 3, page 5 of the notes. These examples used a fake HDL. Rewrite these examples in proper Verilog.

Solution:

Verilog Modules:

```
1  //Structural
2  module Decoder_strucural(
3    output x0,
4    output x1,
5    output x2,
6    output x3,
7    input   a,
8    input   b);
9
10   wire abar, bbar;
11   not(bbar, b);
12   not(abar, a);
13   and(x0, abar, bbar);
14   and(x1, abar, b);
15   and(x2, a, bbar);
16   and(x3, a, b);
17 endmodule
18
19 //Behavioral
20 module Decoder_behavioral(
21   output reg x0,
22   output reg x1,
23   output reg x2,
24   output reg x3,
25   input   a,
26   input   b);
27
28   always @(*) begin
29     case({a,b})
30       //You can concatenate on the left hand side
31       2'b00: {x3, x2, x1, x0} = 4'h1;
32       2'b01: {x3, x2, x1, x0} = 4'h2;
33       2'b10: {x3, x2, x1, x0} = 4'h4;
34       2'b11: {x3, x2, x1, x0} = 4'h8;
35     endcase
36   end
37
38 endmodule
```

See the appendix for an example testbench.

Problem 4: Testing a Grey-code to Binary Converter [6,4 pts.]

(a) Write a Verilog testbench that would instantiate and exhaustively test a 4-bit Gray-code to binary-code converter. (Hint: you may use the generator presented in class.)

(b) Draw the gate-level circuit diagram of the 4-bit Grey-code to binary converter.

Solution:

Part (a): Verilog Module:

```
// From Slides
module gray2bin1 (bin, gray);
 parameter SIZE = 8;
 output [SIZE-1:0] bin;
 input [SIZE-1:0] gray;

 genvar i;

  generate for (i=0; i<SIZE; i=i+1) begin:bitchain
         assign bin[i] = ^gray[SIZE-1:i];
 end endgenerate
endmodule
```

Verilog Testbench:

```
`timescale 1ns/1ns
module bin_gray_tester();
  reg [3:0] in;
  reg [3:0] out_bin2gray;
  wire [3:0] out_gray2bin;
  wire match;

  reg fail;

  initial in = 0;
  initial fail = 0;

  always @(*) begin
    case(in)
      4'b0000: out_bin2gray = 4'b0000;
      4'b0001: out_bin2gray = 4'b0001;
      4'b0010: out_bin2gray = 4'b0011;
```

```
18        4'b0011: out_bin2gray = 4'b0010;
19        4'b0100: out_bin2gray = 4'b0110;
20        4'b0101: out_bin2gray = 4'b0111;
21        4'b0110: out_bin2gray = 4'b0101;
22        4'b0111: out_bin2gray = 4'b0100;
23        4'b1000: out_bin2gray = 4'b1100;
24        4'b1001: out_bin2gray = 4'b1101;
25        4'b1010: out_bin2gray = 4'b1111;
26        4'b1011: out_bin2gray = 4'b1110;
27        4'b1100: out_bin2gray = 4'b1010;
28        4'b1101: out_bin2gray = 4'b1011;
29        4'b1110: out_bin2gray = 4'b1001;
30        4'b1111: out_bin2gray = 4'b1000;
31      endcase
32    end
33
34    //Instantiate the DUTs
35    gray2bin1 orig(.gray(out_bin2gray), .bin(out_gray2bin));
36
37    assign match = (in == out_gray2bin);
38
39    always #2 begin
40      in = in + 4'd1;
41    end
42
43    initial begin
44      $dumpfile("dump.vcd"); //Setup file dump (for waveform viewer)
45      $dumpvars; //Dump signals to dumpfile
46
47      #33
48      if(fail)
49        $display("Test FAIL");
50      else
51        $display("Test PASS");
52      $finish();
53    end
54
55    initial begin
56      #1
57      forever begin
58        $strobe("time: %4d, in: %b, out_bin2gray: %b, out_bin2gray: %b,
          ↪  match %b", $time, in, out_bin2gray, out_gray2bin, match);
59        #2;
60      end
61    end
62
63    initial begin
```

```
    #1
    forever begin
      if(!match)
        fail = 1;
      #2;
    end
  end
endmodule
```

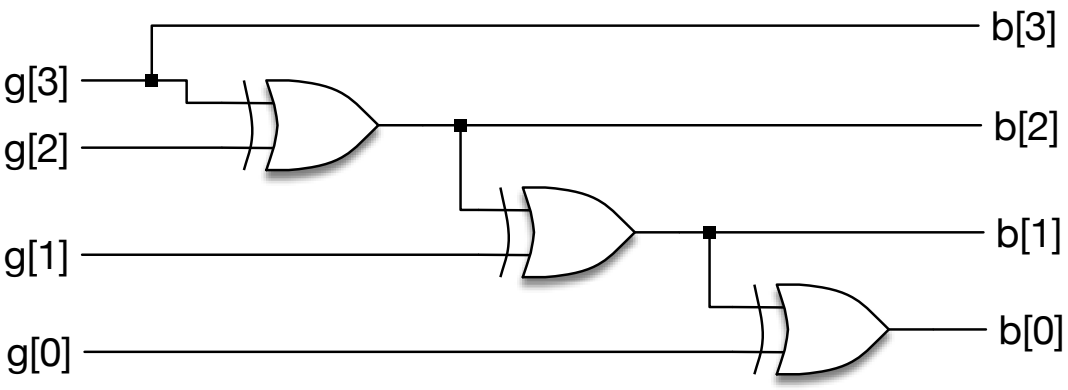
Another method is to use a binary to Gray-code converter to produce a series of gray codes. These Gray-codes can be fed to our Gray-code to binary converter which should convert to the origional binary number. The binary to Gray-code converter used here comes from Weste & Harris pg 471.

```
//Weste & Harris pg 471
//Binary -> Gray Code
//G_{N-1} = B_{N-1}
//G_{i} = B_{i+1} ^ B_{i} for N-1>i>=0

module bin2gray #(parameter N = 4)(
  input  [N-1:0] bin,
  output [N-1:0] gray);

  //Implementing Weste & Harris pg 471

  //Calculate the N-1 bit
  assign gray[N-1] = bin[N-1];

  genvar i;
  generate
    for(i = N-2; i>=0; i = i-1) begin:gray_chain
      assign gray[i] = bin[i+1] ^ bin[i];
    end
  endgenerate
endmodule
```

```
`timescale 1ns/1ns
module bin_gray_tester();
  reg [3:0] in;
  wire [3:0] out_bin2gray;
  wire [3:0] out_gray2bin;
  wire match;

  reg fail;

```

```
10    initial in = 0;
11    initial fail = 0;
12
13    //Instantiate the DUTs
14    bin2gray  dut(.bin(in), .gray(out_bin2gray));
15    gray2bin1 orig(.gray(out_bin2gray), .bin(out_gray2bin));
16
17    assign match = (in == out_gray2bin);
18
19    always #2 begin
20      in = in + 4'd1;
21    end
22
23    initial begin
24      $dumpfile("dump.vcd"); //Setup file dump (for waveform viewer)
25      $dumpvars; //Dump signals to dumpfile
26
27      #33
28      if(fail)
29        $display("Test FAIL");
30      else
31        $display("Test PASS");
32      $finish();
33    end
34
35    initial begin
36      #1
37      forever begin
38        $strobe("time: %4d, in: %b, out_bin2gray: %b, out_bin2gray: %b,
          ↪  match %b", $time, in, out_bin2gray, out_gray2bin, match);
39        #2;
40      end
41    end
42
43    initial begin
44      #1
45      forever begin
46        if(!match)
47          fail = 1;
48        #2;
49      end
50    end
51  endmodule
```

Part (b):



Problem 5: Implementing an Adder/Subtractor [7 pts.]

Subtraction, for instance A-B, can be implemented as A+(-B). Therefore, any adder circuit can be converted to a subtractor by converting B to its 2's complement, defined as ∼B+1.

Write a *structural* Verilog module as follows: `module add_sub(A, B, R, add)` which produces A+B if "add" is equal to 1 and produces A-B otherwise.

(hint, for an adder, the carry-in to the first stage provides an easy way to add 1 to the final result).

Solution:

Verilog Modules:

```
1  module full_adder (
2      input a,
3      input b,
4      input carry_in,
5      output sum,
6      output carry_out
7  );
8
9      wire aXORb;
10     xor(aXORb, a, b);
11     xor(sum, aXORb, carry_in);
12
13     wire carryAB;
14     and(carryAB, a, b);
15
16     wire carryCIN;
17     and(carryCIN, aXORb, carry_in);
18
19     or(carry_out, carryAB, carryCIN);
20 endmodule
21
```

```
22 module structural_adder_subtractor
23   #(parameter N=14)(
24     input [N-1:0] A,
25     input [N-1:0] B,
26     output [N-1:0] R,
27     input add);
28
29   wire [N:0] carry;
30   assign carry[0] = ~add;
31
32   wire [N-1:0] second_operand;
33
34   genvar i;
35   generate
36     for(i = 0; i<N; i = i + 1) begin:adder_chain
37       xor(second_operand[i], B[i], ~add);
38
39       full_adder fa(.a(A[i]),
40                     .b(second_operand[i]),
41                     .carry_in(carry[i]),
42                     .sum(R[i]),
43                     .carry_out(carry[i+1]));
44     end
45   endgenerate
46
47 endmodule
```

Problem 6: 251A only — *Optional* Challenge Question for 151 [5,5 pts.]

(a) Write the Verilog description for a LUT3 (3 input LUT with 1 output) module. It will have 4 inputs. The first three are the single bit LUT inputs: a, b, c. The last input, F, specifies the LUT function as an 8 bit number. For instance if F=8'h01, the LUT implements an AND gate. If F = 8'h7F, it implements an OR gate.

(b) Using only the LUT3 module you created, write a LUT4 module using *structural* Verilog.

Solution:

Verilog Modules:

```
1 module lut3(
2   input a,
3   input b,
4   input c,
5   input [7:0] F,
```

```
    output out);

    wire [2:0] index;
    assign index = {~c, ~b, ~a};

    //The index of F is the opposite of that of the index wire
    //The LSB of F is the case when a, b, and c are all true
    //The MSB of F is the case when a, b, and c are all false
    //This can be addressed by inverting the inputs

    //Slice based on the index (this synthesizable - at least in Vivado - infers a bit selector with index port -> becomes luts and muxes)
    assign out = F[index];
endmodule

module lut4(
    input a,
    input b,
    input c,
    input d,
    input [15:0] F,
    output out);
    //Assuming d is the MSB of the input

    //=== Build a Tree with LUT3s ===
    wire d_true_lut_out;
    wire d_false_lut_out;

    //The lower half of F is when d is true.  Within that half, the LSB is when a, b, c are true
    lut3 f_true_lut(.a(a), .b(b), .c(c), .F(F[7:0]), .out(d_true_lut_out));

    //The upper half of F is when d is false. Within that half, the LSB is when a, b, c are true
    lut3 f_false_lut(.a(a), .b(b), .c(c), .F(F[15:8]),
    .out(d_false_lut_out));

    //Use a LUT3 as a mux
    //With LSB = All false, F would be 8'b11100100.  With LSB = all true, F is 8'b00100111
    lut3 mux_lut(.a(d), .b(d_false_lut_out), .c(d_true_lut_out),
    .F(8'b00100111), .out(out));
endmodule
```

Appendix

Solution:

Problem 2: Example Testbench:

```
`timescale 1ns/1ns
module citcuit2_tester();
  reg [3:0] in;
  wire [1:0] out_orig;
  wire [1:0] out_case;

  wire match;
  reg fail;

  initial in = 0;
  initial fail = 0;

  //Instantiate the DUTs
  circuit2_case  dut(.a(in), .y(out_case));
  circuit2_orig orig(.a(in), .y(out_orig));

  assign match = out_orig == out_case;

  //Increment Input Every 2 Steps (Check Between Increments)
  always #2 begin
    in = in + 4'd1;
  end

  initial begin
    $dumpfile("dump.vcd"); //Setup file dump (for waveform viewer)
    $dumpvars; //Dump signals to dumpfile

    #33
    if(fail)
      $display("Test FAIL");
    else
      $display("Test PASS");
    $finish();
  end

  //Print
  initial begin
    #1
    forever begin
      $strobe("time: %4d, in: %b, out_orig: %b, out_case: %b, match %b",
      ↪  $time, in, out_orig, out_case, match);
      #2;
    end
```

```
43    end
44
45    //Check for Failure 1 Step After Input Change
46    initial begin
47      #1
48      forever begin
49        if(!match)
50          fail = 1;
51        #2;
52      end
53    end
54
55  endmodule
```

Problem 3: Example Testbench:

```
1   `timescale 1ns/1ns
2   module decoder_tester();
3     reg  [1:0] in;
4
5     wire x0_stru;
6     wire x1_stru;
7     wire x2_stru;
8     wire x3_stru;
9     wire [3:0] out_stru;
10
11    wire x0_beha;
12    wire x1_beha;
13    wire x2_beha;
14    wire x3_beha;
15    wire [3:0] out_beha;
16
17    reg fail;
18    wire match;
19
20    initial in = 0;
21    initial fail = 0;
22
23    assign out_stru = {x0_stru, x1_stru, x2_stru, x3_stru};
24    assign out_beha = {x0_beha, x1_beha, x2_beha, x3_beha};
25
26    //Instantiate the DUTs
27    Decoder_strucural  stru(.x0(x0_stru), .x1(x1_stru), .x2(x2_stru),
      ↪  .x3(x3_stru), .a(in[1]), .b(in[0]));
28    Decoder_behavioral beha(.x0(x0_beha), .x1(x1_beha), .x2(x2_beha),
      ↪  .x3(x3_beha), .a(in[1]), .b(in[0]));
```

```
29
30    assign match = out_stru == out_beha;
31
32    always #2 begin
33      in = in + 2'd1;
34    end
35
36    initial begin
37      $dumpfile("dump.vcd"); //Setup file dump (for waveform viewer)
38      $dumpvars; //Dump signals to dumpfile
39
40      #9
41      if(fail)
42        $display("Test FAIL");
43      else
44        $display("Test PASS");
45      $finish();
46    end
47
48    initial begin
49      #1
50      forever begin
51        $strobe("time: %4d, in: %b, out_stru: %b, out_beha: %b, match %b",
          ↪  $time, in, out_stru, out_beha, match);
52        #2;
53      end
54    end
55
56    initial begin
57      #1
58      forever begin
59        if(!match)
60          fail = 1;
61        #2;
62      end
63    end
64  endmodule
```

Problem 5: Example Testbench:

```
1   module behavioral_adder_subtractor
2     #(parameter N=14)(
3       input [N:0] A,
4       input [N:0] B,
5       output [N:0] R,
6       input add);
```

```
7
8     assign R = add ? A+B : A-B;
9
10  endmodule
11
12  `timescale 1ns/1ns
13  module adder_tester();
14    reg  [6:0] in;
15
16    wire signed [2:0] out_stru;
17    wire signed [2:0] out_beha;
18
19    reg fail;
20    wire match;
21
22    initial in = 0;
23    initial fail = 0;
24
25    wire signed [2:0] op_a;
26    wire signed [2:0] op_b;
27    wire op_add;
28    assign op_a = in[2:0];
29    assign op_b = in[5:3];
30    assign op_add = in[6];
31
32
33    //Instantiate the DUTs
34    structural_adder_subtractor #(.N(3)) stru (.A(op_a), .B(op_b),
      ↪  .R(out_stru), .add(op_add));
35    behavioral_adder_subtractor #(.N(3)) beha (.A(op_a), .B(op_b),
      ↪  .R(out_beha), .add(op_add));
36
37    assign match = out_stru == out_beha;
38
39    always #2 begin
40      in = in + 2'd1;
41    end
42
43    initial begin
44      $dumpfile("dump.vcd"); //Setup file dump (for waveform viewer)
45      $dumpvars; //Dump signals to dumpfile
46
47      #257
48      if(fail)
49        $display("Test FAIL");
50      else
51        $display("Test PASS");
```

```
52      $finish();
53    end
54
55    initial begin
56      #1
57      forever begin
58        $strobe("time: %4d, a: %2d, b: %2d, add: %b, out_stru: %2d,
          ↪  out_beha: %2d, match %b", $time, op_a, op_b, op_add, out_stru,
          ↪  out_beha, match);
59        #2;
60      end
61    end
62
63    initial begin
64      #1
65      forever begin
66        if(!match)
67          fail = 1;
68        #2;
69      end
70    end
71  endmodule
```

Problem 6: Example Testbench:

```
1   `timescale 1ns/1ns
2   module lut4_tester();
3     reg  [5:0] in;
4
5     wire out_lut4;
6
7     reg fail;
8     wire match;
9
10    initial in = 0;
11    initial fail = 0;
12
13    wire op_a;
14    wire op_b;
15    wire op_c;
16    wire op_d;
17    wire [1:0] func_ind;
18
19    assign op_a = in[0];
20    assign op_b = in[1];
21    assign op_c = in[2];
```

```
22    assign op_d = in[3];
23    assign func_ind = in[5:4];
24
25    //Set the function to test (we'll test 4 different functions)
26    reg [15:0] func;
27    initial func = 0;
28    always @(*) begin
29      case(func_ind)
30        2'd0: func = 16'h7FFF; //OR all
31        2'd1: func = 16'h0001; //AND all
32        2'd2: func = 16'b0010001000000011; //dcb+(!d)b(!a)
33        2'd3: func = 16'b0100000000000000; //(!d)(!c)(!b)a
34        default: func = 16'h0000;
35      endcase
36    end
37
38    //Test against static
39    reg out_static;
40    initial out_static = 0;
41    always @(*) begin
42      case(func_ind)
43        2'd0: out_static = op_a | op_b | op_c | op_d; //OR all
44        2'd1: out_static = op_a & op_b & op_c & op_d; //AND all
45        2'd2: out_static = (op_d & op_c & op_b) | ((~op_d) & op_b &
          ↪  (~op_a)); //dcb+(!d)b(!a)
46        2'd3: out_static = (~op_d) & (~op_c) & (~op_b) & op_a;
          ↪  //(!d)(!c)(!b)a
47        default: out_static = 16'h0000;
48      endcase
49    end
50
51    //Instantiate the DUTs
52    lut4 lut(.a(op_a), .b(op_b), .c(op_c), .d(op_d), .F(func),
      ↪  .out(out_lut4));
53
54    assign match = out_lut4 == out_static;
55
56    always #2 begin
57      in = in + 6'd1;
58    end
59
60    initial begin
61      $dumpfile("dump.vcd"); //Setup file dump (for waveform viewer)
62      $dumpvars; //Dump signals to dumpfile
63
64      #129
65      if(fail)
```

```
66        $display("Test FAIL");
67      else
68        $display("Test PASS");
69      $finish();
70    end
71
72    initial begin
73      #1
74      forever begin
75        $strobe("time: %4d, a: %b, b: %b, c: %b, d: %b, func_ind: %d, func:
          ↪  %b, out_lut4: %b, out_static: %b, match %b", $time, op_a, op_b,
          ↪  op_c, op_d, func_ind, func, out_lut4, out_static, match);
76        #2;
77      end
78    end
79
80    initial begin
81      #1
82      forever begin
83        if(!match)
84          fail = 1;
85        #2;
86      end
87    end
88  endmodule
```