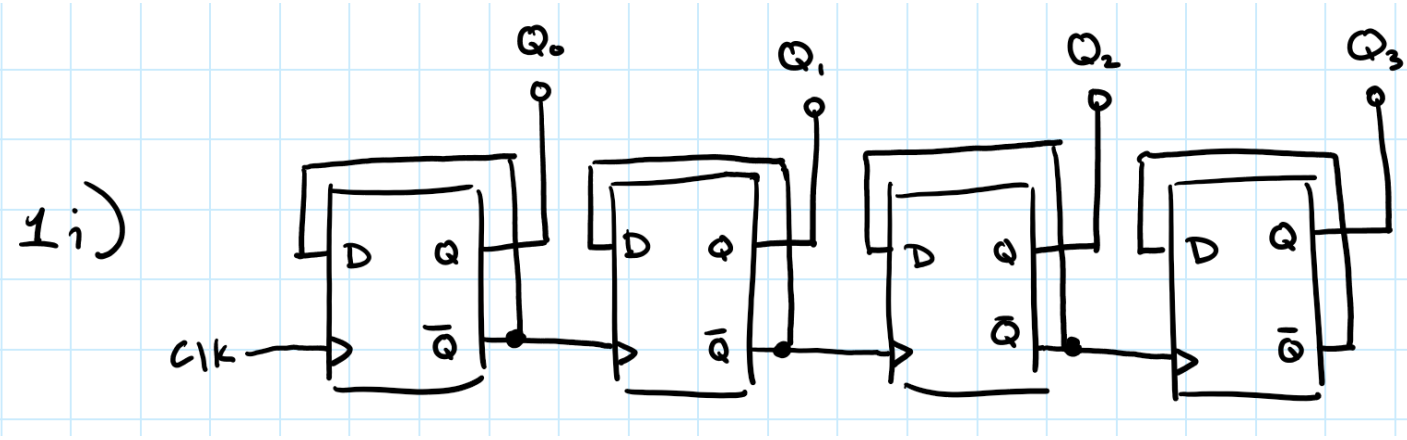
# ECE 581 - Project 1 Report

*Fall 2021*

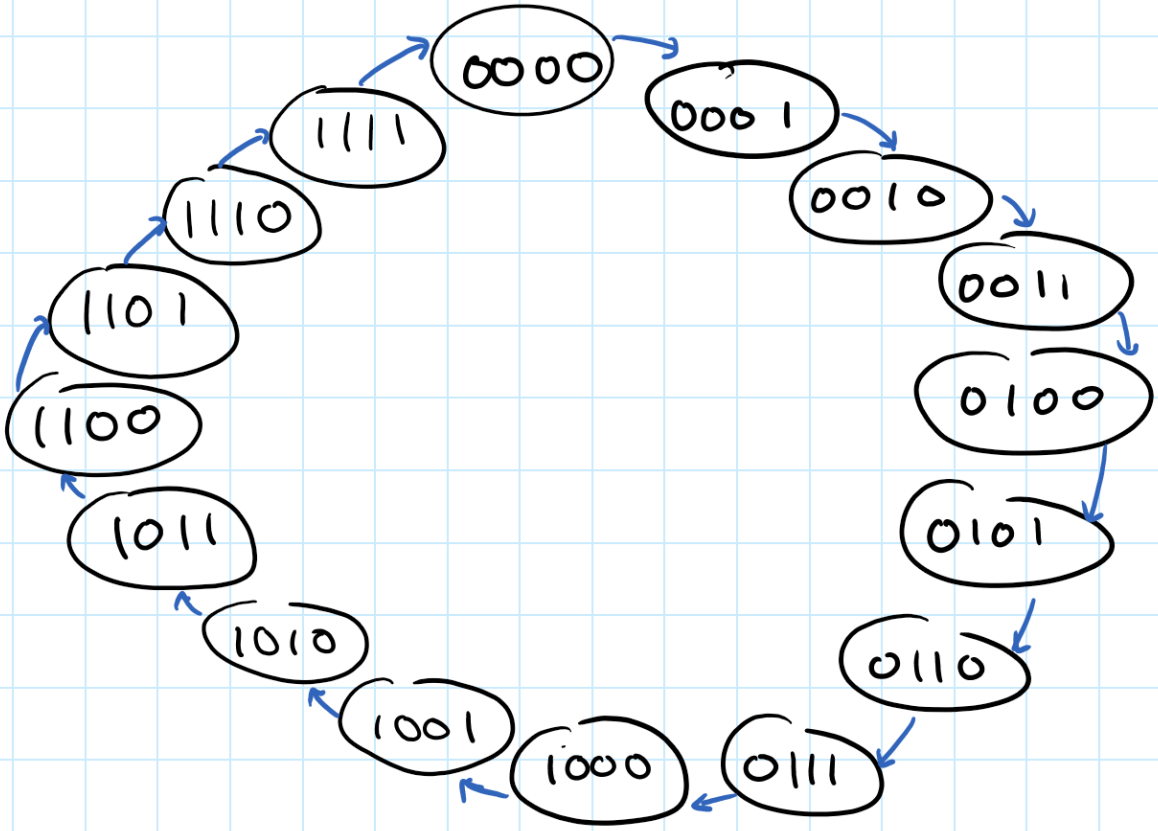
Chuck Faber (cfaber@pdx.edu)
Chris Mersman (cmersman@pdx.edu)

Problem 1

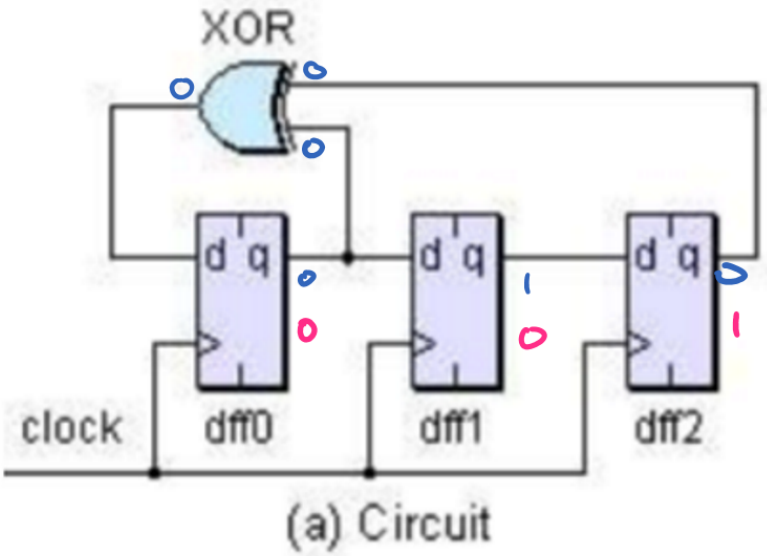
1 i)



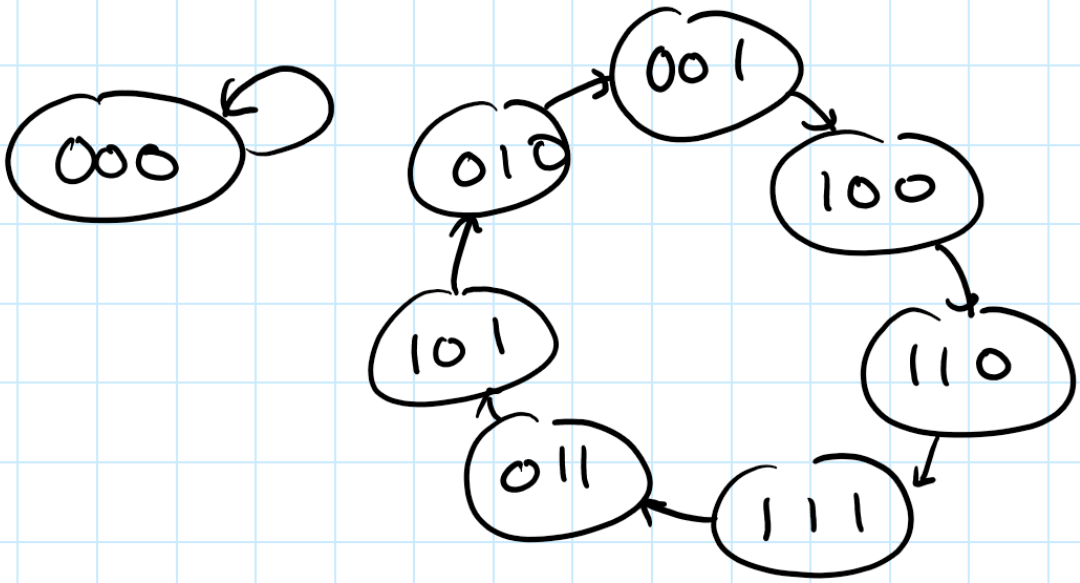

1 ii)

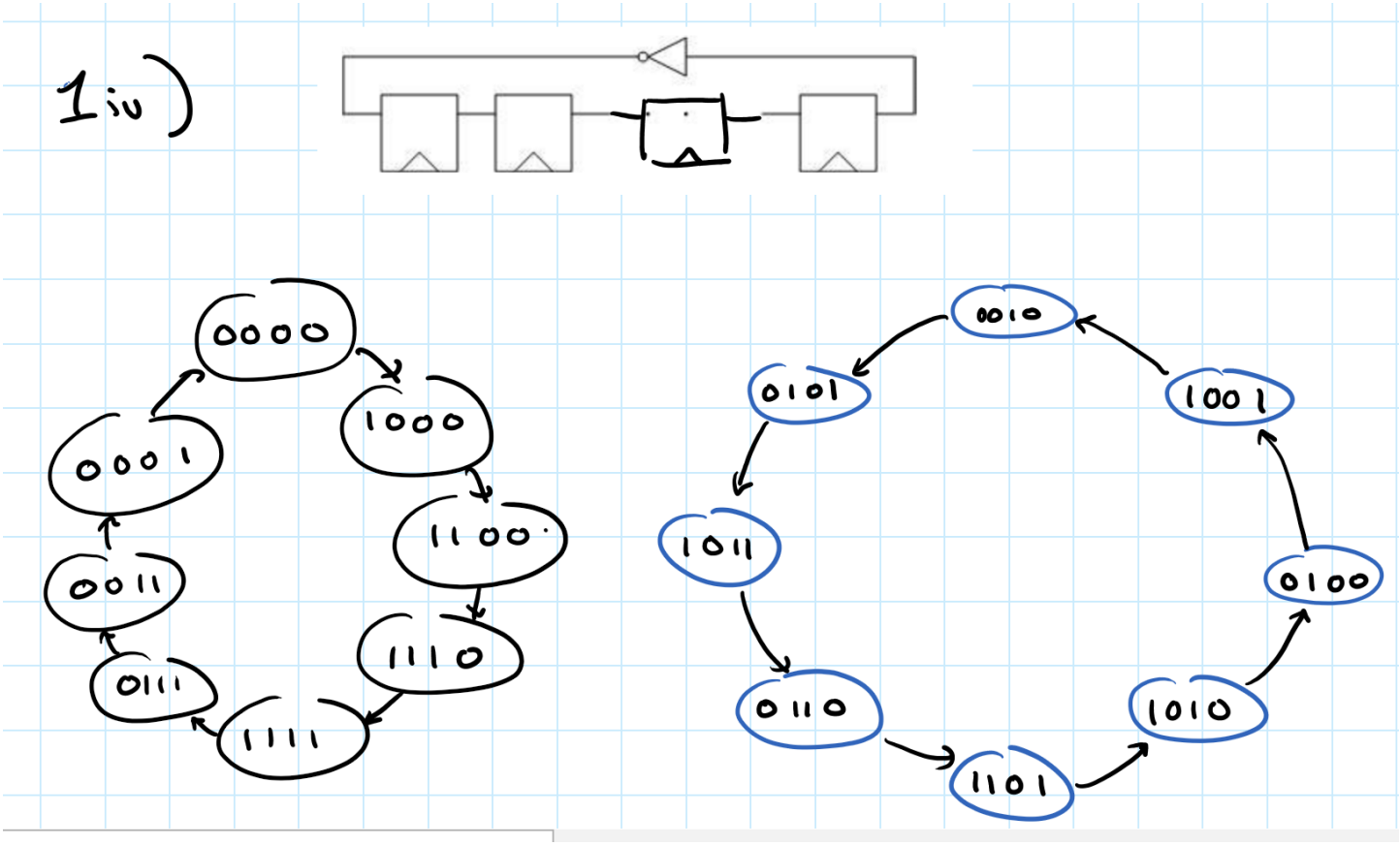


1 iii)



(a) Circuit






## Problem 2

**Priority Encoder Dataflow Model**

```
module priority_enc_df #(parameter N = 3) (
    input [2**N-1:0] in,
    output logic [N-1:0] out
    );

    function logic [N-1:0] pri(input logic [2**N-1:0] in);
        begin
            logic [N-1:0] out;
            for (int i = 0; i < 2**N; i++) begin
                out = in[i] ? i : out;
            end
            return out;
        end
    endfunction

    assign out = pri(in);
endmodule
```

**Priority Encoder Algorithmic Model**

```
module priority_enc_alg #(parameter N = 3) (
    input [2**N-1:0] in,
    output logic [N-1:0] out
    );

    logic [N-1:0] mask = '1;
```

```
    always_comb begin
        for (int i = 0; i < 2**N; i++) begin
            if (in[i]) out = i & mask;
        end
    end

endmodule
```

**Priority Encoder Testbench**

```
module top();
    parameter N = 3;
    parameter nTESTS = 20;
    logic [2**N-1:0] in;
    logic [N-1:0] out_df, out_alg;

    priority_enc_df #(N) df (
        .out(out_df),
        .*
    );
    priority_enc_alg #(N) alg (
        .out(out_alg),
        .*
    );

    initial begin
        $display("2N+1 Sequential Tests");
        for (int i=0; i <= 2**N; i++) begin
            in = i;
            #10;
            $display("in: %08b\tout_df: %03b\tout_alg: %03b", in, out_df, out_alg);
        end

        $display("Randomized Tests");
        for (int i=0; i < nTESTS; i++) begin
            in = $random() & {2**N{1'b1}};
            #10;
            $display("in: %08b\tout_df: %03b\tout_alg: %03b", in, out_df, out_alg);
        end
    end
endmodule
```

**Priority Encoder Transcript**

```
# 2020.1

# vsim -c top
# Start time: 17:46:37 on Oct 24,2021
# Loading sv_std.std
# Loading work.top
# Loading work.priority_enc_df
# Loading work.priority_enc_alg
VSIM 1> run -all
# 2N+1 Sequential Tests
# in: 00000000  out_df: xxx     out_alg: xxx
```

```
# in: 00000001  out_df: 000     out_alg: 000
# in: 00000010  out_df: 001     out_alg: 001
# in: 00000011  out_df: 001     out_alg: 001
# in: 00000100  out_df: 010     out_alg: 010
# in: 00000101  out_df: 010     out_alg: 010
# in: 00000110  out_df: 010     out_alg: 010
# in: 00000111  out_df: 010     out_alg: 010
# in: 00001000  out_df: 011     out_alg: 011
# Randomized Tests
# in: 00100100  out_df: 101     out_alg: 101
# in: 10000001  out_df: 111     out_alg: 111
# in: 00001001  out_df: 011     out_alg: 011
# in: 01100011  out_df: 110     out_alg: 110
# in: 00001101  out_df: 011     out_alg: 011
# in: 10001101  out_df: 111     out_alg: 111
# in: 01100101  out_df: 110     out_alg: 110
# in: 00010010  out_df: 100     out_alg: 100
# in: 00000001  out_df: 000     out_alg: 000
# in: 00001101  out_df: 011     out_alg: 011
# in: 01110110  out_df: 110     out_alg: 110
# in: 00111101  out_df: 101     out_alg: 101
# in: 11101101  out_df: 111     out_alg: 111
# in: 10001100  out_df: 111     out_alg: 111
# in: 11111001  out_df: 111     out_alg: 111
# in: 11000110  out_df: 111     out_alg: 111
# in: 11000101  out_df: 111     out_alg: 111
# in: 10101010  out_df: 111     out_alg: 111
# in: 11100101  out_df: 111     out_alg: 111
# in: 01110111  out_df: 110     out_alg: 110
```

## Problem 3

**CLA Dataflow Model**

```
//////////////////////////////////////////////////////////////////////
// Problem 3a. Write code for a 32 bit CLA adder in dataflow model. //
//////////////////////////////////////////////////////////////////////

module CLA_adder #(parameter nBITS = 32) (
    output logic [nBITS-1:0] sum,
    output logic co,
    input logic [nBITS-1:0] ain, bin,
    input cin
);    genvar i;

    wire [nBITS-1:0] P, G;
    wire [nBITS:0] C;

    assign C[0] = cin;
    assign co = C[nBITS];

    generate
        for (i=0; i<nBITS; i++) begin
            assign P[i] = ain[i] ^ bin[i];
            assign G[i] = ain[i] & bin[i];
        end
```

```
        for (i=0; i<nBITS; i++) begin
            assign sum[i] = P[i] ^ C[i];
            assign C[i+1] = G[i] | P[i]&C[i];
        end
    endgenerate

endmodule
```

**CLA Algorithmic Model**

```
////////////////////////////////////////////////////
// Problem 3b. Write the CLA in algorithmic model. //
////////////////////////////////////////////////////

module CLA_adder_alg #(parameter nBITS = 32) (
    output logic [nBITS-1:0] sum,
    output logic co,
    input logic [nBITS-1:0] ain, bin,
    input cin
);
    logic [nBITS-1:0] P, G;
    logic [nBITS:0] C;

    always_comb begin
        P = ain ^ bin;
        G = ain & bin;
        C[0] = cin;
        for (int i=0; i < nBITS; i++) begin
            C[i+1] = G[i] | (P[i] & C[i]);
        end
        sum = P ^ C[nBITS-1:0];
        co = C[nBITS];
    end

endmodule
```

**CLA Testbench**

```
module top ();
    parameter NTESTS = 20;
    logic [31:0] sum, ain, bin;
    logic co, cin;
    int failure = 0;

    CLA_adder_alg CLA1(.*);

    initial begin
        cin = 0;
        for (int i = 0; i < NTESTS; i++) begin
            ain = $random() & 32'hFFFFFFFF;
            bin = $random() & 32'hFFFFFFFF;
            cin = $random() & 1'h1;
            #10;
            `ifdef VERBOSE
                $display("ain=0x%08h  bin=0x%08h  cin=%01b. Expected: 0x%08h  Received: 0x%08h", ain, bin,
```

```
cin, ain+bin+cin, sum);
            `endif
            assert (sum == (ain+bin+cin)) else begin
                failure++;
                $display("ERROR: \nain=%032b  \nbin=%032b  \ncin=%01b. \nExpected: %032b  \nReceived:
%032b", ain, bin, cin, ain+bin+cin, sum);
            end
        end
        if (!failure) $display("SUCCESS: No errors occured in testing!");
        else $display("FAILURE: %0d/%0d mismatches occurred in testbench.", failure, NTESTS);
    end

endmodule
```

**CLA Transcript**

```
# vsim -c top
# Start time: 21:21:02 on Oct 21,2021
# Loading sv_std.std
# Loading work.top
# Loading work.CLA_adder_alg
VSIM 1> run -all
# ain=0x12153524  bin=0xc0895e81  cin=1. Expected: 0xd29e93a6  Received: 0xd29e93a6
# ain=0xb1f05663  bin=0x06b97b0d  cin=1. Expected: 0xb8a9d171  Received: 0xb8a9d171
# ain=0xb2c28465  bin=0x89375212  cin=1. Expected: 0x3bf9d678  Received: 0x3bf9d678
# ain=0x06d7cd0d  bin=0x3b23f176  cin=1. Expected: 0x41fbbe84  Received: 0x41fbbe84
# ain=0x76d457ed  bin=0x462df78c  cin=1. Expected: 0xbd024f7a  Received: 0xbd024f7a
# ain=0xe33724c6  bin=0xe2f784c5  cin=0. Expected: 0xc62ea98b  Received: 0xc62ea98b
# ain=0x72aff7e5  bin=0xbbd27277  cin=0. Expected: 0x2e826a5c  Received: 0x2e826a5c
# ain=0x47ecdb8f  bin=0x793069f2  cin=0. Expected: 0xc11d4581  Received: 0xc11d4581
# ain=0xf4007ae8  bin=0xe2ca4ec5  cin=0. Expected: 0xd6cac9ad  Received: 0xd6cac9ad
# ain=0xde8e28bd  bin=0x96ab582d  cin=1. Expected: 0x753980eb  Received: 0x753980eb
# ain=0xb1ef6263  bin=0x0573870a  cin=0. Expected: 0xb762e96d  Received: 0xb762e96d
# ain=0x10642120  bin=0x557845aa  cin=1. Expected: 0x65dc66cb  Received: 0x65dc66cb
# ain=0xcb203e96  bin=0x8983b813  cin=1. Expected: 0x54a3f6aa  Received: 0x54a3f6aa
# ain=0xa9a7d653  bin=0x359fdd6b  cin=1. Expected: 0xdf47b3bf  Received: 0xdf47b3bf
# ain=0x81174a02  bin=0xd7563eae  cin=1. Expected: 0x586d88b1  Received: 0x586d88b1
# ain=0xe7c572cf  bin=0x11844923  cin=0. Expected: 0xf949bbf2  Received: 0xf949bbf2
# ain=0xe5730aca  bin=0x9e314c3c  cin=0. Expected: 0x83a45706  Received: 0x83a45706
# ain=0x452e618a  bin=0x20c4b341  cin=0. Expected: 0x65f314cb  Received: 0x65f314cb
# ain=0x3c20f378  bin=0xc48a1289  cin=1. Expected: 0x00ab0602  Received: 0x00ab0602
# ain=0x5b0265b6  bin=0x634bf9c6  cin=0. Expected: 0xbe4e5f7c  Received: 0xbe4e5f7c
# SUCCESS: No errors occured in testing!
```

## Problem 4

**ID Detector Algorithmic Model**

```
// Problem 4
// Given a 9 bit boolean vector M, write an SV model for the following:
// If the number of high bits in M equals the first digit of the PSU ID number
// of one member of your group, the detector output 1, else 0.
// a. algorithmic model (always_comb) with detector having 10ns delay
// b. dataflow model (assign) with 10ns delay
//
// Assumptions: all PSU IDs are 9 digits long. The first digit is the digit in the ones place
```

```
`timescale 1ns/1ns

module detector_alg (
    input [8:0] M,
    input unsigned [31:0] ID1, ID2,
    output logic out
);

    int ones=0;
    logic out_tmp;

    always_comb begin
        ones = 0;
        for (int i = 0; i < 9; i++) begin
            ones += M[i];
        end
        out_tmp = ((ones==ID1%10) || (ones==ID2%10)) ? 1'b1 : 1'b0;
    end

    assign #10 out = out_tmp;
endmodule
```

**Detector Dataflow Model**

```
module detector_df (
    input [8:0] M,
    input unsigned [31:0] ID1, ID2,
    output logic out
);

    logic [4:0] ones;
    logic [2:0] ms3, mid3, ls3;
    logic [1:0] ms3_cnt, mid3_cnt, ls3_cnt;

    assign ms3 = M[8:6];
    assign mid3 = M[5:3];
    assign ls3 = M[2:0];

    assign ms3_cnt[1] = (ms3[1]&ms3[0]) | (ms3[2]&!ms3[1]&ms3[0]) | (ms3[2]&ms3[1]&!ms3[0]);
    assign ms3_cnt[0] = (ms3[2]&!ms3[1]&!ms3[0]) | (!ms3[2]&!ms3[1]&ms3[0]) | (ms3[2]&ms3[1]&ms3[0]) |
(!ms3[2]&ms3[1]&!ms3[0]);

    assign mid3_cnt[1] = (mid3[1]&mid3[0]) | (mid3[2]&!mid3[1]&mid3[0]) | (mid3[2]&mid3[1]&!mid3[0]);
    assign mid3_cnt[0] = (mid3[2]&!mid3[1]&!mid3[0]) | (!mid3[2]&!mid3[1]&mid3[0]) |
(mid3[2]&mid3[1]&mid3[0]) | (!mid3[2]&mid3[1]&!mid3[0]);

    assign ls3_cnt[1] = (ls3[1]&ls3[0]) | (ls3[2]&!ls3[1]&ls3[0]) | (ls3[2]&ls3[1]&!ls3[0]);
    assign ls3_cnt[0] = (ls3[2]&!ls3[1]&!ls3[0]) | (!ls3[2]&!ls3[1]&ls3[0]) | (ls3[2]&ls3[1]&ls3[0]) |
(!ls3[2]&ls3[1]&!ls3[0]);

    assign ones = ms3_cnt + mid3_cnt + ls3_cnt;
    assign #10 out = (ones == (ID1 % 10)) || (ones == (ID2 % 10));
endmodule
```

**ID Detector Testbench**

```
module top();
    logic [8:0] M;
    int ID1, ID2;
    logic out_da, out_df;
    int failure=0;
    int fd;

    detector_alg da (
        .out(out_da),
        .*
    );
    detector_df dd (
        .out(out_df),
        .*
    );

    initial begin
        // Open file in append mode
        fd = $fopen("./p1_4.log", "a");
        if (fd)
            $display("File opened successfully.");
        else begin
            $display("File was not opened successfully.");
            $stop;
        end
    end

    initial begin
        ID1 = 985740900;                // Chuck's ID
        ID2 = $urandom() % 1000000000;  // Random ID
        $fdisplay(fd, "T(ns):\tID1: %09d\tID2: %09d", ID1, ID2);
        for (int i = 0; i < (2**9); i++) begin
            M = i & 9'h1FF;
            #15;
            $fdisplay(fd, "@%04t\tM: %09b\tOUT_ALG: %01b\tOUT_DF: %01b", $time, M, out_da, out_df);
            `ifdef VERBOSE
            $display("@%04t\tM: %09b\tID1: %09d\tID2: %09d\tExpected: %01b\tOUT_DA: %01b\tOUT_DF: %01b",
$time, M, ID1, ID2, ($countones(M)==ID1%10 || $countones(M)==ID2%10), out_da, out_df);
            `endif
            alg: assert ((out_da == ($countones(M)==ID1%10)) || (out_da == ($countones(M)==ID2%10))) else
begin
                failure++;
                $display("ERROR: M: %09b\tID1: %09d\tID2: %09d\tExpected: %01b\tOUT_DA: %01b", M, ID1, ID2,
($countones(M)==ID1%10 || $countones(M)==ID2%10), out_da);
            end
            df: assert ((out_df == ($countones(M)==ID1%10)) || (out_df == ($countones(M)==ID2%10))) else
begin
                failure++;
                $display("ERROR: M: %09b\tID1: %09d\tID2: %09d\tExpected: %01b\tOUT_DF: %01b", M, ID1, ID2,
($countones(M)==ID1%10 || $countones(M)==ID2%10), out_df);
            end
        end
        $fclose(fd);
    end

endmodule
```

## ID Detector Log File

```
T(ns):  ID1: 985740900  ID2: 318257127
@  15   M: 000000000    OUT_ALG: 1  OUT_DF: 1
@  30   M: 000000001    OUT_ALG: 0  OUT_DF: 0
@  45   M: 000000010    OUT_ALG: 0  OUT_DF: 0
@  60   M: 000000011    OUT_ALG: 0  OUT_DF: 0
@  75   M: 000000100    OUT_ALG: 0  OUT_DF: 0
@  90   M: 000000101    OUT_ALG: 0  OUT_DF: 0
@ 105   M: 000000110    OUT_ALG: 0  OUT_DF: 0
@ 120   M: 000000111    OUT_ALG: 0  OUT_DF: 0
@ 135   M: 000001000    OUT_ALG: 0  OUT_DF: 0
@ 150   M: 000001001    OUT_ALG: 0  OUT_DF: 0
@ 165   M: 000001010    OUT_ALG: 0  OUT_DF: 0
@ 180   M: 000001011    OUT_ALG: 0  OUT_DF: 0
@ 195   M: 000001100    OUT_ALG: 0  OUT_DF: 0
@ 210   M: 000001101    OUT_ALG: 0  OUT_DF: 0
@ 225   M: 000001110    OUT_ALG: 0  OUT_DF: 0
@ 240   M: 000001111    OUT_ALG: 0  OUT_DF: 0
@ 255   M: 000010000    OUT_ALG: 0  OUT_DF: 0
@ 270   M: 000010001    OUT_ALG: 0  OUT_DF: 0
@ 285   M: 000010010    OUT_ALG: 0  OUT_DF: 0
@ 300   M: 000010011    OUT_ALG: 0  OUT_DF: 0
@ 315   M: 000010100    OUT_ALG: 0  OUT_DF: 0
@ 330   M: 000010101    OUT_ALG: 0  OUT_DF: 0
@ 345   M: 000010110    OUT_ALG: 0  OUT_DF: 0
@ 360   M: 000010111    OUT_ALG: 0  OUT_DF: 0
@ 375   M: 000011000    OUT_ALG: 0  OUT_DF: 0
@ 390   M: 000011001    OUT_ALG: 0  OUT_DF: 0
@ 405   M: 000011010    OUT_ALG: 0  OUT_DF: 0
@ 420   M: 000011011    OUT_ALG: 0  OUT_DF: 0
@ 435   M: 000011100    OUT_ALG: 0  OUT_DF: 0
@ 450   M: 000011101    OUT_ALG: 0  OUT_DF: 0
@ 465   M: 000011110    OUT_ALG: 0  OUT_DF: 0
@ 480   M: 000011111    OUT_ALG: 0  OUT_DF: 0
@ 495   M: 000100000    OUT_ALG: 0  OUT_DF: 0
@ 510   M: 000100001    OUT_ALG: 0  OUT_DF: 0
@ 525   M: 000100010    OUT_ALG: 0  OUT_DF: 0
@ 540   M: 000100011    OUT_ALG: 0  OUT_DF: 0
@ 555   M: 000100100    OUT_ALG: 0  OUT_DF: 0
@ 570   M: 000100101    OUT_ALG: 0  OUT_DF: 0
@ 585   M: 000100110    OUT_ALG: 0  OUT_DF: 0
@ 600   M: 000100111    OUT_ALG: 0  OUT_DF: 0
@ 615   M: 000101000    OUT_ALG: 0  OUT_DF: 0
@ 630   M: 000101001    OUT_ALG: 0  OUT_DF: 0
@ 645   M: 000101010    OUT_ALG: 0  OUT_DF: 0
@ 660   M: 000101011    OUT_ALG: 0  OUT_DF: 0
@ 675   M: 000101100    OUT_ALG: 0  OUT_DF: 0
@ 690   M: 000101101    OUT_ALG: 0  OUT_DF: 0
@ 705   M: 000101110    OUT_ALG: 0  OUT_DF: 0
@ 720   M: 000101111    OUT_ALG: 0  OUT_DF: 0
@ 735   M: 000110000    OUT_ALG: 0  OUT_DF: 0
@ 750   M: 000110001    OUT_ALG: 0  OUT_DF: 0
@ 765   M: 000110010    OUT_ALG: 0  OUT_DF: 0
@ 780   M: 000110011    OUT_ALG: 0  OUT_DF: 0
@ 795   M: 000110100    OUT_ALG: 0  OUT_DF: 0
@ 810   M: 000110101    OUT_ALG: 0  OUT_DF: 0
@ 825   M: 000110110    OUT_ALG: 0  OUT_DF: 0
@ 840   M: 000110111    OUT_ALG: 0  OUT_DF: 0
@ 855   M: 000111000    OUT_ALG: 0  OUT_DF: 0
@ 870   M: 000111001    OUT_ALG: 0  OUT_DF: 0
@ 885   M: 000111010    OUT_ALG: 0  OUT_DF: 0
@ 900   M: 000111011    OUT_ALG: 0  OUT_DF: 0
@ 915   M: 000111100    OUT_ALG: 0  OUT_DF: 0
@ 930   M: 000111101    OUT_ALG: 0  OUT_DF: 0
@ 945   M: 000111110    OUT_ALG: 0  OUT_DF: 0
@ 960   M: 000111111    OUT_ALG: 0  OUT_DF: 0
@ 975   M: 001000000    OUT_ALG: 0  OUT_DF: 0
@ 990   M: 001000001    OUT_ALG: 0  OUT_DF: 0
@1005   M: 001000010    OUT_ALG: 0  OUT_DF: 0
@1020   M: 001000011    OUT_ALG: 0  OUT_DF: 0
@1035   M: 001000100    OUT_ALG: 0  OUT_DF: 0
@1050   M: 001000101    OUT_ALG: 0  OUT_DF: 0
@1065   M: 001000110    OUT_ALG: 0  OUT_DF: 0
@1080   M: 001000111    OUT_ALG: 0  OUT_DF: 0
@1095   M: 001001000    OUT_ALG: 0  OUT_DF: 0
@1110   M: 001001001    OUT_ALG: 0  OUT_DF: 0
@1125   M: 001001010    OUT_ALG: 0  OUT_DF: 0
@1140   M: 001001011    OUT_ALG: 0  OUT_DF: 0
@1155   M: 001001100    OUT_ALG: 0  OUT_DF: 0
@1170   M: 001001101    OUT_ALG: 0  OUT_DF: 0
@1185   M: 001001110    OUT_ALG: 0  OUT_DF: 0
@1200   M: 001001111    OUT_ALG: 0  OUT_DF: 0
@1215   M: 001010000    OUT_ALG: 0  OUT_DF: 0
@1230   M: 001010001    OUT_ALG: 0  OUT_DF: 0
@2325   M: 010011010    OUT_ALG: 0  OUT_DF: 0
@2340   M: 010011011    OUT_ALG: 0  OUT_DF: 0
@2355   M: 010011100    OUT_ALG: 0  OUT_DF: 0
@2370   M: 010011101    OUT_ALG: 0  OUT_DF: 0
@2385   M: 010011110    OUT_ALG: 0  OUT_DF: 0
@2400   M: 010011111    OUT_ALG: 0  OUT_DF: 0
@2415   M: 010100000    OUT_ALG: 0  OUT_DF: 0
@2430   M: 010100001    OUT_ALG: 0  OUT_DF: 0
@2445   M: 010100010    OUT_ALG: 0  OUT_DF: 0
@2460   M: 010100011    OUT_ALG: 0  OUT_DF: 0
@2475   M: 010100100    OUT_ALG: 0  OUT_DF: 0
@2490   M: 010100101    OUT_ALG: 0  OUT_DF: 0
@2505   M: 010100110    OUT_ALG: 0  OUT_DF: 0
@2655   M: 010110000    OUT_ALG: 0  OUT_DF: 0
@2670   M: 010110001    OUT_ALG: 0  OUT_DF: 0
@2685   M: 010110010    OUT_ALG: 0  OUT_DF: 0
@2700   M: 010110011    OUT_ALG: 0  OUT_DF: 0
@2715   M: 010110100    OUT_ALG: 0  OUT_DF: 0
@2730   M: 010110101    OUT_ALG: 0  OUT_DF: 0
@2745   M: 010110110    OUT_ALG: 0  OUT_DF: 0
@2760   M: 010110111    OUT_ALG: 0  OUT_DF: 0
@2775   M: 010111000    OUT_ALG: 0  OUT_DF: 0
@2790   M: 010111001    OUT_ALG: 0  OUT_DF: 0
@2805   M: 010111010    OUT_ALG: 0  OUT_DF: 0
@2820   M: 010111011    OUT_ALG: 0  OUT_DF: 0
@2835   M: 010111100    OUT_ALG: 0  OUT_DF: 0
@2850   M: 010111101    OUT_ALG: 0  OUT_DF: 0
@2865   M: 010111110    OUT_ALG: 0  OUT_DF: 0
@2880   M: 010111111    OUT_ALG: 1  OUT_DF: 1
@2895   M: 011000000    OUT_ALG: 0  OUT_DF: 0
@2910   M: 011000001    OUT_ALG: 0  OUT_DF: 0
@2925   M: 011000010    OUT_ALG: 0  OUT_DF: 0
@2940   M: 011000011    OUT_ALG: 0  OUT_DF: 0
@2955   M: 011000100    OUT_ALG: 0  OUT_DF: 0
@2970   M: 011000101    OUT_ALG: 0  OUT_DF: 0
@2985   M: 011000110    OUT_ALG: 0  OUT_DF: 0
@3000   M: 011000111    OUT_ALG: 0  OUT_DF: 0
@3015   M: 011001000    OUT_ALG: 0  OUT_DF: 0
@3030   M: 011001001    OUT_ALG: 0  OUT_DF: 0
@3045   M: 011001010    OUT_ALG: 0  OUT_DF: 0
@3060   M: 011001011    OUT_ALG: 0  OUT_DF: 0
@3075   M: 011001100    OUT_ALG: 0  OUT_DF: 0
@3090   M: 011001101    OUT_ALG: 0  OUT_DF: 0
@3105   M: 011001110    OUT_ALG: 0  OUT_DF: 0
@3120   M: 011001111    OUT_ALG: 0  OUT_DF: 0
@3135   M: 011010000    OUT_ALG: 0  OUT_DF: 0
@3150   M: 011010001    OUT_ALG: 0  OUT_DF: 0
@3165   M: 011010010    OUT_ALG: 0  OUT_DF: 0
@3180   M: 011010011    OUT_ALG: 0  OUT_DF: 0
@3195   M: 011010100    OUT_ALG: 0  OUT_DF: 0
@3210   M: 011010101    OUT_ALG: 0  OUT_DF: 0
@3225   M: 011010110    OUT_ALG: 0  OUT_DF: 0
@3240   M: 011010111    OUT_ALG: 0  OUT_DF: 0
@3255   M: 011011000    OUT_ALG: 0  OUT_DF: 0
@3270   M: 011011001    OUT_ALG: 0  OUT_DF: 0
@3285   M: 011011010    OUT_ALG: 0  OUT_DF: 0
@3300   M: 011011011    OUT_ALG: 0  OUT_DF: 0
@3315   M: 011011100    OUT_ALG: 0  OUT_DF: 0
@3330   M: 011011101    OUT_ALG: 0  OUT_DF: 0
@3345   M: 011011110    OUT_ALG: 0  OUT_DF: 0
@3360   M: 011011111    OUT_ALG: 1  OUT_DF: 1
@3375   M: 011100000    OUT_ALG: 0  OUT_DF: 0
@3390   M: 011100001    OUT_ALG: 0  OUT_DF: 0
@3405   M: 011100010    OUT_ALG: 0  OUT_DF: 0
@3420   M: 011100011    OUT_ALG: 0  OUT_DF: 0
@3435   M: 011100100    OUT_ALG: 0  OUT_DF: 0
@3450   M: 011100101    OUT_ALG: 0  OUT_DF: 0
@3465   M: 011100110    OUT_ALG: 0  OUT_DF: 0
@3480   M: 011100111    OUT_ALG: 0  OUT_DF: 0
@3495   M: 011101000    OUT_ALG: 0  OUT_DF: 0
@3510   M: 011101001    OUT_ALG: 0  OUT_DF: 0
@3525   M: 011101010    OUT_ALG: 0  OUT_DF: 0
@3540   M: 011101011    OUT_ALG: 0  OUT_DF: 0
@3555   M: 011101100    OUT_ALG: 0  OUT_DF: 0
@3570   M: 011101101    OUT_ALG: 0  OUT_DF: 0
@3585   M: 011101110    OUT_ALG: 0  OUT_DF: 0
@3600   M: 011101111    OUT_ALG: 1  OUT_DF: 1
@3615   M: 011110000    OUT_ALG: 0  OUT_DF: 0
@3630   M: 011110001    OUT_ALG: 0  OUT_DF: 0
@3645   M: 011110010    OUT_ALG: 0  OUT_DF: 0
@3660   M: 011110011    OUT_ALG: 0  OUT_DF: 0
@3675   M: 011110100    OUT_ALG: 0  OUT_DF: 0
@3690   M: 011110101    OUT_ALG: 0  OUT_DF: 0
@3705   M: 011110110    OUT_ALG: 0  OUT_DF: 0
@3720   M: 011110111    OUT_ALG: 1  OUT_DF: 1
@3735   M: 011111000    OUT_ALG: 0  OUT_DF: 0
@3750   M: 011111001    OUT_ALG: 0  OUT_DF: 0
@3765   M: 011111010    OUT_ALG: 0  OUT_DF: 0
@3780   M: 011111011    OUT_ALG: 1  OUT_DF: 1
@3795   M: 011111100    OUT_ALG: 0  OUT_DF: 0
@3810   M: 011111101    OUT_ALG: 1  OUT_DF: 1
@3825   M: 011111110    OUT_ALG: 1  OUT_DF: 1
@3840   M: 011111111    OUT_ALG: 0  OUT_DF: 0
@3855   M: 100000000    OUT_ALG: 0  OUT_DF: 0
@3870   M: 100000001    OUT_ALG: 0  OUT_DF: 0
@3885   M: 100000010    OUT_ALG: 0  OUT_DF: 0
@5025   M: 101001110    OUT_ALG: 0  OUT_DF: 0
@5040   M: 101001111    OUT_ALG: 0  OUT_DF: 0
@5055   M: 101010000    OUT_ALG: 0  OUT_DF: 0
@5070   M: 101010001    OUT_ALG: 0  OUT_DF: 0
@5085   M: 101010010    OUT_ALG: 0  OUT_DF: 0
@5100   M: 101010011    OUT_ALG: 0  OUT_DF: 0
@5115   M: 101010100    OUT_ALG: 0  OUT_DF: 0
@5130   M: 101010101    OUT_ALG: 0  OUT_DF: 0
@5145   M: 101010110    OUT_ALG: 0  OUT_DF: 0
@5160   M: 101010111    OUT_ALG: 0  OUT_DF: 0
@5175   M: 101011000    OUT_ALG: 0  OUT_DF: 0
@5190   M: 101011001    OUT_ALG: 0  OUT_DF: 0
@5205   M: 101011010    OUT_ALG: 0  OUT_DF: 0
@5220   M: 101011011    OUT_ALG: 0  OUT_DF: 0
@5235   M: 101011100    OUT_ALG: 0  OUT_DF: 0
@5250   M: 101011101    OUT_ALG: 0  OUT_DF: 0
@5265   M: 101011110    OUT_ALG: 0  OUT_DF: 0
@5280   M: 101011111    OUT_ALG: 1  OUT_DF: 1
@5295   M: 101100000    OUT_ALG: 0  OUT_DF: 0
@5310   M: 101100001    OUT_ALG: 0  OUT_DF: 0
@5325   M: 101100010    OUT_ALG: 0  OUT_DF: 0
@5340   M: 101100011    OUT_ALG: 0  OUT_DF: 0
@5355   M: 101100100    OUT_ALG: 0  OUT_DF: 0
@5370   M: 101100101    OUT_ALG: 0  OUT_DF: 0
@5385   M: 101100110    OUT_ALG: 0  OUT_DF: 0
@5400   M: 101100111    OUT_ALG: 0  OUT_DF: 0
@5415   M: 101101000    OUT_ALG: 0  OUT_DF: 0
@5430   M: 101101001    OUT_ALG: 0  OUT_DF: 0
@5445   M: 101101010    OUT_ALG: 0  OUT_DF: 0
@5460   M: 101101011    OUT_ALG: 0  OUT_DF: 0
@5475   M: 101101100    OUT_ALG: 0  OUT_DF: 0
@5490   M: 101101101    OUT_ALG: 0  OUT_DF: 0
@5505   M: 101101110    OUT_ALG: 0  OUT_DF: 0
@5520   M: 101101111    OUT_ALG: 1  OUT_DF: 1
@5535   M: 101110000    OUT_ALG: 0  OUT_DF: 0
@5550   M: 101110001    OUT_ALG: 0  OUT_DF: 0
@5565   M: 101110010    OUT_ALG: 0  OUT_DF: 0
@5580   M: 101110011    OUT_ALG: 0  OUT_DF: 0
@5595   M: 101110100    OUT_ALG: 0  OUT_DF: 0
@5610   M: 101110101    OUT_ALG: 0  OUT_DF: 0
@5625   M: 101110110    OUT_ALG: 0  OUT_DF: 0
@5640   M: 101110111    OUT_ALG: 1  OUT_DF: 1
@5655   M: 101111000    OUT_ALG: 0  OUT_DF: 0
@5670   M: 101111001    OUT_ALG: 0  OUT_DF: 0
@5685   M: 101111010    OUT_ALG: 0  OUT_DF: 0
@5700   M: 101111011    OUT_ALG: 1  OUT_DF: 1
@5715   M: 101111100    OUT_ALG: 0  OUT_DF: 0
@5730   M: 101111101    OUT_ALG: 1  OUT_DF: 1
@5745   M: 101111110    OUT_ALG: 1  OUT_DF: 1
@5760   M: 101111111    OUT_ALG: 0  OUT_DF: 0
@5775   M: 110000000    OUT_ALG: 0  OUT_DF: 0
@5790   M: 110000001    OUT_ALG: 0  OUT_DF: 0
@5805   M: 110000010    OUT_ALG: 0  OUT_DF: 0
@5820   M: 110000011    OUT_ALG: 0  OUT_DF: 0
@5835   M: 110000100    OUT_ALG: 0  OUT_DF: 0
@5850   M: 110000101    OUT_ALG: 0  OUT_DF: 0
@5865   M: 110000110    OUT_ALG: 0  OUT_DF: 0
@5880   M: 110000111    OUT_ALG: 0  OUT_DF: 0
@5895   M: 110001000    OUT_ALG: 0  OUT_DF: 0
@5910   M: 110001001    OUT_ALG: 0  OUT_DF: 0
@5925   M: 110001010    OUT_ALG: 0  OUT_DF: 0
@5940   M: 110001011    OUT_ALG: 0  OUT_DF: 0
@5955   M: 110001100    OUT_ALG: 0  OUT_DF: 0
@5970   M: 110001101    OUT_ALG: 0  OUT_DF: 0
@5985   M: 110001110    OUT_ALG: 0  OUT_DF: 0
@6000   M: 110001111    OUT_ALG: 0  OUT_DF: 0
@6015   M: 110010000    OUT_ALG: 0  OUT_DF: 0
@6030   M: 110010001    OUT_ALG: 0  OUT_DF: 0
@6045   M: 110010010    OUT_ALG: 0  OUT_DF: 0
@6060   M: 110010011    OUT_ALG: 0  OUT_DF: 0
@6075   M: 110010100    OUT_ALG: 0  OUT_DF: 0
@6090   M: 110010101    OUT_ALG: 0  OUT_DF: 0
@6105   M: 110010110    OUT_ALG: 0  OUT_DF: 0
@6120   M: 110010111    OUT_ALG: 0  OUT_DF: 0
@6135   M: 110011000    OUT_ALG: 0  OUT_DF: 0
@6150   M: 110011001    OUT_ALG: 0  OUT_DF: 0
@6165   M: 110011010    OUT_ALG: 0  OUT_DF: 0
@6180   M: 110011011    OUT_ALG: 0  OUT_DF: 0
@6195   M: 110011100    OUT_ALG: 0  OUT_DF: 0
@6210   M: 110011101    OUT_ALG: 0  OUT_DF: 0
@6225   M: 110011110    OUT_ALG: 0  OUT_DF: 0
@6240   M: 110011111    OUT_ALG: 1  OUT_DF: 1
@6255   M: 110100000    OUT_ALG: 0  OUT_DF: 0
@6270   M: 110100001    OUT_ALG: 0  OUT_DF: 0
@6285   M: 110100010    OUT_ALG: 0  OUT_DF: 0
@6300   M: 110100011    OUT_ALG: 0  OUT_DF: 0
@6315   M: 110100100    OUT_ALG: 0  OUT_DF: 0
@6330   M: 110100101    OUT_ALG: 0  OUT_DF: 0
@6345   M: 110100110    OUT_ALG: 0  OUT_DF: 0
@6360   M: 110100111    OUT_ALG: 0  OUT_DF: 0
@6375   M: 110101000    OUT_ALG: 0  OUT_DF: 0
@6390   M: 110101001    OUT_ALG: 0  OUT_DF: 0
@6405   M: 110101010    OUT_ALG: 0  OUT_DF: 0
@6420   M: 110101011    OUT_ALG: 0  OUT_DF: 0
@6435   M: 110101100    OUT_ALG: 0  OUT_DF: 0
@6450   M: 110101101    OUT_ALG: 0  OUT_DF: 0
```

```
@1245   M: 001010010    OUT_ALG: 0  OUT_DF: 0
@1260   M: 001010011    OUT_ALG: 0  OUT_DF: 0
@1275   M: 001010100    OUT_ALG: 0  OUT_DF: 0
@1290   M: 001010101    OUT_ALG: 0  OUT_DF: 0
@1305   M: 001010110    OUT_ALG: 0  OUT_DF: 0
@1320   M: 001010111    OUT_ALG: 0  OUT_DF: 0
@1335   M: 001011000    OUT_ALG: 0  OUT_DF: 0
@1350   M: 001011001    OUT_ALG: 0  OUT_DF: 0
@1365   M: 001011010    OUT_ALG: 0  OUT_DF: 0
@1380   M: 001011011    OUT_ALG: 0  OUT_DF: 0
@1395   M: 001011100    OUT_ALG: 0  OUT_DF: 0
@1410   M: 001011101    OUT_ALG: 0  OUT_DF: 0
@1425   M: 001011110    OUT_ALG: 0  OUT_DF: 0
@1440   M: 001011111    OUT_ALG: 0  OUT_DF: 0
@1455   M: 001100000    OUT_ALG: 0  OUT_DF: 0
@1470   M: 001100001    OUT_ALG: 0  OUT_DF: 0
@1485   M: 001100010    OUT_ALG: 0  OUT_DF: 0
@1500   M: 001100011    OUT_ALG: 0  OUT_DF: 0
@1515   M: 001100100    OUT_ALG: 0  OUT_DF: 0
@1530   M: 001100101    OUT_ALG: 0  OUT_DF: 0
@1545   M: 001100110    OUT_ALG: 0  OUT_DF: 0
@1560   M: 001100111    OUT_ALG: 0  OUT_DF: 0
@1575   M: 001101000    OUT_ALG: 0  OUT_DF: 0
@1590   M: 001101001    OUT_ALG: 0  OUT_DF: 0
@1605   M: 001101010    OUT_ALG: 0  OUT_DF: 0
@1620   M: 001101011    OUT_ALG: 0  OUT_DF: 0
@1635   M: 001101100    OUT_ALG: 0  OUT_DF: 0
@1650   M: 001101101    OUT_ALG: 0  OUT_DF: 0
@1665   M: 001101110    OUT_ALG: 0  OUT_DF: 0
@1680   M: 001101111    OUT_ALG: 0  OUT_DF: 0
@1695   M: 001110000    OUT_ALG: 0  OUT_DF: 0
@1710   M: 001110001    OUT_ALG: 0  OUT_DF: 0
@1725   M: 001110010    OUT_ALG: 0  OUT_DF: 0
@1740   M: 001110011    OUT_ALG: 0  OUT_DF: 0
@1755   M: 001110100    OUT_ALG: 0  OUT_DF: 0
@1770   M: 001110101    OUT_ALG: 0  OUT_DF: 0
@1785   M: 001110110    OUT_ALG: 0  OUT_DF: 0
@1800   M: 001110111    OUT_ALG: 0  OUT_DF: 0
@1815   M: 001111000    OUT_ALG: 0  OUT_DF: 0
@1830   M: 001111001    OUT_ALG: 0  OUT_DF: 0
@1845   M: 001111010    OUT_ALG: 0  OUT_DF: 0
@1860   M: 001111011    OUT_ALG: 0  OUT_DF: 0
@1875   M: 001111100    OUT_ALG: 0  OUT_DF: 0
@1890   M: 001111101    OUT_ALG: 0  OUT_DF: 0
@1905   M: 001111110    OUT_ALG: 0  OUT_DF: 0
@1920   M: 001111111    OUT_ALG: 1  OUT_DF: 1
@1935   M: 010000000    OUT_ALG: 0  OUT_DF: 0
@1950   M: 010000001    OUT_ALG: 0  OUT_DF: 0
@1965   M: 010000010    OUT_ALG: 0  OUT_DF: 0
@1980   M: 010000011    OUT_ALG: 0  OUT_DF: 0
@1995   M: 010000100    OUT_ALG: 0  OUT_DF: 0
@2010   M: 010000101    OUT_ALG: 0  OUT_DF: 0
@2025   M: 010000110    OUT_ALG: 0  OUT_DF: 0
@2040   M: 010000111    OUT_ALG: 0  OUT_DF: 0
@2055   M: 010001000    OUT_ALG: 0  OUT_DF: 0
@2070   M: 010001001    OUT_ALG: 0  OUT_DF: 0
@2085   M: 010001010    OUT_ALG: 0  OUT_DF: 0
@2100   M: 010001011    OUT_ALG: 0  OUT_DF: 0
@2115   M: 010001100    OUT_ALG: 0  OUT_DF: 0
@2130   M: 010001101    OUT_ALG: 0  OUT_DF: 0
@2145   M: 010001110    OUT_ALG: 0  OUT_DF: 0
@2160   M: 010001111    OUT_ALG: 0  OUT_DF: 0
@2175   M: 010010000    OUT_ALG: 0  OUT_DF: 0
@2190   M: 010010001    OUT_ALG: 0  OUT_DF: 0
@2205   M: 010010010    OUT_ALG: 0  OUT_DF: 0
@2220   M: 010010011    OUT_ALG: 0  OUT_DF: 0
@2235   M: 010010100    OUT_ALG: 0  OUT_DF: 0
@2250   M: 010010101    OUT_ALG: 0  OUT_DF: 0
@2265   M: 010010110    OUT_ALG: 0  OUT_DF: 0
@2280   M: 010010111    OUT_ALG: 0  OUT_DF: 0
@2295   M: 010011000    OUT_ALG: 0  OUT_DF: 0
@2310   M: 010011001    OUT_ALG: 0  OUT_DF: 0
```

```
@3900   M: 100000011    OUT_ALG: 0  OUT_DF: 0
@3915   M: 100000100    OUT_ALG: 0  OUT_DF: 0
@3930   M: 100000101    OUT_ALG: 0  OUT_DF: 0
@3945   M: 100000110    OUT_ALG: 0  OUT_DF: 0
@3960   M: 100000111    OUT_ALG: 0  OUT_DF: 0
@3975   M: 100001000    OUT_ALG: 0  OUT_DF: 0
@3990   M: 100001001    OUT_ALG: 0  OUT_DF: 0
@4005   M: 100001010    OUT_ALG: 0  OUT_DF: 0
@4020   M: 100001011    OUT_ALG: 0  OUT_DF: 0
@4035   M: 100001100    OUT_ALG: 0  OUT_DF: 0
@4050   M: 100001101    OUT_ALG: 0  OUT_DF: 0
@4065   M: 100001110    OUT_ALG: 0  OUT_DF: 0
@4080   M: 100001111    OUT_ALG: 0  OUT_DF: 0
@2520   M: 010100111    OUT_ALG: 0  OUT_DF: 0
@2535   M: 010101000    OUT_ALG: 0  OUT_DF: 0
@2550   M: 010101001    OUT_ALG: 0  OUT_DF: 0
@2565   M: 010101010    OUT_ALG: 0  OUT_DF: 0
@2580   M: 010101011    OUT_ALG: 0  OUT_DF: 0
@2595   M: 010101100    OUT_ALG: 0  OUT_DF: 0
@2610   M: 010101101    OUT_ALG: 0  OUT_DF: 0
@2625   M: 010101110    OUT_ALG: 0  OUT_DF: 0
@2640   M: 010101111    OUT_ALG: 0  OUT_DF: 0
@4095   M: 100010000    OUT_ALG: 0  OUT_DF: 0
@4110   M: 100010001    OUT_ALG: 0  OUT_DF: 0
@4125   M: 100010010    OUT_ALG: 0  OUT_DF: 0
@4140   M: 100010011    OUT_ALG: 0  OUT_DF: 0
@4155   M: 100010100    OUT_ALG: 0  OUT_DF: 0
@4170   M: 100010101    OUT_ALG: 0  OUT_DF: 0
@4185   M: 100010110    OUT_ALG: 0  OUT_DF: 0
@4200   M: 100010111    OUT_ALG: 0  OUT_DF: 0
@4215   M: 100011000    OUT_ALG: 0  OUT_DF: 0
@4230   M: 100011001    OUT_ALG: 0  OUT_DF: 0
@4245   M: 100011010    OUT_ALG: 0  OUT_DF: 0
@4260   M: 100011011    OUT_ALG: 0  OUT_DF: 0
@4275   M: 100011100    OUT_ALG: 0  OUT_DF: 0
@4290   M: 100011101    OUT_ALG: 0  OUT_DF: 0
@4305   M: 100011110    OUT_ALG: 0  OUT_DF: 0
@4320   M: 100011111    OUT_ALG: 0  OUT_DF: 0
@4335   M: 100100000    OUT_ALG: 0  OUT_DF: 0
@4350   M: 100100001    OUT_ALG: 0  OUT_DF: 0
@4365   M: 100100010    OUT_ALG: 0  OUT_DF: 0
@4380   M: 100100011    OUT_ALG: 0  OUT_DF: 0
@4395   M: 100100100    OUT_ALG: 0  OUT_DF: 0
@4410   M: 100100101    OUT_ALG: 0  OUT_DF: 0
@4425   M: 100100110    OUT_ALG: 0  OUT_DF: 0
@4440   M: 100100111    OUT_ALG: 0  OUT_DF: 0
@4455   M: 100101000    OUT_ALG: 0  OUT_DF: 0
@4470   M: 100101001    OUT_ALG: 0  OUT_DF: 0
@4485   M: 100101010    OUT_ALG: 0  OUT_DF: 0
@4500   M: 100101011    OUT_ALG: 0  OUT_DF: 0
@4515   M: 100101100    OUT_ALG: 0  OUT_DF: 0
@4530   M: 100101101    OUT_ALG: 0  OUT_DF: 0
@4545   M: 100101110    OUT_ALG: 0  OUT_DF: 0
@4560   M: 100101111    OUT_ALG: 0  OUT_DF: 0
@4575   M: 100110000    OUT_ALG: 0  OUT_DF: 0
@4590   M: 100110001    OUT_ALG: 0  OUT_DF: 0
@4605   M: 100110010    OUT_ALG: 0  OUT_DF: 0
@4620   M: 100110011    OUT_ALG: 0  OUT_DF: 0
@4635   M: 100110100    OUT_ALG: 0  OUT_DF: 0
@4650   M: 100110101    OUT_ALG: 0  OUT_DF: 0
@4665   M: 100110110    OUT_ALG: 0  OUT_DF: 0
@4680   M: 100110111    OUT_ALG: 0  OUT_DF: 0
@4695   M: 100111000    OUT_ALG: 0  OUT_DF: 0
@4710   M: 100111001    OUT_ALG: 0  OUT_DF: 0
@4725   M: 100111010    OUT_ALG: 0  OUT_DF: 0
@4740   M: 100111011    OUT_ALG: 0  OUT_DF: 0
@4755   M: 100111100    OUT_ALG: 0  OUT_DF: 0
@4770   M: 100111101    OUT_ALG: 0  OUT_DF: 0
@4785   M: 100111110    OUT_ALG: 0  OUT_DF: 0
@4800   M: 100111111    OUT_ALG: 1  OUT_DF: 1
@4815   M: 101000000    OUT_ALG: 0  OUT_DF: 0
@4830   M: 101000001    OUT_ALG: 0  OUT_DF: 0
@4845   M: 101000010    OUT_ALG: 0  OUT_DF: 0
@4860   M: 101000011    OUT_ALG: 0  OUT_DF: 0
@4875   M: 101000100    OUT_ALG: 0  OUT_DF: 0
@4890   M: 101000101    OUT_ALG: 0  OUT_DF: 0
@4905   M: 101000110    OUT_ALG: 0  OUT_DF: 0
@4920   M: 101000111    OUT_ALG: 0  OUT_DF: 0
@4935   M: 101001000    OUT_ALG: 0  OUT_DF: 0
@4950   M: 101001001    OUT_ALG: 0  OUT_DF: 0
@4965   M: 101001010    OUT_ALG: 0  OUT_DF: 0
@4980   M: 101001011    OUT_ALG: 0  OUT_DF: 0
@4995   M: 101001100    OUT_ALG: 0  OUT_DF: 0
@5010   M: 101001101    OUT_ALG: 0  OUT_DF: 0
```

```
@6465   M: 110101110    OUT_ALG: 0  OUT_DF: 0
@6480   M: 110101111    OUT_ALG: 1  OUT_DF: 1
@6495   M: 110110000    OUT_ALG: 0  OUT_DF: 0
@6510   M: 110110001    OUT_ALG: 0  OUT_DF: 0
@6525   M: 110110010    OUT_ALG: 0  OUT_DF: 0
@6540   M: 110110011    OUT_ALG: 0  OUT_DF: 0
@6555   M: 110110100    OUT_ALG: 0  OUT_DF: 0
@6570   M: 110110101    OUT_ALG: 0  OUT_DF: 0
@6585   M: 110110110    OUT_ALG: 0  OUT_DF: 0
@6600   M: 110110111    OUT_ALG: 1  OUT_DF: 1
@6615   M: 110111000    OUT_ALG: 0  OUT_DF: 0
@6630   M: 110111001    OUT_ALG: 0  OUT_DF: 0
@6645   M: 110111010    OUT_ALG: 0  OUT_DF: 0
@6660   M: 110111011    OUT_ALG: 1  OUT_DF: 1
@6675   M: 110111100    OUT_ALG: 0  OUT_DF: 0
@6690   M: 110111101    OUT_ALG: 1  OUT_DF: 1
@6705   M: 110111110    OUT_ALG: 1  OUT_DF: 1
@6720   M: 110111111    OUT_ALG: 0  OUT_DF: 0
@6735   M: 111000000    OUT_ALG: 0  OUT_DF: 0
@6750   M: 111000001    OUT_ALG: 0  OUT_DF: 0
@6765   M: 111000010    OUT_ALG: 0  OUT_DF: 0
@6780   M: 111000011    OUT_ALG: 0  OUT_DF: 0
@6795   M: 111000100    OUT_ALG: 0  OUT_DF: 0
@6810   M: 111000101    OUT_ALG: 0  OUT_DF: 0
@6825   M: 111000110    OUT_ALG: 0  OUT_DF: 0
@6840   M: 111000111    OUT_ALG: 0  OUT_DF: 0
@6855   M: 111001000    OUT_ALG: 0  OUT_DF: 0
@6870   M: 111001001    OUT_ALG: 0  OUT_DF: 0
@6885   M: 111001010    OUT_ALG: 0  OUT_DF: 0
@6900   M: 111001011    OUT_ALG: 0  OUT_DF: 0
@6915   M: 111001100    OUT_ALG: 0  OUT_DF: 0
@6930   M: 111001101    OUT_ALG: 0  OUT_DF: 0
@6945   M: 111001110    OUT_ALG: 0  OUT_DF: 0
@6960   M: 111001111    OUT_ALG: 1  OUT_DF: 1
@6975   M: 111010000    OUT_ALG: 0  OUT_DF: 0
@6990   M: 111010001    OUT_ALG: 0  OUT_DF: 0
@7005   M: 111010010    OUT_ALG: 0  OUT_DF: 0
@7020   M: 111010011    OUT_ALG: 0  OUT_DF: 0
@7035   M: 111010100    OUT_ALG: 0  OUT_DF: 0
@7050   M: 111010101    OUT_ALG: 0  OUT_DF: 0
@7065   M: 111010110    OUT_ALG: 0  OUT_DF: 0
@7080   M: 111010111    OUT_ALG: 1  OUT_DF: 1
@7095   M: 111011000    OUT_ALG: 0  OUT_DF: 0
@7110   M: 111011001    OUT_ALG: 0  OUT_DF: 0
@7125   M: 111011010    OUT_ALG: 0  OUT_DF: 0
@7140   M: 111011011    OUT_ALG: 1  OUT_DF: 1
@7155   M: 111011100    OUT_ALG: 0  OUT_DF: 0
@7170   M: 111011101    OUT_ALG: 1  OUT_DF: 1
@7185   M: 111011110    OUT_ALG: 1  OUT_DF: 1
@7200   M: 111011111    OUT_ALG: 0  OUT_DF: 0
@7215   M: 111100000    OUT_ALG: 0  OUT_DF: 0
@7230   M: 111100001    OUT_ALG: 0  OUT_DF: 0
@7245   M: 111100010    OUT_ALG: 0  OUT_DF: 0
@7260   M: 111100011    OUT_ALG: 0  OUT_DF: 0
@7275   M: 111100100    OUT_ALG: 0  OUT_DF: 0
@7290   M: 111100101    OUT_ALG: 0  OUT_DF: 0
@7305   M: 111100110    OUT_ALG: 0  OUT_DF: 0
@7320   M: 111100111    OUT_ALG: 1  OUT_DF: 1
@7335   M: 111101000    OUT_ALG: 0  OUT_DF: 0
@7350   M: 111101001    OUT_ALG: 0  OUT_DF: 0
@7365   M: 111101010    OUT_ALG: 0  OUT_DF: 0
@7380   M: 111101011    OUT_ALG: 1  OUT_DF: 1
@7395   M: 111101100    OUT_ALG: 0  OUT_DF: 0
@7410   M: 111101101    OUT_ALG: 1  OUT_DF: 1
@7425   M: 111101110    OUT_ALG: 1  OUT_DF: 1
@7440   M: 111101111    OUT_ALG: 0  OUT_DF: 0
@7455   M: 111110000    OUT_ALG: 0  OUT_DF: 0
@7470   M: 111110001    OUT_ALG: 0  OUT_DF: 0
@7485   M: 111110010    OUT_ALG: 0  OUT_DF: 0
@7500   M: 111110011    OUT_ALG: 1  OUT_DF: 1
@7515   M: 111110100    OUT_ALG: 0  OUT_DF: 0
@7530   M: 111110101    OUT_ALG: 1  OUT_DF: 1
@7545   M: 111110110    OUT_ALG: 1  OUT_DF: 1
@7560   M: 111110111    OUT_ALG: 0  OUT_DF: 0
@7575   M: 111111000    OUT_ALG: 0  OUT_DF: 0
@7590   M: 111111001    OUT_ALG: 1  OUT_DF: 1
@7605   M: 111111010    OUT_ALG: 1  OUT_DF: 1
@7620   M: 111111011    OUT_ALG: 0  OUT_DF: 0
@7635   M: 111111100    OUT_ALG: 1  OUT_DF: 1
@7650   M: 111111101    OUT_ALG: 0  OUT_DF: 0
@7665   M: 111111110    OUT_ALG: 0  OUT_DF: 0
@7680   M: 111111111    OUT_ALG: 0  OUT_DF: 0
```

## Problem 5

### Binary to Gray Code Converter Dataflow Model

```
module bin2gray #(parameter N = 8) (
```

```
    output logic [N-1:0] gray,
    input [N-1:0] bin
);

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

endmodule
```

**Gray Code to Binary Converter Algorithmic Model**

```
module gray2bin #(parameter N = 8) (
    output logic [N-1:0] bin,
    input [N-1:0] gray
);

    always_comb begin
        bin[N-1] = gray[N-1];
        for (int i=N-2; i >= 0; i--) begin
            bin[i] = bin[i+1] ^ gray[i];
        end
    end

endmodule
```

**Gray Code Testbench**

```
module top ();
    parameter N = 8;

    int failure=0;
    logic [N-1:0] bin_in, gray_out, bin_out;
    logic [N-1:0] mask = {N{1'b1}};
    string s, s1, s2;

    bin2gray #(N) b2g(
        .bin(bin_in),
        .gray(gray_out)
    );

    gray2bin #(N) g2b(
        .gray(gray_out),
        .bin(bin_out)
    );
    initial begin
        $display("\tBINARY\t\tGRAY CODE");
        for (int i=0; i < 2**N; i++) begin
            bin_in = i & mask;
            #10
            $display("%0d:\t%b\t%b", i, bin_in, gray_out);
            assert(bin_out == bin_in) else failure++;
```

```
        end
        if (!failure) $display("SUCCESS: No failures were logged!");
        else $display("FAILURE: %0d/%0d errors were logged.", failure, 2**N);
    end

endmodule
```

### Gray Code Transcript

```
$ vsim -c top
Reading pref.tcl

# 2020.1

# vsim -c top
# Start time: 22:19:38 on Oct 21,2021
# Loading sv_std.std
# Loading work.top
# Loading work.bin2gray
# Loading work.gray2bin
VSIM 1> run -all
#       BINARY          GRAY CODE
# 0:    00000000        00000000
# 1:    00000001        00000001
# 2:    00000010        00000011
# 3:    00000011        00000010
# 4:    00000100        00000110
# 5:    00000101        00000111
# 6:    00000110        00000101
# 7:    00000111        00000100
# 8:    00001000        00001100
# 9:    00001001        00001101
# 10:   00001010        00001111
# 11:   00001011        00001110
# 12:   00001100        00001010
# 13:   00001101        00001011
# 14:   00001110        00001001
# 15:   00001111        00001000
# 16:   00010000        00011000
# 17:   00010001        00011001
# 18:   00010010        00011011
# 19:   00010011        00011010
# 20:   00010100        00011110
# 21:   00010101        00011111
# 22:   00010110        00011101
# 23:   00010111        00011100
# 24:   00011000        00010100
# 25:   00011001        00010101
# 26:   00011010        00010111
# 27:   00011011        00010110
# 28:   00011100        00010010
# 29:   00011101        00010011
# 30:   00011110        00010001
# 31:   00011111        00010000
# 32:   00100000        00110000
# 33:   00100001        00110001
# 34:   00100010        00110011
# 35:   00100011        00110010
# 36:   00100100        00110110
# 37:   00100101        00110111
# 38:   00100110        00110101
# 39:   00100111        00110100
# 40:   00101000        00111100
# 41:   00101001        00111101
# 42:   00101010        00111111
# 43:   00101011        00111110
# 44:   00101100        00111010
# 45:   00101101        00111011
# 46:   00101110        00111001
# 47:   00101111        00111000
# 78:   01001110        01101001
# 79:   01001111        01101000
# 80:   01010000        01111000
# 81:   01010001        01111001
# 82:   01010010        01111011
# 83:   01010011        01111010
# 84:   01010100        01111110
# 85:   01010101        01111111
# 86:   01010110        01111101
# 87:   01010111        01111100
# 88:   01011000        01110100
# 89:   01011001        01110101
# 90:   01011010        01110111
# 91:   01011011        01110110
# 92:   01011100        01110010
# 93:   01011101        01110011
# 94:   01011110        01110001
# 95:   01011111        01110000
# 96:   01100000        01010000
# 97:   01100001        01010001
# 98:   01100010        01010011
# 99:   01100011        01010010
# 100:  01100100        01010110
# 101:  01100101        01010111
# 102:  01100110        01010101
# 103:  01100111        01010100
# 104:  01101000        01011100
# 105:  01101001        01011101
# 106:  01101010        01011111
# 107:  01101011        01011110
# 108:  01101100        01011010
# 109:  01101101        01011011
# 110:  01101110        01011001
# 111:  01101111        01011000
# 112:  01110000        01001000
# 113:  01110001        01001001
# 114:  01110010        01001011
# 115:  01110011        01001010
# 116:  01110100        01001110
# 117:  01110101        01001111
# 118:  01110110        01001101
# 119:  01110111        01001100
# 120:  01111000        01000100
# 121:  01111001        01000101
# 122:  01111010        01000111
# 123:  01111011        01000110
# 124:  01111100        01000010
# 125:  01111101        01000011
# 126:  01111110        01000001
# 127:  01111111        01000000
# 128:  10000000        11000000
# 129:  10000001        11000001
# 130:  10000010        11000011
# 131:  10000011        11000010
# 132:  10000100        11000110
# 133:  10000101        11000111
# 134:  10000110        11000101
# 135:  10000111        11000100
# 136:  10001000        11001100
# 137:  10001001        11001101
# 138:  10001010        11001111
# 169:  10101001        11111101
# 170:  10101010        11111111
# 171:  10101011        11111110
# 172:  10101100        11111010
# 173:  10101101        11111011
# 174:  10101110        11111001
# 175:  10101111        11111000
# 176:  10110000        11101000
# 177:  10110001        11101001
# 178:  10110010        11101011
# 179:  10110011        11101010
# 180:  10110100        11101110
# 181:  10110101        11101111
# 182:  10110110        11101101
# 183:  10110111        11101100
# 184:  10111000        11100100
# 185:  10111001        11100101
# 186:  10111010        11100111
# 187:  10111011        11100110
# 188:  10111100        11100010
# 189:  10111101        11100011
# 190:  10111110        11100001
# 191:  10111111        11100000
# 192:  11000000        10100000
# 193:  11000001        10100001
# 194:  11000010        10100011
# 195:  11000011        10100010
# 196:  11000100        10100110
# 197:  11000101        10100111
# 198:  11000110        10100101
# 199:  11000111        10100100
# 200:  11001000        10101100
# 201:  11001001        10101101
# 202:  11001010        10101111
# 203:  11001011        10101110
# 204:  11001100        10101010
# 205:  11001101        10101011
# 206:  11001110        10101001
# 207:  11001111        10101000
# 208:  11010000        10111000
# 209:  11010001        10111001
# 210:  11010010        10111011
# 211:  11010011        10111010
# 212:  11010100        10111110
# 213:  11010101        10111111
# 214:  11010110        10111101
# 215:  11010111        10111100
# 216:  11011000        10110100
# 217:  11011001        10110101
# 218:  11011010        10110111
# 219:  11011011        10110110
# 220:  11011100        10110010
# 221:  11011101        10110011
# 222:  11011110        10110001
# 223:  11011111        10110000
# 224:  11100000        10010000
# 225:  11100001        10010001
# 226:  11100010        10010011
# 227:  11100011        10010010
# 228:  11100100        10010110
# 229:  11100101        10010111
```

```
# 48:   00110000        00101000
# 49:   00110001        00101001
# 50:   00110010        00101011
# 51:   00110011        00101010
# 52:   00110100        00101110
# 53:   00110101        00101111
# 54:   00110110        00101101
# 55:   00110111        00101100
# 56:   00111000        00100100
# 57:   00111001        00100101
# 58:   00111010        00100111
# 59:   00111011        00100110
# 60:   00111100        00100010
# 61:   00111101        00100011
# 62:   00111110        00100001
# 63:   00111111        00100000
# 64:   01000000        01100000
# 65:   01000001        01100001
# 66:   01000010        01100011
# 67:   01000011        01100010
# 68:   01000100        01100110
# 69:   01000101        01100111
# 70:   01000110        01100101
# 71:   01000111        01100100
# 72:   01001000        01101100
# 73:   01001001        01101101
# 74:   01001010        01101111
# 75:   01001011        01101110
# 76:   01001100        01101010
# 77:   01001101        01101011
```

```
# 139:  10001011        11001110
# 140:  10001100        11001010
# 141:  10001101        11001011
# 142:  10001110        11001001
# 143:  10001111        11001000
# 144:  10010000        11011000
# 145:  10010001        11011001
# 146:  10010010        11011011
# 147:  10010011        11011010
# 148:  10010100        11011110
# 149:  10010101        11011111
# 150:  10010110        11011101
# 151:  10010111        11011100
# 152:  10011000        11010100
# 153:  10011001        11010101
# 154:  10011010        11010111
# 155:  10011011        11010110
# 156:  10011100        11010010
# 157:  10011101        11010011
# 158:  10011110        11010001
# 159:  10011111        11010000
# 160:  10100000        11110000
# 161:  10100001        11110001
# 162:  10100010        11110011
# 163:  10100011        11110010
# 164:  10100100        11110110
# 165:  10100101        11110111
# 166:  10100110        11110101
# 167:  10100111        11110100
# 168:  10101000        11111100
```

```
# 230:  11100110        10010101
# 231:  11100111        10010100
# 232:  11101000        10011100
# 233:  11101001        10011101
# 234:  11101010        10011111
# 235:  11101011        10011110
# 236:  11101100        10011010
# 237:  11101101        10011011
# 238:  11101110        10011001
# 239:  11101111        10011000
# 240:  11110000        10001000
# 241:  11110001        10001001
# 242:  11110010        10001011
# 243:  11110011        10001010
# 244:  11110100        10001110
# 245:  11110101        10001111
# 246:  11110110        10001101
# 247:  11110111        10001100
# 248:  11111000        10000100
# 249:  11111001        10000101
# 250:  11111010        10000111
# 251:  11111011        10000110
# 252:  11111100        10000010
# 253:  11111101        10000011
# 254:  11111110        10000001
# 255:  11111111        10000000
# SUCCESS: No failures were logged!
```

## Problem 6

### Dataflow Comparator

```
////////////////////////////////////////////////////////////////////////////////
// Problem 6a. Write the SV code for a 16-bit comparator in the dataflow model //
////////////////////////////////////////////////////////////////////////////////

module comparator_df #(parameter N = 16) (
    output logic out,
    input [N-1:0] a, b
    );

    assign out = &(~(a^b));

endmodule
```

### Algorithmic Comparator

```
////////////////////////////////////////////////////////////////////////////////
// Problem 6b. Write the SV code for a 16-bit comparator in the algorithmic model //
////////////////////////////////////////////////////////////////////////////////

module comparator_alg #(parameter N = 16) (
    output logic out,
    input [N-1:0] a, b
    );


    always_comb begin
        out = 1'b1;
```

```
        for (int i = 0; i < N; i++) begin
            if (a[i] != b[i]) out = 1'b0;
        end
    end

endmodule
```

**Comparator Testbench**

```
module top();
    parameter N = 16;
    parameter nTESTS = 20;
    logic [N-1:0] a,b, mask={N{1'b1}};
    logic out_df, out_alg;
    int failure = 0;
    string s, s1;

    comparator_df #(N) cd (
        .out(out_df),
        .*
    );

    comparator_alg #(N) ca (
        .out(out_alg),
        .*
    );

    initial begin
        for(int i=0; i < nTESTS; i++) begin
            a = $random() & mask;
            b = $random() & mask;
            // randomly make some of the cases equal
            if ($random() & 1'b1) b = a;
            #10;
            a_df: assert(out_df == (a==b)) else begin
                failure++;
                $error("ERROR - a_df: a: %016b\tb: %016b\tExpected: %0b\tReceived: %0b", a, b, (a==b),
out_df);
            end
            a_alg: assert(out_alg == (a==b)) else begin
                failure++;
                $error("ERROR - a_alg: a: %016b\tb: %016b\tExpected: %0b\tReceived: %0b", a, b, (a==b),
out_alg);
            end
            `ifdef VERBOSE
            $display("%3d:\ta: %016b\tb: %016b\tExpected: %0b\tout_df: %0b\tout_alg: %0b", i, a, b, (a==b),
out_df, out_alg);
            `endif
        end
        if (!failure) $display("SUCCESS: No failures were logged!");
        else $display("FAILURE: %0d errors were logged.", failure);
    end

endmodule
```

**Comparator Transcript**

```
# 2020.1

# vsim -c top
# Start time: 16:11:12 on Oct 23,2021
# Loading sv_std.std
# Loading work.top
# Loading work.comparator_df
# Loading work.comparator_alg
VSIM 1> run -all
#   0:  a: 0011010100100100     b: 0011010100100100     Expected: 1     out_df: 1       out_alg: 1
#   1:  a: 0101011001100011     b: 0101011001100011     Expected: 1     out_df: 1       out_alg: 1
#   2:  a: 1000010001100101     b: 1000010001100101     Expected: 1     out_df: 1       out_alg: 1
#   3:  a: 1100110100001101     b: 1100110100001101     Expected: 1     out_df: 1       out_alg: 1
#   4:  a: 0101011111101101     b: 0101011111101101     Expected: 1     out_df: 1       out_alg: 1
#   5:  a: 0010010011000110     b: 1000010011000101     Expected: 0     out_df: 0       out_alg: 0
#   6:  a: 1111011111100101     b: 0111001001110111     Expected: 0     out_df: 0       out_alg: 0
#   7:  a: 1101101110001111     b: 0110100111110010     Expected: 0     out_df: 0       out_alg: 0
#   8:  a: 0111101011101000     b: 0100111011000101     Expected: 0     out_df: 0       out_alg: 0
#   9:  a: 0010100010111101     b: 0010100010111101     Expected: 1     out_df: 1       out_alg: 1
#  10:  a: 0110001001100011     b: 1000011100001010     Expected: 0     out_df: 0       out_alg: 0
#  11:  a: 0010000100100000     b: 0010000100100000     Expected: 1     out_df: 1       out_alg: 1
#  12:  a: 0011111010010110     b: 0011111010010110     Expected: 1     out_df: 1       out_alg: 1
#  13:  a: 1101011001010011     b: 1101011001010011     Expected: 1     out_df: 1       out_alg: 1
#  14:  a: 0100101000000010     b: 0100101000000010     Expected: 1     out_df: 1       out_alg: 1
#  15:  a: 0111001011001111     b: 0100100100100011     Expected: 0     out_df: 0       out_alg: 0
#  16:  a: 0000101011001010     b: 0100110000111100     Expected: 0     out_df: 0       out_alg: 0
#  17:  a: 0110000110001010     b: 1011001101000001     Expected: 0     out_df: 0       out_alg: 0
#  18:  a: 1111001101111000     b: 1111001101111000     Expected: 1     out_df: 1       out_alg: 1
#  19:  a: 0110010110110110     b: 1111100111000110     Expected: 0     out_df: 0       out_alg: 0
# SUCCESS: No failures were logged!
```

## Problem 7

**Hamming Encoder**

```
module hamming_enc #(
    parameter r = 4,            // Defines hamming(n,k) code
                                // r = 3: hamming(7,4); r = 4: hamming(15,11), etc.
    localparam n = 2**r-1,      // Block length including payload and parity bits
    localparam k = n-r          // Payload length
    ) (
    output logic [n:0] out,
    input  [k-1:0] in
);

    logic [k-1:0] in_tmp;
    logic [n:0] bit_vec = '0;   // Bit vector
    logic [r-1:0] ohv [r];      // One-hot Vector
    int ohv_idx = 0;
    logic parity, b;

    always_comb begin
        in_tmp = in;
        bit_vec = '0;
        ohv_idx = 0;
```

```
        // Create array of all possible onehot vectors in an R-bit value
        for (int i = 0; i < r; i++) ohv[i] = 1'b1 << i;
        // Populate the output vector
        for (int i = 1; i <= n; i++) begin
            if (i != ohv[ohv_idx]) begin
                // if the index isn't one of the parity spot
                // fill it with bit from payload
                b = in_tmp[0];
                in_tmp = in_tmp >> 1;
                bit_vec[i] = b;
            end else begin
                // if it is a parity spot, skip it for now
                if (ohv_idx < (r-1)) ohv_idx++;
            end
        end

        // for each onehot vector find all indices which share a bit with this vector
        // and xor the values at these indices to calculate the parity of this collection
        // of bits from the payload
        foreach(ohv[i]) begin
            parity = 1'b0;
            for (int j = 1; j <= n; j++) begin
                if (j & ohv[i]) begin
                    parity = bit_vec[j] ^ parity;
                end
            end
            // store the parity bit in the correct parity location
            bit_vec[ohv[i]] = parity;
        end

        // Calculate overall parity bit and store in 0th location
        bit_vec[0] = ^bit_vec[n:1];
        out = bit_vec;
    end
endmodule
```

**Hamming Decoder**

```
module hamming_dec #(
    parameter r = 3,            // Integer >= 2 that defines hamming code architecture
    localparam n = 2**r-1,      // Block length including payload and parity bits
    localparam k = n-r          // Payload length
    ) (
    output logic [k-1:0] out,
    input  [n:0] in
);

    logic [n:0] in_tmp;
    logic [k-1:0] out_tmp;
    logic [r-1:0] v = '0;
    logic [r-1:0] ohv [r];
    logic [r-1:0] ohv_idx, out_idx;

    always_comb begin
        in_tmp = in;
```

```
        out_tmp = '0;
        v = '0;

        // collect index of all bits whose value is 1 and xor them together
        for (int i = 0; i <= n; i++) if (in_tmp[i]) v = v ^ i;
        // if v is non-zero (error exists), flip in[v]. (error correction)
        if (v) in_tmp[v] = !in_tmp[v];
        // create onehot array of hamming index elements. ohv = (1, 2, 4, 8, etc.)
        for (int i = 0; i < r; i++) ohv[i] = 4'b0001 << i;
        // construct new vector removing the parity bits
        ohv_idx = 0;
        out_idx = 0;
        for (int i = 1; i <=n; i++) begin
            if (i == ohv[ohv_idx]) begin
                if (ohv_idx < r-1) ohv_idx++;
                continue;
            end else begin
                out_tmp[out_idx] = in_tmp[i];
                out_idx++;
            end
        end
        out = out_tmp;
    end
endmodule
```

**Hamming Testbench**

```
module top ();
    parameter R = 4;
    parameter NTESTS = 20;
    localparam N = 2**R-1;
    localparam K = N - R;

    logic [K-1:0] in_e, out_d;
    logic [N:0] out_e, chk_out, out_e_orig;
    logic [K-1:0] in_mask = {K{1'b1}};
    int rand_idx, failure=0;
    string sN, sK, s1, s2;

    hamming_enc #(R) he (
        .in(in_e),
        .out(out_e)
    );

    hamming_dec #(R) hd (
        .in(out_e),
        .out(out_d)
    );

    hamming_enc #(R) he_chk (
        .in(out_d),
        .out(chk_out)
    );


    initial begin
```

```
        sN = $sformatf("%%0%0db", N+1);
        sK = $sformatf("%%0%0db", K);
        s1 = $sformatf("ENC: Payload\t: %s\t\tEncoded\t: %s", sK, sN);
        s2 = $sformatf("DEC: Encoded\t: %s\tPayload\t: %s", sN, sK);
        for (int i=0; i < NTESTS; i++) begin
            // generate input
            in_e = $random() & in_mask;
            #20;
            `ifdef VERBOSE
                $display(s1, in_e, out_e);
                $display(s2, out_e, out_d);
            `endif
            assert(out_d == in_e) else begin
                $error("Encoding/Decoding Error");
                failure++;
            end
            assert(out_e == chk_out) else begin
                $error("Decoding/Encoding Error");
                failure++;
            end

            // error injection
            out_e_orig = out_e;
            rand_idx = $urandom_range(K-1);
            out_e[rand_idx] = !out_e[rand_idx];
            #20;
            `ifdef VERBOSE
                $display("--Error Injection into Encoded Message--");
                $display(s2, out_e, out_d);
                $display(s1, out_d, chk_out);
                $display("\n");
            `endif
            assert(out_d == in_e) else begin
                $error("Payload Error Correction Failure");
                failure++;
            end
            assert(out_e_orig == chk_out) else begin
                $error("Encoded Message Correction Mismatch");
                failure++;
            end
        end

        if (!failure) $display("SUCCESS: No failures were logged!");
        else $display("FAILURE: %0d errors were logged.", failure);
    end
endmodule
```

**Hamming Transcript**

```
# vsim -c top
# Start time: 23:24:32 on Oct 21,2021
# Loading sv_std.std
# Loading work.top
# Loading work.hamming_enc
# Loading work.hamming_dec
VSIM 1> run -all
```

```
# ENC: Payload  : 10100100100           Encoded : 1010010101010101
# DEC: Encoded  : 1010010101010101      Payload : 10100100100
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1010010111010101      Payload : 10100100100
# ENC: Payload  : 10100100100           Encoded : 1010010101010101
#
#
# ENC: Payload  : 11010000001           Encoded : 1101000100011101
# DEC: Encoded  : 1101000100011101      Payload : 11010000001
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1101010100011101      Payload : 11010000001
# ENC: Payload  : 11010000001           Encoded : 1101000100011101
#
#
# ENC: Payload  : 11000001001           Encoded : 1100000010011010
# DEC: Encoded  : 1100000010011010      Payload : 11000001001
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1100000110011010      Payload : 11000001001
# ENC: Payload  : 11000001001           Encoded : 1100000010011010
#
#
# ENC: Payload  : 11001100011           Encoded : 1100110000111100
# DEC: Encoded  : 1100110000111100      Payload : 11001100011
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1100110000110100      Payload : 11001100011
# ENC: Payload  : 11001100011           Encoded : 1100110000111100
#
#
# ENC: Payload  : 01100001101           Encoded : 0110000011001010
# DEC: Encoded  : 0110000011001010      Payload : 01100001101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0110000011101010      Payload : 01100001101
# ENC: Payload  : 01100001101           Encoded : 0110000011001010
#
#
# ENC: Payload  : 00110001101           Encoded : 0011000011001111
# DEC: Encoded  : 0011000011001111      Payload : 00110001101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0011001011001111      Payload : 00110001101
# ENC: Payload  : 00110001101           Encoded : 0011000011001111
#
#
# ENC: Payload  : 10001100101           Encoded : 1000110101001110
# DEC: Encoded  : 1000110101001110      Payload : 10001100101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1000110101000110      Payload : 10001100101
# ENC: Payload  : 10001100101           Encoded : 1000110101001110
#
#
# ENC: Payload  : 01000010010           Encoded : 0100001000100100
# DEC: Encoded  : 0100001000100100      Payload : 01000010010
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0100011000100100      Payload : 01000010010
# ENC: Payload  : 01000010010           Encoded : 0100001000100100
#
#
# ENC: Payload  : 01100000001           Encoded : 0110000000001001
```

```
# DEC: Encoded  : 0110000000001001      Payload : 01100000001
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0110000001001001      Payload : 01100000001
# ENC: Payload  : 01100000001           Encoded : 0110000000001001
#
#
# ENC: Payload  : 10100001101           Encoded : 1010000011001001
# DEC: Encoded  : 1010000011001001      Payload : 10100001101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1010000011000001      Payload : 10100001101
# ENC: Payload  : 10100001101           Encoded : 1010000011001001
#
#
# ENC: Payload  : 00101110110           Encoded : 0010111001110100
# DEC: Encoded  : 0010111001110100      Payload : 00101110110
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0010110001110100      Payload : 00101110110
# ENC: Payload  : 00101110110           Encoded : 0010111001110100
#
#
# ENC: Payload  : 10100111101           Encoded : 1010011011001111
# DEC: Encoded  : 1010011011001111      Payload : 10100111101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1010011011000111      Payload : 10100111101
# ENC: Payload  : 10100111101           Encoded : 1010011011001111
#
#
# ENC: Payload  : 11111101101           Encoded : 1111110011001111
# DEC: Encoded  : 1111110011001111      Payload : 11111101101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1111110011011111      Payload : 11111101101
# ENC: Payload  : 11111101101           Encoded : 1111110011001111
#
#
# ENC: Payload  : 11110001100           Encoded : 1111000011000011
# DEC: Encoded  : 1111000011000011      Payload : 11110001100
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1111000011000111      Payload : 11110001100
# ENC: Payload  : 11110001100           Encoded : 1111000011000011
#
#
# ENC: Payload  : 00111111001           Encoded : 0011111110011010
# DEC: Encoded  : 0011111110011010      Payload : 00111111001
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0011101110011010      Payload : 00111111001
# ENC: Payload  : 00111111001           Encoded : 0011111110011010
#
#
# ENC: Payload  : 10011000110           Encoded : 1001100101100110
# DEC: Encoded  : 1001100101100110      Payload : 10011000110
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1001110101100110      Payload : 10011000110
# ENC: Payload  : 10011000110           Encoded : 1001100101100110
#
#
# ENC: Payload  : 10011000101           Encoded : 1001100101011010
# DEC: Encoded  : 1001100101011010      Payload : 10011000101
```

```
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1001100101111010      Payload : 10011000101
# ENC: Payload  : 10011000101           Encoded : 1001100101011010
#
#
# ENC: Payload  : 01010101010           Encoded : 0101010110100101
# DEC: Encoded  : 0101010110100101      Payload : 01010101010
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0101010110100001      Payload : 01010101010
# ENC: Payload  : 01010101010           Encoded : 0101010110100101
#
#
# ENC: Payload  : 11111100101           Encoded : 1111110001011001
# DEC: Encoded  : 1111110001011001      Payload : 11111100101
# --Error Injection into Encoded Message--
# DEC: Encoded  : 1111110001111001      Payload : 11111100101
# ENC: Payload  : 11111100101           Encoded : 1111110001011001
#
#
# ENC: Payload  : 01001110111           Encoded : 0100111001111101
# DEC: Encoded  : 0100111001111101      Payload : 01001110111
# --Error Injection into Encoded Message--
# DEC: Encoded  : 0100111001110101      Payload : 01001110111
# ENC: Payload  : 01001110111           Encoded : 0100111001111101
#
#
# SUCCESS: No failures were logged!
```

## Problem 8

**Circuit Code**

```
// Problem 8

// Chuck Faber

// Write an SV model for a combinational circuit with three inputs and one output.
// The output is 1 when the binary value of the inputs is less than 3. The outputs is 0 otherwise

// if !(|bits[N-1:2]) && !(bits[1]&&bits[0]) -> out = 1, else out = 0

module circuit (
    input [2:0] in,
    output logic out
    );

    always_comb begin
        out = in[2]|(in[1]&in[0]) ? 1'b0 : 1'b1;
    end

endmodule
```

**Testbench Code**

```
module top();
    logic [2:0] in;
```

```
    logic out;

    circuit c(.*);

    initial begin
        for (int i = 0; i < 8; i++) begin
            in = i & 3'b111;
            #10;
            if (i < 3) assert(out == 1'b1);
            if (i >= 3) assert(out == 1'b0);
            `ifdef VERBOSE
            $display("in: %03b\tout: %01b", in, out);
            `endif
        end
    end

endmodule
```

**Circuit Transcript**

```
$ vsim -c top
Reading pref.tcl

# 2020.1

# vsim -c top
# Start time: 22:04:38 on Oct 23,2021
# Loading sv_std.std
# Loading work.top
# Loading work.circuit
VSIM 1> run -all
# in: 000       out: 1
# in: 001       out: 1
# in: 010       out: 1
# in: 011       out: 0
# in: 100       out: 0
# in: 101       out: 0
# in: 110       out: 0
# in: 111       out: 0
```