ELECTRICAL & COMPUTER
ENGINEERING

Portland State
UNIVERSITY

ECE 4/581 - Modeling and Synthesis

ASIC Modeling and Synthesis

Fall 2020

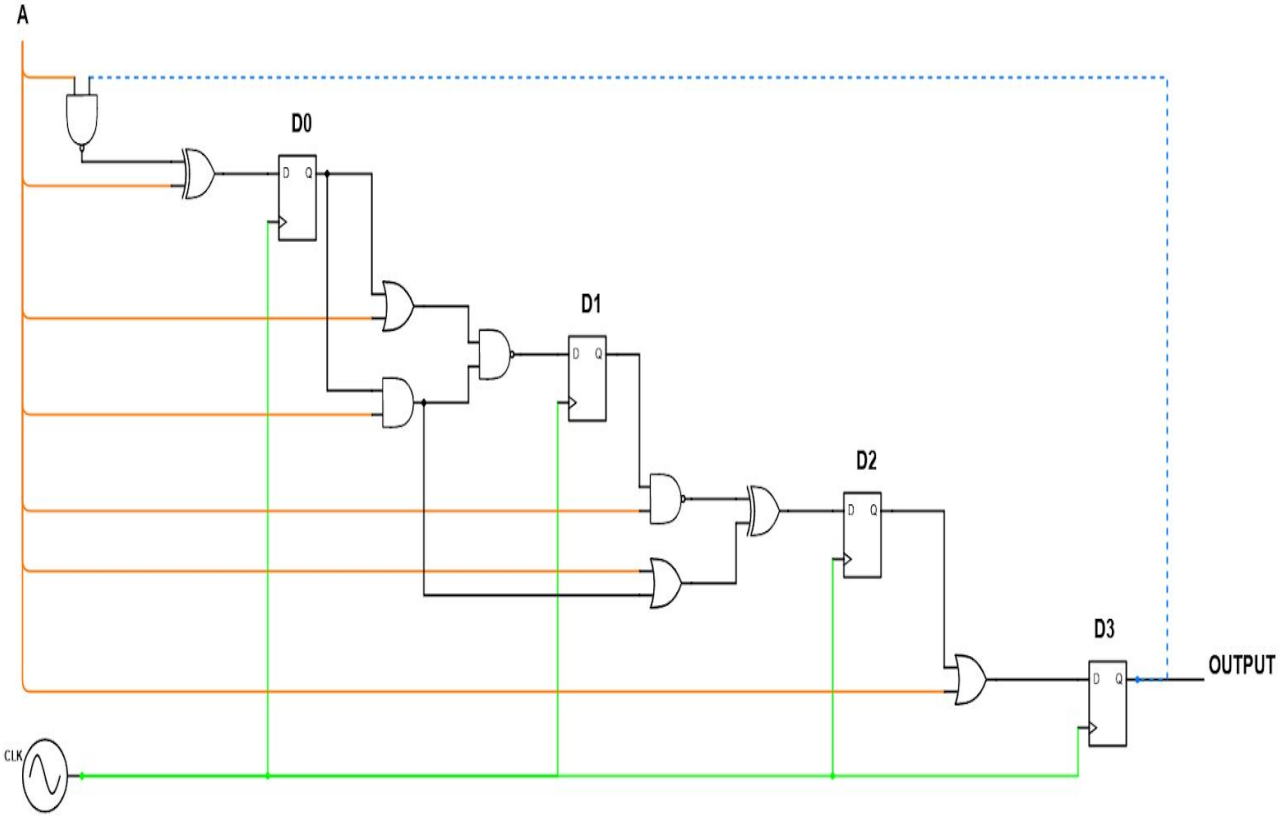
PROJECT 1 REPORT

Project Title	Project 1
Prepared For	Professor Xiaoyu Song
Prepared By	Nguyen Pham Ngan Ho
Revision Number	1
Date of Revision	10/14/2020
Date of Submission	10/18/2020

Table of contents

*** Problem 1:

a. Design a sequential circuit S1 with the combinational logic part for the next state function should contain 3 NAND gates, 2 XOR gates, 3 OR gates, 1 AND gate, and 4 flip-flops.



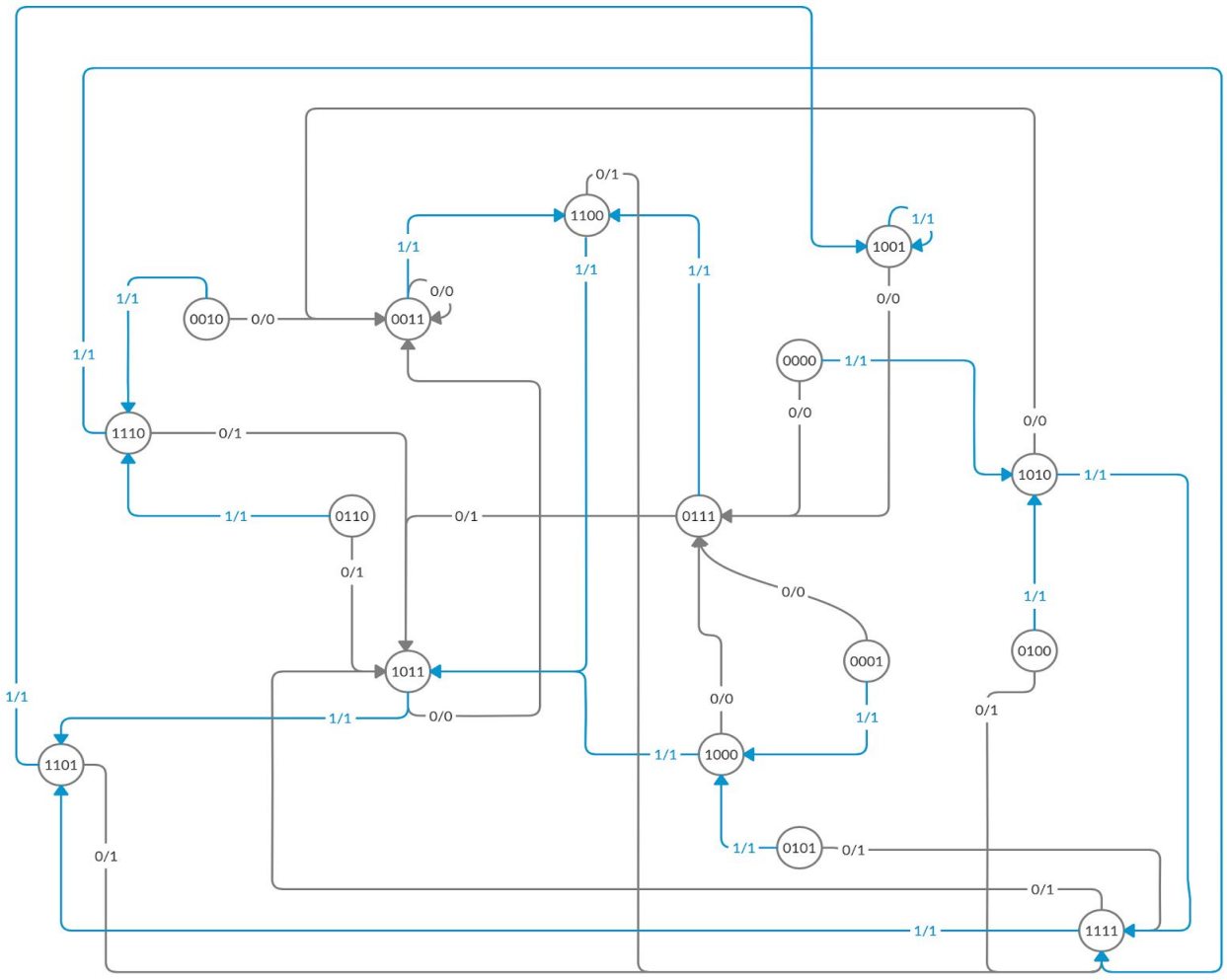
Sequential circuit S1

b. The next state function table for S1

Input	Current States				Next States				
A	**Q3**	**Q2**	**Q1**	**Q0**	**Q'3**	**Q'2**	**Q'1**	**Q'0**	**Output**
0	0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	1	0	0	0	1	1	0
0	0	0	1	1	0	0	1	1	0
0	0	1	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1
0	0	1	1	0	1	0	1	1	1
0	0	1	1	1	1	0	1	1	1
0	1	0	0	0	0	1	1	1	0
0	1	0	0	1	0	1	1	1	0
0	1	0	1	0	0	0	1	1	0
0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1
0	1	1	1	0	1	0	1	1	1
0	1	1	1	1	1	0	1	1	1
1	0	0	0	0	1	0	1	0	1
1	0	0	0	1	1	0	0	0	1
1	0	0	1	0	1	1	1	0	1
1	0	0	1	1	1	1	0	0	1
1	0	1	0	0	1	0	1	0	1
1	0	1	0	1	1	0	0	0	1
1	0	1	1	0	1	1	1	0	1
1	0	1	1	1	1	1	0	0	1
1	1	0	0	0	1	0	1	1	1

1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	0	1	1	1	1	0	1	1
1	1	1	0	0	1	0	1	1	1
1	1	1	0	1	1	0	0	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1

c. The state transition graph of S1 including all the POSSIBLE states.



*** Problem 2:

a. Write the SV code for a 2n to n priority encoder in the dataflow model (continuous assignments)

Designed module:

```
module problem2a
#(      parameter       N = 2)                  // 2-bit
(       input logic     [(2**N)-1:0] sigIn,     // 4-bit input signal
        output logic    [(N-1):0] sigOut        // 2-bit output signal
);

        // Initiate intances
        timeunit 1ns/1ns;
        logic       [(2**N):0][(N-1):0] temp;   // temporary variable

        // if all inputs are 0 -> output is 'x'
        assign temp[0] = {N{1'bx}};
         // Iteration
            for (genvar i = (2**N) - 1; i >= 0 ; i--) begin: loop
            // If high bit is x or z -> ouput is x,
            // and if high bit is 1-> output is an equivalent binary code
                assign temp[i+1] = (sigIn[i]===1'bx || sigIn[i]===1'bz) ?
            {N{1'bx}}: (sigIn[i])? i : temp[i];
            end: loop
        assign sigOut = temp[2**N];
endmodule: problem2a
```

Testbench:

```
module tb_problem2a (sigIn, sigOut);
    timeunit 1ns/1ns;
    parameter       NUM_BITS = 3;               //4-bits
    parameter       CASE = 10;                  // 10 Random cases
    input logic     [(2**NUM_BITS)-1:0] sigIn;
    output logic    [(NUM_BITS-1):0] sigOut;
        // Initiate instances
    problem2a #(NUM_BITS) p_2a(sigIn, sigOut);
    logic [(2**NUM_BITS)-1:0]temp = 4'bxz01;            //temp array
    initial begin
        $monitor($time, "\t input: %b \t output: %b", sigIn, sigOut);
        // Test common combinations
        $display("Test for problem 2b:\n\n");
        //time 0, input is 0
```

```
            sigIn = {NUM_BITS{1'b0}};
            #5;
            // next time unit
            for (int k = 0; k < 2**NUM_BITS; k++) begin
              sigIn = {NUM_BITS{1'b0}};
              sigIn[k] = 1'b1;
              #5;
            end
            // Test random cases
             $display("Test random cases:");
            for (int i = 0; i < CASE; i++) begin: i_loop
              for (int j = 0; j < 2**NUM_BITS; j++) begin: j_loop
                 sigIn[j] = temp[$urandom() % 4];
              end: j_loop
              #5;
            end: i_loop
            #5 $stop;
          end
        endmodule: tb_problem2a
```

Test result:

```
# Compile of problem2a.sv was successful.
# Compile of tb_problem2a.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -voptargs=+acc work.tb_problem2a
# vsim -voptargs="+acc" work.tb_problem2a
# Start time: 22:56:42 on Oct 14,2020
# ** Note: (vsim-3813) Design is being optimized due to module recompilation…
# Loading sv_std.std
# Loading work.tb_problem2a(fast)
# Loading work.problem2a(fast)
run -all
# Test for problem 2b:
#
#
#               0   input: 00000000        output: xxx
#               5   input: 00000001        output: 000
#              10   input: 00000010        output: 001
#              15   input: 00000100        output: 010
#              20   input: 00001000        output: 011
#              25   input: 00010000        output: 100
#              30   input: 00100000        output: 101
#              35   input: 01000000        output: 110
#              40   input: 10000000        output: 111
```

```
# Test random cases:
#              45   input: 11zx1111        output: 111
#              50   input: z01z0x10        output: xxx
#              55   input: 01zxzx1z        output: 110
#              60   input: 11zxxzx1        output: 111
#              65   input: 00zxz0z1        output: xxx
#              70   input: x101z101        output: xxx
#              75   input: 0x100x10        output: xxx
#              80   input: 000zz100        output: xxx
#              85   input: 0x1z0xzx        output: xxx
#              90   input: z100101z        output: xxx
# ** Note: $stop    : N:/Desktop/ECE 581/P1/tb_problem2a.sv(44)
#    Time: 100 ns  Iteration: 0  Instance: /tb_problem2a
# Break in Module tb_problem2a at N:/Desktop/ECE 581/P1/tb_problem2a.sv line 44
```

b. Write the SV code for a 2n to n priority encoder in the algorithmic model (always_comb).

Designed module:

```
module problem2b
    #(parameter N = 2)
    (
    input               [2**N-1 : 0] sigIn,
    output reg          [N-1 : 0] sigOut
    );
    timeunit 1ns/1ns;
    always_comb begin: combination
    // default output value when all input bits are 0
    sigOut = {N{1'bx}};
    // traverse from highest-priority bit to the lowest one
    for (int i = (2**N) - 1; i > -1; i--) begin: loop
    // x or z signal -> output is don't care
        if(sigIn[i] === 1'bx || sigIn[i] === 1'bz) begin
            sigOut = {N{1'bx}};
            break;
            end
    // priority bit is 1, output is number of bit
    if(sigIn[i]) begin
        sigOut = i;
        break;
        end
      end: loop
    end: combination
endmodule: problem2b
```

Testbench:

```
module tb_problem2b (sigIn, sigOut);
        timeunit 1ns/1ns;
        parameter           NUM_BITS = 3;
        parameter           CASE = 10;                          // 10 Random cases
        input logic         [(2**NUM_BITS)-1 : 0] sigIn;
        output logic   [(NUM_BITS-1) : 0] sigOut;

            // Initiate instances
        problem2b #(NUM_BITS) p_2b(sigIn, sigOut);
        logic [(2**NUM_BITS)-1:0]temp = 4'bxz01;                //temp array

        initial begin
                $monitor($time, "\t input: %b \t output: %b", sigIn, sigOut);
                // Test 01: test common combinations
                $display("Test for problem 2b:\n");
                //time 0, input is 0
                sigIn = {NUM_BITS{1'b0}};
                #5;
                // next time unit
                        for (int k = 0; k < 2**NUM_BITS; k++) begin
                                sigIn = {NUM_BITS{1'b0}};
                                sigIn[k] = 1'b1;
                                #5;
                            end

                // Test random cases
                        $display("Test random cases:");
                                for (int i = 0; i < CASE; i++) begin: i_loop
                                        for (int j = 0; j < 2**NUM_BITS; j++) begin: j_loop
                                                sigIn[j] = temp[$urandom() % 4];
                        end: j_loop
                    #5;
                end: i_loop
                #5 $stop;
        end
endmodule: tb_problem2b
```

Test result:

```
# Compile of problem2b.sv was successful.
# Compile of tb_problem2b.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -voptargs=+acc work.tb_problem2b
# vsim -voptargs="+acc" work.tb_problem2b
# Start time: 22:59:16 on Oct 14,2020
# ** Note: (vsim-3812) Design is being optimized…
# Loading sv_std.std
# Loading work.tb_problem2b(fast)
# Loading work.problem2b(fast)
run -all
# Test for problem 2b:
#
#               0       input: 00000000         output: xxx
#               5       input: 00000001         output: 000
#              10       input: 00000010         output: 001
#              15       input: 00000100         output: 010
#              20       input: 00001000         output: 011
#              25       input: 00010000         output: 100
#              30       input: 00100000         output: 101
#              35       input: 01000000         output: 110
#              40       input: 10000000         output: 111

# Test random cases:
#              45       input: 11zx1111         output: 111
#              50       input: z01z0x10         output: xxx
#              55       input: 01zxzx1z         output: 110
#              60       input: 11zxxzx1         output: 111
#              65       input: 00zxz0z1         output: xxx
#              70       input: x101z101         output: xxx
#              75       input: 0x100x10         output: xxx
#              80       input: 000zz100         output: xxx
#              85       input: 0x1z0xzx         output: xxx
#              90       input: z100101z         output: xxx
# ** Note: $stop    : N:/Desktop/ECE 581/P1/tb_problem2b.sv(46)
#    Time: 100 ns  Iteration: 0  Instance: /tb_problem2b
# Break in Module tb_problem2b at N:/Desktop/ECE 581/P1/tb_problem2b.sv line 46
```

*** Problem 3:

a. Write the SV code for a 16-bit carry look ahead adder in the dataflow model (continuous assignments)

Designed module:

```
module CarryLookAheadAdderContinuousAssigment
#(
parameter Nbits = 16)
(
        input logic[Nbits-1:0] a_in,          // first 16 bits input
        input logic[Nbits-1:0] b_in,          // second 16 bits input
        input bit              Cin,           // 1 bit carry in input
        output logic [Nbits-1:0] sum,         // 16 bits sum
        output bit             Cout           // 1 bit Cout
);

        logic [Nbits:0] C;                    // internal  16 bits carry value
        logic [Nbits-1:0] S;                  // internal 16 bits sum
        assign C[0] = Cin;                    // Set the bit 0 equals Carry in
        genvar i;
        generate
                for (i = 0; i < Nbits; i++)   // Using generate and for loop go through all bits of
2 inputs
                begin
                        assign S[i] = a_in[i] ^ b_in[i] ^ C[i];          // Set S[i] = a_in[i] XOR
b_in[i] XOR C[i]
                        assign C[i+1] = (a_in[i]& b_in[i]) | ((a_in[i] ^ b_in[i])& C[i]);   // Set
C[i+1] = (a_in[i] AND b_in[i]) OR (a_in[i] XOR b_in[i] and C[i])
                end
        endgenerate
        assign sum = S;               // Set sum = S
        assign Cout = C[Nbits];       // Set Cout equals MSB of C
endmodule
```

Testbench:

```
module TestbenchCarryLookAheadContinuous;
        parameter           Nbits = 16;       // 16 bits
        bit [Nbits-1:0]     a_in;             // first 16 bits input
        bit [Nbits-1:0]     b_in;             // second 16 bits input
        bit                 Cin;              // 1 bit carry in input
        bit [Nbits-1:0]     sum;              // 16 bits sum
        bit                 Cout;             // 1 bit Cout
        wire [Nbits-1:0]    C;                // internal 16 bits carry value
        wire [Nbits-1:0]    S;                // internal 16 bits sum
        integer             Error;            // Error variable
```

```
        reg [Nbits:0]       result;             //16 bits results use for comparing, checking the correctness of module
        CarryLookAheadAdderContinuousAssigment # (Nbits) CLAACA (.*); // Instantiate CarryLookAheadAdderContinuousAssigment module

        initial
        begin
            Cin = 0;                            // Set initial value for Cin
            Error = 0;                          // Set initial value for Error = 0
            repeat (2)
            begin
                for (int i = 0; i < 2**Nbits; i++)// Go through all values of a_in
                begin
                    a_in = i;                   // Set a_in = i
                    for (int j = 0; j < 2**Nbits; j++) // Go through all values of b_in
                    begin
                        b_in = j;               // Set b_in = j
                        result = a_in + b_in + Cin;  // Set result = a_in + b_in + Cin

                        #50;
                        if ({Cout,sum} != result)   // Check the yield value with expected result
                        begin
                            Error = Error +1;   // Increase value of error when the result is not correct
                            $display ("Error founded, adding %b + %b + %b, we expeced Cout is %b, sum is %b, however we got Cout is %b, sum is %b", a_in, b_in,Cin, result[Nbits], result[Nbits-1:0], Cout, sum);
                        end
                        else
                        begin
                        $display("Adding %b + %b + %b, Sum = %b, Cout = %b", a_in, b_in, Cin, sum, Cout);
                        end
                    end
                end
                Cin = !Cin;             // Getting inverse value of Cin for second run
            end
            if (Error == 0)
            $display("Congratulation, your module correctly");
        end
    endmodule
```

Test results:

Please note that we used exhaustive tests, so this is only the snapshot of the result. Also, since we use exhaustive tests, we only test 8 bits instead of 16 bits for the purpose of saving time. For the complete test results of 8 bits, please look at the *TranscriptCarryLookAheadContinuousAssignment* file.

```
vsim -gui work.TestbenchCarryLookAheadContinuous
# vsim -gui work.TestbenchCarryLookAheadContinuous
# Start time: 16:44:45 on Oct 11,2020
# ** Note: (vsim-8009) Loading existing optimized design _opt2
# Loading sv_std.std
# Loading work.TestbenchCarryLookAheadContinuous(fast)
run -all
# Adding 00000000 + 00000000 + 0, Sum = 00000000, Cout = 0
# Adding 00000000 + 00000001 + 0, Sum = 00000001, Cout = 0
# Adding 00000000 + 00000010 + 0, Sum = 00000010, Cout = 0
# Adding 00000000 + 00000011 + 0, Sum = 00000011, Cout = 0
# Adding 00000000 + 00000100 + 0, Sum = 00000100, Cout = 0
# Adding 00000000 + 00000101 + 0, Sum = 00000101, Cout = 0
# Adding 00000000 + 00000110 + 0, Sum = 00000110, Cout = 0
# Adding 00000000 + 00000111 + 0, Sum = 00000111, Cout = 0
# Adding 00000000 + 00001000 + 0, Sum = 00001000, Cout = 0
# Adding 00000000 + 00001001 + 0, Sum = 00001001, Cout = 0
# Adding 00000000 + 00001010 + 0, Sum = 00001010, Cout = 0
# Adding 00000000 + 00001011 + 0, Sum = 00001011, Cout = 0
# Adding 00000000 + 00001100 + 0, Sum = 00001100, Cout = 0
# Adding 00000000 + 00001101 + 0, Sum = 00001101, Cout = 0
# Adding 00000000 + 00001110 + 0, Sum = 00001110, Cout = 0
# Adding 00000000 + 00001111 + 0, Sum = 00001111, Cout = 0
# Adding 00000000 + 00010000 + 0, Sum = 00010000, Cout = 0
# Adding 00000000 + 00010001 + 0, Sum = 00010001, Cout = 0
# Adding 00000000 + 00010010 + 0, Sum = 00010010, Cout = 0
# Adding 00000000 + 00010011 + 0, Sum = 00010011, Cout = 0
# Adding 00000000 + 00010100 + 0, Sum = 00010100, Cout = 0
# Adding 00000000 + 00010101 + 0, Sum = 00010101, Cout = 0
# Adding 00000000 + 00010110 + 0, Sum = 00010110, Cout = 0
# Adding 00000000 + 00010111 + 0, Sum = 00010111, Cout = 0
# Adding 00000000 + 00011000 + 0, Sum = 00011000, Cout = 0
# Adding 00000000 + 00011001 + 0, Sum = 00011001, Cout = 0
# Adding 00000000 + 00011010 + 0, Sum = 00011010, Cout = 0
# Adding 00000000 + 00011011 + 0, Sum = 00011011, Cout = 0
# Adding 00000000 + 00011100 + 0, Sum = 00011100, Cout = 0
# Adding 00000000 + 00011101 + 0, Sum = 00011101, Cout = 0
# Adding 00000000 + 00011110 + 0, Sum = 00011110, Cout = 0
# Adding 00000000 + 00011111 + 0, Sum = 00011111, Cout = 0
# Adding 00000000 + 00100000 + 0, Sum = 00100000, Cout = 0
# Adding 00000000 + 00100001 + 0, Sum = 00100001, Cout = 0
# Adding 00000000 + 00100010 + 0, Sum = 00100010, Cout = 0
# Adding 00000000 + 00100011 + 0, Sum = 00100011, Cout = 0
```

b. Write the SV code for a 16-bit carry look ahead adder in the algorithmic model (always_comb).

Designed module:

```
module CarryLookAheadAdderAlwaysComb
#(
parameter Nbits = 16                                    // 16 bits
)
(
        input logic[Nbits-1:0] a_in,                    // first 16 bits input
        input logic[Nbits-1:0] b_in,                    // second 16 bits input
        input bit              Cin,                     // 1 bit carry in input
        output logic [Nbits-1:0] sum,                   // 16 bits sum
        output bit             Cout                     // 1 bit Cout
);

        logic [Nbits:0] C;                              // internal  16 bits carry value
        logic[Nbits-1:0] G;                             // internal 16 bits  carry generate
        logic [Nbits-1:0] P;                            // internal 16 bits carry propagate
        logic [Nbits-1:0] S;                            // internal 16 bits sum

        always_comb
                begin
                        C[0] = Cin;                     // Set the bit 0 equals Carry in
                        P = 'b0;                        // Set 16 bits carry generate equals zero for
preventing latch
                        G = 'b0;                        // Set 16 bits carry propagate equals zero
for preventing latch
                        S = 'b0;                        // Set internal sum equals zero for
preventing latch
                        sum = 'b0;                      // Set sum equals zero for preventing latch
                        Cout = 0;                       // Set Cout equals zero
                        for (int i = 0; i < Nbits; i++) // Go through all bits of 2 inputs
                        begin
                                P[i] = a_in[i] ^ b_in[i];       // Pi = ai XOR bi
                                G[i] = a_in[i] & b_in[i];       // Gi = ai AND bi
                                S[i] = P[i] ^ C[i];             // Si = pi XOR cin_i
                                C[i+1] = G[i] | (C[i] & P[i]); //C[i+1] = Gi OR (C[i] AND P[i]

                        end
                sum = S;                                        // Set sum = S
                Cout = C[Nbits];                                // Set Cout equals MSB of C
        end
endmodule
```

Testbench:

```
module TestbenchCarryLookAheadAlwaysComb;
    parameter           Nbits = 16;     // 16 bits
    bit [Nbits-1:0]     a_in;           // first 16 bits input
    bit [Nbits-1:0]     b_in;           // second 16 bits input
    bit                 Cin;            // 1 bit carry in input
    bit [Nbits-1:0]     sum;            // 16 bits sum
    bit                 Cout;           // 1 bit Cout

    wire [Nbits-1:0] C;                 // internal 16 bits carry value
    wire [Nbits-1:0] G;                 // internal 16 bits  carry generate
    wire [Nbits-1:0] P;                 // internal 16 bits carry propagate
    wire [Nbits-1:0] S;                 // internal 16 bits sum

    integer Error;                      // Error variable
    reg [Nbits:0] result;               //16 bits results use for comparing,
checking the correctness of module

    CarryLookAheadAdderAlwaysComb # (Nbits) CLAAAC (.*);// Instantiate
CarryLookAheadAdderAlwaysComb module

    initial
    begin
    Cin = 0;                            // Set initial value for Cin
    Error = 0;                          // Set initial value for Error = 0

    repeat (2)
    begin
        for (int i = 0; i < 2**Nbits; i++)      // Go through all values of a_in
        begin
            a_in = i;                           // Set a_in = i
            for (int j = 0; j < 2**Nbits; j++)  // Go through all values of b_in
            begin
                b_in = j;                       // Set b_in = j
                result = a_in + b_in + Cin;     // Set result = a_in + b_in + Cin
                #50;
                if ({Cout,sum} != result)       // Check the yield value with
expected result
                begin
                    Error = Error +1;           // Increase value of error when the
result is not correct
                    $display ("Error founded, adding %b + %b + %b, we
expected Cout is %b, sum is %b, however we got Cout is %b, sum is %b", a_in, b_in,Cin, result[Nbits],
result[Nbits-1:0], Cout, sum);
```

```
                    end
                    else
                    begin
                        Error = Error;
                        $display("Adding %b + %b + %b, Sum = %b, Cout =
%b", a_in, b_in, Cin, sum, Cout);
                    end
                end
            end
            Cin = !Cin;                    // Getting inverse value of Cin for second run
        end
        if (Error == 0)
        $display("Congratulation, your module run correctly");
        end
    endmodule
```

Test results:

Please note that we used exhaustive tests, so this is only the snapshot of the result. Also, since we use exhaustive tests, we only test 8 bits instead of 16 bits for the purpose of saving time. For the complete test results of 8 bits, please look at the *TranscriptCarryLookAheadAlwaysComb* file.

```
# Compile of CarrayLookAheadAdderAlwaysComb.sv was successful.
# Compile of TestbenchCarryLookAheadAlwaysComb.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -gui work.TestbenchCarryLookAheadAlwaysComb
# vsim -gui work.TestbenchCarryLookAheadAlwaysComb
# Start time: 16:49:04 on Oct 11,2020
# ** Note: (vsim-3812) Design is being optimized...
# ** Warning:
//thoth.cecs.pdx.edu/Home03/hongan/Desktop/ECE581_Fall20/CarrayLookAheadAdderAlwaysCo
mb.sv(46): (vopt-2182) 'C' might be read before written in always_comb or always @* block.
# ** Note: (vsim-12126) Error and warning message counts have been restored: Errors=0,
Warnings=1.
# Loading sv_std.std
# Loading work.TestbenchCarryLookAheadAlwaysComb(fast)
run -all
# Adding 00000000 + 00000000 + 0, Sum = 00000000, Cout = 0
# Adding 00000000 + 00000001 + 0, Sum = 00000001, Cout = 0
# Adding 00000000 + 00000010 + 0, Sum = 00000010, Cout = 0
# Adding 00000000 + 00000011 + 0, Sum = 00000011, Cout = 0
# Adding 00000000 + 00000100 + 0, Sum = 00000100, Cout = 0
# Adding 00000000 + 00000101 + 0, Sum = 00000101, Cout = 0
# Adding 00000000 + 00000110 + 0, Sum = 00000110, Cout = 0
# Adding 00000000 + 00000111 + 0, Sum = 00000111, Cout = 0
```

Adding 00000000 + 00001000 + 0, Sum = 00001000, Cout = 0
Adding 00000000 + 00001001 + 0, Sum = 00001001, Cout = 0
Adding 00000000 + 00001010 + 0, Sum = 00001010, Cout = 0
Adding 00000000 + 00001011 + 0, Sum = 00001011, Cout = 0
Adding 00000000 + 00001100 + 0, Sum = 00001100, Cout = 0
Adding 00000000 + 00001101 + 0, Sum = 00001101, Cout = 0
Adding 00000000 + 00001110 + 0, Sum = 00001110, Cout = 0
Adding 00000000 + 00001111 + 0, Sum = 00001111, Cout = 0
Adding 00000000 + 00010000 + 0, Sum = 00010000, Cout = 0
Adding 00000000 + 00010001 + 0, Sum = 00010001, Cout = 0
Adding 00000000 + 00010010 + 0, Sum = 00010010, Cout = 0
Adding 00000000 + 00010011 + 0, Sum = 00010011, Cout = 0
Adding 00000000 + 00010100 + 0, Sum = 00010100, Cout = 0
Adding 00000000 + 00010101 + 0, Sum = 00010101, Cout = 0
Adding 00000000 + 00010110 + 0, Sum = 00010110, Cout = 0
Adding 00000000 + 00010111 + 0, Sum = 00010111, Cout = 0
Adding 00000000 + 00011000 + 0, Sum = 00011000, Cout = 0
Adding 00000000 + 00011001 + 0, Sum = 00011001, Cout = 0
Adding 00000000 + 00011010 + 0, Sum = 00011010, Cout = 0
Adding 00000000 + 00011011 + 0, Sum = 00011011, Cout = 0
Adding 00000000 + 00011100 + 0, Sum = 00011100, Cout = 0
Adding 00000000 + 00011101 + 0, Sum = 00011101, Cout = 0

*** Problem 4:

a. Write the SV code for a circuit whose 32-bit output is formed by shifting its 32-bit input three positions to the right and filling the vacant positions with the bit that was in the MSB before the shift occurred (shift arithmetic right).

Designed module:

```
module ShiftArithmeticRight
#(
        parameter Nbits = 32
)
(
        input logic signed      [Nbits-1:0] In,         // 32 bits Input
        output logic signed     [Nbits-1:0] Out         // 32 bits Output
);
        assign Out = In >>> 3;                          // Using Shift Arithmetic Right to
get the value for Output
endmodule
```

***Testbench*:**

```
module TestbenchShiftArithmeticRight;

        parameter                       Nbits = 32;     // 32 bits
        bit signed [Nbits-1:0]          In;             // 32 bits input
        bit signed [Nbits-1:0]          Out;            // 32 bits output
        integer                         Error;          // Error variable

        ShiftArithmeticRight # (Nbits) SAR(.*);         // Instantiate ShiftArithmeticRight module

        initial
        Error = 0;                                      // Set initial value for zero

        initial
        begin
        for (int i = 0; i < 2**Nbits; i++)              // using for loop to go through all input
values
        begin
                In = i;                                 // Set input equals value of i
                #5;
                $display ("In = %b, Out = %b", In, Out);
                #5;
                if (Out != (In >>>3))                   / Seft checking
                begin
                        Error = Error + 1;              // Increasing error value
```

```
                $display ("Error found, In = %b with expected result = %b, however Out = %b", In, In >>> 3, Out);
            end
            else
            begin
                Error = Error;
            end
        end
        if (Error == 0)
            $display ("Good job, no error found");
        else
            $display ("Sorry, we found %d errors in your module", Error);
        end
    endmodule
```

Test results:

Please note that we used exhaustive tests, so this is only the snapshot of the result. Also, since we use exhaustive tests, we only test 8 bits instead of 32 bits for the purpose of saving time. For the complete test results of 8 bits, please look at the *TranscriptShiftArithmeticRight* file.

```
# Compile of ShiftArithmeticRight.sv was successful.
# Compile of TestbenchShiftArithmeticRight.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -gui work.TestbenchShiftArithmeticRight
# vsim -gui work.TestbenchShiftArithmeticRight
# Start time: 19:49:29 on Oct 11,2020
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.TestbenchShiftArithmeticRight(fast)
run -all
# In = 00000000, Out = 00000000
# In = 00000001, Out = 00000000
# In = 00000010, Out = 00000000
# In = 00000011, Out = 00000000
# In = 00000100, Out = 00000000
# In = 00000101, Out = 00000000
# In = 00000110, Out = 00000000
# In = 00000111, Out = 00000000
# In = 00001000, Out = 00000001
# In = 00001001, Out = 00000001
# In = 00001010, Out = 00000001
# In = 00001011, Out = 00000001
# In = 00001100, Out = 00000001
# In = 00001101, Out = 00000001
# In = 00001110, Out = 00000001
# In = 00001111, Out = 00000001
# In = 00010000, Out = 00000010
```

```
# In = 00010001, Out = 00000010
# In = 00010010, Out = 00000010
# In = 00010011, Out = 00000010
# In = 00010100, Out = 00000010
# In = 00010101, Out = 00000010
# In = 00010110, Out = 00000010
# In = 00010111, Out = 00000010
# In = 00011000, Out = 00000011
# In = 00011001, Out = 00000011
# In = 00011010, Out = 00000011
# In = 00011011, Out = 00000011
# In = 00011100, Out = 00000011
# In = 00011101, Out = 00000011
# In = 00011110, Out = 00000011
```

b. Write the SV code for a circuit whose 32-bit output is formed by shifting its 32-bit input three positions to the left and filling the vacant positions with 0 (shift logical left).

***Designed module*:**

```
module ShiftLogicalLeft
#(
        parameter Nbits = 32
)
(
        input logic signed      [Nbits-1:0] In,             // 32 bits Input
        output logic signed     [Nbits-1:0] Out             // 32 bits Output
);
        assign Out = In << 3;                   // Using Shift Logical Left to get the value
for Output
endmodule
```

Test bench:

```
module TestbenchShiftLogicalLeft;

        parameter               Nbits = 32;     // 32 bits
        bit signed [Nbits-1:0]  In;             // 32 bits input
        bit signed [Nbits-1:0]  Out;            // 32 bits output
        integer                 Error;          // Error variable

    ShiftLogicalLeft # (Nbits) SLL(.*);                 // Instantiate ShiftLogicalLeft
module

        initial
        Error = 0;                                      // Set initial value for zero
```

```
        initial
        begin
        for (int i = 0; i < 2**Nbits; i++)                  // Using for loop to go through all
input values
        begin
                In = i;                                     // Set input equals value of i
                #5;
                $display ("In = %b, Out = %b", In, Out);
                #5;
                if (Out != (In << 3))                       // Seft checking
                begin
                        Error = Error + 1;                          // Increasing error value
                        $display ("Error found, In = %b with expected result = %b, however Out
= %b", In, In << 2, Out);
                end
                else
                begin
                        Error = Error;
                end
        end
        if (Error == 0)
                $display ("Good job, no error found");
        else
                $display ("Sorry, we found %d errors in your module", Error);
    end
endmodule
```

Test results:

Please note that we used exhaustive tests, so this is only the snapshot of the result. Also, since we use exhaustive tests, we only test 8 bits instead of 32 bits for the purpose of saving time. For the complete test results of 8 bits, please look at the *TranscriptShiftLogicalLeft* file.

```
# Compile of ShiftLogicalLeft.sv was successful.
# Compile of TestbenchShiftLogicalLeft.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -gui work.TestbenchShiftLogicalLeft
# vsim -gui work.TestbenchShiftLogicalLeft
# Start time: 22:00:21 on Oct 11,2020
# ** Note: (vsim-8009) Loading existing optimized design _opt1
# Loading sv_std.std
# Loading work.TestbenchShiftLogicalLeft(fast)
run -all
# In = 00000000, Out = 00000000
# In = 00000001, Out = 00001000
# In = 00000010, Out = 00010000
# In = 00000011, Out = 00011000
```

In = 00000100, Out = 00100000
In = 00000101, Out = 00101000
In = 00000110, Out = 00110000
In = 00000111, Out = 00111000
In = 00001000, Out = 01000000
In = 00001001, Out = 01001000
In = 00001010, Out = 01010000
In = 00001011, Out = 01011000
In = 00001100, Out = 01100000
In = 00001101, Out = 01101000
In = 00001110, Out = 01110000
In = 00001111, Out = 01111000
In = 00010000, Out = 10000000
In = 00010001, Out = 10001000
In = 00010010, Out = 10010000
In = 00010011, Out = 10011000
In = 00010100, Out = 10100000
In = 00010101, Out = 10101000
In = 00010110, Out = 10110000
In = 00010111, Out = 10111000
In = 00011000, Out = 11000000

*** Problem 5:

In this problem, I have a parameter to check with another number of bits instead of a fix number (6) in the designed module. Then, I set 6 for the parameter or 6-bits data in and out

a. Write the SV code for a 6-bit Binary-to-Gray code converter in the dataflow model (continuous assign)

Designed module:

```
module problem5a
    #(parameter NUM = 6)                        //6-bits
    (
       input logic [5:0] b_in,                          //binary input
      output wire [5:0] g_out                           //gray code output
    );

    timeunit 1ns/1ns;

    genvar i;
    assign g_out[NUM-1] = b_in[NUM-1];

    //Iteration
    for  (i = NUM-2; i >= 0; i--) begin: loop
       assign g_out[i] = b_in[i+1] ^ b_in[i];
    end: loop
endmodule: problem5a
```

Testbench:

```
module tb_problem5a (b_in ,g_out);
    parameter NUM = 6;                          // 6-bits
    input logic [(NUM -1):0] b_in;          // 6-bits input
    output wire [(NUM -1):0] g_out;             // 6-bits output

    timeunit 1ns/1ns;
    // Instantiate the unit
       problem5a #(NUM) b2g(.*);

    //monitor
    initial begin
       $monitor($time, "\t\t Binary: %b \t\t Gray: %b", b_in, g_out);
    end
       // Test common combinations
    initial begin
       $display("Test for problem 5a:\n");
```

```
        // stimulus 000000 -> 111111
        for (int i = 0; i< 2**NUM; i++) begin: loop
          b_in <= i;
          #1;
        end: loop
       #5 $stop;
    end
endmodule: tb_problem5a
```

Test results:

```
# Compile of problem5a.sv was successful.
# Compile of tb_problem5a.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -voptargs=+acc work.tb_problem5a
# vsim -voptargs="+acc" work.tb_problem5a
# Start time: 21:54:31 on Oct 13,2020
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.tb_problem5a(fast)
# Loading work.problem5a(fast)
run -all
# Test for problem 5a:
#
#              0        Binary: 000000          Gray: 000000
#              1        Binary: 000001          Gray: 000001
#              2        Binary: 000010          Gray: 000011
#              3        Binary: 000011          Gray: 000010
#              4        Binary: 000100          Gray: 000110
#              5        Binary: 000101          Gray: 000111
#              6        Binary: 000110          Gray: 000101
#              7        Binary: 000111          Gray: 000100
#              8        Binary: 001000          Gray: 001100
#              9        Binary: 001001          Gray: 001101
#             10        Binary: 001010          Gray: 001111
#             11        Binary: 001011          Gray: 001110
#             12        Binary: 001100          Gray: 001010
#             13        Binary: 001101          Gray: 001011
#             14        Binary: 001110          Gray: 001001
#             15        Binary: 001111          Gray: 001000
#             16        Binary: 010000          Gray: 011000
#             17        Binary: 010001          Gray: 011001
#             18        Binary: 010010          Gray: 011011
#             19        Binary: 010011          Gray: 011010
#             20        Binary: 010100          Gray: 011110
```

```
#          21          Binary: 010101              Gray: 011111
#          22          Binary: 010110              Gray: 011101
#          23          Binary: 010111              Gray: 011100
#          24          Binary: 011000              Gray: 010100
#          25          Binary: 011001              Gray: 010101
#          26          Binary: 011010              Gray: 010111
#          27          Binary: 011011              Gray: 010110
#          28          Binary: 011100              Gray: 010010
#          29          Binary: 011101              Gray: 010011
#          30          Binary: 011110              Gray: 010001
#          31          Binary: 011111              Gray: 010000
#          32          Binary: 100000              Gray: 110000
#          33          Binary: 100001              Gray: 110001
#          34          Binary: 100010              Gray: 110011
#          35          Binary: 100011              Gray: 110010
#          36          Binary: 100100              Gray: 110110
#          37          Binary: 100101              Gray: 110111
#          38          Binary: 100110              Gray: 110101
#          39          Binary: 100111              Gray: 110100
#          40          Binary: 101000              Gray: 111100
#          41          Binary: 101001              Gray: 111101
#          42          Binary: 101010              Gray: 111111
#          43          Binary: 101011              Gray: 111110
#          44          Binary: 101100              Gray: 111010
#          45          Binary: 101101              Gray: 111011
#          46          Binary: 101110              Gray: 111001
#          47          Binary: 101111              Gray: 111000
#          48          Binary: 110000              Gray: 101000
#          49          Binary: 110001              Gray: 101001
#          50          Binary: 110010              Gray: 101011
#          51          Binary: 110011              Gray: 101010
#          52          Binary: 110100              Gray: 101110
#          53          Binary: 110101              Gray: 101111
#          54          Binary: 110110              Gray: 101101
#          55          Binary: 110111              Gray: 101100
#          56          Binary: 111000              Gray: 100100
#          57          Binary: 111001              Gray: 100101
#          58          Binary: 111010              Gray: 100111
#          59          Binary: 111011              Gray: 100110
#          60          Binary: 111100              Gray: 100010
#          61          Binary: 111101              Gray: 100011
#          62          Binary: 111110              Gray: 100001
#          63          Binary: 111111              Gray: 100000
# ** Note: $stop    : N:/Desktop/ECE 581/P1/tb_problem5a.sv(29)
```

```
#    Time: 69 ns  Iteration: 0  Instance: /tb_problem5a
# Break in Module tb_problem5a at N:/Desktop/ECE 581/P1/tb_problem5a.sv line 29
```

b. Write the SV code for a 6-bit Gray-to-Binary code converter in the algorithmic model (always_comb).

Designed module:

```
module problem5b
    #(parameter NUM = 6)                    //6-bits
    (input logic [(NUM-1):0] g_in,          // 6-bits input
    output logic [(NUM-1):0] b_out          // 6-bits output
    );

    timeunit 1ns/1ns;

    // Initiate instances
    always_comb begin: combination
        b_out[NUM-1] = g_in[NUM-1];
        //Iteration
        for(int i = NUM-2; i >= 0; i--) begin: loop
                b_out[i] = b_out[i+1] ^ g_in[i];
        end: loop
    end: combination
endmodule: problem5b
```

Testbench:

```
module tb_problem5b(g_in, b_out);
    parameter NUM = 6;
    input logic [(NUM -1):0] g_in;
    output logic [(NUM-1):0] b_out;

    timeunit 1ns/1ns;

    // Initiate unit
    problem5b #(NUM) g2b (.*);

    // monitor
    initial begin: monitor
        $monitor($time, "\t Gray: %b \t\t Binary: %b", g_in, b_out);
    end: monitor

    // Test common combinations
    initial begin
```

```
        $display("Test Gray code to Binary: \n");
        // stimulus 000000 -> 111111
        for(int i = 0; i < 2**NUM; i++) begin:loop
                g_in <= i;
                #1;
        end: loop
        #5 $stop;
    end
endmodule: tb_problem5b
```

Test results:

```
# Compile of problem5b.sv was successful.
# Compile of tb_problem5b.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -voptargs=+acc work.tb_problem5b
# vsim -voptargs="+acc" work.tb_problem5b
# Start time: 22:36:10 on Oct 13,2020
# ** Note: (vsim-3812) Design is being optimized...
# ** Warning: (vopt-31) Unable to unlink file "N:/Desktop/ECE
581/P1/work/@_opt2/_lib3_0.qpg".
# ** Warning: (vopt-31) Unable to unlink file "N:/Desktop/ECE
581/P1/work/@_opt2/_lib3_0.qtl".
# The process cannot access the file because it is being used by another process.
(GetLastError() = 32)
# ** Warning: (vopt-133) Unable to remove directory "N:/Desktop/ECE
581/P1/work/@_opt2".
# ** Note: (vsim-12126) Error and warning message counts have been restored: Errors=0,
Warnings=3.
# Loading sv_std.std
# Loading work.tb_problem5b(fast)
# Loading work.problem5b(fast)
run -all
# Test Gray code to Binary:
#
#              0    Gray: 000000              Binary: 000000
#              1    Gray: 000001              Binary: 000001
#              2    Gray: 000010              Binary: 000011
#              3    Gray: 000011              Binary: 000010
#              4    Gray: 000100              Binary: 000111
#              5    Gray: 000101              Binary: 000110
#              6    Gray: 000110              Binary: 000100
#              7    Gray: 000111              Binary: 000101
#              8    Gray: 001000              Binary: 001111
#              9    Gray: 001001              Binary: 001110
```

```
#    10  Gray: 001010    Binary: 001100
#    11  Gray: 001011    Binary: 001101
#    12  Gray: 001100    Binary: 001000
#    13  Gray: 001101    Binary: 001001
#    14  Gray: 001110    Binary: 001011
#    15  Gray: 001111    Binary: 001010
#    16  Gray: 010000    Binary: 011111
#    17  Gray: 010001    Binary: 011110
#    18  Gray: 010010    Binary: 011100
#    19  Gray: 010011    Binary: 011101
#    20  Gray: 010100    Binary: 011000
#    21  Gray: 010101    Binary: 011001
#    22  Gray: 010110    Binary: 011011
#    23  Gray: 010111    Binary: 011010
#    24  Gray: 011000    Binary: 010000
#    25  Gray: 011001    Binary: 010001
#    26  Gray: 011010    Binary: 010011
#    27  Gray: 011011    Binary: 010010
#    28  Gray: 011100    Binary: 010111
#    29  Gray: 011101    Binary: 010110
#    30  Gray: 011110    Binary: 010100
#    31  Gray: 011111    Binary: 010101
#    32  Gray: 100000    Binary: 111111
#    33  Gray: 100001    Binary: 111110
#    34  Gray: 100010    Binary: 111100
#    35  Gray: 100011    Binary: 111101
#    36  Gray: 100100    Binary: 111000
#    37  Gray: 100101    Binary: 111001
#    38  Gray: 100110    Binary: 111011
#    39  Gray: 100111    Binary: 111010
#    40  Gray: 101000    Binary: 110000
#    41  Gray: 101001    Binary: 110001
#    42  Gray: 101010    Binary: 110011
#    43  Gray: 101011    Binary: 110010
#    44  Gray: 101100    Binary: 110111
#    45  Gray: 101101    Binary: 110110
#    46  Gray: 101110    Binary: 110100
#    47  Gray: 101111    Binary: 110101
#    48  Gray: 110000    Binary: 100000
#    49  Gray: 110001    Binary: 100001
#    50  Gray: 110010    Binary: 100011
#    51  Gray: 110011    Binary: 100010
#    52  Gray: 110100    Binary: 100111
#    53  Gray: 110101    Binary: 100110
```

```
#               54    Gray: 110110                Binary: 100100
#               55    Gray: 110111                Binary: 100101
#               56    Gray: 111000                Binary: 101111
#               57    Gray: 111001                Binary: 101110
#               58    Gray: 111010                Binary: 101100
#               59    Gray: 111011                Binary: 101101
#               60    Gray: 111100                Binary: 101000
#               61    Gray: 111101                Binary: 101001
#               62    Gray: 111110                Binary: 101011
#               63    Gray: 111111                Binary: 101010
# ** Note: $stop    : N:/Desktop/ECE 581/P1/tb_problem5b.sv(30)
#    Time: 69 ns  Iteration: 0  Instance: /tb_problem5b
# Break in Module tb_problem5b at N:/Desktop/ECE 581/P1/tb_problem5b.sv line 30
```

*** Problem 6:

A comparator is a logic macro circuit that compares the magnitude of two n-bit binary operands. A 16-bit comparator determines if a 16-bit vector a[15:0] is equal to a 16-bit vector b[15:0].

a. Write the SV code for a 16-bit comparator in the dataflow model (continuous assignments)

Designed module:

```
module ComparatorContinuousAssignments
#(parameter Nbits = 16)                    // 16 bits
(
   input logic[Nbits-1:0] a_in,            // first 16 bits input
   input logic[Nbits-1:0] b_in,            // second 16 bits input
   output logic           out              // 1 bit output
);
   logic [Nbits:0] [1:0]          temp;          // temporary array to hold value for output
   localparam                     True = 1;
   localparam                     False = 0;

   genvar i;
   generate
         for (i = 0; i < Nbits; i++)              // For loop go through all bits of a_in and
b_in
         begin
               assign temp [i] = ((a_in[i] != b_in[i]) ? False :temp[i+1]); // When a_in[i]
not equal value b_in[i] result will be false,
                                                  // otherwise, set temp[i] = temp[i+1] to check next
bit
         end
   endgenerate
   assign temp[Nbits] = True;                          // Assign the last array in temp
equals True since we already checked all bits and they are equal
   assign out = temp[0];                        // Assign out = first array in temp as the
condition for checking and getting value of out

endmodule
```

Testbench:

```
module TestbenchComparatorContinuous;

        parameter               Nbits = 16;             // 16 bits
        parameter               True = 1;
        parameter               False = 0;
        bit [Nbits-1:0]         a_in;                   // first 16 bits input
        bit [Nbits-1:0]     b_in;                       // second 16 bits input
        bit                     out;                    // 1 bit output
        reg                     Error = 0;              // Error variable
        bit                     result;                 // 1 bit result

        ComparatorContinuousAssignments # (Nbits) CCA(.*); // Instantiate
ComparatorContinuousAssignments module
        initial
        begin
                begin
                        for (int i = 0; i < 2**Nbits; i++)          // Go through all values of
a_in
                        begin
                                a_in = i;                           // Set a_in = i
                                for (int j = 0; j < 2**Nbits; j++)  // Go through all values of
b_in
                                begin
                                        b_in = j;                   // Set b_in = j
                                        #10
                                        if (a_in > b_in | b_in > a_in )
                                                result = False;
                                        else
                                                result = True;
                                        #10;
                                        if (out!= result)           // Check the yield value with
expected result
                                        begin
                                                Error = Error +1;   // Increase value of error
when the result is not correct
                                                $display ("Error founded,Compare %b with %b,
out = %b, result = %b", a_in, b_in, out, result);
                                        end
                                        else
                                        begin
                                                Error = Error;
                                                $display("Compare %b with %b, out = %b", a_in,
b_in, out);
```

```
                    end
                end
            end
        end
        if (Error == 0)
        $display("Congratulation, your module correctly");
    end
endmodule
```

Test results:

Please note that we used exhaustive tests, so this is only the snapshot of the result. Also, since we use exhaustive tests, we only test 8 bits instead of 16 bits for the purpose of saving time. For the complete test results of 8 bits, please look at the *TranscriptComparatorContinuousAssignment* file.

```
# Compile of ComparatorContinuousAssignments.sv was successful.
# Compile of TestbenchComparatorContinuous.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -gui work.TestbenchComparatorContinuous
# vsim -gui work.TestbenchComparatorContinuous
# Start time: 23:00:03 on Oct 13,2020
# ** Note: (vsim-8009) Loading existing optimized design _opt
# Loading sv_std.std
# Loading work.TestbenchComparatorContinuous(fast)
run -all
# Compare 00000000 with 00000000, out = 1
# Compare 00000000 with 00000001, out = 0
# Compare 00000000 with 00000010, out = 0
# Compare 00000000 with 00000011, out = 0
# Compare 00000000 with 00000100, out = 0
# Compare 00000000 with 00000101, out = 0
# Compare 00000000 with 00000110, out = 0
# Compare 00000000 with 00000111, out = 0
# Compare 00000000 with 00001000, out = 0
# Compare 00000000 with 00001001, out = 0
# Compare 00000000 with 00001010, out = 0
# Compare 00000000 with 00001011, out = 0
# Compare 00000000 with 00001100, out = 0
# Compare 00000000 with 00001101, out = 0
# Compare 00000000 with 00001110, out = 0
# Compare 00000000 with 00001111, out = 0
# Compare 00000000 with 00010000, out = 0
# Compare 00000000 with 00010001, out = 0
# Compare 00000000 with 00010010, out = 0
# Compare 00000000 with 00010011, out = 0
# Compare 00000000 with 00010100, out = 0
# Compare 00000000 with 00010101, out = 0
# Compare 00000000 with 00010110, out = 0
# Compare 00000000 with 00010111, out = 0
# Compare 00000000 with 00011000, out = 0
```

```
# Compare 00000000 with 00011001, out = 0
# Compare 00000000 with 00011010, out = 0
# Compare 00000000 with 00011011, out = 0
# Compare 00000000 with 00011100, out = 0
# Compare 00000000 with 00011101, out = 0
# Compare 00000000 with 00011110, out = 0
```

b. Write the SV code for a 16-bit comparator in the algorithmic model (always_comb).

Designed module:

```
module ComparatorAlwaysComb
#(
parameter Nbits = 16                                    // 16 bits
)
(
        input logic[Nbits-1:0]      a_in,               // first 16 bits input
        input logic[Nbits-1:0]      b_in,               // second 16 bits input
        output logic                out                 // 1 bit output
);

        localparam                  True = 1;           // Set True equals 1
        localparam                  False = 0;          // Set False equals 0

        always_comb
        begin
                for (int i = 0; i < Nbits; i++)         // Go through all bits of a_in and
b_in
                begin
                        if (a_in[i] != b_in[i])         // When bit a_in[i] not equal b_in[i]
                        begin
                                out = False;            // Set output equals False and exit
from for loop
                                break;
                        end
                        else
                                out = True;             // Else set out equals True and
continue until the last bit
                end
        end
endmodule
```

Testbench:

```
module TestbenchComparatorAlwaysComb;

        parameter               Nbits = 16;             // 16 bits
        parameter               True = 1;
        parameter               False = 0;
        bit [Nbits-1:0]         a_in;                   // first 16 bits input
        bit [Nbits-1:0]         b_in;                   // second 16 bits input
        bit                     out;                    // 1 bit output
        reg                     Error = 0;              // Error variable
        bit                     result;                 // 1 bit result
         ComparatorAlwaysComb # (Nbits) CAC(.*);        // Instantiate
TestbenchComparatorAlwaysComb module

        initial
        begin
                begin
                        for (int i = 0; i < 2**Nbits; i++)      // Go through all values of
a_in
                        begin
                                a_in = i;
                // Set a_in = i
                                for (int j = 0; j < 2**Nbits; j++)      // Go through all values of
b_in
                                begin
                                        b_in = j;
                // Set b_in = j
                                        #10
                                        if (a_in > b_in | b_in > a_in )
                                                result = False;
                                        else
                                                result = True;
                                        #10;
                                        if (out!= result)               // Check the yield value with
expected result
                                        begin
                                                Error = Error +1;       // Increase value of error
when the result is not correct
                                                $display ("Error founded,Compare %b with %b,
out = %b, result = %b", a_in, b_in, out, result);
                                        end
                                        else
                                        begin
                                                Error = Error;
```

```
                                        $display("Compare %b with %b, out = %b", a_in,
    b_in, out);
                                    end
                                end
                            end
                        end
                        if (Error == 0)
                        $display("Congratulation, your module correctly");
                end
        endmodule
```

Test results:

Please note that we used exhaustive tests, so this is only the snapshot of the result. Also, since we use exhaustive tests, we only test 8 bits instead of 16 bits for the purpose of saving time. For the complete test results of 8 bits, please look at the *TranscriptComparatorAlwaysComb* file.

```
# Compile of ComparatorAlwaysComb.sv was successful.
# Compile of TestbenchComparatorAlwaysComb.sv was successful.
# 2 compiles, 0 failed with no errors.
vsim -gui work.TestbenchComparatorAlwaysComb
# vsim -gui work.TestbenchComparatorAlwaysComb
# Start time: 23:36:45 on Oct 13,2020
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.TestbenchComparatorAlwaysComb(fast)
run -all
# Compare 00000000 with 00000000, out = 1
# Compare 00000000 with 00000001, out = 0
# Compare 00000000 with 00000010, out = 0
# Compare 00000000 with 00000011, out = 0
# Compare 00000000 with 00000100, out = 0
# Compare 00000000 with 00000101, out = 0
# Compare 00000000 with 00000110, out = 0
# Compare 00000000 with 00000111, out = 0
# Compare 00000000 with 00001000, out = 0
# Compare 00000000 with 00001001, out = 0
# Compare 00000000 with 00001010, out = 0
# Compare 00000000 with 00001011, out = 0
# Compare 00000000 with 00001100, out = 0
# Compare 00000000 with 00001101, out = 0
# Compare 00000000 with 00001110, out = 0
# Compare 00000000 with 00001111, out = 0
# Compare 00000000 with 00010000, out = 0
# Compare 00000000 with 00010001, out = 0
# Compare 00000000 with 00010010, out = 0
# Compare 00000000 with 00010011, out = 0
# Compare 00000000 with 00010100, out = 0
# Compare 00000000 with 00010101, out = 0
```

```
# Compare 00000000 with 00010110, out = 0
# Compare 00000000 with 00010111, out = 0
# Compare 00000000 with 00011000, out = 0
# Compare 00000000 with 00011001, out = 0
# Compare 00000000 with 00011010, out = 0
# Compare 00000000 with 00011011, out = 0
```