ECE 366: Project 1 Progress Report Due: 02/27/2025

Team Members: Shahed Alhanbali, Cindy Jurado, Qudsia Sultana
Link to GitHub Repository: https://github.com/shahedalhanbali/366-Project-1

1. Summary of Project Status

As of the deadline (February 27, 2025, at 11:59 PM Chicago Time), the team has made significant progress on the project. The following tasks have been completed:

- Completion of Problem 1 (Split Among Team Members):
    - **Qudsia Sultana:** Worked on designing the 1-bit full adder using both behavioral and structural modeling. She ensured that the logic correctly implemented sum and carry operations, verifying its accuracy through testbench simulations. Additionally, Qudsia assisted in integrating the 1-bit full adder into the 4-bit Ripple-Carry Adder (RCA), laying the foundation for the subsequent steps in the project.
    - **Shahed Alhanbali**: Focused on implementing and testing the 4-bit Ripple-Carry Adder (RCA). Using the 1-bit full adder as a building block, Shahed connected multiple instances to create a functional multi-bit adder. She also ran extensive simulations to verify that the RCA performed correctly under various test cases, ensuring accurate binary addition.
    - **Cindy Jurado:** Took responsibility for implementing the 4-bit Ripple-Carry Subtractor (RCS) by modifying the RCA to support subtraction via two’s complement arithmetic. She carefully handled bitwise operations and tested the circuit under different scenarios, validating the correct execution of subtraction, including handling negative numbers using two's complement representation.
- Initiation of Problem 2: The team has started reviewing the enhancements required for constructing a 4-bit Ripple-Carry Subtractor (RCS) and plans to begin implementation soon.
- Challenges and Next Steps: The team is currently verifying the correctness of the models before moving forward with the rest of the project

---

2. Team Responsibilities

Team Member	Responsibilities	Effort
Qudsia Sultana	- Designed 1-bit full adder using behavioral and structural modeling. - Integrated the 1-bit full adder into the 4-bit Ripple-Carry Adder. - Verified correctness of the RCA logic through testbench simulations.	100%

	- Reviewed and checked the overall implementation to ensure correctness.	
Shahed Alhanbali	- Implemented and tested the 4-bit Ripple-Carry Adder (RCA). - Connected multiple 1-bit full adders to create the multi-bit adder. - Verified RCA functionality with extensive test cases. - Set up the GitHub repository for collaboration and version control and also updated the repository with changes made in the project.	100%
Cindy Jurado	- Developed the 4-bit Ripple-Carry Subtractor (RCS) using two's complement. - Modified RCA logic to support binary subtraction. - Tested negative number handling using two's complement representation. - Set up the project progress report for team tracking and documentation.	100%

Each team member has actively contributed to the project by engaging in coding, debugging, and documentation. The collaborative efforts have ensured steady progress towards completion.

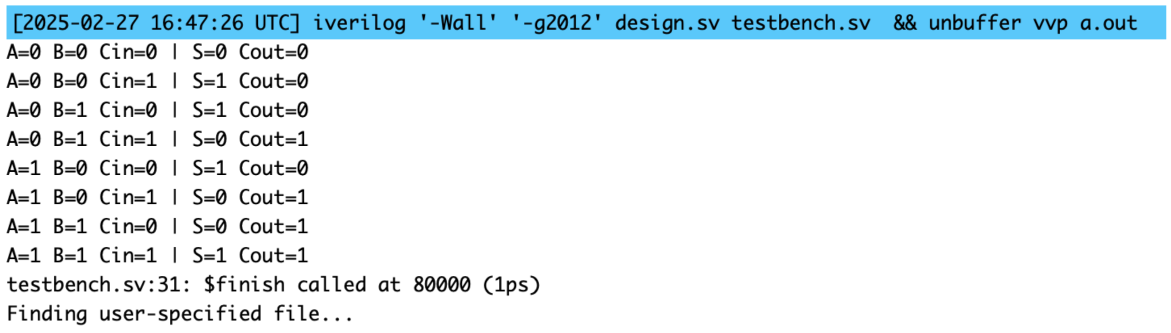
Problem 1

a. Design a 1-bit full-adder in Verilog using behavioral modeling. Assume that the inputs are A, B, and Cin, and the outputs are S, Cout. Use the module template of Figure 1(a). [Points: 5]

Code Design

```
`timescale 1ns / 1ps
module one_bit_full_adder(A, B, Cin, S, Cout);
input A, B, Cin;
output S, Cout;

assign S = A ^ B ^ Cin;
assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule
```

Output



```
[2025-02-27 16:47:26 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A=0 B=0 Cin=0 | S=0 Cout=0
A=0 B=0 Cin=1 | S=1 Cout=0
A=0 B=1 Cin=0 | S=1 Cout=0
A=0 B=1 Cin=1 | S=0 Cout=1
A=1 B=0 Cin=0 | S=1 Cout=0
A=1 B=0 Cin=1 | S=0 Cout=1
A=1 B=1 Cin=0 | S=0 Cout=1
A=1 B=1 Cin=1 | S=1 Cout=1
testbench.sv:31: $finish called at 80000 (1ps)
Finding user-specified file...
```

Code for Testbench

```
`timescale 1ns / 1ps

module tb_one_bit_full_adder;
   reg A, B, Cin;
   wire S, Cout;

   // Instantiate the full adder module
   one_bit_full_adder uut (
      .A(A),
      .B(B),
      .Cin(Cin),
      .S(S),
      .Cout(Cout)
   );

   // Enable waveform dumping
   initial begin
      $dumpfile("waveform.vcd");
      $dumpvars(0, tb_one_bit_full_adder);
   end

   initial begin
      $monitor("A=%b B=%b Cin=%b | S=%b Cout=%b", A, B, Cin, S, Cout);

      // Test all possible input combinations
      A = 0; B = 0; Cin = 0; #10;
      A = 0; B = 0; Cin = 1; #10;
      A = 0; B = 1; Cin = 0; #10;
      A = 0; B = 1; Cin = 1; #10;
      A = 1; B = 0; Cin = 0; #10;
```

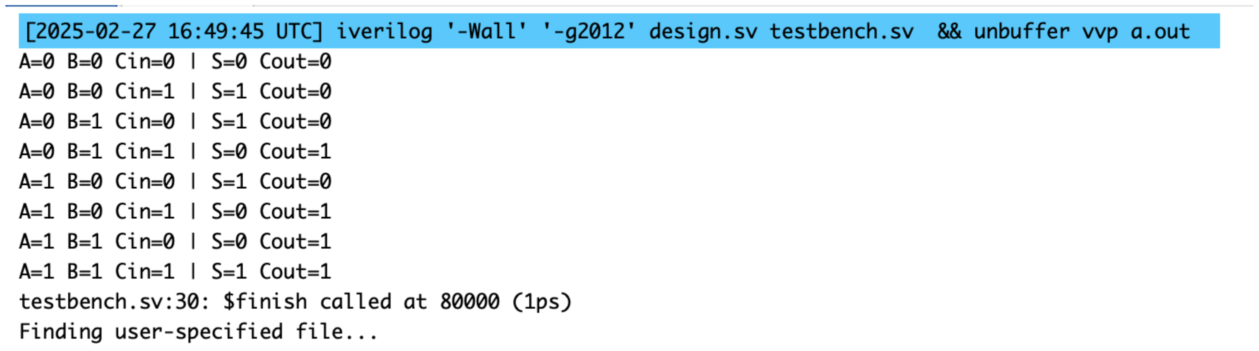
```
        A = 1; B = 0; Cin = 1; #10;
        A = 1; B = 1; Cin = 0; #10;
        A = 1; B = 1; Cin = 1; #10;

        $finish;
    end
endmodule
```

b. Design a 1-bit full-adder in Verilog using structural modeling. Assume that the inputs are A, B, and Cin, and outputs are S, Cout. Use the module template of Figure 1(a). [Points : 5]

<u>Code Design</u>

```
`timescale 1ns / 1ps
module one_bit_full_adder_structural(A, B, Cin, S, Cout);
    input A, B, Cin;
    output S, Cout;

    wire w1, w2, w3;

    xor (w1, A, B);
    xor (S, w1, Cin);
    and (w2, A, B);
    and (w3, w1, Cin);
    or  (Cout, w2, w3);

endmodule
```

<u>Output</u>



```
[2025-02-27 16:49:45 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A=0 B=0 Cin=0 | S=0 Cout=0
A=0 B=0 Cin=1 | S=1 Cout=0
A=0 B=1 Cin=0 | S=1 Cout=0
A=0 B=1 Cin=1 | S=0 Cout=1
A=1 B=0 Cin=0 | S=1 Cout=0
A=1 B=0 Cin=1 | S=0 Cout=1
A=1 B=1 Cin=0 | S=0 Cout=1
A=1 B=1 Cin=1 | S=1 Cout=1
testbench.sv:30: $finish called at 80000 (1ps)
Finding user-specified file...
```

Code for testbench

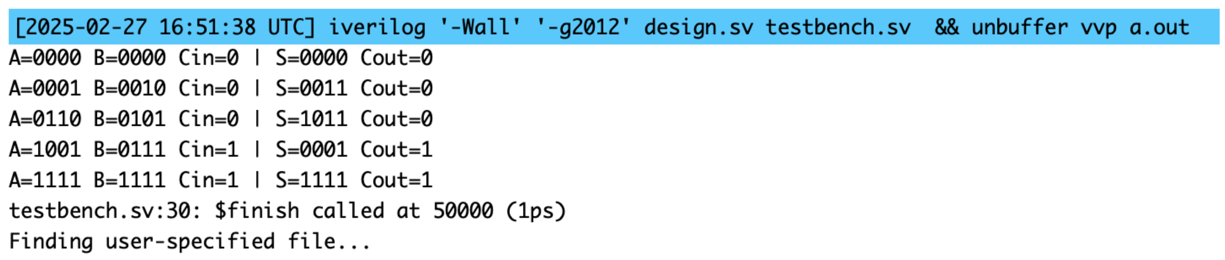
```
`timescale 1ns / 1ps

module tb_one_bit_full_adder_structural;
   reg A, B, Cin;
   wire S, Cout;

   one_bit_full_adder_structural uut (
      .A(A),
      .B(B),
      .Cin(Cin),
      .S(S),
      .Cout(Cout)
   );

   // Enable waveform dumping
   initial begin
      $dumpfile("waveform.vcd");
      $dumpvars(0, tb_one_bit_full_adder_structural);
   end

   initial begin
      $monitor("A=%b B=%b Cin=%b | S=%b Cout=%b", A, B, Cin, S, Cout);

      // Test all possible input combinations
      A = 0; B = 0; Cin = 0; #10;
      A = 0; B = 0; Cin = 1; #10;
      A = 0; B = 1; Cin = 0; #10;
      A = 0; B = 1; Cin = 1; #10;
      A = 1; B = 0; Cin = 0; #10;
      A = 1; B = 0; Cin = 1; #10;
      A = 1; B = 1; Cin = 0; #10;
      A = 1; B = 1; Cin = 1; #10;

      $finish;
   end
endmodule
```

c. Using the 1-bit full adder from either Part (a) or Part (b), design a 4-bit Ripple-Carry Adder (RCA). Use the module template of 1(b). [Points : 10]

Design Code

```
`timescale 1ns / 1ps
module one_bit_full_adder(A, B, Cin, S, Cout);
   input A, B, Cin;
   output S, Cout;

   // Full adder logic
   assign S = A ^ B ^ Cin;
   assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule

module four_bit_RCA(A, B, Cin, S, Cout);
   input [3:0] A, B;
   input Cin;
   output [3:0] S;
   output Cout;

   wire C1, C2, C3;

   // Instantiate four 1-bit full adders
   one_bit_full_adder FA0 (A[0], B[0], Cin,  S[0], C1);
   one_bit_full_adder FA1 (A[1], B[1], C1,   S[1], C2);
   one_bit_full_adder FA2 (A[2], B[2], C2,   S[2], C3);
   one_bit_full_adder FA3 (A[3], B[3], C3,   S[3], Cout);

endmodule
```

Output



```
[2025-02-27 16:51:38 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A=0000 B=0000 Cin=0 | S=0000 Cout=0
A=0001 B=0010 Cin=0 | S=0011 Cout=0
A=0110 B=0101 Cin=0 | S=1011 Cout=0
A=1001 B=0111 Cin=1 | S=0001 Cout=1
A=1111 B=1111 Cin=1 | S=1111 Cout=1
testbench.sv:30: $finish called at 50000 (1ps)
Finding user-specified file...
```

Code for Testbench

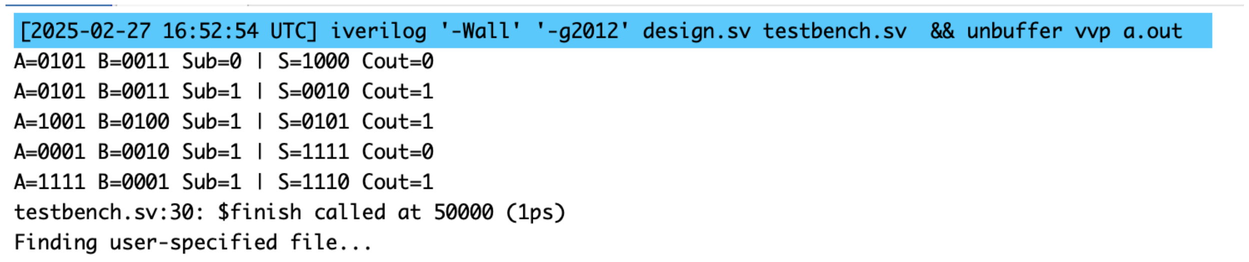
```
`timescale 1ns / 1ps

module tb_four_bit_RCA;
  reg [3:0] A, B;
  reg Cin;
  wire [3:0] S;
  wire Cout;

  // Instantiate the RCA module
  four_bit_RCA uut (
    .A(A),
    .B(B),
    .Cin(Cin),
    .S(S),
    .Cout(Cout)
  );

  initial begin
    $dumpfile("waveform.vcd");
    $dumpvars(0, tb_four_bit_RCA);
  end

  initial begin
    $monitor("A=%b B=%b Cin=%b | S=%b Cout=%b", A, B, Cin, S, Cout);

    // Test cases
    A = 4'b0000; B = 4'b0000; Cin = 0; #10;
    A = 4'b0001; B = 4'b0010; Cin = 0; #10;
    A = 4'b0110; B = 4'b0101; Cin = 0; #10;
    A = 4'b1001; B = 4'b0111; Cin = 1; #10;
    A = 4'b1111; B = 4'b1111; Cin = 1; #10;

    $finish;
  end
endmodule
```

d. Enhance the 4-bit RCA of Part (c) to construct 4-bit Ripple-Carry Subtractor (RCS). Use the module template of 1(b). [Hint: Look at Lecture Slide 2, Slide 18.] [Points: 10]

Design Code

```
`timescale 1ns / 1ps
module one_bit_full_adder(A, B, Cin, S, Cout);
   input A, B, Cin;
   output S, Cout;

   assign S = A ^ B ^ Cin;
   assign Cout = (A & B) | (B & Cin) | (A & Cin);
endmodule

module four_bit_RCS(A, B, Sub, S, Cout);
   input [3:0] A, B;
   input Sub;  // Control signal: 0 for addition, 1 for subtraction
   output [3:0] S;
   output Cout;

   wire [3:0] B_comp; // Complemented B
   wire C1, C2, C3;

   // If Sub = 1, invert B (two's complement)
   assign B_comp = B ^ {4{Sub}};

   // Instantiate four 1-bit full adders
   one_bit_full_adder FA0 (A[0], B_comp[0], Sub,  S[0], C1);
   one_bit_full_adder FA1 (A[1], B_comp[1], C1,   S[1], C2);
   one_bit_full_adder FA2 (A[2], B_comp[2], C2,   S[2], C3);
   one_bit_full_adder FA3 (A[3], B_comp[3], C3,   S[3], Cout);

endmodule
```

Output



```
[2025-02-27 16:52:54 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A=0101 B=0011 Sub=0 | S=1000 Cout=0
A=0101 B=0011 Sub=1 | S=0010 Cout=1
A=1001 B=0100 Sub=1 | S=0101 Cout=1
A=0001 B=0010 Sub=1 | S=1111 Cout=0
A=1111 B=0001 Sub=1 | S=1110 Cout=1
testbench.sv:30: $finish called at 50000 (1ps)
Finding user-specified file...
```

Code for Testbench

```
`timescale 1ns / 1ps

module tb_four_bit_RCS;
   reg [3:0] A, B;
   reg Sub; // Subtraction control signal
   wire [3:0] S;
   wire Cout;

   // Instantiate the RCS module
   four_bit_RCS uut (
      .A(A),
      .B(B),
      .Sub(Sub),
      .S(S),
      .Cout(Cout)
   );

   // Enable waveform dumping
   initial begin
      $dumpfile("waveform.vcd");  // Creates waveform file
      $dumpvars(0, tb_four_bit_RCS);  // Dumps all signals from the testbench
   end

   initial begin
      $monitor("A=%b B=%b Sub=%b | S=%b Cout=%b", A, B, Sub, S, Cout);

      // Test cases
      A = 4'b0101; B = 4'b0011; Sub = 0; #10;  // 5 + 3 = 8
      A = 4'b0101; B = 4'b0011; Sub = 1; #10;  // 5 - 3 = 2
      A = 4'b1001; B = 4'b0100; Sub = 1; #10;  // 9 - 4 = 5
      A = 4'b0001; B = 4'b0010; Sub = 1; #10;  // 1 - 2 = -1 (should show 1111 in 2's
complement)
      A = 4'b1111; B = 4'b0001; Sub = 1; #10;  // -1 - 1 = -2 (should show 1110 in 2's
complement)

      $finish;
   end
endmodule
```

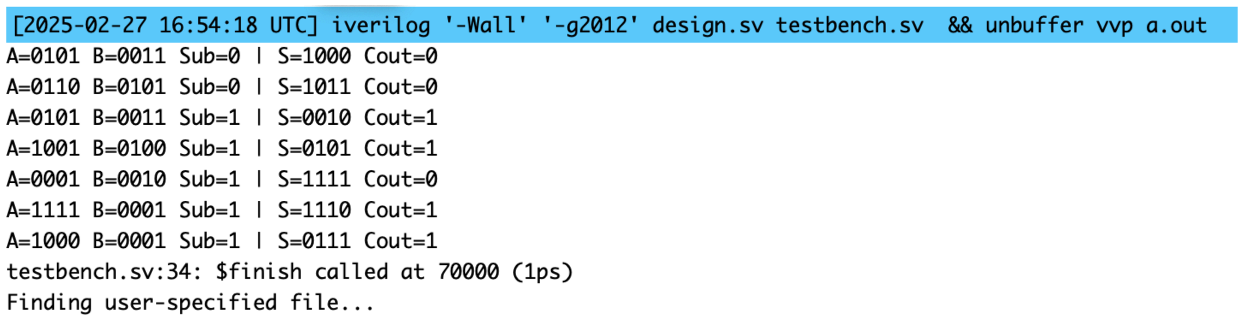
e. Design a testbench to test the 4-bit RCA/RCS. The testbench must perform at least one addition and one subtraction of two numbers (both signed and unsigned numbers). (See the link at Testbench writing). [Hint: See Lecture 1 and Lecture 2 for signed number arithmetic.] [Points: 20]

Design Code

```
`timescale 1ns / 1ps

// 1-bit Full Adder
module one_bit_full_adder(A, B, Cin, S, Cout);
   input A, B, Cin;
   output S, Cout;

   assign S = A ^ B ^ Cin;
   assign Cout = (A & B) | (B & Cin) | (A & Cin);
endmodule

// 4-bit Ripple Carry Adder (RCA)
module four_bit_RCA(A, B, Cin, S, Cout);
   input [3:0] A, B;
   input Cin;
   output [3:0] S;
   output Cout;

   wire C1, C2, C3;

   // Instantiating four 1-bit full adders
   one_bit_full_adder FA0 (A[0], B[0], Cin,  S[0], C1);
   one_bit_full_adder FA1 (A[1], B[1], C1,   S[1], C2);
   one_bit_full_adder FA2 (A[2], B[2], C2,   S[2], C3);
   one_bit_full_adder FA3 (A[3], B[3], C3,   S[3], Cout);
endmodule

// 4-bit Ripple Carry Subtractor (RCS)
module four_bit_RCS(A, B, Sub, S, Cout);
   input [3:0] A, B;
   input Sub;  // Sub = 0 for addition, Sub = 1 for subtraction
   output [3:0] S;
   output Cout;
```

```
    wire [3:0] B_comp;
    wire C1, C2, C3;

    // Perform two's complement if Sub = 1 (invert B and add 1)
    assign B_comp = B ^ {4{Sub}};  // If Sub=1, invert B

    // Instantiating four 1-bit full adders
    one_bit_full_adder FA0 (A[0], B_comp[0], Sub,  S[0], C1);
    one_bit_full_adder FA1 (A[1], B_comp[1], C1,   S[1], C2);
    one_bit_full_adder FA2 (A[2], B_comp[2], C2,   S[2], C3);
    one_bit_full_adder FA3 (A[3], B_comp[3], C3,   S[3], Cout);
endmodule
```

Output



```
[2025-02-27 16:54:18 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A=0101 B=0011 Sub=0 | S=1000 Cout=0
A=0110 B=0101 Sub=0 | S=1011 Cout=0
A=0101 B=0011 Sub=1 | S=0010 Cout=1
A=1001 B=0100 Sub=1 | S=0101 Cout=1
A=0001 B=0010 Sub=1 | S=1111 Cout=0
A=1111 B=0001 Sub=1 | S=1110 Cout=1
A=1000 B=0001 Sub=1 | S=0111 Cout=1
testbench.sv:34: $finish called at 70000 (1ps)
Finding user-specified file...
```

Code for Testbench

```
`timescale 1ns / 1ps

module tb_four_bit_RCA_RCS;
    reg [3:0] A, B;
    reg Sub; // Sub = 0 (Addition), Sub = 1 (Subtraction)
    wire [3:0] S;
    wire Cout;

    // Instantiate the RCS module
    four_bit_RCS uut (
        .A(A),
        .B(B),
        .Sub(Sub),
        .S(S),
        .Cout(Cout)
    );
    initial begin
        $monitor("A=%b B=%b Sub=%b | S=%b Cout=%b", A, B, Sub, S, Cout);

        // Unsigned Addition
        A = 4'b0101; B = 4'b0011; Sub = 0; #10;  // 5 + 3 = 8
        A = 4'b0110; B = 4'b0101; Sub = 0; #10;  // 6 + 5 = 11

        // Unsigned Subtraction
        A = 4'b0101; B = 4'b0011; Sub = 1; #10;  // 5 - 3 = 2
        A = 4'b1001; B = 4'b0100; Sub = 1; #10;  // 9 - 4 = 5

        // Signed Operations (Negative results)
        A = 4'b0001; B = 4'b0010; Sub = 1; #10;  // 1 - 2 = -1 (should show 1111 in 2's
complement)
        A = 4'b1111; B = 4'b0001; Sub = 1; #10;  // -1 - 1 = -2 (should show 1110 in 2's
complement)
        A = 4'b1000; B = 4'b0001; Sub = 1; #10;  // -8 - 1 = -9 (should show 0111 in 2's
complement)

        $finish;
    end
endmodule
```