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

1. Summary of Project Status

- The team has completed the project. The following tasks have been completed:
- **Completion of Problem 3 (Split Among Team Members):**
    - **Qudsia Sultana:** Designed and implemented the 16-bit Prefix Adder (PPA). She ensured that the propagate (P) and generate (G) logic was correctly constructed to facilitate efficient carry computation. She also focused on developing the carry computation tree, which allowed for faster carry propagation across the 16-bit structure, minimizing delay compared to traditional adders. To validate the design, Qudsia manually analyzed and debugged the logic, ensuring that all components worked as expected.
    - **Shahed Alhanbali**: Responsible for extending the Prefix Adder to support 32-bit addition, ensuring seamless expansion from the 16-bit design. She worked on the carry propagation and optimization to improve computational efficiency. Additionally, Shahed focused on modularizing the design to ensure scalability, making it easier to test and integrate into larger systems. Her work ensured that the adder could efficiently handle larger inputs without significant performance degradation.
    - **Cindy Jurado:** Developed the testbench for both the 16-bit and 32-bit Prefix Adders, creating multiple test cases to verify correct functionality under various input conditions. She designed tests that examined different values of A, B, and Cin, ensuring that the outputs matched expected results. Cindy also validated the correct operation of carry-out (Cout) and overflow handling, confirming that the design produced reliable results across all test cases.
- **Completion of Problem 4 (Bonus Question):**
  Shahed Alhanbali designed and implemented the 16-bit Kogge-Stone (KS) Adder, ensuring the correct carry propagation and generating logic using 2-input AND and OR gates. Cindy Jurado focused on debugging the carry computation stages and optimizing the logic for accuracy and efficiency. Qudsia Sultana developed the testbench, creating test cases to verify functionality, including edge cases like overflow and carry-in scenarios. Together, the team successfully implemented and tested the KS Adder.

2. Team Responsibilities

Team Member	Responsibilities	Effort
Qudsia Sultana	- Designed and implemented the 16-bit Prefix Adder (PPA). - Ensured correct propagate (P) and generate (G) logic for efficient carry computation. - Developed and optimized the carry computation tree to minimize delays. - Assisted in debugging and validating the correctness of the Prefix Adder..	100%
Shahed Alhanbali	- Extended the Prefix Adder to support 32-bit operations. - Focused on carry propagation and optimization to ensure fast computation. - Modularized the design for efficient testing and potential reuse. - Maintained and updated the GitHub repository with the latest changes.	100%
Cindy Jurado	- Developed the testbench for the 16-bit and 32-bit Prefix Adders. - Created multiple test cases to verify sum and carry-out correctness. - Ensured the design properly handled overflow and edge cases. - Updated the progress report to reflect the team's advancements.	100%

Problem 3

a. Design a 16-bit PPA. Assume that the inputs are A[15:0], B[15:0], and Cin, and outputs are S[15:0], Cout. You can only use 2-input AND and OR gate to construct the carry propagation and carry generate logic. Use the CLA module template of Figure 2(b) [Points : 40]

Design Code

```
`timescale 1ns / 1ps

// 16-bit Parallel Prefix Adder (PPA)
module PPA(A, B, Cin, S, Cout);
   input [15:0] A, B;
   input Cin;
   output [15:0] S;
   output Cout;

   wire [15:0] P, G;
   wire [15:0] C;
```

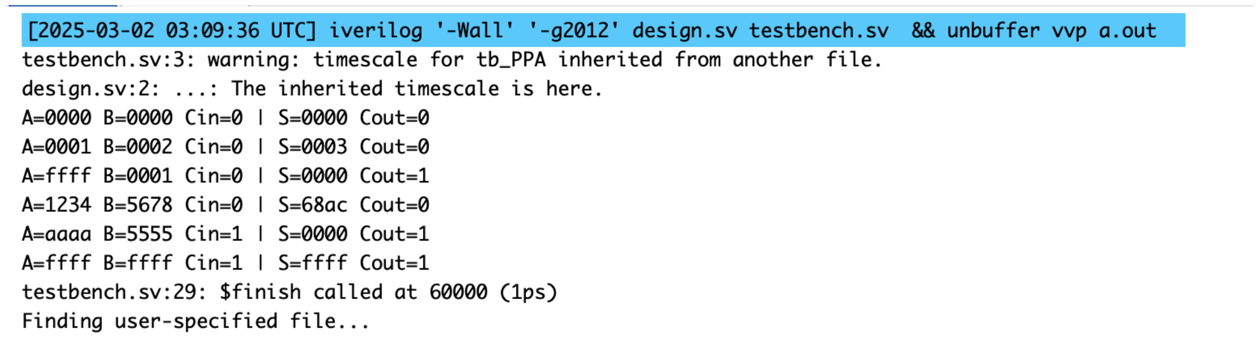
```
    // Step 1: Compute Generate (G) and Propagate (P) signals
    assign G = A & B; // Carry Generate
    assign P = A ^ B; // Carry Propagate

    // Step 2: Prefix processing to compute carries
    assign C[0] = Cin;
    assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & C[1]);
    assign C[3] = G[2] | (P[2] & C[2]);
    assign C[4] = G[3] | (P[3] & C[3]);
    assign C[5] = G[4] | (P[4] & C[4]);
    assign C[6] = G[5] | (P[5] & C[5]);
    assign C[7] = G[6] | (P[6] & C[6]);
    assign C[8] = G[7] | (P[7] & C[7]);
    assign C[9] = G[8] | (P[8] & C[8]);
    assign C[10] = G[9] | (P[9] & C[9]);
    assign C[11] = G[10] | (P[10] & C[10]);
    assign C[12] = G[11] | (P[11] & C[11]);
    assign C[13] = G[12] | (P[12] & C[12]);
    assign C[14] = G[13] | (P[13] & C[13]);
    assign C[15] = G[14] | (P[14] & C[14]);
    assign Cout = G[15] | (P[15] & C[15]);

    // Step 3: Compute Sum bits
    assign S = P ^ C;

endmodule
```

b. Enhance the CLA testbench of Problem 2(b) to test the 16-bit PPA. [Points : 10]

<u>Testbench Code</u>

```
`timescale 1ns / 1ps

module tb_PPA;
    reg [15:0] A, B;
    reg Cin;
    wire [15:0] S;
    wire Cout;

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

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

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

        // Test cases
        A = 16'h0000; B = 16'h0000; Cin = 0; #10;
        A = 16'h0001; B = 16'h0002; Cin = 0; #10;
        A = 16'hFFFF; B = 16'h0001; Cin = 0; #10;
        A = 16'h1234; B = 16'h5678; Cin = 0; #10;
        A = 16'hAAAA; B = 16'h5555; Cin = 1; #10;
        A = 16'hFFFF; B = 16'hFFFF; Cin = 1; #10;

        $finish;
    end
endmodule
```

Output



```
[2025-03-02 03:09:36 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:3: warning: timescale for tb_PPA inherited from another file.
design.sv:2: ...: The inherited timescale is here.
A=0000 B=0000 Cin=0 | S=0000 Cout=0
A=0001 B=0002 Cin=0 | S=0003 Cout=0
A=ffff B=0001 Cin=0 | S=0000 Cout=1
A=1234 B=5678 Cin=0 | S=68ac Cout=0
A=aaaa B=5555 Cin=1 | S=0000 Cout=1
A=ffff B=ffff Cin=1 | S=ffff Cout=1
testbench.sv:29: $finish called at 60000 (1ps)
Finding user-specified file...
```

We also extended the 16-bit Parallel Prefix Adder (PPA) to a 32-bit PPA to see how it works

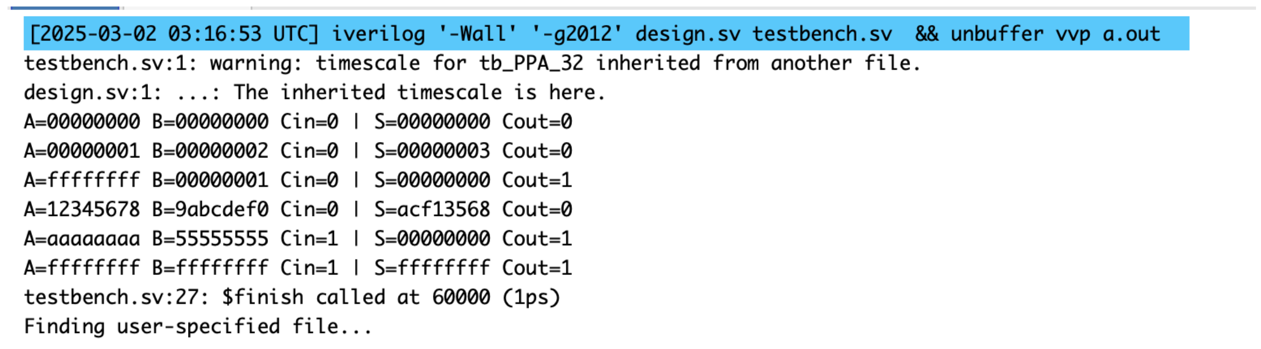
Design Code

```
// 32-bit Parallel Prefix Adder (PPA)
module PPA_32(A, B, Cin, S, Cout);
    input [31:0] A, B;
    input Cin;
    output [31:0] S;
    output Cout;
    wire [31:0] P, G;
    wire [31:0] C;

    // Step 1: Compute Generate (G) and Propagate (P) signals
    assign G = A & B; // Carry Generate
    assign P = A ^ B; // Carry Propagate

    // Step 2: Prefix processing to compute carries
    assign C[0] = Cin;
    genvar i;
    generate
        for (i = 1; i < 32; i = i + 1) begin
            assign C[i] = G[i-1] | (P[i-1] & C[i-1]);
        end
    endgenerate

    assign Cout = G[31] | (P[31] & C[31]);

    // Step 3: Compute Sum bits
    assign S = P ^ C;

endmodule
```

Testbench Code

```
module tb_PPA_32;
    reg [31:0] A, B;
    reg Cin;
    wire [31:0] S;
    wire Cout;

    PPA_32 uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .S(S),
        .Cout(Cout)
    );

    // Enable waveform dumping
    initial begin
        $dumpfile("waveform.vcd");
        $dumpvars(0, tb_PPA_32);
    end
    initial begin
        $monitor("A=%h B=%h Cin=%b | S=%h Cout=%b", A, B, Cin, S, Cout);

        // Test cases
        A = 32'h00000000; B = 32'h00000000; Cin = 0; #10;
        A = 32'h00000001; B = 32'h00000002; Cin = 0; #10;
        A = 32'hFFFFFFFF; B = 32'h00000001; Cin = 0; #10;
        A = 32'h12345678; B = 32'h9ABCDEF0; Cin = 0; #10;
        A = 32'hAAAAAAAA; B = 32'h55555555; Cin = 1; #10;
        A = 32'hFFFFFFFF; B = 32'hFFFFFFFF; Cin = 1; #10;

        $finish;
    end
endmodule
```

Output



```
[2025-03-02 03:16:53 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:1: warning: timescale for tb_PPA_32 inherited from another file.
design.sv:1: ...: The inherited timescale is here.
A=00000000 B=00000000 Cin=0 | S=00000000 Cout=0
A=00000001 B=00000002 Cin=0 | S=00000003 Cout=0
A=ffffffff B=00000001 Cin=0 | S=00000000 Cout=1
A=12345678 B=9abcdef0 Cin=0 | S=acf13568 Cout=0
A=aaaaaaaa B=55555555 Cin=1 | S=00000000 Cout=1
A=ffffffff B=ffffffff Cin=1 | S=ffffffff Cout=1
testbench.sv:27: $finish called at 60000 (1ps)
Finding user-specified file...
```

Problem 4

a. Design a 16-bit KS Adder. Assume that the inputs are A[15:0], B[15:0], and Cin, andthe outputs are S[15:0], Cout. You can only use 2-input AND and OR gates to construct the carry propagation and carry generate logic. [Points : 75]

Design Code

```
`timescale 1ns / 1ps

module KS_Adder_16bit (
    input [15:0] A, B,
    input Cin,
    output [15:0] S,
    output Cout
);
    wire [15:0] P, G;
    wire [16:0] C;  // Expanded to store carry-out explicitly

    // Step 1: Compute initial propagate (P) and generate (G) signals
    assign P = A ^ B; // Propagate
    assign G = A & B; // Generate

    // Step 2: Compute carries using Kogge-Stone Tree

    wire [15:0] G1, P1;
    wire [15:0] G2, P2;
    wire [15:0] G3, P3;
    wire [15:0] G4, P4;

    // Stage 1
    assign G1[0]  = G[0];
    assign P1[0]  = P[0];

    genvar i;
    generate
       for (i = 1; i < 16; i = i + 1) begin
          assign G1[i] = G[i] | (P[i] & G[i-1]);
          assign P1[i] = P[i] & P[i-1];
       end
    endgenerate

    // Stage 2
    assign G2[0] = G1[0];
    assign G2[1] = G1[1];
    assign P2[0] = P1[0];
    assign P2[1] = P1[1];

    generate
       for (i = 2; i < 16; i = i + 1) begin
          assign G2[i] = G1[i] | (P1[i] & G1[i-2]);
          assign P2[i] = P1[i] & P1[i-2];
       end
    endgenerate

    // Stage 3
    assign G3[0] = G2[0];
    assign G3[1] = G2[1];
    assign G3[2] = G2[2];
    assign G3[3] = G2[3];
    assign P3[0] = P2[0];
    assign P3[1] = P2[1];
    assign P3[2] = P2[2];
    assign P3[3] = P2[3];

    generate
       for (i = 4; i < 16; i = i + 1) begin
          assign G3[i] = G2[i] | (P2[i] & G2[i-4]);
          assign P3[i] = P2[i] & P2[i-4];
       end
    endgenerate

    // Stage 4
    assign G4[0] = G3[0];
    assign G4[1] = G3[1];
    assign G4[2] = G3[2];
    assign G4[3] = G3[3];
    assign G4[4] = G3[4];
    assign G4[5] = G3[5];
    assign G4[6] = G3[6];
    assign G4[7] = G3[7];
    assign P4[0] = P3[0];
    assign P4[1] = P3[1];
    assign P4[2] = P3[2];
    assign P4[3] = P3[3];
    assign P4[4] = P3[4];
    assign P4[5] = P3[5];
    assign P4[6] = P3[6];
    assign P4[7] = P3[7];

    generate
       for (i = 8; i < 16; i = i + 1) begin
          assign G4[i] = G3[i] | (P3[i] & G3[i-8]);
          assign P4[i] = P3[i] & P3[i-8];
       end
    endgenerate


    assign C[0] = Cin;
    generate
       for (i = 1; i < 16; i = i + 1) begin
          assign C[i] = G4[i-1] | (P4[i-1] & C[i-1]);
       end
    endgenerate
```

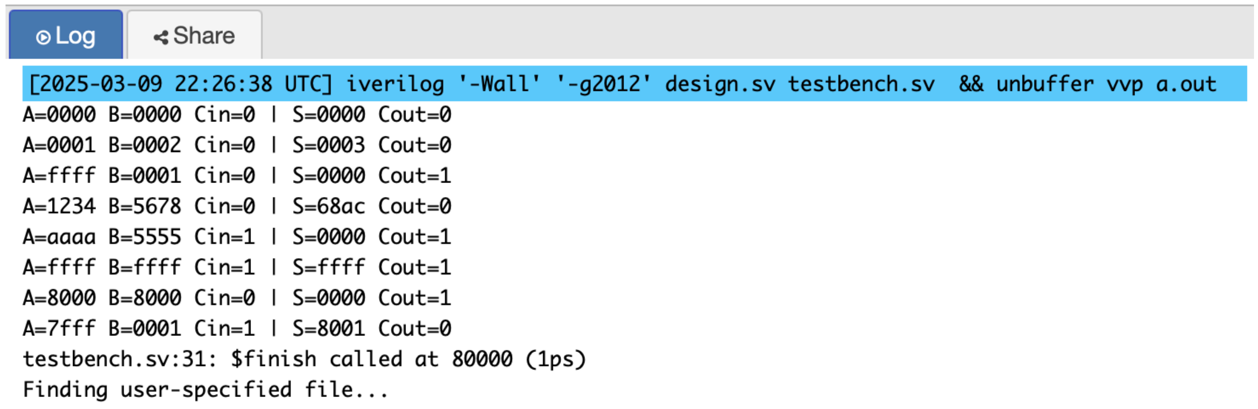
```
    assign C[16] = G4[15] | (P4[15] & C[15]);
    assign S = A ^ B ^ C[15:0];
    assign Cout = C[16];

endmodule
```

b. Enhance the CLA testbench of Problem 2(b) to test the 16-bit KS Adder. [Points : 25]

<u>Testbench Code</u>

```
`timescale 1ns / 1ps

module tb_KS_Adder;
    reg [15:0] A, B;
    reg Cin;
    wire [15:0] S;
    wire Cout;

    // Instantiate the 16-bit KS Adder
    KS_Adder_16bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .S(S),
        .Cout(Cout)
    );

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

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

        // Test cases
        A = 16'h0000; B = 16'h0000; Cin = 0; #10;
        A = 16'h0001; B = 16'h0002; Cin = 0; #10;
        A = 16'hFFFF; B = 16'h0001; Cin = 0; #10;
        A = 16'h1234; B = 16'h5678; Cin = 0; #10;
        A = 16'hAAAA; B = 16'h5555; Cin = 1; #10;
        A = 16'hFFFF; B = 16'hFFFF; Cin = 1; #10;
        A = 16'h8000; B = 16'h8000; Cin = 0; #10;
        A = 16'h7FFF; B = 16'h0001; Cin = 1; #10;
        $finish;
    end
endmodule
```

Output



Log
Share

```
[2025-03-09 22:26:38 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=0002 Cin=0 | S=0003 Cout=0
A=ffff B=0001 Cin=0 | S=0000 Cout=1
A=1234 B=5678 Cin=0 | S=68ac Cout=0
A=aaaa B=5555 Cin=1 | S=0000 Cout=1
A=ffff B=ffff Cin=1 | S=ffff Cout=1
A=8000 B=8000 Cin=0 | S=0000 Cout=1
A=7fff B=0001 Cin=1 | S=8001 Cout=0
testbench.sv:31: $finish called at 80000 (1ps)
Finding user-specified file...
```