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 made significant progress on the project. The following tasks have been completed:

- Completion of Problem 2 (Split Among Team Members):
    - **Qudsia Sultana:** Designed and implemented the 4-bit Carry Look Ahead Adder (CLA) module, ensuring correct propagate (P) and generate (G) logic. She verified the correctness of the carry computation and optimized the module for seamless integration into the 32-bit CLA.
    - **Shahed Alhanbali**: Focused on constructing the 32-bit Carry Look Ahead Adder (CLA) using multiple 4-bit CLA blocks. She worked on interconnecting the CLA modules correctly to ensure proper carry propagation across all 32 bits, avoiding delays associated with ripple carry.
    - **Cindy Jurado:** Developed the testbench for the 32-bit CLA, creating multiple test cases to verify addition operations for different values of A, B, and Cin. She ensured the design handled overflow correctly and validated the correct operation of carry-out (Cout).
- Initiation of Problem 3: The team has now begun working on Problem 3, which involves designing a 16-bit Parallel Prefix Adder (PPA) using 2-input AND and OR gates for carry propagation and carry generation. The design will follow the CLA module template provided in the problem statement.
  Next Steps:
    - Implement the 16-bit PPA using a structured approach to compute carries efficiently.
    - Modify and enhance the existing CLA testbench from Problem 2(b) to test the 16-bit PPA for correctness.
    - Plan how to extend this design to a 32-bit PPA as a future step.
- Challenges and Next Steps: The team is verifying the correctness of carry propagation and sum calculations for the PPA before finalizing the implementation.

2. Team Responsibilities

Team Member	Responsibilities	Effort
Qudsia Sultana	- Designed and implemented the 4-bit Carry Lookahead Adder (CLA). - Ensured correct propagate (P) and generate (G) logic for carry computation. - Verified CLA correctness and optimized its design for integration into the 32-bit CLA. - Assisted in debugging and ensuring correct carry propagation.	100%
Shahed Alhanbali	- Constructed the 32-bit Carry Look Ahead Adder (CLA) using 4-bit CLA blocks. - Verified carry propagation across all blocks and prevented ripple carry delays. - Ensured that the CLA structure met the design requirements and was correctly interconnected. - Maintained and updated the GitHub repository with the latest changes.	100%
Cindy Jurado	- Developed the testbench for the 32-bit CLA, writing multiple test cases to validate its functionality. - Verified sum and carry-out correctness under different conditions. - Debugged test failures and ensured that edge cases (e.g., overflow) were handled correctly. - Updated the progress report to reflect the team's advancements.	100%

Problem 2

a. Design a 32-bit Carry Lookahead Adder (CLA) using a block size of four (4). Assume that the inputs are A[31:0], B[31:0], and Cin, and outputs are S[31:0], Cout. Use the 4-bit full adder (RCA) that you designed in Problem 1 as the component. You can only use 2-input AND and OR gates to construct the carry propagation and carry generate logic. No need to include subtraction operation for the CLA. Use the CLA module template of Figure 2(a). [Points : 30]

Design Code:

```
`timescale 1ns / 1ps

// Part (a): 4-bit Carry Look Ahead Adder (CLA) - Qudsia
module four_bit_CLA(A, B, Cin, S, Cout);
   input [3:0] A, B;
   input Cin;
```

```
    output [3:0] S;
    output Cout;

    wire [3:0] P, G;
    wire [4:0] C;

    assign C[0] = Cin;

    // Generate and Propagate signals
    assign G = A & B; // Generate
    assign P = A ^ B; // Propagate

    // Carry computation
    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]);

    // Sum computation
    assign S = P ^ C[3:0];
    assign Cout = C[4];
endmodule

// Part (a): 32-bit Carry Lookahead Adder (CLA) - Shahed
module CLA(A, B, Cin, S, Cout);
    input [31:0] A, B;
    input Cin;
    output [31:0] S;
    output Cout;

    wire [7:0] carry;

    // Instantiate eight 4-bit CLAs
    four_bit_CLA CLA0 (A[3:0],   B[3:0],   Cin,      S[3:0],   carry[0]);
    four_bit_CLA CLA1 (A[7:4],   B[7:4],   carry[0], S[7:4],   carry[1]);
    four_bit_CLA CLA2 (A[11:8],  B[11:8],  carry[1], S[11:8],  carry[2]);
    four_bit_CLA CLA3 (A[15:12], B[15:12], carry[2], S[15:12], carry[3]);
    four_bit_CLA CLA4 (A[19:16], B[19:16], carry[3], S[19:16], carry[4]);
    four_bit_CLA CLA5 (A[23:20], B[23:20], carry[4], S[23:20], carry[5]);
    four_bit_CLA CLA6 (A[27:24], B[27:24], carry[5], S[27:24], carry[6]);
    four_bit_CLA CLA7 (A[31:28], B[31:28], carry[6], S[31:28], carry[7]);

    assign Cout = carry[7];
```

```
endmodule
```

b. Enhance the 4-bit RCA testbench of Problem 1(e) to test the 32-bit CLA (see the link at Testbench writing) above. [Points : 20]

Testbench Code:

```
`timescale 1ns / 1ps

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

  // Instantiate the 32-bit CLA module
  CLA uut (
    .A(A),
    .B(B),
    .Cin(Cin),
    .S(S),
    .Cout(Cout)
  );

  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'h87654321; Cin = 0; #10;
    A = 32'hAAAAAAAA; B = 32'h55555555; Cin = 1; #10;
    A = 32'hFFFFFFFF; B = 32'hFFFFFFFF; Cin = 1; #10;

    $finish;
  end
endmodule
```

Log Share

```
[2025-03-02 01:13:20 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
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=87654321 Cin=0 | S=99999999 Cout=0
A=aaaaaaaa B=55555555 Cin=1 | S=00000000 Cout=1
A=ffffffff B=ffffffff Cin=1 | S=ffffffff Cout=1
testbench.sv:30: $finish called at 60000 (1ps)
Finding user-specified file...
```