Deadline:

- **Deadlines: 11:59 PM Chicago time, on 27th February, 6th March, and 13th March, 2025 (Chicago Clock).**
- The submission link will be closed automatically after the deadline. **No submissions will be accepted after the deadline (including via email)**. Please have sufficient time before the deadline to scan your work and upload it to Blackboard.

Submission Instructions:

- Total Points: 150
- Optional Bonus Points: 100
- Work in a group of three (3) students. We require one submission per group at each checkpoint. Please see the submission details at the end of the document.
- Submit at "Submissions ⟶ Project Submission Links ⟶ Project 1".

Resource for Verilog Hardware Description Language: ASIC World is a concise resource for Verilog description language. You can find it here. In addition to the Verilog tutorial that Suresh provided and the Verilog primer that was discussed in the class, please look at the following modules before you embark into the project.

1. **Verilog in One Day**: Click Here.
2. **Verilog Operators**: Click Here.
3. **Behavioral Modeling**: Click Here.
4. **Structural and Gate-level Modeling**: Click Here.
5. **Testbench Writing**: Click Here.

1 Problem Sets

Problem 1 (**Designing a 1-bit/4-bit Full Adder/Subtractor).** Refer to Lecture Slide 2 for the following problems.

(a) Design a 1-bit full-adder in Verilog using ***behavioral 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]

(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]

(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]

```
module one_bit_full_adder(A, B, Cin, S,
    Cout);

input A, B, Cin;
output S, Cout;

// Your code

endmodule
```

(a) **Module definition of 1-bit full adder**.

```
module four_bit_RCA_RCS(A, B, Cin, S,
    Cout);

input [3:0] A, B;
input Cin;
output [3:0] S;
output Cout;

// Your code

endmodule
```

(b) **Module definition of 4-bit RCA/RCS**.

Figure 1: **Module definitions for Problem 1**.

(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]

(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]


```
module CLA(A, B, Cin, S, Cout);

input [31:0] A, B;
input Cin;
output [31:0] S;
output Cout;

// Your code

endmodule
```

(a) **Module definition of 32-bit CLA**.

```
module PPA(A, B, Cin, S, Cout);

input [15:0] A, B;
input Cin;
output [15:0] S;
output Cout;

// Your code

endmodule
```

(b) **Module definition of 16-bit PPA**.

Figure 2: **Module definitions for Problem 2 and Problem 3**.

Problem 2 (Designing a 32-bit CLA Adder/Subtractor). Refer to Lecture Slide 2 for the following problems.

(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]

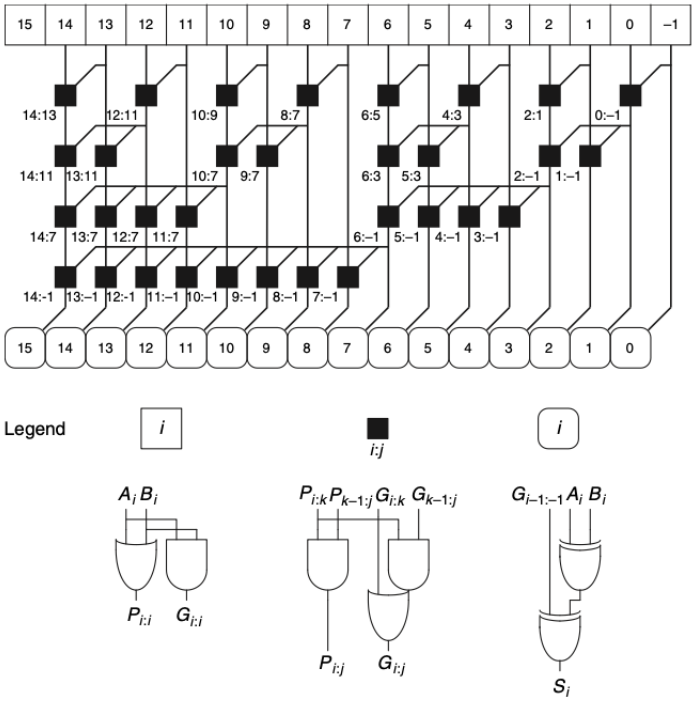

Figure 3: **A 16-bit Parallel Prefix Adder**.

(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]

Problem 3 (**Designing a 16-bit Prefix Adder (PPA)).** Refer to the textbook "Digital Design and Computer Architecture, By Harris and Harris, 2nd edition", Section 5.2.1 "Addition", Subsection "Prefix Adders." The textbook is available online and also for free at Digital Design and Computer Architecture. An example 16-bit Parallel Prefix Adder is given in Figure 3. Here $P_{-1} = 0$ and $G_{-1} = C_{in}$. You can extend this to make a 32-bit PPA.

(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]

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

The following problem is a bonus problem and optional. If you solve this, you will earn additional points.

Problem 4 (**Designing a 16-bit Kogge-Stone (KS) Adder).** Please refer to the research article titled "A Taxonomy of Parallel Prefix Networks" by David Harris. You can find it at Kogge-Stone (KS).

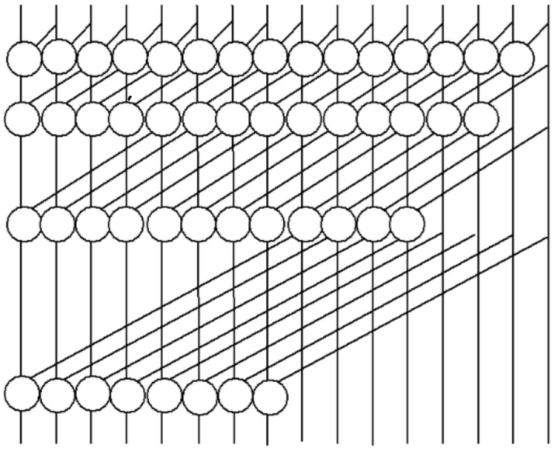

Figure 4: **The computation tree structure of a 16-bit Kogge-Stone (KS) Adder**.

You can also refer to Figure 4. This is similar to the PPA in Figure 3. You might have to change $P_{i:i}$ as $P_{i:i} = A_i \oplus B_i$.

(a) Design a 16-bit KS Adder. 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. [Points **:** 75]

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

2 Submission Deadlines, Expectations, and Artifacts

Create a GitHub repository for your group. The GitHub repo will be used for collaborative coding by your team members. Commit and push your codes as frequently as possible. The commit history may be used as an indication of the contribution of each of the team members. A nice GitHub cheat sheet is available here.

(A) **Progress Report 1**: Complete the progress report with the following details – (i) Summary of Project Status and (ii) Team Member Responsibilities. Submit a PDF file and include the GitHub link in the PDF file.

- **Deadline**: **02/27/2025 at 11:59 pm Chicago Time**.
- **Expectation**: Completion of Problem 1 and initiation of Problem 2.

(B) **Progress Report 2**: Complete the progress report with the following details – (i) Summary of Project Status and (ii) Team Member Responsibilities. Submit a PDF file and include the GitHub link in the PDF file.

- **Deadline**: **03/06/2025 at 11:59 pm Chicago Time**.

- **Expectation**: Completion of Problem 2 and initiation of Problem 3.

(C) **Final Report**: Complete the progress report with the following details – (i) Summary of Project Status, (ii) Team Member Responsibilities, (iii) Screenshots of Waveforms of each of the Problem (*i.e.*, the waveform for 1-bit full adder, 4-bit RCA, 32-bit CLA, etc.), and (iv) All codes as a tarball (How to tar?). Submit a PDF file and include the GitHub link in the PDF file. We may pull code from your GitHub, and please ensure that your submitted code and GitHub code match and are the latest versions.

- **Deadline**: **03/13/2025 at 11:59 pm Chicago Time**.
- **Expectation**: Completion of all mandatory problems and (optional) bonus problem.

NO EXTENSION OF ANY DEADLINE WILL BE GRANTED UNDER ANY CIRCUMSTANCES.