Lab 5: Ripple Carry and Carry Lookahead Adders

Foundations of Computer Systems Design Lab
CS2310
Due: September 10, 2024 4:45pm(inclass)

Instructions

1. Use only **structural description** for all the logic expressions you code in verilog.
2. There is no restriction on fan-in of the gates you use.
3. Name the files containing modules for a specific question as **q1_a.v,q1_b.v,...**
4. Read the questions carefully and implement the logic as mentioned.
5. **Module names should be same as mentioned in each question.**
6. Test benches for all the questions are provided.

1 Ripple Carry Adder

Q1 a

Design a 4-bit unsigned Ripple carry adder (RCA) using the module name as

RCA4 (input A[3:0],input B[3:0],input Cin, output Cout, output [3:0] sum);

(Hint: Try to use a half adder to implement a full adder.Use this designed full Adder module in designing RCA4)

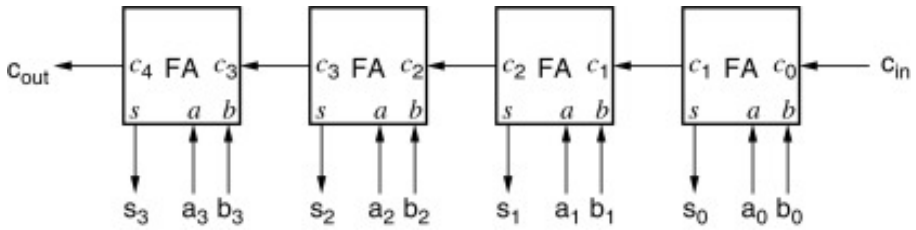

Figure 1: Ripple Carry Adder

Q1 b (bonus)

Implement 16-bit unsigned Ripple Carry Adder using 4-bit RCA designed in the above subquestion and the module name as

RCA16 (input A[15:0],input B[15:0],input Cin, output Cout, output [15:0] sum);

2 Carry Look-ahead Adder

Q2 a

Design a 4-bit unsigned CLA using the module name as

CLA4 (input A[3:0],input B[3:0],input Cin, output Cout, output [3:0] sum);

Note: The CLA should correctly compute the sum and carry bits in an optimized manner, with a focus on minimizing the delay introduced by carry propagation.

Q2 b (bonus)

Implement 16-bit unsigned Carry Lookahead Adder using 4-bit CLA designed in the above subquestion and the module name as

CLA16 (input A[15:0],input B[15:0],input Cin, output Cout, output [15:0] sum)

3 Addition of 4 4-bit unsigned integers in order

You are given four unsigned 4-bit integers: X_1, X_2, X_3, and X_4. Your task is to add them together in the following manner:

1. Start by adding X_1 and X_2.
2. If there is no overflow (i.e., the sum is 15 or less), proceed to add X_3.
3. If there is still no overflow, add X_4.

Your goal is to Output:

- The **final 4-bit sum** of the integers until it is valid .
- A **2-bit indicator** to indicate how many additions (X_1, X_2, X_3, X_4) you were able to perform without causing any overflow:
  - `00`: If you cannot add any integers without overflow, (sum = 0000).
  - `01`: If you can add X_1 and X_2 without overflow.(sum = $X_1 + X_2$)
  - `10`: If you can add X_1, X_2, and X_3 without overflow (sum = $X_1 + X_2 + X_3$).
  - `11`: If you can add all four integers: X_1, X_2, X_3, and X_4 without overflow (sum = $X_1 + X_2 + X_3 + X_4$).

Implement the above using module name as

Adder4 (input [3:0] x1,x2,x3,x4, output [3:0] sum, output [1:0] count);

Hint: Think of using **priority encoder** and **MUX** that you have designed in previous labs.