Recitation 2

Ripple Carry Adder

# Introduction

In this recitation, you will learn to build a 4-bit Ripple Carry Adder (RCA) in Verilog and test your adder using waveforms and testbenches.

# Collaboration Policy

You will be working in groups of 2 or 3. Groups are allowed to collaborate.

# Equipment

* Computer with Quartus Prime software

# Tasks

To receive credit for this recitation, you must complete:

* Task 1: Building a full adder and then using the validated full adder to build a 4-bit RCA
* Task 2: Using a vector waveform to validate the operation of your circuit
* Task 3: Writing a testbench to validate the operation of your circuit

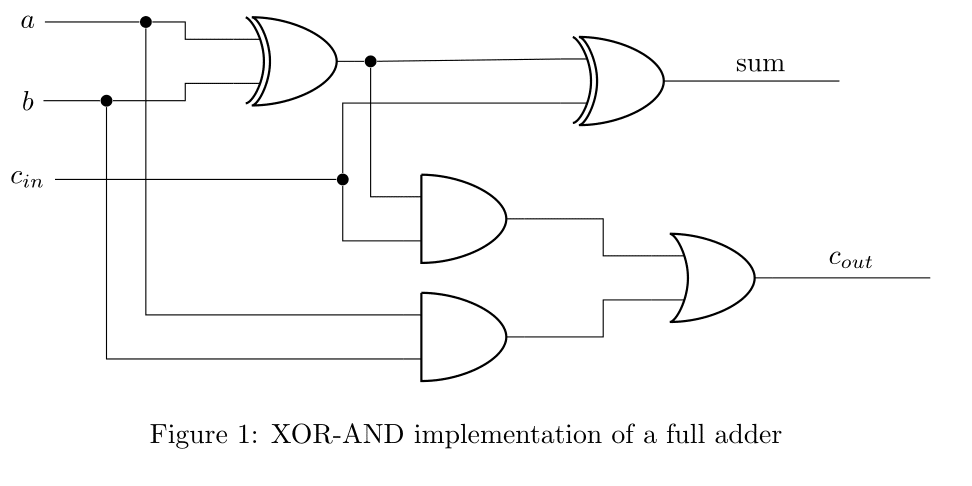
As you complete tasks, a TA must sign-off on the completion of each task. Ensure that the TA marks the completion of the tasks in Sakai. This recitation acts as a reference for writing and testing Verilog code. You do not need to finish all of the tasks by the end of recitation but you may not leave until either the end of the recitation period or you complete all the provided tasks. If you do not finish and would like to continue working on the tasks outside of recitation, you may go to office hours to receive assistance from the office hours TAs.

# Grading

* Completing Recitation Tasks: 1 point (pass/fail)

Build a Full Adder

Just like the decimal addition process you are used to, a Ripple Carry Adder (RCA) does binary additions bit by bit. To build a 4-bit RCA, you are going to build a 1-bit Full Adder (FA) first. An FA takes in three inputs: the two bits we are adding, and, as well as a carry-in, . The outputs are the sum for this specific bit, and a carry-out, . A circuit diagram for the FA is provided below. Build an FA in Verilog then use the Netlist Viewer and Waveform tools to validate it (you do not need to show this part to your TAs). A rule of thumb is to build and test your components incrementally, so please make sure your FA works before using it to build your RCA. This will also help you avoid having to debug and waste significantly more time later.



Build and Test a 4-Bit Ripple Carry Adder

Now you have successfully built and tested a 1-bit FA. Concatenate 4 FAs to build a 4-bit RCA. You have seen this during the lecture. For this exercise, keep the size of output sum as four bits, but note that an overflow could happen.

Test your 4-bit RCA using both waveforms and testbenches and show the results to a TA to be checked off. You have to include at least **2** test cases in both the waveform and testbench.

Refer to the document in Sakai→Resources if you want to learn more about testbenches.

The testbench file you need for this task is:

// set the timescale

`timescale 1 ns / 100 ps

module RCA\_tb(); // testbenches take no arguments

// set up inputs of NAND gate as registers so they can be manipulated at will

reg [3:0]in1;

reg [3:0]in2;

reg cin;

// clocks are useful for testbenches because they allow you to check your

// circuit at regular intervals large enough for signals to propagate

reg clock;

// set up output of NAND gate as wire

wire cout;

wire [3:0] sum;

// prepare any other variables you want - nothing is off-limits in a TB

integer num\_errors;

// instantiate the RCA

RCA test\_RCA (in1, in2, cin,cout,sum);

// begin simulation

initial begin

$display($time, " simulation start");

clock = 1'b0;

@(negedge clock);

in1 = 4'b1101;

in2 = 4'b1000;

cin = 1'b1;

@(negedge clock); // wait for the clock to go negative

in1 = 4'b1010;

in2 = 4'b1001;

cin = 1'b0;

@(negedge clock);

$stop;

end

always

#10 clock = ~clock; // toggle clock every 10 timescale units

endmodule

Feel free to modify the test cases in the above code and ask your TA how they verify your RCA (you can look at the ‘wave’ section after a successful RTL simulation, or you can insert ‘$display’ instructions). Note that it would be more beneficial for you to get used to writing testbench files on your own, so try not to copy-paste the code, but get used to typing it instead. You can name your testbench file ‘RCA\_tb.v’, and remember to follow similar steps as you did in Recitation 1 for setting up your testbench file.