System Verilog code

Step 1: Create a combinational RTL model for a 4-bit expandable carry lookahead adder (CLA). **You do not need to create or use gate-level modules for full or half adders.**

A combinational RTL model uses only continuous assignment (assign) statements or `always_comb` procedural block with or without gate delays.

Use these gate delays in your model:

- AND gate 2ns
- OR gate 2ns
- XOR gate 3ns

Step 2: Simulate your 4-bit CLA module with the testbench we provided. Save a transcript of your successful simulation results.

Step 3: Create an expandable 8-bit adder using the 4-bit CLA module you created and verified.

Step 4: Simulate your 8-bit CLA adder. Save a transcript of your successful simulation results.

Your design modules MUST be declared as follows to be instantiated correctly in the testbench:

```
module CLA4Bit(ain, bin, cin, sum, cout);
     input [3:0]    ain, bin;
     input          cin;
     output logic [3:0] sum;
     output logic cout;

module CLA8Bit(ain, bin, cin, sum, cout);
     input [7:0]    ain, bin;
     input          cin;
     output logic [7:0] sum;
     output logic cout;
```

Refer to the schematic below for a 4-bit CLA. FA is a full adder without the carry-out logic:

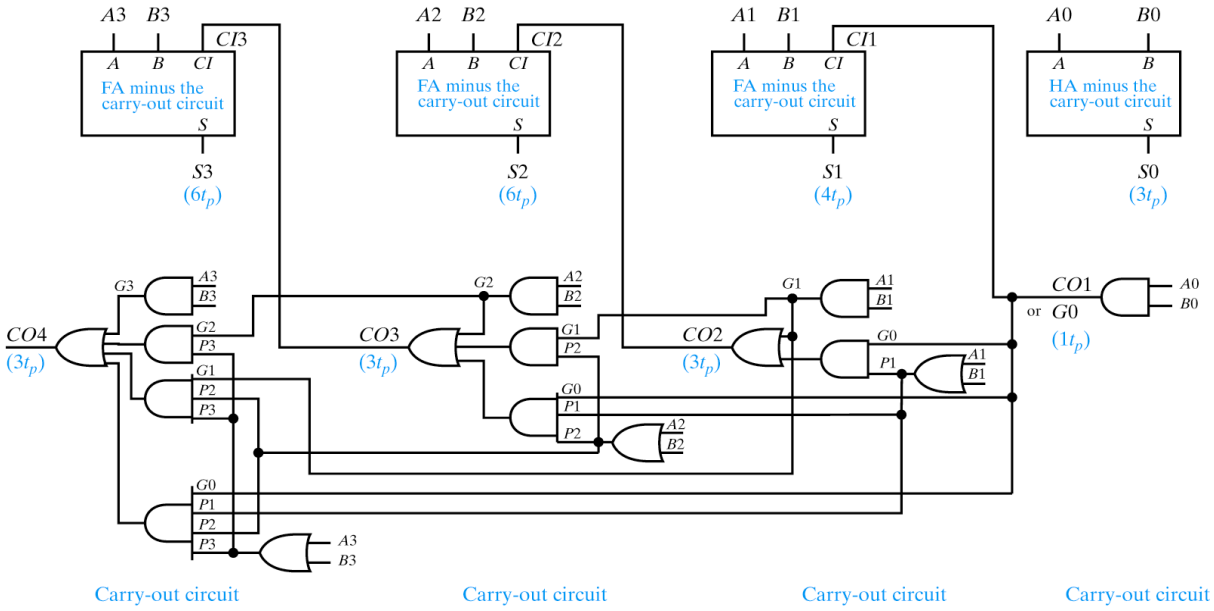

Figure 4.5.9 A 4-bit Carry Look-Ahead Adder showing the carry-out circuitry.

Refer below equations for the carry out logic:

Cout = A·B + A·Cin + B·Cin =

Cout = A·B + Cin · (A+ B) =

Cout = G + Cin · P

where,
Gn = An.Bn and Pn = An + Bn

G (carry generate) is dependent upon current stage's ability to generate a carry.

P (carry propagate) is dependent upon current (and prior) stages' ability to propagate a carry.

A carry is generated at a stage i if it is generated by stage i's inputs (Ai, Bi) or by any prior stage and propagated by every succeeding stage.

Hence,

CO1 = G0 + CI0.P0
=> CO1 = G0

CO2 = G1 + CI1.P1
=> CO2 = G1 + CO1.P1
=> CO2 = G1 + G0.P1

CO3 = G2 + CI2.P2
=> CO3 = G2 + CO2.P2
=> CO3 = G2 +(G1 + G0.P1).P2
=> CO3 = G2 +G1.P2 + G0.P1.P2

And so on...

And the equation for Sum output will be as follows:

S = A ^ B ^ CIN;

(Hint: For the sum output, you need to figure out the logic for CIN using the schematic, it is not straight forward, unlike inputs A and B which can be used as is!)

How to use the test bench provided?

Part 1: Testing your 4 bit CLA design module.

1. Create a top module as shown below and add this module to your CLA4Bit module and save this single file as CLA4Top.sv:

```
module CLA4Top;
     parameter nBITS = 4;
     logic [nBITS - 1 : 0] ain, bin, sum;
     logic cin, cout;

     // instantiate your 4 bit CLA design module here

     // instantiate the test bench module as follows
     test #(4) TB(.*);
endmodule
```

2. Add CLA4Top.sv and test.sv to your project in QuestaSim.
3. Simulate the CLA4Top.sv file
4. Save the transcript.

Part 2: Testing your 8 bit CLA design module.

1. Create a top module as shown below and add this module to your CLA8Bit module and save this single file as CLA8Top.sv:

```
module CLA8Top;
     parameter nBITS = 8;
     logic [nBITS - 1 : 0] ain, bin, sum;
     logic cin, cout;

     // instantiate your 8 bit CLA design module here

     // instantiate the test bench module as follows
     test #(8) TB(.*);
endmodule
```

2. Add CLA8Top.sv and test.sv to your project in QuestaSim.

3. Simulate the CLA8Top.sv file

4. Save the transcript.

You are encouraged to do further readings on the working of a CLA and ask questions if things are unclear.

Submit your System Verilog code for both design modules with the transcript via D2L in a zipped file named as **<yourname>_HW1.zip** with names as follows:

- ***CLA4Top.sv your 4-bit CLA design module***
- ***CLA8Top.sv your 8-bit CLA design module***
- ***transcripts for both simulations***

NOTE: If you do not follow these instructions for declaring, naming and submitting your modules to D2L, you will have points deducted even if your answers are correct.