

San Jose State University
Department of Computer Engineering

CMPE 125 Lab Report

Lab 4 Report

Title Hierarchical Design of a CLA Adder

Semester FALL 2019

Date 9/30/19

by

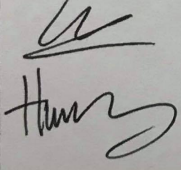
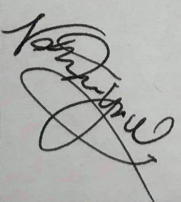
Name HUNG NGUYEN
(typed)

SID 012392081
(typed)

Name PHAT LE
(typed)

SID 012067666
(typed)

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1				75%	

Introduction

The purpose of this lab is to get familiar with the structured and hierarchical design methodology. Moreover, this lab also help familiar with the design, verification, validation flow and EDA tools.

Design Methodology

The first task of the lab is to design and verify an unsigned 4-bit integer adder via “inferred”
Beside that, the adder should be inferred by operation “+” in Verilog. Moreover , Complete the experiment with self design “ self_checking” test bench to verify the “adder”.

Table 1 : List of module “ Inferred adder”

Module	Function
inferred_adder.v	This module adding 2 4 bit number, but need 5-bit holder to store the result.we only take 4-bit sum and the 5th bit will be a carry out.
inferred_adder_tb.v	This is a self design “self_checking” test bench. This module is designed to test all the possibility of “ inferred_adder.v ”

The second task of the lab is to design and verify 4-bit integer adder using the “
Carry_look_ahead” technique . Unlike task 1 , we will design the 4-bit integer with half-adder ,CLA generator and XOR-gate which can reflect hierarchical design methodology that given in lecture.Like task 1, Complete the experiment with self design “ self_checking” test bench to verify the “adder”. At the end , Validate the 4-bit CLA with Basys3 FPGA board .

Table 2 : List of module “ CLA adder”

Module	Function
CLA_adder_top.v	This module take 5-bit holder to store the input which are 4-bit input and 1 bit carry_in . Same with the input. Output also need 4-bit sum and 1 bit carry out.
half_adder.v	This module adding 2 4-bit input and give 4-bit output , 1 carry_out.However, The output should go through CLA generator as input.
XOR2.v	This module will consume output of CLA generator as input and give the sum as output.

CLA_adder_tb.v	This module will test all the possibility of CLA_adder_top.v
-----------------------	--

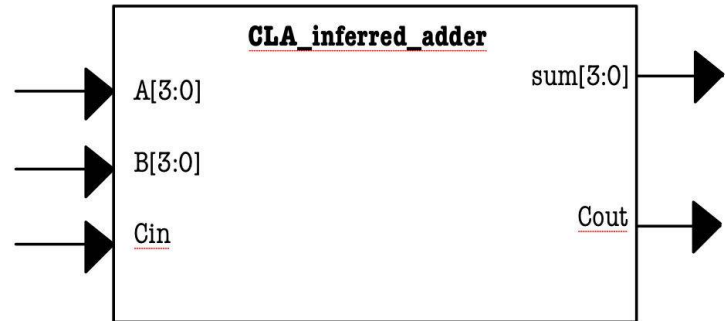


Figure 3: Block diagram for CLA_inferred_adder module

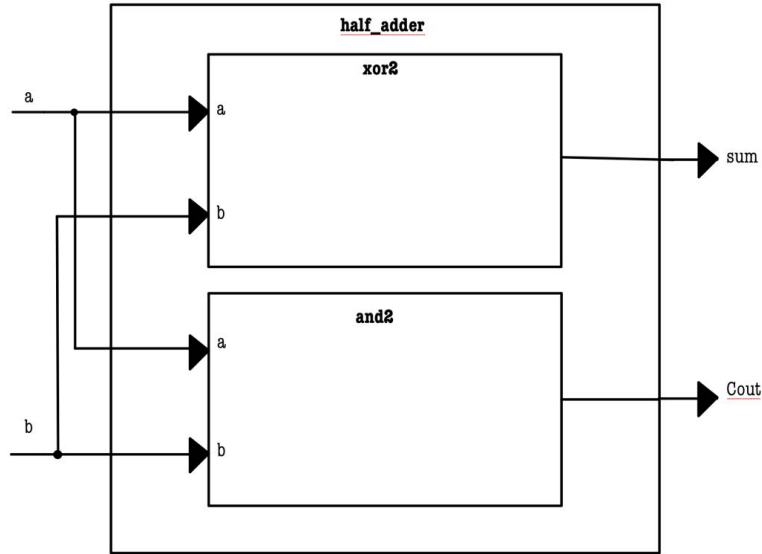


Figure 4: Block diagram for half_adder module

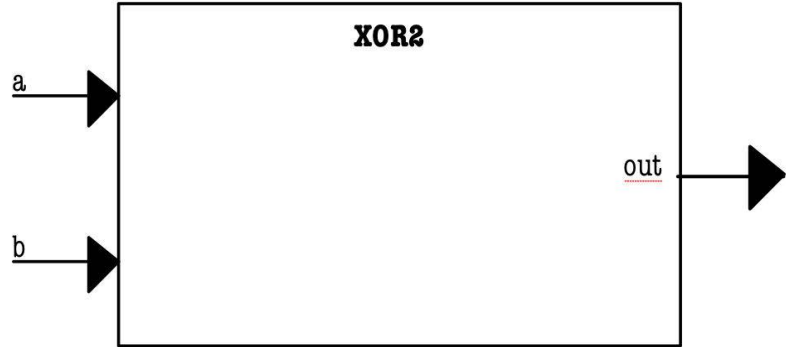


Figure 5:Block diagram for XOR2 module

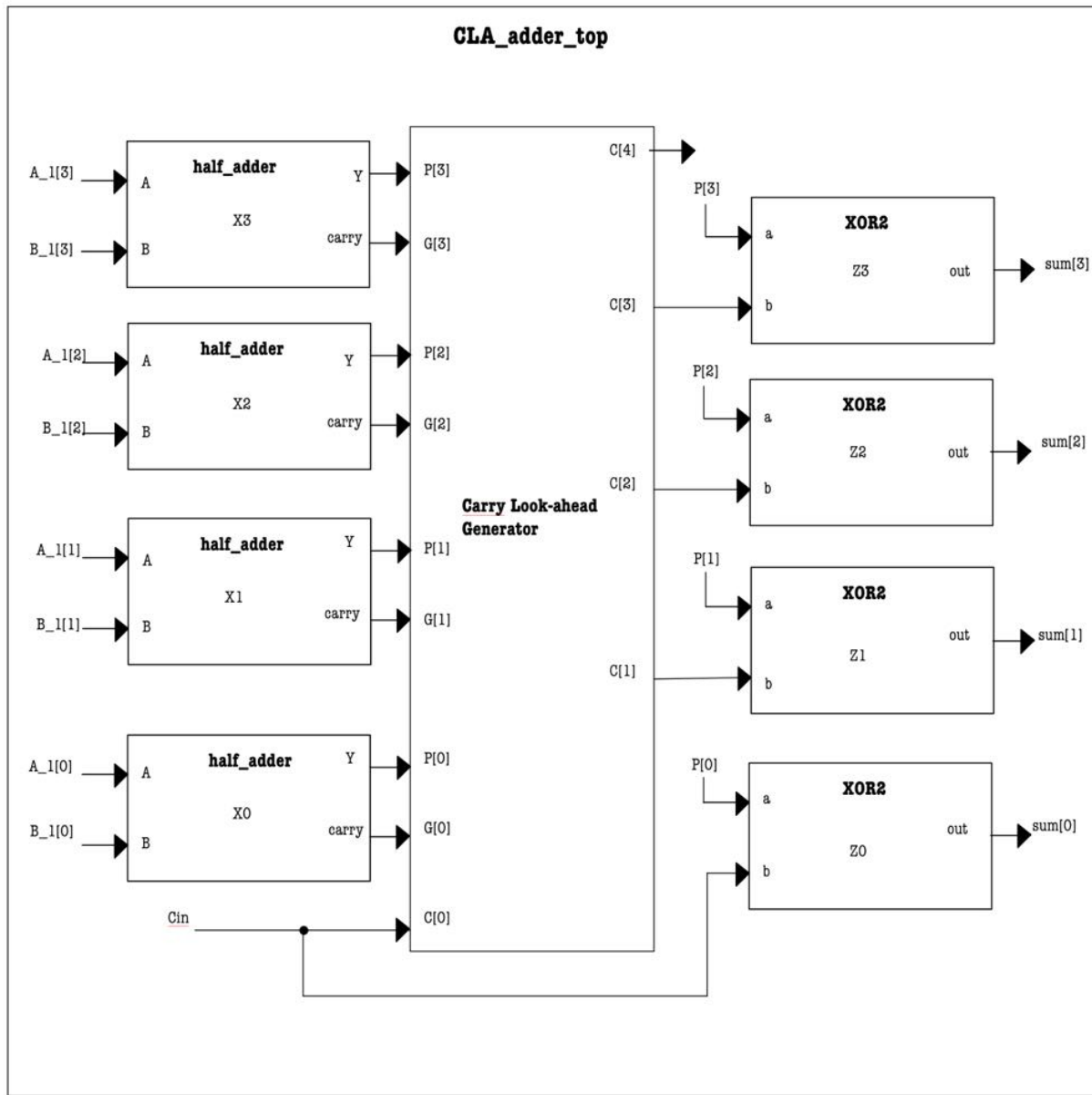


Figure 6: 4 bit right shifter/rota

Simulation Result

Task 1

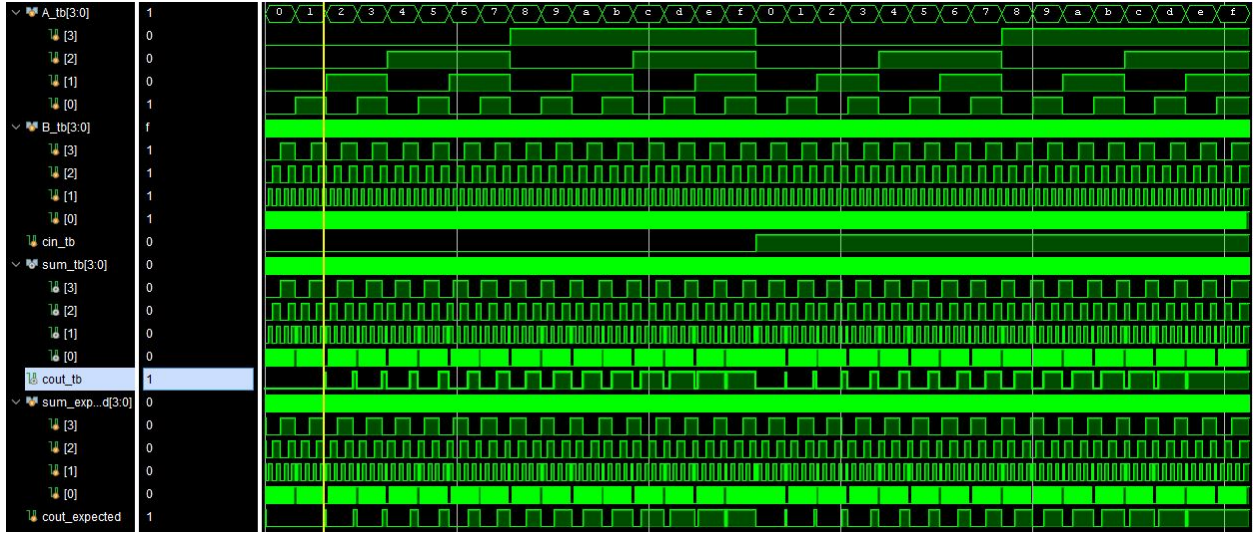


Figure 7: Simulation waveforms produced from Task 1

For this task, we try to test all possible cases. According to Figure 7 above, the cursor is at the input A(A_tb) of 0001, input B(B_tb) is 1111, carry in (Cin_tb) is 0. We can observe that the output(sum_tb and cout_tb) matches the expected output (sum_expected and cout_expected). Specifically, the sum should be 0 because $1111 + 0001 = 10000$. Since this is a 4-bit output, the output is 0000 and carry the fifth bit to Cout. Furthermore, we randomly check other cases and the result is as expected. For this reason, the simulation process for Task 1 is successfully built.

Task 2

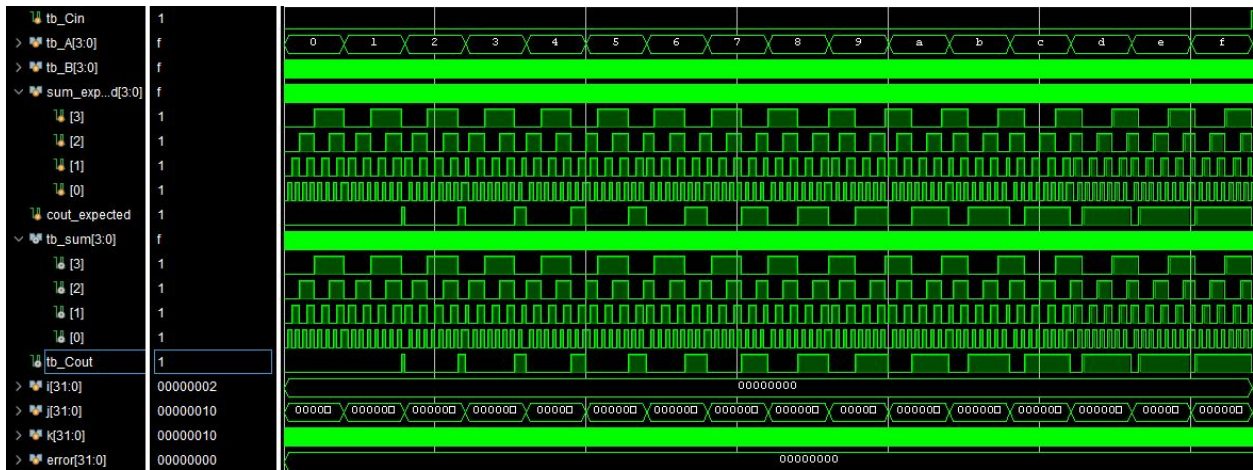


Figure 8: Simulation waveforms produced from Task 2 when Cin is 0

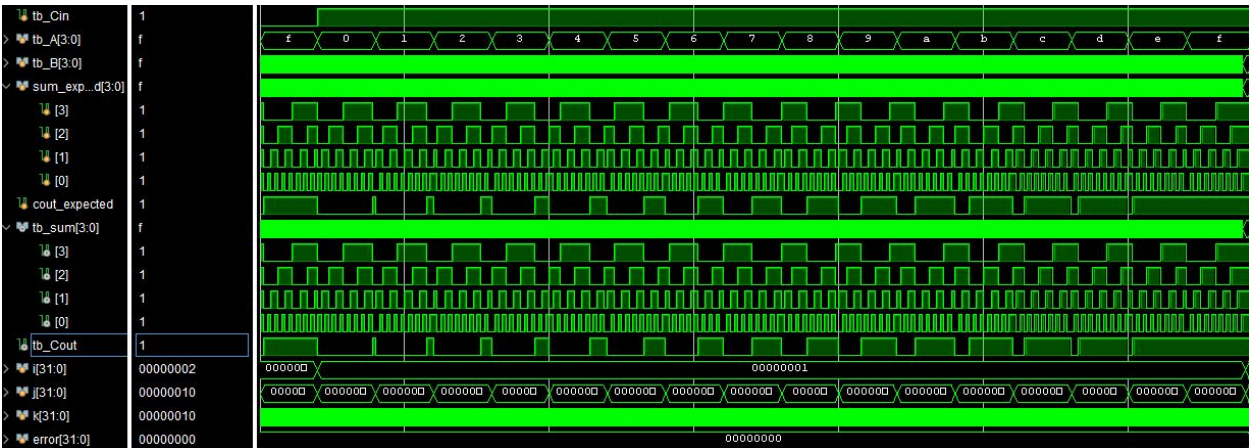


Figure 9: Simulation waveforms produced from Task 2 when Cin is 0

Because Cin cannot be observed in one figure, we break the result into 2 cases when Cin is 1 (Figure 9) and when it is 0 (Figure 8). Similar to Task 1, for this Task, it is easy to see that the testbench check every possible input and shows that the actual output matches the output produced from testbench file. Therefore, the simulation process has been succeeded.

FPGA Validation

The FPGA validation process is Task 3 for the lab. As shown in Figure 10 below, it is asked to use 4 switches on the left as input A, 4 switches on the right as input B, a push button as carry in signal. Furthermore, The board need to output LED above the input switches to display the chosen current input signal. For the carry in signal, it need to display on the 7 segment LED. Lastly, display another 2 of the 7 segment LED as the sum of A, B, and carryin inputs.

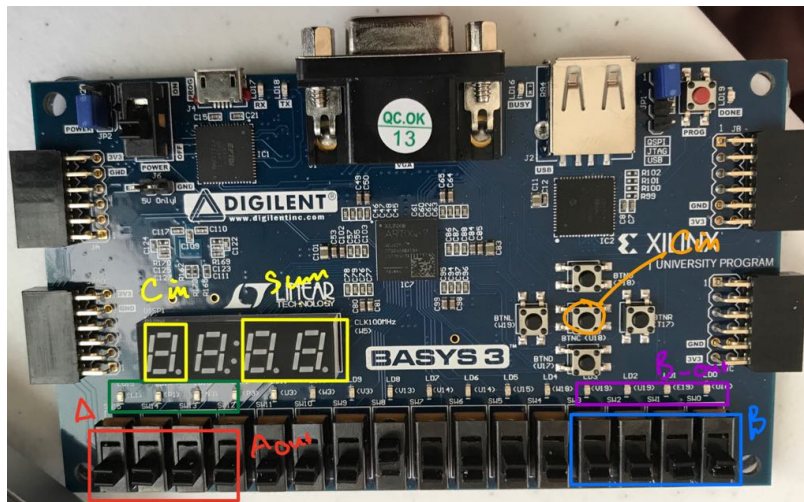


Figure 10: inputs and outputs layout for the FPGA validation process.

Unfortunately, this task is successfully performed because we were not able to convert the binary signals into BCD and then 7 segments signals.

Conclusion

Overall, the simulation task is succeeded because all the outputs produced by testbench files (sum and carry out) are having the exact waveforms as the outputs produced by the design code files. We also learn how to concatenate the signal to assign as well as compare the signals in the testbench process. On the other hand, we failed the FPGA validation because we cannot design a code to convert the sum (binary signal) to the two's 7 segments LED signals for the sum. Having a hint that we can replace the Voting Rule module from previous lab to use the 7 segments, our group were not able to produce the expected output. Other than that, we can produce the LED inputs of A and B as well as carry in signal but were not able to have a chance to demo it. Moreover, we realized that we did a bad job for this lab because we cannot fully understand the concept of 7 segments LED which is very important to use in the upcoming labs.

Appendix

a. Source Code

Task 1

inferred_adder.v	
<pre>module CLA_inferred_adder(input[3:0] A, B, input Cin, output reg[3:0] Y, output reg Cout); integer temp; always@(*) begin temp = A + B + Cin; assign Y = {temp[3:0]}; assign Cout = {temp[4]}; end endmodule</pre>	

inferred_adder_tb.v

```
module CLA_inferred_adder_tb;
    reg [3:0] A_tb, B_tb;
    reg cin_tb;
    wire [3:0] sum_tb;
    wire cout_tb;
    reg [3:0] sum_expected;
    reg cout_expected;
    CLA_inferred_adder DUT(.A(A_tb), .B(B_tb), .Cin(cin_tb), .Y(sum_tb),
    .Cout(cout_tb));

    integer i, j, k;
    initial
    begin

        for (i=0; i<2; i=i+1)
        begin
            cin_tb = i;
            for (j=0; j<16; j=j+1)
            begin
                A_tb = j;
                for (k=0; k<16; k=k+1)
                begin
                    B_tb = k;
                    #5
                    assign {cout_expected, sum_expected} = A_tb + B_tb + cin_tb;
                    if (sum_tb != sum_expected)
                    begin
                        $display("Failed when A=%b, B=%b, Cin=%b, the
expected sum is %d, but the actual value is %d",
                        A_tb, B_tb, cin_tb, sum_expected, sum_tb);
                    end
                    if (cout_tb != cout_expected)
                    begin
                        $display("Failed when A=%b, B=%b, Cin=%b, the expected
carry out is %d, but the actual value is %d",
                        A_tb, B_tb, cin_tb, cout_expected, cout_tb);
                    end
                    #5;
                end
            end
        end
        #10 $finish;
    end
endmodule
```


Task 2:

CLA_adder_top.v

```
module CLA_adder_top(
    input [3:0]      A_1, B_1,
    input            Cin,
    output [3:0]     sum,
    output           Cout
);

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

    half_adder X0(.A(A_1[0]), .B(B_1[0]), .Y(P[0]), .carry(G[0]));
    half_adder X1(.A(A_1[1]), .B(B_1[1]), .Y(P[1]), .carry(G[1]));
    half_adder X2(.A(A_1[2]), .B(B_1[2]), .Y(P[2]), .carry(G[2]));
    half_adder X3(.A(A_1[3]), .B(B_1[3]), .Y(P[3]), .carry(G[3]));

    assign C[0] = Cin;
    assign C[1] = G[0] | (P[0]&Cin);
    assign C[2] = G[1] | (P[1]&G[0]) | (P[1]&P[0]&C[0]);
    assign C[3] = G[2] | (P[2]&G[1]) | (P[2]&P[1]&G[0]) | (P[2]&P[1]&P[0]&C[0]);
    assign      C[4] = G[3] | (P[3]&G[2]) | (P[3]&P[2]&G[1]) | (P[3]&P[2]&P[1]&G[0]) | (P[3]&P[2]&P[1]&P[0]&C[0]);

    assign Cout = C[4];
    XOR2 Z3(.a(C[3]), .b(P[3]), .out(sum[3]));
    XOR2 Z2(.a(C[2]), .b(P[2]), .out(sum[2]));
    XOR2 Z1(.a(C[1]), .b(P[1]), .out(sum[1]));
    XOR2 Z0(.a(C[0]), .b(P[0]), .out(sum[0]));
endmodule
```

half_adder.v

```
module half_adder(
    input  A,B,
    output Y,carry);

    assign Y = A^B;
    assign carry = (A&B);
endmodule
```

XOR2.v

```
module XOR2(input a,
            input b,
            output out
);
    assign out=a^b;
endmodule
```

CLA_adder_tb.v

```
module CLA_adder_tb;
    reg tb_Cin;
    reg[3:0] tb_A;
    reg[3:0] tb_B;
    wire[3:0] tb_sum;
    wire tb_Cout;
    CLA_adder_top DUT(.Cin(tb_Cin), .A_1(tb_A), .B_1(tb_B), .sum(tb_sum), .Cout(tb_Cout));

    integer i;
    integer j;
    integer k;
    integer error;
    reg sum_expected;
    reg cout_expected;

    Initial begin
        error = 0;
        for (i=0; i<2; i=i+1)
            begin
                tb_Cin = i;
                for (j=0; j<16; j=j+1)
                    begin
                        tb_A = j;
                        for (k=0; k<16; k=k+1)
                            begin
                                tb_B = k;
                                {cout_expected,sum_expected} = tb_A + tb_B + tb_Cin;
                                if ({cout_expected, sum_expected} != (tb_A + tb_B + tb_Cin))
                                    begin
                                        $display("error");
                                        error = error + 1;
                                    end
                            end
                        #5;
                    end
            end
        end

        if(!error) $display("Test finished with %0d errors", error);
        #10 $finish;
    end
endmodule
```

```
end  
endmodule
```