Indian Institute of Technology (IIT-Bombay)

AUTUMN Semester, 2024
COMPUTER SCIENCE AND ENGINEERING

CS231: Digital Logic Design + Computer Architecture Lab

Lab 2

Full Marks: 100

Time allowed: (3 + ϵ) hours

Students who do not follow the naming convention will be awarded no marks.

1. In this assignment, you need to create a Ripple-Carry-Adder (RCA) following the instructions given below:
    1. First create a module called half_adder. You are only allowed to use basic gates and *assign* statements.
       Module name = "half_adder"
       input a, b : the numbers to be added
       output S, cout : S is the sum and cout is the carry bit.
    2. Next, create a module called full_adder by instantiating the half_adder module and maybe a few basic gates. Once again, you are only allowed to use basic gates and *assign* statements.
       Module name = "full_adder"
       input a, b : the numbers to be added
       input cin : the input carry bit
       output S, cout : S is the sum and cout is the output carry bit.
    3. Create your RCA using your full_adder module. The length of the adder will be decided by a parameter N, which you need to define. Also, you need to use *generate* statement to construct your adder. An example of how to do parameterized design and use *generate* statements is given in Figure. 1. The module name should be rca_Nbit.
    4. Write a proper testbench tb_rca_32bit.v and test the 32-bit version of your adder.

2. This question asks you to create a Carry-Lookahead-Adder (CLA). The goal is to design a 4-bit CLA and use that to generate a 16-bit CLA adder in a hierarchical manner.
    1. First design the 4-bit CLA module named CLA_4bit. Clearly specify the logic equation of the lookahead carry signals (C_1, C_2, C_3, C_4) in terms of generate (G_0, G_1, G_2, G_3) and propagate $(P_0, P_1, P_2, P_3$ signals.
    2. Now, use these 4-bit CLAs to generate a 16-bit CLA (module name : CLA_16bit). Note that **you are not supposed to cascade the blocks as discussed in the class**. Instead, we follow a different design style. Here, you need to generate the carry-in for each 4-bit CLA block using a lookahead carry unit (a module named lookahead_carry_unit_16_bit). For that, first modify your 4-bit CLA module to generate *block propagate* and *block generate* signals (P, G) (module name : CLA_4bit_P_G). The equations for block propagate and generate signals are as follows:

$$P = P_0P_1P_2P_3, G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

•Paramaterized module example

```
module vector_and(z, a, b);
    parameter cardinality = 1;
    input [cardinality-1:0] a, b;
    output [cardinality-1:0] z;
    wire [cardinality-1:0] z = a & b;
  endmodule
```

•We override these parameters when we instantiate the module as:

```
module Four_and_gates(OutBus, InBusA, InBusB);
    input [3:0] InBusA, InBusB; output[3:0] OutBus;
    Vector_And #(4) My_And(OutBus, InBusA, InBusB);
  endmodule
```

• Generate Example:

```
module genvar_example #(parameter N = 32) (a, b, out);
input [N-1:0] a;
input [N-1:0] b;
output [N-1:0] out;


generate
    genvar i;
    for (i = 0; i < N; i = i + 1) begin
           and  andi (.out(out[i]), .a(a[i]), .b(b[I]));
    end
endgenerate


endmodule
```

Figure 1: Example of Parameterized Modules and Generate Statements

. Observe that, we can also write $C_4 = G + PC_0$. This is supposed to be an output of your lookahead carry unit. Similarly, we can calculate C_8 and C_12, and finally, the C_16, all from block propagate and block generate signals for each 4-bit CLA. A diagram is given in Figure 2:

3. Finally, design a 32-bit CLA (module name : CLA_32bit) by extending the above mentioned philosophy. You will need to create a module for 32 bit lookahead carry called lookahead_carry_unit_32_bit Maintain the naming convention of the modules.

3. In this question, you are asked to design a combinational multiplier (of 16 bit numbers) using Karatsuba's algorithm. Follow the Wikipedia document provided to understand the algorithm.

   1. Your design has to be fully hierarchical and structural, i.e., you cannot use anything more than module instantiation and *assign* statements.
   2. You cannot use any multiplier or adder/subtractor unless it is structurally designed.
   3. If you need to use any multiplier (designed by you structurally), it should only be multiplying numbers whose bit-width is a power of 2. No marks will be given if you are using anything else.
   4. Submit the design module and testbench in separate files. Strictly follow the naming convention.
   5. Module name : karatsuba_16, input parameters X[15:0], Y[15:0] and output Z[31:0] = X×Y

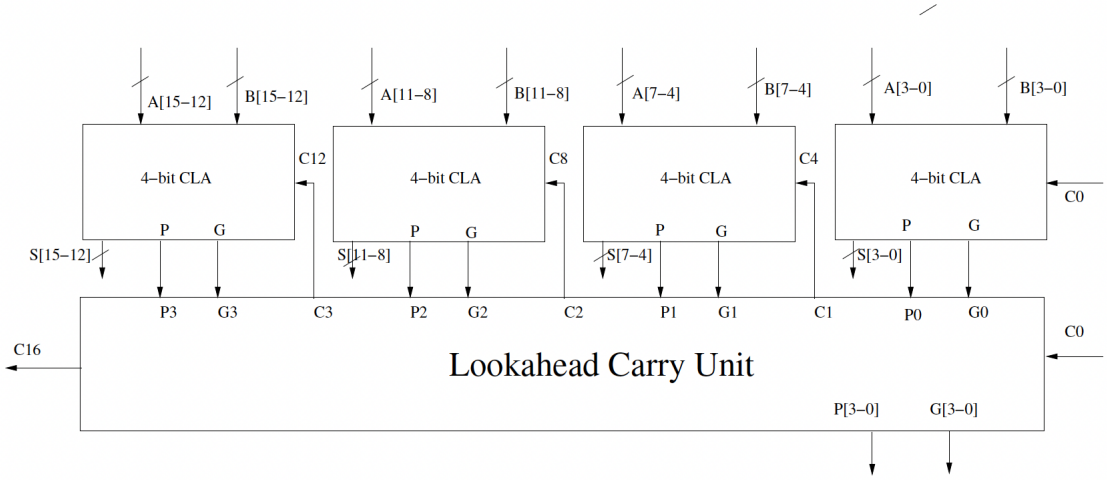

Figure 2: 16-bit hierarchical CLA

```
<rollnumber_lab2_q1q2>/
 |____rca_Nbit.v
 |____tb_rca_32bit.v
 |____cla_Nbit.v
 |____tb_cla_32bit.v

 <rollnumber_lab2_q3>/
 |____combinational_karatsuba.v
 |____tb_combinational_karatsuba.v
```

Figure 3: Submission Format