EECS 151/251A Homework 2

Due Monday, Feb 5th, 2024

Introduction

You will be asked to write several Verilog modules as part of this HW assignment. You will need to test your modules by running them through a simulator. We recommend the following free, online Verilog simulator: `https://www.edaplayground.com`.

Problem 1: Full Adder Truth Table

In lecture, we discussed a full adder. This circuit takes three bits as inputs (an addend bit, a second addend bit, and a carry-in bit) and outputs two bits (the sum bit, and the carry-out bit). This circuit is closely related to a half adder which takes two bits as inputs (an addend bit and a second addend bit) and outputs two bits (a single sum bit and carry bit).

(a) Draw a simple diagram demonstrating how to construct a full adder from half adders. Each half adder should be represented as a block with the two inputs and a single output, and must have clear, labelled connections to any other half adder.

(b) Below we provide a Verilog module for half adder:

```
module half_adder (
  input a,      // addend 'a'
  input b,      // addend 'b'
  output s,     // sum result
  output c_out  // carry out
);

  assign s = a ^ b;
  assign c_out = a & b;
endmodule
```

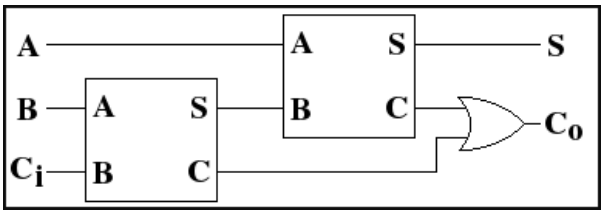
Write a simple Verilog module for a full adder by instantiating the `half_adder` module, and create wiring to replicate the connections in the diagram you made it part (a).

(c) Simulate your module and show the two outputs bits in a waveform for `a`=1, `b`=1, and `c_in`=0

(d) Draw a truth table for the full adder. **Be sure to include all possible input combinations (we assume bits can only be '0' or '1').**

Solution:

(a)



(b)

```
module full_adder (
  input a,      // addend 'a'
  input b,      // addend 'b'
  input c_in,   // carry in
  output s,     // sum result
  output c_out  // carry out
);
```

```
    wire a_i [1:0];
    wire b_i [1:0];
    wire sum [1:0];
    wire c_o [1:0];


    genvar i;
    generate
      for (i=0; i < 2; i=i+1) begin
        half_adder ha_X(  .a(a[i]),
                          .b(b[i]),
                          .s(sum[i]),
                          .c_out(c_o[i]));
      end
    endgenerate


    // Feed module inputs to first half adder
    assign a_i[0] = b;
    assign a_i[1] = a;
    assign b_i[0] = c_in;
    assign b_i[1] = sum[0]
    assign s = sum[1];

    assign c_out = c_o[0] || c_o[1];

endmodule
```

(c) Waveform should should inputs as requested, and `s` = 0 and `c` = 1

(d)

INPUTS			OUTPUTS	
A	B	CARRY IN	CARRY OUT	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Problem 2: Decoder with Logic Gates

Decoders are simple combinational circuits often used to expand a binary value to a one-hot vector. Consider a 3-bit input and convert that to a one-hot 8-bit output. For example, an input value of

`011` would result in the output `00001000`.

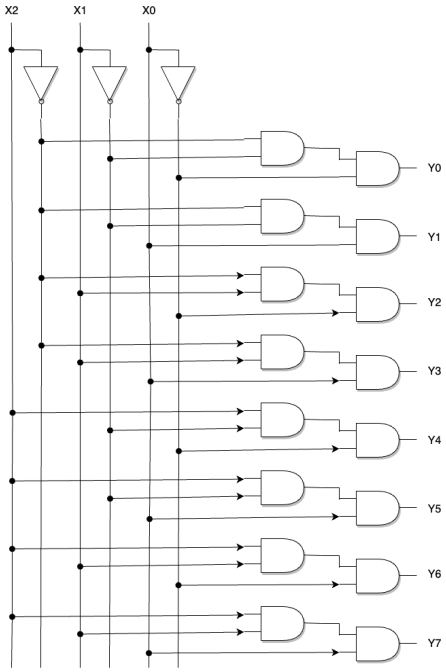
(a) Draw a schematic diagram for a 3-to-8 decoder using only simple logic gates. Such a decoder wold take a 3-bit input and output a one-hot 8-bit output. **You may use only 2-input AND, OR, XOR gates, and inverters.**

(b) Create a Verilog module which represents a 3-to-8 decoder using continuous assign statements based upon your schematic.

(c) Now, create a Verilog module which represents a 3-to-8 decoder using a single case statement. **Proper synthesizable Verilog should always include a default case. Do not forget it!**

Solution:

(a)



(b)

```
module decoder_assign (
  input [2:0] x,
  output [7:0] one_hot
);

  assign one_hot[0] = ~|x;  // Example of reduction operator
  assign one_hot[1] = ~x[2] & ~x[1] &  x[0];
  assign one_hot[2] = ~x[2] &  x[1] & ~x[0];
  assign one_hot[3] = ~x[2] &  x[1] &  x[0];
  assign one_hot[4] =  x[2] & ~x[1] & ~x[0];
  assign one_hot[5] =  x[2] & ~x[1] &  x[0];
  assign one_hot[6] =  x[2] &  x[1] & ~x[0];
```

```
  assign one_hot[7] =  x[2] &  x[1] &  x[0];


endmodule
```

(c)

```
module decoder_case (
  input [2:0] x,
  output reg [7:0] one_hot);

  always @(*) begin
    case (x)
      3'h0: one_hot <= 8'h0;
      3'h1: one_hot <= 8'h1;
      3'h2: one_hot <= 8'h2;
      3'h3: one_hot <= 8'h4;
      3'h4: one_hot <= 8'h8;
      3'h5: one_hot <= 8'h16;
      3'h6: one_hot <= 8'h32;
      3'h7: one_hot <= 8'h64;
      default: one_hot <= 8'h0;   // Proper, synthesizable Verilog needs a c
    endcase
  end

endmodule
```

Problem 3: Simple 2-bit Comparator

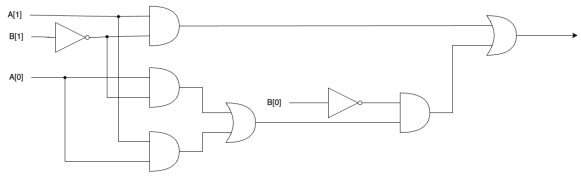
A comparator is a combinational logic circuit that takes as input two N-bit signals, a and b, and outputs a single value indicating whether the values are equal ($a == b$), $a > b$, or $b > a$. Consider a processor using base and bounds virtual memory which is trying to detect whether an address is accessible. If the given address is above some set limit, then an exception occurs. In this scenario, whether the given address is equal to or less than is irrelevant. You have been assigned to design a comparator with reduced functionality which only outputs `'1'` if and only if $a > b$.

(a) Create a schematic for the comparator. **You may use only 2-input AND, OR, XOR gates, and inverters.**

(b) Use your schematic to create a Verilog module for the comparator using continuous assign statements.

Solution:

(a)



(b)

```
module comp_grt(
  input [1:0] a,
  input [1:0] b,
  output o);

  wire and0, and1, and2, and3;
  wire or0, or1;

  assign and0 = a[1] & ~b[1];
  assign and1 = a[0] & ~b[1];
  assign and2 = a[1] & a[0];
  assign and3 = ~b[0] & or0;

  assign or0 = and1 || and2;
  assign or1 = and0 || and3;

  assign o = or1;

  /*

  A more succint way:

  wire tmp;

  assign tmp = (a[0] & ~b[1]) || (a[1] & a[0]);
  assign o   = (a[1] & ~b[1]) || (~b[0] & tmp);
  */

endmodule
```

Problem 4: Chiplets?

Die yield is the percentage of functional chips produced after the manufacturing process. Modern die yields for major semiconductor companies are greater than 90% since the manufacturing process is perfected by the time the product enters high volume production. That said, it is impossible to reach 100% yield.

Consider the following scenario: you have been assigned to assess the die yield of a new chip using a new 4nm process node. The new processor is 16mm x 16mm monolithic die and the wafer has 300mm diameter. With manufacturing you can produce wafer with a 0.2 defect density/cm^2 which is high by modern standards.

Definition: "Defect density refers to the number of defects or imperfections present in a given unit area of a semiconductor wafer"

(a) What is the die yield using the method described in lecture?

(b) What is the die yield using Poisson Model ? (**only use this model for this part**)

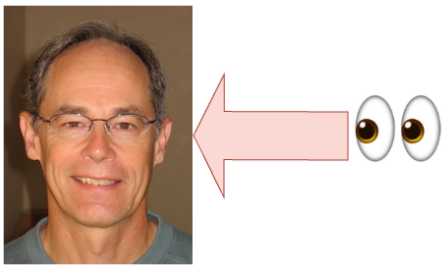
(c) Your manager is very, very mad. This isn't good enough! But, you are super smart, worked really hard with your team, and reduced the logic enough to shrink the chip size to 12mm x 12mm. What is the die yield now?

(d) Since you are super smart, you realize your processor could be composed of 4mm x 4mm chiplets! What would the die yield be if you built your processor with chiplets? How many processors could you make (assuming each chiplet is identical)?

Chiplet Definition: "... an integrated circuit block that has been specifically designed to communicate with other chiplets, to form larger more complex ICs. Thus, in large and complex chip designs the design is subdivided into functional circuit blocks, often reusable IP blocks, called "chiplets", that are manufactured and recombined on high density interconnect."

Solution:

I wonder who coined the term "chiplet"...



Poisson die yield model is a simplest model because it assume uniform probability of defects across the wafer. This should be interpreted as the lower bound. The equation is: $Y = e^{-AD}$; A is chip area, and D is defect density.

(a) $Y = (1 + \frac{1.6^2 * 0.2}{3})^3 \approx 62.33\%$

(b) $Y = e^{-1.6^2 0.2} \approx 60\%$

(c) $Y = (1 + \frac{1.2^2 * 0.2}{3})^3 \approx 76\%$

(d) $Y = (1 + \frac{0.4^2 * 0.2}{3})^3 \approx 96\%$

$\frac{\pi(\frac{300mm}{2})^2}{4mm^2} - \frac{\pi 300mm}{\sqrt{2*4mm^2}}$ = 17338.242 die, It takes 9 die (chiplets) compose a single 12mm x 12mm die. Therefore, $\frac{17338}{9}$ = **1926** processors from each wafer.

Problem 5: Automotive World Affair

The automotive industry is taking a special interest in hardware in order to integrate more technology into their vehicles to improve consumer experience. Edison, a new automotive startup, is gearing to rival Tesla and their new model AC. Edison's new sedan, model DC, relies on a intricate vision system which needs a processing unit to aid autonomous driving. You, the only intern, have the big decision of deciding whether this processing unit should be FPGA-based or ASIC-based.

(a) Uh oh! News was leaked that Tesla's model AC is going to production next month. Earlier than Edison anticipated! Your manager says you need to make a decision quick. Would you choose an FPGA or an ASIC? Why one over the other?

(b) You made a report explaining your decision, but you are not satisfied. This decision determines cost to the company. Below is the summary of your cost analysis you put in the report:

	FPGA	ASIC
Design Cost	\$100K	\$125K
Design Verification Cost	-	\$200K
Photomask Creation	-	\$100K
Cost per Chip	\$160	\$0.56

You already have 2000 pre-orders for the model DC. Did you make the right decision?

(c) Great news! Reports on the model AC state:

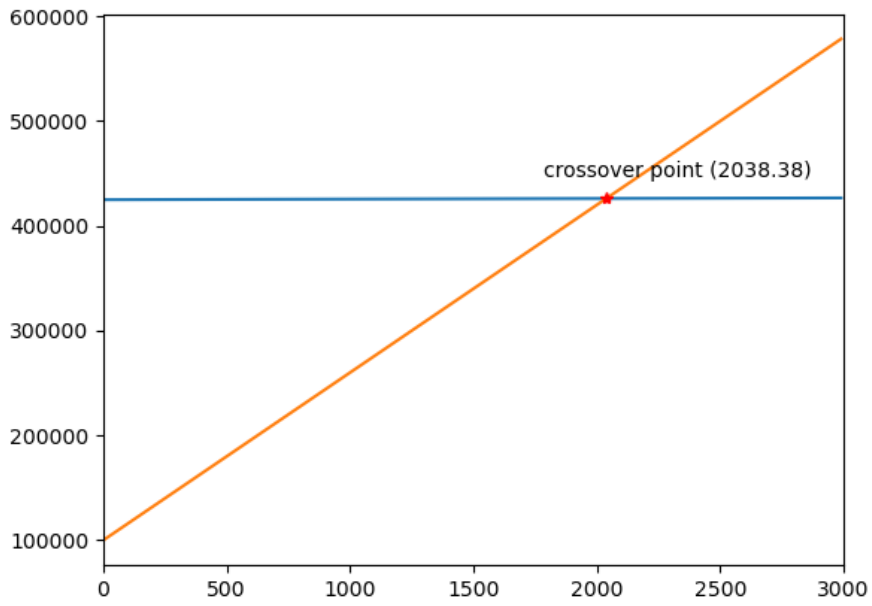
"The model AC suffers from a fatal flaw. Overheating! Due to a manufacturing error, poor cooling of the power system causes the main processing unit to overheat, resulting in sporadic failure to the autonomous driving system. Fix will take years!"

Consumers respond and another 350 pre-orders for the model DC come in. Should you recommend to your boss to switch hardware platforms? Why?

Solution:

(a) FPGA. FPGAs are products premade by semiconductor companies guaranteeing functionality. ASICs are custom and therefore, must be designed, verified, and tested which takes years. If time to market is the primary concern and under 2 years, FPGAs are always the way to go.

(b) Yes, but best way to visualize the tradeoff between FPGA and ASIC is to plot the cost. The total cost per volume is linear: the cost per chip is the slope and inital cost (NRE) is the intercept. Below is the plot for this problem. Here we see the crossover point, the number of unit at which cost is the same for both platforms, is 2038.38. Therefore, at 2000 units it is still better to use FPGAs.



(c) Yes, you should recommend to develop an ASIC. With the additional 350 preorders, the total number of units that need to be produced is 2350, which greater that the crossover point.