Last Lecture

- Admin stuff
- Finish up error correction
- Course review/overview
- Review questions for the final

Admin stuff

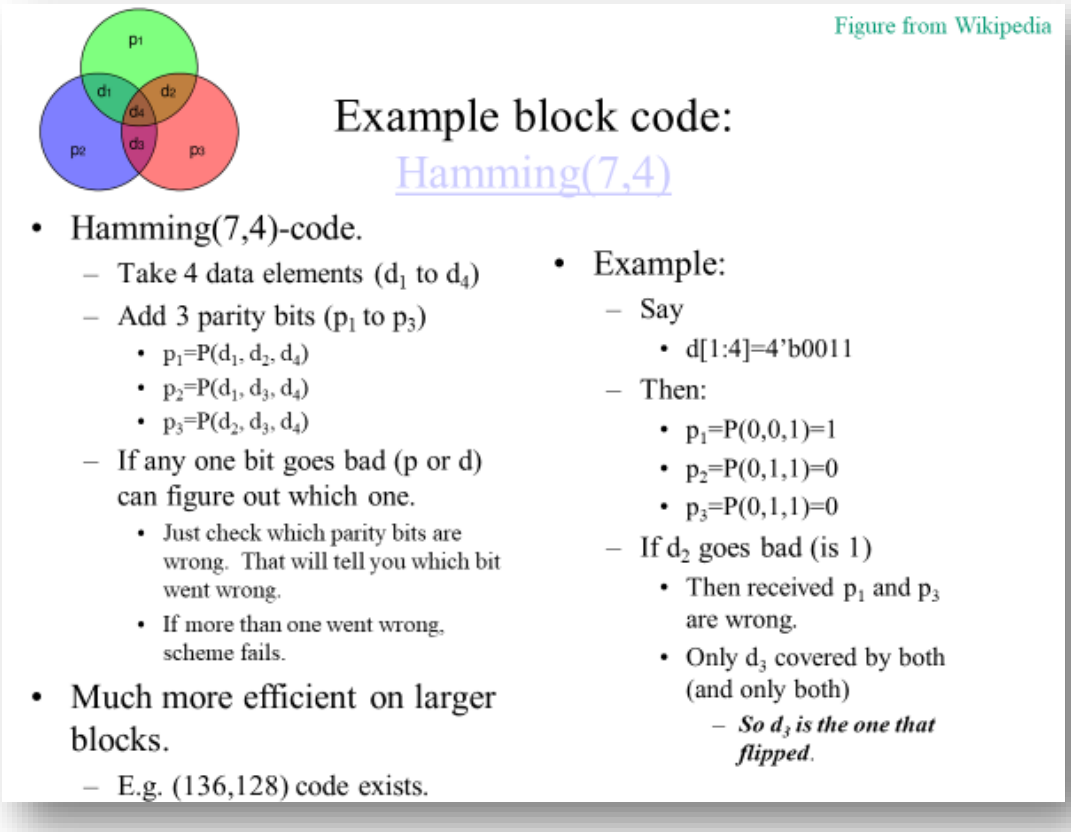
- Lab 7 part 2 and the postlab are due today. We will hold regular lab hours on Tuesday
- Final, Thursday 4-6pm.
- I'm holding my regular office hours on Tuesday. Wednesday office hours will be from 12-2 in lab.

Error Correction

Even's one parity: take a set of bits and add either a zero or one such that the number of ones, total, is even.

P(1,1,1)=1; P(1,0,1)=0. P(1,1,1,1)=_____.

With parity, any one-bit error can be detected. How?
Now, let's consider *correcting* a one-bit error.



With this particular scheme, we can send 4 bits of data with 3 bits of error correction (parity bits) for a packet of 7 bits. If any one bit flips, we can figure out which what is was and flip it back!

What if d_1 gets flipped?

What if d_4 gets flipped?

What if p_1 gets flipped? (we only care about the data, but we can still tell…)

- Consider receiving the packet {d[1:4],p[1:3]} as 0000110. What was the data that was sent?
- What about the packet 1111011?

Quick Course Review

(Most of this is on the board, not in the notes)

The focus of this class is on combinational and sequential logic as well as their implementations and applications.

We started with gates and logic rules. We learned that any combinational circuit can be represented as a truth table, but that doing so isn't viable as the number of inputs grows because the size of that table is exponential in the input size. So, we looked at ways to design beyond truth tables by using hierarchy and functional decomposition. Renee, in lab 3, was an example where the number of inputs (10) was just too large to do with a truth table and it wasn't even clear how to attack that problem by any means other than breaking the problem into subproblems and then combining those. Lab 5 (the calculator) continued the idea of breaking things down into parts and roughly introduced the idea of having control and data separate.

On the sequential side, we covered how to build latches and flip-flops out of gates and how to use sequential logic to solve problems that couldn't be solved with just combinational logic. We learned about different types of flip-flops (T, SR, JK) and about sequential MSI devices (registers, shift registers, memories, etc.)

We then looked a bit at a level below gates: CMOS and FPGA architecture. Hopefully those two convinced you that while there is more than one way to implement gates, memories, and the like, the gate-to-MSI-device level of abstraction is useful in either case.

We also looked at applications. This included state machines, embedded systems (lab 7, SPI), and a very basic computer.

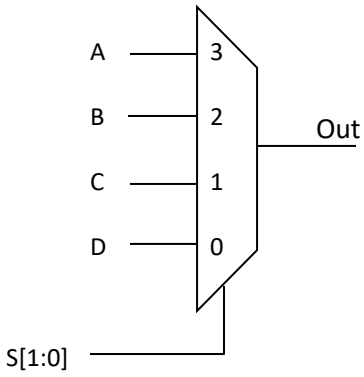
The hope is that this class will give you a solid foundation for embedded systems (373), computer architecture (470), VLSI (312/427), and related CAD tools including verification (478).

Review questions

We will be doing mostly review problems today. We aren't going to do all of these today, but they make for a good review, so you should consider doing them…

Combinational logic:

1. Using only tri-state devices, 2-input AND gates, 2-input OR gates, 2 to 4 decoders and inverters, build a 4 to 1 MUX. Use the same labels as the diagram below. Your design must use *seven* or fewer devices to receive credit. **[8]**



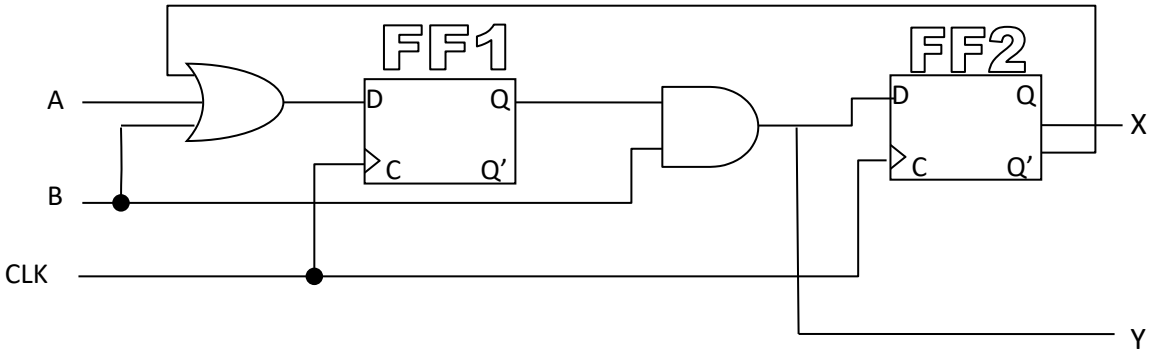
2. If you had to build a 4 to 1 MUX using only two-levels of gates (not including inverters) how would you do so?

3. Using only 2-input AND, OR, and XOR gates as well as inverters draw a circuit which takes 6 bits as an input and outputs a 1 if the parity of those bits is odd. Label the inputs as I[0:5] and the output as "Odd_P". Your solution must have 8 or less gates (including the inverters).

Sequential logic

1. Design a D flip-flop using a J-K flip-flop and standard gates. (JK is "hold" if both 0, clear if J=0&K=1, set if J=1&K=0, invert if both 1).

2. Draw the state-transition diagram associated with the circuit below.



3. (This one is hard)
   a. Explain why you can't build a FSM that solves the following problem:

      There are two inputs X and Y and one output Z. Z is to be a "1" iff X has been a 1 more times than Y has been a 1.

   b. If you were told by your boss: "no really, we need this solved", what would you do?

Minimization

1. Using a Kmap minimize the following function W=(A⊕B⊕C)+A*!B*C.

2. Do Problem 1 again, but using Q-M.

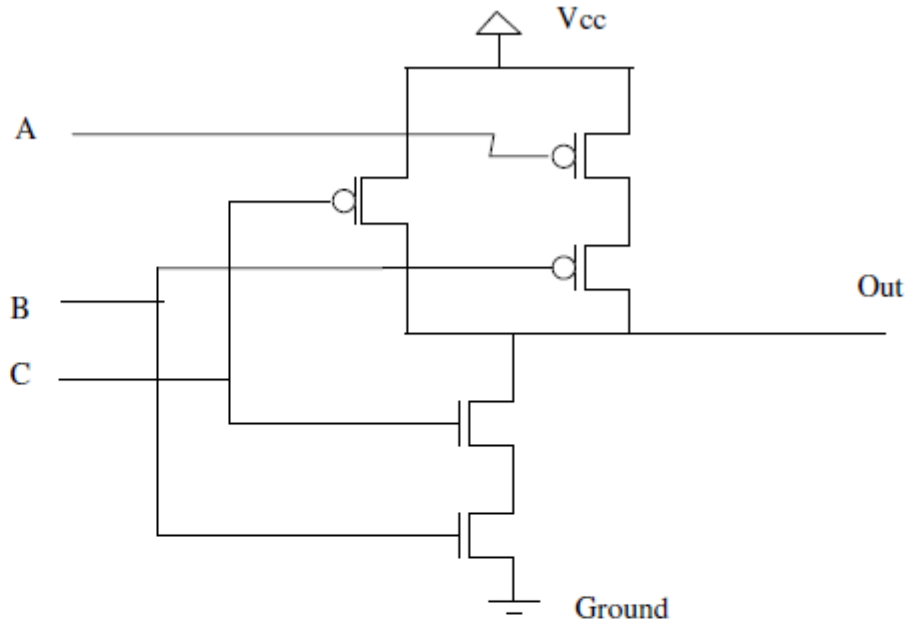
3. [K-Map Minimization—7 Points]

Derive a minimal SOP *and* a minimal POS expression for the function $f(a, b, c, d) = \sum_{(a, b, c, d)} (1, 2, 4, 5, 6, 13) + d(7, 9, 10, 11, 12)$. Mark your solution on the K-Maps below.

ab \ cd	00	01	11	10
00	0	1	0	1
01	1	1	d	1
11	d	1	0	0
10	0	d	d	d

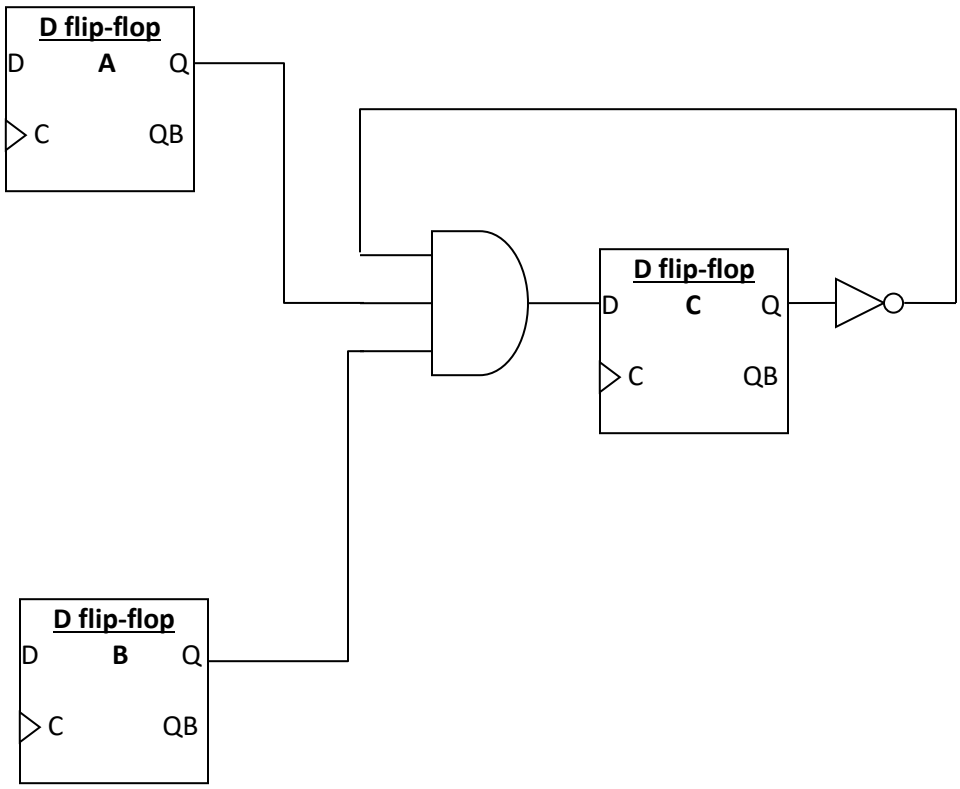
ab \ cd	00	01	11	10
00	0	1	0	1
01	1	1	d	1
11	d	1	0	0
10	0	d	d	d

Misc.



A	B	C	Out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Setup/hold and clock skew.



1) Consider flip-flops A, B and C, each nominally clocked off of the same clock. Assume
   - Each flip-flop has a set-up time of 5ns and a clock-to-Q delay of 2ns to 4ns.
   - The AND gate has a delay of 2 to 6ns.
   - The NOT gate has a delay of 1 to 2 ns.
   - Flip-flops A and B have no clock skew between them.
   - Flip-flop C's rising edge may be as much as 0.5 ns before A and B's rising edge or as much 2.5ns after.

   a) What is the fastest clock period you could safely clock this system at?

   b) What is the (non-negative) range of values for the hold time that would be sufficient?

   c) Redo part a) assuming the NOT gate always had a delay of 0ns

   d) Redo part b) assuming the NOT gate always had a delay of 0ns.

This problem is quite hard. There was at least one semester where a company basically asked this question as their first-level filter on interviews even though they knew we didn't teach this in 270 (or 312, at least at the time).