1) **Bubble pushing and Boolean algebra**

Express the following circuit:

![Circuit Diagram]

Using

a) **2-input NAND gates only**

b) **2-input NOR gates only**

Three concepts are key to approaching this problem:

First, splitting gates:

![Splitting Gates Diagram]

is equivalent to

Same holds for OR gates. NAND and NOR gates are a bit trickier: you have to remember that a NAND is an AND with an inverter at the output.

To build an inverter, you can use NAND or NOR gates as follows:

![Inverter Diagram]

Bubble pushing is the 3rd key concept here.

* A bubble is an inverter.
* You can add 2 bubbles to any wire without changing what your circuit does (inversions will cancel).
* Finally, by DeMorgan’s law:

![DeMorgan's Law Diagram]

Applying these concepts, we get the solutions to this problem:
2) Logic minimization
Minimize the following expression. Using a giant 6-input K-map is NOT a good way to do this!

$$Out = A\overline{B} + BCD + EF + AC + E$$

Before jumping into K-Maps (K-Maps are usually a good idea for 2,3, and 4-variable functions), observe that the EF term is redundant:

$$EF + E = E(1+F), \text{ and since } 1+F = 1, \text{ The expression evaluates to } E.$$

Our expression is now the following: $AB' + BCD + AC + E$

Since the last term has nothing to do with the rest of the expression, it cannot be minimized away. We can leave it off to the side and use a K-Map to optimize $AB'+BCD+AC$.

Unfortunately, we have given you an expression that is already minimal:

Thus our final, minimized expression is $Out = AB' + BCD + AC + E$

3) Clock division and counters.
   a) Create a circuit to divide a clock by 3 circuit using a ring counter.
The initial value of the registers is 0 0 1

a. Can the output clock have a 50% duty cycle (Reasonably)?
   Not reasonably. For the output clock to be 1 50% of the time, the output has to be a logic high 1.5 out of every 3 cycles. This isn’t easy, as a mix of positive and negative-edge triggered flip-flops is needed, and the resulting circuit has very complex behavior. Most importantly, the duty cycle of the output clock would depend heavily on the input clock having a 50% duty cycle.

b. Does the output clock’s duty cycle depend on the input clock’s duty cycle?
   No! The duty cycle of the output does not depend on the duty cycle of the input since our registers are all positive-edge triggered (this is always the case unless otherwise stated. If you assumed these to be negedge-triggered, you are safe as long as you are consistent). We only trigger on positive edges, so we do not care when the negedge occurs. Thus the output duty cycle is independent of the input.

b) Create a circuit to divide a clock by 4 circuit using a Möbius counter.

The initial value of the registers is 0 0

a. Can the output clock have a 50% duty cycle (Reasonably)?
   Yes! Division by even numbers easily allows for a 50% duty cycle output: just make sure your output is 1 50% of the time, 0 50% of the time. A Möbius counter makes the very simple due to the inverted loop.

c) Create a circuit to divide a clock by 8 circuit using a binary counter. You may use an adder block and a comparator block.
You do not need to know how the adder is built: you will learn this in gory detail in the weeks to come. The comparator to zero is an AND gate with 3 inverted inputs for each bit of the bus. The single bit of the output is 1 only when the 3 inputs are all 0.

Notice that the adder + register make an accumulator (HW1).

Note that the accumulator with a 1 input is a Counter (You’ve seen these in lab).

Since we divide by 8, which is $2^3$, there is no need to reset the register using the output of the comparator – the counter will simply roll over to 0 once it passes 7.

d) Create a circuit to divide a clock by 15 using an LFSR. You may use a comparator block.

An LFSR is a very important circuit. For a chain of N registers, it will generate a pseudo-random sequence of $(2^N)-1$ values.

Note: The value stored in the LFSR is never 0 (Why? See what happens when the LFSR is zero).

The initial value of the registers is 0 0 0 1

4) CMOS!

Implement the following expression using CMOS logic (PMOS and NMOS transistors).

Recall: A transistor is a perfect switch!

$$Out = \overline{AB} + CD$$

If you forgot how to do this, read solutions to HW1 very carefully.
I suspect this is more complex than anything you will need to do on the test. If in doubt, go read the very complete guide in the HW1 solutions (P1.2).

5) K-Maps
   a) Minimize the following expressions using a K-Map:
      
      \[ \text{Out} = AB + \overline{B}C + CD + A\overline{C}D + AD \]

      The final expression reads: \( AB + CD + AD + B'C \)

   b) Draw a minimal gate-level implementation for the following
The final expression reads $A'B' + B'D'$

Know how to express this in POS $(B'(D'+A'))$