# CSci 2021, Spring 2015
# Homework Assignment V
**Due: Friday, May 8th 2015, at beginning of lecture**

## Problem 0: (1 point)
Clearly label your assignment with your name, UMN email address, and the time of your recitation section (8:00, 9:05, 10:10, 11:15, 12:20, 1:25, 2:30). This will help us turn back your graded assignments more efficiently.

## Problem 1:
Show that the following two Boolean functions are equivalent, by writing truth tables for them:

```
(b ^ (a & c)) & (a | c)
((a & b) ^ c) & (b | (a ^ b))
```

## Problem 2:
Your goal for this question is to design a small multiplication circuit, which takes two 2-bit signed integers as input (i.e., `00` representing 0, `01` representing 1, `10` representing -2, or `11` representing -1), and produces a 4-bit signed integer representing their product. Call the first input $x_1x_0$, where $x_1$ is the more significant/sign bit, and similarly call the second input $y_1y_0$. Call the product $z_3z_2z_1z_0$, where $z_3$ is the most significant/sign bit.

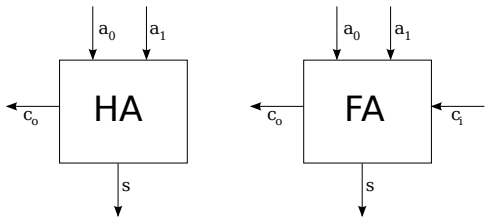
First, write a truth table showing the desired values of $z_3$, $z_2$, $z_1$, and $z_0$ based on $x_1$, $x_0$, $y_1$, and $y_0$. Then, draw four 4-input Karnaugh maps, and use them to write a minimal sum-of-products Boolean formula for each output. You don't need to actually draw the final circuit.

## Problem 3:
Your goal for this question is to design a small multiplication circuit, which takes two 3-bit **unsigned** integers as inputs (representing numbers between 0 and 7), and produces a 6-bit unsigned integer representing their product. Label the inputs and output $x_2x_1x_0$, $y_2y_1y_0$, and $z_5z_4z_3z_2z_1z_0$.

But this time, design the circuit using the same approach you learned for decimal multiplication in grade school, using AND gates for multiplying bits and half-adders and/or full-adders for adding the results. I.e., something like:

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | $x_2$ | $x_1$ | $x_0$ |
| | | | $\times$ | $y_2$ | $y_1$ | $y_0$ |
| | | | | $x_2 \times y_0$ | $x_1 \times y_0$ | $x_0 \times y_0$ |
| | | | $x_2 \times y_1$ | $x_1 \times y_1$ | $x_0 \times y_1$ | 0 |
| $+$ | | $x_2 \times y_2$ | $x_1 \times y_2$ | $x_0 \times y_2$ | 0 | 0 |
| $z_5$ | | $z_4$ | $z_3$ | $z_2$ | $z_1$ | $z_0$ |

Draw a circuit, representing half adders and full adders by boxes labeled HA and FA respectively, where the bits $a_0$ and $a_1$ being added come in to the top of the box, the sum bit $s$ leaves from the bottom, and carry bits go right to left, with the carry-out $c_0$ on the left and the carry-in $c_i$ for a full-adder on the right:



## Problem 4:

You've been hired by an electronics component company that likes to add features to their devices. Your job is to help design the circuit for a new product, which is called an HSRLTD flip-flop. This one-bit storage device has 5 control inputs (H, S, R, L, and T), one data input (D), one clock input, and one output Q. On a rising clock edge, the flip-flop's stored value is updated according to the following rules:

- If the H ("hold") signal is true, the value is unchanged.
- If the S ("set") signal is true, the value is set to 1.
- If the R ("reset") signal is true, the value is set to 0.
- If the L ("load") signal is true and T ("toggle") is false, the value is replaced with the value of D.
- If the L ("load") signal is true and T is true, the value is replaced with `!D`.
- If the T ("toggle") signal is true but L is not, the value is replaced by the negation of the old value.

Any other combination of the control signals is undefined, so it doesn't matter what the flip-flop does then.

a. Draw a circuit to implement an HSRLTD flip-flop using a positive-edge-triggered D flip-flop.

b. A year later, it turns out that the HSRLTD flip-flops were not big sellers, and the company has a lot still in the warehouse. Draw a circuit for how the company could use an HSRLTD flip-flop to make the more common JK flip-flop (also positive edge-triggered).

## Problem 5:

In electronics that connect straight to simple output systems, it can be convenient to represent use bits to represent decimal values. For instance we can use 4 bits to build a counter that goes from 0 to 9 and back to 0, called a decade counter.

We can consider the decade counter to be a state machine with no input signals and ten states. If we use the normal representation for 0-9, the state update function is just an increment, except that after 9 it should wrap back around to 0. (A mathematician would call this incrementing modulo 10, like the C expression `(x + 1) % 10`. The remaining six states should never arise, so it shouldn't matter what the state update function does to them.

a. Find minimal sum-of-products representations for the four bits of the state update function using four 4-input Karnaugh maps. The results of incrementing 1010 (10) through 1111 (15) are undefined, so they should be don't-care entries in the map; take advantage of them to make your formulas simpler.

b. One practical issue circuit designers have to think about is what happens when a device is first turned on. Unless you take care to ensure otherwise, flip-flops can start in a random state when they first get power. This could lead to a problem if they're in an undefined state in a state machine that they can't get out of. On the other hand, we say that a counter design is self-starting if it doesn't have this problem: from any state, it will follow a sequence of transitions to get to an intended state.

   Assuming you use your minimal formulas from (a), draw an expanded version of the state diagram for the decade counter that includes the extra 1010 through 1111 states and what they transition to. Is this state machine self-starting? If it isn't, propose a small change to one of the update functions to fix it.

## Problem 6:

In this problem, we'll design a state-machine control for a simple thermostat controlling an HVAC system that can produce either heat or cooling. There's another circuit which measures the temperature and compares it to the desired temperature, and produces three signals encoding five possible states (where x means "don't care"):

| $T_g$ | $T_h$ | $T_b$ | Interpretation |
|---|---|---|---|
| 0 | 1 | 1 | Much too hot (MH) |
| 0 | 1 | 0 | A little too hot (LH) |
| 1 | x | x | Just right (JR) |
| 0 | 0 | 0 | A little too cold (LC) |
| 0 | 0 | 1 | Much too cold (MC) |

The goal of a thermostat is to keep the temperature close to the desired range. There are two output signals, H which enables the heater and C which enables the cooling; H and C can both be off if the temperature is fine, but they should not be both on at the same time.

We don't want the heating or cooling to switch on and off too frequently, or to switch repeatedly between heating and cooling, because those would waste energy. One thing we'll do to help with this is to clock the state machine every minute, using a temperature based on the average over the last minute (the measurement circuit is responsible for this). But we also want the state machine to obey the following principles:

- The thermostat should stay in any given output state for at least two clock cycles (two minutes). Once we decide to turn the heater on, it can stay on for 2, 3, 4, 5, etc. minutes, but never just for 1, and similarly for cooling or the periods when H and C are both off.
- If the last system we ran was the heater, we shouldn't turn on C until we get to "much too hot". If the last system was the cooler, we shouldn't turn on the heater until it gets "much too cold".
- If the last system we ran was the heater, we should turn H back on when we get to the "a little too cold" condition, and leave it on until we get too "a little too hot". Conversely if the last system we ran was the cooler, we should turn it on when we get "a little too hot", and leave it on until it is "a little too cold".

Note these principles are in priority order: the first one overrules the others if there's a conflict, and the second overrules the third if there's a conflict. When the thermostat is first turned on, it won't have either heated or cooled in the past, but we will choose the initial state so that it as if it was last heating.

Your job is to design a state machine that implements the above principles. Use a Moore machine design, where the inputs are the three T signals, and the output is the H and/or C signals. Try to have as few states as possible to achieve the desired behavior.

a. Draw a state diagram for your state machine. Give each state a unique label, show whether either H or C should be enabled in that state, and draw transitions between states labeled with combinations of MH, LH, JR, LC, and MC.

b. Choose a binary encoding for your states, and based on that, write truth tables for the the output function and update function of your state machine, including don't cares.

(You don't have to make circuits for the output and update functions.)

**Problems 0, 2, and 4 should be submitted for grading.**