ECE 120: Introduction to Computing Spring 2019 ZJUI

due at start of section on Friday 12 April 2019

Homework 7: Basic Sequential Logic

1. Safe Drawbridge Control

A drawbridge is a bridge that can be raised to allow boats to pass, but can also be lowered to allow people and vehicles to cross. You must design logic to control the operation of the bridge. The bridge position is an unsigned N-bit number P. When P=0, the bridge is fully lowered, and when P=R (an N-bit unsigned constant), the bridge is fully raised.

Here are the rules:

- If a vehicle is on the bridge (V=1), lower the bridge until it has been fully lowered, then keep it in place.
- Otherwise, if a boat is approaching the bridge or passing through (B=1), raise the bridge until it has been completely raised, then keep it in place.
- Otherwise, if the bridge is raised, lower it.
- Otherwise, stay put.

Given the inputs P (the current bridge position, as N bits), V (vehicle), and B (boat), you must produce an N-bit number Q corresponding to the next desired bridge position, following the rules given above. If the bridge is to be raised, Q should be P + 1. If the bridge is to be lowered, Q should be P – 1. If the bridge is not to move, Q should be equal to P.

You are given the following components to complete your task: an N-bit "+1" (adds 1), an N-bit "-1" (subtracts 1), an N-bit zero-checker (outputs 1 iff input is exactly 0), and an N-bit R-checker (outputs 1 iff input is the constant R). Draw these as rectangles with appropriate labels, inputs, and outputs.

You are also given a 4N-to-N mux (N copies of a 4-to-1 mux with common control inputs).

Finally, you may use three additional 2-input NAND or NOR gates, and as many inverters as you want, to complete your logic and produce the next bridge position Q.

2. Vending Machine Change

Generations of students have complained about Prof. Lumetta's vending machine (you'll see it soon): it produces no change! In this exercise, you will solve that problem.

Given a 4-to-16 decoder, up to 12 four-input gates (any type), and at most two levels of gates after the decoder (for any function), you must convert a 4-bit unsigned number $M=m_3m_2m_1m_0$ into appropriate bill-release signals. In particular, since M is in the range 0 to 15, you must produce a signal B_{10} that should be 1 iff a 10 RMB should be produced, a signal B_5 that should be 1 iff a 5 RMB bill should be produced, and four signals B_4, B_3, B_2, and B_1 that each produce 1 RMB bills when set to 1. For example, if M=8, you circuit should produce B_{10}=0, B_5=1, B_4=0, B_3=1, B_2=1, and B_1=1 (one 5 RMB bill and three 1 RMB bills).

3. Understanding Sequential Logic

Update your Subversion repository to obtain the subdirectory **`hw7dist`**, which contains the program **`latch.c`**. The program finds and prints all stable states for an $\overline{R}$-$\overline{S}$ latch. Read the program and examine its output to make sure that you understand how it works.

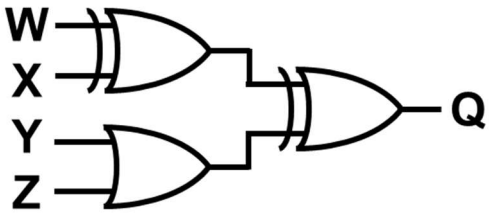
a. Modify the program to compute stable states for two cross-coupled AND gates. In other words, replace the two NAND calculations with AND. Execute the program to find the stable states of such a circuit. Commit your modified code back to your repository for grading.

b. For which combination of inputs $\overline{R}$ and $\overline{S}$ is the "latch" that you simulated in **part (a)** bistable? In other words, for which combination of inputs does the "latch" have two stable states?

c. Since the "latch" simulated in **part (a)** is bistable, one can use it to store a bit. Give two reasons that such a design (using two AND gates) is inferior to the $\overline{R}$-$\overline{S}$ latch (using two NAND gates) in CMOS technology..

4. Sequential Feedback Analysis

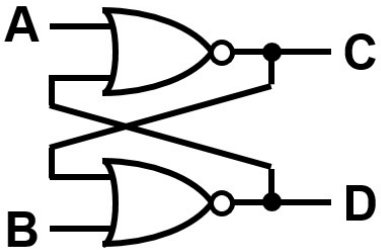
a. Write a truth table for the circuit shown below.



b. Now imagine that we connect Q back to W. For what combinations of the remaining inputs (XYZ) does the output Q oscillate between 0 and 1? In other words, which values of XYZ prevent the output Q from holding a constant value?

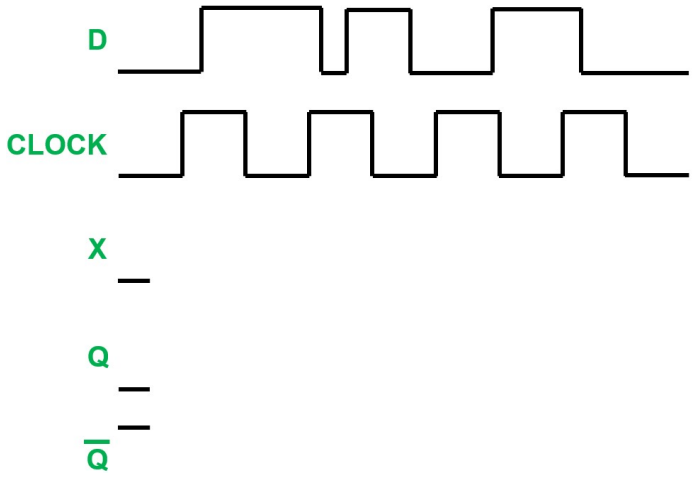
5. Stable States

For the circuit below, write a table of all stable configurations of the outputs C and D for each possible combination of inputs A and B. Use only as many rows as necessary.



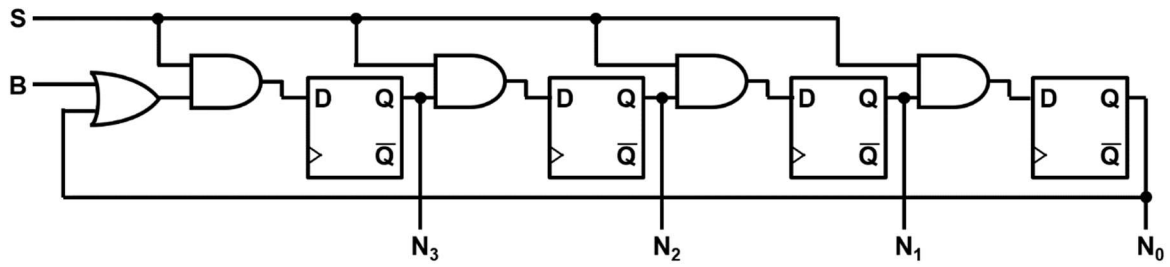
6. Positive Edge-Triggered D Flip-flop

Based on the implementation diagram for the positive edge-triggered D flip-flop on page 68 of the notes, complete the timing diagram below by drawing the signal values for X, Q, and $\overline{Q}$ as a function of time.



7. Registers with Logic

Consider the register below, which is constructed from four positive-edge-triggered D flip-flops and five gates. A serial input B is fed into the leftmost flip-flop, a control signal S affects all of the flip-flops' next values, and the values of all flip-flops can be read in parallel as the output $N=N_3N_2N_1N_0$. All four flip-flops share a common clock signal.



Fill in the table below with the four-bit value held in the register for each clock cycle (row) shown. If a particular bit's value is not possible to know, write a question mark in the box, as shown for cycle 0.

	B	S	N_3	N_2	N_1	N_0
cycle 0	1	1	?	?	?	?
cycle 1	0	1				
cycle 2	0	1				
cycle 3	1	1				
cycle 4	1	0				
cycle 5	1	1				
cycle 6	0	1				
cycle 7	1	1				
cycle 8	1	1				
cycle 9	?	?				

8. A Bit-Sliced Palindrome Checker

A palindrome is a word that is spelled the same way in reverse. Examples in English include the words "level," "did," and "eye." In binary, we can define a palindrome as a set of bits that has the same bit values when reversed, such as 101010101, or 10011011001. You must design a bit-slice that allows one to check whether an N-bit number is a palindrome.

a. First, draw an abstract model of your bit slice: it should accept two bits of input from the number. Determine the remaining design parameters yourself.
b. Given the number of bits passed between bit slices in your design, provide a representation and meanings for the possible bit patterns (you have complete freedom here, but we need to know what you chose to check the rest of your work).
c. Draw a circuit using the abstract model of your bit slice (rectangles with appropriate inputs, outputs, and labels) to show how one can use your design to check whether a 4-bit number $A=a_3a_2a_1a_0$ is a palindrome. Be sure to label all inputs to each of the bit slices and to connect the bit slices appropriately.
d. Repeat **part (c)** for a 5-bit number $B=b_4b_3b_2b_1b_0$.
e. Implement your bit slice using NAND, NOR, NOT, XOR, and XNOR gates (no AND, no OR).
f. Analyze the area and delay of your design when used to check whether an N-bit number is a palindrome. Here, you may assume that N is even. Remember, however, that you do now need to count inverters that are needed between bit slices. Complemented bits from the number are free, since they probably come from flip-flops. If you use an XOR/XNOR gate, count it as one operator and one gate delay.
g. Re-implement your bit-sliced design shown using the *serialization* approach presented in Lecture Notes Set 3.1. Use the bit-slice circuit as a building block (draw it as a rectangle—don't redraw the circuit) and add the storage and logic necessary to turn the bit-slice into a serial implementation.
h. Calculate the area and minimum delay when using your serialized implementation from **part (g)** on N-bit inputs. As before, you may assume that N is even. And, since the bits are stored in flip-flops, you can ignore inverters needed for the logic between cycles (between bit slices, but now using the same physical slice). Count flip-flops as area 18 and 4 gate delays on either side of the clock edge, as we did in lecture (the serialization examples).