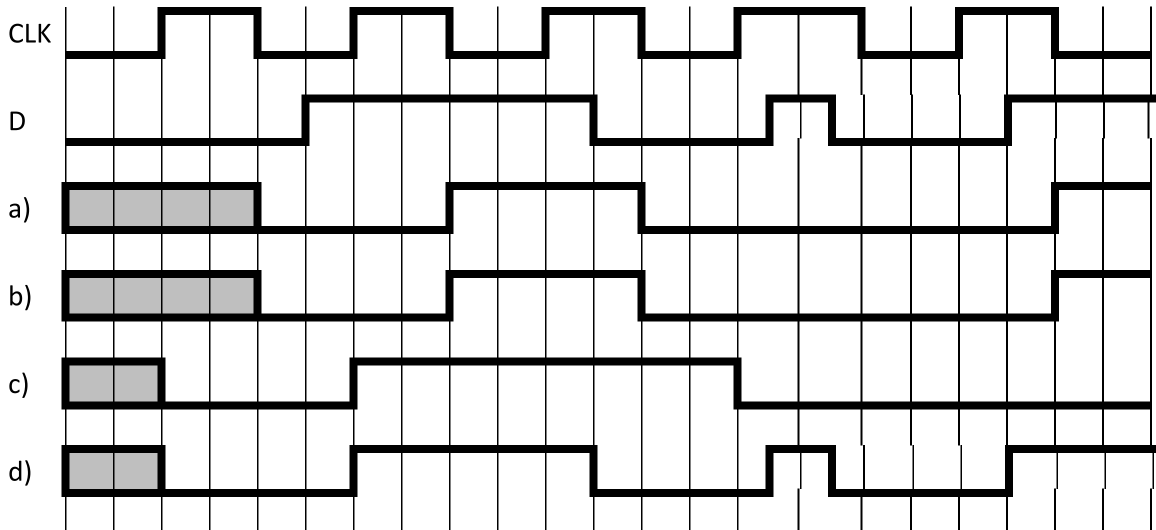
CSE370 HW6 Solutions (Winter 2010)

1. **CLD2e, 6.10**
   For this problem we are given a blank waveform with clock and input and asked to draw out the how different flip-flops and latches would behave.



   For (a) and (b), both are negative edge triggered which means that we can basically sample D each negative edge of the clock. Those are the only locations that the waveform can change.

   For (c), a positive edge triggered flip-flop, we only sample D at each positive edge and so that is the only time that the output can change.

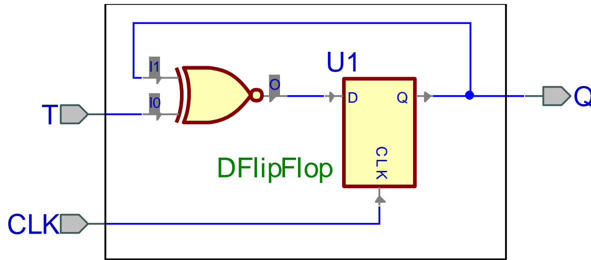
   For (d), this is a latch so that means the output can change at any time D changes while the clock is 1.

2. **CLD2e, 6.17**
   For this problem we need to show how to implement a T flip-flop using just a D flip-flop. The behavior of a T flip-flop is to toggle its output whenever the clock ticks and the input is 1. Therefore we get the following truth table to describe the behavior (let T be the input, Q be the current output, and Q+ be the next output):

T	Q	Q+
0	0	0
0	1	1
1	0	1
1	1	0

   We can minimize this to show that Q+ = T XOR Q. Now if want to build this circuit out of a D flip-flop we can use the process we've been using for building state machines. Let the flip-flop hold the current output (the "state"). Then the "next state" can be described by the "current state"

and the inputs. In this case the logic is described by our equation. Now we have the state and combinational logic so we can realize the circuit:

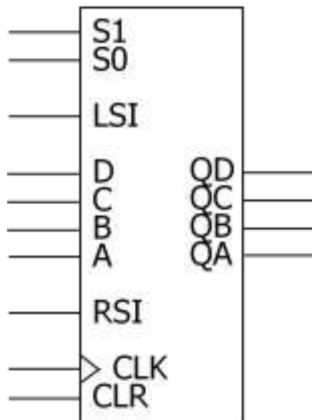


3. CLD2e, 6.25

We saw this example in class so I will pretty much repeat what the professor said and try to expand on any points that might be of interest.

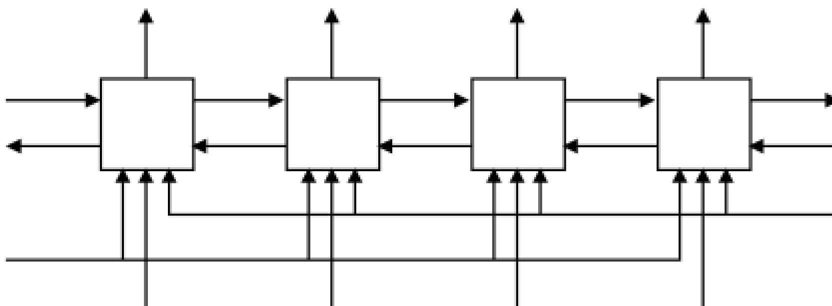
Remember the basic problem is to take the 74194 (a universal 4-bit shift register) and then implement some additional logic on top of it.

To begin we'll review what is in the 74194. It has a block diagram as follows:

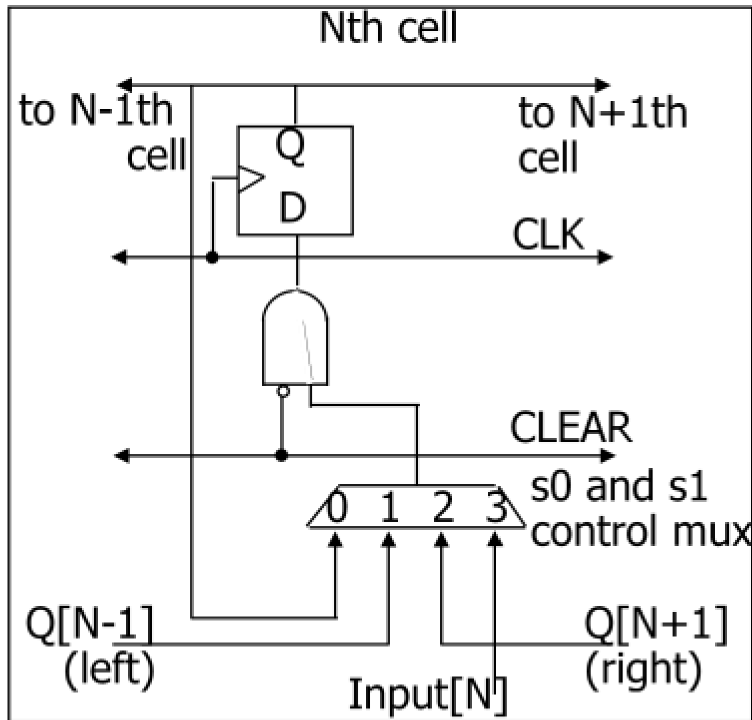


S1 and S0 are two bits that select the operation that the shifter performs. 00 = Hold, 01 = Shift Right, 10= Shift left, 11 = Load. LSI is "Left Shift In" which is the value to shift into the right-most position if we do a "shift left". Similarly RSI is "Right Shift In" and gives the value to put in for a "shift right". DCBA are new inputs to store if the register is set to "load". CLK is the clock. CLR clears the shift register. Finally, QA-QD are the outputs of the registers.

Internally, the shift register has the following circuit diagram:

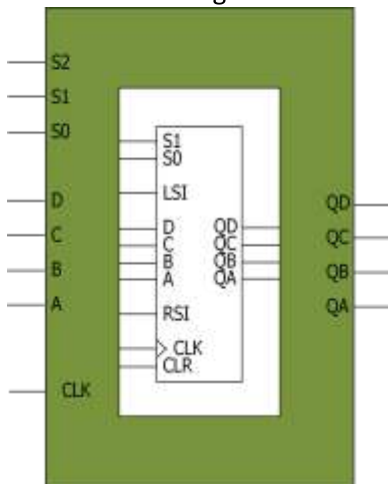


…where each block is a sub-circuit around a D flip-flop:

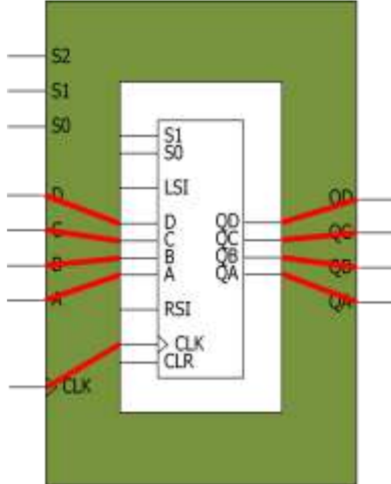


The MUX handles whether we are doing the hold, shift-right, shift-left, or load.

With this basic building block we need to build a circuit with other types of shift operations. Namely, we need to implement: hold, circular shift right, circular shift left, logical shift right, logical shift left, arithmetic shift right, arithmetic shift left, and load. Load and hold seem pretty straight forward since our 194 has this built right in. Also notice that for all the other shifts, the only thing we really need to worry about is what to shift into the register when we do the shift. For circular shifts we take the value we are shifting out. For the logic shifts, we will be shifting in 0's. For the arithmetic shifts left, we will be shifting in 0's, but for shift-right we will shift in the previous value of that register (0 causes a 0 to shift in, 1 causes a 1). More generally, we want to build the following:



Many of the inputs can be connected directly up (D-A, QD-QA, and CLK):



So now we just need to figure out the logic from the outer inputs to the 194 inputs and we can use a truth table for this:

Outer				Inner				
S2	S1	S0	Operation	LSI	RSI	s1	s0	CLR
0	0	0	Hold	X	X	0	0	0
0	0	1	Circular shift right	QA	X	1	0	0
0	1	0	Circular shift left	X	QD	0	1	0
0	1	1	Logical shift right	0	X	1	0	0
1	0	0	Logical shift left	X	0	0	1	0
1	0	1	Arithmetic shift right	QD	X	1	0	0
1	1	0	Arithmetic shift left	X	0	0	1	0
1	1	1	Parallel load	X	X	1	1	0

CLR is always 0. s1 is just S0 in the outer block. s0 can be minimized with a K-map:

s0			S2	S2
	0	1	1	1
S0	0	0	1	0
		S1	S1	

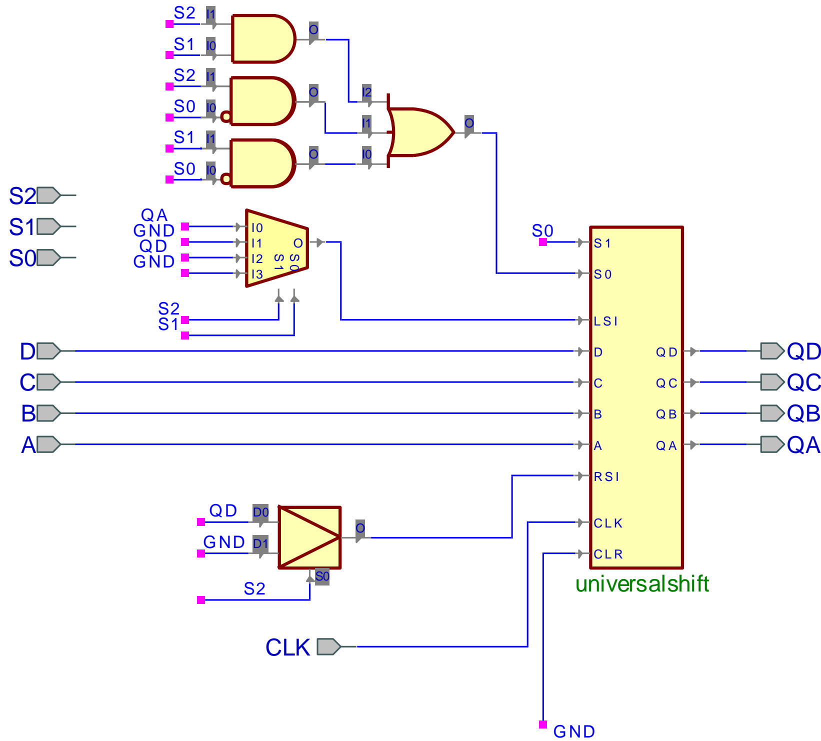
This minimizes to: s0 = S2S1 + S2S0' + S1S0'

We got LSI and RSI by just looking at what the behavior should be for each of the operations. For hold and load we don't care. For shift lefts we don't care about LSI and for right shifts we don't care about RSI. For the logical shifts we always shift in 0's. For the circular shifts we always shift from the opposite end (QA for right shift, QD for left shift). Finally, for arithmetic shifts we either shift in a 0 for the left shift or the value of the high order bit QD for right shift.

Now, how do we get logic for LSI and RSI. RSI is easier, look at the first four rows when S2=0. We have three X's and QD…let's assign all the X's to QD and now we see that when S2=0 then RSI=QD. Similarly, looking at the bottom four rows we can set the X's to 0 and now when S2=1, RSI=0. So we can "choose" between QD and 0 for the input to RSI based on S2…this is exactly what a MUX does! So we use a 2:1 MUX with S2 as the select bit and QD for the I0 and 0 for I1.

For LSI, we need a 4:1 MUX based on S2 and S1. Notice that row pairs 0/1, 2/3, 4/5, and 6/7 each have a “don’t care”. Let’s set the “don’t care” to the other value in the pair: QA/QA, 0/0, QD/QD, X/X. Now we can choose between these four values based on S2 and S1. Thus we use a 4:1 MUX with S2/S1 as the input and QA, 0, QD, and 0 as the inputs.

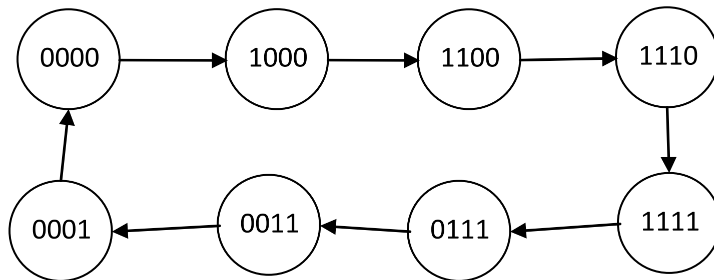
Putting it all together now we get the following circuit:



4. CLD2e, 7.5

This question is asking about a 4-bit Johnson counter. We need to go through the sequential circuit decision process and show that the resulting circuit is that same as the one in Figure 7.2.

The first step is to draw out the state diagram for the counter:



Once we have the state diagram, we can go ahead and draw the state transition table/truth table:

Current				Next			
A	B	C	D	A+	B+	C+	D+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	X	X	X	X
0	0	1	1	0	0	0	1
0	1	0	0	X	X	X	X
0	1	0	1	X	X	X	X
0	1	1	0	X	X	X	X
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0
1	0	0	1	X	X	X	X
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	1	1	1	0
1	1	0	1	X	X	X	X
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

Now from this we can go ahead and minimize our next state logic using K-maps.

A+			A	
	1	X	1	1
	0	X	X	X
C	0	0	0	X
C	X	X	1	X
		B		

B+			A	
	0	X	1	1
	0	X	X	X
C	0	0	1	X
C	X	X	1	X
		B		

C+			A	
	0	X	1	0
	0	X	X	X
C	0	1	1	X
C	X	X	1	X
		B		

D+			A	
	0	X	0	0
	0	X	X	X
C	1	1	1	X
C	X	X	1	X
		B		

(D spans the middle two rows of each map.)

Giving us the following minimized functions:

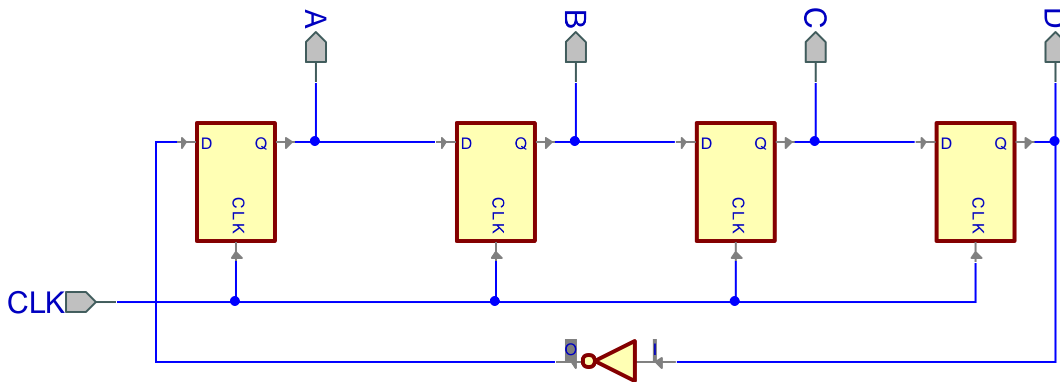
A+ = D'

B+ = A

C+ = B

D+ = C

Now from this we can go ahead and implement the circuit using D flip-flops and a tiny bit of logic:

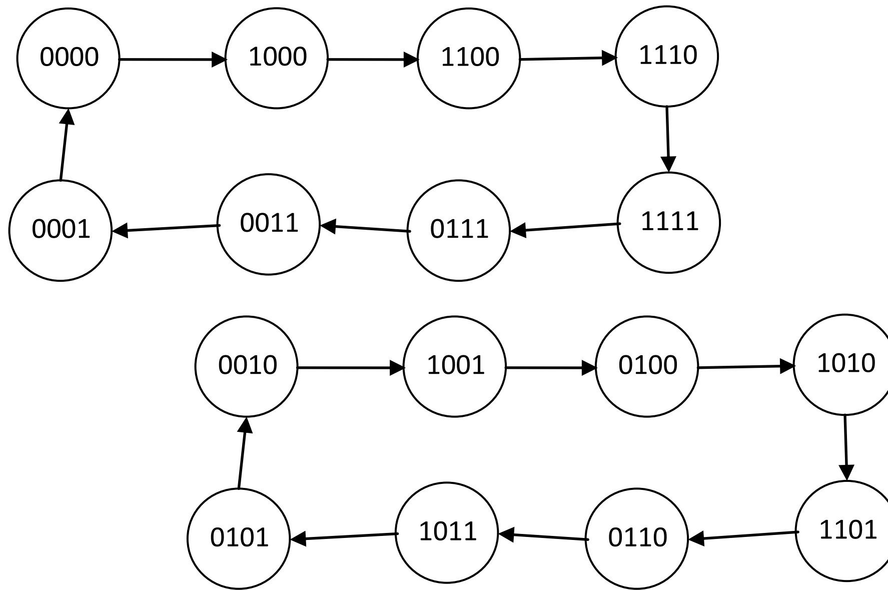


This is the same as Figure 7.2!

5. **CLD2e, 7.6**
In 7.5 we made some decisions about how to assign the “don’t cares” and that led us to a simple circuit at the end. Let’s see how these decisions for the “don’t care” effect the state diagram. This time we will want to draw all 16 potential states that our counter could start out in. The first step is to take our truth table from 7.5, but we will take all the values that were don’t cares and assigned them as we did in or K-maps. This gives the following truth table:

Current				Next			
A	B	C	D	A+	B+	C+	D+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	1	0	1	0	1	1
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	1	0	0	1	1	1	0
1	1	0	1	0	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

Once we have the truth table, we can create the state diagram. This is simply a matter of writing out each state and connecting them to the appropriate next state. Here is the result (with all the states laid out nicely):



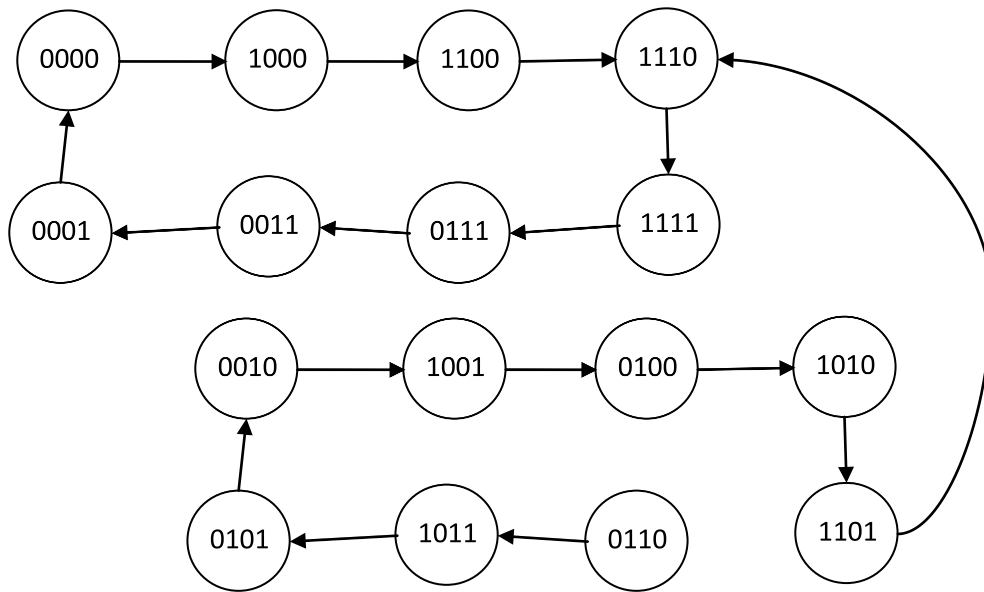
We can see that this state diagram has two loops, the one we want and the one that would result if we started in an invalid state. Therefore, it is **not self starting**.

To fix this, we need to "break the loop", our only flexibility is in how we assigned the "don't cares". To do this we need to look at the un-shaded rows in the truth table and change one of the 1's to 0's or vice versa so that the next state is a valid counter state. I am going to choose the last un-shaded row 1101 → 0110. I will change the first 0 to a 1, giving 1101 → 1110. Now I have to recalculate the function for A+:

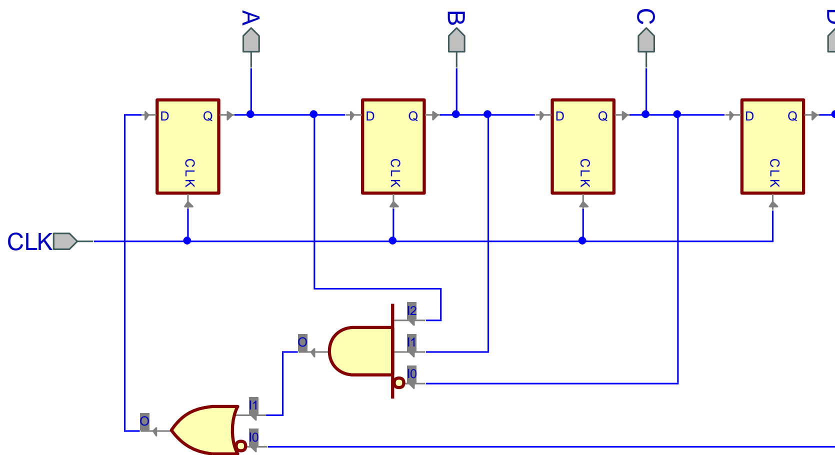
A+			A	A	
	1	1	1	1	
	0	0	1	0	D
C	0	0	0	0	D
C	1	1	1	1	
		B	B		

A+ = D + ABC'

Giving us the following new self-starting state diagram:

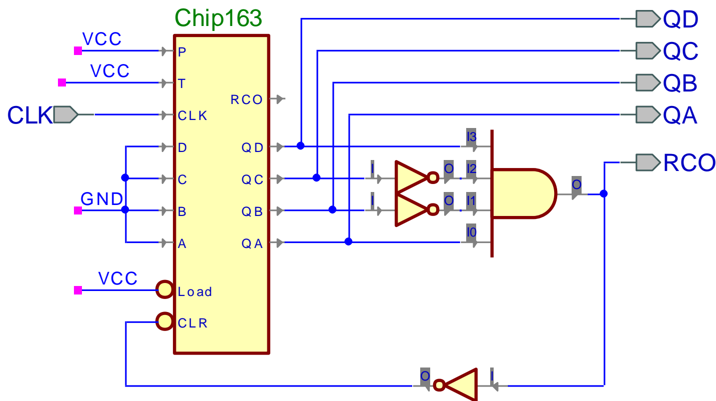


…And new self-starting circuit:



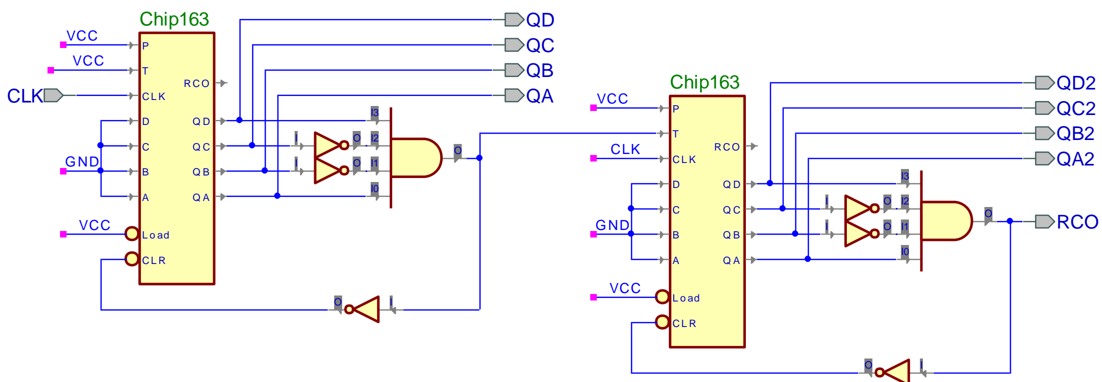
6. **CLD2e, 7.9**

A BCD counter is just a binary counter that goes from 0-9 and then starts back around at 0 again. For this problem, it ends up that the book has almost the exact thing that we need. Basically, we need a variation on Figure 7.21 "Counter with limit". This counter works because when it reaches the limit value the CLR goes active (0) and so the counter goes back to 0. We can do the same thing where we go back to 0, after the output in 9 (1'b1001). This gives the following circuit diagram.



Notice that I also added logic for the RCO output. This will go high when we are about to go back to 0, which is the exact behavior we want.

We can now chain several BCD counters together by noticing that the second counter we only want to increment when the first counter goes back to 0. The signal that enables our counter tick is T, and so if we hook up the T for the second counter to the RCO of the first, it second counter will only tick when RCO is high! Giving the following circuit:



This design should generalize to more digits by chaining more of the circuits together.

Note: It is also possible attach the RCO to the CLK input of the next stage. The tradeoff being that using T allows each component to have the same synchronized clock. On the other hand, using CLK lets you get rid of the enable input to your component. In general, clock inputs should only be used for the clock, not for signals! But you may be forced to use the clock though if the component you are using doesn't have an enable input.