Sequential Machines

Up until now, all we have dealt with is static logic. Next we will work with Active (Sequential) logic.

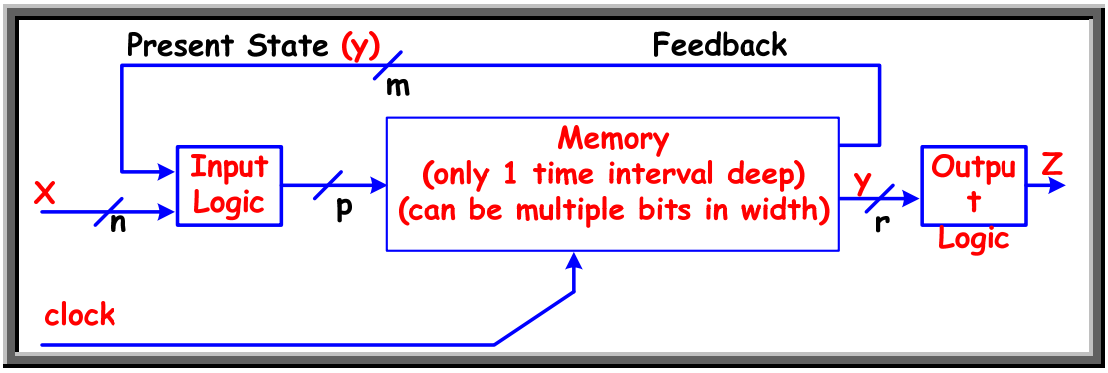


Figure 1 (Moore Machine)

Figure 1 is an example of a **Moore Machine**. In a **Moore Machine**, the output is **purely dependent on the Present State (y)**. Nothing happens until the clock pulse. **Note that the Present State is fed back from the Memory to the Input logic**. Hopefully this makes sense. Why would Present State be the result of feedback? If you want to know where you are or what you are doing, you think back to how you got there.

It is important to note that these outputs belong to the Present State, not to the Next State !

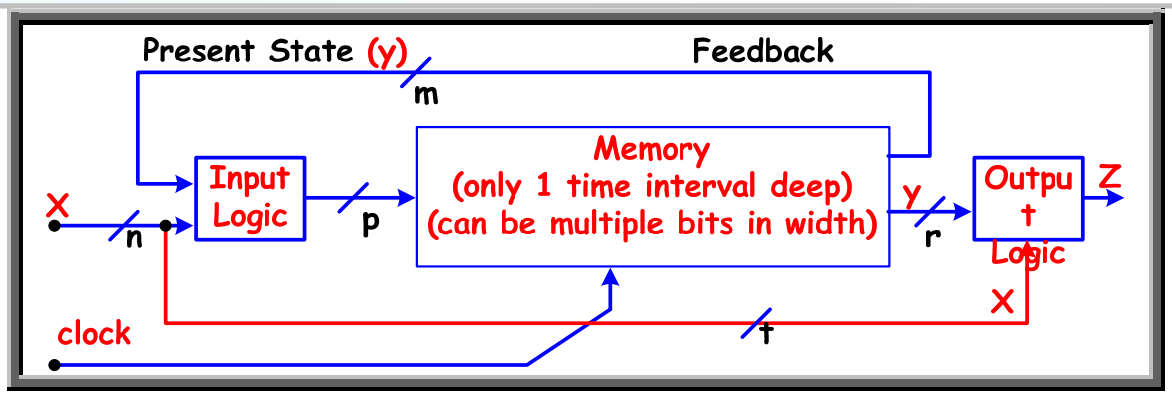


Figure 2 (Mealy Machine)

Figure 2 represents the other simple machine, the **Mealy machine.** The Mealy machine **doesn't just depend just on the present state. It is also a function of the inputs.** Therefore, **it isn't necessarily synchronous since the output can change at any time dependent on the input as well as the present state.**

It is very important in a Mealy model to sample the output only when the circuit has stabilized after an input change.

Note that each circuit has a memory element. This element is naturally made up of some kind of Flip-flop and we will assume that it is either leading or trailing edge triggered. **Which edge is very important to the answer and needs to be determined as early in the problem as possible**.

EXAMPLE 1: Let's look at an example of a simple machine.

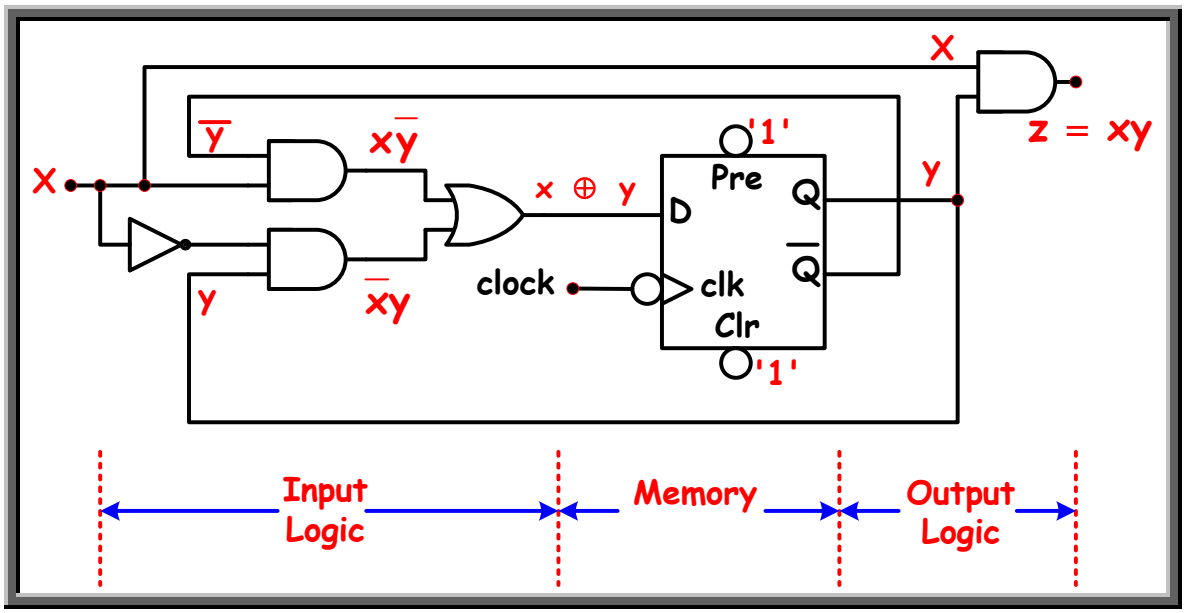
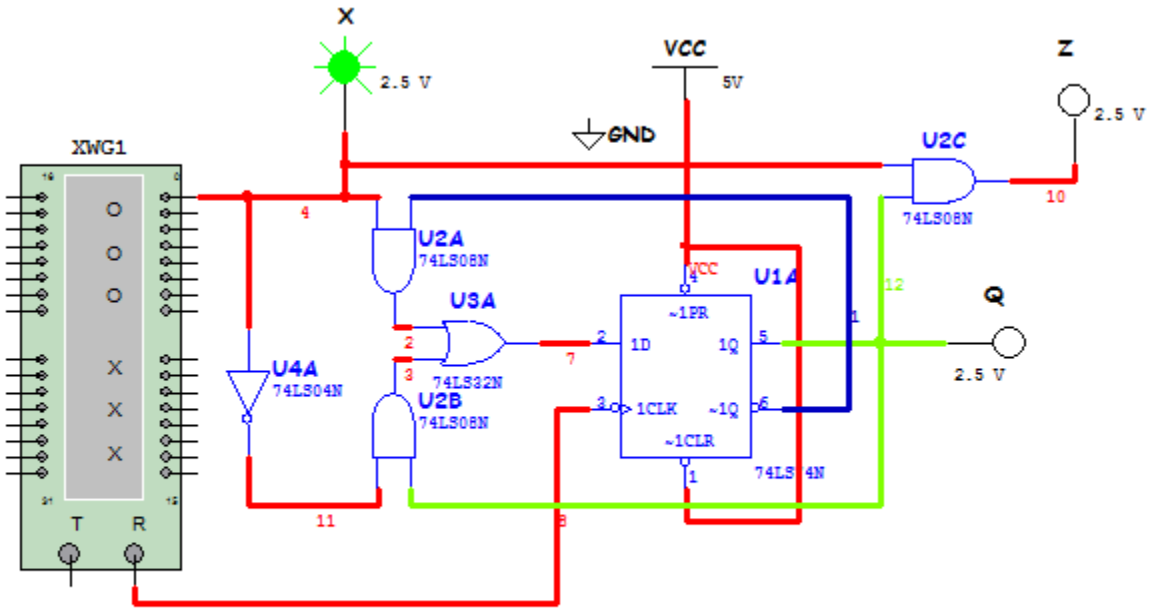


Figure 3: Example 1



Chap8_fig3

Next, we determine the Boolean equations:

$$z = x\ y$$
$$Y = D = x\ \bar{y} + \bar{x}\ y = x \oplus y$$

Q: What kind of machine is this?

A: Since the output (z) depends on the input as well as the present state, this new machine must be a Mealy machine.

Let's build a **state table**:

(The input is ALWAYS the MSB)

As long as the **input (x)** is a **'0'**, the **output (z)** is a **'0'**. However, **when (x) is a 1**, the **output toggles with every clock pulse**.

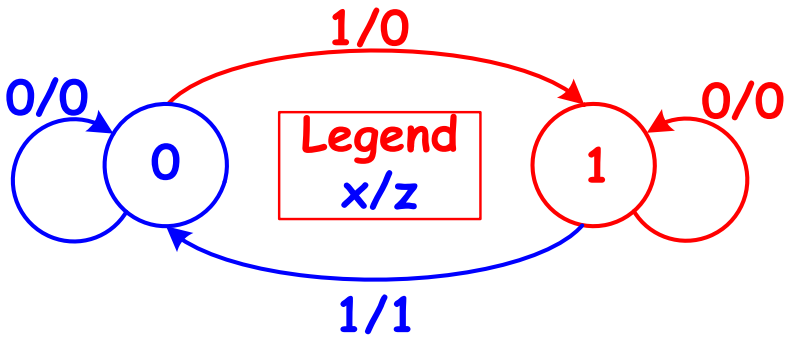
STATE TABLE

x	P.S. y	N.S. Y	z
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
		xor	and

Q: What other device does this describe?

A: The T-flip flop with the additional feature of an output (z).

Next, draw the state diagram. Since we are reverse engineering, we already know what the output states are; **(0, 1).** If we are doing it as a forward engineering problem, we could **assign state variables**, such as (**A**, **B**) vice **(0, 1).**



To economize, the author likes to combine the state table with the k-maps all into one k-map (pg510-516). I believe that this complicates the issue so **DON'T follow his example**!

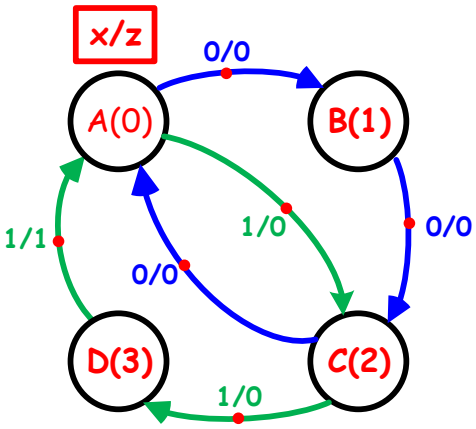
Example 2: Forward Engineering (Synthesis) Example

Specifications:

When x = 0 counts 0 - 1 - 2 - 0 - 1 - .....
When x = 1 counts 3 - 0 - 2 - 3 - 0 - .....
z = 1 when in state 3

Use the following State Assignment Table:

State Variable	State
A	0
B	1
C	2
D	3



State Diagram:

The State Diagram which results from the provided design specifications is provided to the right.

Note that the state diagram plainly demonstrates that the output (z) **BELONGS TO PRESENT STATE.**

Q: Have we forgotten any transitions that the specs didn't cover?
A: Yes

Q: How do we know?
A: Quick check for any missing transitions:

$2^{\#\ inputs}$ = # of arrows which should exit every state
therefore, $2^1 = 2$ exits.

Checking the state machine, it is noted that all the states except state 1 and 3 are correct. These are the only states which only have a single arrow exiting it. All of the others have two, which was required by the equation.

We will treat these transitions as illegal states and define them in the design process.

State Table

Q: How many rows do we need?

A: 4 states x 2 exits per state = 8 arrows = 8 rows

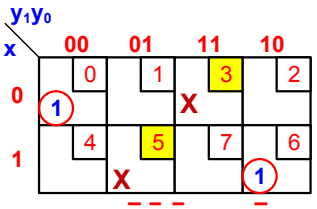
Q: But we only have an (x) and a (y). What do we do?

A: You need 2 flip-flops for 4 states, so you have 2 bit state variables (y_1y_0).

We will start out the design process using two D-ff's to provide the two state variables. Note the presence of the two blank rows in the table resulting from the two states which were not defined in the specifications.

Input	Present state		Next state		Output	Control Circuitry	
x	y_1	y_0	Y_1	Y_0	z	D_1	D_0
0	0	0	0	1	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	0	0
0	1	1				X	X
1	0	0	1	0	0	1	0
1	0	1				X	X
1	1	0	1	1	0	1	1
1	1	1	0	0	1	0	0

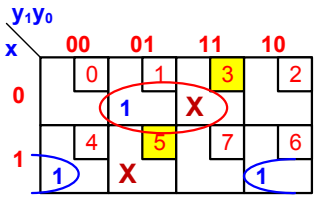
Next, let's perform some k-maps in order to get the Boolean expressions for the output and the next states.



$$D_0 = \bar{x}\,\bar{y}_1\bar{y}_0 + x\bar{y}_1\bar{y}_0$$

However, as you will learn in EET 420, this is also

$$X \oplus y_1 \oplus \bar{y}_0$$

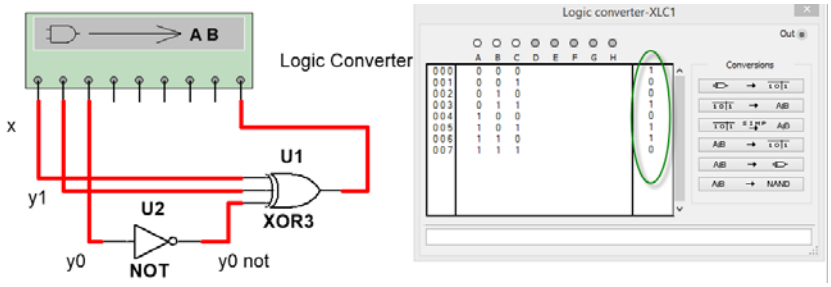


$$D_1 = \bar{x}\,y_0 + x\bar{y}_0 = x \oplus y_0$$

- 1st thing to note here is the presence of the Don't Cares in the illegal states 3 and 5. They will either be used as 0's or as 1's. We will not be able to solve for the equation for Z until we determine how we are going to use the Don't Cares because in order for Z to be high, the system must be in state 3. We will not know all of the state 3's until we determine how we are going to use the Don't Cares.

- The next thing to note is the XOR solution for D_0 if we use the two Don't Cares as 1's. We don't teach XOR k-maps in this course. You will have to wait until EET 420 to recognize that the K-map will result in the given expression. However, let's take a look at a truth table and see if it really does result in this equation.
  - As you should note, the states in the table which have 1's in the D_0 column either have a 1 or an X in the D_0 k-map!

X	y_1	y_0	$\overline{y_0}$	D_0 $x \oplus y_1 \oplus \overline{y_0}$	cell
0	0	0	1	1	0
0	0	1	0	0	1
0	1	0	1	0	2
0	1	1	0	1	3
1	0	0	1	0	4
1	0	1	0	1	5
1	1	0	1	1	6
1	1	1	0	0	7

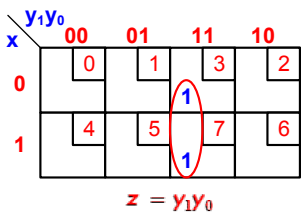


  - We can see the same thing if we use Multisim's Logic Converter to create a state table.

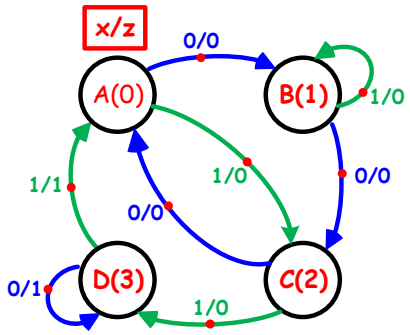
- So, if we intend on using the XOR equation for D_0, then all of the X's in the two k-maps are being used as 1's except for cell 5 in the D_1 k-map which is being used as a 0.
  - Let's now take the state table and make those adjustments.

Input		Present state		Next state			Output	Control Circuitry	
x	y_1		y_0	Y_1		Y_0	z	D_1	D_0
0	0	A	0	0	B	1	0	0	1
0	0	B	1	1	C	0	0	1	0
0	1	C	0	0	A	0	0	0	0
0	1	D	1	1	D	1	1	1	1
1	0	A	0	1	C	0	0	1	0
1	0	B	1	0	B	1	0	0	1
1	1	C	0	1	D	1	0	1	1
1	1	D	1	0	A	0	1	0	0

- Note that since the D column will always look like the next state column, we can also update those blank spaces.
- Finally we can place a 1 in the Z column for present state 3 and find an equation for Z (actually, since Z belongs to present state, we could have done this at the beginning!)



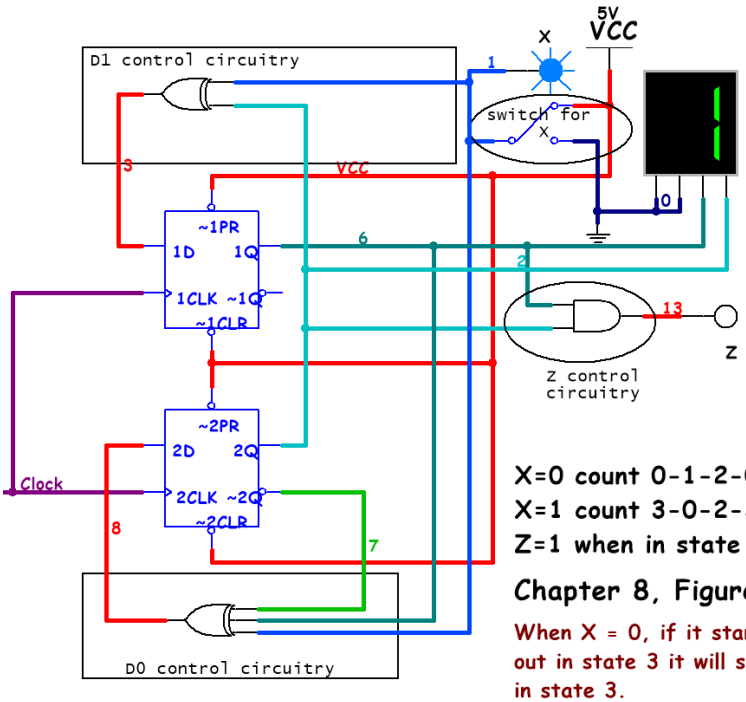
- And now, we add in the two missing arrows onto the state diagram. The resulting circuit follows.



The resulting circuit is shown here.

Q: Is this a Moore or a Mealy machine?

A: Moore. Z does not have any input terms in it.



X=0 count 0-1-2-0-...
X=1 count 3-0-2-3-...
Z=1 when in state 3

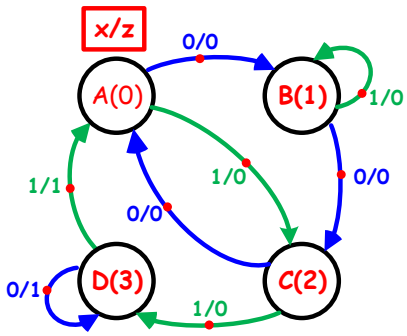
Chapter 8, Figure 8

When X = 0, if it starts out in state 3 it will stay in state 3.

Before you get too excited however, as will be shown on the following page, there is a potential problem with the design results!

HOUSTON, we have a problem!!!

- Let's take a second look at that state diagram:
  - When we do, we note that there are now two potential problems with what we have done. Note that when the input (x) changes from 1 to 0 while the machine is in state 3, it will be stuck in state 3 as long as the input remains at 0.
  - The same thing applies if the input (x) changes from 0 to 1 while the machine is in state 1. It will be stuck there until the input changes. **This MIGHT be a deal breaker with the customer!**



This demonstrates one of the things that you need to watch out for when working with illegal states.

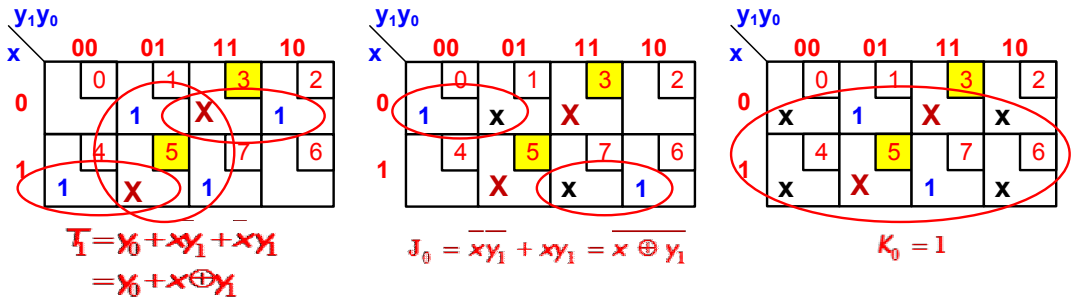
- **The illegal state can't go to itself, and**
- **it can't go to another illegal state which then sends it back to the original illegal state.**

Rather than redesign the k-maps and use all of the Don't Cares as 0's to fix this problem, let's instead change the type of FF's used. Let's rework the example with a **T-flip flop as the MSB** device and a **JK as the LSB**. In order to implement these devices, we need to remember the **T** and **JK excitation tables** below:

y	Y	T_x		y	Y	J_x	K_x
0	0	0		0	0	0	x
0	1	1		0	1	1	x
1	0	1		1	0	x	1
1	1	0		1	1	x	0

With the aid of the excitation tables, a new state table is created. Note that **Z** has not changed since it **belongs to PRESENT STATE**, not next state which is what we are determining.

Input		Present state		Next state			Output	Control Circuitry		
x	y_1		y_0	Y_1		Y_0	z	T_1	J_0	K_0
0	0	A	0	0	B	1	0	0	1	x
0	0	B	1	1	C	0	0	1	x	1
0	1	C	0	0	A	0	0	1	0	x
0	1	D	1				1	x	x	x
1	0	A	0	1	C	0	0	1	0	x
1	0	B	1				0	x	x	x
1	1	C	0	1	D	1	0	0	1	x
1	1	D	1	0	A	0	1	1	x	1



Both of the X's in the T k-map are used as 1's. So, let's show those on the table and view the resulting MSB of the new states.

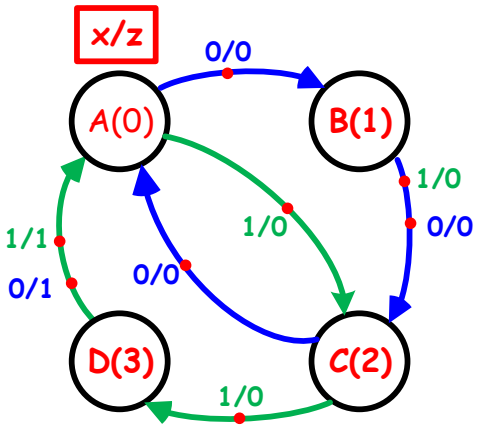
Input		Present state		Next state			Output	Control Circuitry		
x	y_1		y_0	Y_1		Y_0	z	T_1	J_0	K_0
0	0	A	0	0	B	1	0	0	1	x
0	0	B	1	1	C	0	0	1	x	1
0	1	C	0	0	A	0	0	1	0	x
0	1	D	1	0				1	x	x
1	0	A	0	1	C	0	0	1	0	x
1	0	B	1	1				1	x	x
1	1	C	0	1	D	1	0	0	1	x
1	1	D	1	0	A	0	1	1	x	1

Both of the illegal Don't Cares in the J k-map were used as 0's or regular x's depending on what the K term is while both of them in the K k-map were used as 1's or regular x's depending on what the paired J term is.

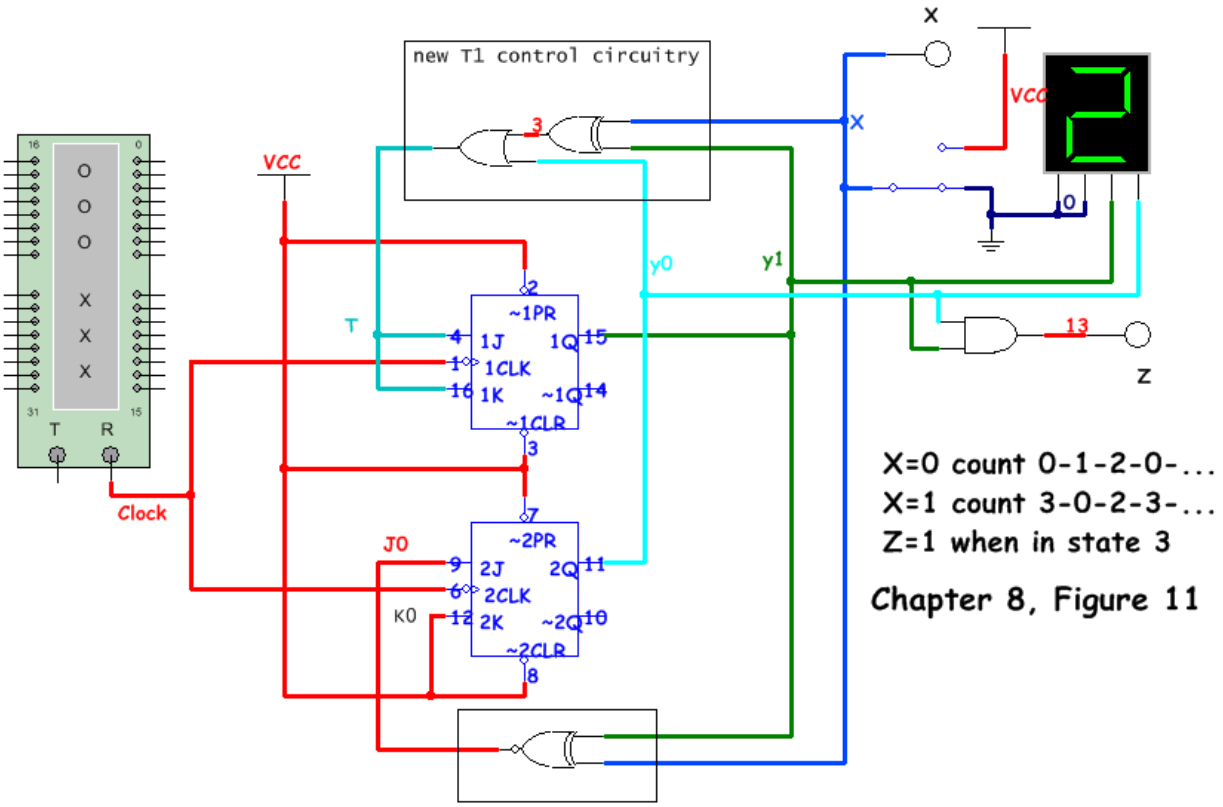
Input		Present state		Next state			Output	Control Circuitry		
x	y_1		y_0	Y_1		Y_0	z	T_1	J_0	K_0
0	0	A	0	0	B	1	0	0	1	x
0	0	B	1	1	C	0	0	1	x	1
0	1	C	0	0	A	0	0	1	0	x
0	1	D	1	0	A	0	1	1	x	1
1	0	A	0	1	C	0	0	1	0	x
1	0	B	1	1	C	0	0	1	x	1
1	1	C	0	1	D	1	0	0	1	x
1	1	D	1	0	A	0	1	1	x	1

Let's build the new State Diagram:

Note that the previous issue where the circuit got stuck while certain conditions occur no longer is an issue!



The hardware ends up looking like:



X=0 count 0-1-2-0-...
X=1 count 3-0-2-3-...
Z=1 when in state 3

Chapter 8, Figure 11