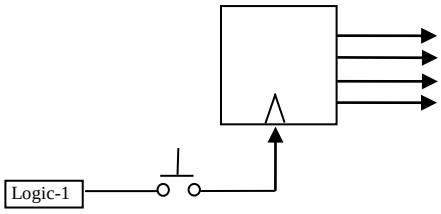
EXAMPLE #1

Design a BCD (Binary Coded Decimal) COUNTER using T flip-flops, and the simplest circuitry possible *(i.e., this means don't have an input, and connect a push-button switch to the Flip-Flops' clocks)***. Also, this counter resets to zero after nine. Don't force the unused states to go anywhere. And as usual, don't try to "minimize the machine," convert to NAND's, or create a Chip Circuit Diagram** *(i.e., only do these things when specifically asked for).*

STEP #1 **DEFINE PROBLEM** (with a Block Diagram)

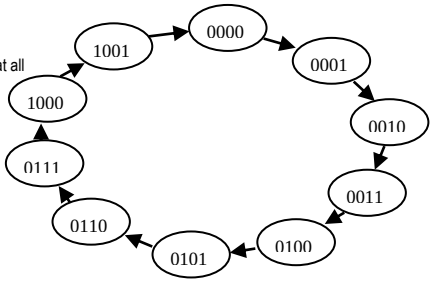
Let's not have an input to trigger each count, but instead let's act as the clock by substituting a push-button switch tied to the positive terminal of our power supply (which is equivalent to a Logic-1)



STEP #2 Draw **STATE DIAGRAM**

No need to label transition-arrows since there are no inputs. And there's only one arrow out of each state because there is no input at all (Thanks to our acting as the clock)

Also, we can use state variables for outputs



STEP #3 **ENCODE** Variables → Already Binary. And we need four State Variables here (A,B,C,D) for the four bits used for BCD

STEP #4 **MINIMIZE FINITE STATE MACHINE** → (not typically needed in this course)

STEP #5 Make **STATE TABLE**

No output columns since we are using state variables for outputs (if you wanted to outputs in the state table, you could, but all four of them would look exactly like the present state)

STEP #6 **APPEND FLIP-FLOP INPUTS** using excitation tables

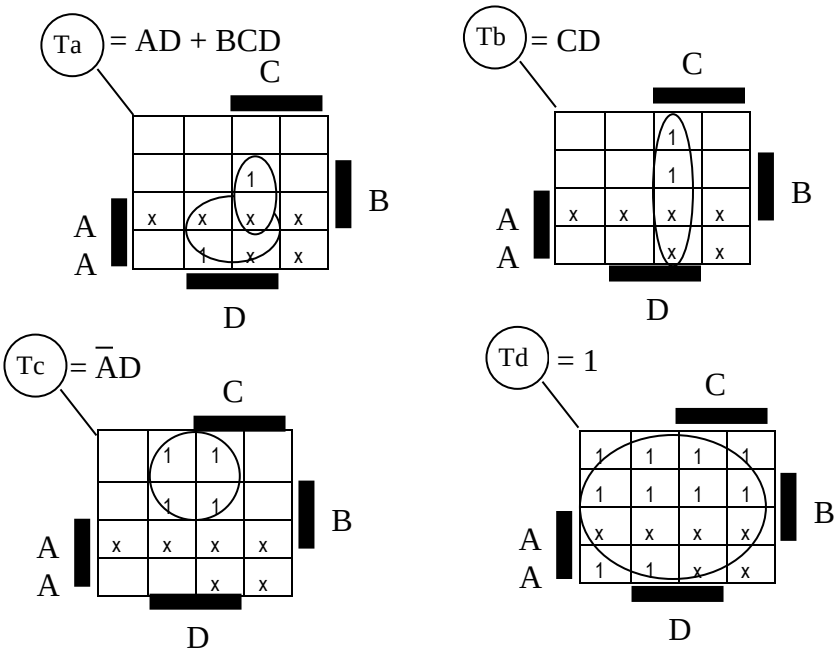
(using **EXCITATION** TABLE for T Flip-Flop for this problem):

Present State Q(t)	Next State Q(t+1)	T Flip-Flop Input to achieve Q(t+1)	
0	**0**	**0**	**No Change**
0	**1**	**1**	**Toggle**
1	**0**	**1**	**Toggle**
1	**1**	**0**	No Change

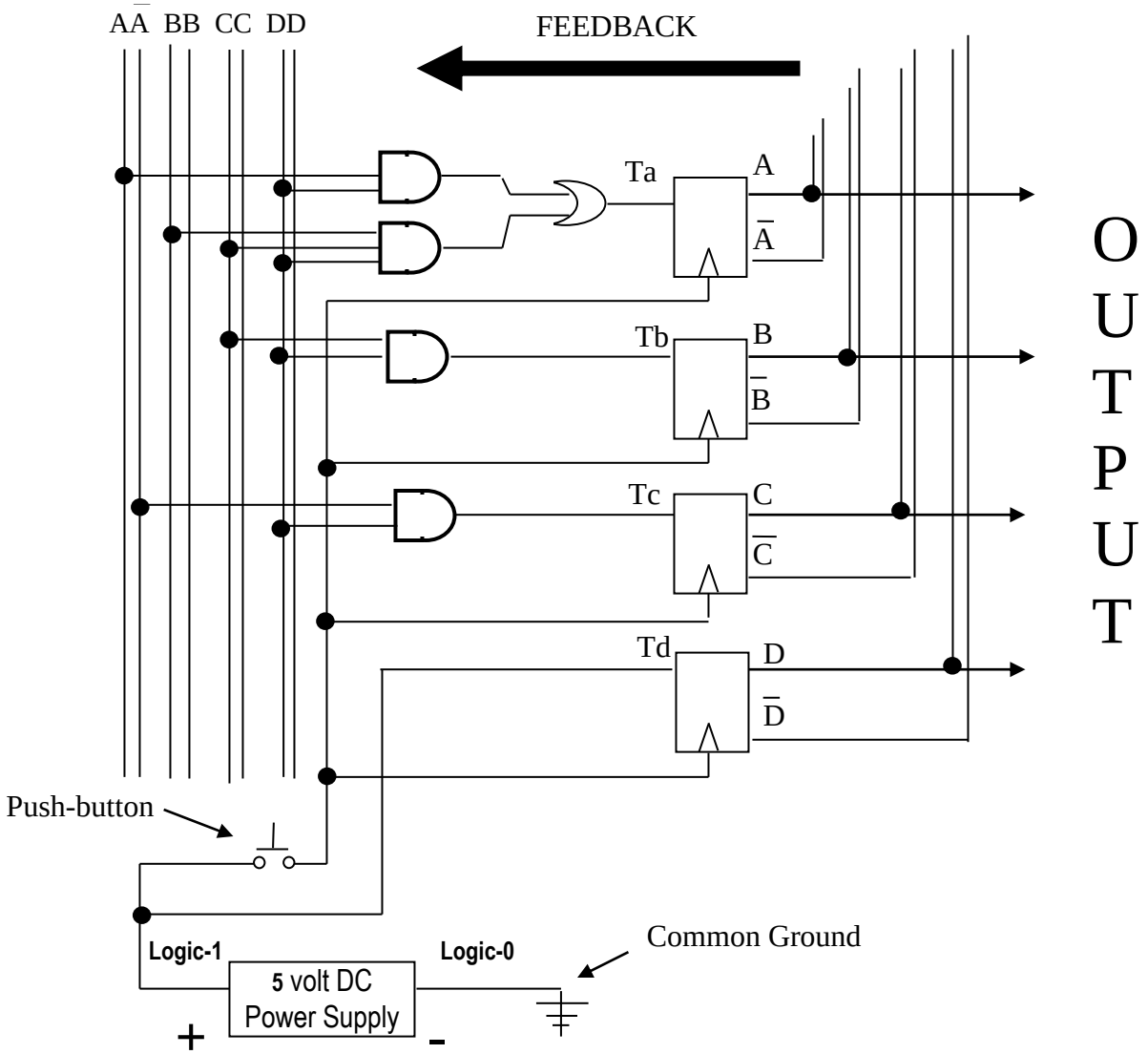
STATE TABLE

PRESENT STATE Q(t)				NEXT STATE Q(t+1)				FLIP-FLOP INPUTS				
A	B	C	D	A	B	C	D	Ta	Tb	Tc	Td	*m*
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	1	0	0	0	1	1	1
0	0	1	0	0	0	1	1	0	0	0	1	2
0	0	1	1	0	1	0	0	0	1	1	1	3
0	1	0	0	0	1	0	1	0	0	0	1	4
0	1	0	1	0	1	1	0	0	0	1	1	5
0	1	1	0	0	1	1	1	0	0	0	1	6
0	1	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	1	0	0	1	0	0	0	1	8
1	0	0	1	0	0	0	0	1	0	0	1	9
1	0	1	0	?	?	?	?	X	X	X	X	10
1	0	1	1	?	?	?	?	X	X	X	X	11
1	1	0	0	?	?	?	?	X	X	X	X	12
1	1	0	1	?	?	?	?	X	X	X	X	13
1	1	1	0	?	?	?	?	X	X	X	X	14
1	1	1	1	?	?	?	?	X	X	X	X	15

STEP #7 **SIMPLIFY** flip-flop inputs using maps



STEP #8 Draw **LOGIC CIRCUIT**



STEP #9 **CONVERT TO NANDS** → Not asked for

STEP #10 **ANALYZE UNUSED STATES** using flip-flop characteristic table and remembering your Flip Flo input functions:

$Ta = AD + BCD$

$Tb = CD$

$Tc = \overline{A}D$

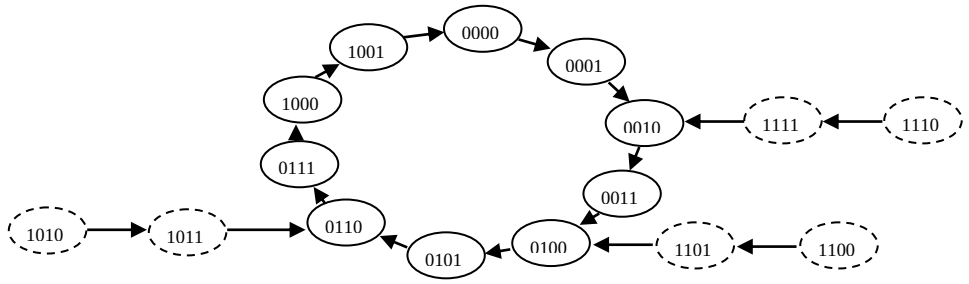
$Td = 1$

After you fill in the part of the table evaluating all Flip-Flop Inputs, use the T Flip-Flop **CHARACTERISTIC** TABLE to look at each State Variable:

Flip-Flop Input T	What happens to State Variable at next clock edge Q(t+1)	
0	Q(t)	No Change
1	$\overline{Q(t)}$	Toggle

PRESENT STATE Q(t)				CALCULATE FLIP-FLOP INPUTS				NEXT STATE Q(t+1)			
A	B	C	D	Ta	Tb	Tc	Td	A	B	C	D
1	0	1	0	0	0	0	1	1	0	1	1
1	0	1	1	1	1	0	1	0	1	1	0
1	1	0	0	0	0	0	1	1	1	0	1
1	1	0	1	1	0	0	1	0	1	0	0
1	1	1	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	0	1	0	0	1	0

STEP #11 **RE-DRAW STATE DIAGRAM TO SHOW UNUSED STATES**



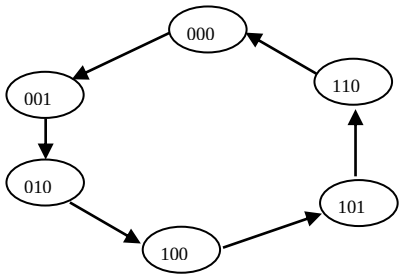
Don't forget that **THESE UNUSED STATES ARE ERROR STATES** and therefore you might want to add an output to your entire machine (at the very beginning of your problem definition), that you would indicate on every arrow as " - /0" or "- /1" where the "-" means that there is no input triggering the transition, and the "0" or "1" would indicate whether or not you were transitioning out of a used state ("0") or unused state ("1")

STEP #12 **CHIP CIRCUIT DIAGRAM** (not asked for)

STEP #13 **REVIEW ASSUMPTIONS** → None made (other than we don't want to force the unused state anywhere for the given problem – however, in reality you probably want to force your machine to reset if an unused state occurs – i.e., force it back to the initial state because the unused state is most likely an error.

EXAMPLE #2

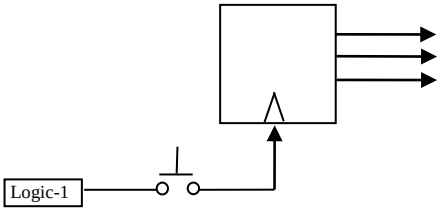
GIVEN THE FOLLOWING STATE DIAGRAM, state in words what it is doing, then draw the block diagram, then continue all the design steps. Use JK flip-flops, and use the simplest circuitry possible *(i.e., this means don't have an input, and connect a push-button switch to the Flip-Flops)***. And as usual, don't try to "minimize the machine," convert to NAND's, or create a Chip Circuit Diagram** ***(i.e., only do these things when specifically asked for). Note that there's no good reason for the arrows to go counterclockwise here, but it that it just doesn't matter to the functioning of the machine.***



In words, what this is doing: This is a counter that counts from 000 to 001 to 010 to 100 to 101 to 110, then resets to 000

STEP #1 **DEFINE PROBLEM** (with a Block Diagram)

Let's not have an input to trigger each count, but instead let's act as the clock by substituting
a push-button switch tied to the positive terminal of our power supply (which is equivalent to a Logic-1)



STEP #2 Draw **STATE DIAGRAM**. GIVEN

STEP #3 **ENCODE VARIABLES** → already Binary. And we need three State Variables (A,B,C) for the three bits used in the given count

STEP #4 **MINIMIZE FINITE MACHINE** → (not done in this course)

STEP #5 Make **STATE TABLE**

No need to label transition-arrows since there are no inputs. And there's only one arrow out of each state because there is no input at al (Thanks to our acting as the clock)

STEP #6 **APPEND FLIP-FLOP INPUTS** using excitation tables

(using **EXCITATION** TABLE for JK Flip-Flop for this problem):

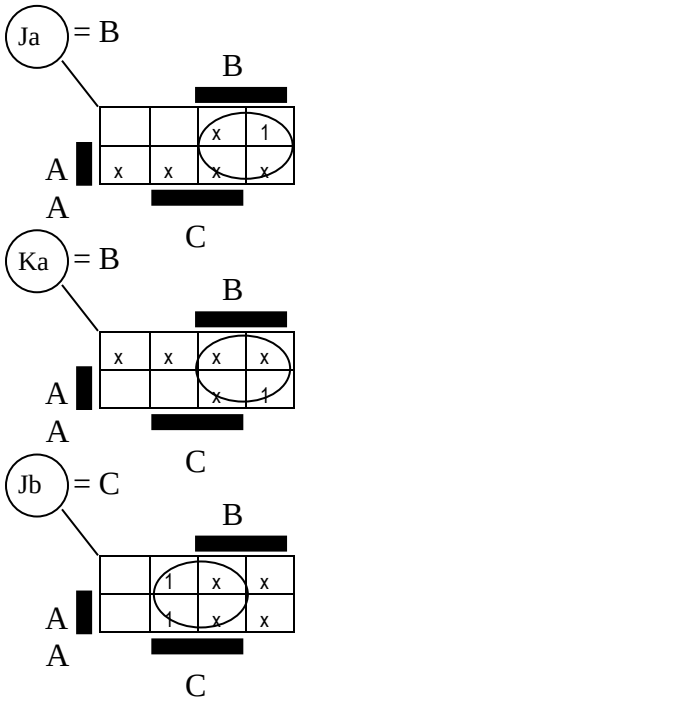
Present State Q(t)	Next State Q(t+1)	JK Flip-Flop Inputs to achieve Q(t+1)	
0	0	0 X	**No Change (00) or Reset (01)**
0	1	1 X	**Set (10) or Toggle (11)**
1	0	X 1	**Reset (01) or Toggle (11)**
1	1	X 0	**No Change or Set**

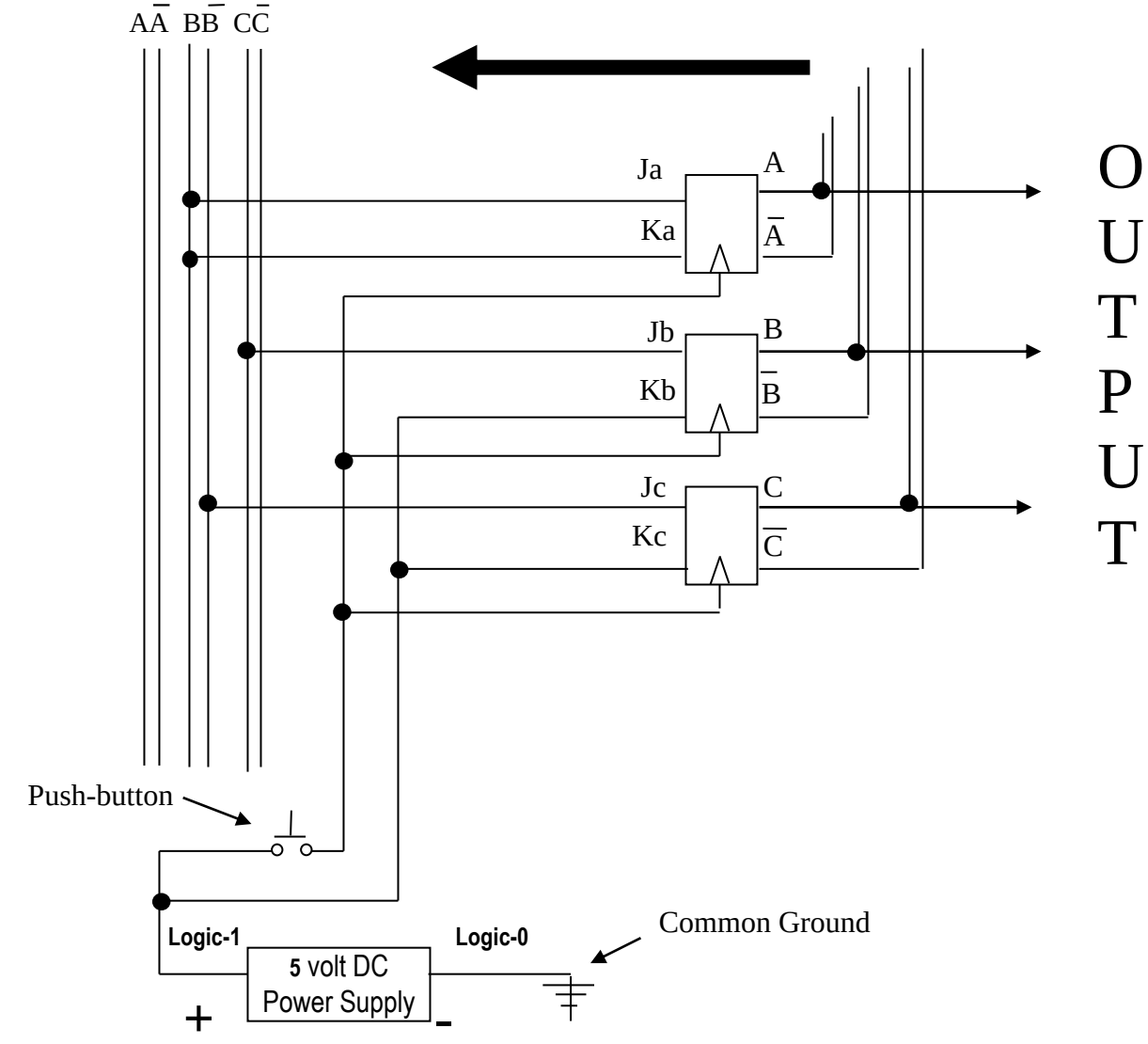
STATE TABLE

PRESENT STATE Q(t)			NEXT STATE Q(t+1)			FLIP-FLOP INPUTS						
A	B	C	A	B	C	Ja	Ka	Jb	Kb	Jc	Kc	*m*
0	0	0	0	0	1	0	X	0	X	1	X	0
0	0	1	0	1	0	0	X	1	X	X	1	1
0	1	0	1	0	0	1	X	X	1	0	X	2
0	1	1	?	?	?	X	X	X	X	X	X	3
1	0	0	1	0	1	X	0	0	X	1	X	4
1	0	1	1	1	0	X	0	1	X	X	1	5
1	1	0	0	0	0	X	1	X	1	0	X	6
1	1	1	?	?	?	X	X	X	X	X	X	7

Don't move the unused states from the row corresponding to the minterm ("*m*") that represents it; this is to preserve the integrity of the Simplification Maps and the order of all the cells within them

STEP #7 **SIMPLIFY** flip-flop inputs using maps





STEP #9 **CONVERT TO NANDS** → not asked for

STEP #10 **ANALYZE UNUSED STATES** using flip-flop characteristic table and remembering your Flip Flop input functions:

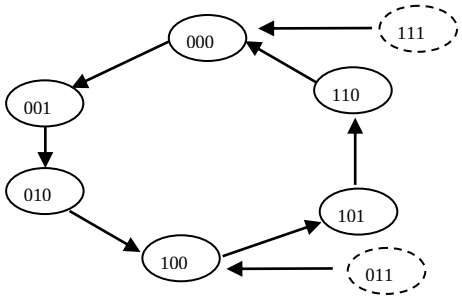
Ja = B
Ka = B
Jb = C
Kb = 1
Jc = $\overline{B}$
Kc = 1

After you fill in the part of the table evaluating all Flip-Flop Inputs use the JK Flip-Flop **CHARACTERISTIC** TABLE to look at each State Variable:

Flip-Flop Inputs J K	What happens to State Variable at next clock edge Q(t+1)	
0 0	Q(t)	No Change
0 1	0	Reset
1 0	1	Set
1 1	$\overline{Q(t)}$	Toggle

PRESENT STATE Q(t)			CALCULATE FLIP-FLOP INPUTS						NEXT STATE Q(t+1)		
A	B	C	Ja	Ka	Jb	Kb	Jc	Kc	A	B	C
1	1	1	1	1	1	1	0	1	0	0	0
0	1	1	1	1	1	1	0	1	1	0	0

STEP #11 **RE-DRAW STATE DIAGRAM TO SHOW UNUSED STATES**



Don't forget that **THESE UNUSED STATES ARE ERROR STATES** and therefore you might want to add an output to your entire machine (at the very beginning of your problem definition), that you would indicate on every arrow as " - /0" or "- /1" where the "-" means that there is no input triggering the transition, and the "0" or "1" would indicate whether or not you were transitioning out of a used state ("0") or unused state ("1")

STEP #12 **CHIP CIRCUIT DIAGRAM** (not asked for)

STEP #13 **REVIEW ASSUMPTIONS** → None made (other than we don't want to force the unused state anywhere for the given problem – however, in reality you probably want to force your machine to reset if an unused state occurs – i.e., force it back to the initial state because the unused state is most likely an error)