ECED2200 – Digital Circuits

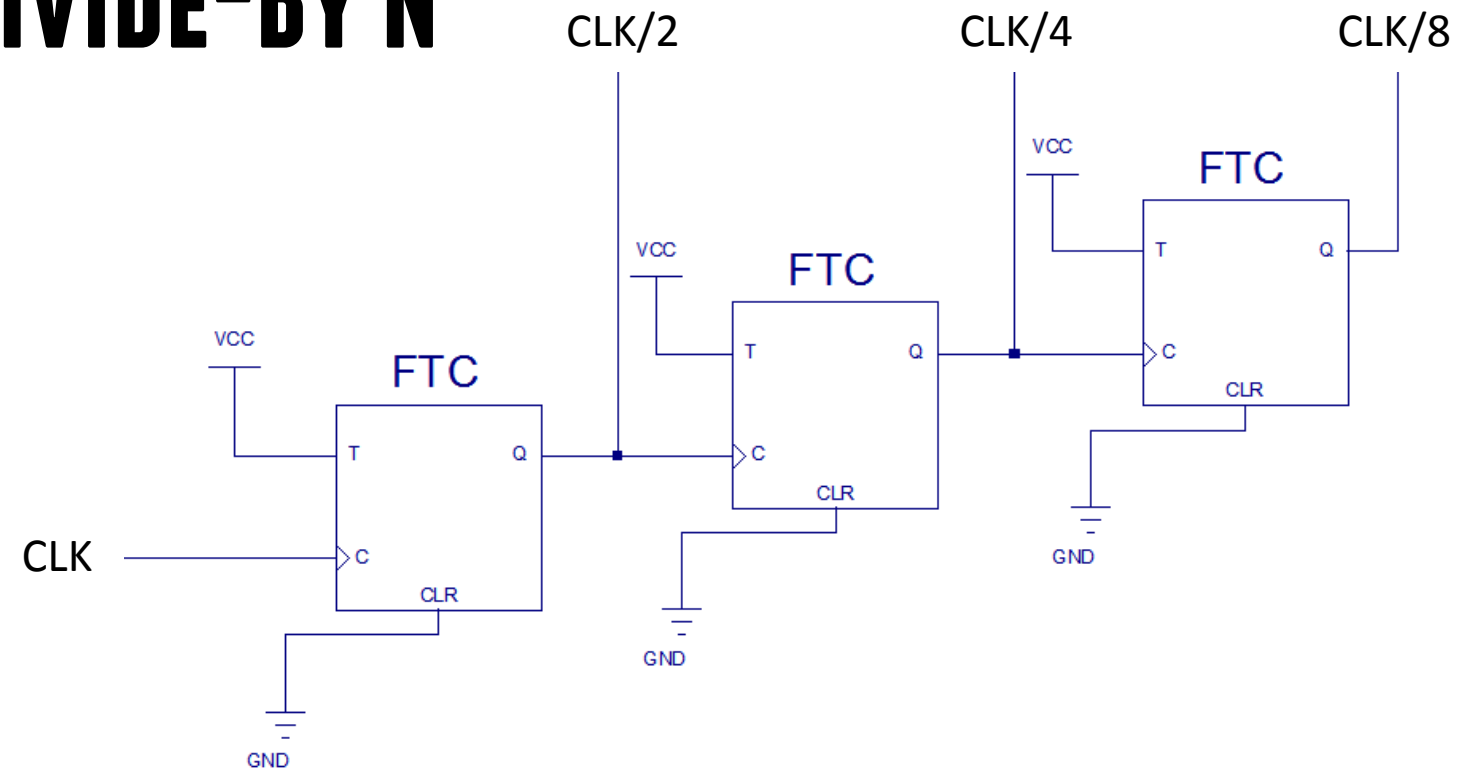
Counters

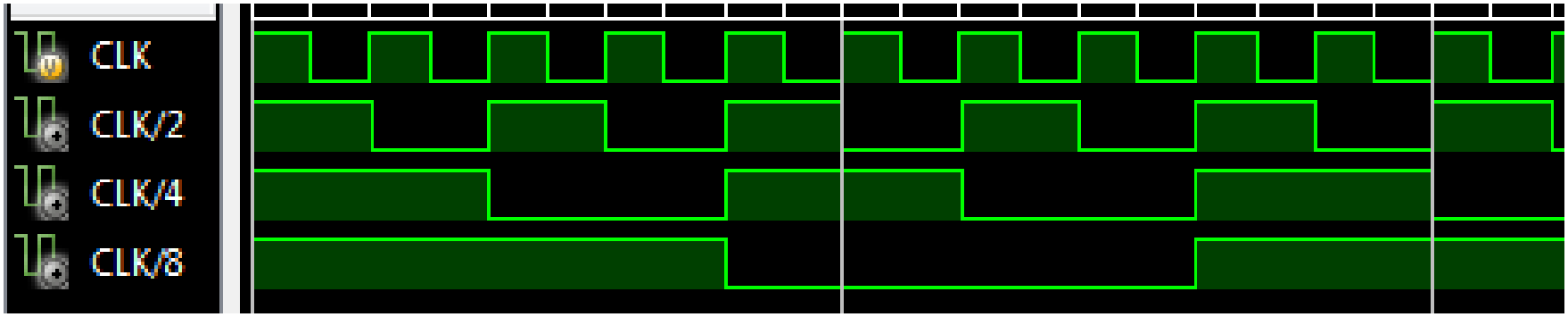
General Notes

- See updates to these slides: www.newae.com/teaching
- These slides licensed under 'Creative Commons Attribution-ShareAlike 3.0 Unported License'
- These slides are not the complete course – they are extended in-class
- You will find the following references useful, see www.newae.com/teaching for more information/links:
  - The book "Bebop to the Boolean Boogie" which is available to Dalhousie Students
  - Course notes (covers almost everything we will discuss in class)
  - Various websites such as e.g.: www.play-hookey.com
  - The book "Contemporary Logic Design", which was used in previous iterations of the class and you may have already

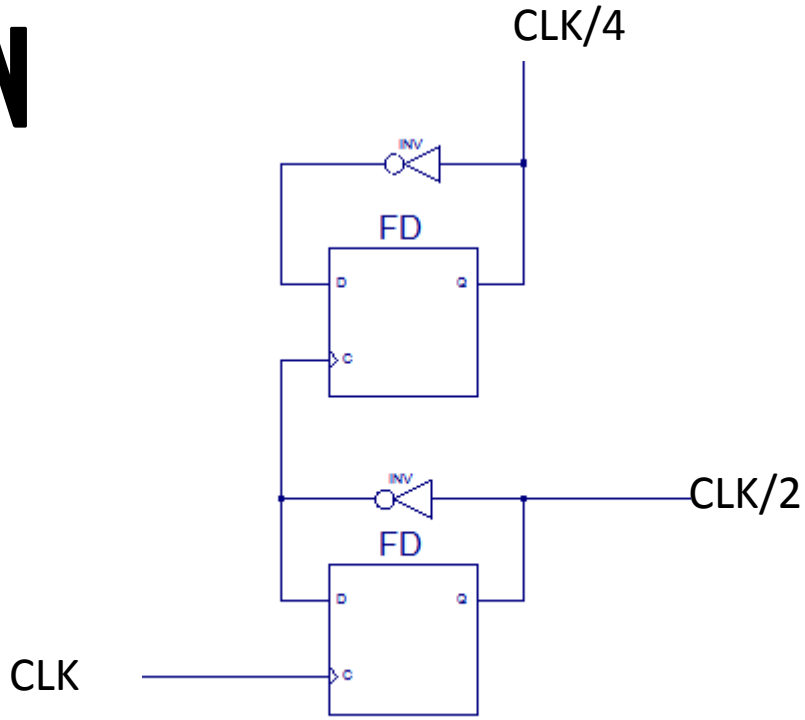
DIVIDE-BY N COUNTER

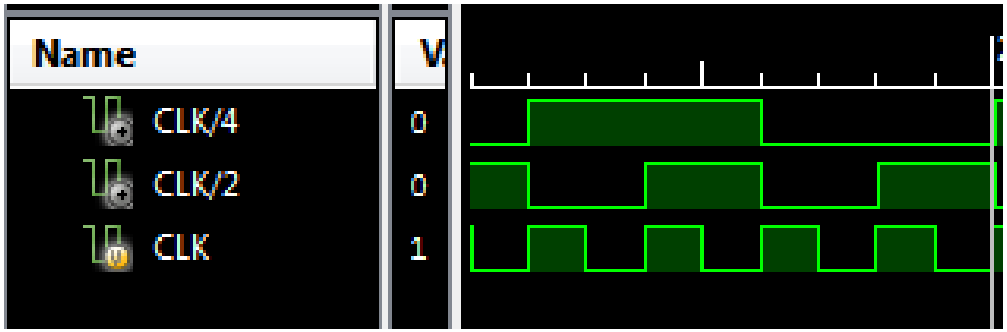
Divide-By N



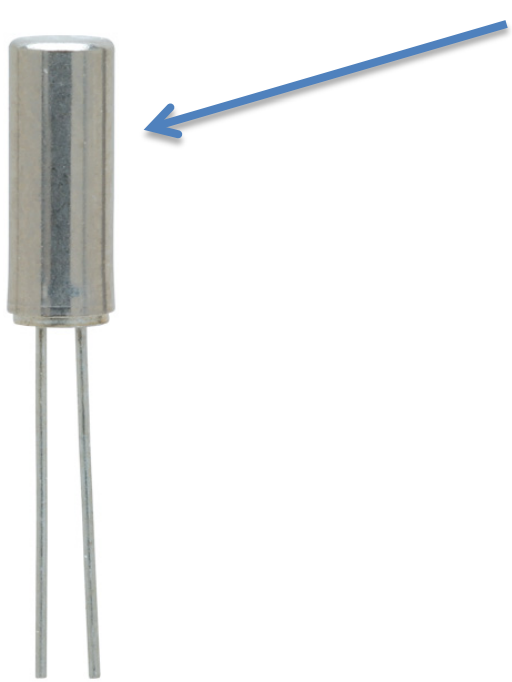


Divide-By N

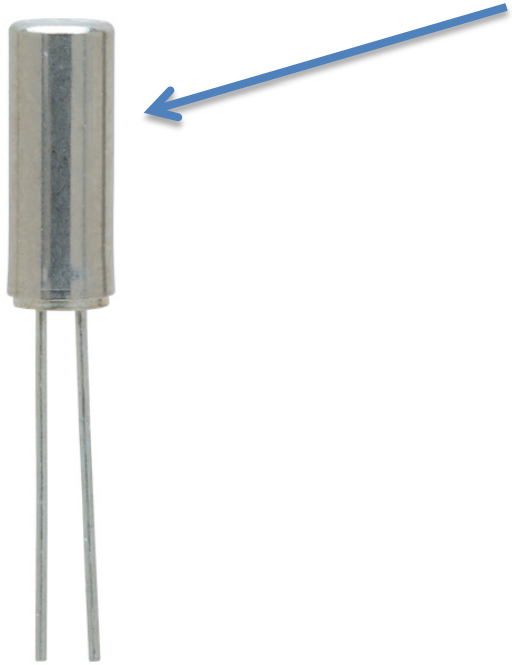




Uses of Divide-By-N



Uses of Divide-By-N

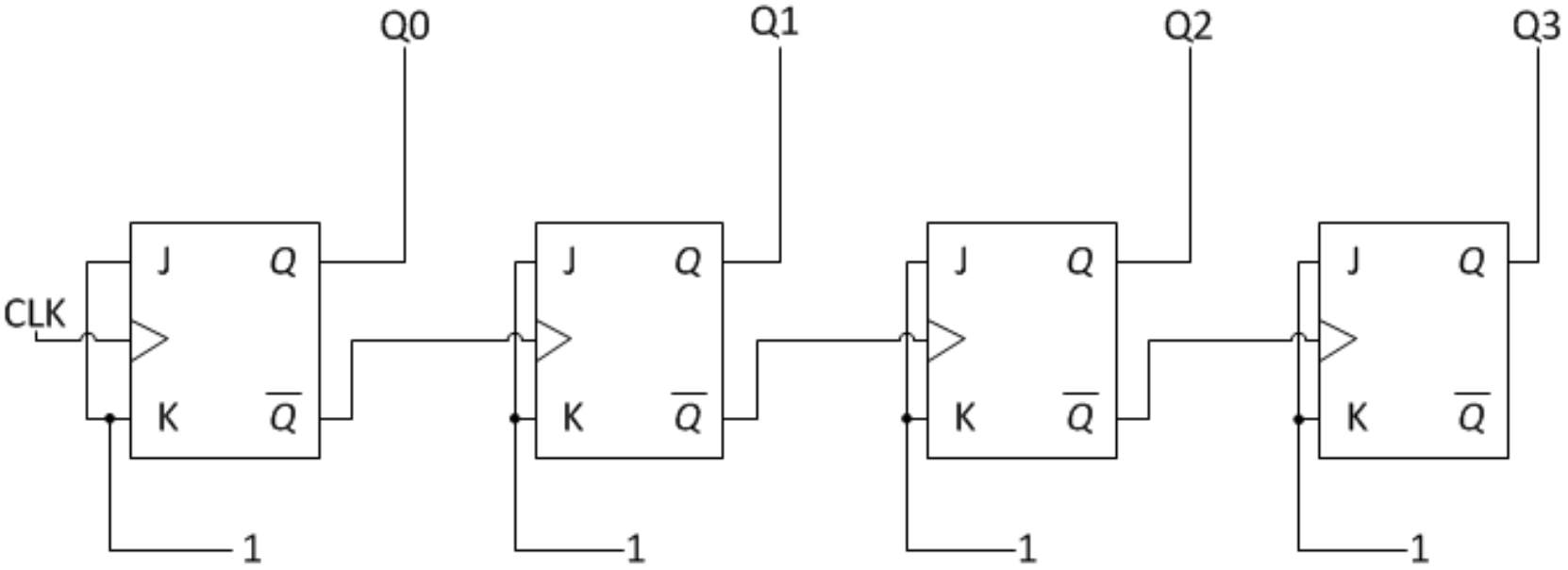


32.768 KHz crystal

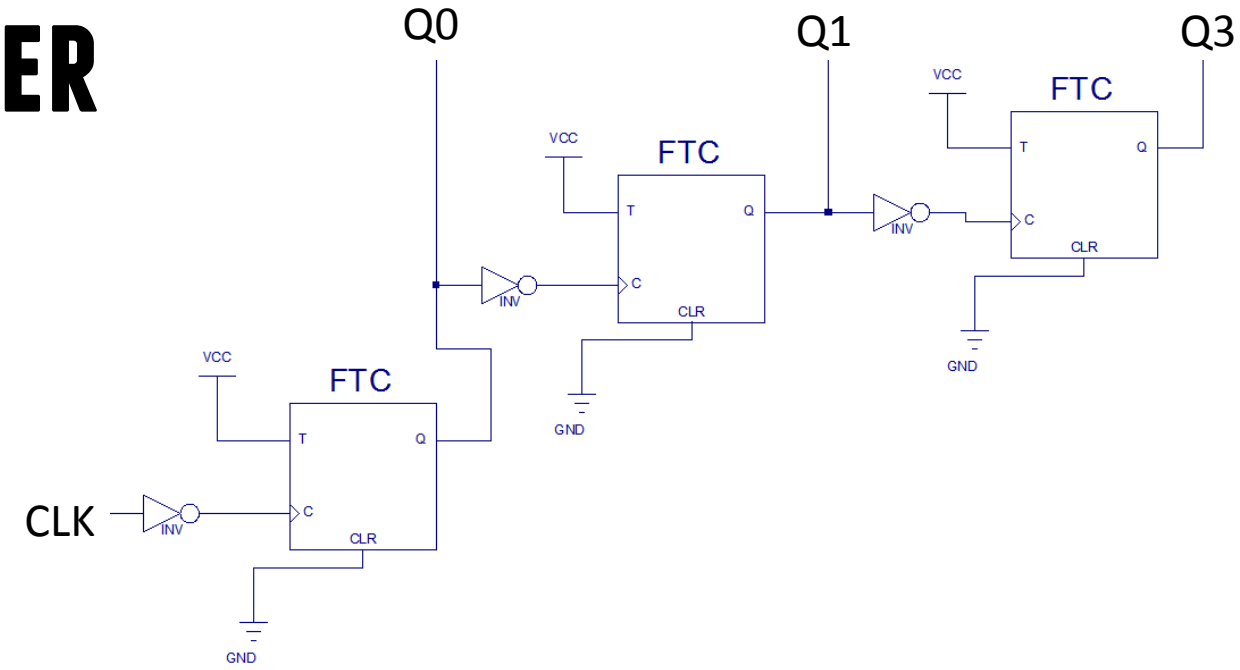
$2^{15} = 32768$

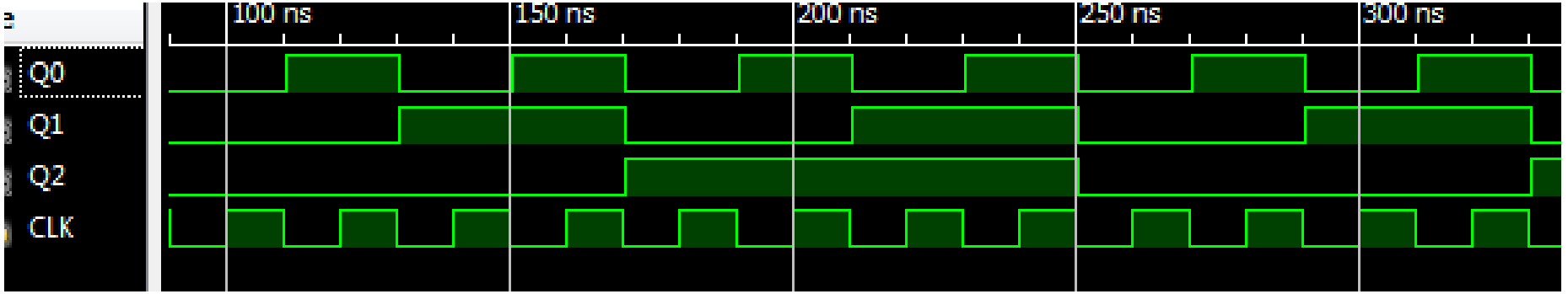
BINARY RIPPLE COUNTER

Binary Ripple Design

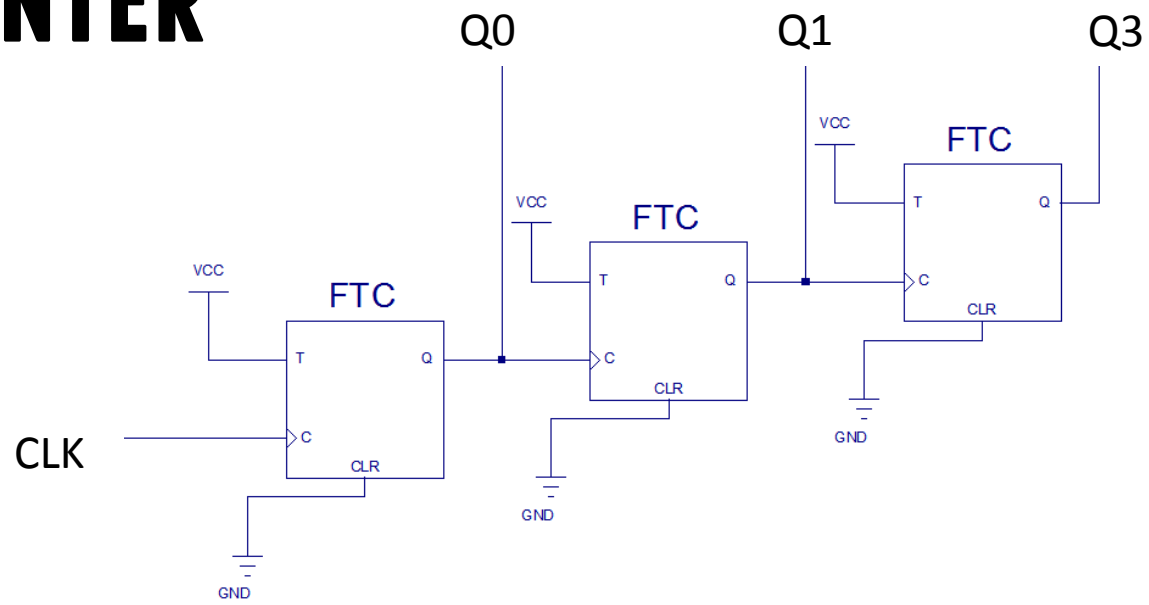


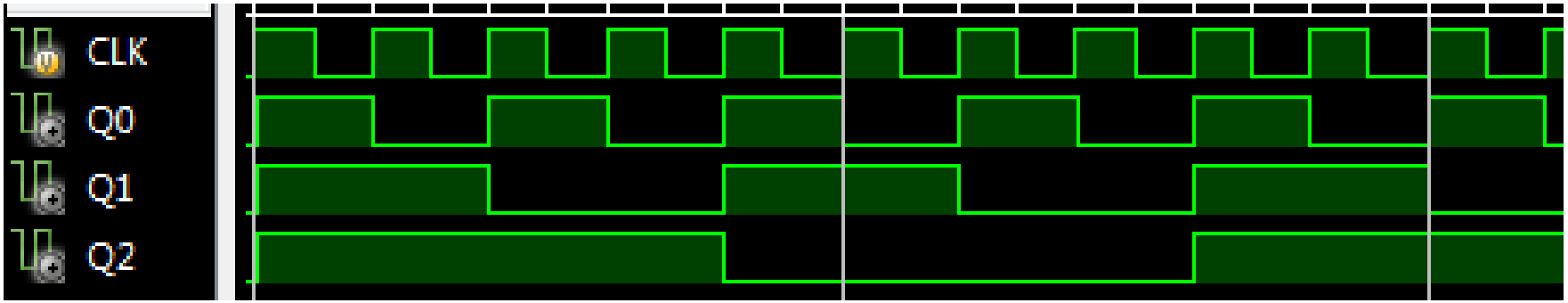
Up Counter



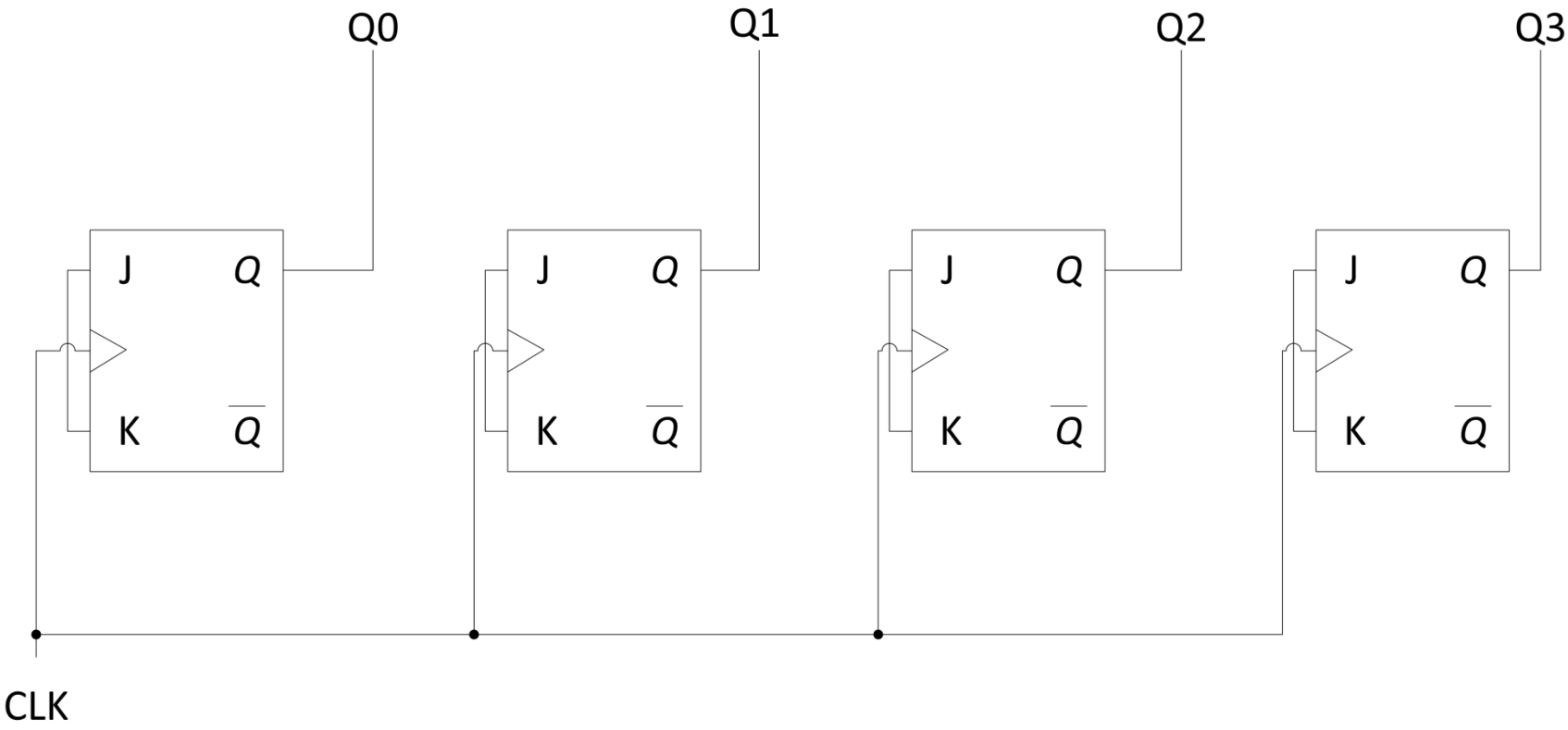


Down Counter



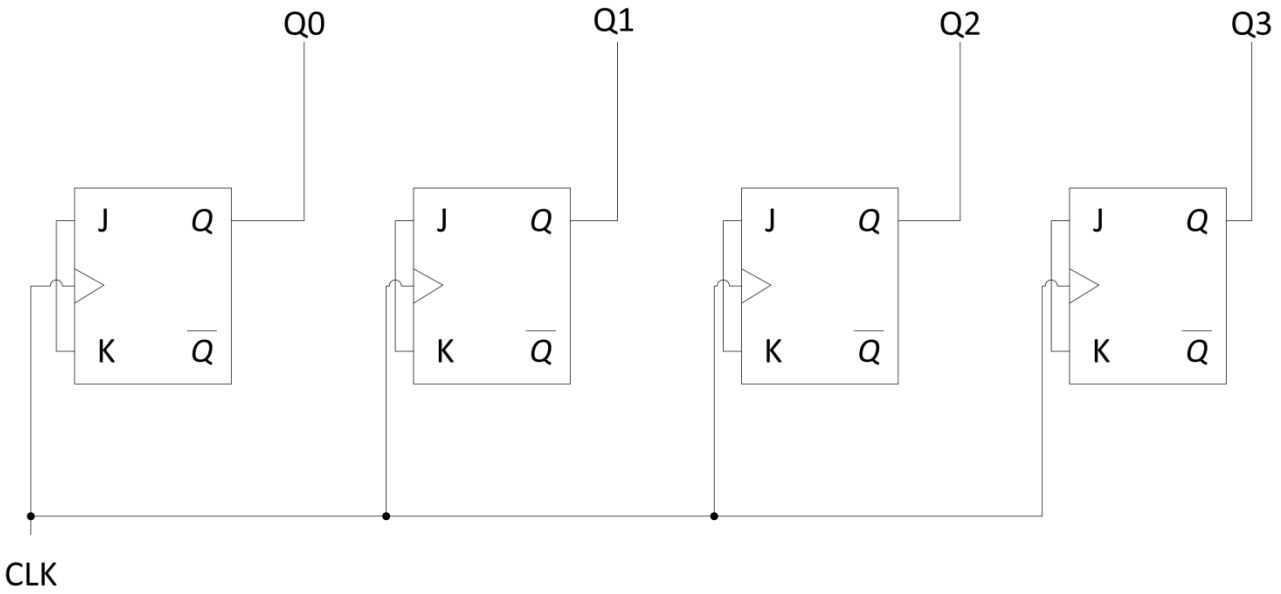


SYNCHRONOUS COUNTER

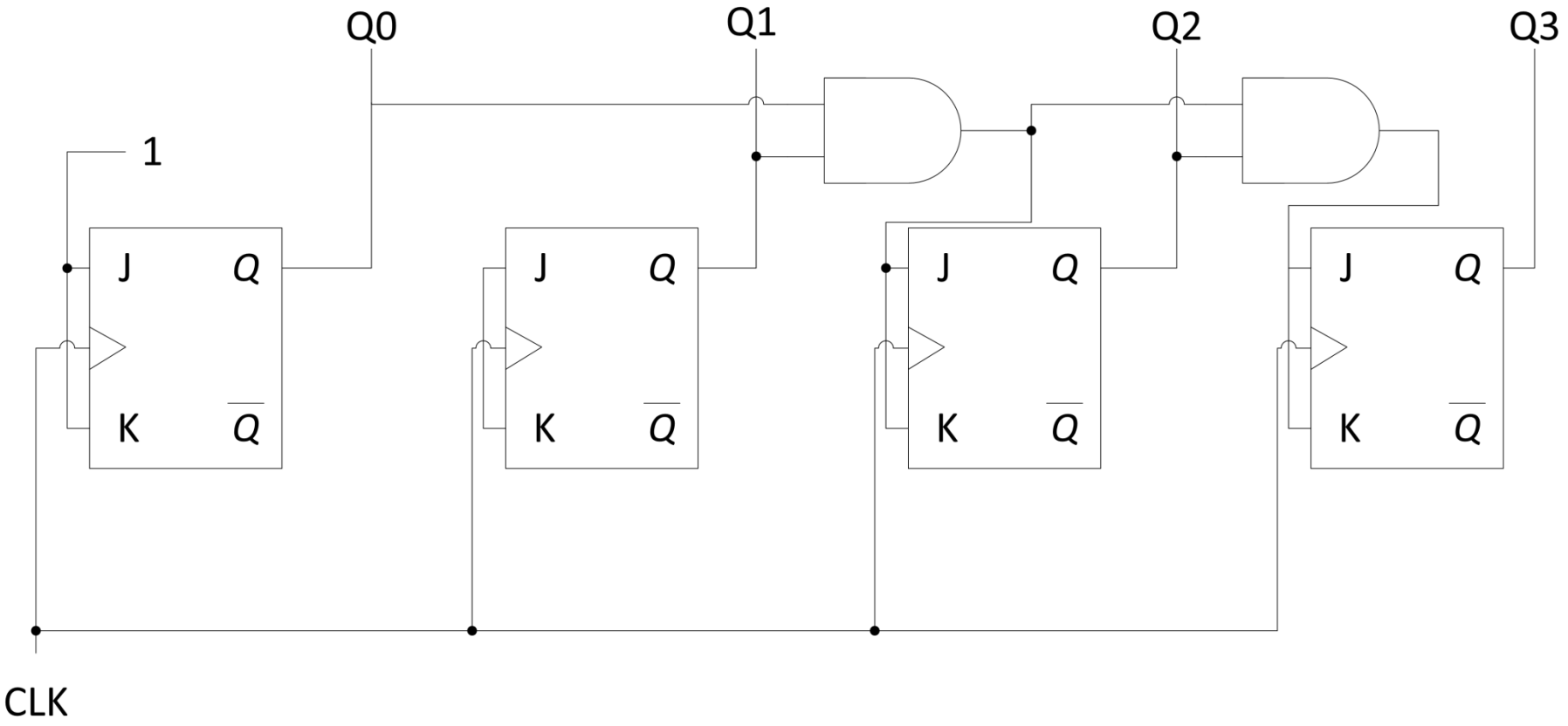


Count Sequence?

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0

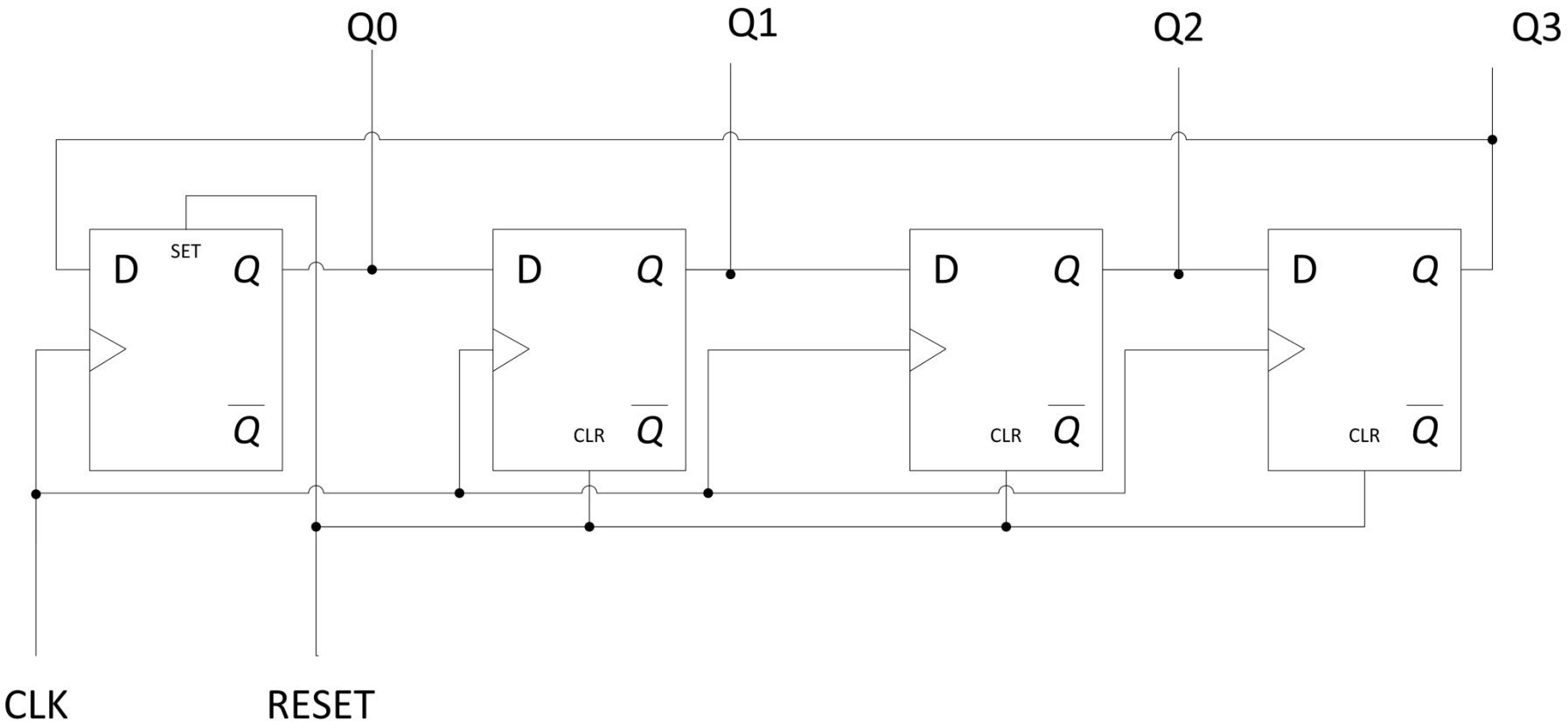


Synchronous Counter Design



RING COUNTER

Design



DESIGN PROCEDURE FOR ARBITRARY COUNTERS

Design Specs

0 0 0

0 1 0

0 1 1

1 0 1

1 1 0

(0, 2, 3, 5, 6)

Procedure

1. Draw state transition diagram
2. Write state transition table
3. Choose flip-flops & determine required inputs
4. Design logic to realize required inputs

Counter Design Example

0 0 0

0 1 0

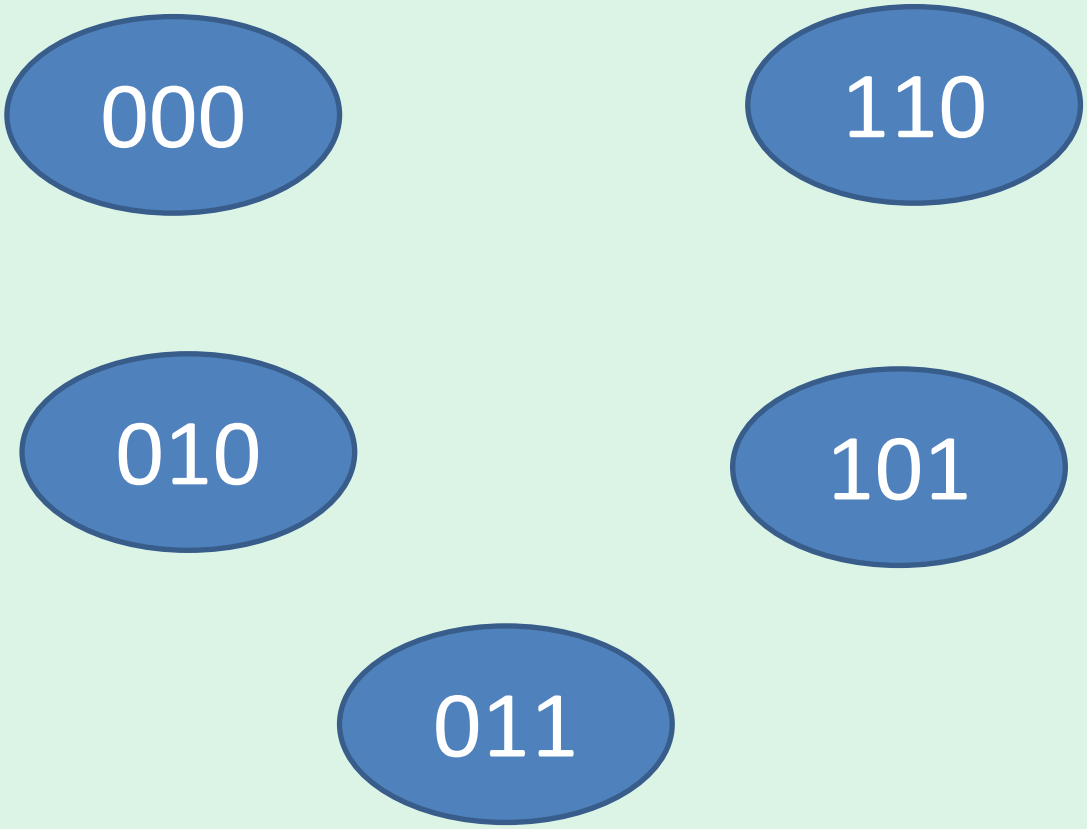
0 1 1

1 0 1

1 1 0

(0, 2, 3, 5, 6)

1. State Transition Diagram



2. State Transition Table

A	B	C	A^+	B^+	C^+
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

2. State Transition Table

A	B	C	A^+	B^+	C^+
0	0	0	0	1	0
0	0	1	?	?	?
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	?	?	?
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	?	?	?

3. Select Flip-Flop

e.g.: We will use T flip-flops:

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

3. Figure out FF Inputs

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

A	B	C	A^+	B^+	C^+	TA	TB	TC
0	0	0	0	1	0			
0	0	1	?	?	?			
0	1	0	0	1	1			
0	1	1	1	0	1			
1	0	0	?	?	?			
1	0	1	1	1	0			
1	1	0	0	0	0			
1	1	1	?	?	?			

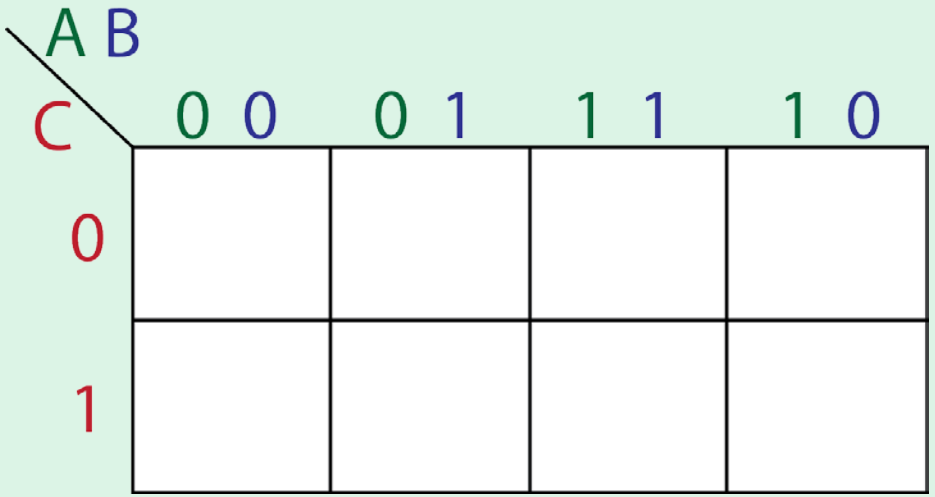
3. Figure out FF Inputs

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

A	B	C	A^+	B^+	C^+	TA	TB	TC
0	0	0	0	1	0	0	1	0
0	0	1	?	?	?	?	?	?
0	1	0	0	1	1	0	0	1
0	1	1	1	0	1	1	1	0
1	0	0	?	?	?	?	?	?
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	?	?	?	?	?	?

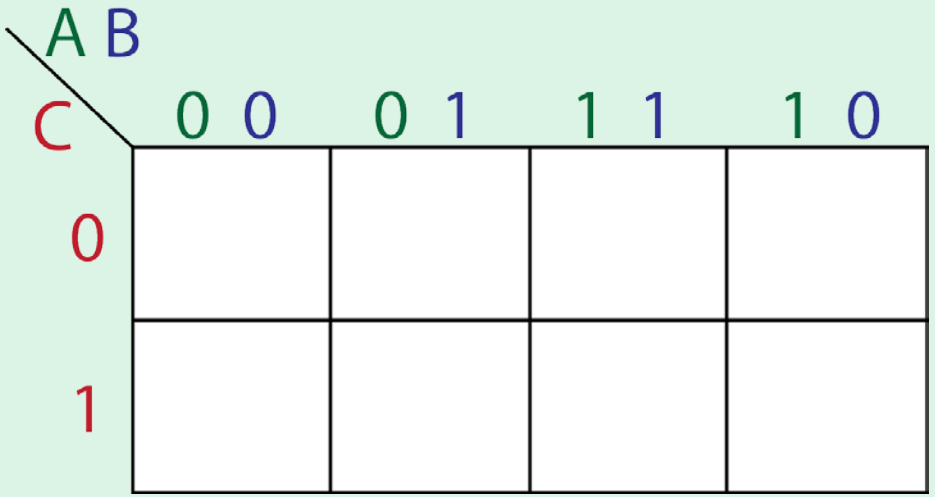
4. Generate Required Logic

A	B	C	TA
0	0	0	0
0	0	1	?
0	1	0	0
0	1	1	1
1	0	0	?
1	0	1	0
1	1	0	1
1	1	1	?



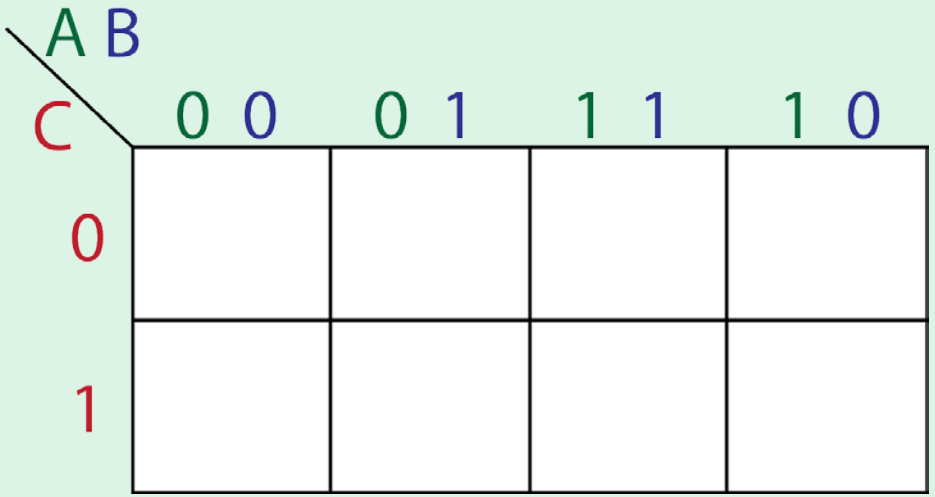
4. Generate Required Logic

A	B	C	TB
0	0	0	1
0	0	1	?
0	1	0	0
0	1	1	1
1	0	0	?
1	0	1	1
1	1	0	1
1	1	1	?

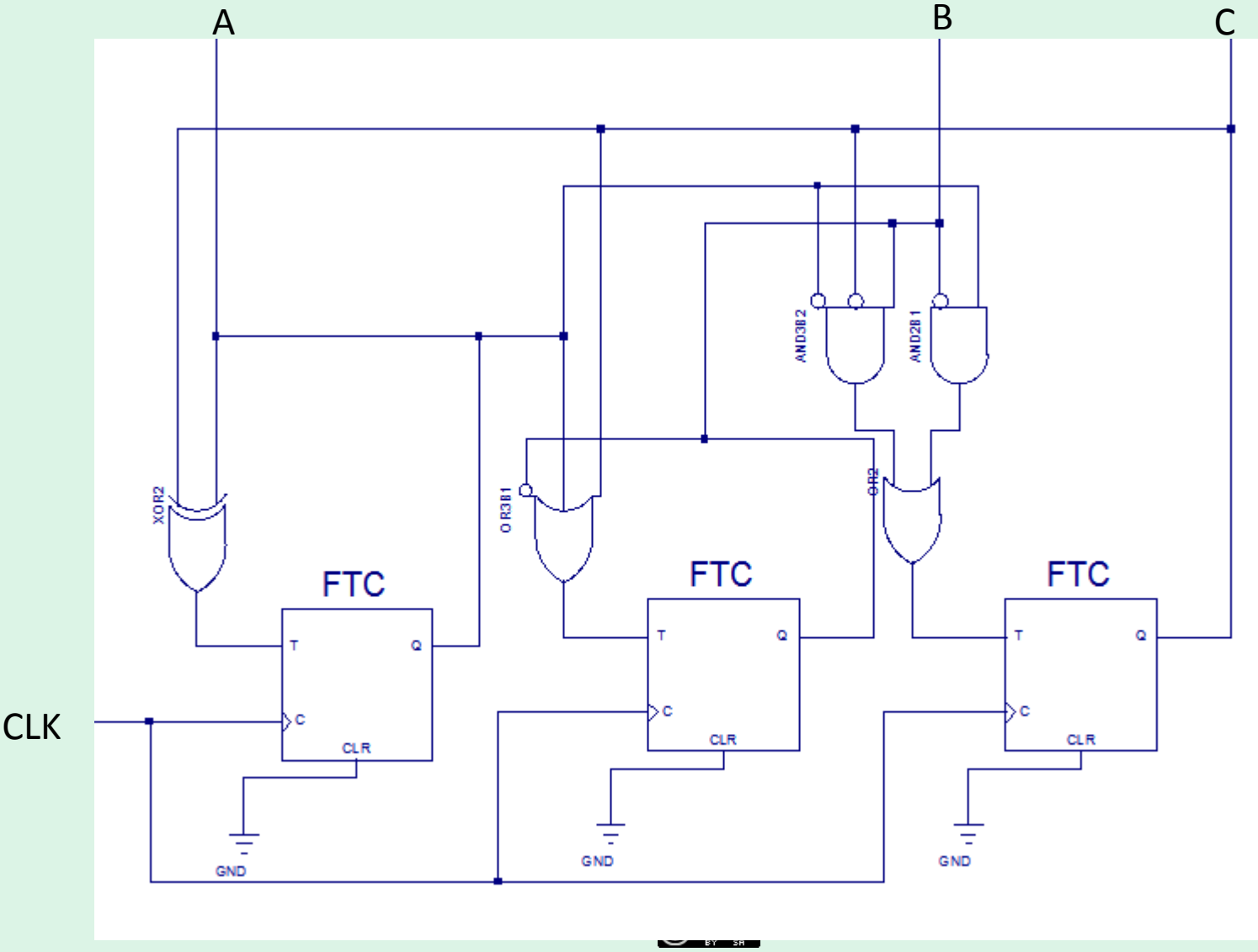


4. Generate Required Logic

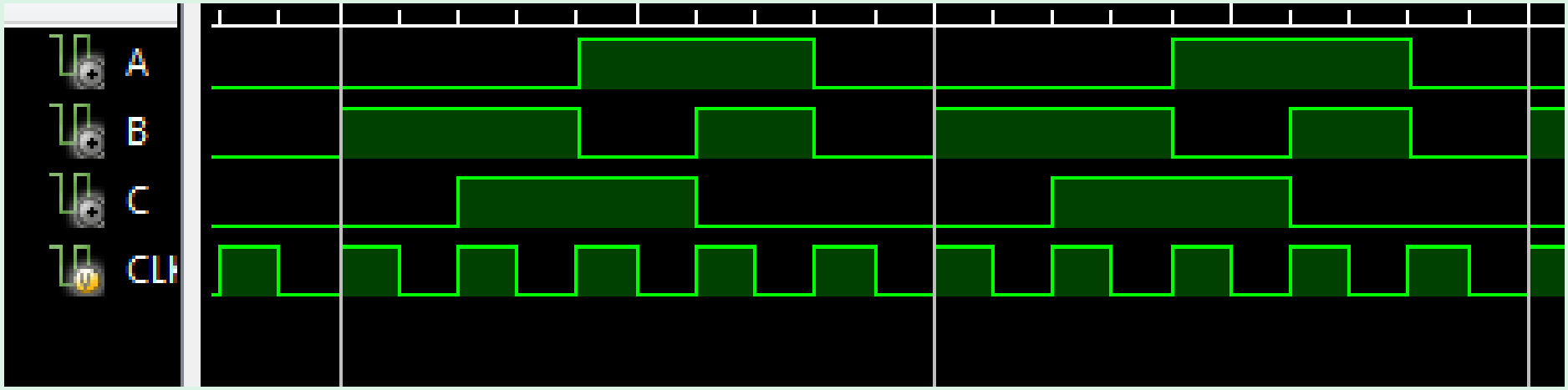
A	B	C	TC
0	0	0	0
0	0	1	?
0	1	0	1
0	1	1	0
1	0	0	?
1	0	1	1
1	1	0	0
1	1	1	?



4. Implement


A
B
C
CLK
XOR2
OR3B1
AND3B2
AND2B1
OR2
FTC
FTC
FTC
T
Q
C
CLR
GND
GND
GND

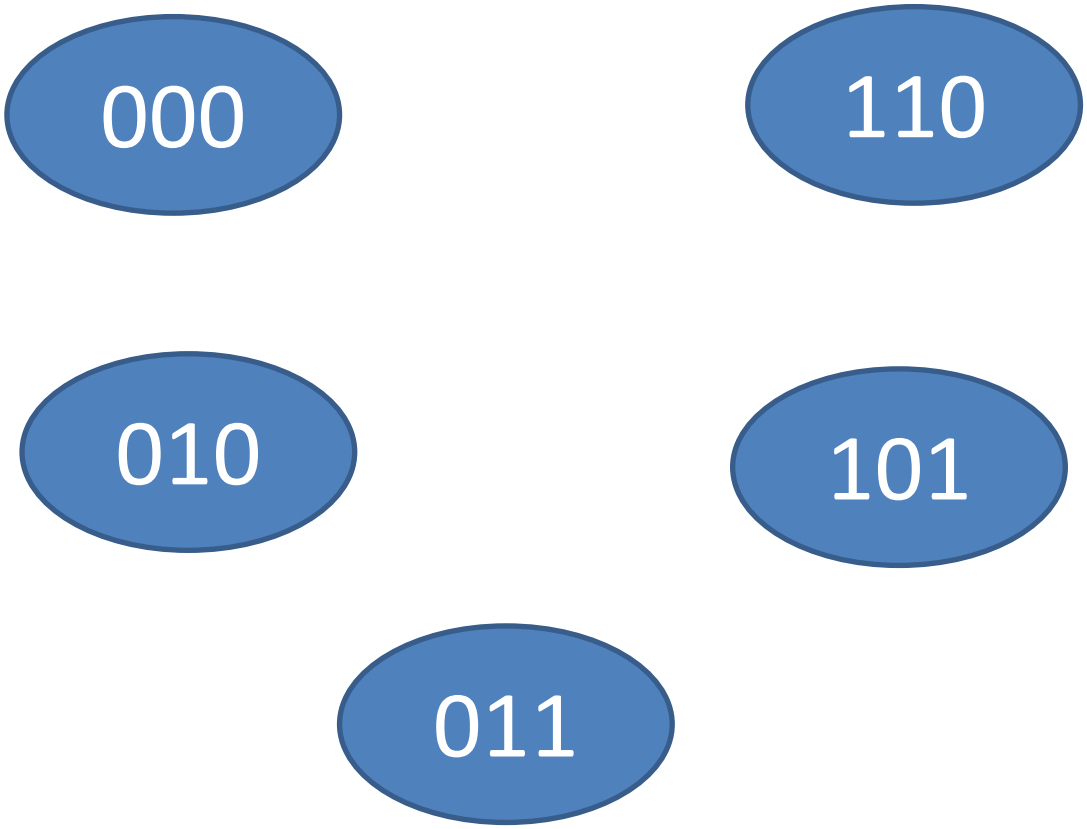
4. Implement



Verifying if Counters are Self-Starting

A	B	C	A^+	B^+	C^+	TA	TB	TC
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

State Transition Diagram



More

See class notes for lots of examples of creating counters & verifying if they are self-starting or not