Latch: Outputs will change whenever the inputs change

Flip Flop: Outputs will change when there is a clock pulse, and will therefore need a clock.

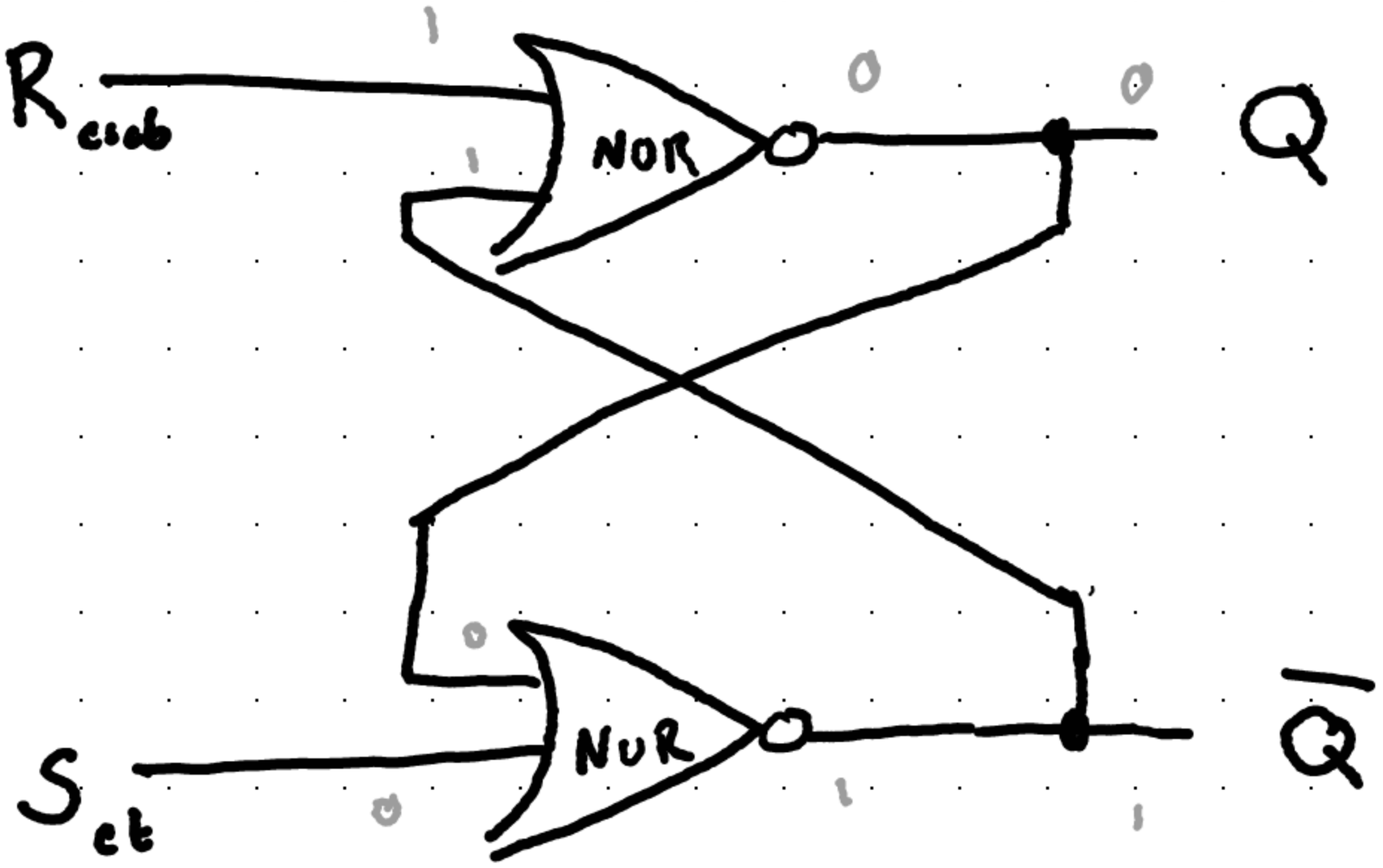
ASync Inputs:

All latches have the possibility of having ASynchronous inputs, which don't require an clock signal to update

Typically:

Clear: Clears element to Zero

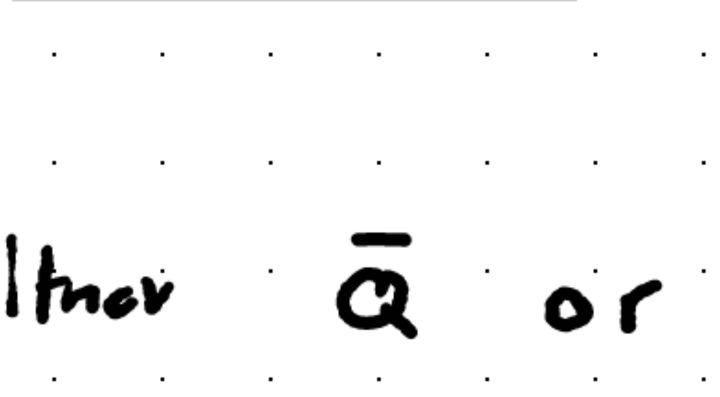
Preset: Sets Q to One



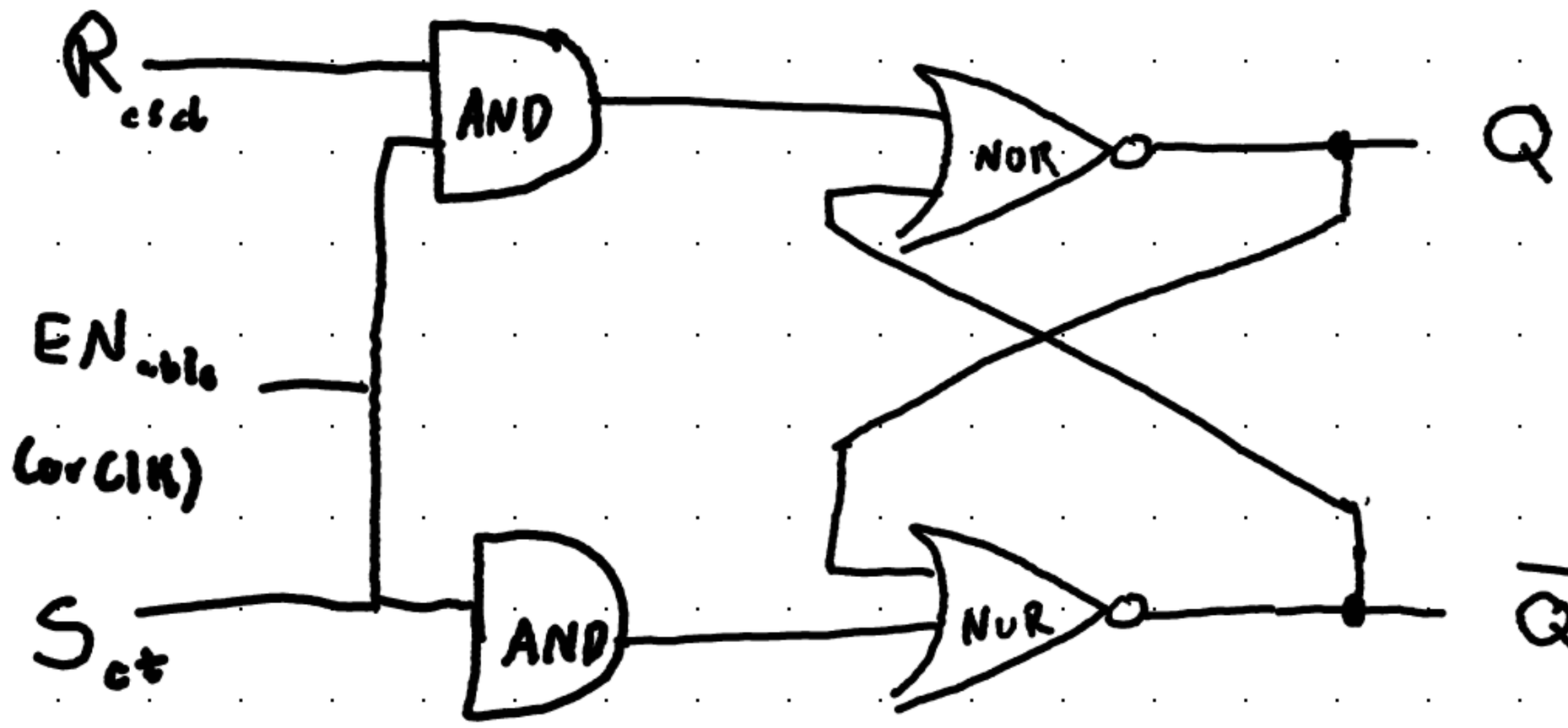
S-R Latch

Set in either $\bar{Q}$ or Q.

Typically, one cares about storing Q, not so much $\bar{Q}$



Memory Element



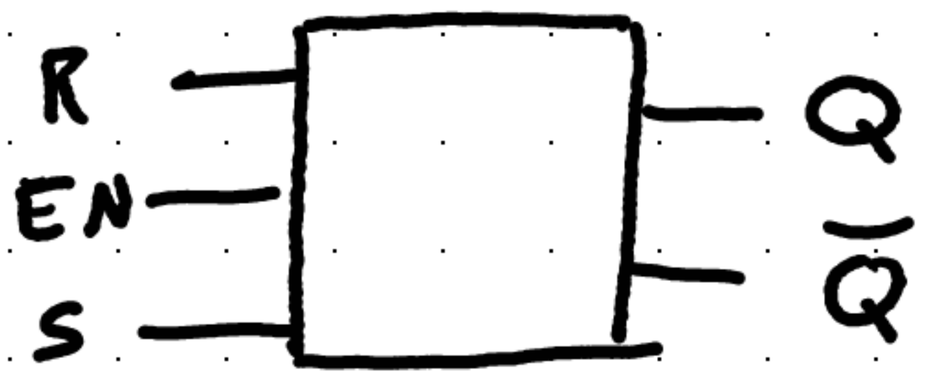
→ Of F-F with clock

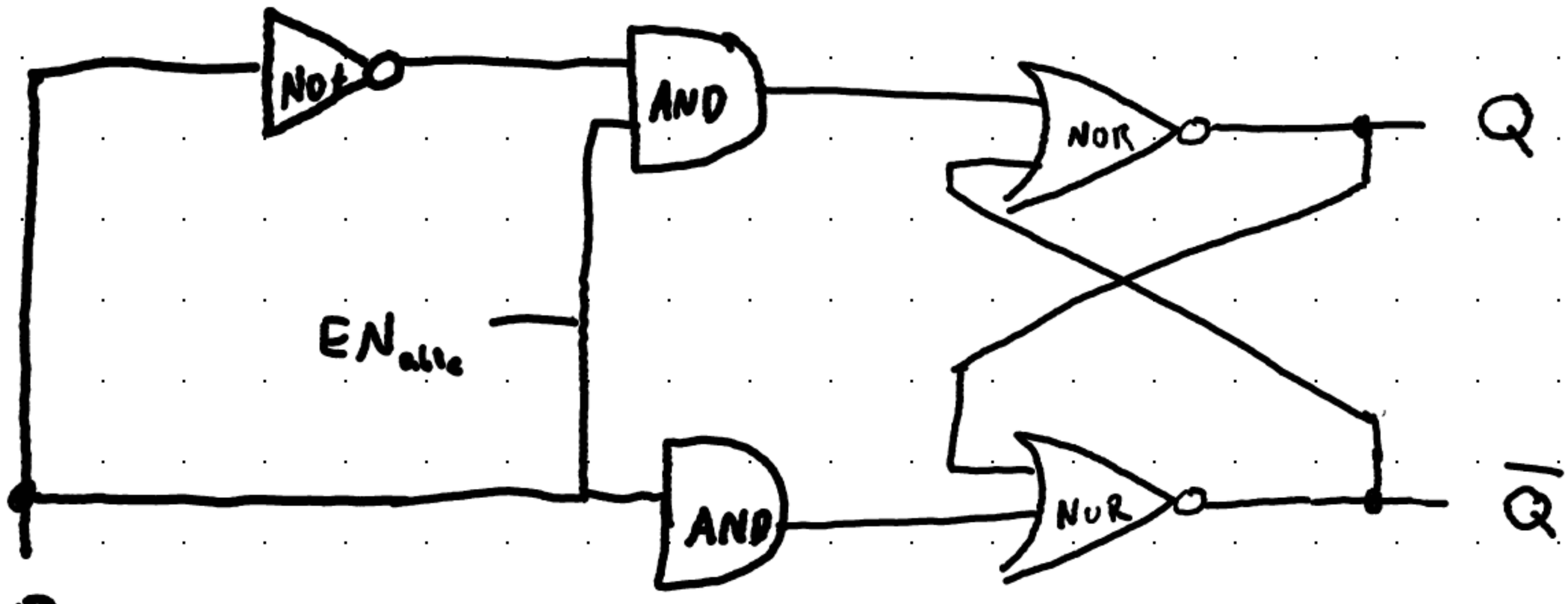
S-R Latch with Enable

Enable must be "pulsed" or "high" to have any inputs register. All future enables require the same. We can store one bit with these latches.

1) The Set=0, Reset=0 condition must be avoided

2) If S or R change state while EN is high the correct latching action may not occur



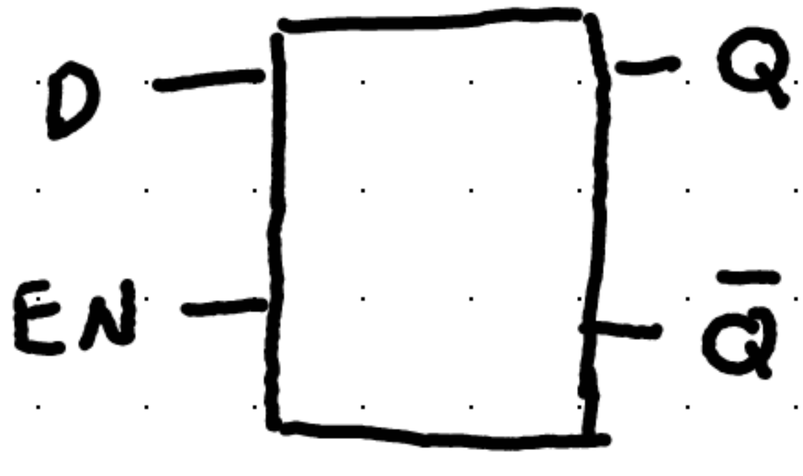


D Latch

Single input toggle latch with Memory.

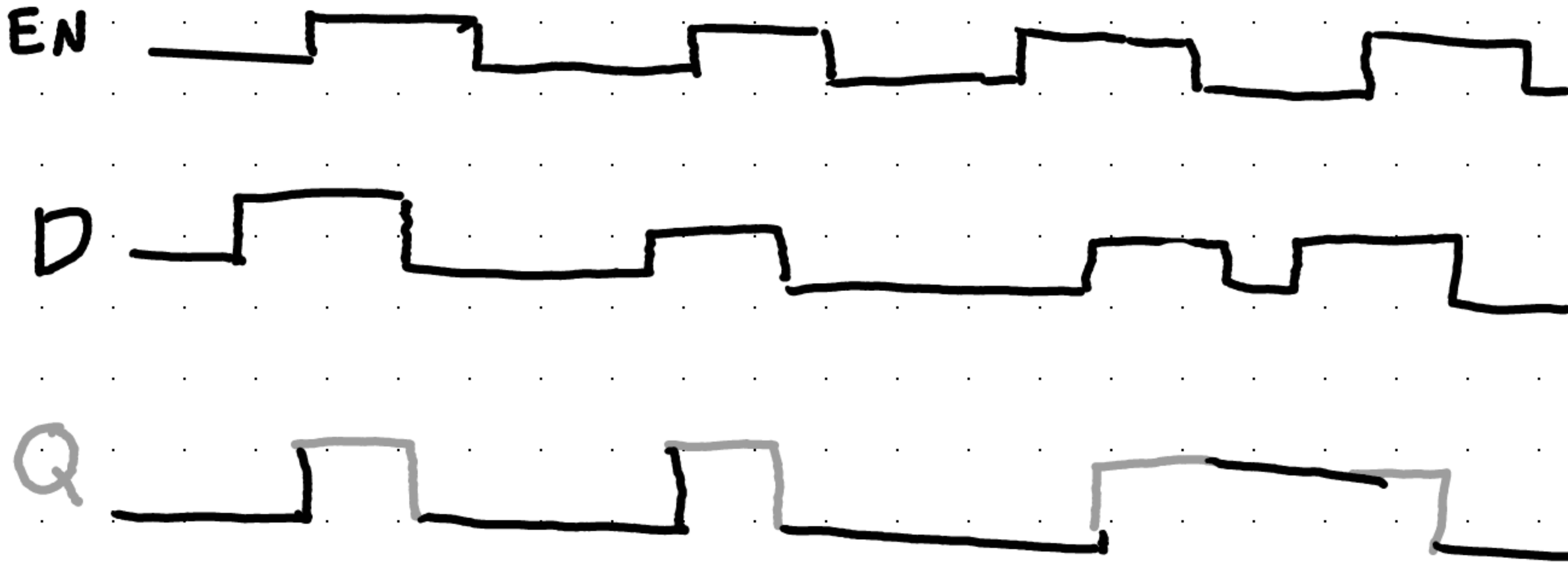
Extra Enable logic needed for the Memory aspect of the Latch

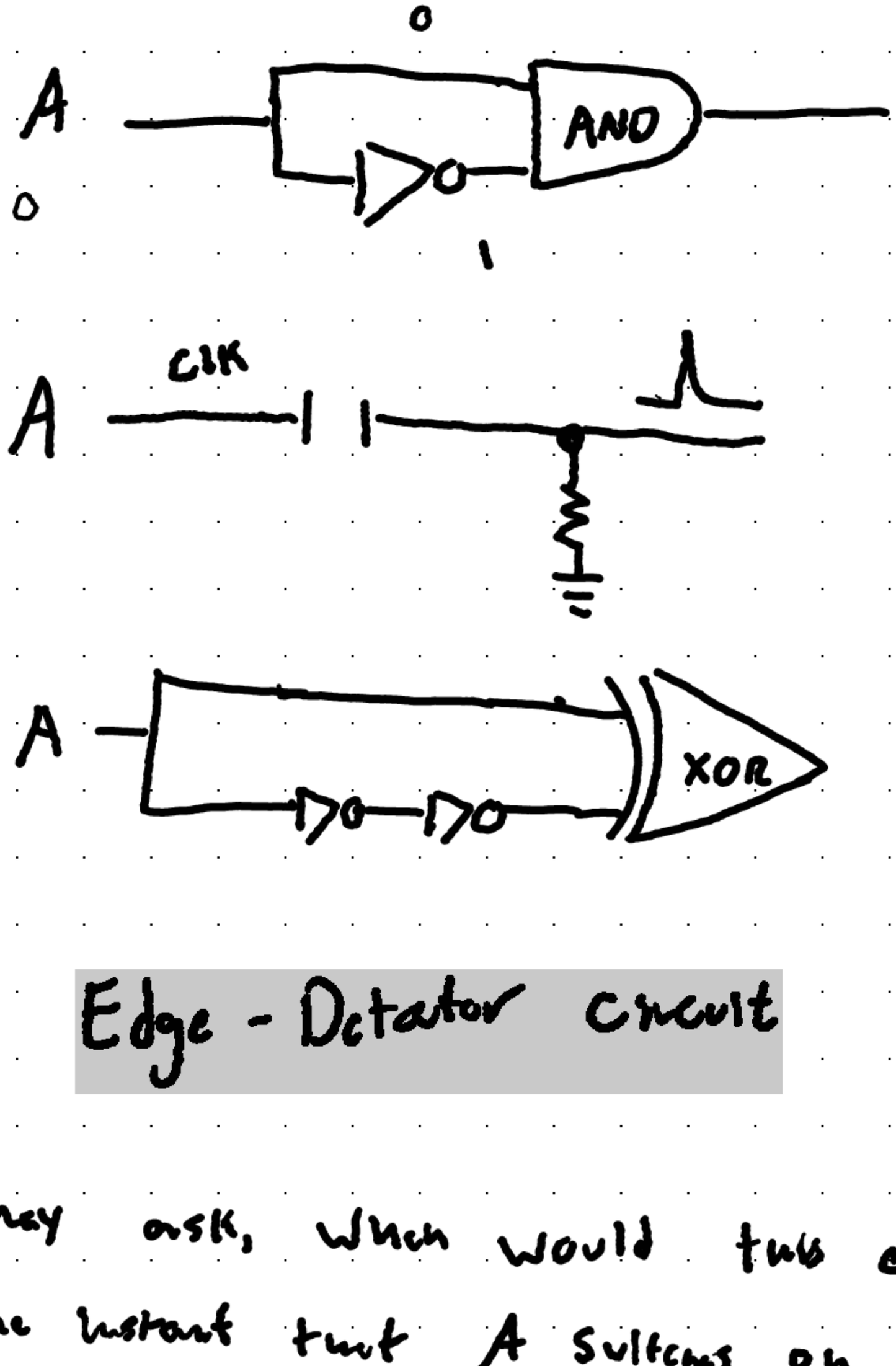
If enable is always held, by pushing D, it will just toggle Q. To "Set" Q, one must hold enable and Push D, and then let go of enable before letting go of D. It won't be able to update back



"Style Diff of Data Latch"

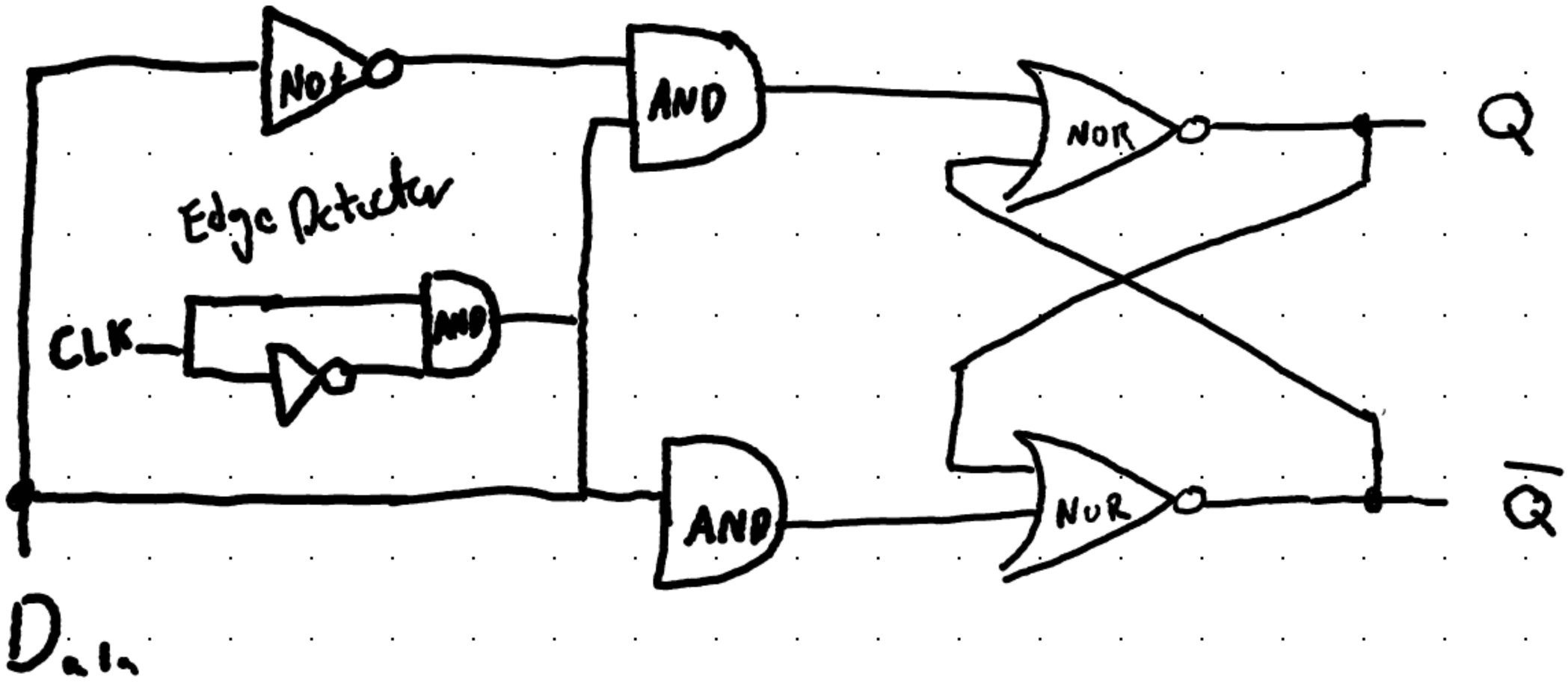
D Latch Simulation





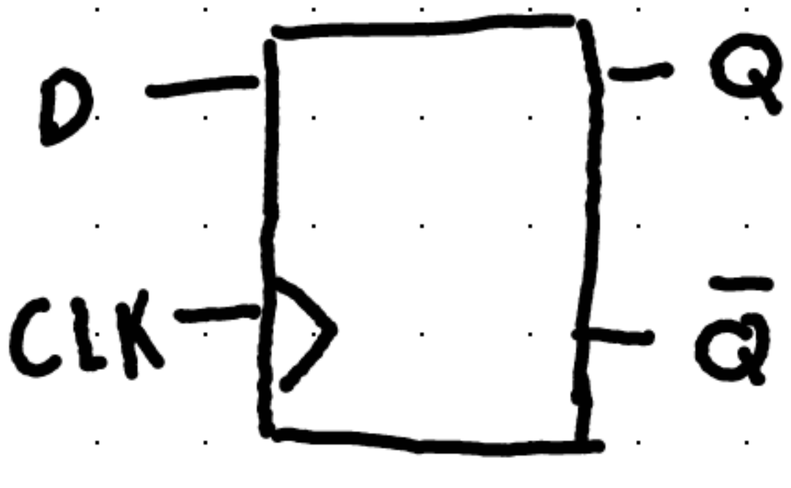
Edge - Detator circuit

One may ask, which would tubs ever be on? In the instant that A switches on, both will be high due to the delay of the NOT gate



D Flip-Flop

The most common way of storing data!



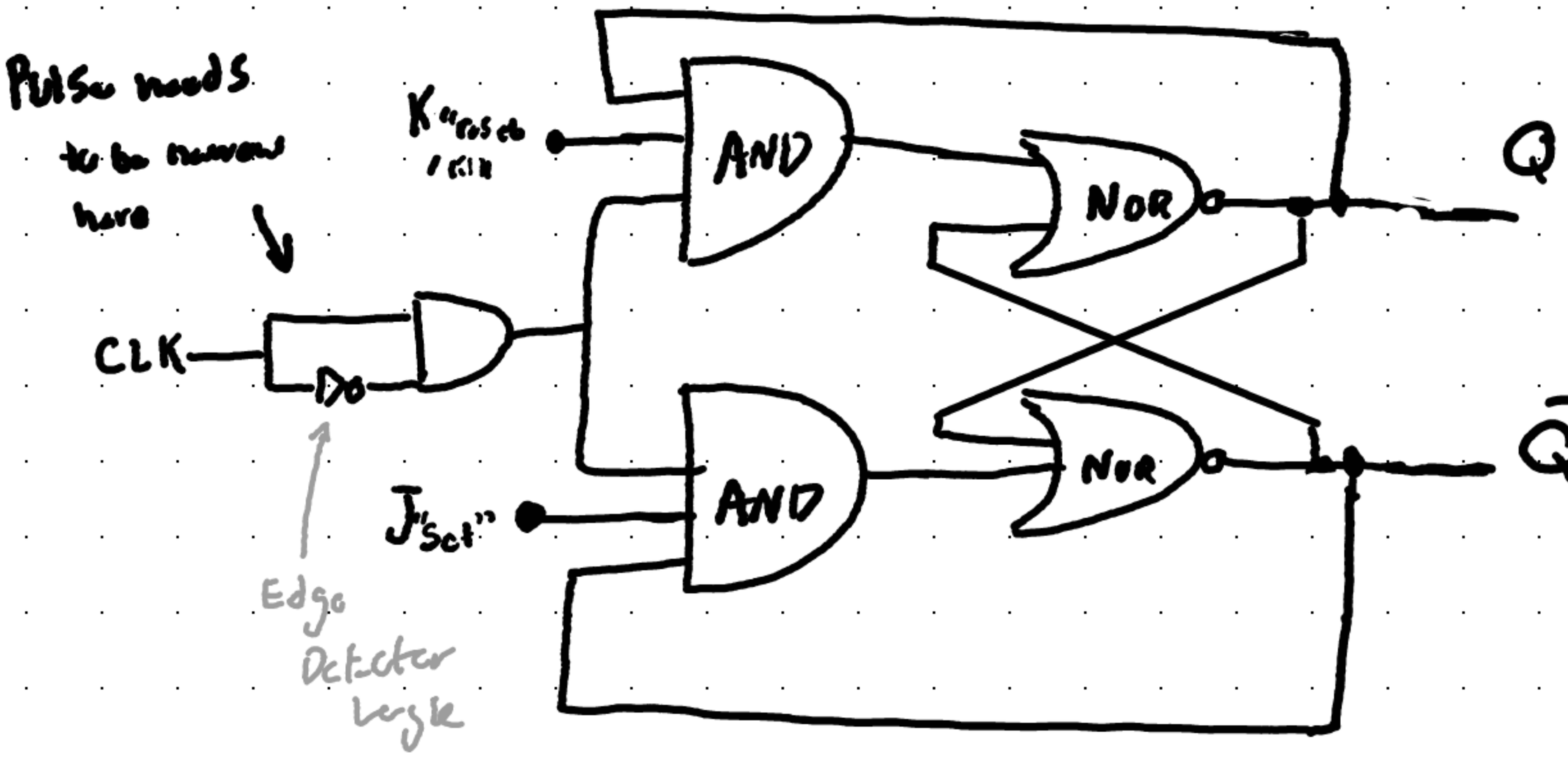
D Flip-Flop Simulation

The rising edge is almost as if...

This is why we need an edge detector on the DFF

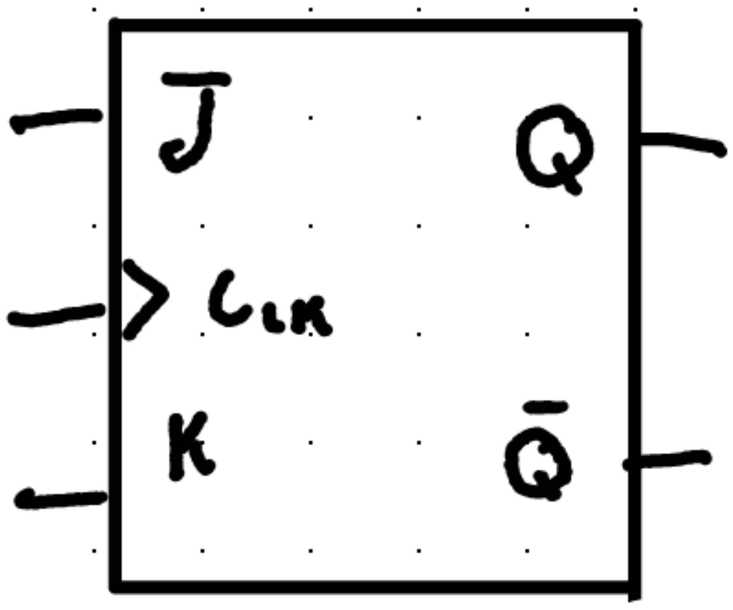
We care about the rising edge of the clock

This is for more common



J-K Flip Flop

- Named after inventor Jack Kilby
- The JK Flip flop is the only Universal flip-flop, and can be configured to act as an SR, D or T flip flop.
- Designed to overcome flaws of S-R Flip-flop NO illegal States.



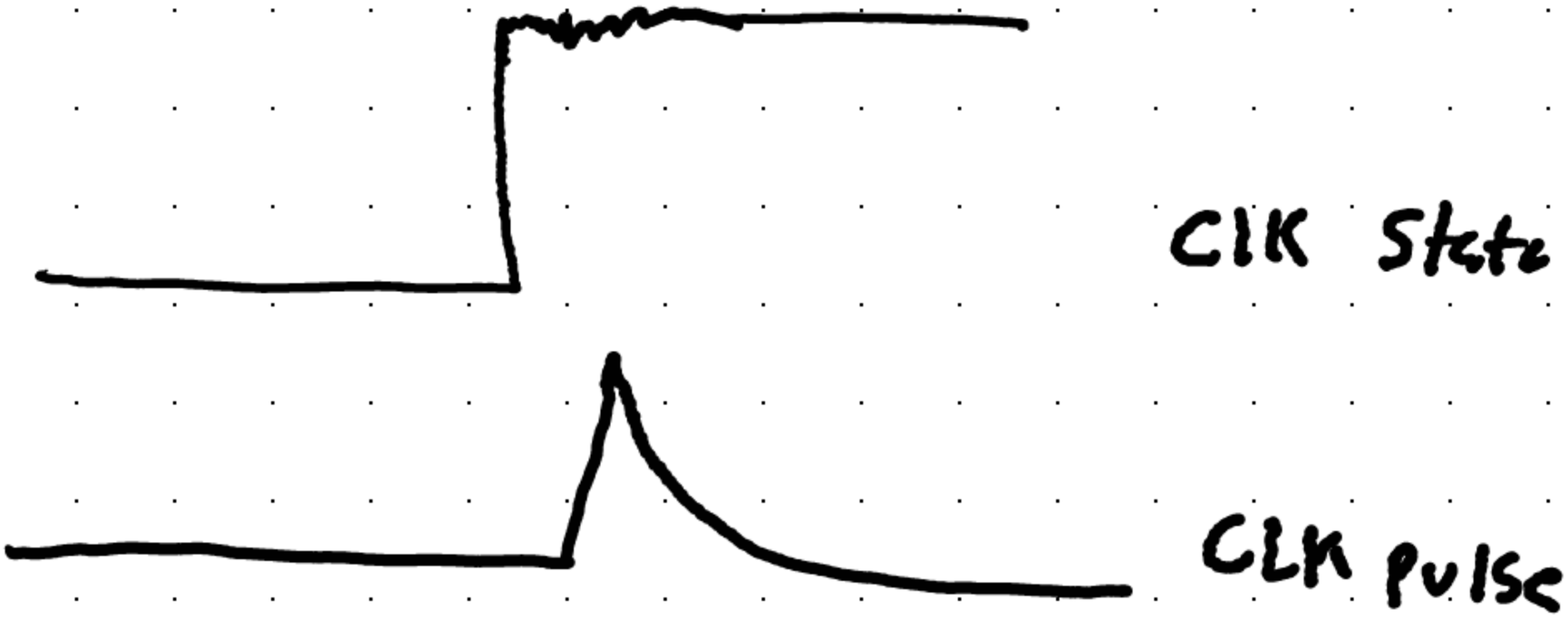
Truth Table for the JK Flip Flop

	Clock	Input		Output		Description
	CLK	Set J	ReSet K	Q	$\overline{Q}$	
Same as SR Latch for the	X	0	0	1	0	Memory, No Change Last State
	X	0	0	0	1	
	↓	0	1	0	1	Reset Q770
	X	0	1	1	0	
	↓	1	0	1	0	Set Q771
	X	1	0	0	1	
Toggle Action	↓	1	1	0	1	Toggle On and Off
	↓	1	1	0	1	

ON Each Clock

J-K Flip Flop Racing

With a fast Clock Signal, with J and K both being on... Weirdness can occur.



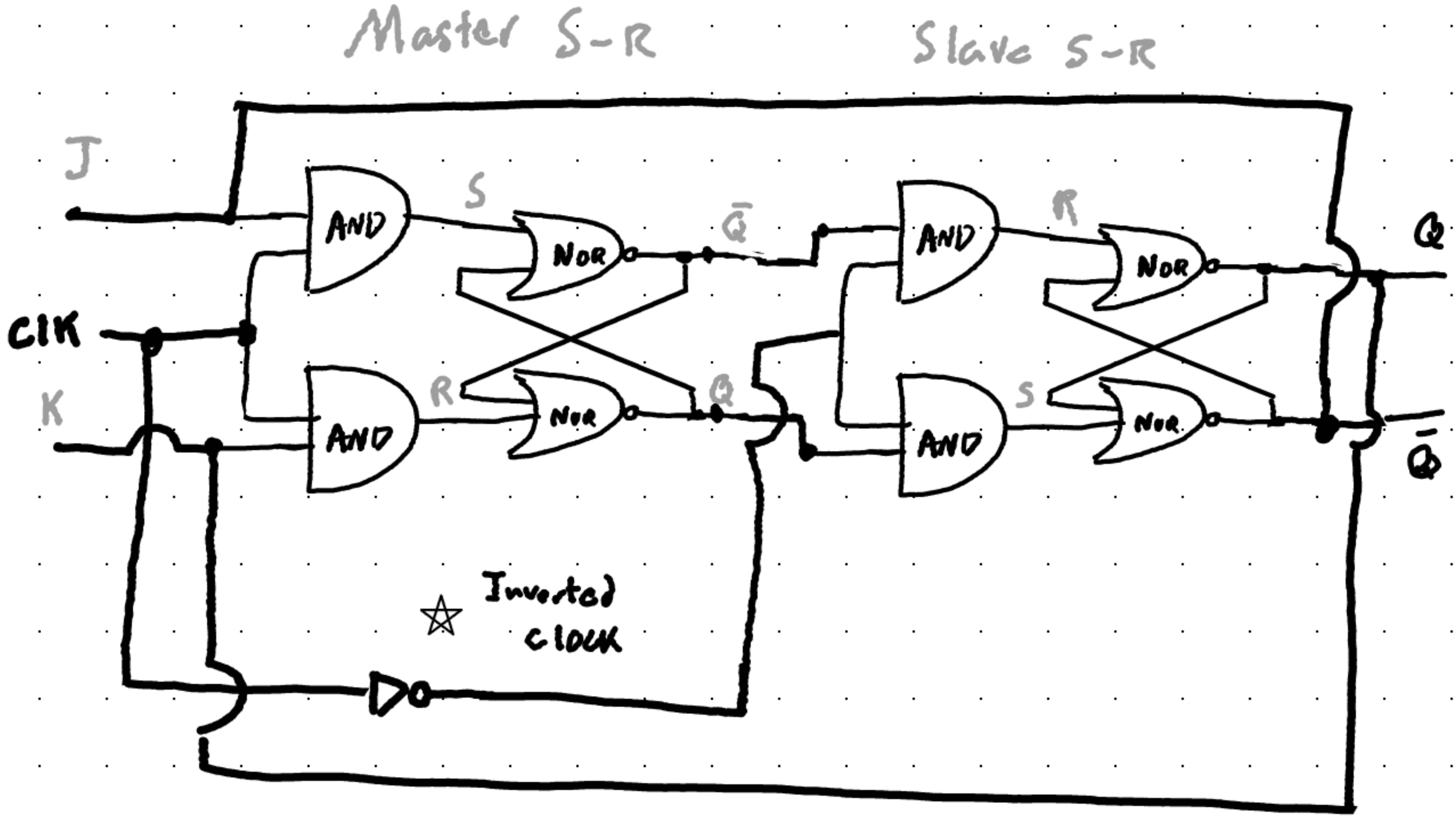
Really Zoomed In on Scope

The RC Clock detecting circuit detects the edge, but turns out as Clock pulse isn't instantaneous due to the RC Circuit's time constant.

This causes the output to toggle extremely fast through the J-K Flip Flop.

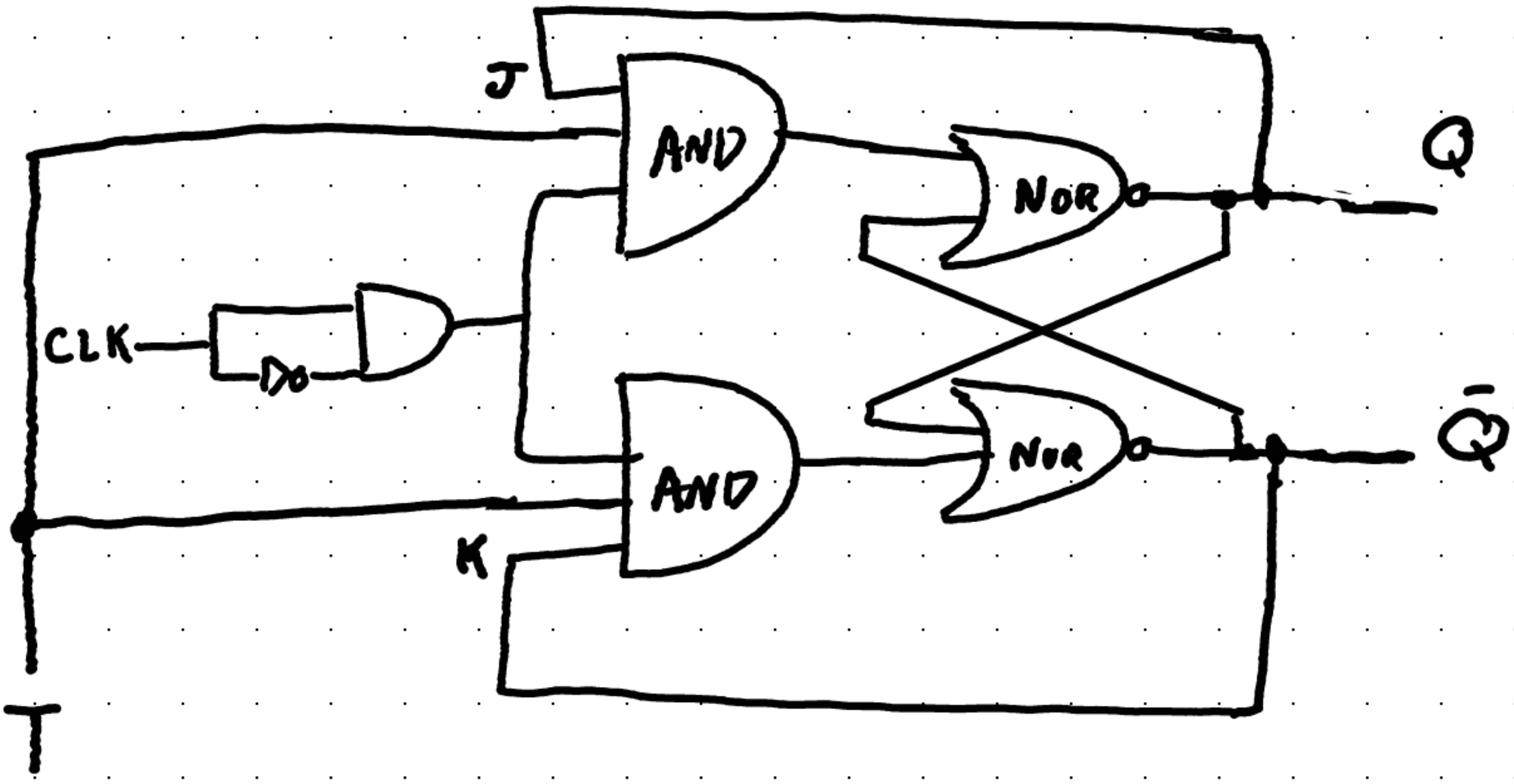
This phenomenon is known as racing

To fix this, and to have a steady toggle system, we employ another kind of circuit



Master-Slave J-K Flip Flop

- Here, One of the two flip flops will be active, whether the Clock is low, or high
- Toggle works much better here.



T Flip Flop

J-K's D flip flop!

Single Input Toggler

Examples:

Draw the circuit for a four-bit shift register using D Flip-Flops that can perform parallel to Serial Conversion, as well as Serial to parallel.

Identify:

i) The Serial Input terminal

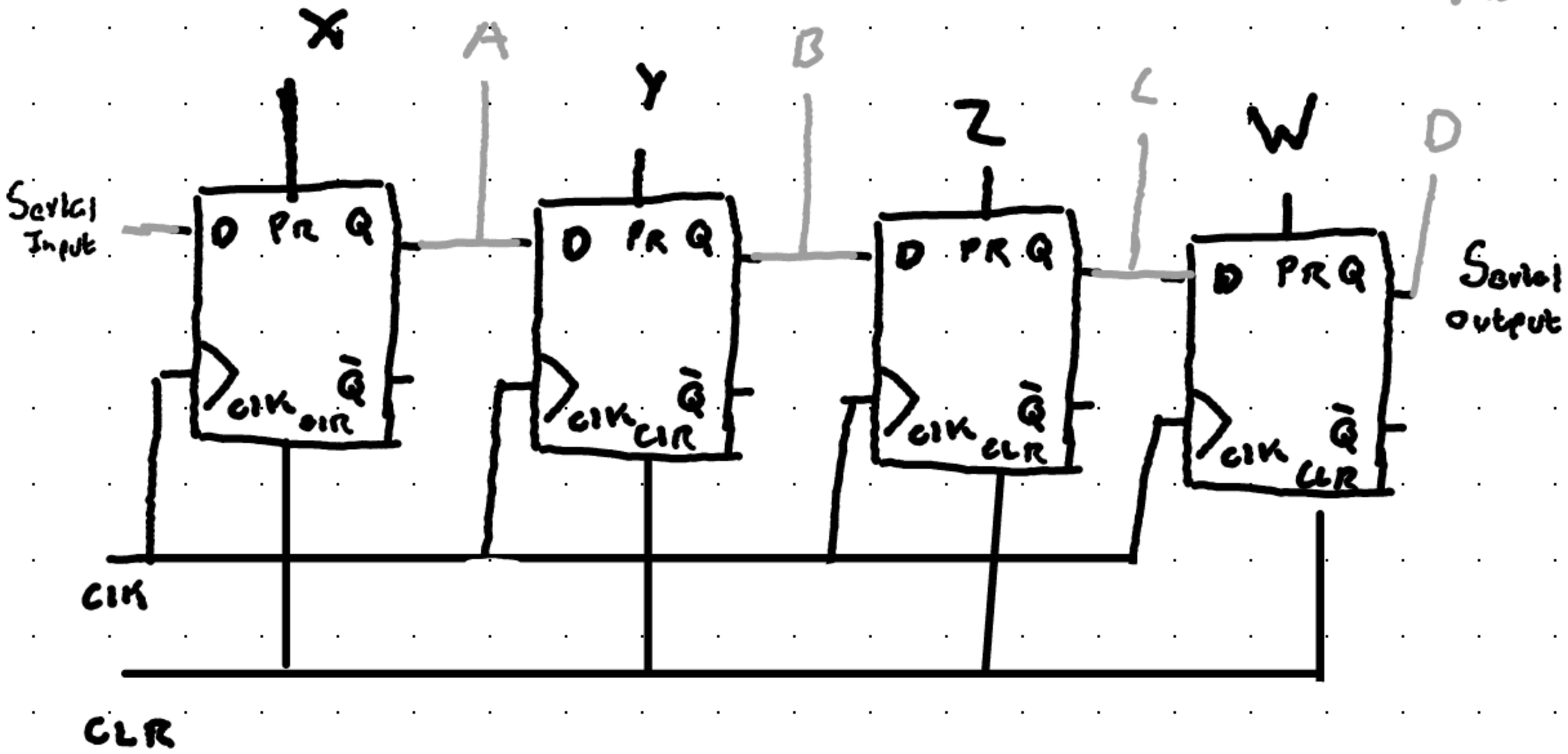
ii) The Serial output terminal

iii) The parallel input terminal

iv) The parallel output terminal

Parallel Input: X, Y, Z, W

Parallel output: A, B, C, D



Shift registers

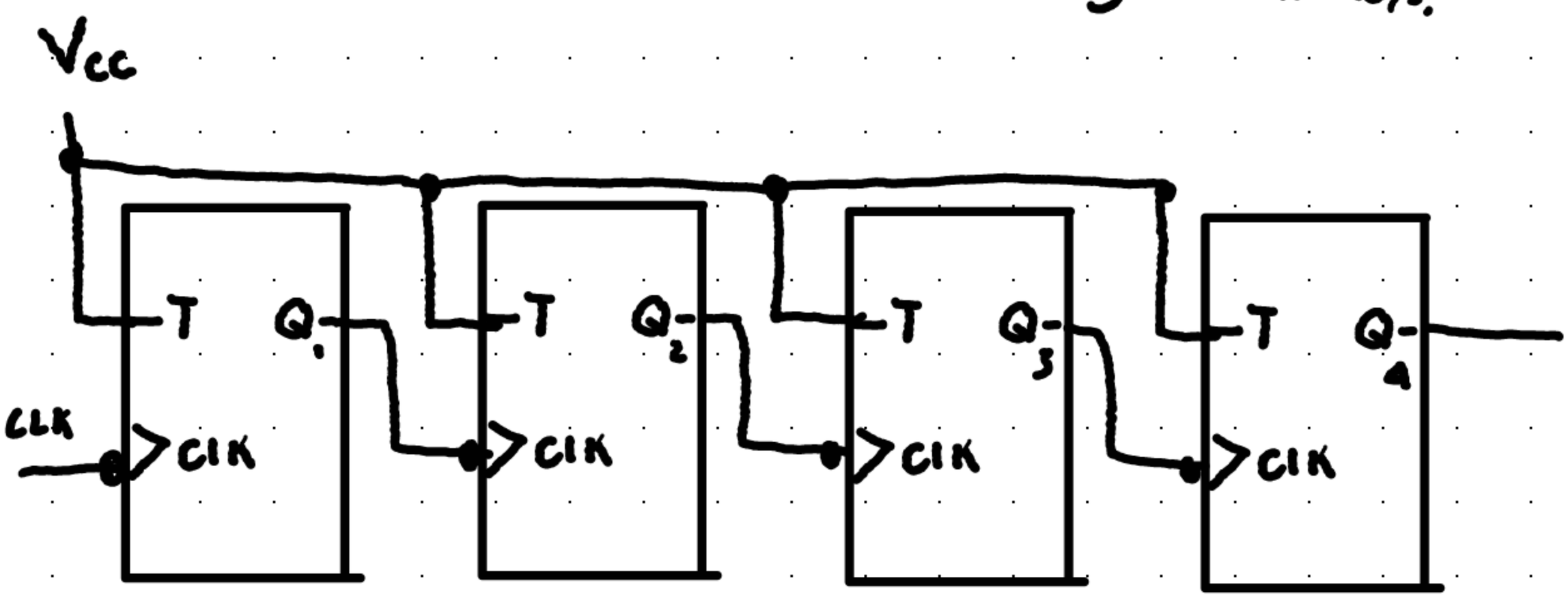
A Shift register is a Sequential logic circuit that Stores data. Each flip flop adds a bit of Storage. Updates on Clock pulse.

1. Serial In - Serial out (SISO)
   - Data is inputted one bit at a time
   - output is taken from the last flip flop after the four clock cycles.
2. Serial In - parallel out (SIPO)
   - Data is loaded serially, but all four bits are available simultaneously at the outputs
3. Parallel in - Serial out (PISO)
   - Data is loaded in parallel (all four bits at once), but shifted out serially.
4. Parallel in - parallel out (PIPO)
   - Data is loaded in, and read in Parallel.

Examples:

Provide the State transition table for an asynchronous binary up-counter that goes through the sequence (0000, 0001, 0010, ..., 1110, 1111, 0000, 0001)

a) Draw the ripple binary up-counter using T flip flops.



$Q_1\ Q_2\ Q_3\ Q_4$	$Q_1^+\ Q_2^+\ Q_3^+\ Q_4^+$
0 0 0 0	0 0 0 1
0 0 0 1	0 0 1 0
0 0 1 0	0 0 1 1
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	1 0 1 0
1 0 1 0	1 0 1 1
1 0 1 1	1 1 0 0
1 1 0 0	1 1 0 1
1 1 0 1	1 1 1 0
1 1 1 0	0 0 0 0

Decade Counter: using T flip flops

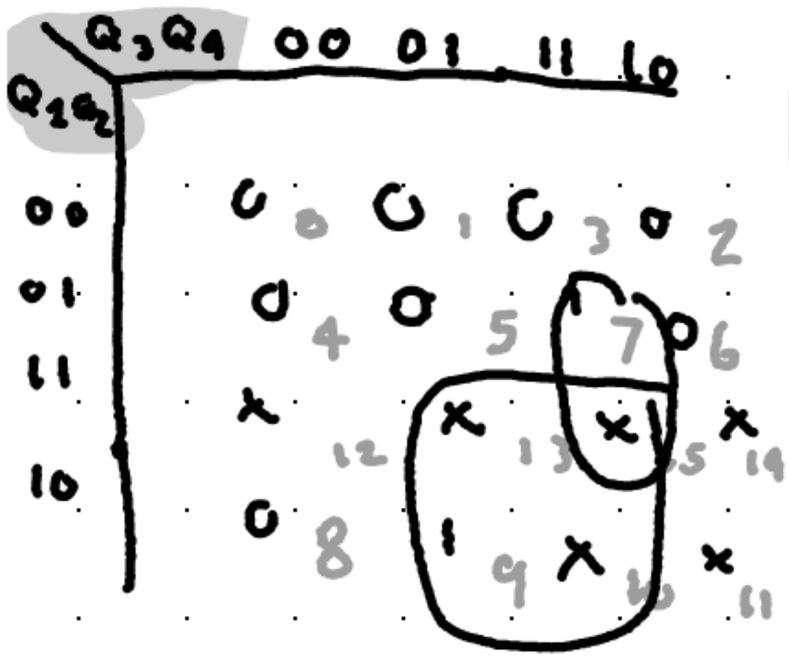
1) State Table

	Q_1	Q_2	Q_3	Q_4	Q_1^+	Q_2^+	Q_3^+	Q_4^+	T_1	T_2	T_3	T_4
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	1	0	0	0	1	1
2	0	0	1	0	0	0	1	1	0	0	0	1
3	0	0	1	1	0	1	0	0	0	1	1	1
4	0	1	0	0	0	1	0	1	0	0	0	1
5	0	1	0	1	0	1	1	0	0	0	1	1
6	0	1	1	0	0	1	1	1	0	0	0	1
7	0	1	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	1	0	0	1	0	0	0	1
9	1	0	0	1	0	0	0	0	1	0	0	1

T_4 is always 1

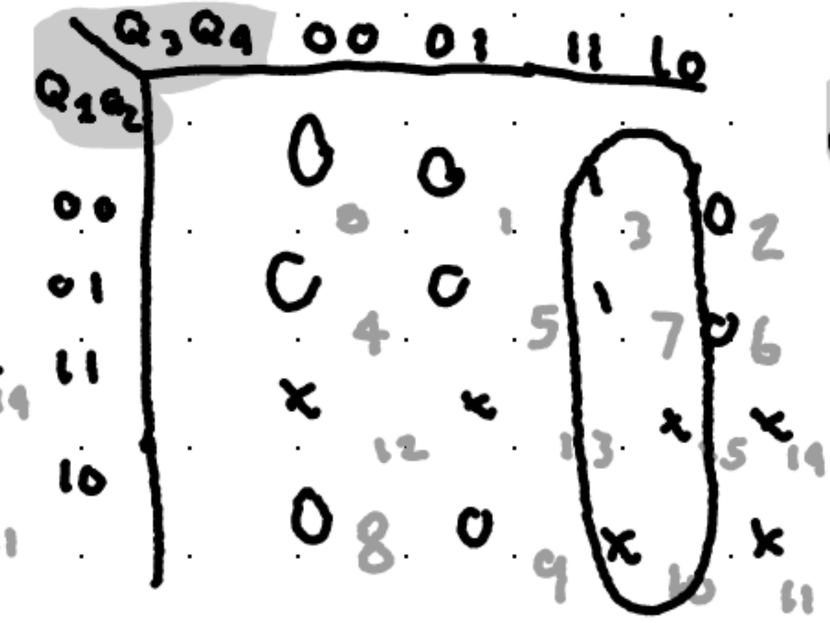
- Need to build K-Maps for Other three using Current Q States

T_1



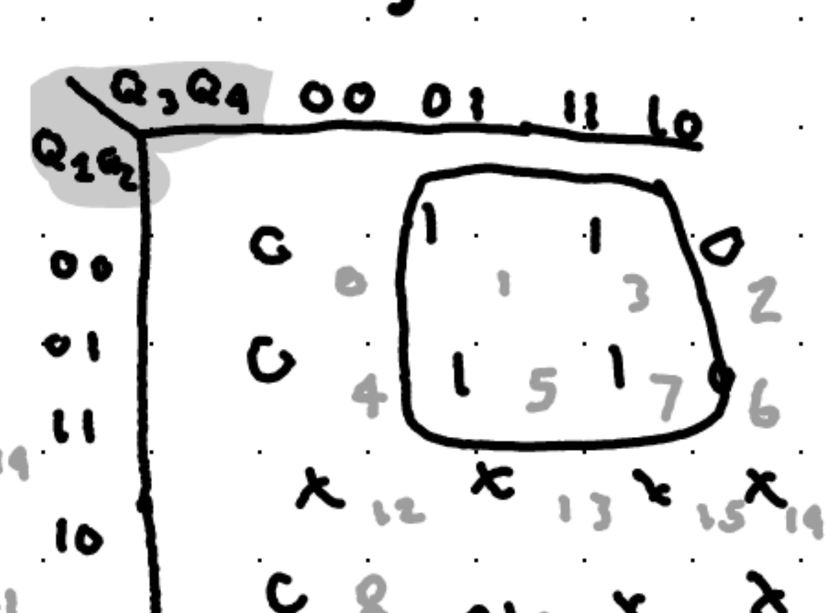
$T_1 = Q_1Q_4 + Q_2Q_3Q_4$

T_2

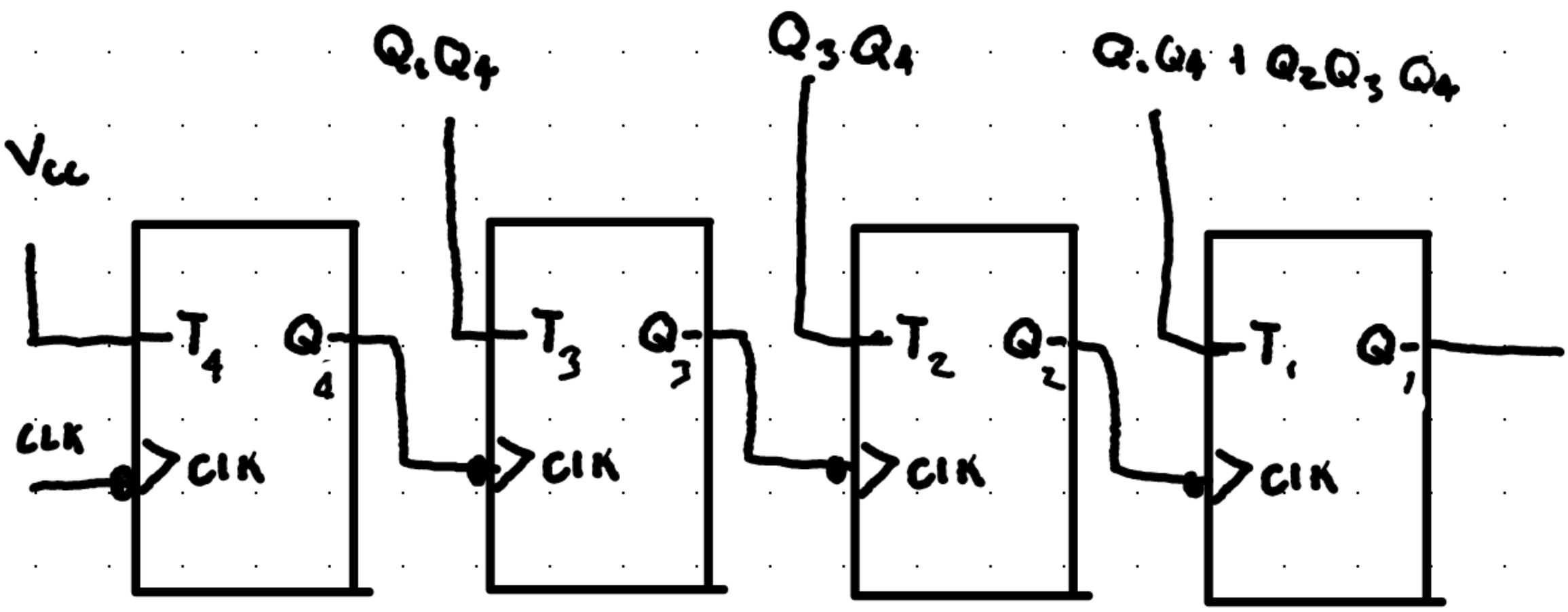


$T_2 = Q_3Q_4$

T_3



$T_3 = Q_1Q_4$



$T_1 = Q_1Q_4 + Q_2Q_3Q_4$

$T_2 = Q_3Q_4$

$T_3 = Q_1Q_4$

$T_4 = 1$

1) Asynchronous operation Meaning in this Context

- Each Flip Flop (except the first) is clocked by the output of the previous flip flop.
- The LSB Flip Flop toggles on every Clock Pulse. Q_1 toggles when Q_0 falls, etc...

2) Each Flip Flop is a <u>negative-edge-triggered T flip flop.</u> (Toggles on clock falling edge)

↳ Because of this, we add <u>not</u> gates on all of the clock <u>inputs</u>

Feature	Decade up Counter	Ripple (Asynchronous) up Counter
Counting Sequence	0000 -> 0001 -> ... -> 1001 -> 0000 (0 to 9)	0000 -> 0001 -> ... -> 1111 -> 0000 (0 to 15)
Modulus	Mod-10 (resets after 9)	Mod-16 (resets after 15 for four bit)
Flip-Flop Usage	4 FF's, but logic resets at 10 (1010)	4 FF's, no forced reset (natural binary count)
Synchronization	Can be async or Synchronous	Always Async
Applications	Base 10 Systems, Digital Clocks, BCD Systems	General purpose Counting, timers, dividers.

Decade = Human readable displays

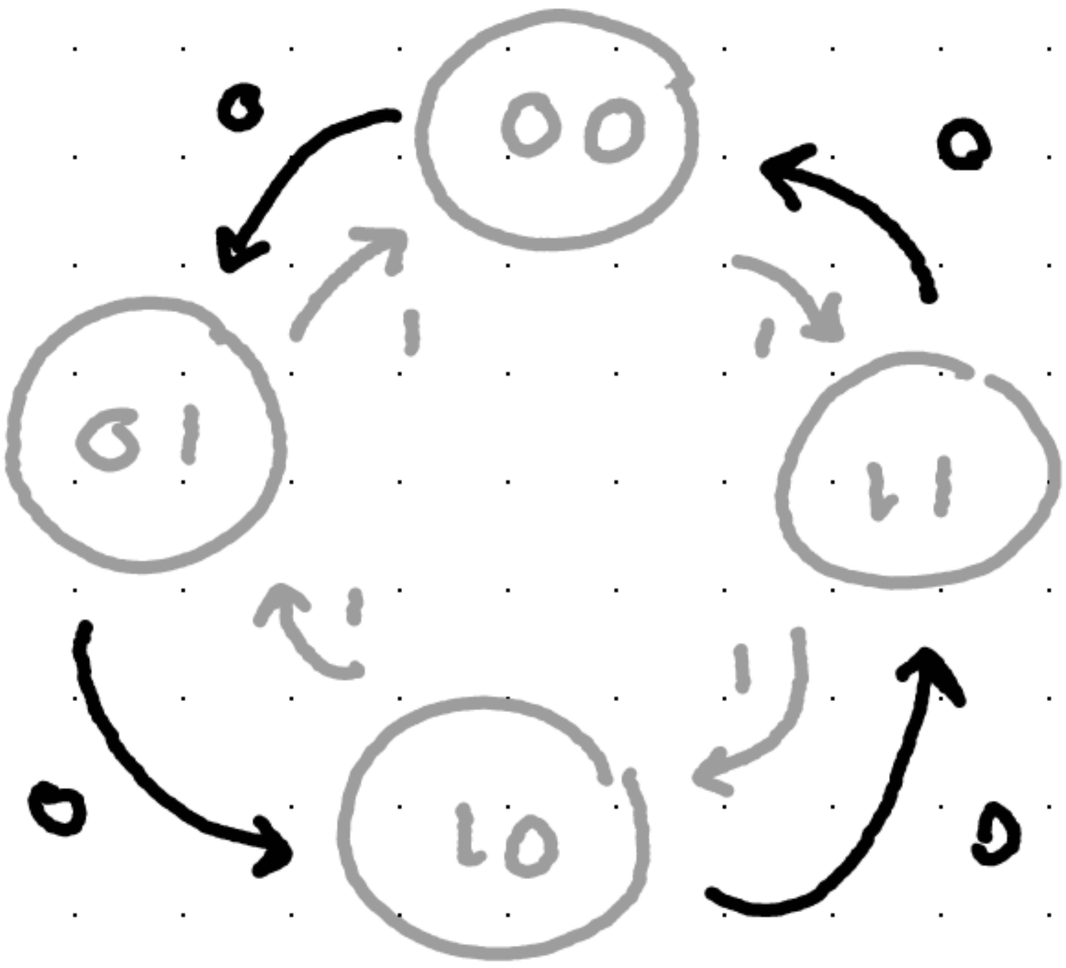
Ripple = Binary operations.

Examples:

Use JK flip flops to design a finite State Machine (FSM) that works as a 2-bit binary Synchronous Up-Counter when its input X=0 and as a 2-bit binary synchronous down-counter when its input X=1

a) draw the state transition diagram
b) build the State transition table. Include values of flip flop inputs
c) find minimized logic expressions for flip flop inputs
d) draw the resulting logic circuit that implements this FSM

a)

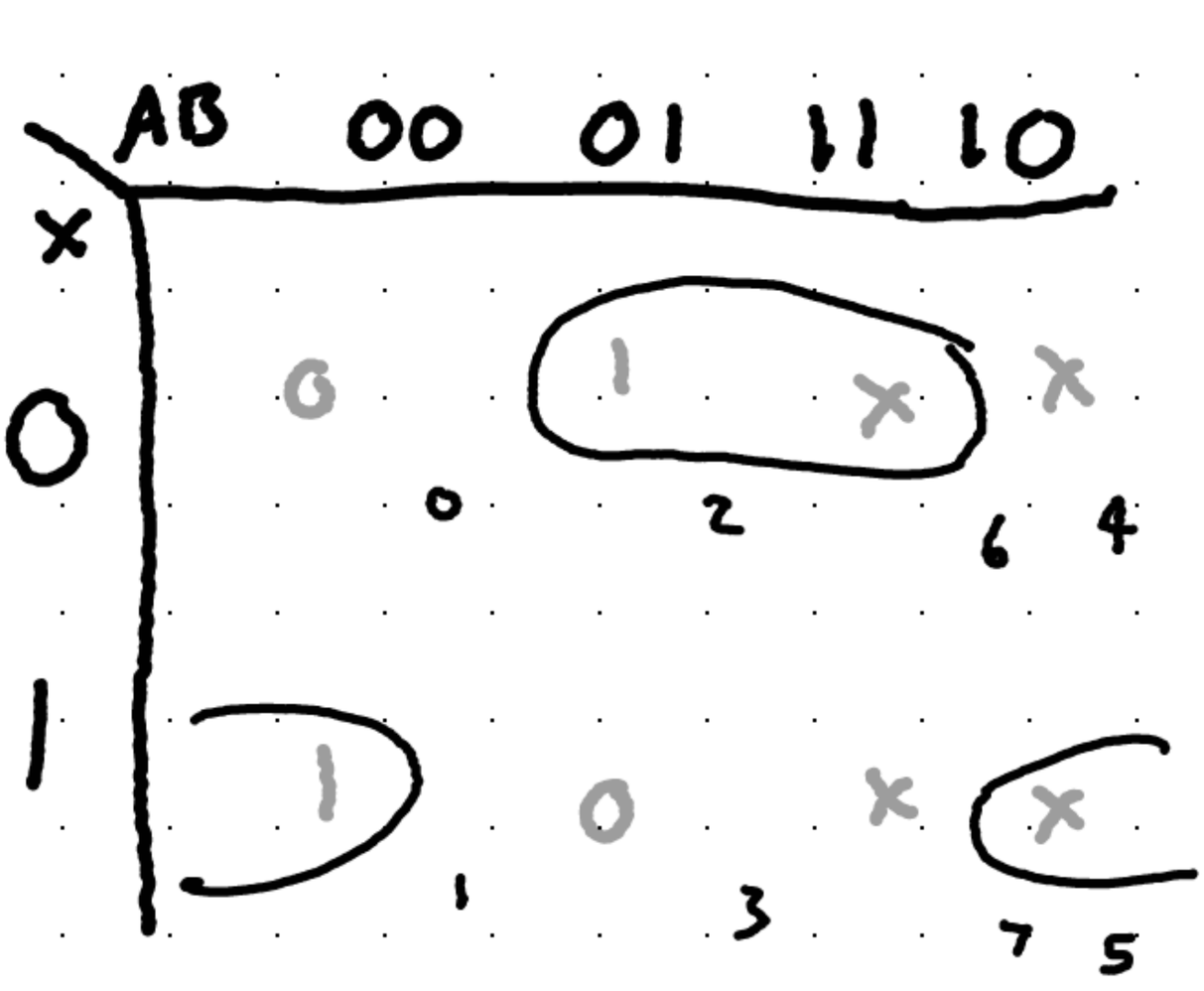


b)

0 = up
1 = down

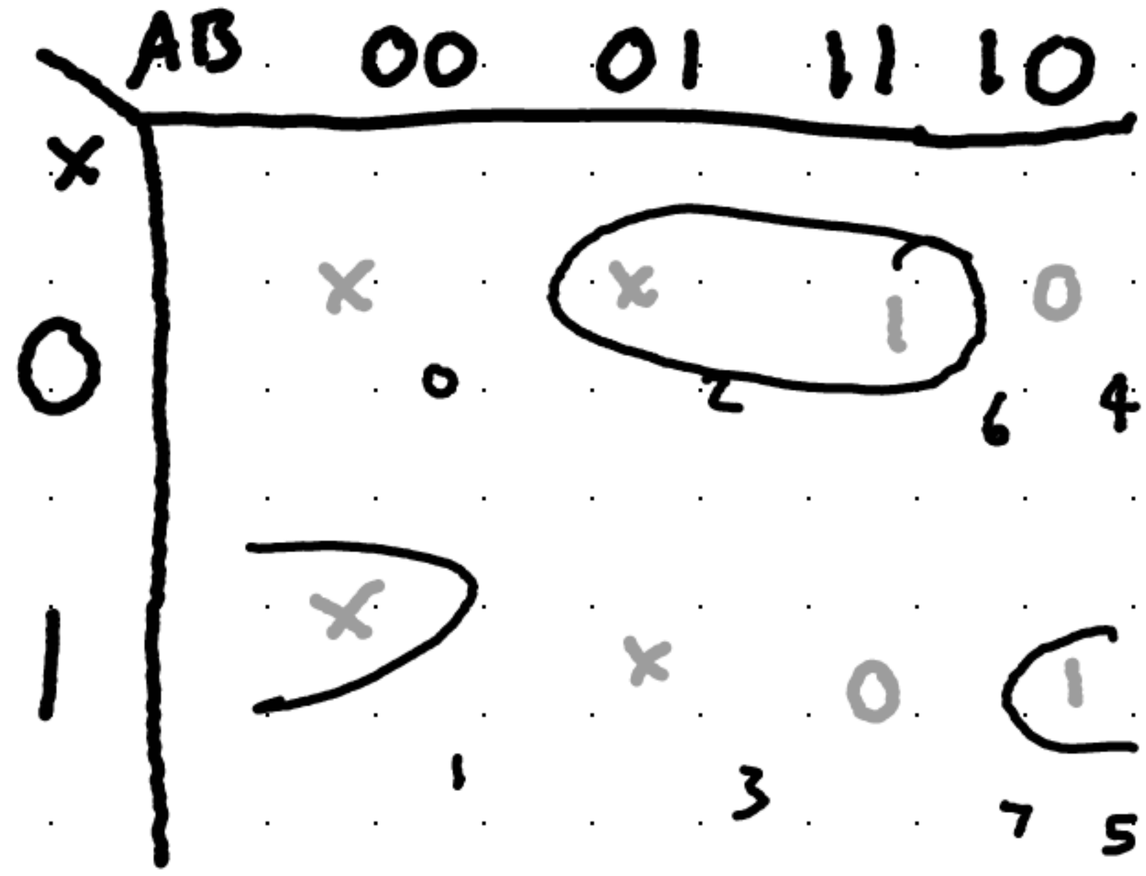
A	B	X	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	1	0	X	1	X
0	0	1	1	1	1	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	0	0	X	X	1
1	0	0	1	1	X	0	1	X
1	0	1	0	1	X	1	1	X
1	1	0	0	0	X	1	X	1
1	1	1	1	0	X	0	X	1

c)



J_A

$J_A = \bar{X}B + X\bar{B}$



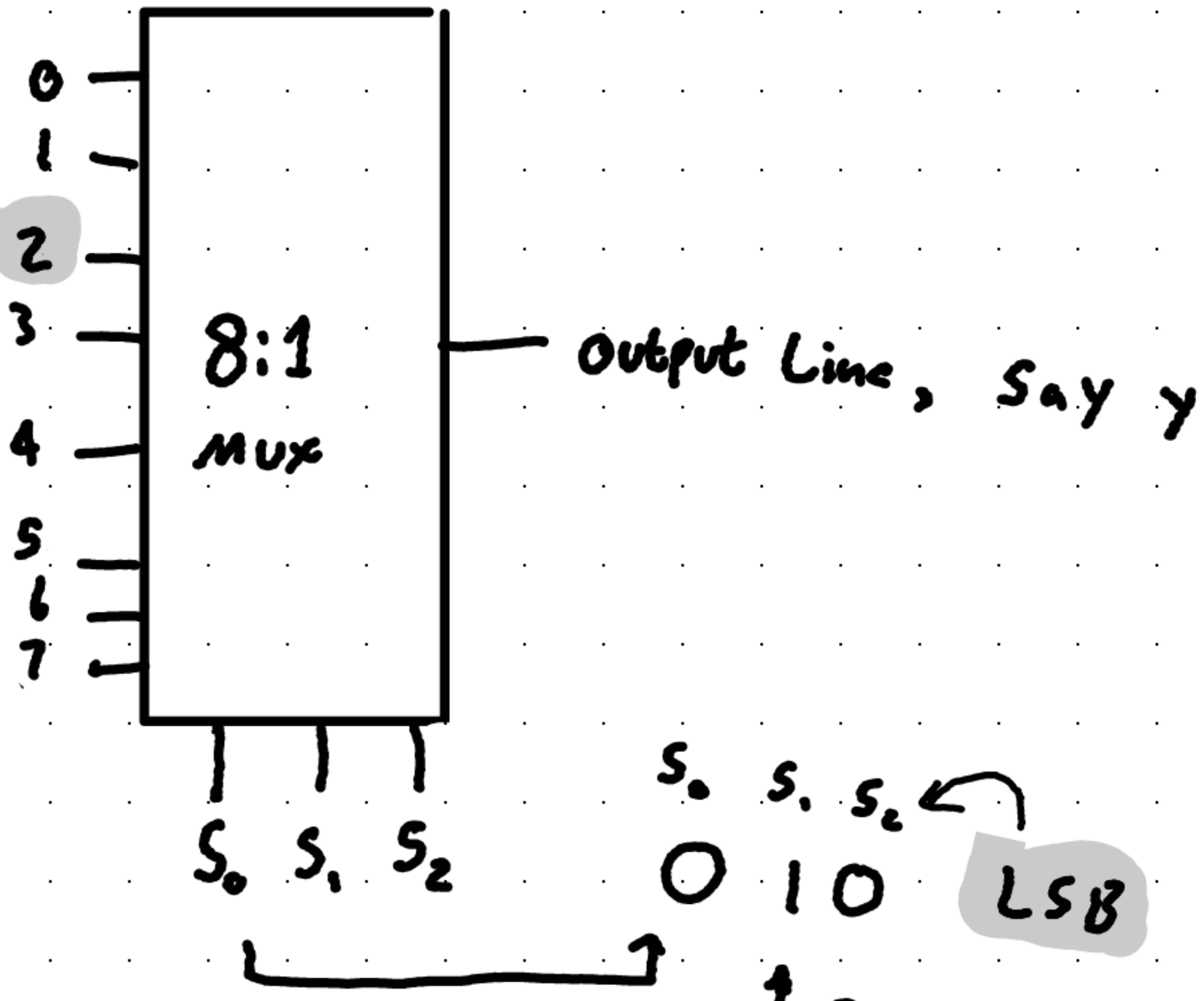
K_A

$K_A = \bar{X}B + X\bar{0}$

$J_B = 1$

$K_B = 1$

Multiplexer Elements (MUX)



- A multiplexer Element acts as a traffic controller. It allows who can go, and who can't.
- In this case, we've allowed line 2 to go We can change this by manipulating S_0, S_1 & S_2. Say we want 5. Send 101.

Examples:

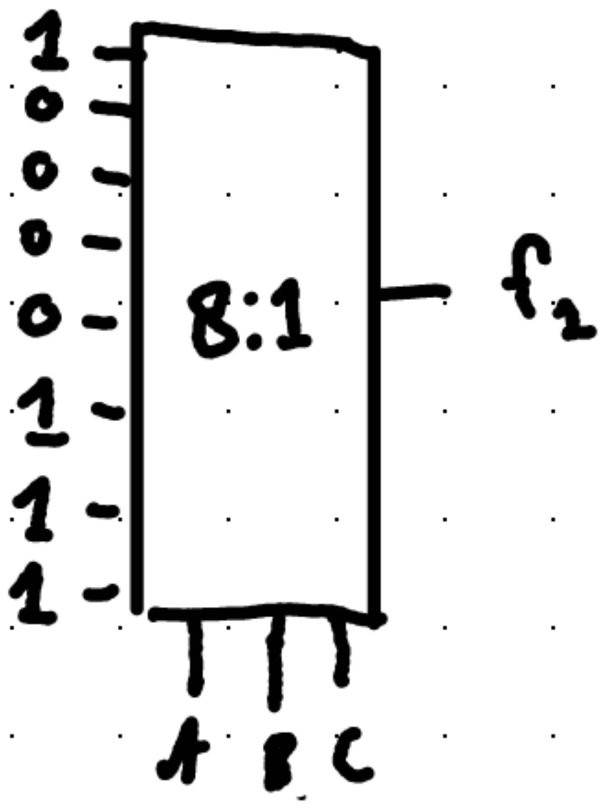
Implement the following Boolean functions using 8:1 multiplexer

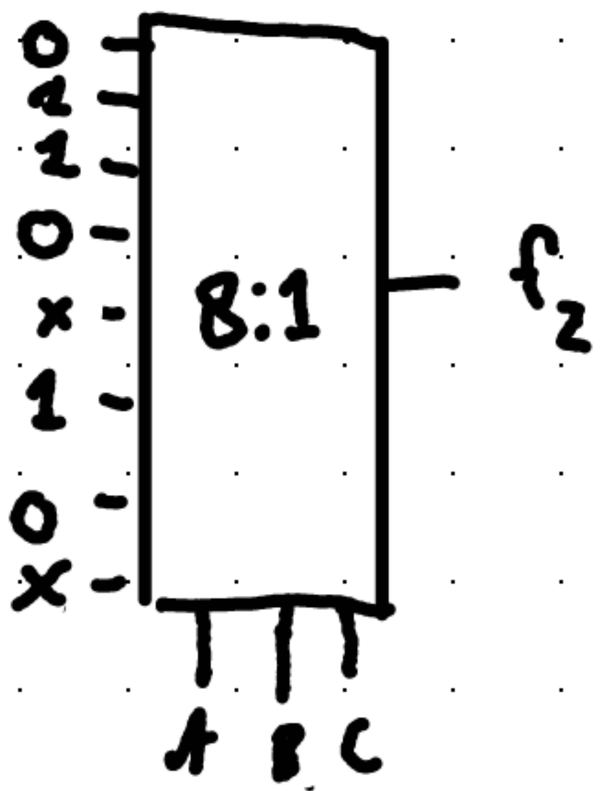
i) $f_1(A,B,C) = \sum m:(0,5,6,7)$: SOP

ii) $f_2(A,B,C) = \prod M_1(0,3,6)$: POS

We are also told that the input combinations ABC = 100 and ABC = 111 are not of concern (don't cares) for f_2 (4) (7)

a)





b)

	A	B	C	f_1	f_2
0	0	0	0	1	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	0	0
4	1	0	0	0	x
5	1	0	1	1	1
6	1	1	0	1	0
7	1	1	1	1	x

b) Implement f_1 and f_2 on a 4:1 MUX

We need to implement 1's of $\sum m:(0,5,6,7)$

- To do this, we need to consider each selection range

- 1. When $A=0$, $B=0$

  * Possible Inputs 000(0), 001(1)

  When C is zero $(\bar{C})$, the 1 value is selected

- 2. When $A=0$, $B=1$

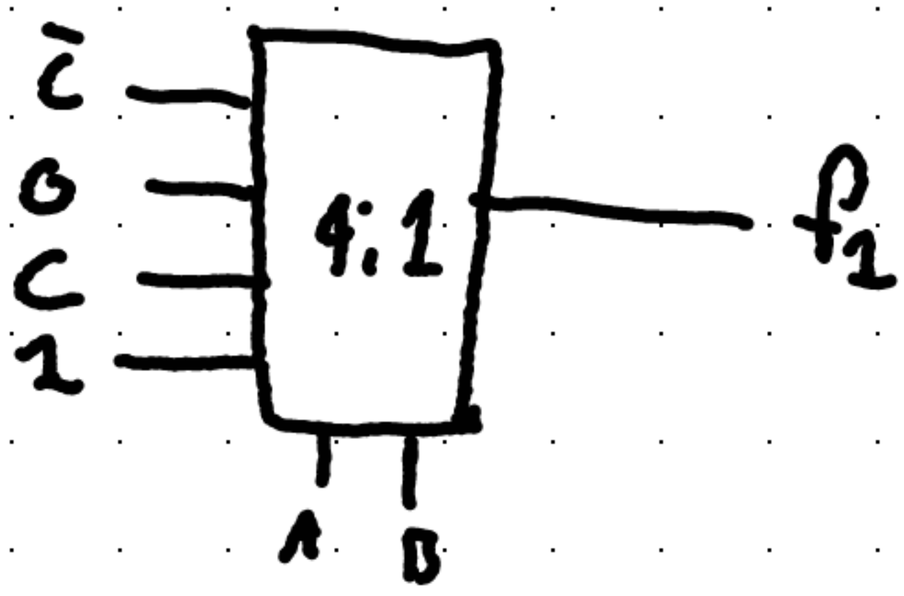
  The output is always zero. 0

- 3. When $A=1$, $B=0$

  We get what we want when $C=1$ C

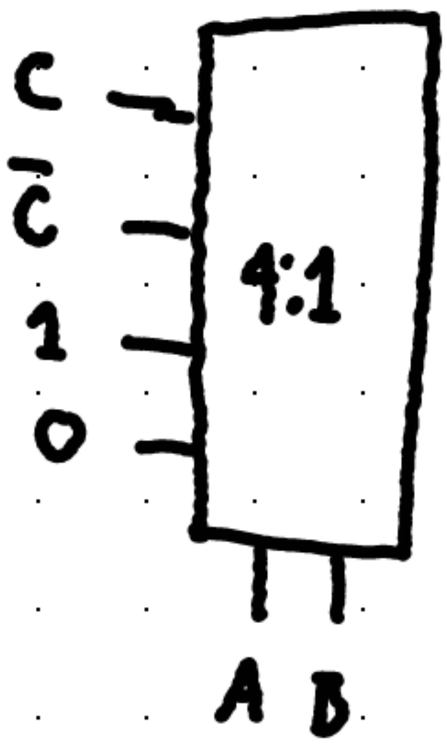
- 4. When $A=1$, $B=1$

  The output is always 1. 1



Implementation of f_2

TT $M_1(0,3,6)$



00	C
01	$\bar{C}$
10	1
11	0

c) Implement the following boolean functions using one 3:8 decoder and three OR gates.

i) $f(A,B,C) = \sum m_i(5,6,7)$

ii) $f(A,B,C) = \bar{A} \cdot (B + \bar{C})$

iii) $f(A,B,C) = \bar{A} \cdot C + A \cdot \bar{B} \cdot \bar{C}$

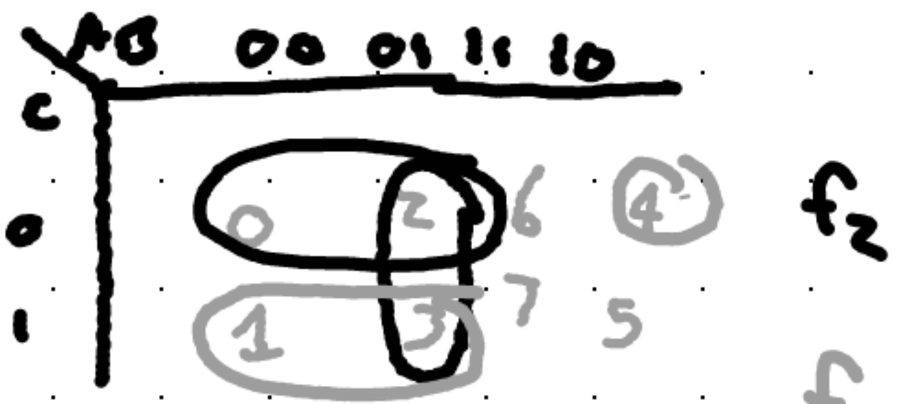
$f_1 = \sum m_i(5,6,7)$

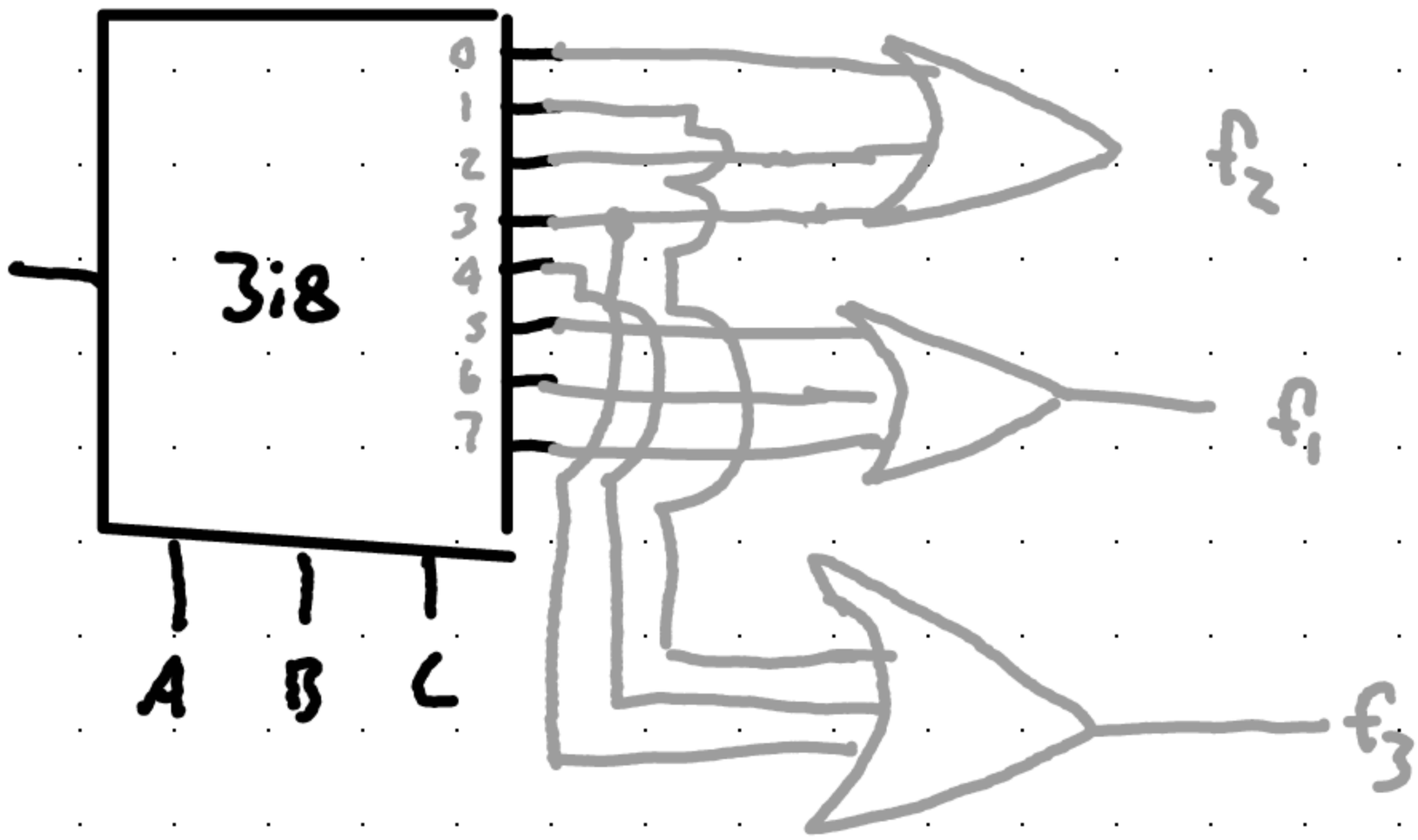
$f_2 = \bar{A}B + A\bar{C}$

$= \sum m_i(0,2,3)$

$f_3 = \bar{A}C + A\bar{B}\bar{C}$

$= \sum m_i(1,3,4)$





$f_1 = \sum m_i(5,6,7)$

$f_2 = \bar{A}B + A\bar{C}$

$= \sum m_i(0,2,3)$

$f_3 = \bar{A}C + A\bar{B}\bar{C}$

$= \sum m_i(1,3,4)$

Example:

A half adder circuit has two inputs A and B, and two outputs, S and C_o

Pattern detector