SEE 3243/4243
Registers & Counters

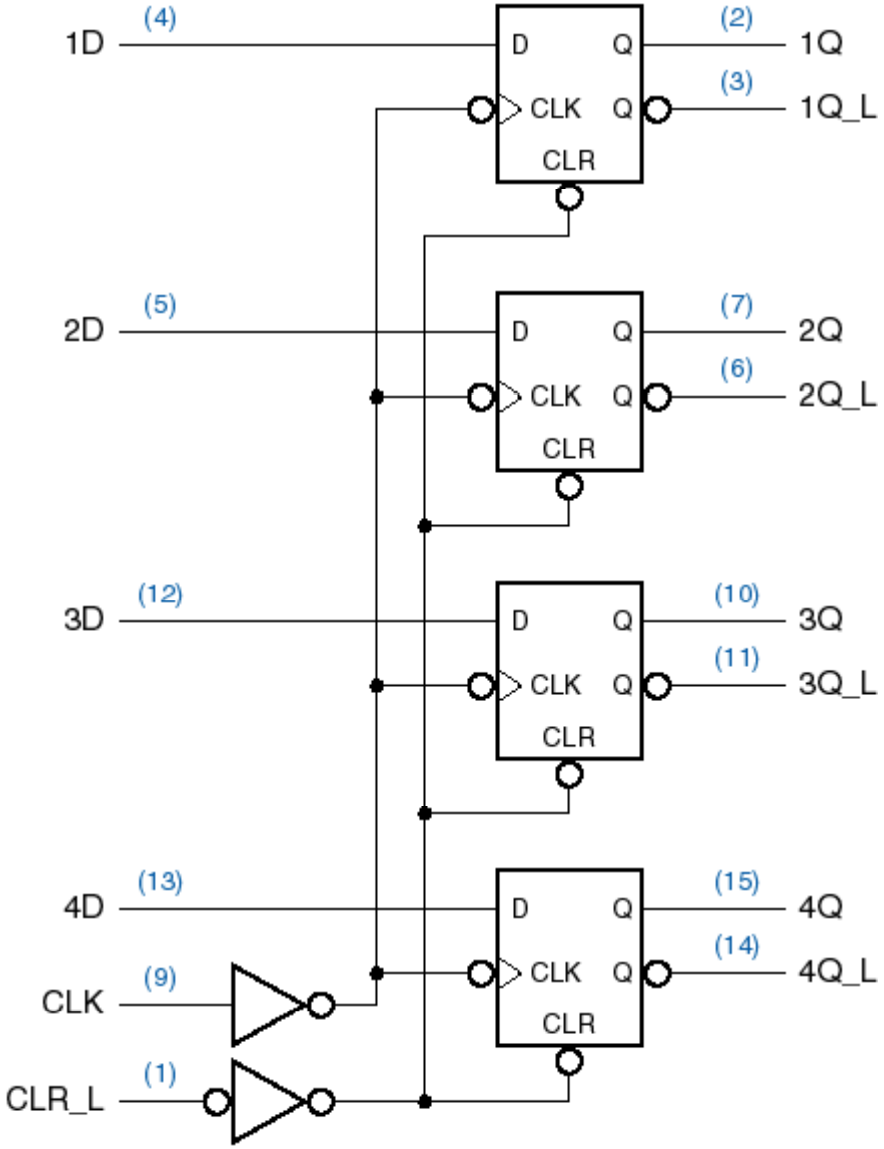
Week 8

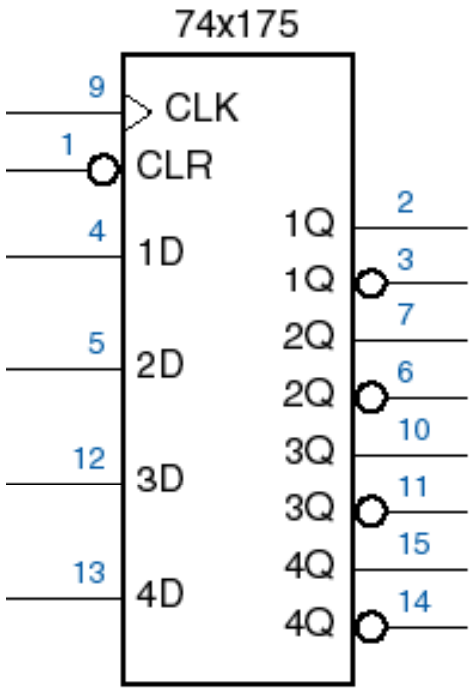
- Storage Registers
- Shift Registers
- Counters
- Design of Synchronous Counters

Registers

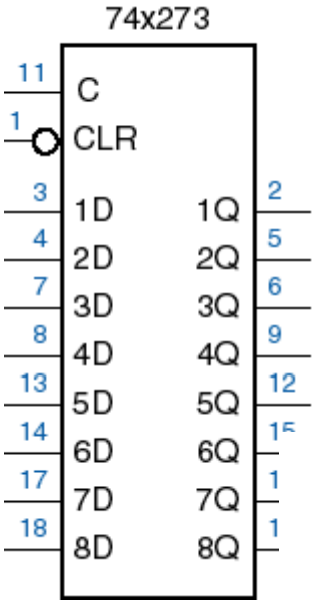
- Register is a group of flip-flops/memory elements that work together to store data or instructions and shift a group of bits or a binary word.
- Variations:
  - Register file - a few registers, each accessible by a register address. Sort of a small memory array.
  - Shift register - temporary circuit able to shift or move the stored word either left or right.
    - the bits stored can be moved/shifted from 1 element to another adjacent element.
    - all the storage registers are actuated simultaneously by a single input clock/shift pulse.
  - Buffer register - a temporary data storage circuit able to store a digital word.
- Sometimes use special names
  - accumulators, program counters, index registers, stack pointer, status register, etc.

Multibit registers and latches

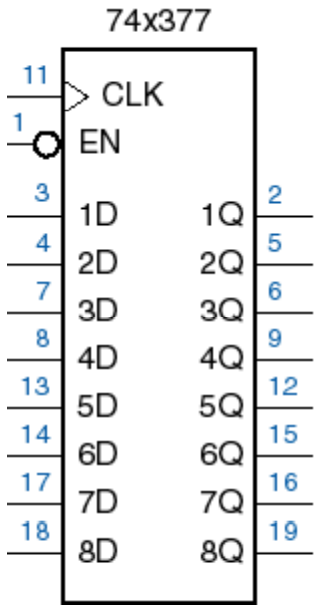


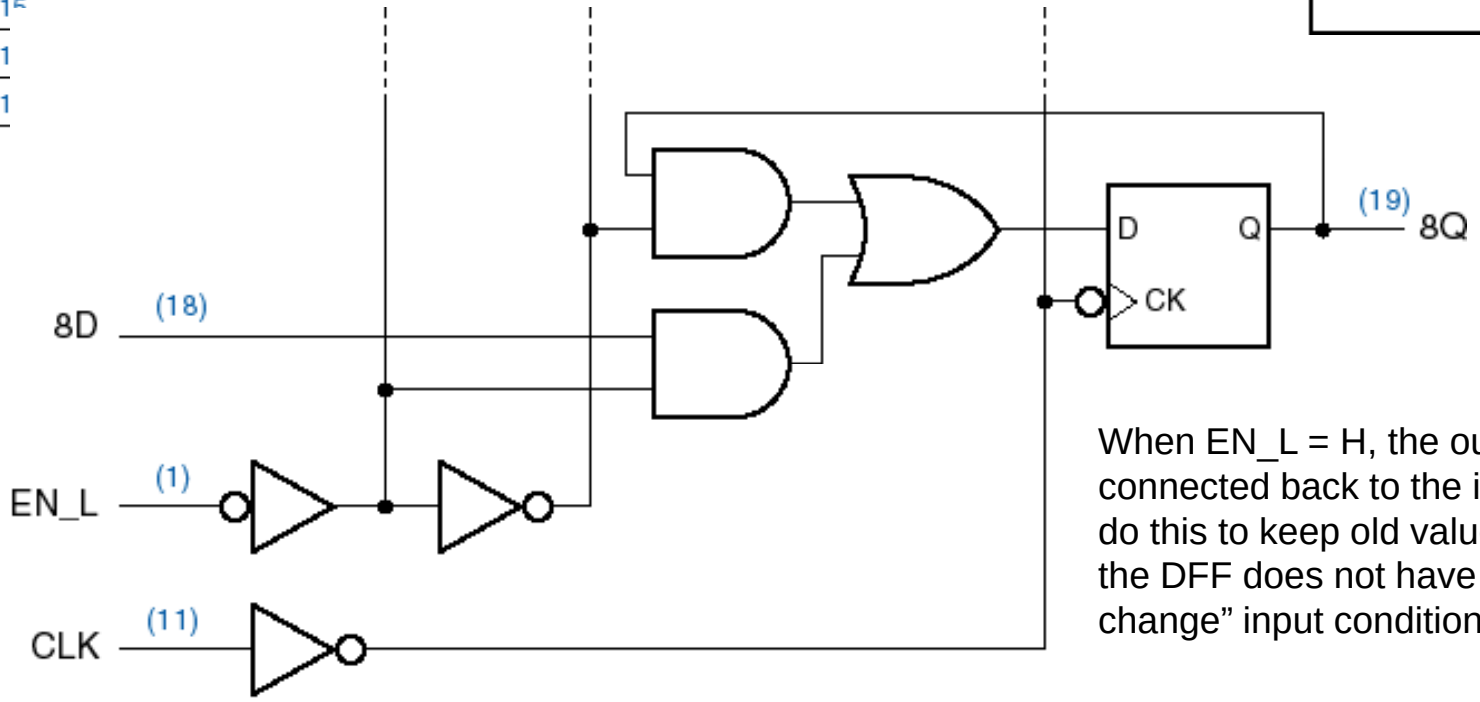


Octal (8-bit) Register & Latch



- 74x273
  - asynchronous clear
- 74x377
  - clock enable

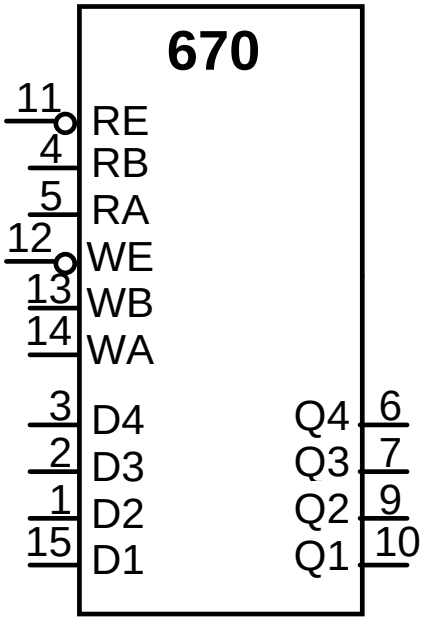




When EN_L = H, the output is connected back to the input. Must do this to keep old value because the DFF does not have a “no change” input condition

74x670 4x4 Register File with Tri-state Outputs

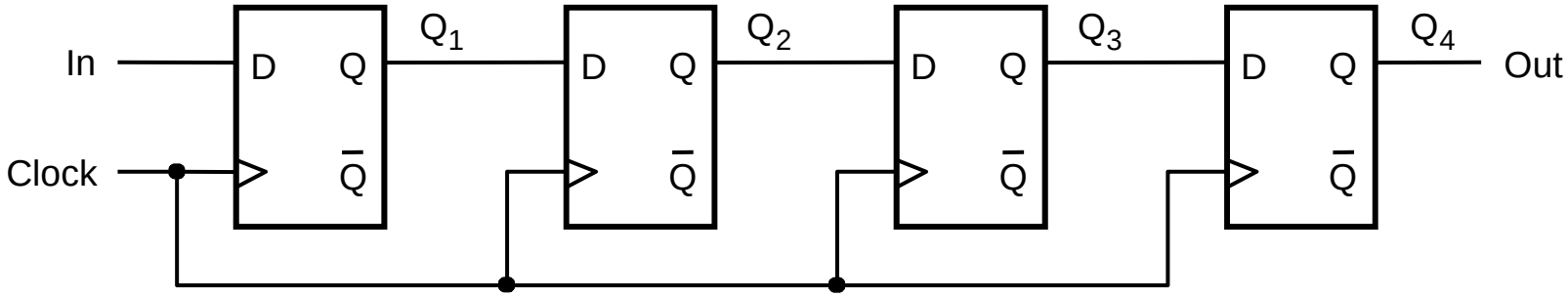
- The 74x670 device contains 16 D flip-flops organized into four words of four flip-flops each.
- Each register in the register file is called a **word** and is identified by a unique index or **address**
- Word contents read or written
  - Separate Read and Write Enables (RE, WE)
  - Separate Read and Write Address (RA, RB, WA, WB) - binary encodings of one of four registers to be read or written
  - Data Input, Q Outputs
- On a read, the selected word is multiplexed to the outputs.
- On a write, data present on D4-D1 inputs are stored in the selected word



Shift Registers

- Register components that shift as well as store
- For handling serial data, such as RS-232 and modem transmission and reception, Ethernet links, SONET, etc.
- Data moves from left to right (or from top to bottom). On every shift pulse, the contents of a given flip-flop are replaced by the contents of the flip-flop to its left. The leftmost device receives its inputs from the rightmost.
- Because flip-flop propagation times far exceed hold times, the values are passed correctly from one stage to the next

Basic Shift Register



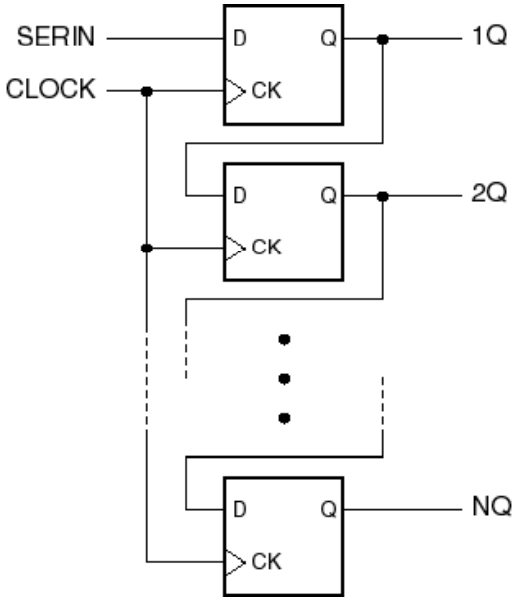
(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

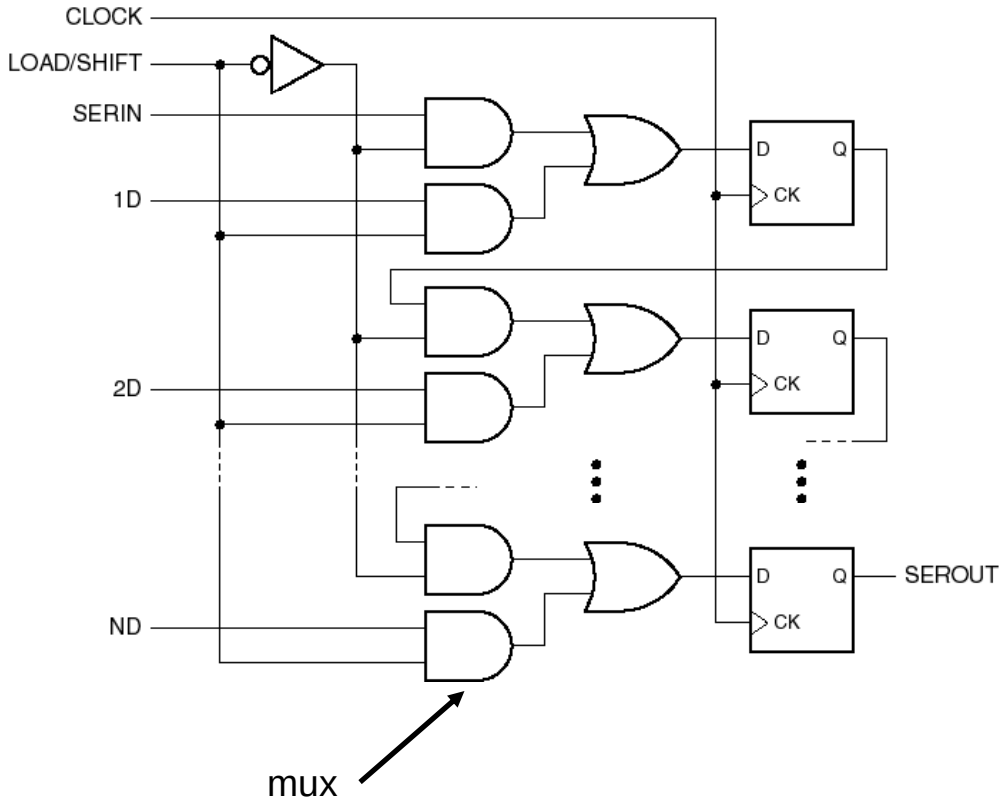
(b) A sample sequence

Parallel-to-serial conversion and vice versa

- Serial to parallel
  - Use a serial-in, parallel-out shift register

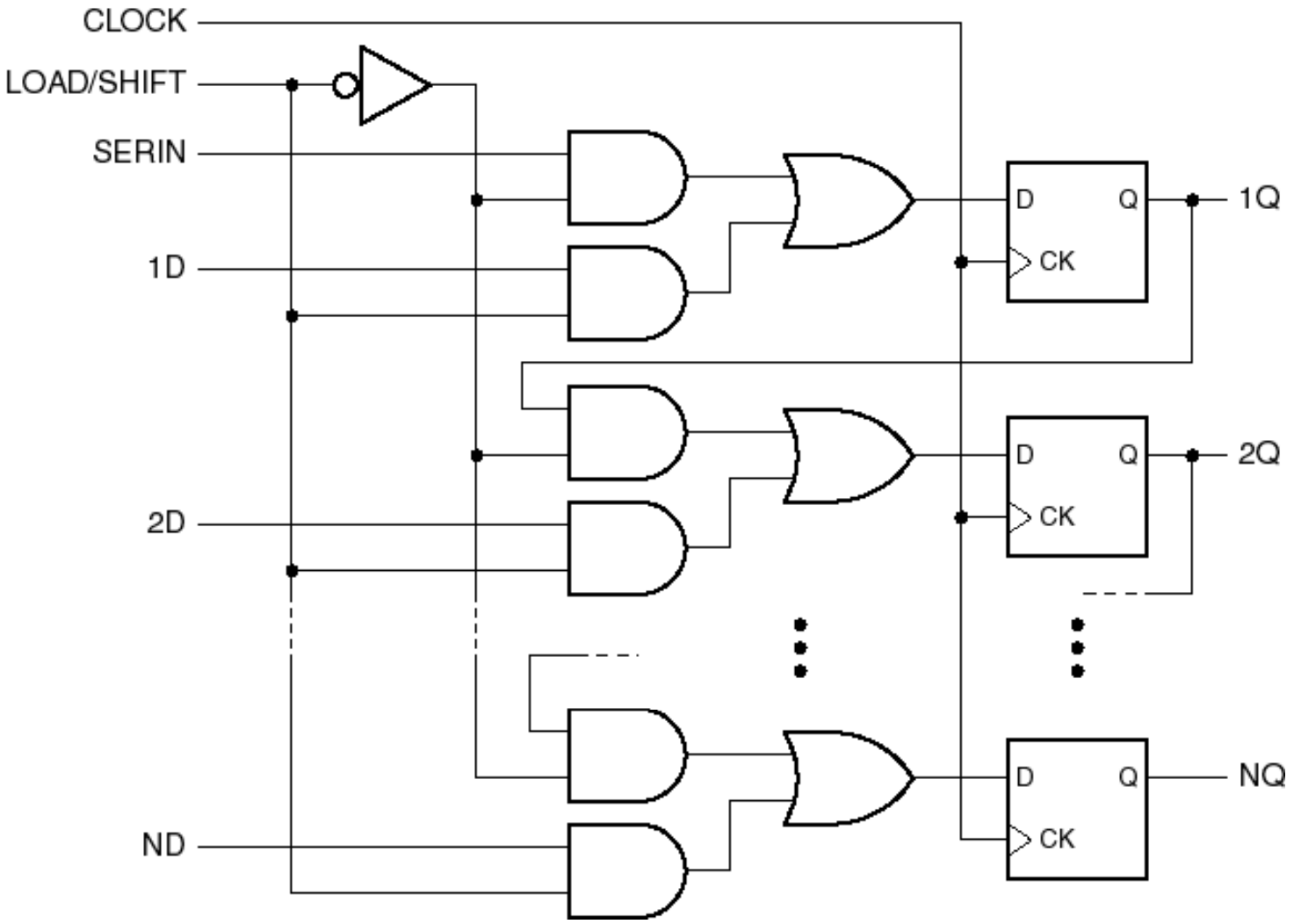


- Parallel to serial
  - Use parallel-in, serial-out shift register



Do both

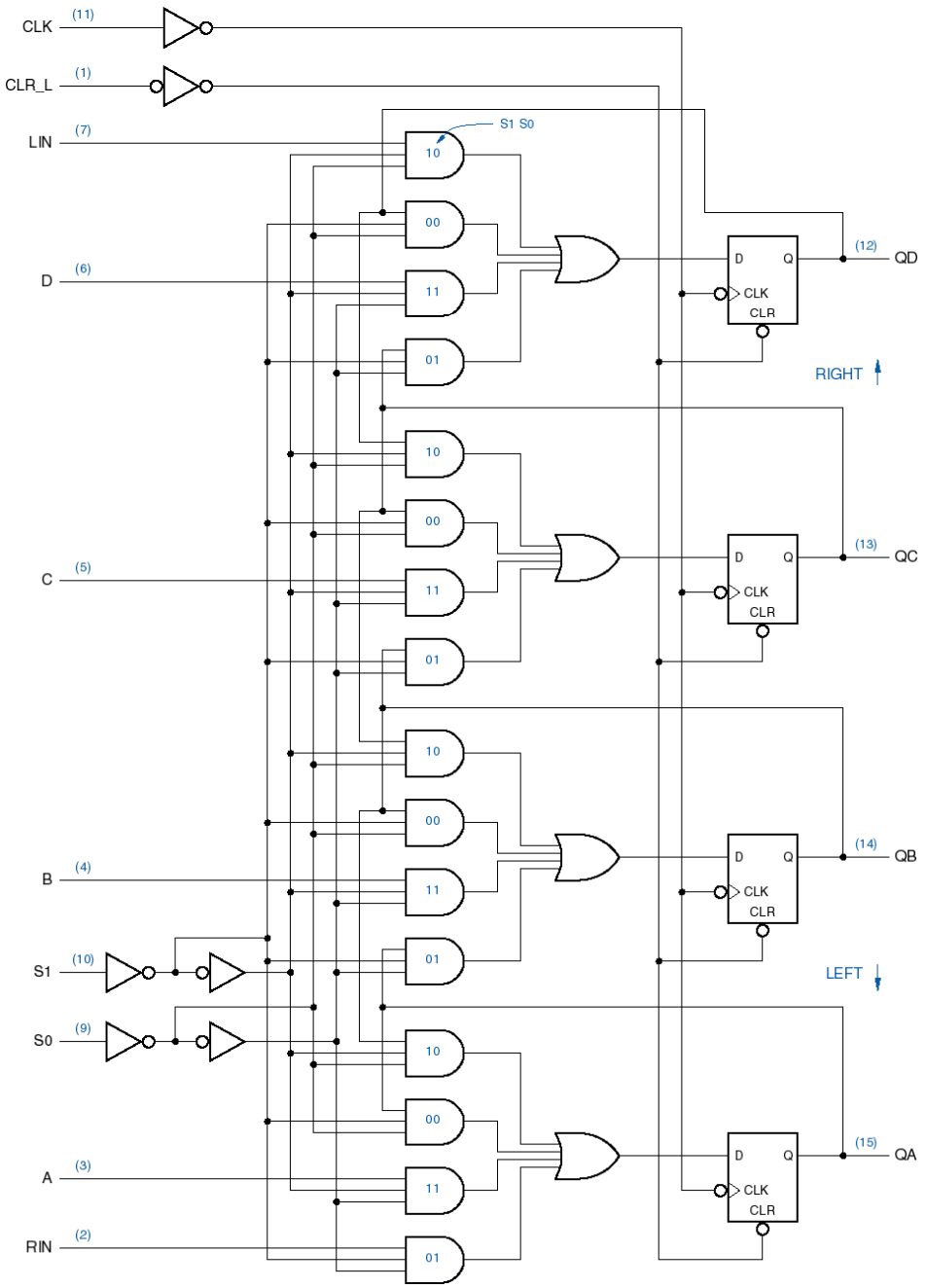
- Parallel-in, parallel-out shift register



"Universal" shift register 74x194

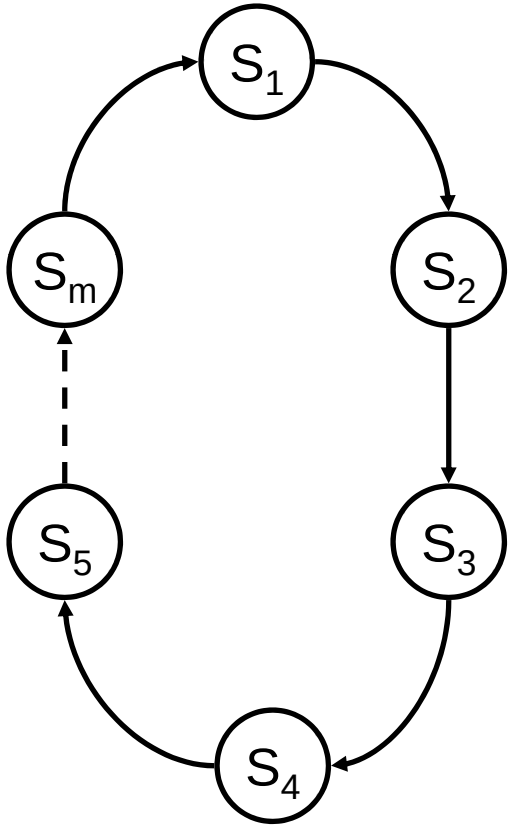
S1	S0	Operation
0	0	Hold
0	1	Shift Up
1	0	Shift Down
1	1	Parallel Load

- S1 & S0 selects which line is connected to D input.
- There's 4 possible inputs to each DFF.



Counters

- A circuit that produces a well-defined output pattern sequence
  - 3 Bit Up-counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...
  - 3 Bit Down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...
  - Binary vs. BCD vs. Gray Code Counters
- The output pattern = state of the counter
- Total number of states = modulus of counter
  - Counter with m states = *modulus-m counter* or *mod-m counter*
- Counting sequence often shown using a state diagram or state transition diagram
- A counter is a "degenerate" finite state machine (FSM) circuit where the state is the only output - more on FSM next week

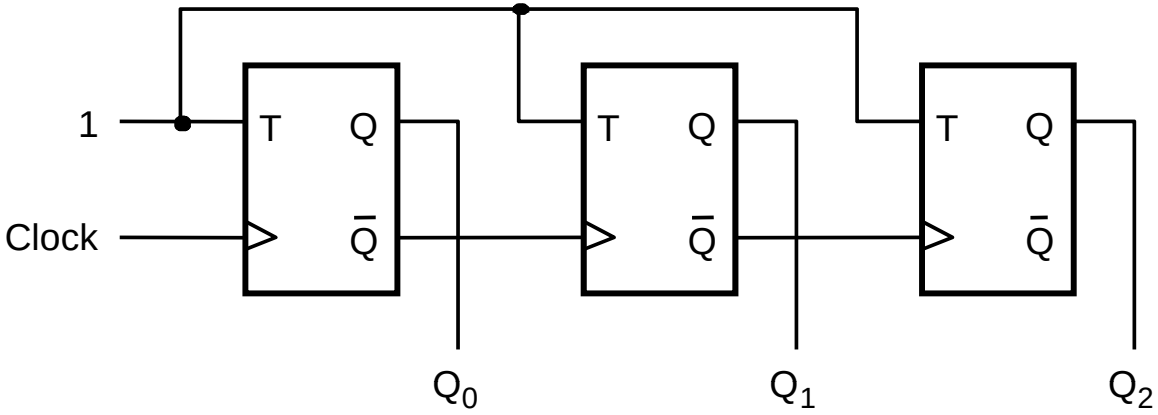


State transition diagram of a counter

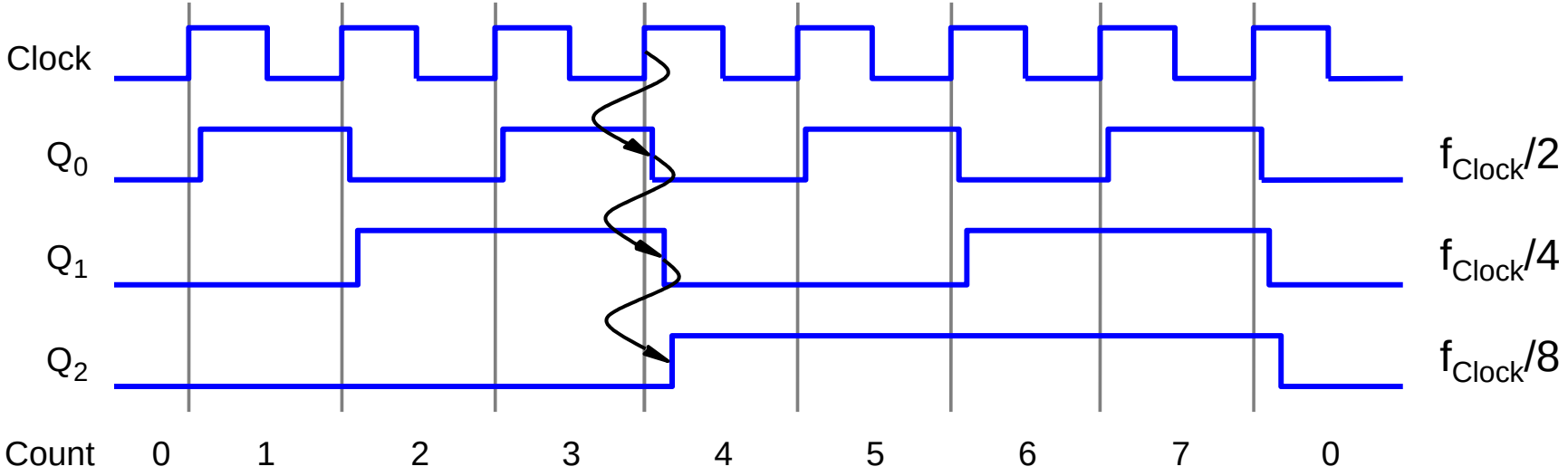
Asynchronous Binary Counters

- Binary counters = counters whose counting sequence corresponds to binary numbers
- Modulus of a binary counter is 2^n, where n is # flip-flops
- Also known as ripple counter since a change in Q_i flip-flop toggles the Q_{i+1} flip-flop
  - Effect of counting must ripple thru the counter
  - Only first FF connected to clock signal
- Rippling affects overall delay between count pulse and when the count stabilizes
  - Worst case in $n \times t_{pd}$ (t_{pd} is propagation delay of each FF)
- However, ripple counters are useful as frequency dividers
  - Frequency at output of Q_{i+1} flip-flop is half at output of Q_i
  - Frequency of last FF of n-stage counter is $f_{input}/2^n$

A 3-bit Asynchronous Up-Counter



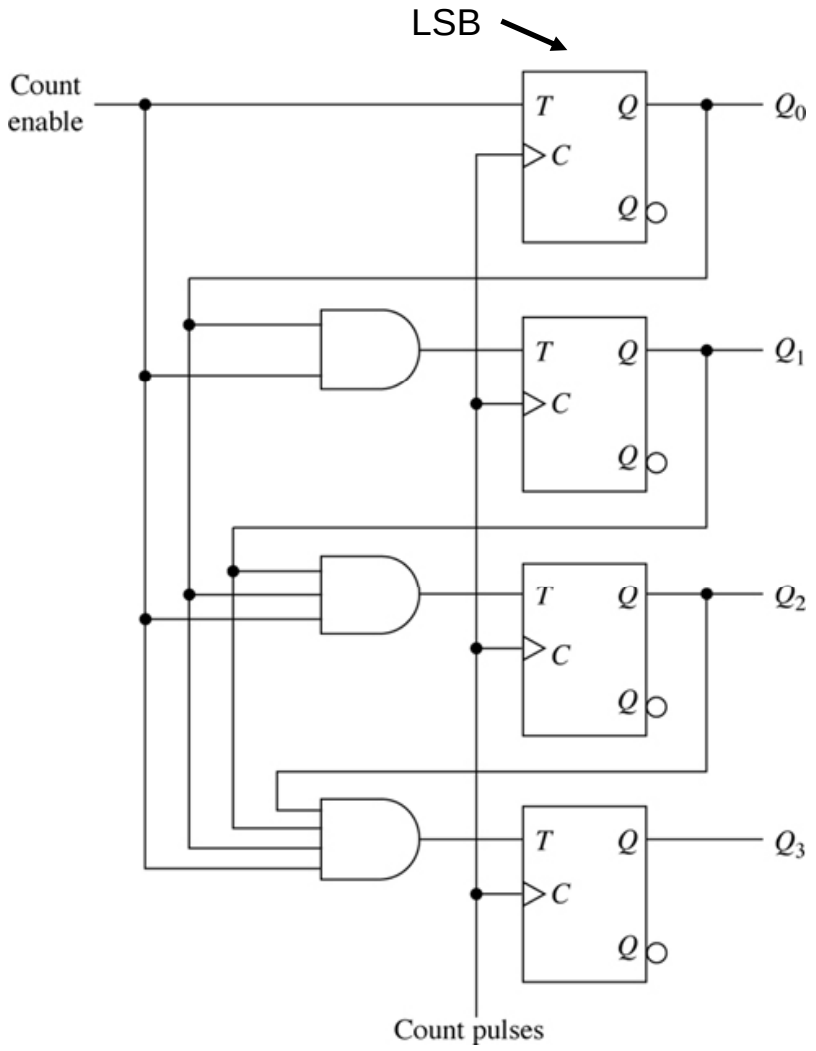
(a) Circuit



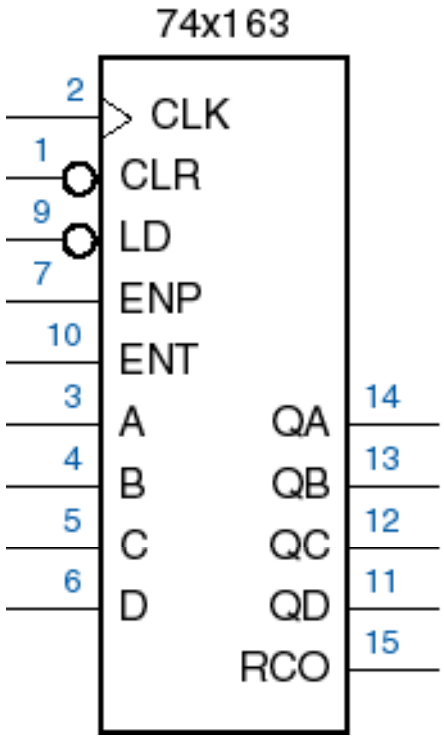
(b) Timing diagram showing ripple effect

Synchronous Counters

- All FFs are triggered simultaneously (in parallel) by clock input pulses.
- All outputs change simultaneously
- Simple counters use TFF or JKFF
- Only LSB FF has its JK inputs permanently at HIGH level.
- JK inputs of the others FFs are driven by some combination of FF outputs.



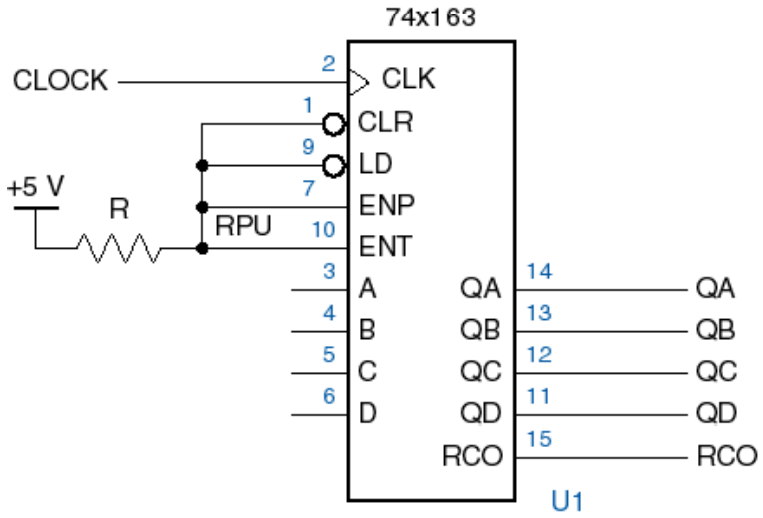
74x163 MSI 4-bit counter

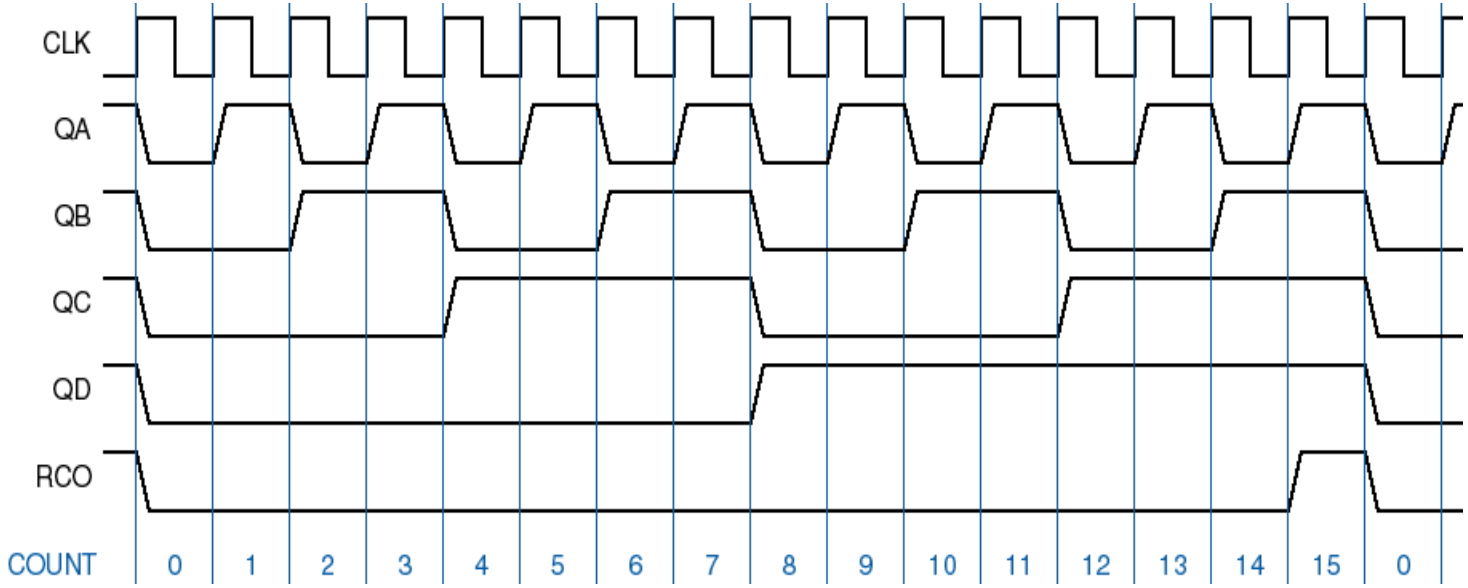


Inputs				Current State				Next State			
CLR_L	LD_L	ENT	ENP	QD	QC	QB	QA	QD*	QC*	QB*	QA*
0	x	x	x	x	x	x	x	0	0	0	0
1	0	x	x	x	x	x	x	D	C	B	A
1	1	0	x	x	x	x	x	QD	QC	QB	QA
1	1	x	0	x	x	x	x	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

Free-Running 4-bit ’163 Counter

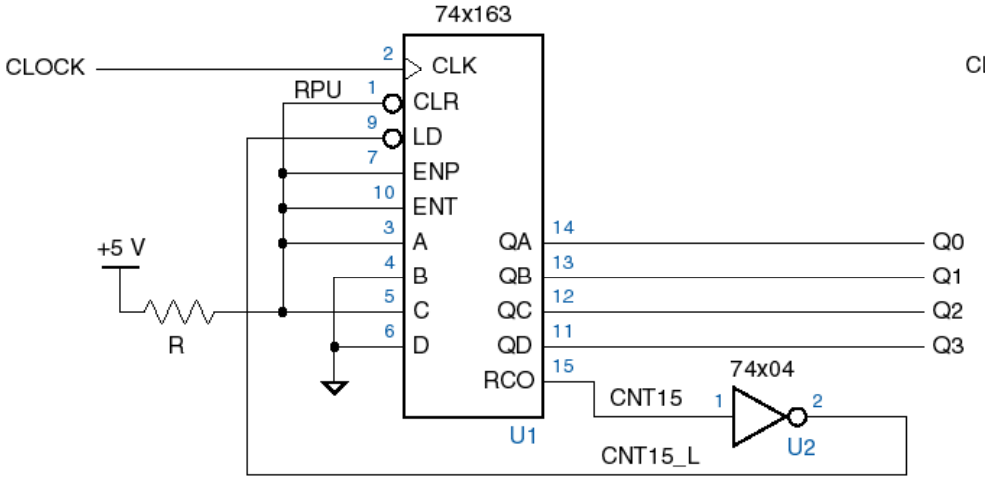
- “divide-by-16” counter

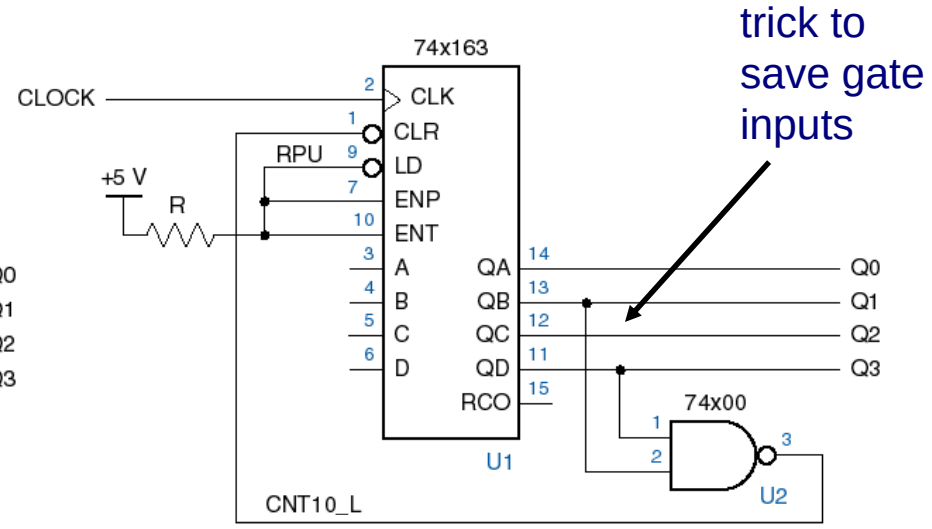




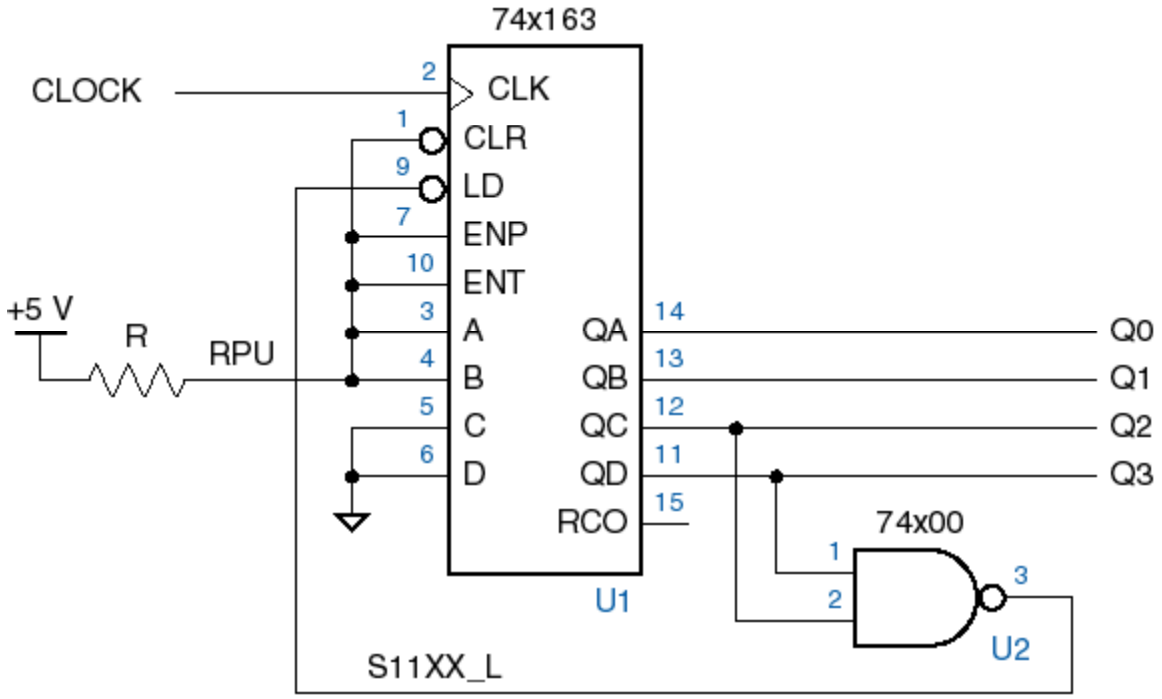
Modified Counting sequence: mod-11 Counter

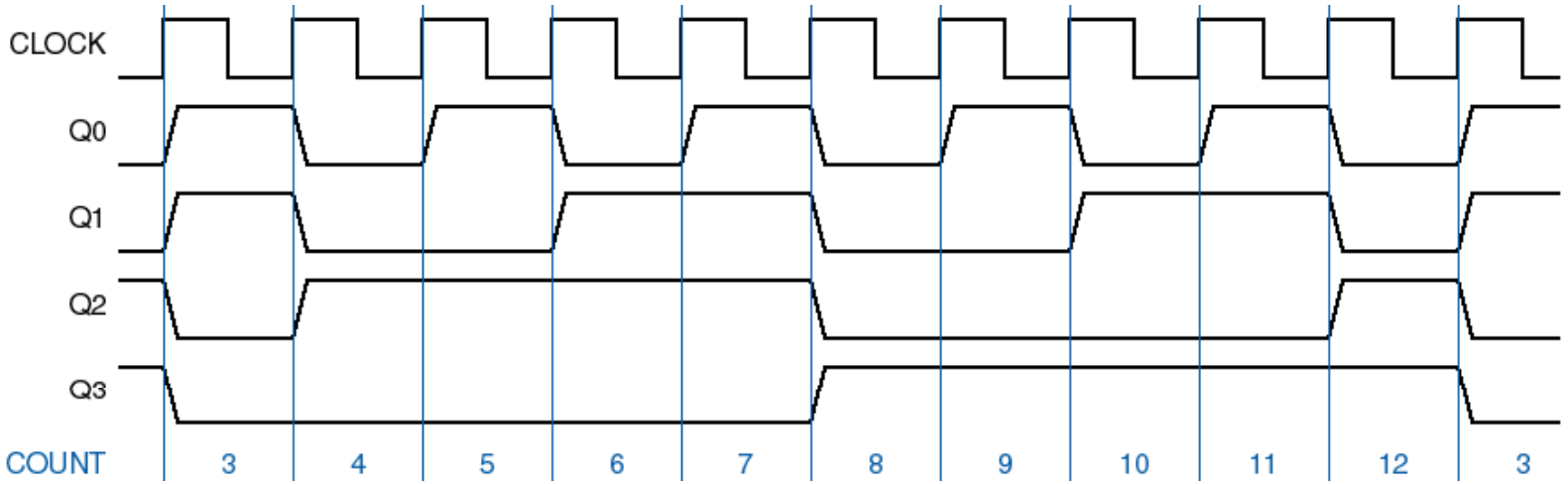
- Load 0101 (5) after Count = 15
- 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 5, 6, …
- Clear after Count = 1010 (10)
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 1, 2, 3, …





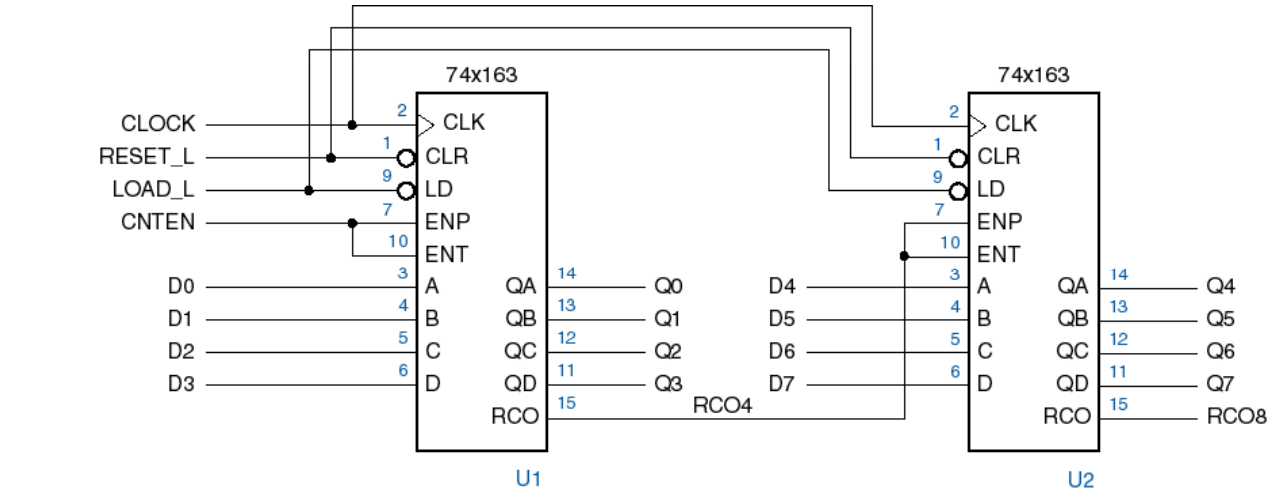
Counting from 3 to 12

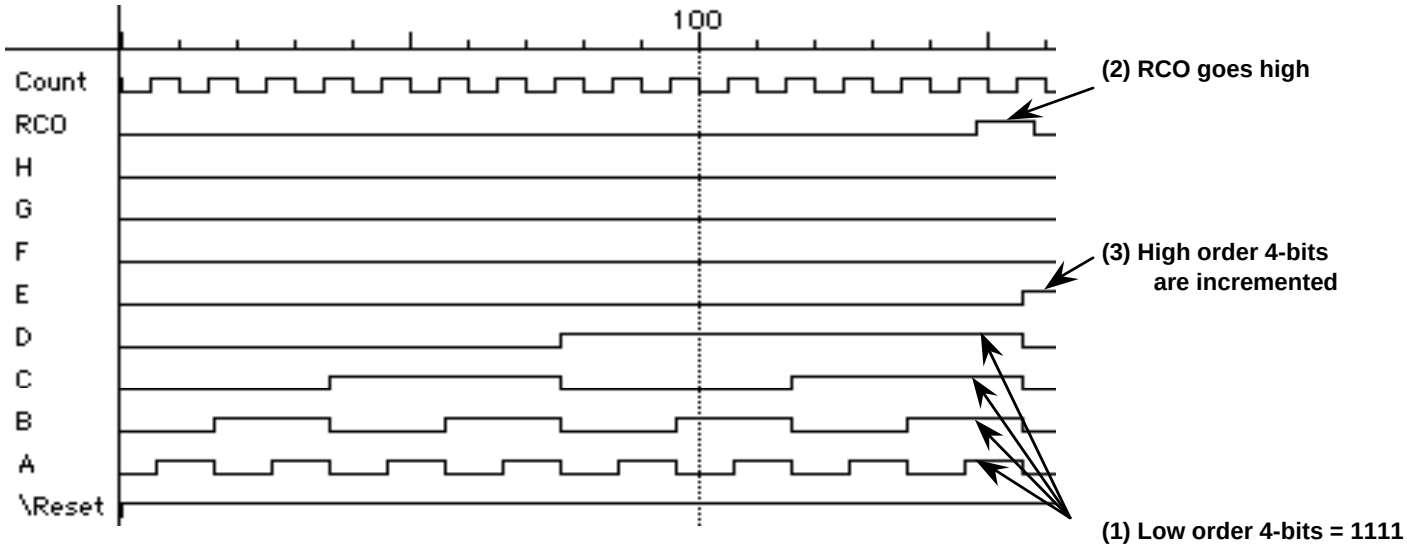




Cascading Counters

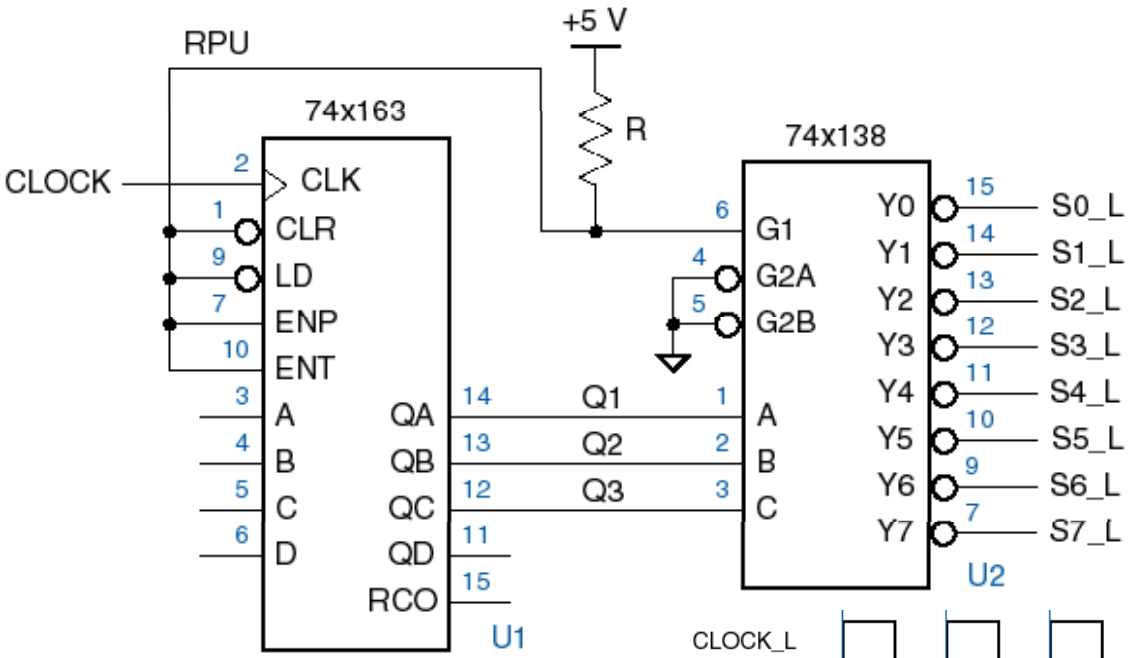
- RCO (ripple carry out) is asserted in state 15, if ENT is asserted.

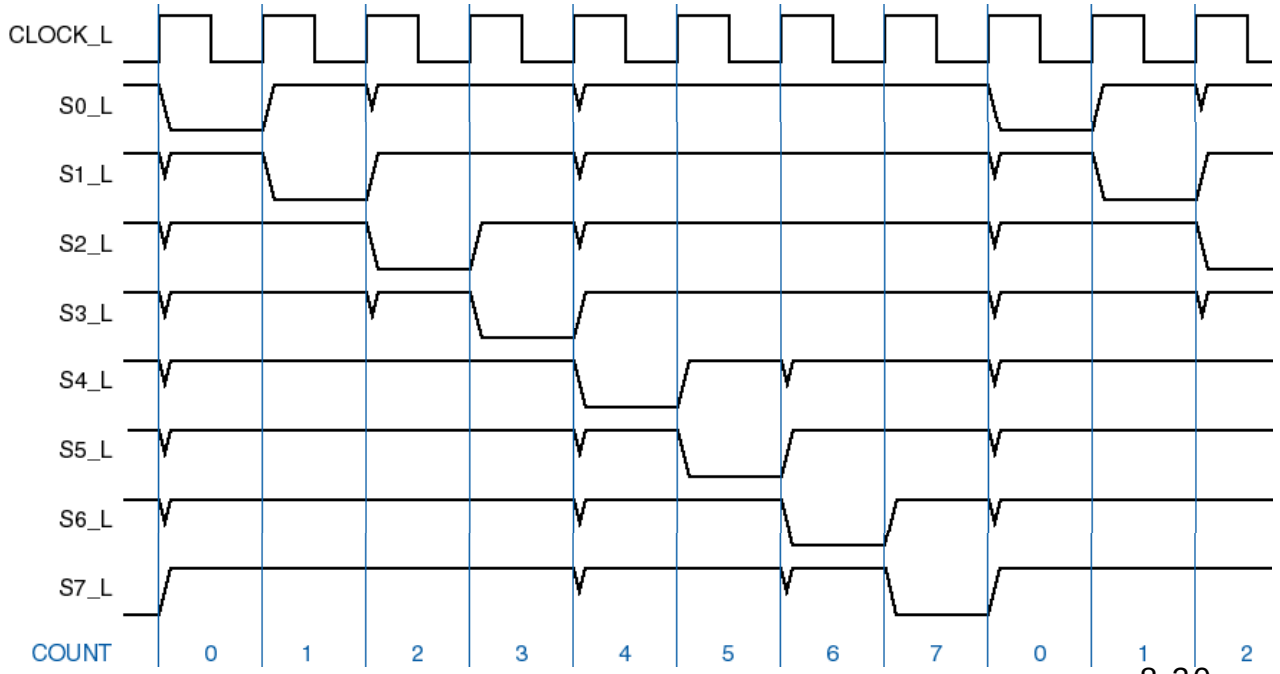




- First stage RCO enables second stage for counting
- RCO asserted soon after stage enters state 1111
- also a function of the T Enable
- Downstream stages lag in their 1111 to 0000 transitions
- Affects Count period and decoding logic

Decoding binary-counter states

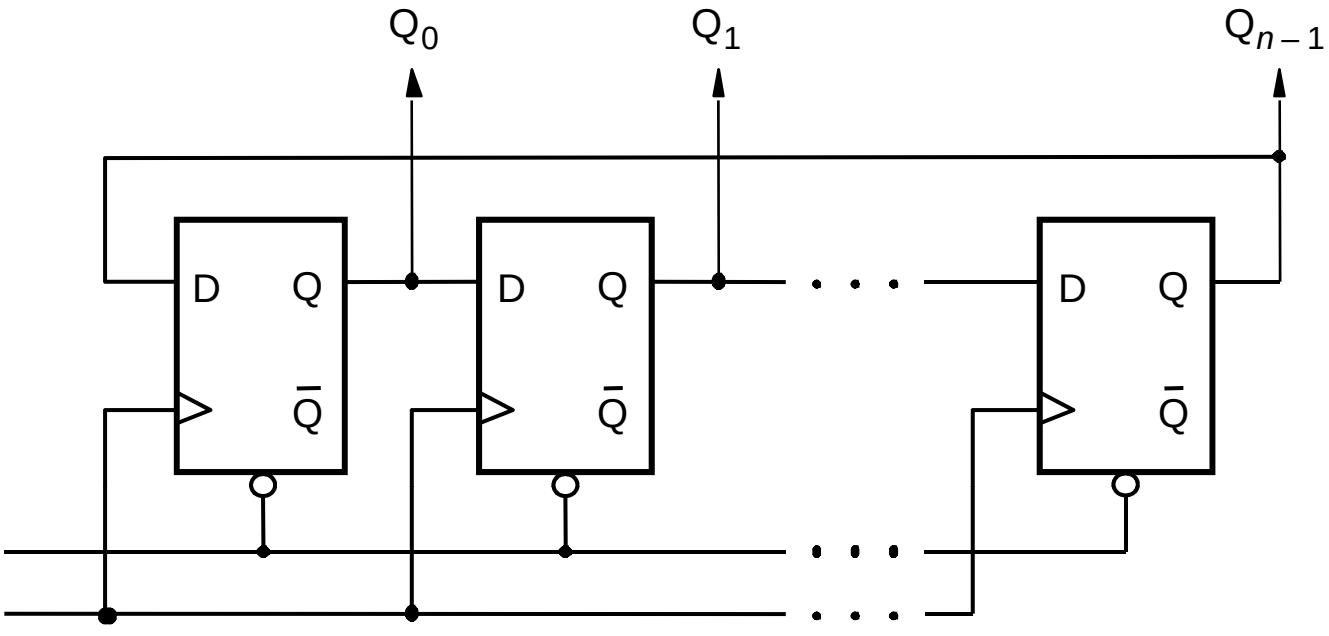


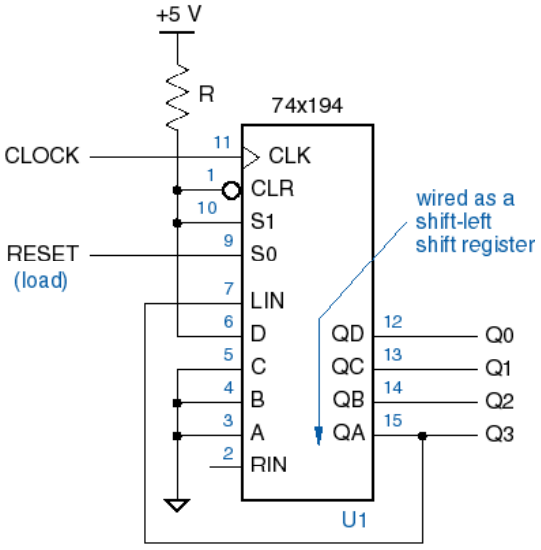


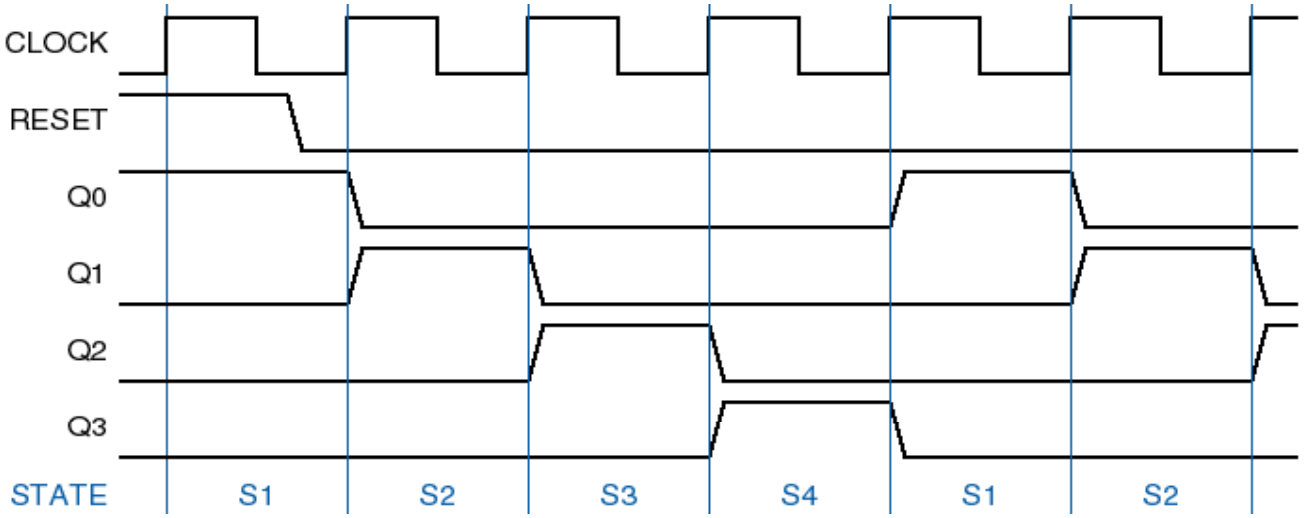
- Glitches may or may not be a concern.

Ring Counter

- Is a circulating shift register

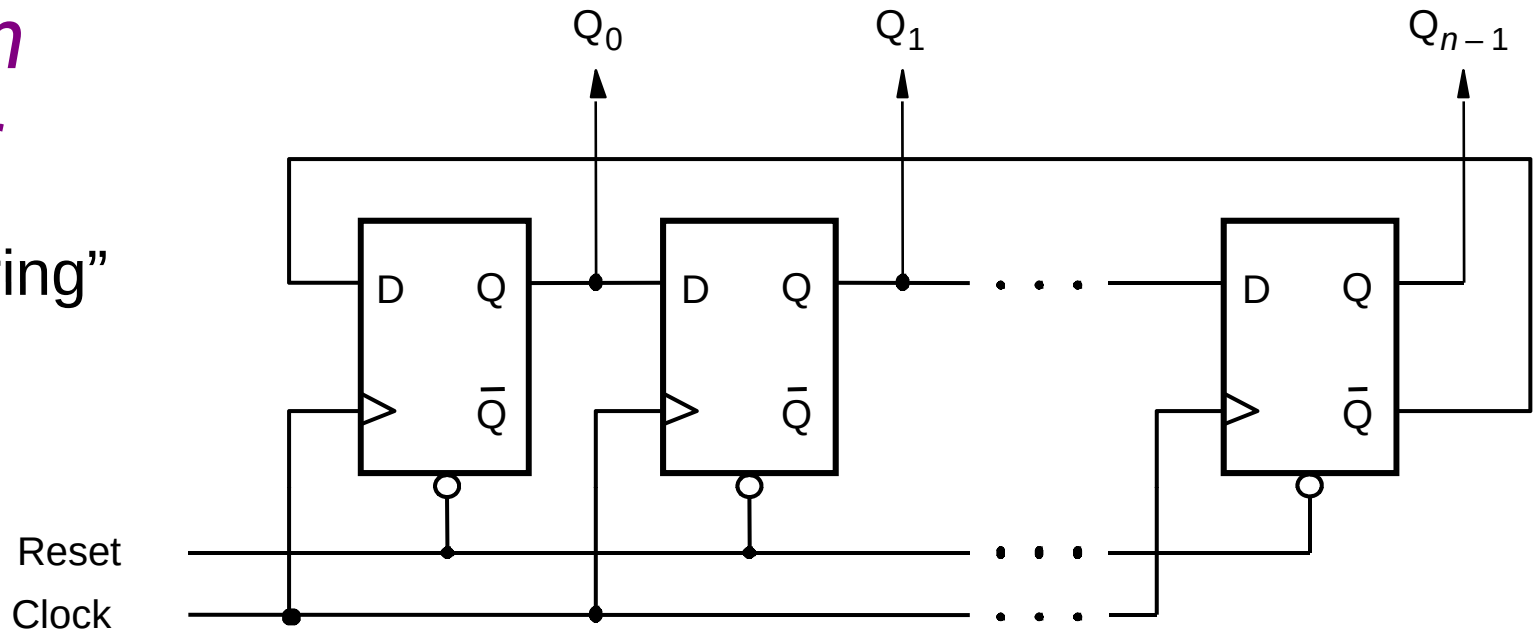


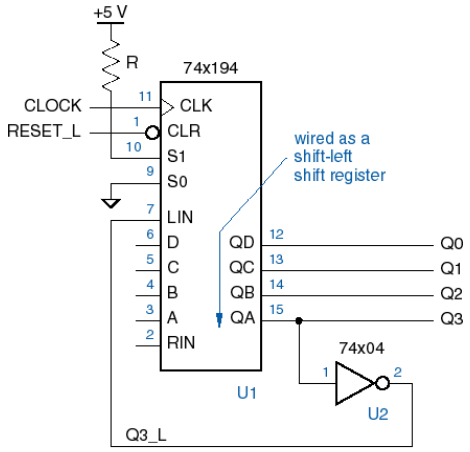


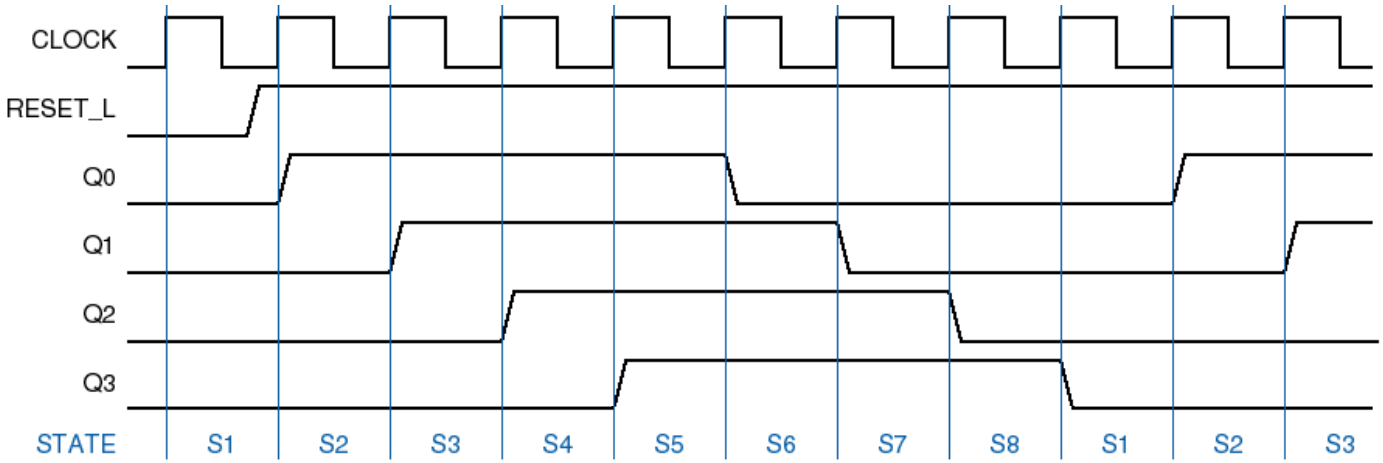


Johnson Counter

- "Twisted ring" counter



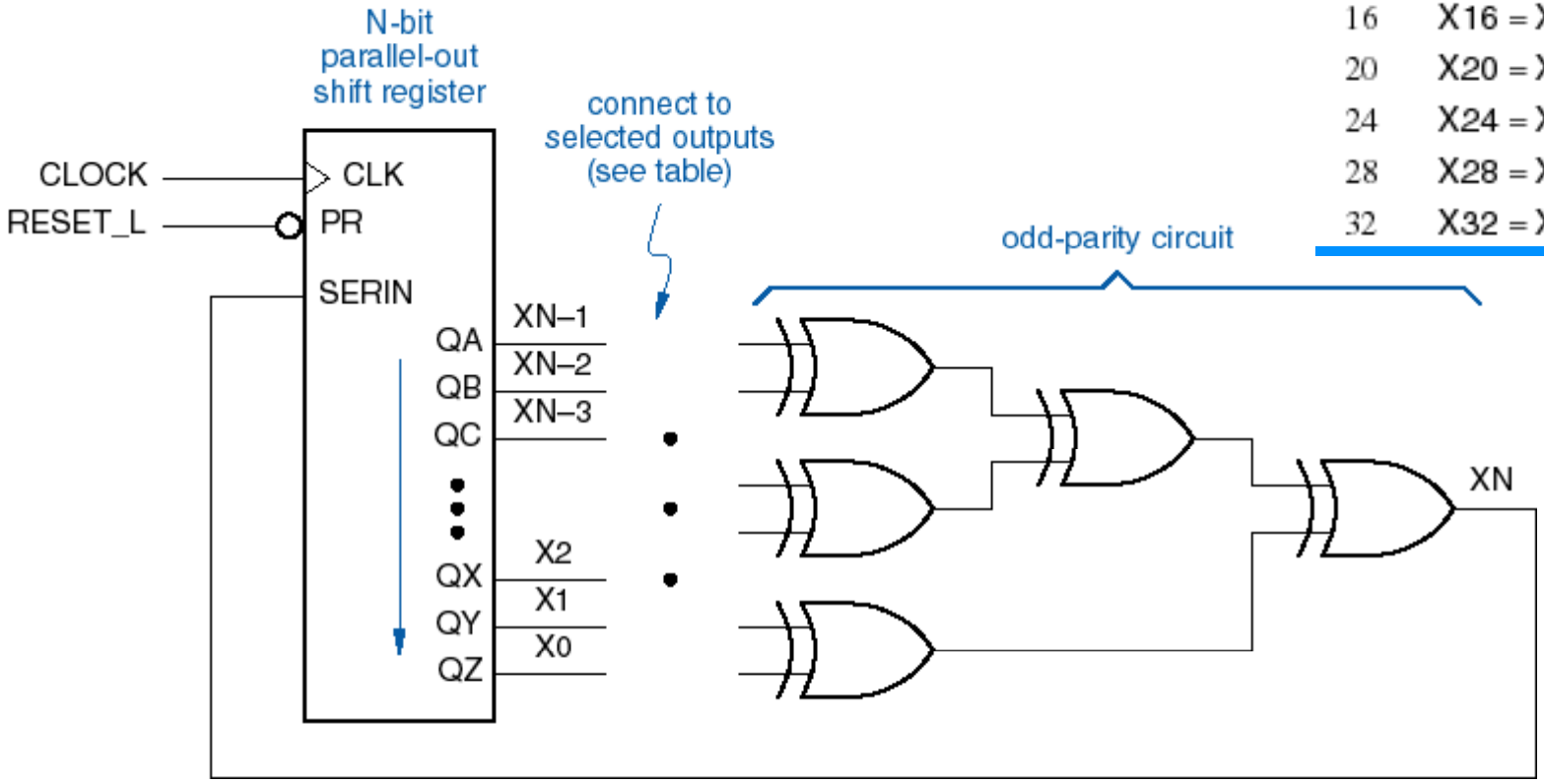




LFSR Counters

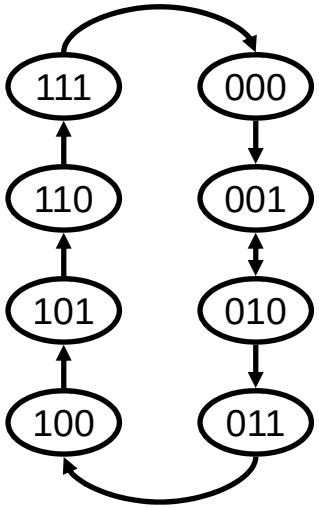
- Pseudo-random number generator
- 2^n - 1 states before repeating
- Same circuits used in CRC error checking in Ethernet networks, etc.

n	*Feedback Equation*
2	X2 = X1 ⊕ X0
3	X3 = X1 ⊕ X0
4	X4 = X1 ⊕ X0
5	X5 = X2 ⊕ X0
6	X6 = X1 ⊕ X0
7	X7 = X3 ⊕ X0
8	X8 = X4 ⊕ X3 ⊕ X2 ⊕ X0
12	X12 = X6 ⊕ X4 ⊕ X1 ⊕ X0
16	X16 = X5 ⊕ X4 ⊕ X3 ⊕ X0
20	X20 = X3 ⊕ X0
24	X24 = X7 ⊕ X2 ⊕ X1 ⊕ X0
28	X28 = X3 ⊕ X0
32	X32 = X22 ⊕ X2 ⊕ X1 ⊕ X0



Design of 3-bit Binary Upcounter

- This procedure can be generalized to implement ANY finite state machine
- Counters are a very simple way to start:
  - no decisions on what state to advance to next
  - current state is the output



State Transition Diagram for
3-bit binary upcounter

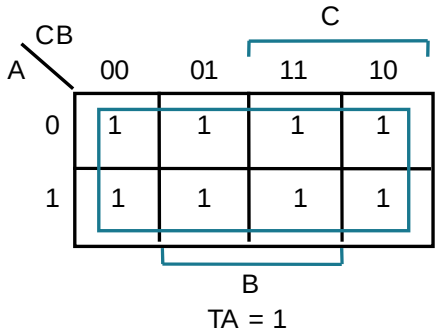
Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

State Transition Table

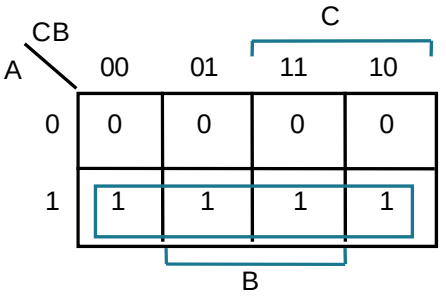
Design of 3-bit Binary Upcounter

- Let's implement with Toggle Flipflops
- What inputs must be presented to the T FFs to get them to change to the desired state bit?
- This is called "Remapping the Next State Function"

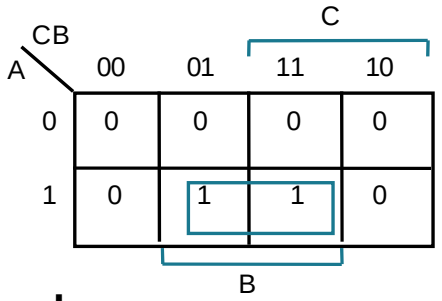
Present State			Next State			Flipflop Inputs		
C	B	A	C+	B+	A+	TC	TB	TA
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



TA = 1



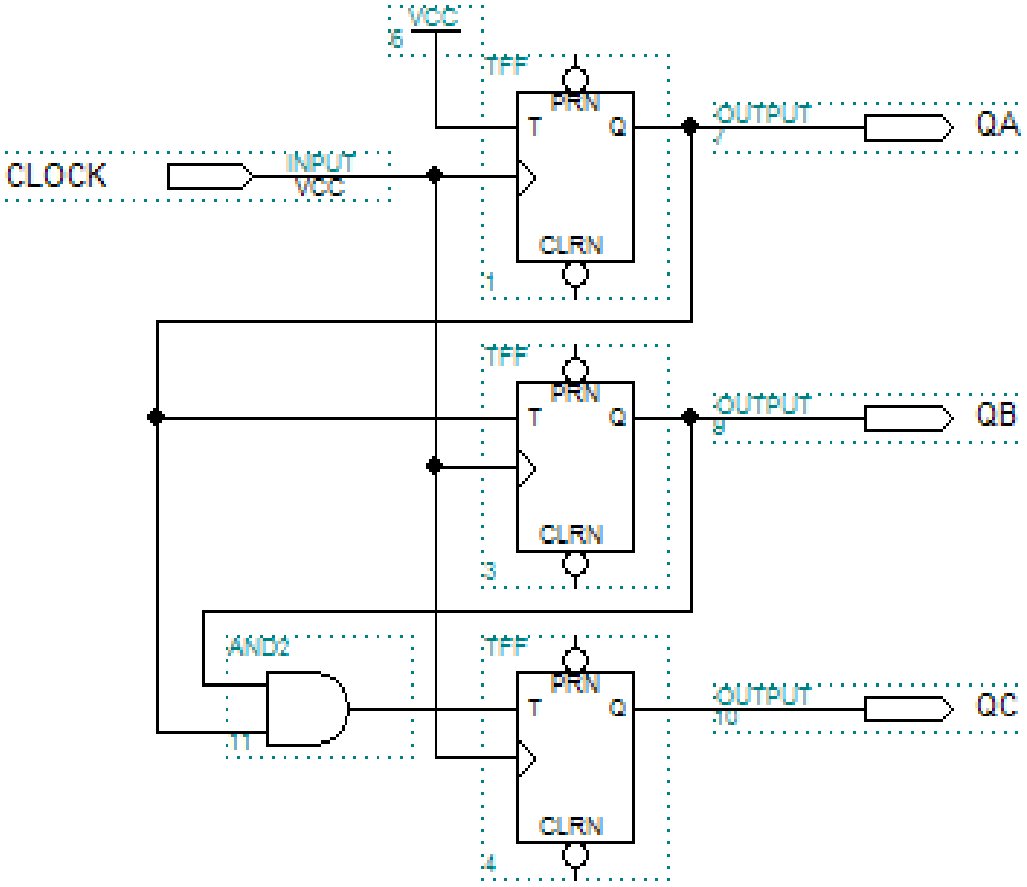
TB = A



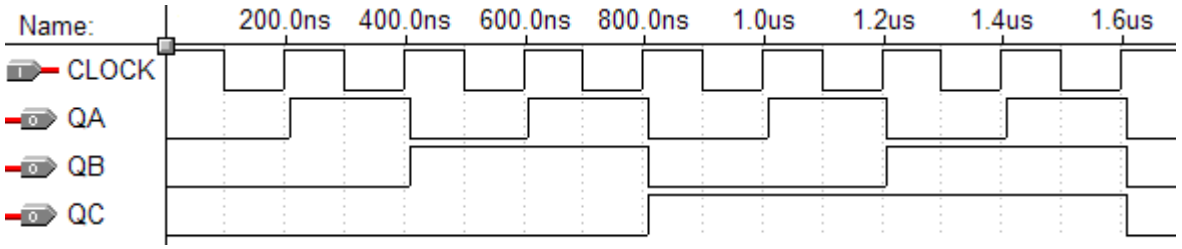
TC = A• B

K-maps for Toggle Inputs:

Design of 3-bit Binary Upcounter



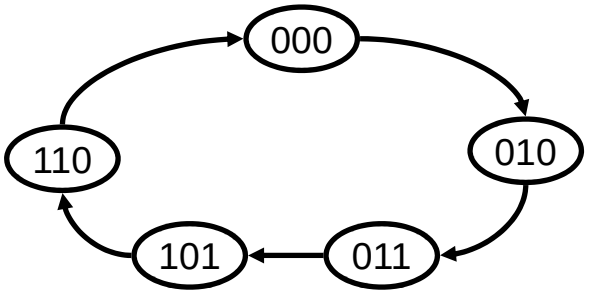
Timing Diagram:



Design of Counter with Complex Count Sequence

Step 1: Derive the State Transition Diagram

Count sequence: 000, 010, 011, 101, 110



Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	X	X	X
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	X	X	X

Step 2: State Transition Table

Note the Don't Care conditions

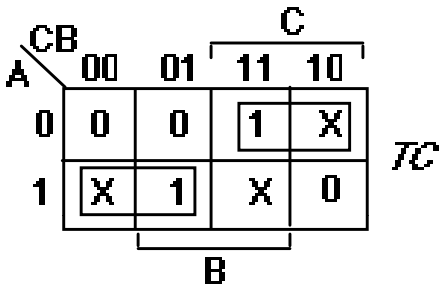
Design of Counter with Complex Count Sequence

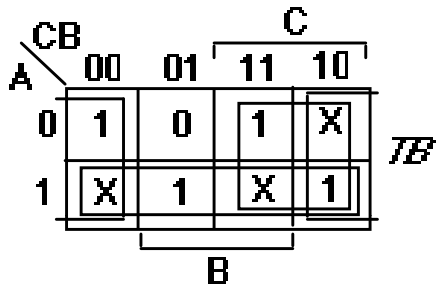
Step 3: Choose Flipflop Type for Implementation
Use **Excitation Table** to Remap Next State Functions

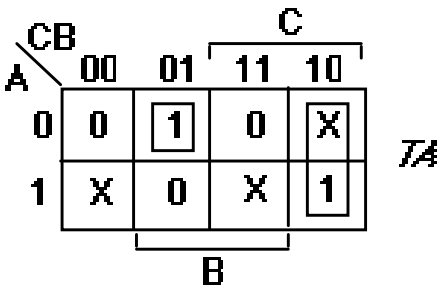
Q	Q+	T
0	0	0
0	1	1
1	0	1
1	1	0

Present State			Next State			Toggle Inputs		
C	B	A	C+	B+	A+	TC	TB	TA
0	0	0	0	1	0	0	1	0
0	0	1	X	X	X	X	X	X
0	1	0	0	1	1	0	0	1
0	1	1	1	0	1	1	1	0
1	0	0	X	X	X	X	X	X
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	X	X	X	X	X	X

Remapped Next State Functions







$TC = A'C + AC' = A \text{ xor } C$

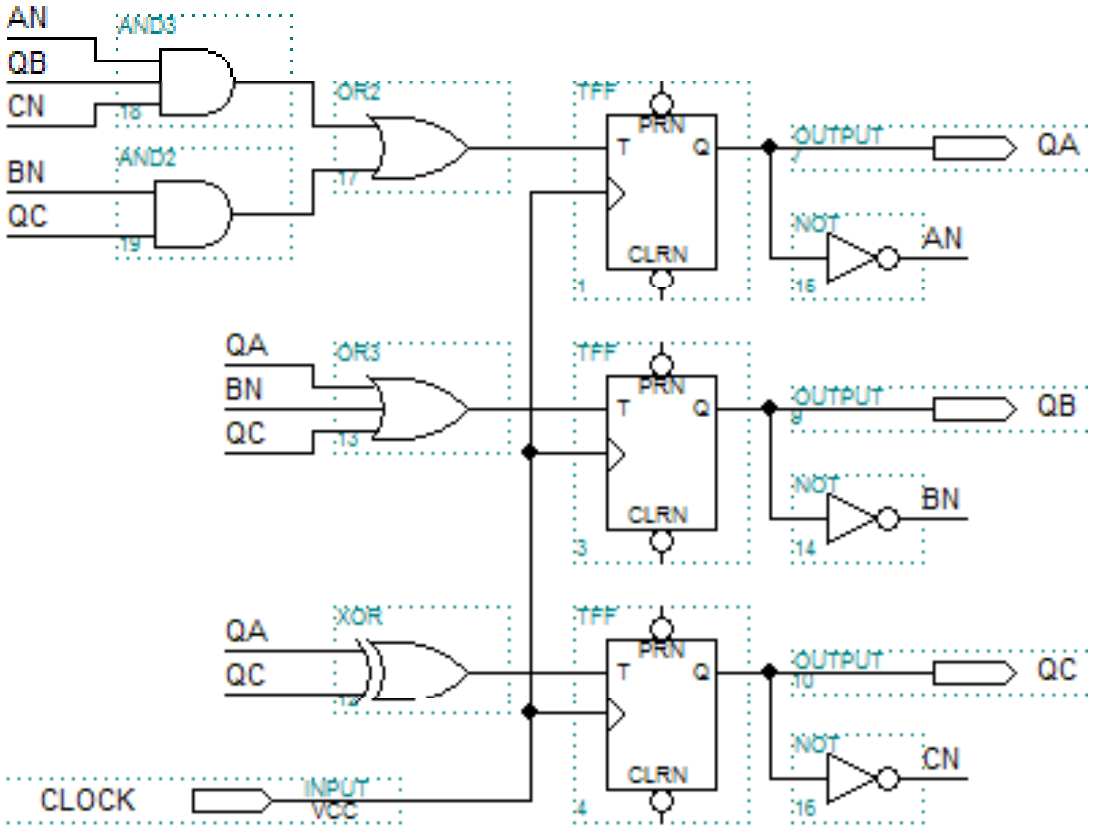
$TB = A + B' + C$

$TA = A'BC' + B'C'$

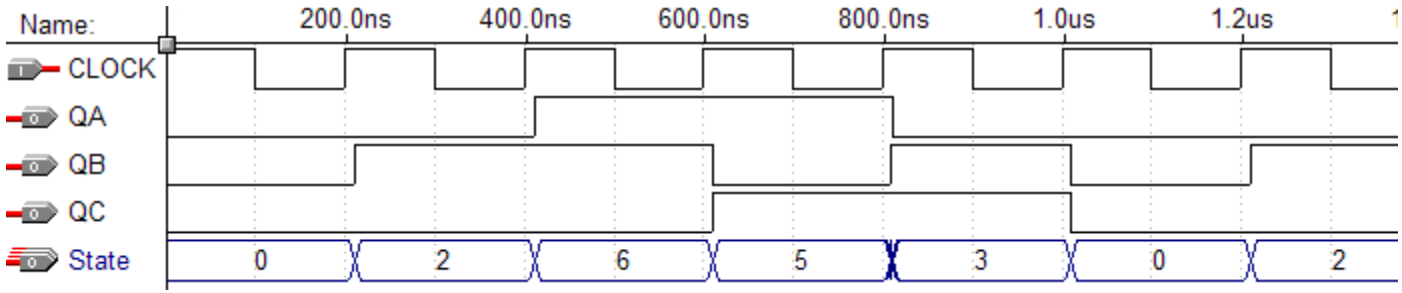
Counter Design Procedure

Resulting Logic:

5 Gates
13 Input Literals +
Flipflop connections



Timing Waveform:



Implementation with Different Kinds of FFs

SR Flipflops

Continuing with the 000, 010, 011, 101, 110, 000, ... counter example

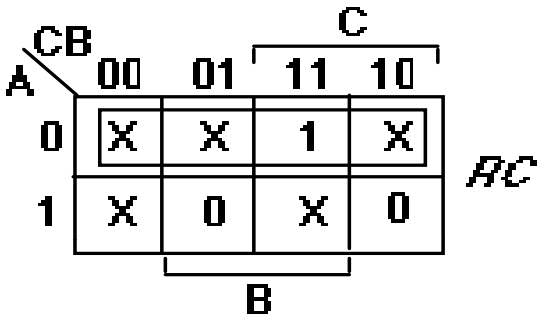
SR Excitation Table
Q+ = S + R' Q

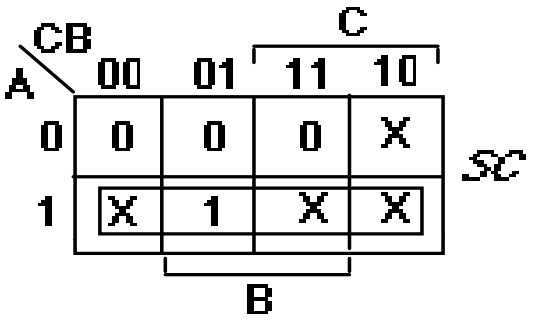
Q	Q+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

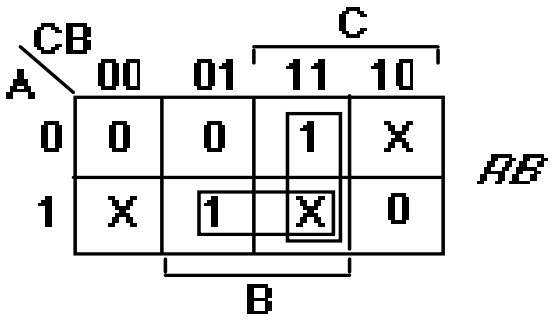
Present State			Next State			Remapped Next State					
C	B	A	C+	B+	A+	SC	RC	SB	RB	SA	RA
0	0	0	0	1	0	0	X	1	0	0	X
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	0	1	1	0	X	X	0	1	0
0	1	1	1	0	1	1	0	0	1	X	0
1	0	0	X	X	X	X	X	X	X	X	X
1	0	1	1	1	0	X	0	1	0	0	1
1	1	0	0	0	0	0	1	0	1	0	X
1	1	1	X	X	X	X	X	X	X	X	X

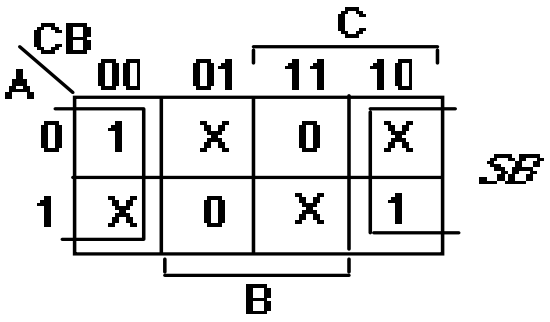
Implementation with Different Kinds of FFs

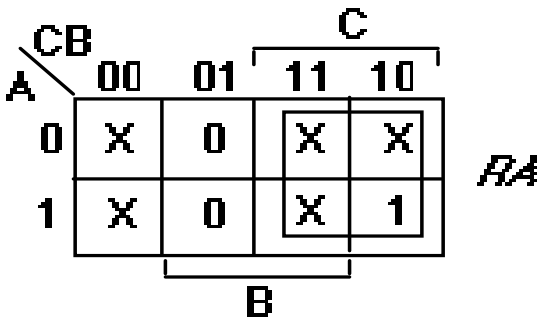
SR FFs Continued

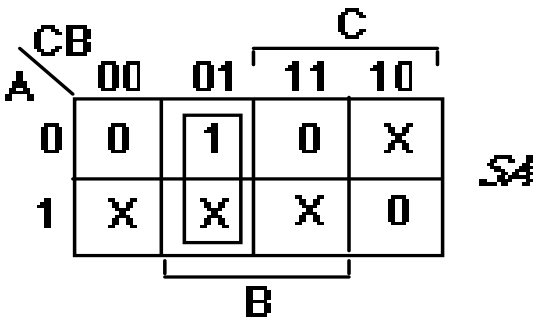












RC = A’

SC = A

RB = A B + B C = B(A+C)

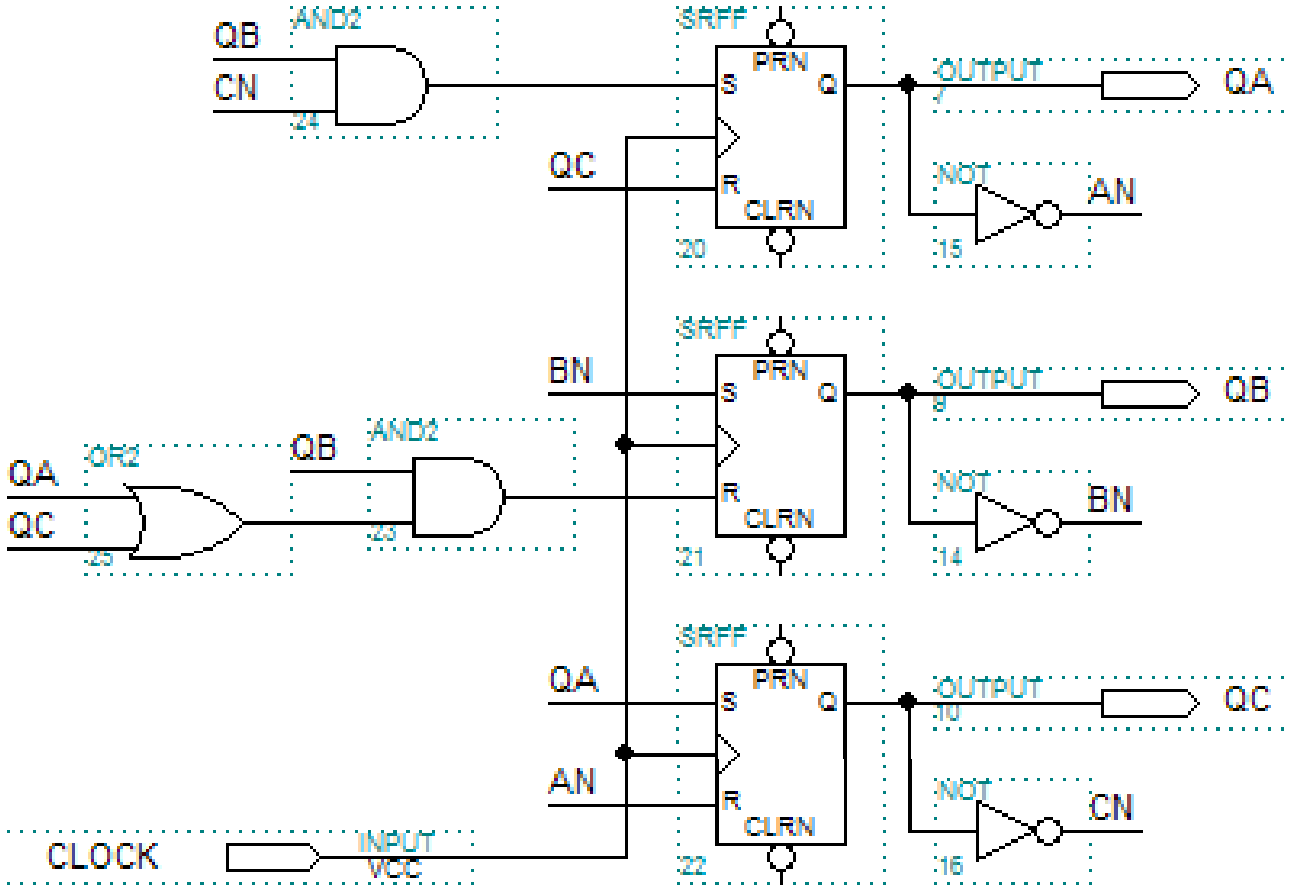
SB = B’

RA = C

SA = B C’

Implementation With Different Kinds of FFs

SR FFs Continued



Resulting Logic Level Implementation:
3 Gates, 11 Input Literals + Flipflop connections

Implementation with Different FF Types

JK FFs

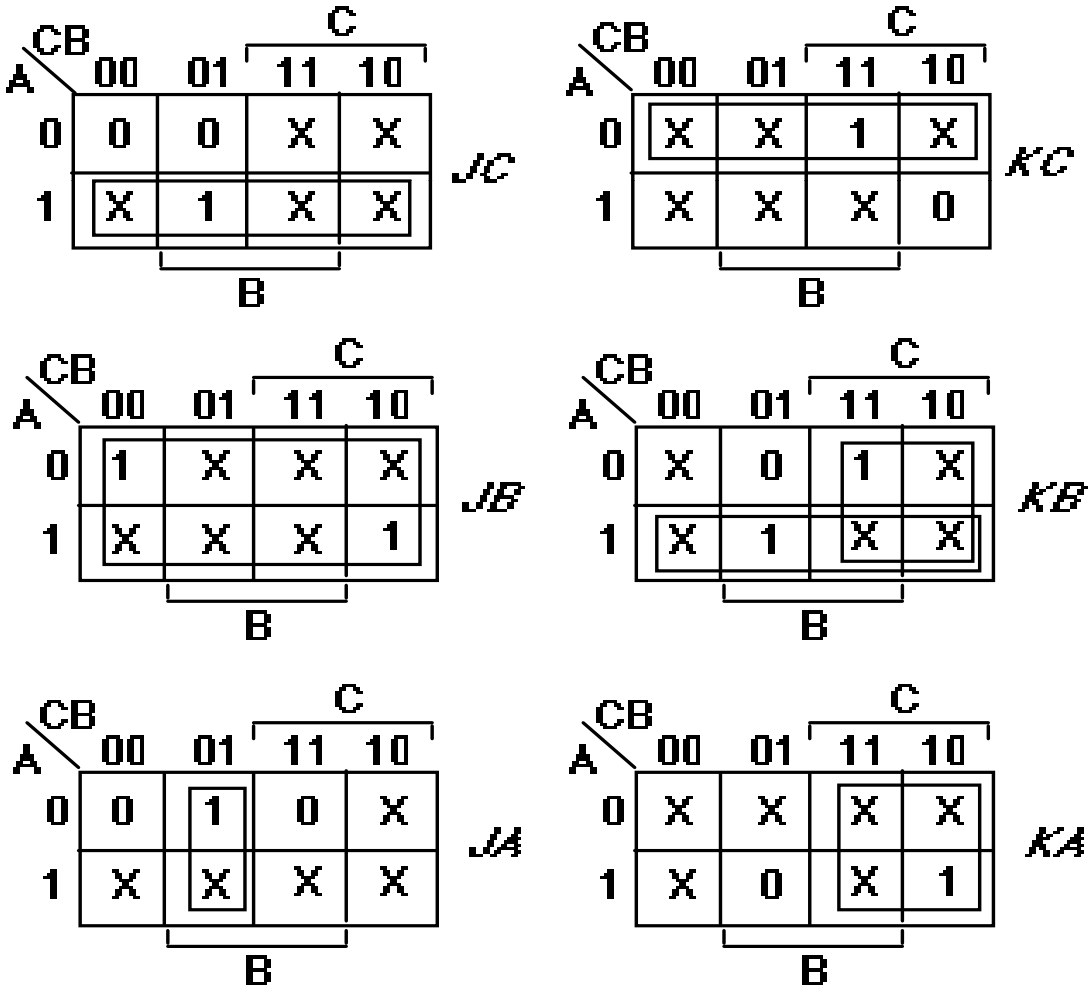
JK Excitation Table
Q+ = JQ' + K'Q

Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present State			Next State			Remapped Next State					
C	B	A	C+	B+	A+	JC	KC	JB	KB	JA	KA
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	1	1	X	X	1	X	0
1	0	0	X	X	X	X	X	X	X	X	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	X	X	X	X	X	X	X	X	X

Implementation with Different FF Types

JK FFs Continued



$JC = A$

$KC = A'$

$JB = 1$

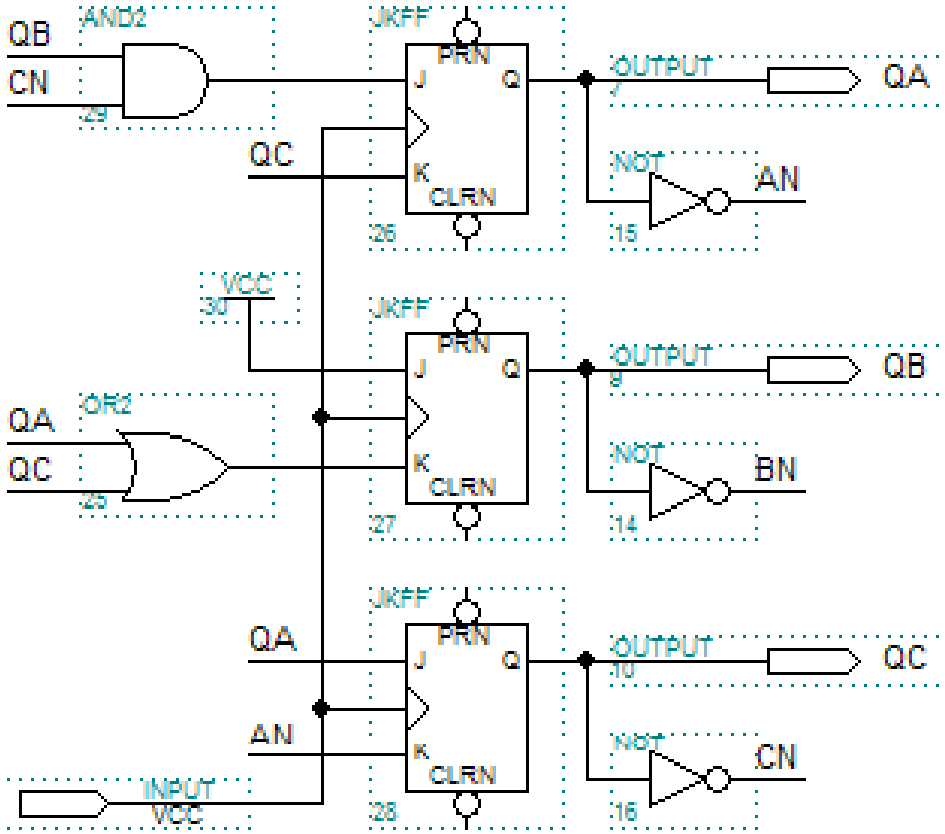
$KB = A + C$

$JA = B\,C'$

$KA = C$

Implementation with Different FF Types

JK FFs Continued

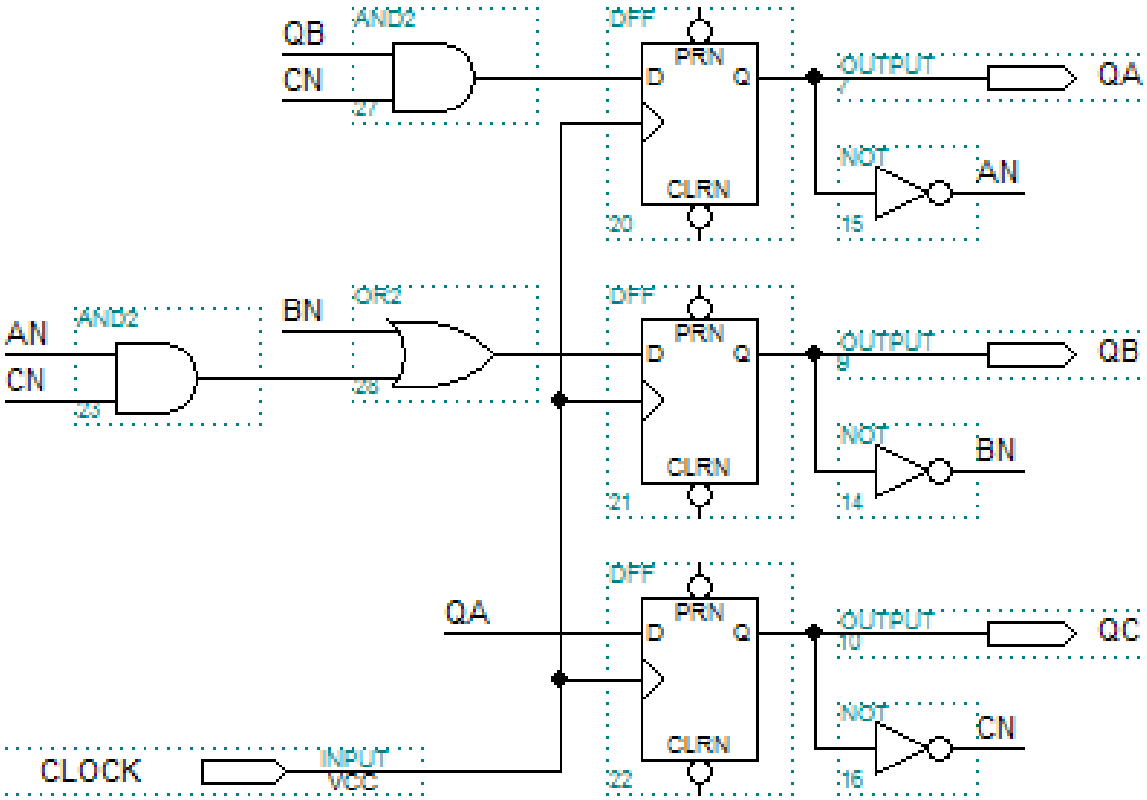


Resulting Logic Level Implementation:
2 Gates, 10 Input Literals + Flipflop Connections

Implementation with Different FF Types

D FFs:
Simplest Design Procedure:
No remapping needed!
DA = BC'
DB = A'C' + B'
DC = A

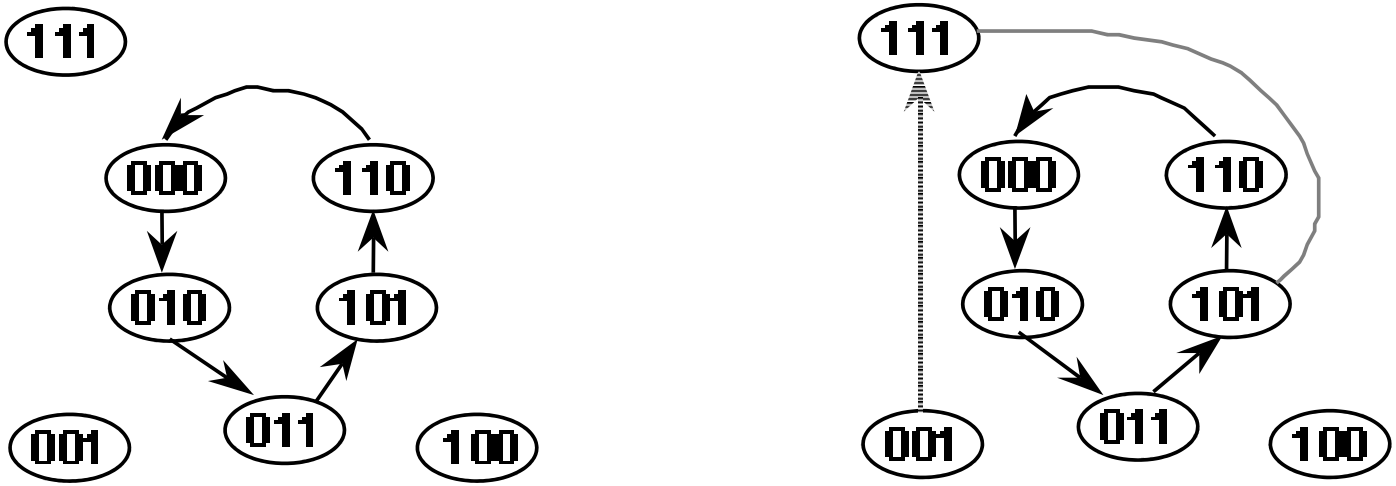
Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	X	X	X
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	X	X	X



Resulting Logic Level Implementation:
3 Gates, 8 Input Literals + Flipflop connections

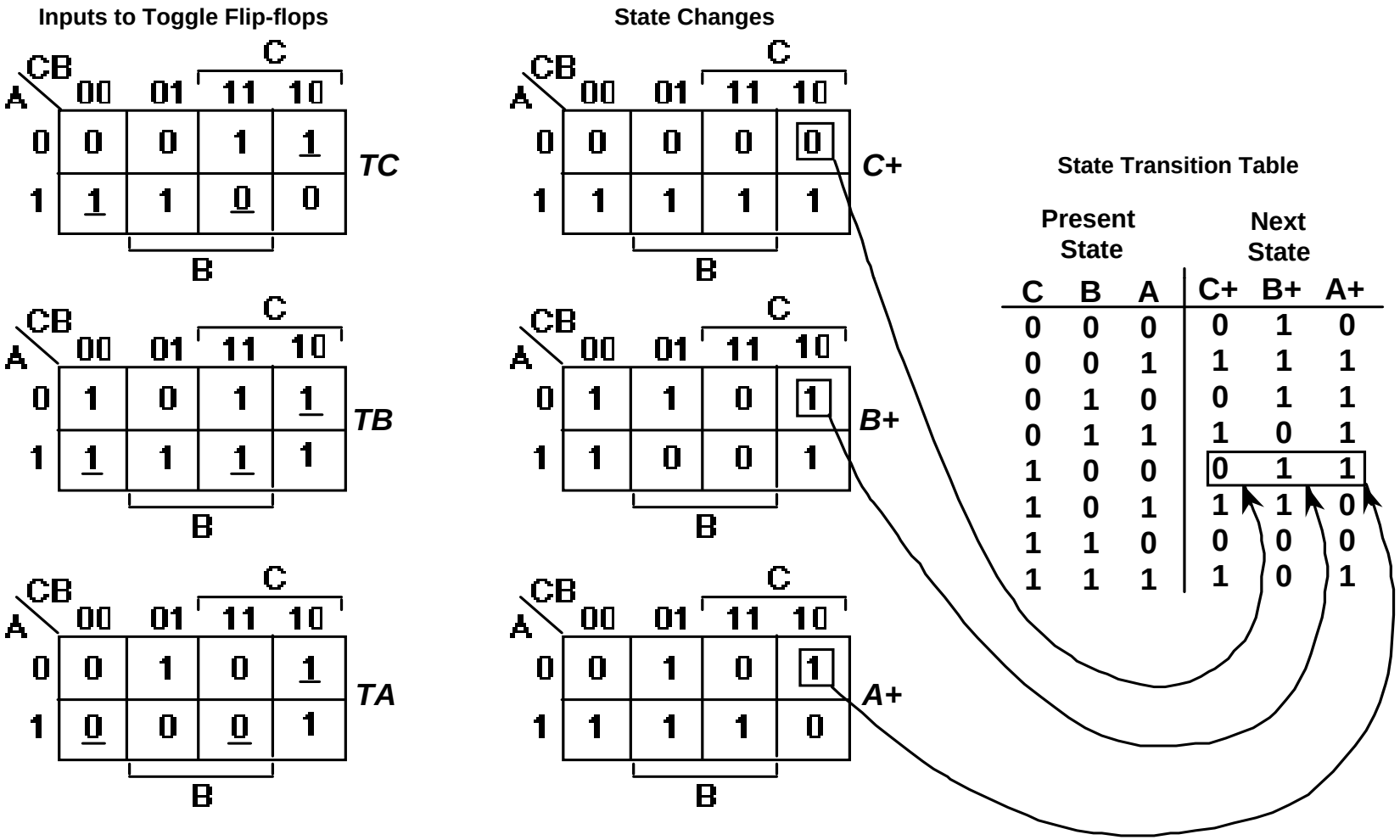
Avoiding Ambiguous States

- Problem with counter with modulo < 2^n
  - At power-up, counter may be in ANY possible state
  - Designer must guarantee that it (eventually) enters a valid state
  - Especially a problem for counters that validly use a subset of states
- Self-Starting Counters
  - Design counter so that even invalid states eventually transition to valid state



Two Self-Starting State Transition Diagrams for the Example Counter

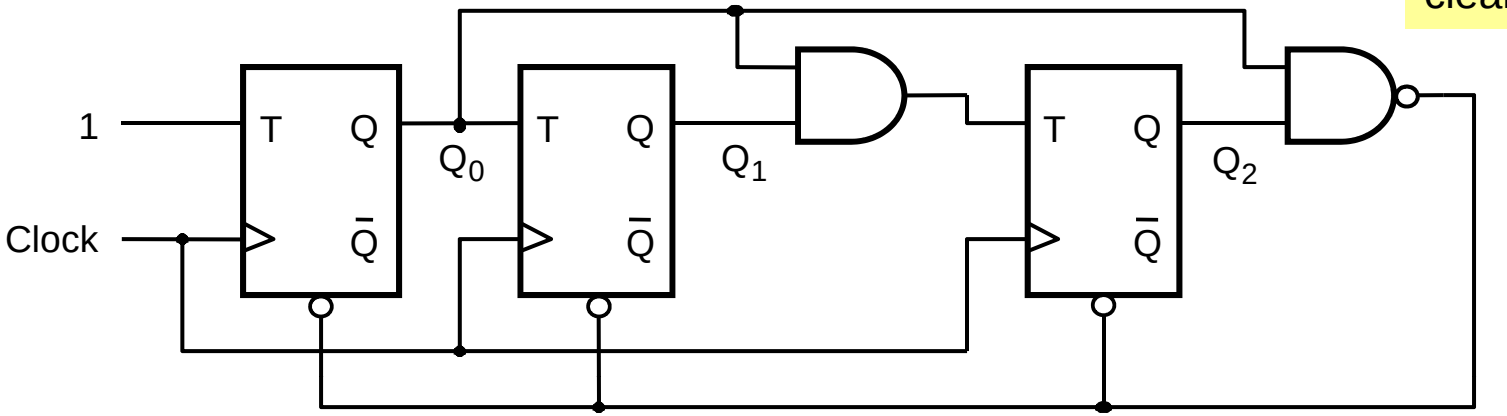
Self-Starting Counters



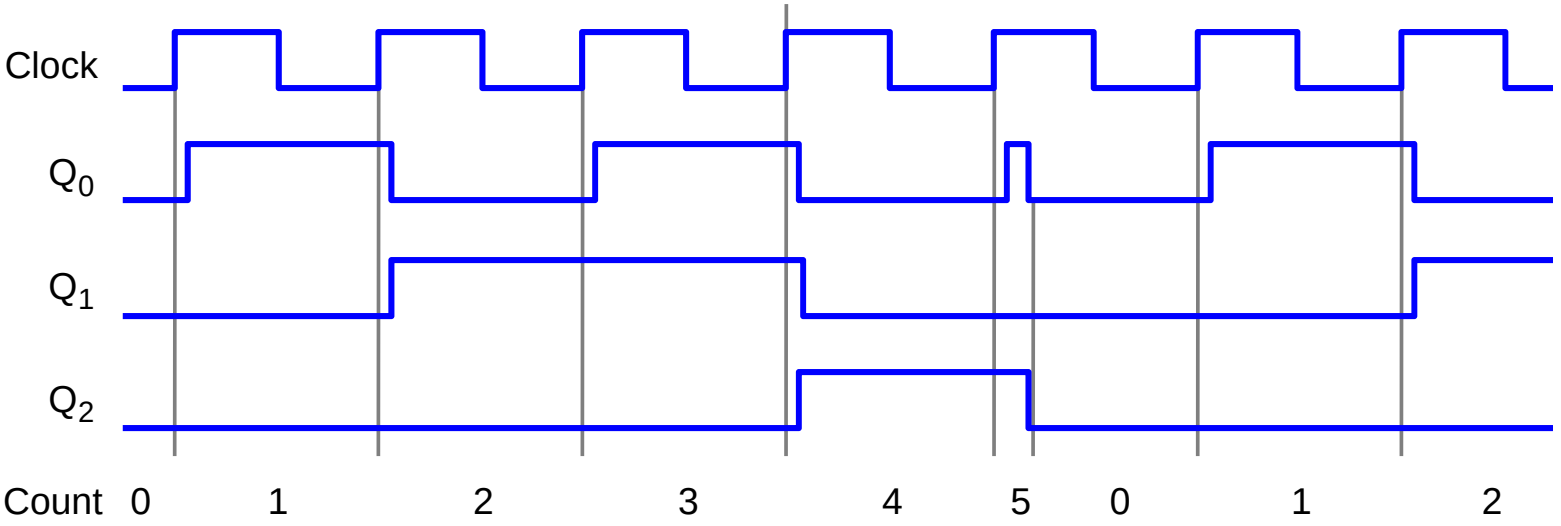
Deriving State Transition Table from Don't Care Assignment

Avoiding Ambiguous States

Never use counters with asynchronous clear



(a) A modulo-6 counter with asynchronous clear



(b) Timing diagram

Counter Implementation with Different FF Types

- T FFs well suited for straightforward binary counters
  - But yielded worst gate and literal count for this example (coz it's not straightforward !)
- No reason to choose SR over JK FFs: it is a proper subset of JK
  - SR FFs don't really exist anyway
- JK FFs yielded lowest gate count
  - Tend to yield best choice for packaged logic where gate count is key
- D FFs yield simplest design procedure
  - Best literal count
  - D storage devices very transistor efficient in VLSI
    - Other flipflops most likely implemented using DFF in VLSI/FPGA
  - Best choice where area/literal count is the key