SPHJC63 – Digital Electronics

Er R Sithikraja M.Sc., M.E., UGC-NET

Department of Physics

CPA College Bodinayakanur

Lecture 13: Sequential Logic Counters and Registers

Counters

- Introduction: Counters
- Asynchronous (Ripple) Counters
- Asynchronous Counters with MOD number < 2^n
- Asynchronous Down Counters
- Cascading Asynchronous Counters

Lecture 13: Sequential Logic Counters and Registers

- Synchronous (Parallel) Counters
- Up/Down Synchronous Counters
- Designing Synchronous Counters
- Decoding A Counter
- Counters with Parallel Load

Lecture 13: Sequential Logic Counters and Registers

Registers

- Introduction: Registers
  - Simple Registers
  - Registers with Parallel Load
- Using Registers to implement Sequential Circuits
- Shift Registers
  - Serial In/Serial Out Shift Registers
  - Serial In/Parallel Out Shift Registers
  - Parallel In/Serial Out Shift Registers
  - Parallel In/Parallel Out Shift Registers

Lecture 13: Sequential Logic Counters and Registers

- Bidirectional Shift Registers
- An Application – Serial Addition
- Shift Register Counters
  - Ring Counters
  - Johnson Counters
- Random-Access Memory (RAM)

Introduction: Counters

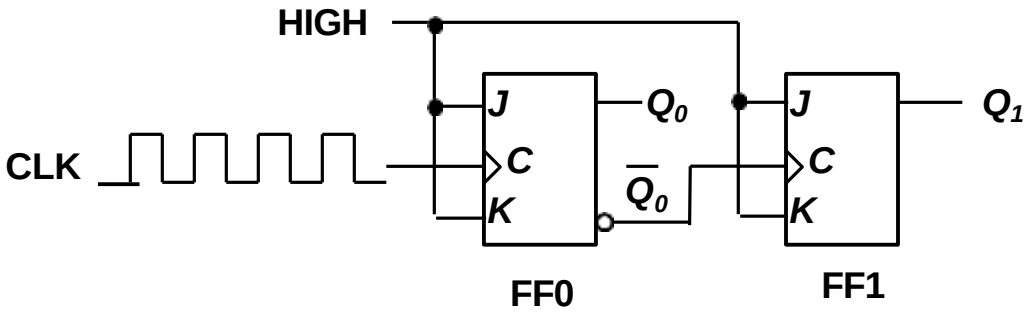
- Counters are circuits that cycle through a specified number of states.
- Two types of counters:
  - synchronous (parallel) counters
  - asynchronous (ripple) counters
- Ripple counters allow some flip-flop outputs to be used as a source of clock for other flip-flops.
- Synchronous counters apply the same clock to all flip-flops.

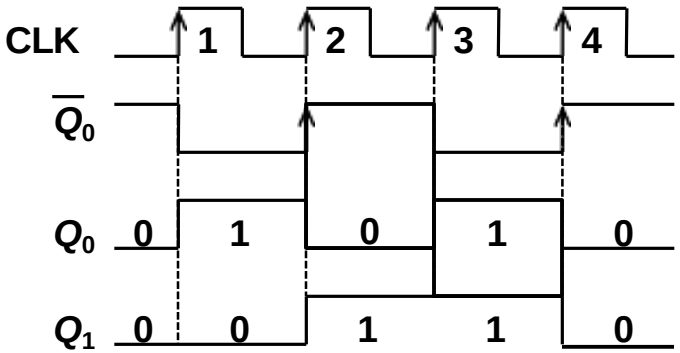
Asynchronous (Ripple) Counters

- Asynchronous counters: the flip-flops do not change states at exactly the same time as they do not have a common clock pulse.
- Also known as ripple counters, as the input clock pulse “ripples” through the counter – cumulative delay is a drawback.
- *n* flip-flops $\rightarrow$ a MOD (modulus) 2^n counter. (Note: A MOD-*x* counter cycles through *x* states.)
- Output of the last flip-flop (MSB) divides the input clock frequency by the MOD number of the counter, hence a counter is also a *frequency divider*.

Asynchronous (Ripple) Counters

- Example: 2-bit ripple binary counter.
- Output of one flip-flop is connected to the clock input of the next more-significant flip-flop.



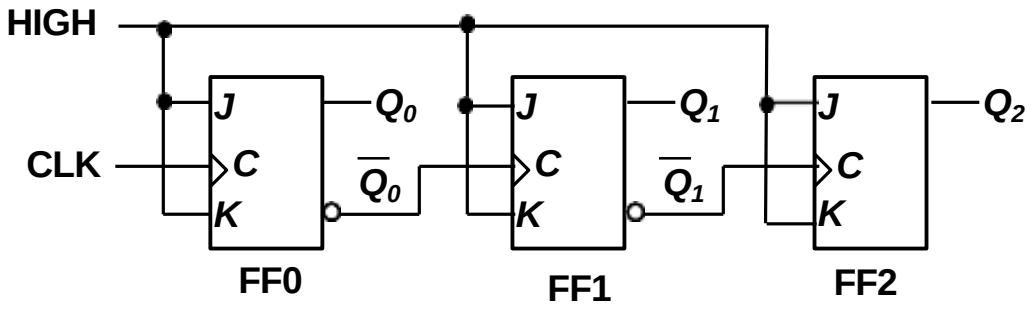


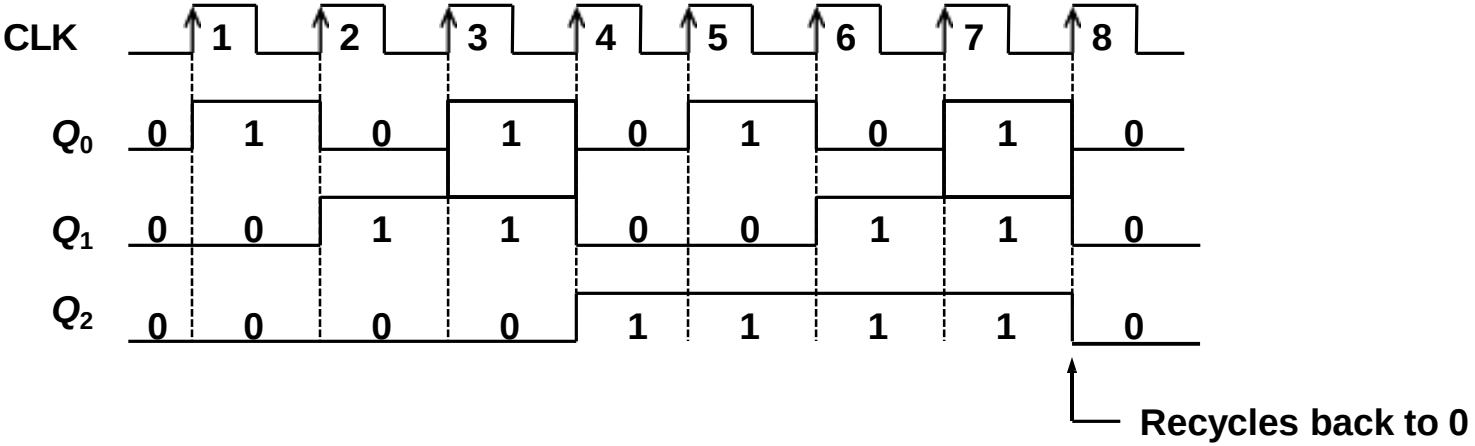
Timing diagram

$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ ...

Asynchronous (Ripple) Counters

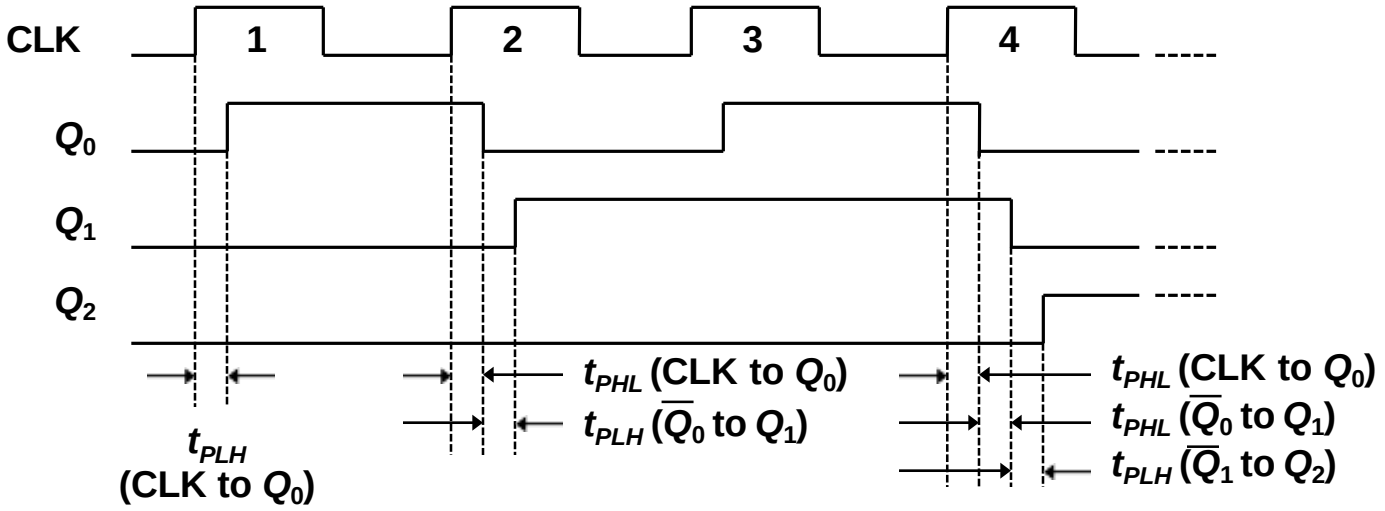
- Example: 3-bit ripple binary counter.





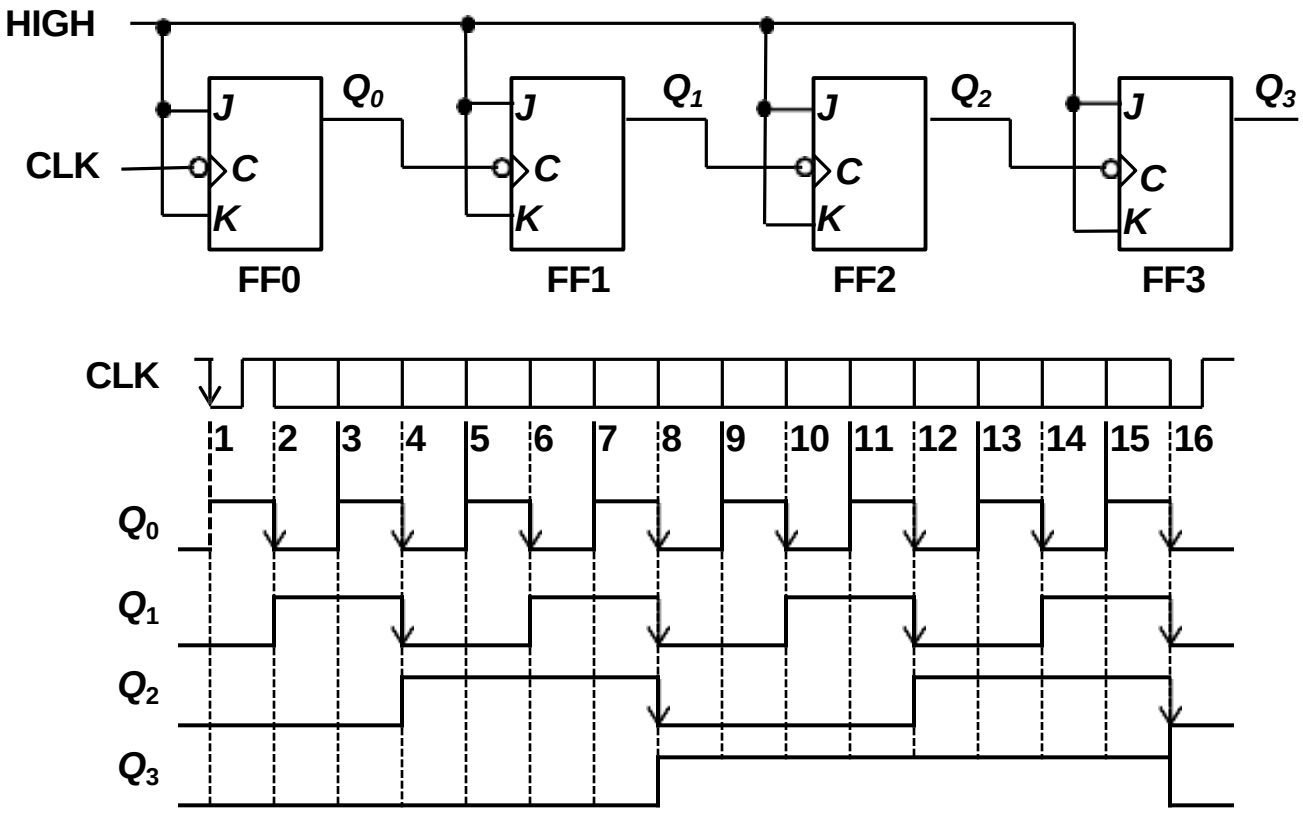
Asynchronous (Ripple) Counters

- Propagation delays in an asynchronous (ripple-clocked) binary counter.
- If the accumulated delay is greater than the clock pulse, some counter states may be misrepresented!



Asynchronous (Ripple) Counters

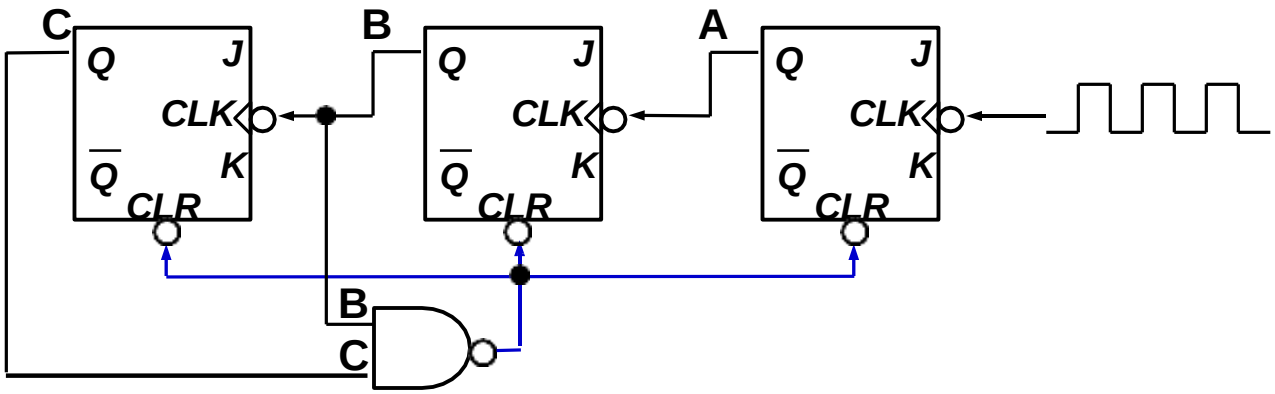
- Example: 4-bit ripple binary counter (negative-edge triggered).



Asyn. Counters with MOD no. < 2^n

- States may be skipped resulting in a truncated sequence.
- Technique: force counter to *recycle before going through all of the states* in the binary sequence.
- Example: Given the following circuit, determine the counting sequence (and hence the modulus no.)

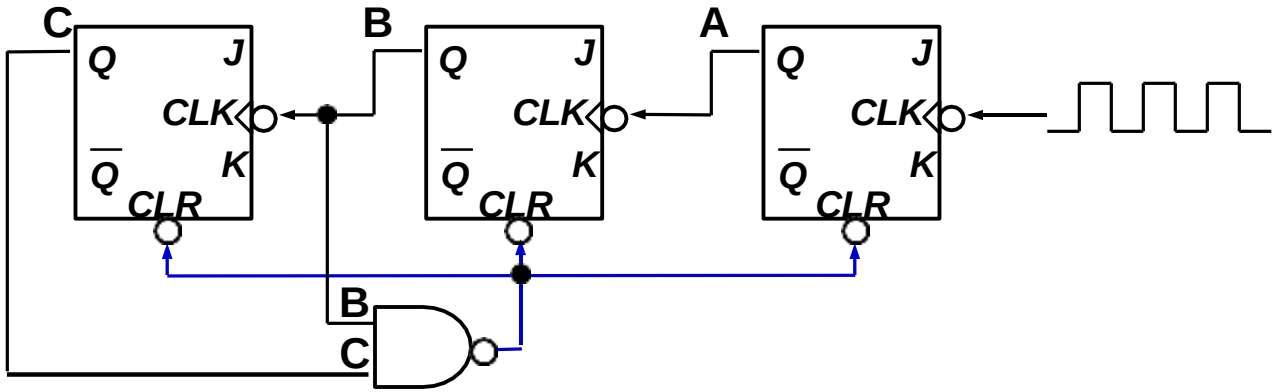
All *J*, *K* inputs are 1 (HIGH).

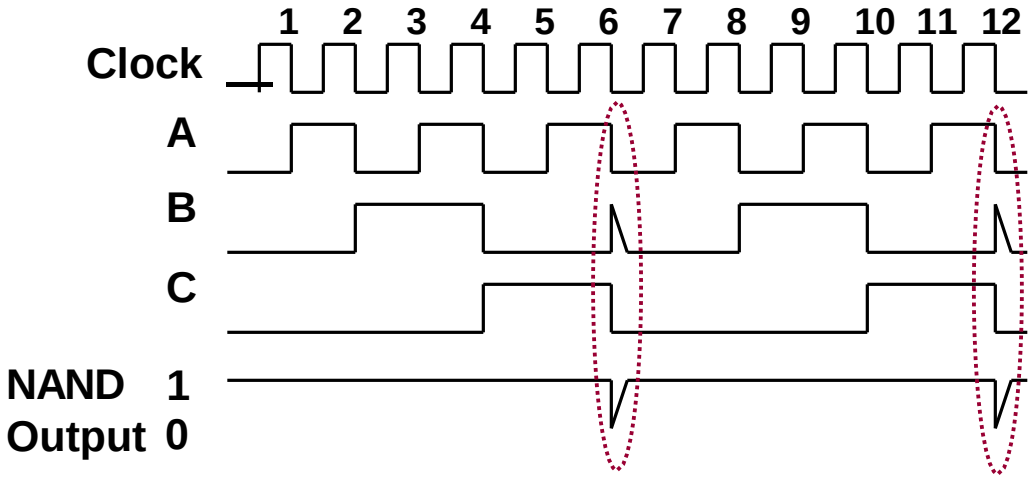


Asyn. Counters with MOD no. < 2^n

- Example (cont'd):

All *J*, *K* inputs are 1 (HIGH).

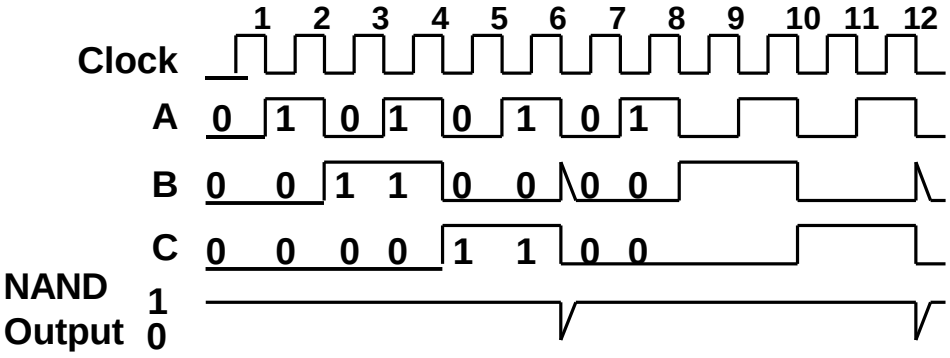


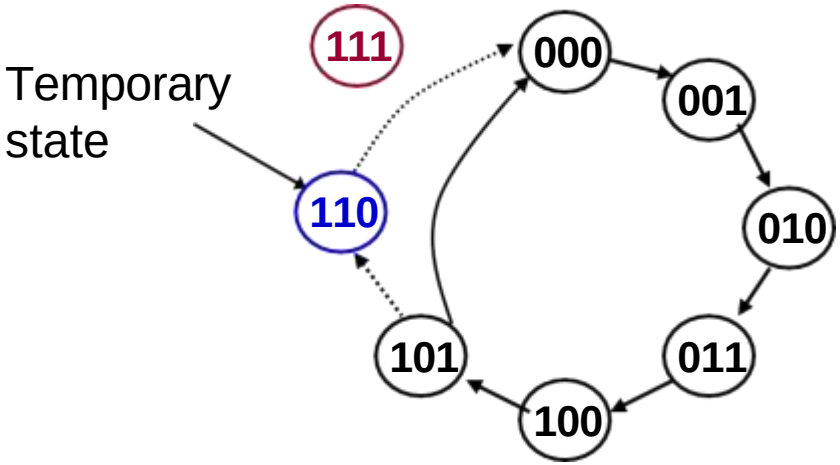


MOD-6 counter produced by clearing (a MOD-8 binary counter) when count of six (110) occurs.

Asyn. Counters with MOD no. < 2^n

- Example (cont'd): Counting sequence of circuit (in CBA order).

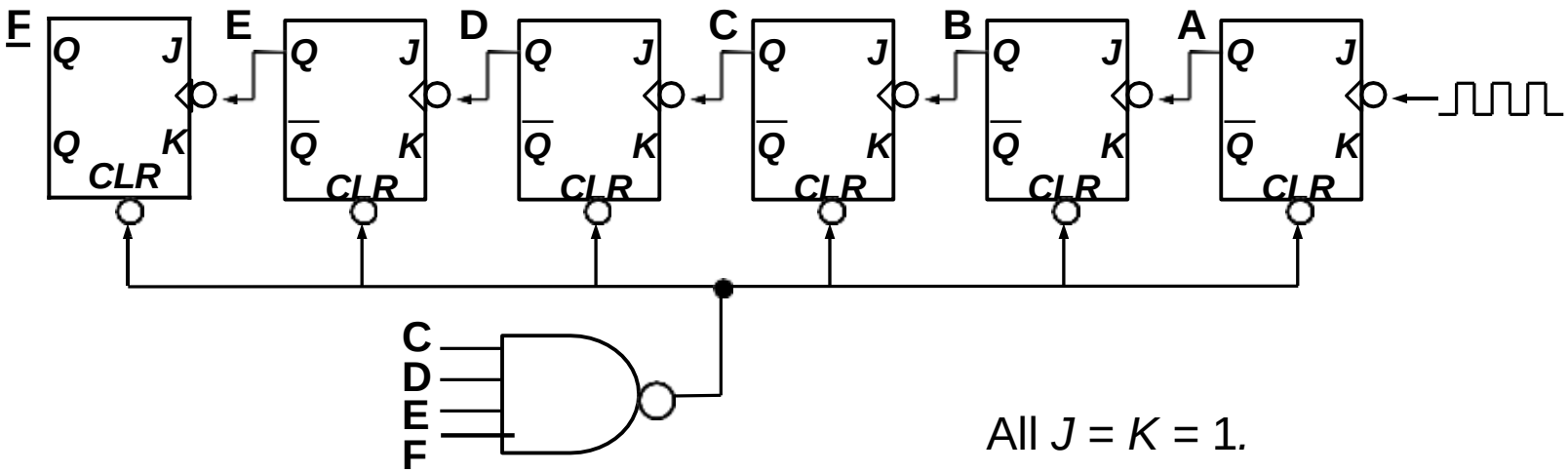




Counter is a MOD-6 counter.

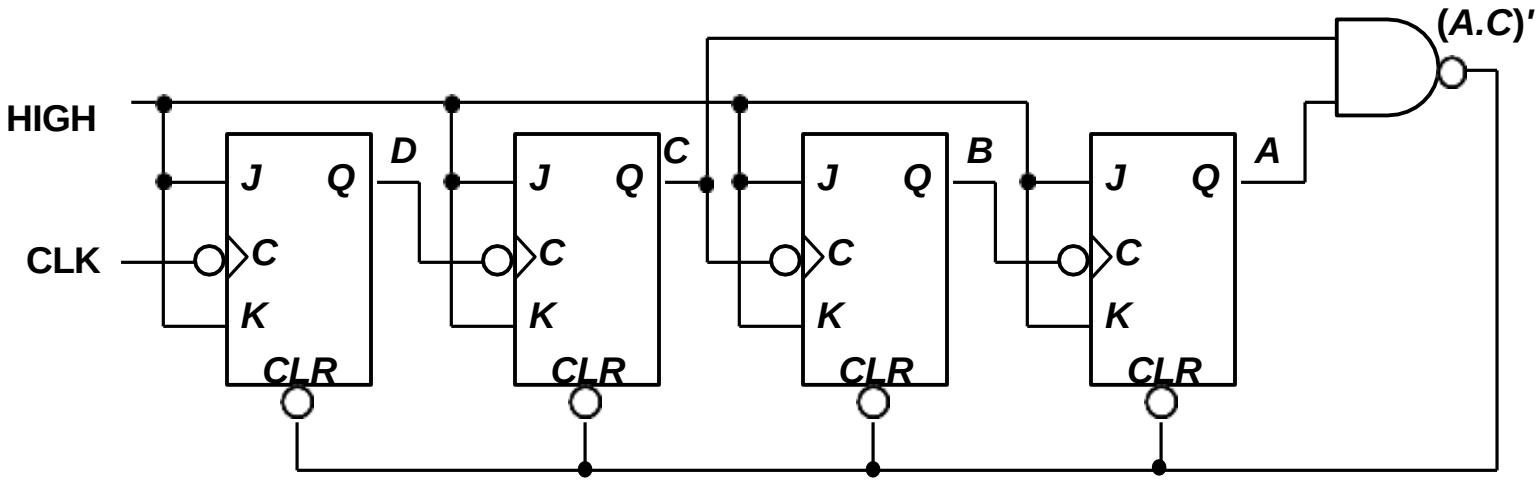
Asyn. Counters with MOD no. < 2^n

- *Exercise:* How to construct an asynchronous MOD-5 counter? MOD-7 counter? MOD-12 counter?
- *Question:* The following is a MOD-? counter?



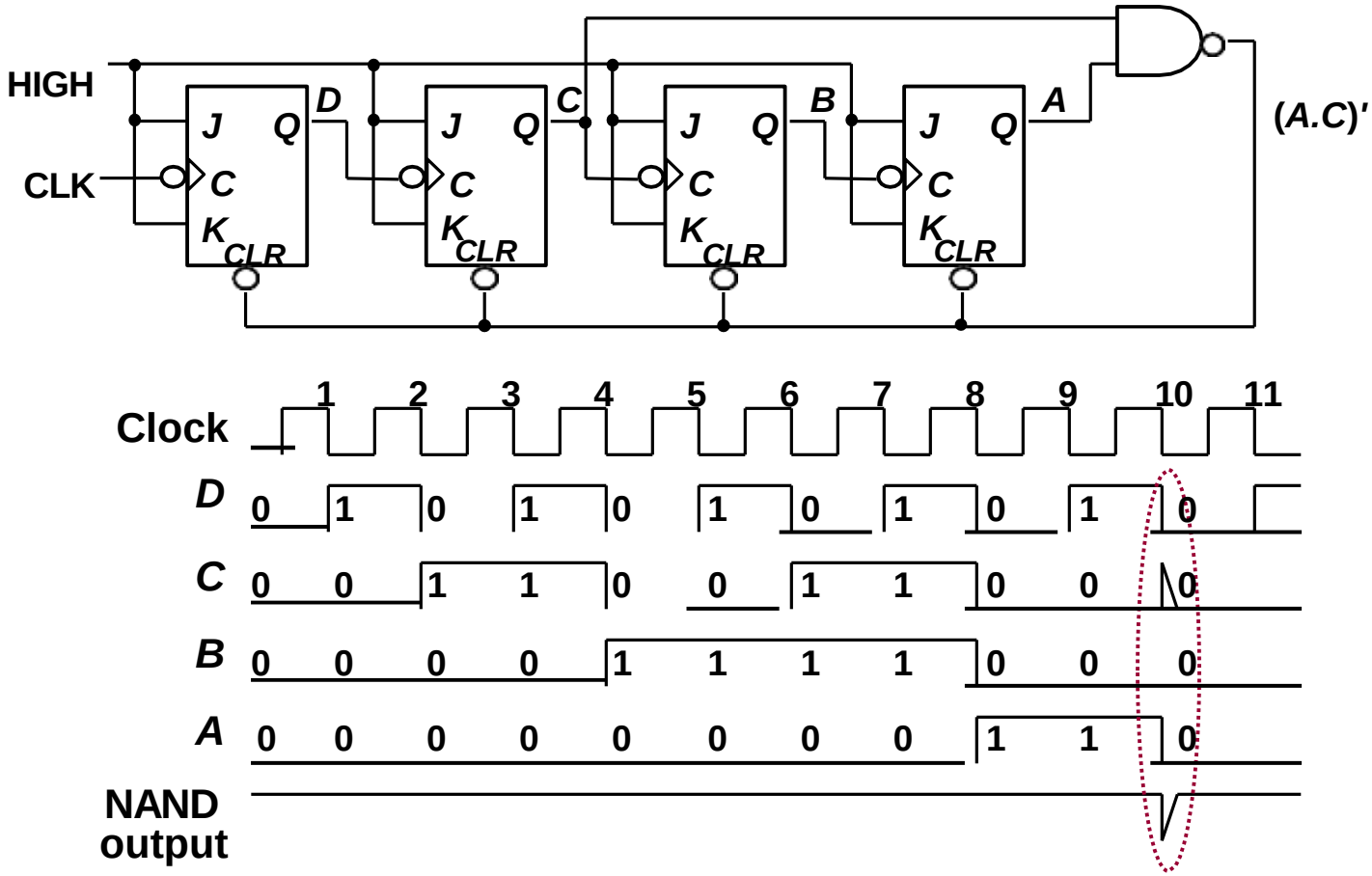
Asyn. Counters with MOD no. < 2^n

- Decade counters (or BCD counters) are counters with 10 states (modulus-10) in their sequence. They are commonly used in daily life (e.g.: utility meters, odometers, etc.).
- Design an asynchronous decade counter.



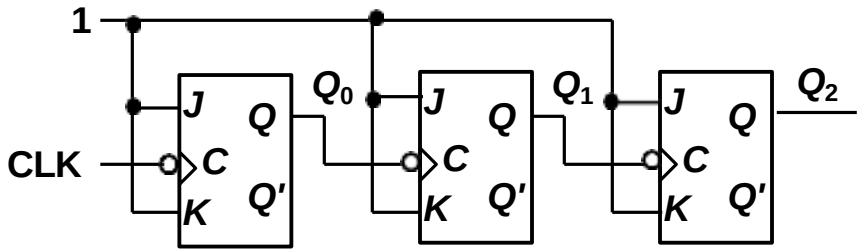
Asyn. Counters with MOD no. < 2^n

- Asynchronous decade/BCD counter (cont'd).

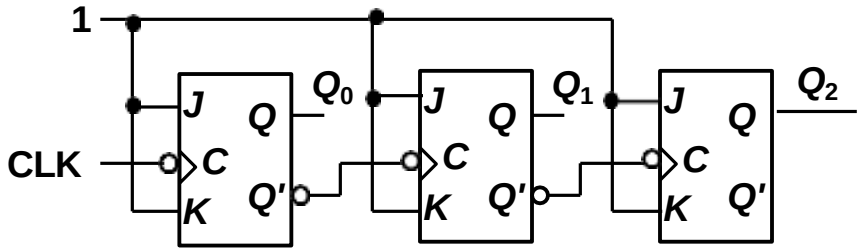


Asynchronous Down Counters

- So far we are dealing with *up counters*. *Down counters*, on the other hand, count downward from a maximum value to zero, and repeat.
- Example: A 3-bit binary (MOD-2^3) down counter.



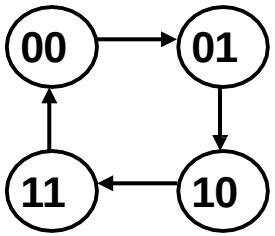
3-bit binary up counter



3-bit binary down counter

Synchronous (Parallel) Counters

- Synchronous (parallel) counters: the flip-flops are clocked at the same time by a common clock pulse.
- We can design these counters using the sequential logic design process (covered in Lecture #12).
- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).



Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

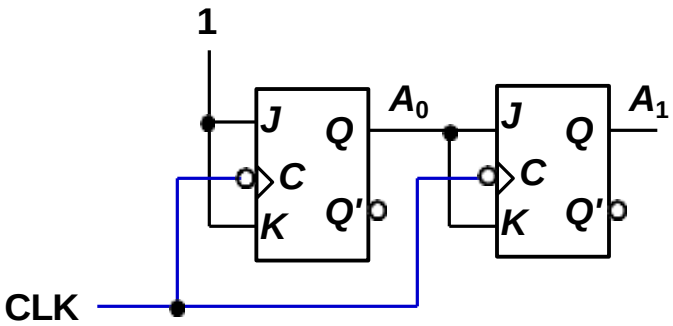
Synchronous (Parallel) Counters

- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).

Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

$TA_1 = A_0$

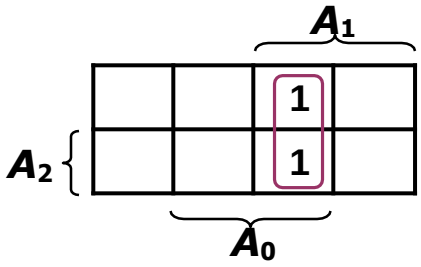
$TA_0 = 1$



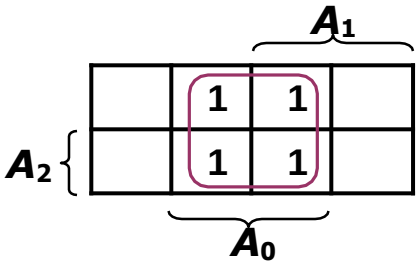
Synchronous (Parallel) Counters

- Example: 3-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J, K inputs).

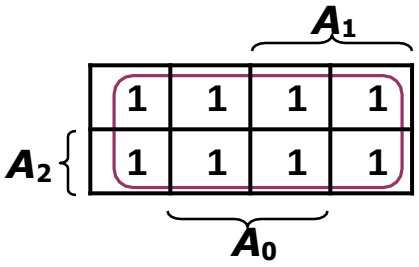
Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	0	0	0	0	1	1	1



$TA_2 = A_1.A_0$



$TA_1 = A_0$

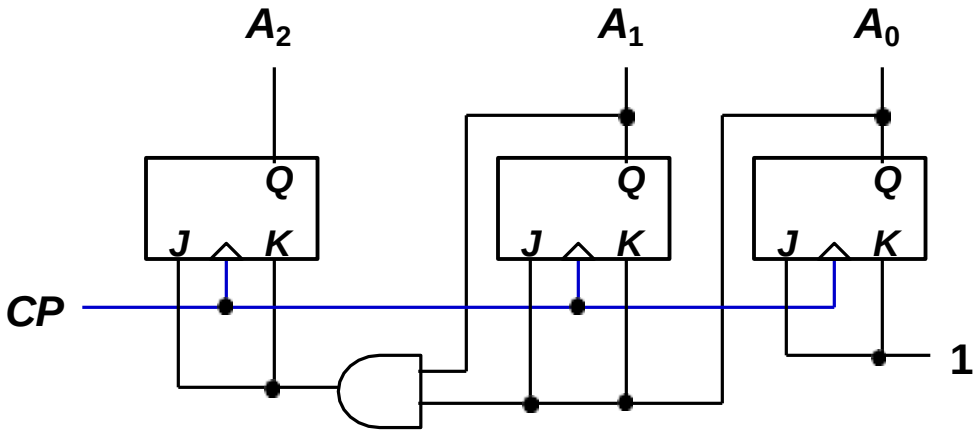


$TA_0 = 1$

Synchronous (Parallel) Counters

- Example: 3-bit synchronous binary counter (cont'd).

$TA_2 = A_1.A_0$ $TA_1 = A_0$ $TA_0 = 1$



Synchronous (Parallel) Counters

- Note that in a binary counter, the n^{th} bit (shown underlined) is always complemented whenever

  $\underline{0}11\ldots11 \rightarrow \underline{1}00\ldots00$

  or $\underline{1}11\ldots11 \rightarrow \underline{0}00\ldots00$

- Hence, X_n is complemented whenever

  $X_{n-1}X_{n-2} \ldots X_1X_0 = 11\ldots11.$

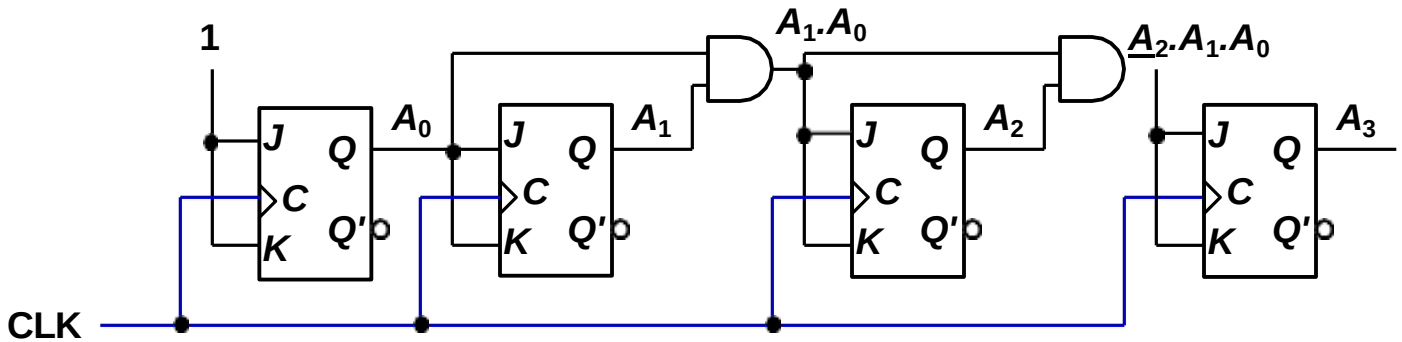
- As a result, if T flip-flops are used, then

  $TX_n = X_{n-1} \cdot X_{n-2} \cdot \ldots \cdot X_1 \cdot X_0$

Synchronous (Parallel) Counters

- Example: 4-bit synchronous binary counter.

$$TA_3 = A_2 \cdot A_1 \cdot A_0$$
$$TA_2 = A_1 \cdot A_0$$
$$TA_1 = A_0$$
$$TA_0 = 1$$



Synchronous (Parallel) Counters

- Example: Synchronous decade/BCD counter.

Clock pulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycle)	0	0	0	0

$T_0 = 1$

$T_1 = Q_3'.Q_0$

$T_2 = Q_1.Q_0$

$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$

Synchronous (Parallel) Counters

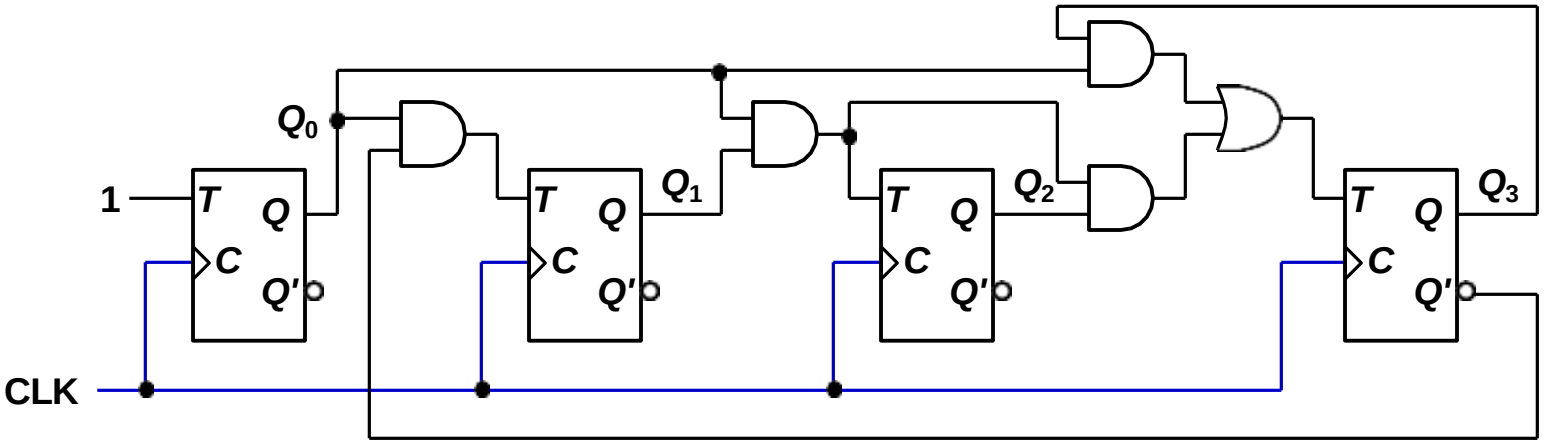
- Example: Synchronous decade/BCD counter (cont'd).

$T_0 = 1$

$T_1 = Q_3'.Q_0$

$T_2 = Q_1.Q_0$

$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$

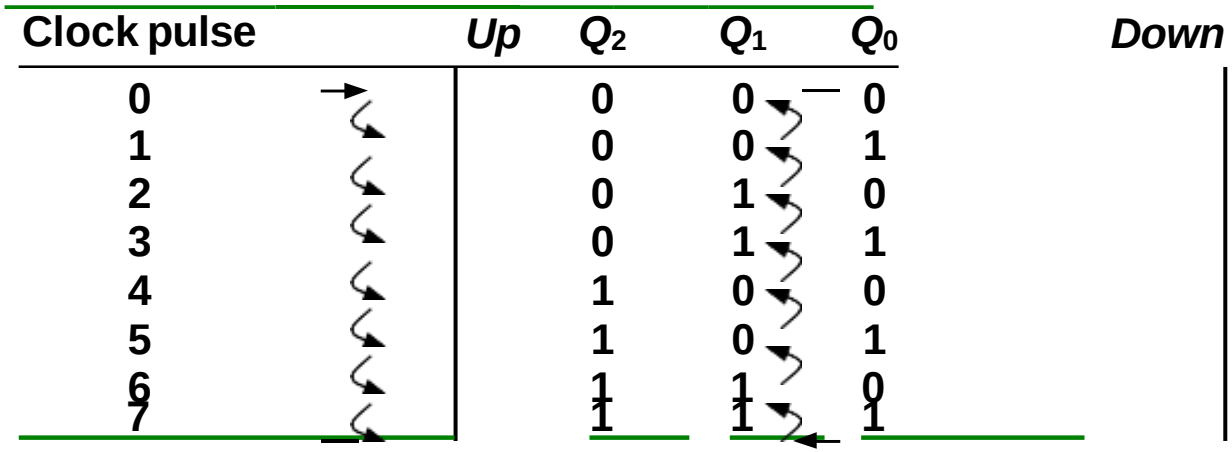


Up/Down Synchronous Counters

- Up/down synchronous counter: a *bidirectional* counter that is capable of counting either up or down.
- An input (control) line *Up*/$\overline{Down}$ (or simply *Up*) specifies the direction of counting.
  - *Up*/$\overline{Down}$ = 1 $\rightarrow$ Count upward
  - *Up*/$\overline{Down}$ = 0 $\rightarrow$ Count downward

Up/Down Synchronous Counters

- Example: A 3-bit up/down synchronous binary counter.



$TQ_0 = 1$

$TQ_1 = (Q_0.Up) + (Q_0'.Up')$

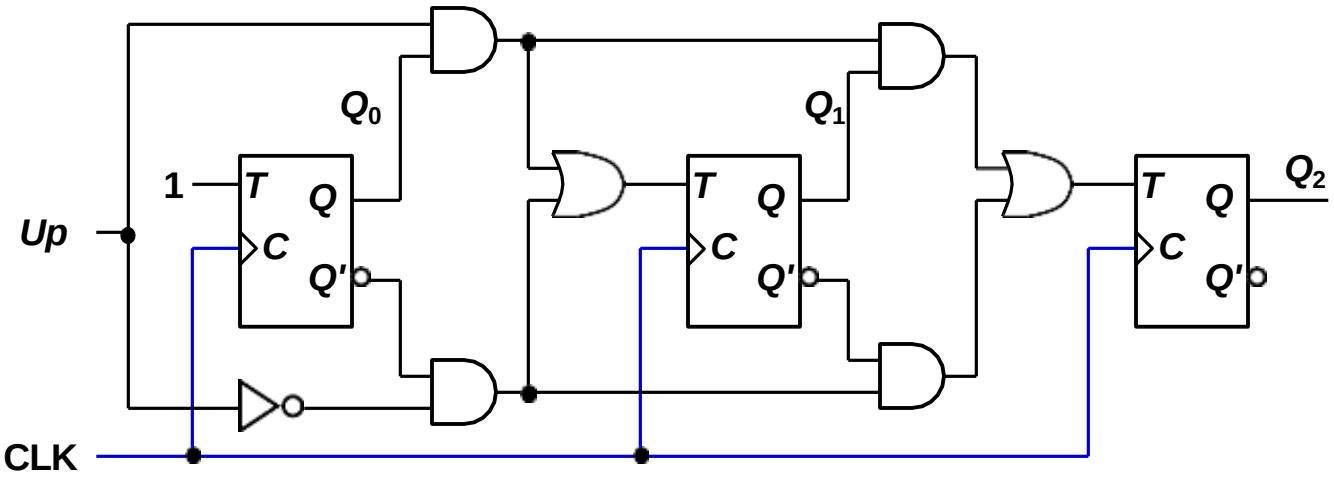
$TQ_2 = (Q_0.Q_1.Up) + (Q_0'.Q_1'.Up')$

Up counter	Down counter
$TQ_0 = 1$	$TQ_0 = 1$
$TQ_1 = Q_0$	$TQ_1 = Q_0'$
$TQ_2 = Q_0.Q_1$	$TQ_2 = Q_0'.Q_1'$

Up/Down Synchronous Counters

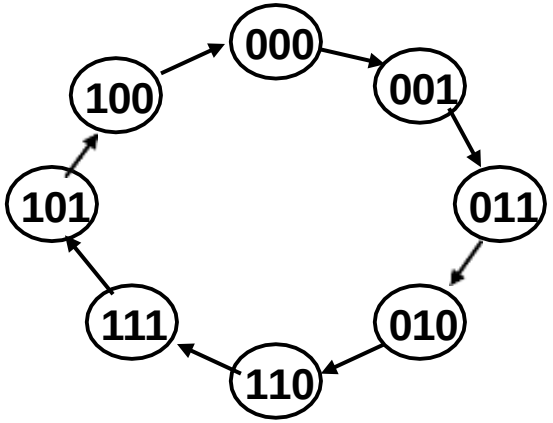
- Example: A 3-bit up/down synchronous binary counter (cont'd).

$$TQ_0 = 1$$
$$TQ_1 = (Q_0.Up) + (Q_0'.Up')$$
$$TQ_2 = (Q_0.Q_1.Up) + (Q_0'.Q_1'.Up')$$



Designing Synchronous Counters

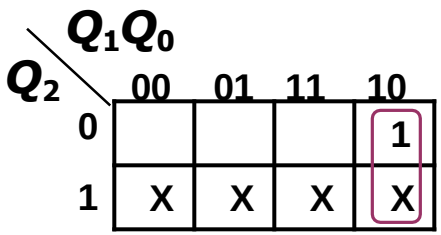
- Covered in Lecture #12.
- Example: A 3-bit Gray code counter (using JK flip-flops).



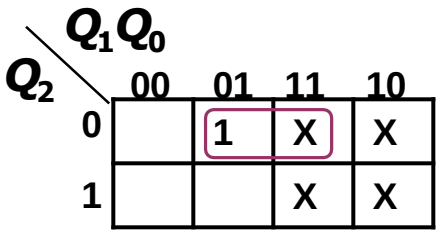
Present state			Next state			Flip-flop inputs					
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	JQ_2	KQ_2	JQ_1	KQ_1	JQ_0	KQ_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	1	1	0	1	X	X	0	0	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0

Designing Synchronous Counters

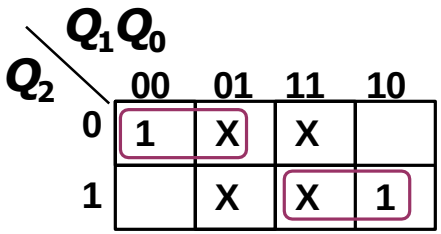
- 3-bit Gray code counter: flip-flop inputs.



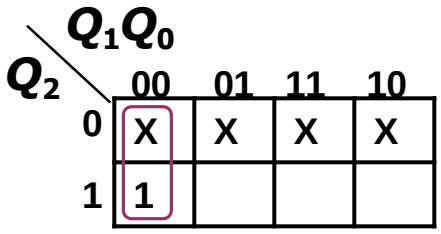
$JQ_2 = Q_1.Q_0'$



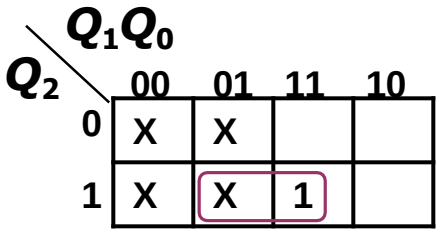
$JQ_1 = Q_2'.Q_0$



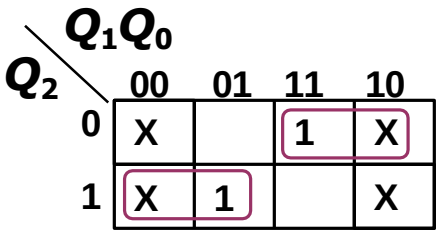
$JQ_0 = Q_2.Q_1 + Q_2'.Q_1'$
$= (Q_2 \oplus Q_1)'$



$KQ_2 = Q_1'.Q_0'$



$KQ_1 = Q_2.Q_0$



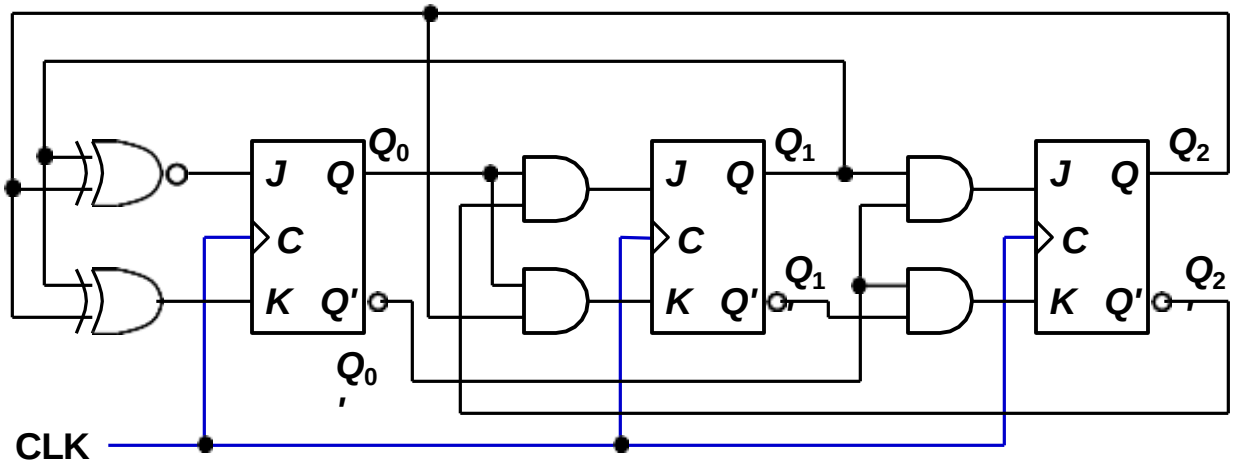
$KQ_0 = Q_2.Q_1' + Q_2'.Q_1$
$= Q_2 \oplus Q_1$

Designing Synchronous Counters

- 3-bit Gray code counter: logic diagram.

$JQ_2 = Q_1.Q_0'$ $JQ_1 = Q_2'.Q_0$ $JQ_0 = (Q_2 \oplus Q_1)'$

$KQ_2 = Q_1'.Q_0'$ $KQ_1 = Q_2.Q_0$ $KQ_0 = Q_2 \oplus Q_1$

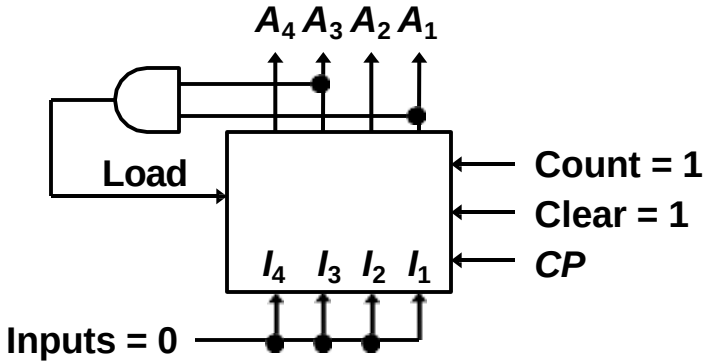


Counters with Parallel Load

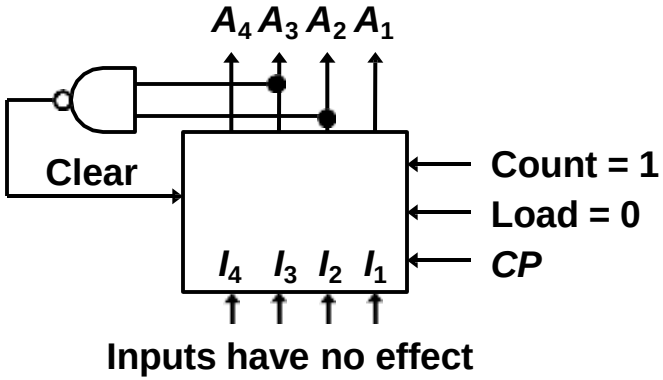
- Counters could be augmented with parallel load capability for the following purposes:
  - To start at a different state
  - To count a different sequence
  - As more sophisticated register with increment/decrement functionality.

Counters with Parallel Load

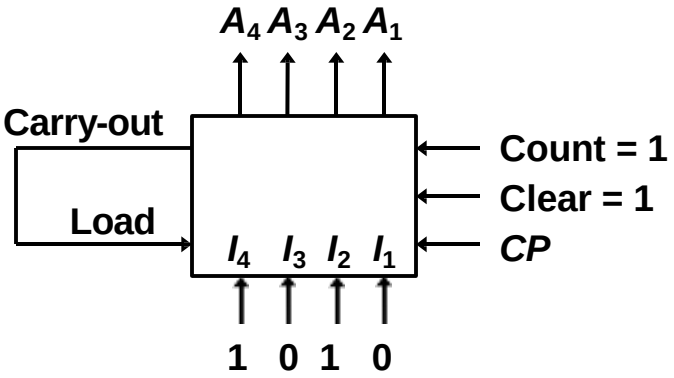
- Different ways of getting a MOD-6 counter:



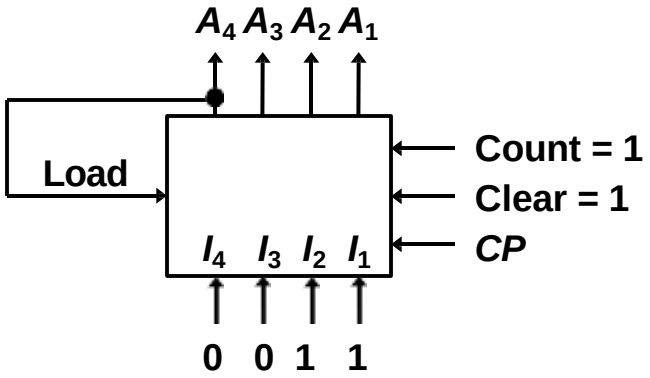
(a) Binary states 0,1,2,3,4,5.



(b) Binary states 0,1,2,3,4,5.



(c) Binary states 10,11,12,13,14,15.

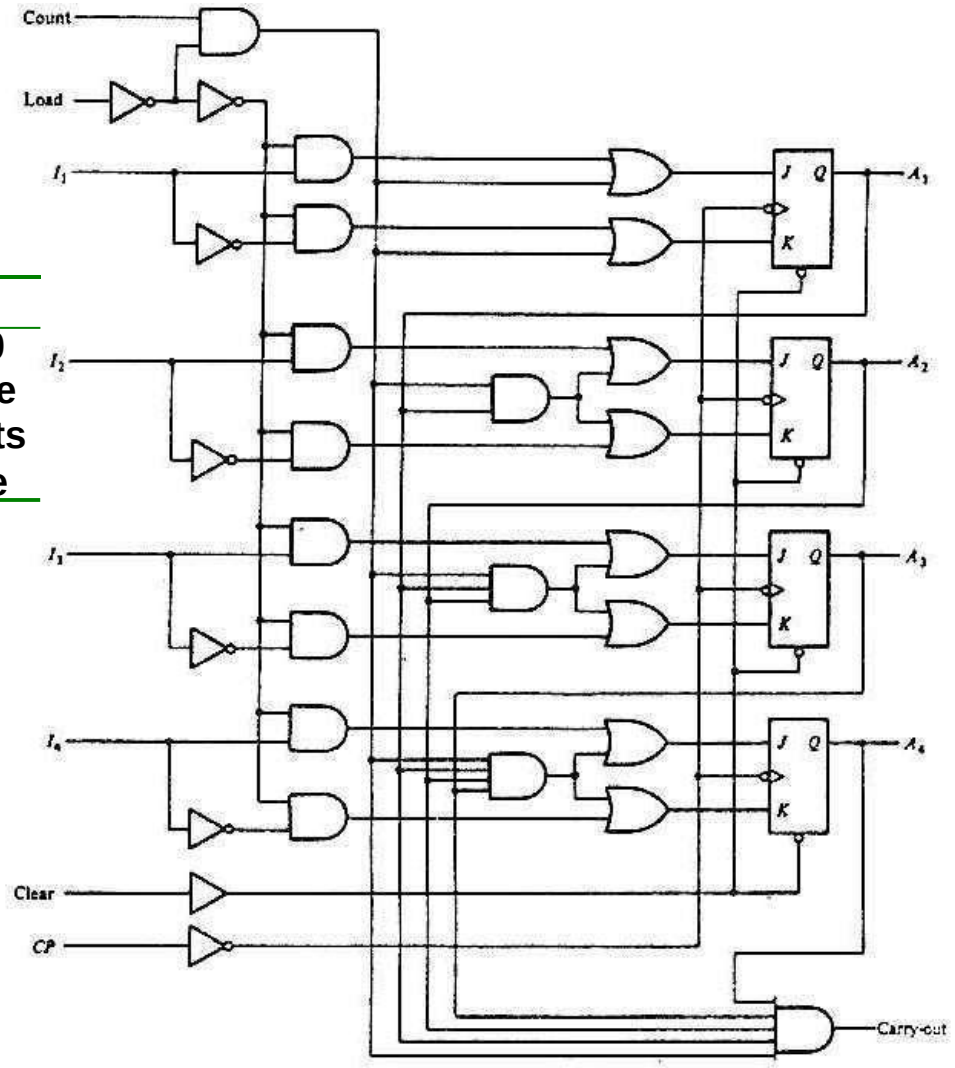


(d) Binary states 3,4,5,6,7,8.

Counters with Parallel Load

- 4-bit counter with parallel load.

Clear	CP	Load	Count	Function
0	X	X	X	Clear to 0
1	X	0	0	No change
1	↑	1	X	Load inputs
1	↑	0	1	Next state



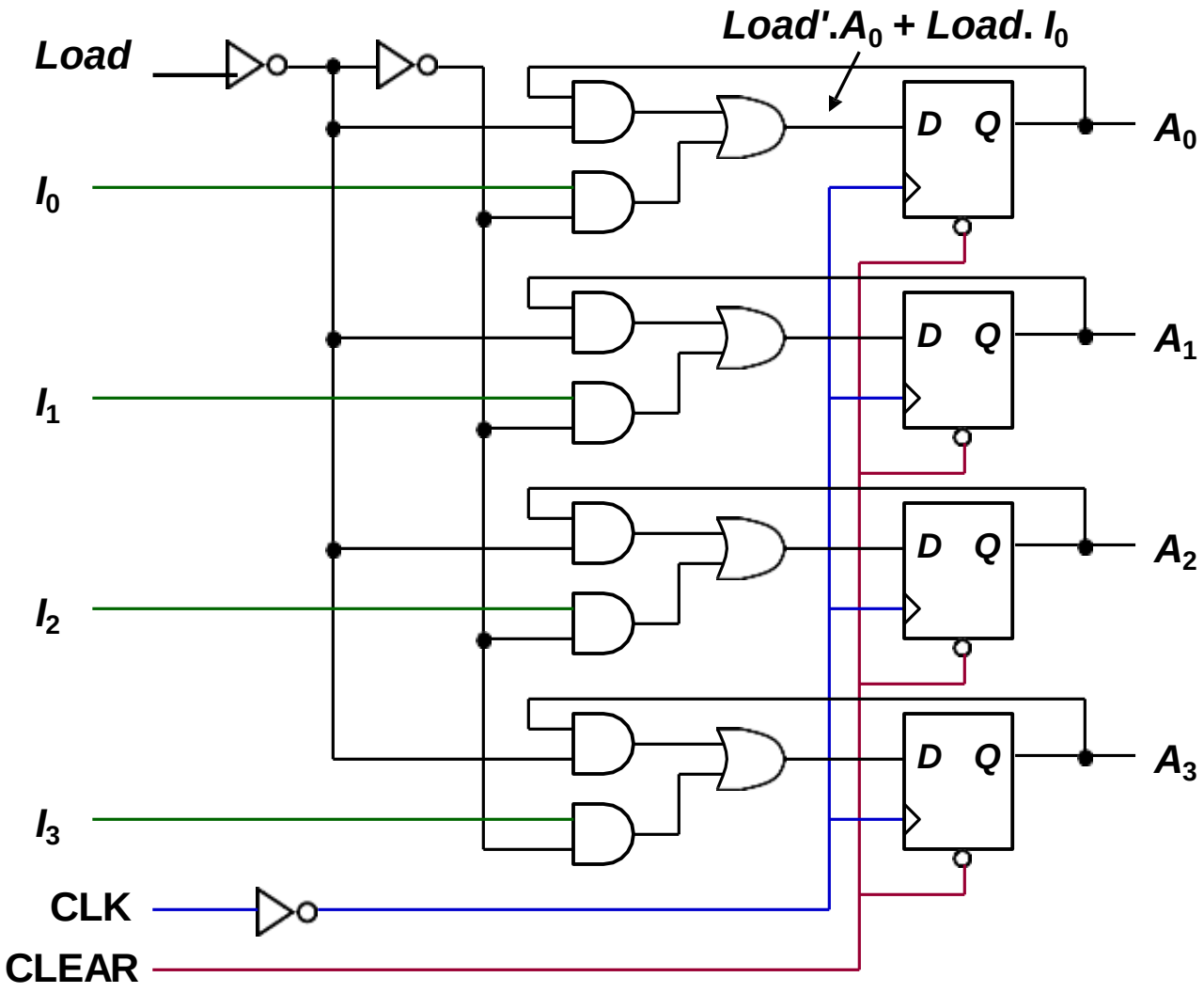
Introduction: Registers

- An *n*-bit register has a group of *n* flip-flops and some logic gates and is capable of storing *n* bits of information.
- The flip-flops store the information while the gates control when and how new information is transferred into the register.
- Some functions of register:
  - retrieve data from register
  - store/load new data into register (serial or parallel)
  - shift the data within register (left or right)

Registers With Parallel Load

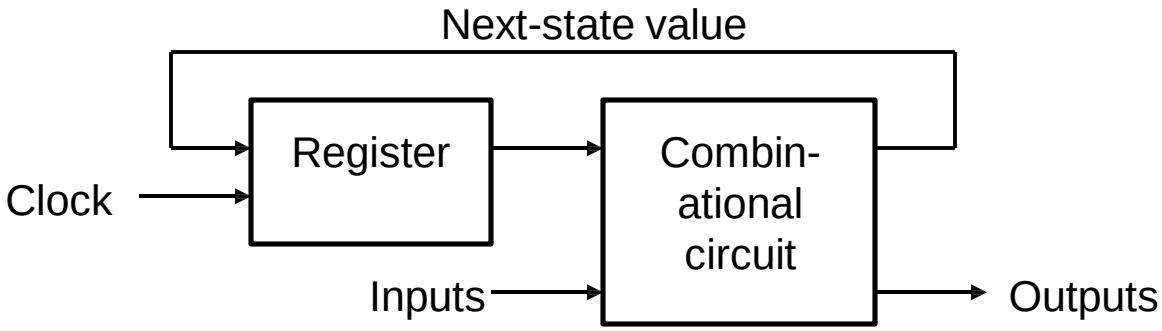
- Instead of loading the register at every clock pulse, we may want to control when to load.
- *Loading* a register: transfer new information into the register. Requires a *load* control input.
- *Parallel loading*: all bits are loaded simultaneously.

Registers With Parallel Load



Using Registers to implement Sequential Circuits

- A sequential circuit may consist of a *register* (memory) and a *combinational circuit*.



- The external inputs and present states of the register determine the next states of the register and the external outputs, through the combinational circuit.
- The combinational circuit may be implemented by any of the methods covered in *MSI components* and *Programmable Logic Devices*.

Using Registers to implement Sequential Circuits

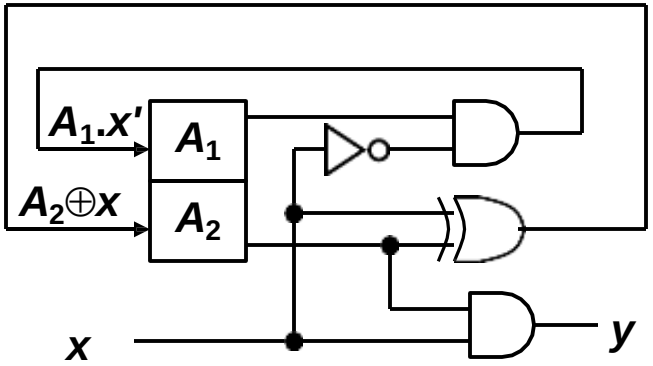
- Example 1:

$A_1^+ = \Sigma\, m(4,6) = A_1.x'$

$A_2^+ = \Sigma\, m(1,2,5,6) = A_2.x' + A_2'.x = A_2 \oplus x$

$y = \Sigma\, m(3,7) = A_2.x$

Present state		Input	Next State		Output
A_1	A_2	x	A_1^+	A_2^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

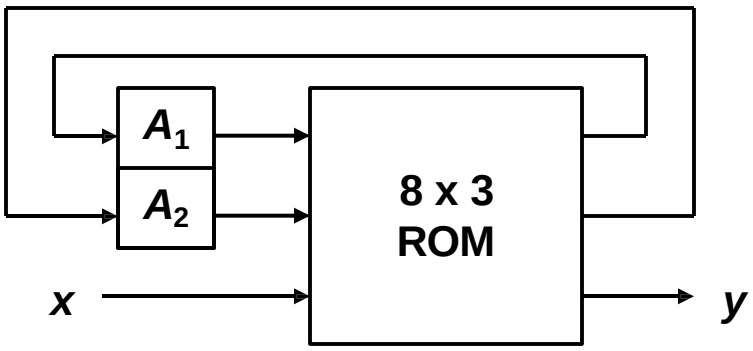


Using Registers to implement Sequential Circuits

- Example 2: Repeat example 1, but use a ROM.

Address			Outputs		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

ROM truth table

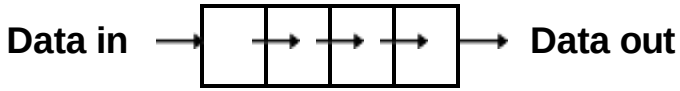


Shift Registers

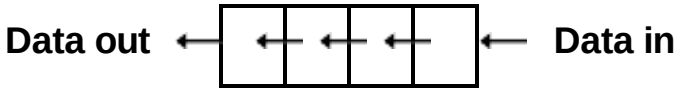
- Another function of a register, besides storage, is to provide for *data movements*.
- Each *stage* (flip-flop) in a shift register represents one bit of storage, and the shifting capability of a register permits the movement of data from stage to stage within the register, or into or out of the register upon application of clock pulses.

Shift Registers

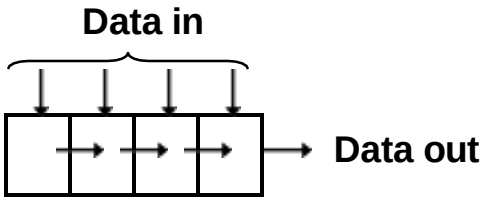
- Basic data movement in shift registers (four bits are used for illustration).



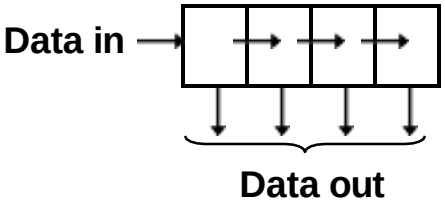
(a) Serial in/shift right/serial out



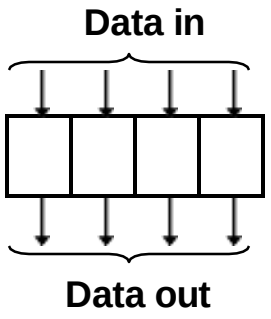
(b) Serial in/shift left/serial out



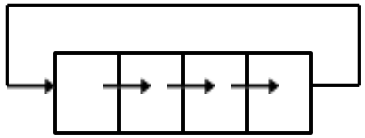
(c) Parallel in/serial out



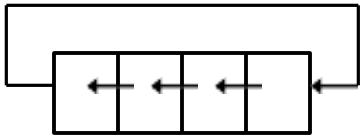
(d) Serial in/parallel out



(e) Parallel in / parallel out



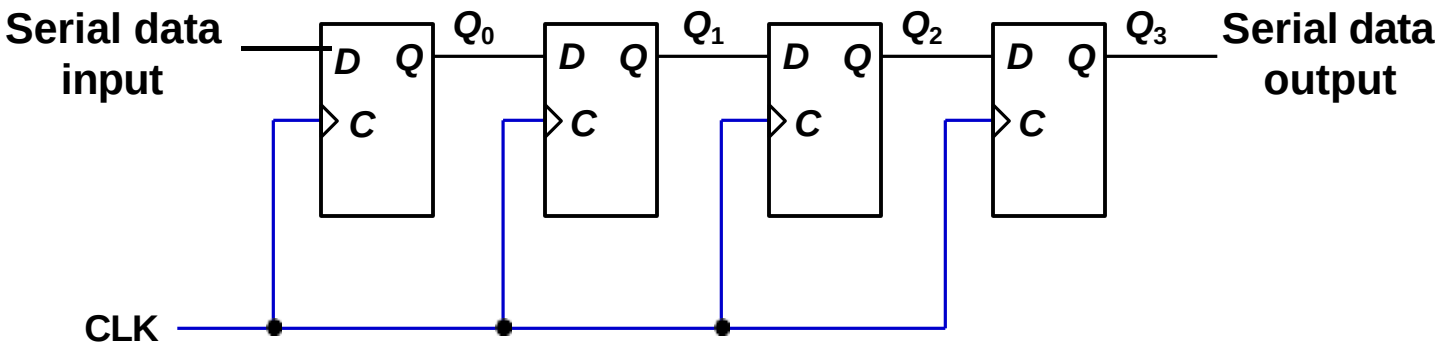
(f) Rotate right



(g) Rotate left

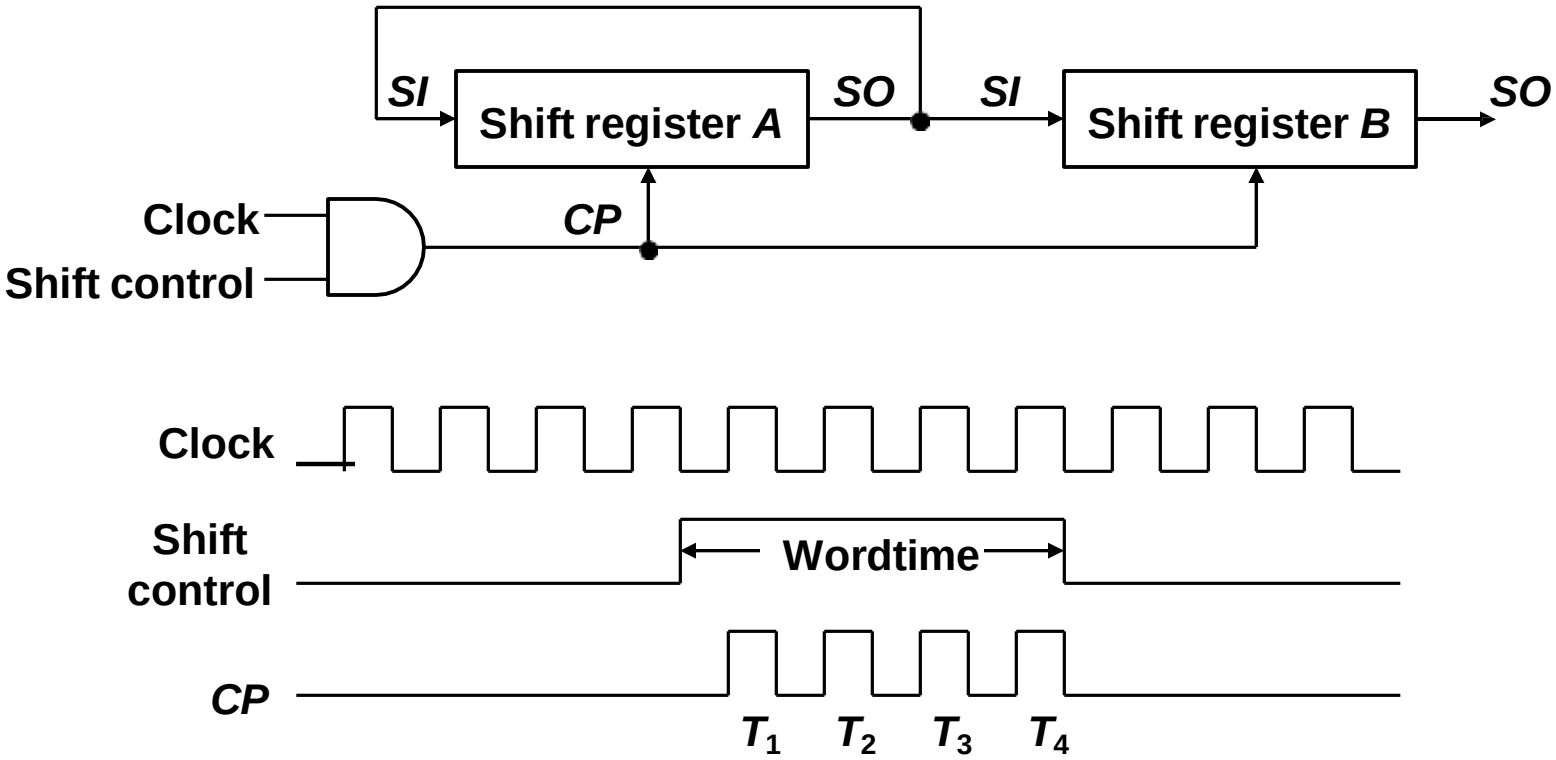
Serial In/Serial Out Shift Registers

- Accepts data serially – one bit at a time – and also produces output serially.



Serial In/Serial Out Shift Registers

- Application: Serial transfer of data from one register to another.



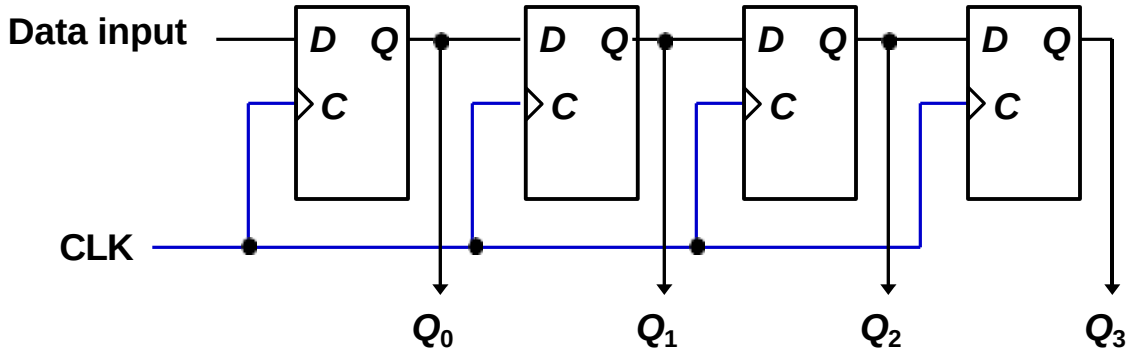
Serial In/Serial Out Shift Registers

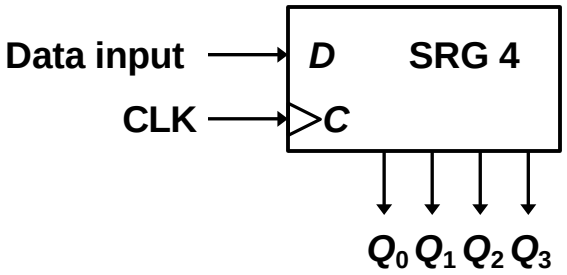
- Serial-transfer example.

Timing Pulse	Shift register *A*				Shift register *B*				Serial output of *B*
Initial value	1	0	1	1	0	0	1	0	0
After T_1	1	1	0	1	1	0	0	1	1
After T_2	1	1	1	0	1	1	0	0	0
After T_3	0	1	1	1	0	1	1	0	0
After T_4	1	0	1	1	1	0	1	1	1

Serial In/Parallel Out Shift Registers

- Accepts data serially.
- Outputs of all stages are available simultaneously.

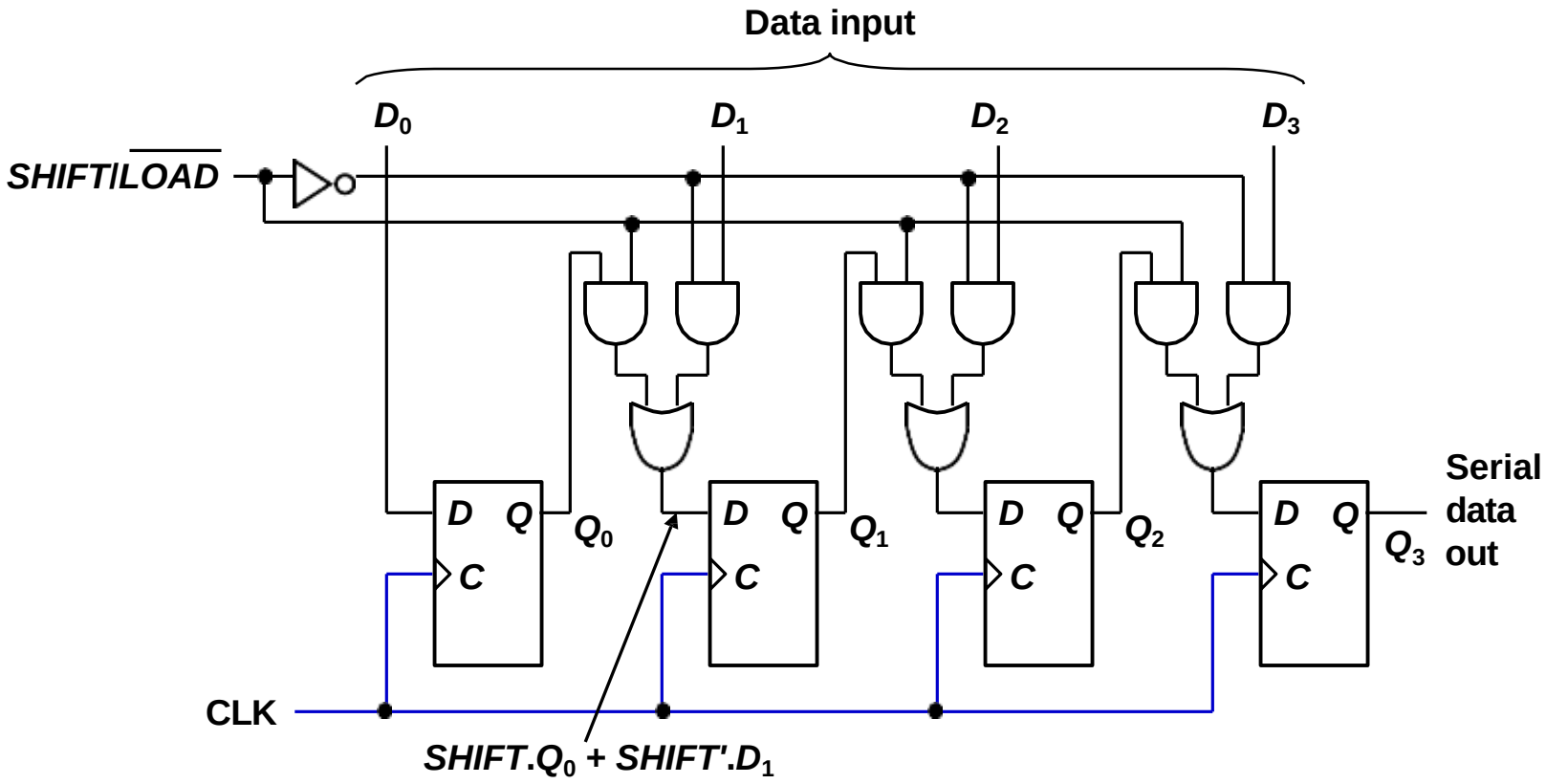




Logic symbol

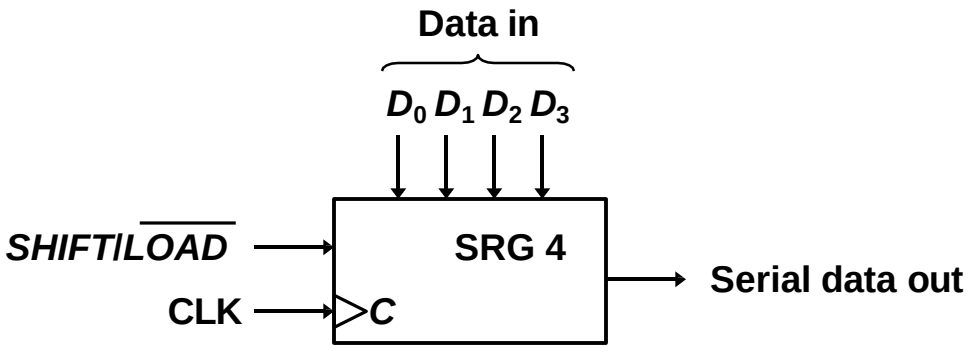
Parallel In/Serial Out Shift Registers

- Bits are entered simultaneously, but output is serial.

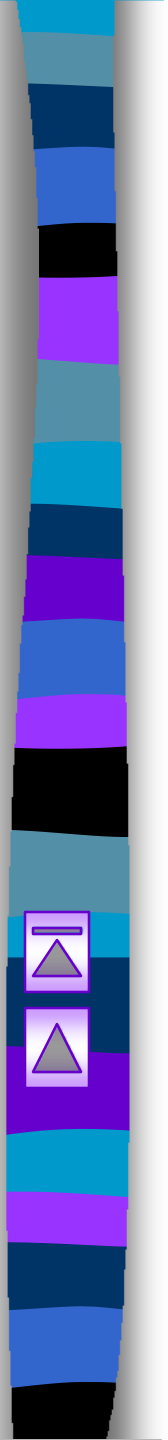


Parallel In/Serial Out Shift Registers

- Bits are entered simultaneously, but output is serial.



Logic symbol



End of segment