EE 200: Digital Logic Circuit Design

Dr Radwan E Abdel-Aal, COE

Unit 5
Registers and Counters

Charles Kime & Thomas Kaminski

Unit 5: Registers and Counters
Useful MSI blocks made of Flip-Flops

Chapter 6: Registers and Counters

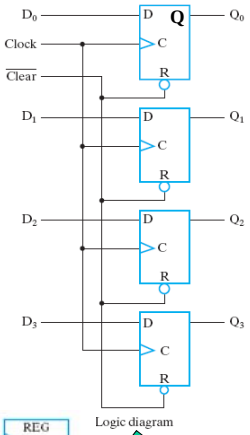
1. Registers with parallel load
2. **Shift** Registers

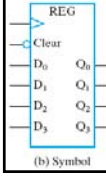
3. Counters
   - Asynchronous (Ripple)
   - Synchronous

Registers

- **Register – a set of binary storage elements**
- **Used to perform simple operations on data such as storage, movement, and processing**
  Examples: load, shift, rotate, increment, etc.
- **A processor processes data by performing operations on registers, e.g. ADD A, B where A and B are say 32-bit registers**

Examples: - 4-bit Register, with Clear
- Selective Parallel Load by gating the clock



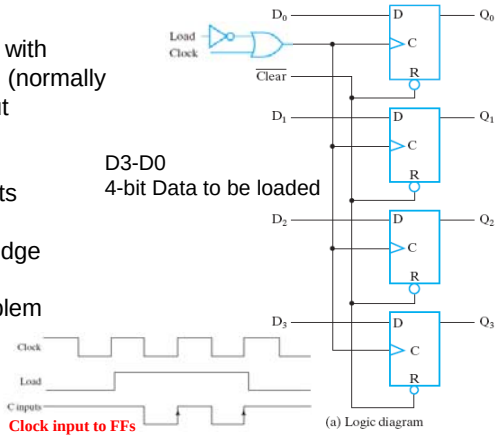


Encapsulate
As a **Register Function Block**

Reset (Cleared) Asynchronously with A 0 pulse on the (normally High) Clear input

Loaded with Data at the inputs (D → Q) **at every** clock edge

What is the problem With this?



D3-D0
4-bit Data to be loaded

We need **controlled loading**, e.g. **only** when input data is valid and a Load Signal arrives - (not on every clock pulse!)

One Solution: Gate the clock to the flip flops so that clock pulses reach the FFs only when Load = 1

Better Approach

Register with Parallel Load & no clock skew

Avoid clock gating. Apply Load control at D input

Avoid clock gating if you can! Why?

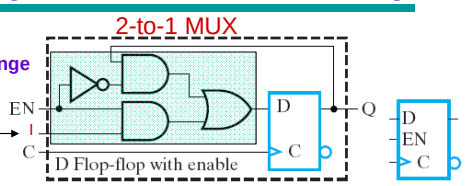
It causes clock skew – clock pulses may arrive to various registers in the system at different times

So such flip flops will change state at different times

No good- could lead to erroneous state transitions or data transfers!

EN = 1 for Load
EN = 0 for no change

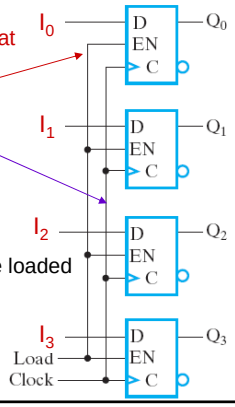
External Data to be loaded



But we now control what Clock pulses do using The EN control on D

Clock pulses go all the time Unobstructed To all FFs

I3-I0 Data to be loaded

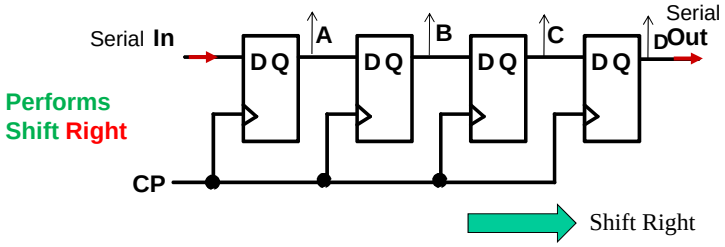


Load now does not gate the clock,
But controls what goes into the D input
Load = 1 → EN = 1, D = Ext. input data will be loaded on the next clock edge
Load = 0 → EN = 0, D = present Q, so no change on the next clock edge

Shift Registers

Shift Register → Serial Loading

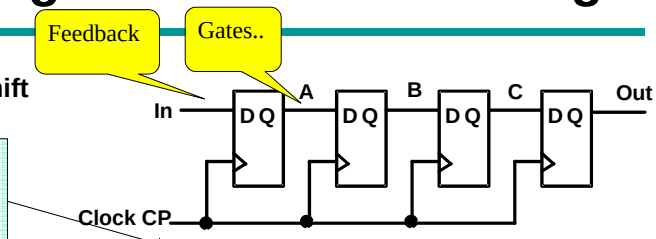
- **A Shift Registers moves data laterally (←→) within the register FFs toward the MSB or LSB position**
- **In its simplest form, the shift register is a set of D flip-flops having a common clock & connected in a row like this:**



- **Data input, In, is the *serial input* (the *shift right input*).**
- **Data output, Out, is the *serial output* (=D).**
- **The vector (ABCD) is called the *parallel output* of the register.**

Shift Registers: Serial Loading

- Behavior of the 4- stage serial shift register
- Initial register state just before the first clock pulse arrives
- Assume Register was cleared to 0000 initially, e.g. by S-R
- T1: State after the first pulse and before the second
- Complete the last two rows of the table



CP	In	A	B	C	D: Serial Out
	0	0	0	0	0
T1	1	0	0	0	0
T2	1	1	0	0	0
T3	0	1	1	0	0
T4	1	0	1	1	0
T5	1				1
T6	1				1
T7					0

After 4 Clock Pulses:

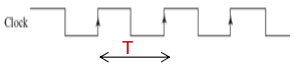
2. Serial I/P Starts to appear Serially at O/P (after 4T delay)

1. 4-bit Serial I/P Appears in parallel In register after 4 clock pulses (serial loading of the register)

Applications: Delay, Serial to Parallel conversion

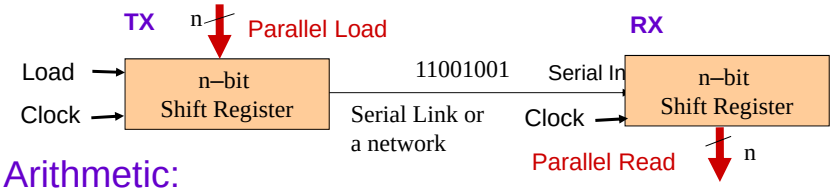
Some Applications of n-stage Shift Register

- Delay:



→ Serial input sequence starts to appear at output after n-clock cycles (i.e. is delayed by nT ns)

- Parallel to Serial and Serial to Parallel conversion:

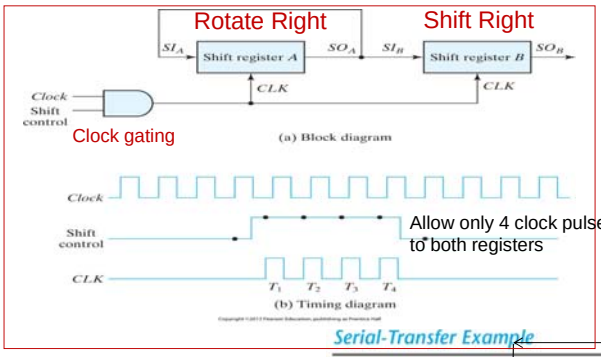


- Arithmetic:

→ Shifting 1 place to the right (feeding in 0s) = dividing by ?

→ Shifting 1 place to the left (feeding in 0s) = multiplying by ?

Shift Registers with Feedback



Each of A, B is a 4-bit Right-Shift register

Clock gating **is not a good practice**

As it leads to Clock skew

Allow only 4 clock pulses to both registers

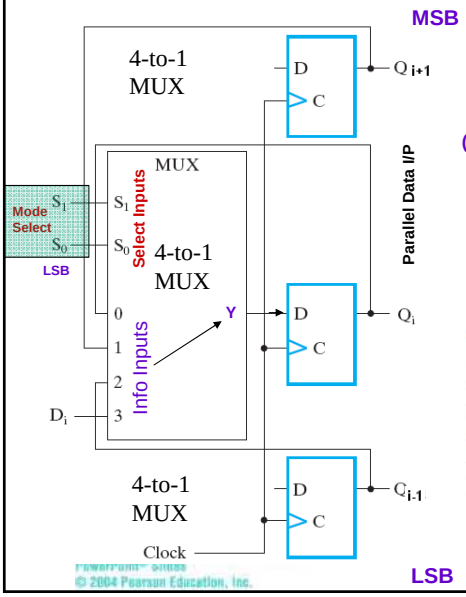
Regardless of initial Contents in B **(flushed out)**

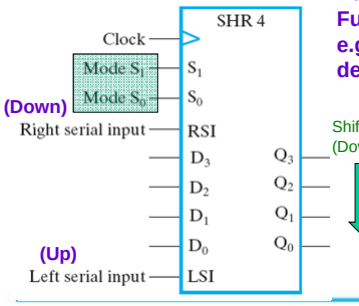
Serial-Transfer Example

Timing Pulse	Shift Register A	Shift Register B
Initial value	1 0 1 1	0 0 1 0
After T_1	1 1 0 1	1 0 0 1
After T_2	1 1 1 0	1 1 0 0
After T_3	0 1 1 1	0 1 1 0
After T_4	1 0 1 1	1 0 1 1

After 4 clock pulses: Initial 4-bit value restored in A and serially loaded into B

Universal (Multi-purpose) Shift Register with 4 Functions:
- Parallel Load - Shift left - Shift Right - No Change





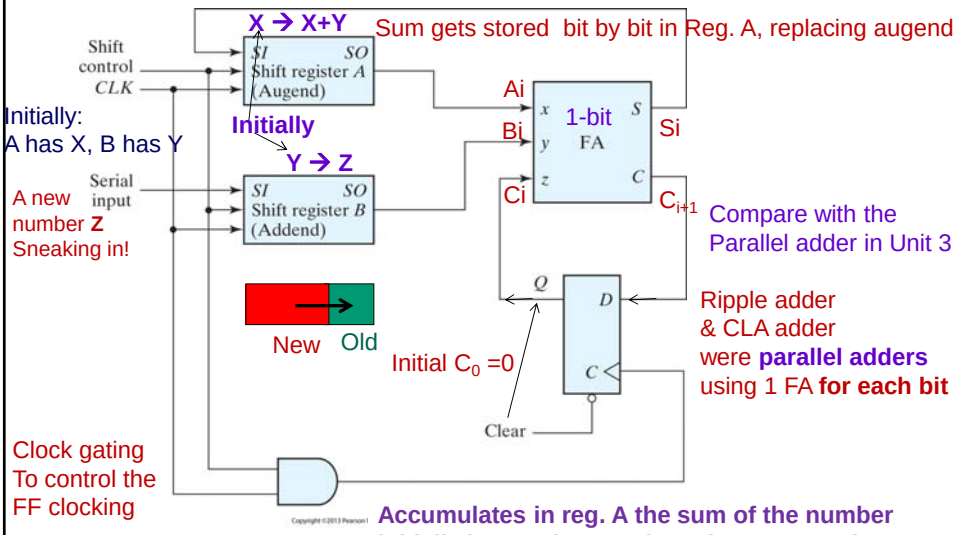
Add other Functions? e.g. increment, decrement

Mode control S_1	S_0	Register Operation
0	0	No change
0	1	Shift down (Right)
1	0	Shift up (Left))
1	1	Parallel load

Right = → LSB
Left = → MSB

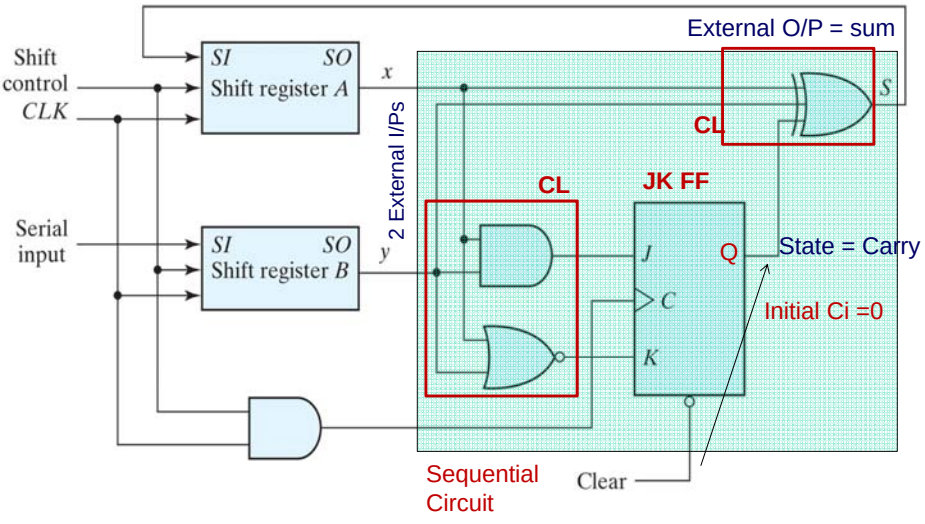
Serial (bit-by-bit) adder

1. Using shift registers, 1 Full Adder, and 1 FF



Serial Adder (bit-by-bit addition)

2. Using shift regs. and a sequential circuit



Serial Adder (bit-by-bit addition)

Using shift regs. and a sequential circuit

3 K-maps → CL on previous slide

Table 6.2
State Table for Serial Adder

Carry Out

Sum

Carry IN

Present State	Inputs		Next State	Output	Flip-Flop Inputs	
Q	x	y	Q	S	J_Q	K_Q
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Excitation

Counters

- A counter is a register that has its parallel outputs go through a specific count sequence as clock pulses arrive
- Counting sequences can be binary, BCD, etc.
- Counting can be up, down
- A modulo-n counter has n different counting states, e.g. it goes through counts 0,1,2, …, (n-1),0,1,…
  e.g. a modulo-10 up counter may count: 0,1,..,9,0,1,..
  → Modulo n counter divides the clock frequency by n
  (show waveforms)

Some Applications of counters:

- Counting events
- Timers
- Frequency division

This is a modulo-?
Up/Down? Counter

Repeated counting cycle
# of clock pulses sent = ?
# of pulses at MSB (A) = ?

ABC (LSB)
000
001
010
011
100
101

Successive Clock Pulses 000 001

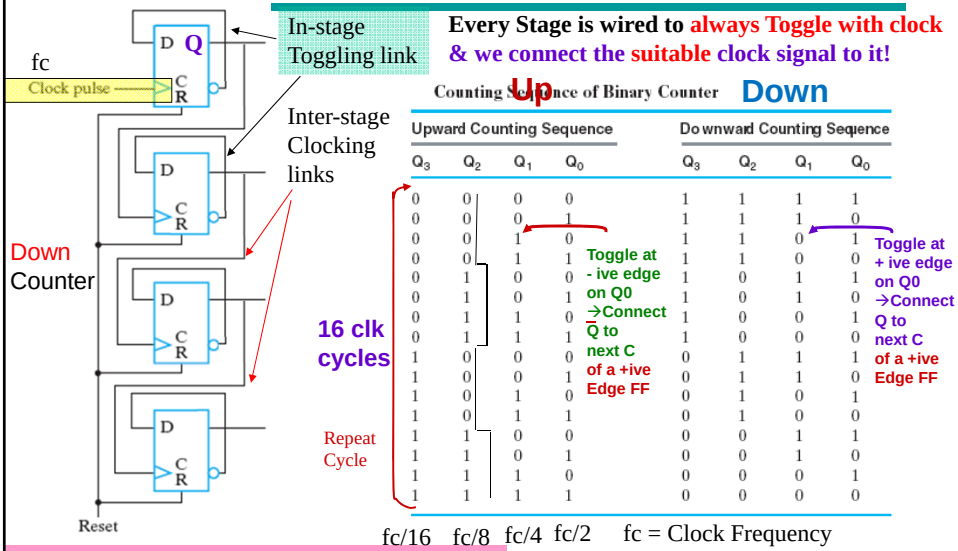
Implementing Counters

Two Basic Approaches: 1. Async (Ripple) & 2. Sync.

- **1. Asynchronous (Ripple) Counters**
  - The external clock is connected only to the clock input of the LSB bit flip-flop (first counter stage)
  - Then the output of a stage provides the clock I/P to the next stage
  - i.e. circuit is not synchronous -
    (since no common clock to *all* stages)
  - Advantage:
    - Simple circuit
  - Disadvantages:
    - Output changes are delayed further for each stage toward the LSB (Ripple Effect)
    - This ripple propagation delay limits the maximum clock frequency that can be used
      (same factor that limited speed of the ripple adder)

Ripple (asynchronous) Counters

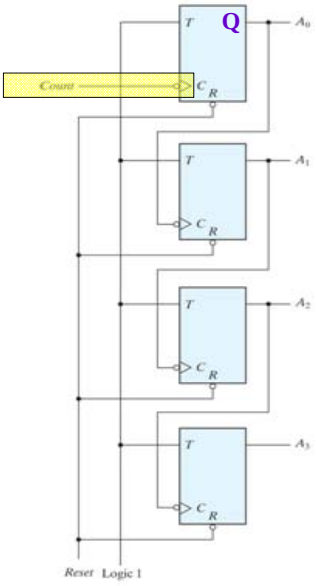
Modulo-16 binary counter (n = 4) using D FFs



How to make an Up/Down ripple counter by adding MUXs?

16

With T FFs- Toggling obtained with T = 1 (no need to externally connect Q' to D for each FF)



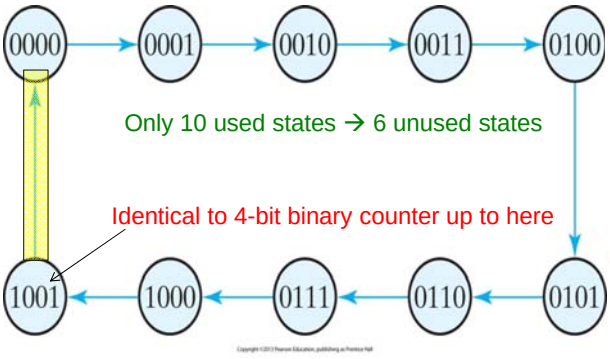
Binary Up Counter
0000 → 1111 (16 Counts)

T FF Advantage: No need for an external Q' to D connection for each FF to toggle

But for Up counting, Clock is taken from Previous Q not Q'- Why?

BCD UP (Decade) Ripple Counter

- **Counts 0, 1, …8, 9, 0 (modulo 10)**
- **4-bits → 4 FFs**

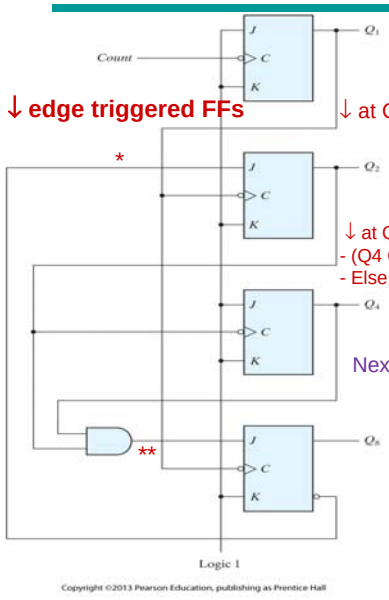


Upward Counting Sequence

Q_3	Q_2	Q_1	Q_0
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
1	0	0	1

6 unused states

BCD (Decade) UP Ripple Counter with JK FF



↓ edge triggered FFs

↓ at Q1, Q8=1 → Q2 to 0*

↓ at Q1:**
- (Q4 Q2)=11: Toggle
- Else: reset to 0

Next pulse at end count

↓ 1
0

JK Characteristic

J	K	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q(t)}$	Toggle

Upward Counting Sequence

Q_8	Q_4	Q_2	Q_1
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
1	0	0	1

Toggles on ↓ at Q2 (As usual)

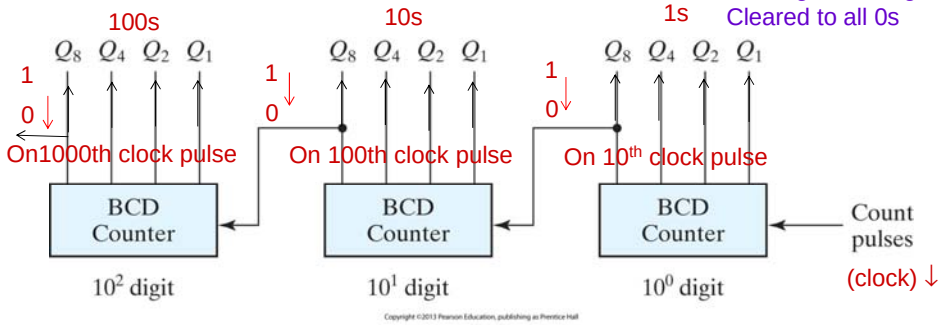
Toggles on ↓ at Q1 if Q8 = 0, otherwise Reset

Toggles Always

19

Cascading Ripple Counters
3-Decade Ripple Up Counter (000-999)

Starting with stages Cleared to all 0s



Divide-by-1000 Counter

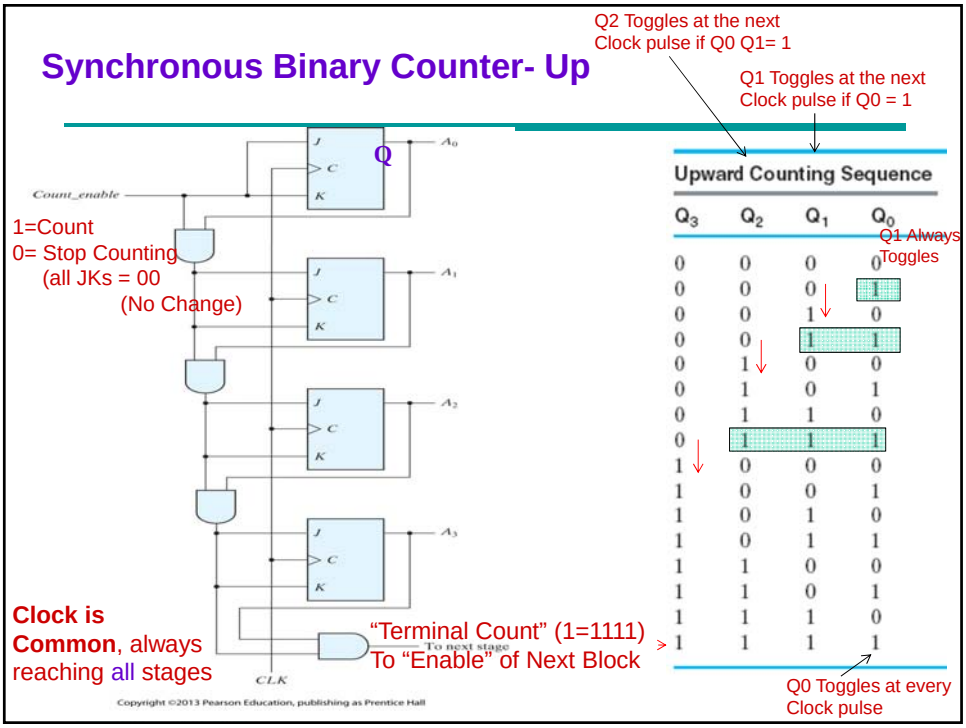
Each stage is a BCD up Decade Counter (last slide)

Clock Rippling at 2 levels:
- Within each decade stage
- and also between the stages

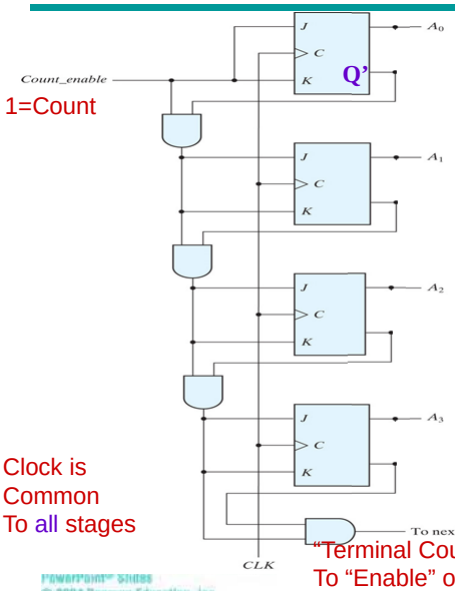
2. Synchronous Counters

- The same System clock is connected **directly** to the clock inputs of **ALL flip-flop stages**(→ truly synchronous)
- **No fiddling with clock- instead, we control operation of individual FFs through data or control inputs e.g. D, JK, T**
- **Any counter can be systematically designed as a sequential cct – see Unit 4**
  - A combinational logic circuit is used to implement the desired state sequencing through inputs to the Ds of the flip-flop stages
- **But simpler ad hoc approaches can be used for regular counting patterns e.g. binary up or down**

Synchronous Binary Counter- Up



Synchronous Binary Counter- Down



Clock is Common To all stages

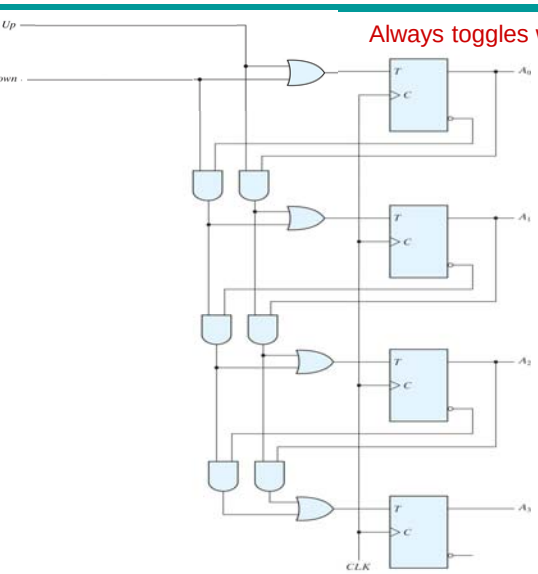
Downward Counting Sequence			
Q_3	Q_2	Q_1	Q_0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

"Terminal Count" (1=0000)
To "Enable" of Next Block

Q0 Toggles at every Clock pulse

Synchronous Binary Counter- Up/Down With T Flip Flop

Down Up
0 1
1 0



Always toggles with every clock pulse (for up or down)

For a binary counter: Irregular counting pattern here

BCD UP Counter: …8 9 0 1 … with T FFs

Additional O/P with 1 pulse of 1 clock period duration every counting cycle (divide by 10)

Table 6.5
State Table for BCD Counter

Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

Toggles when Q1=1 & Q8 = 0

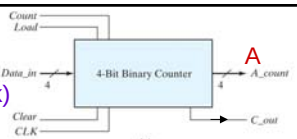
Toggles Always

$$T_{Q1} = 1$$
$$T_{Q2} = Q_8' Q_1$$
$$T_{Q4} = Q_2 Q_1$$
$$T_{Q8} = Q_4 Q_2 Q_1 + Q_8 Q_1$$
$$y = Q_8 Q_1$$

From T K-maps (use Xs)

A more systematic approach to Design, compared to slide 19 (Async)

Binary UP Counter with Parallel Load (Functional Block)



Count Enable (1)

Synchronous Load Enable (1)

S

R

Select between Load/Count Conditions

LSB

Count cond. for the FF

External Data to be loaded in

Function Table

Clear (async)	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next
1	↑	0	0	No change

Note: JK FF is **more cumbersome** than D For loading data!

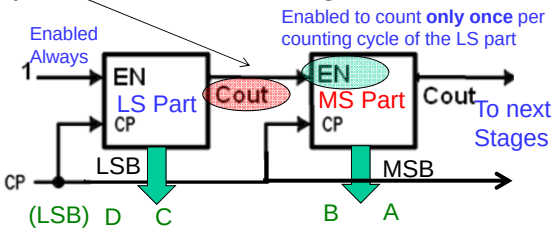
Direct Clear (asynchronous) Active Low (1 for normal operation)

1 at "terminal count" (1111) to enable Next 4-bit stage (for cascading)

Designing Larger Counters by cascading smaller ones Using the Cout (Terminal Count) to EN of next stage

- Given two 2-bit (modulo-4) counter, how to design a 4-bit (modulo-16) counter?

MS Part		LS Part		
A	B	C	D (LSB)	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0 Cout=1	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1 Cout=1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0 Cout=1	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1 Cout=1	1	1	15



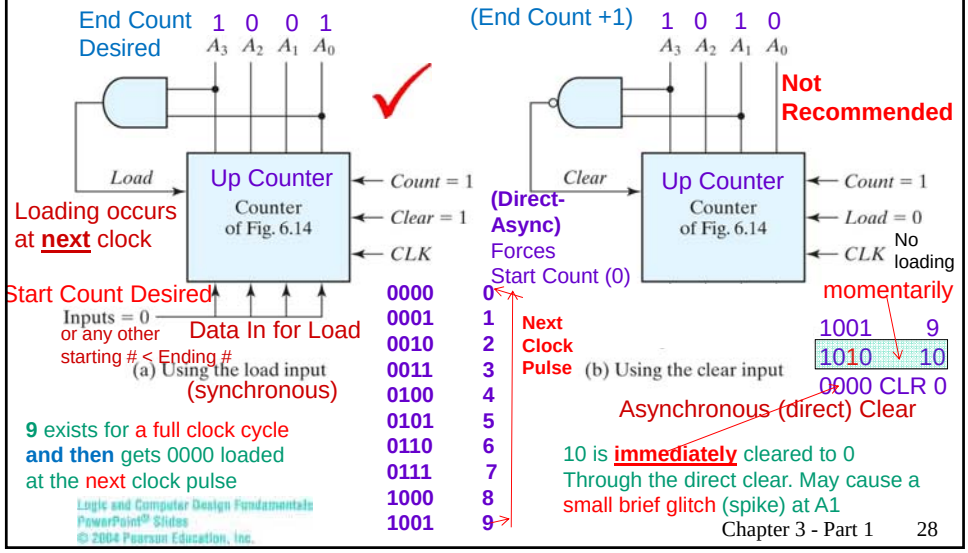
LS Part generates a single Cout pulse (lasting for 1 clk Cycle only) every time it finishes its counting cycle

MS Part is enabled to increment only once when it receives this pulse at its EN I/P

Example: Design a 16-bit (0-255) binary counter using two of the 4-bit (0-15) universal binary counters on the previous slide

Using this count/load/clr functional block to implement a modulo-n Counter

Two Approaches for a **BCD** (0 to 9) Counter



9 exists for a full clock cycle **and then** gets 0000 loaded at the next clock pulse

10 is **immediately** cleared to 0 Through the direct clear. May cause a small brief glitch (spike) at A1

Modulo 6 Counter with Arbitrary Counting Sequence: Investigate Effect of Unused States on performance

Unused State 011 → X X X

Unused State 111 → X X X

Present State			Next State		
A	B	C	DA = A(t+1)	DB = B(t+1)	DC= C(t+1)
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0

Modulo?

At O/P A, Divide by ?

Count Sequence?

Number of FFs Needed?

Used states?

Unused states?

Three 3-variable K-maps for CL Optimization, with Xs

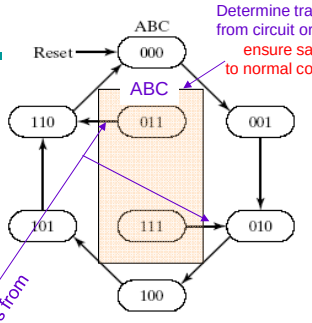
$$D_A = A \oplus B$$

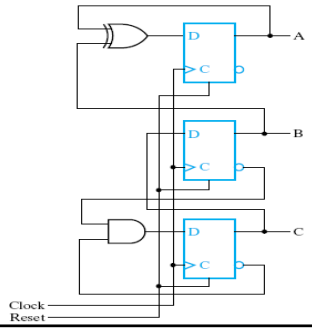
$$D_B = C$$

$$D_C = \overline{B}\,\overline{C}$$

Derive these transitions from The D eqns.

Two Unused states: Determine transitions from circuit or eqns and ensure safe return to normal counting seq.





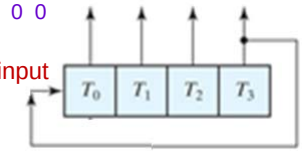
Generating a sequence of non-overlapping timing signals 1. Ring counter (Shift-Right with Rotate - D-type FFs)

T	0	1	2	3	
	0	0	0	1	Initially loaded by direct S/R
1	0	0	0		Pulse 1
	0	1	0	0	Pulse 2
	0	0	1	0	…
	0	0	0	1	
1	0	0	0		

4 different states

0 0 0 1 Q

D input



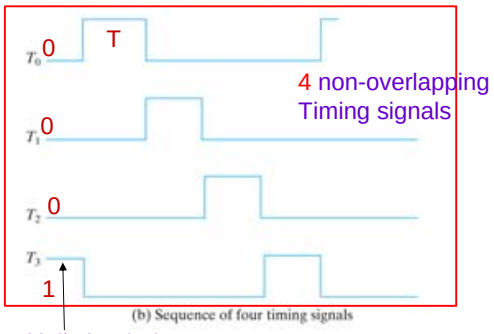
(a) Ring-counter (initial value = 0001)

Inserted directly with Set/Reset

Shift right register with 4 D-type FFs (i.e. Q to next RHS D),
+ Rotate Feedback: Q_3 → D_0

T = clock period

n intervals
Need n FFs

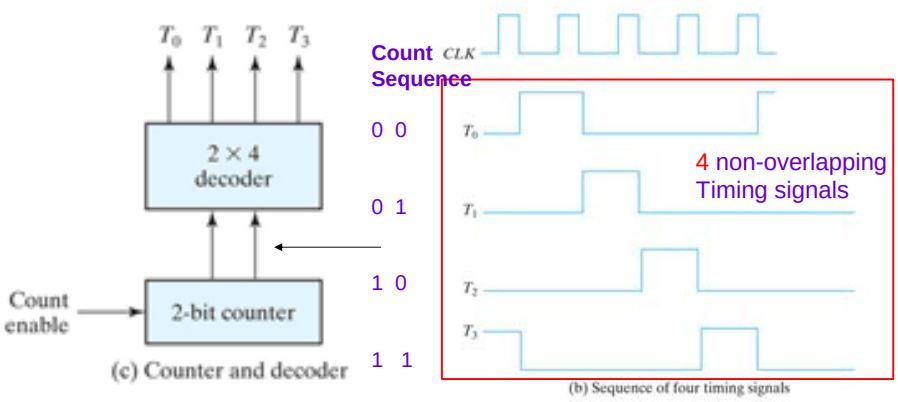


4 non-overlapping Timing signals

(b) Sequence of four timing signals

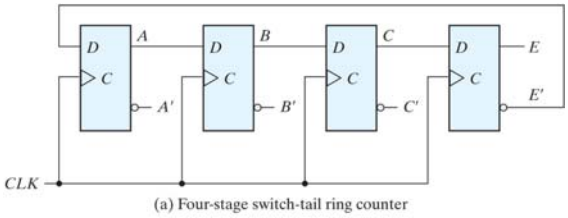
Initially loaded through directly Set-Rest of the D FFs

Generating a sequence of timing signals: 2. Counter-Decoder Implementation



Johnson Counter: 8 non-overlapping signals from 4-bit shift register with rotate - but E' instead of E!

n intervals
Need only n/2 FFs
- More efficient cct!



(a) Four-stage switch-tail ring counter

8 additional 2-input ANDs

8 non-overlapping timing signals are produced by the 8 AND gates shown In the table opposite

Sequence number	Flip-flop outputs A	B	C	E	AND gate required for output
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

8 non-overlapping Timing signals

(b) Count sequence and required decoding

8 different states
not 4 as on normal ring counter!