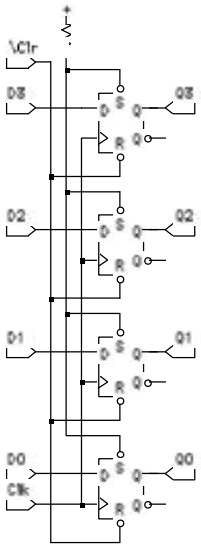
Chapter #7: Sequential Logic Case Studies

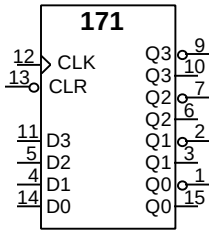
Contemporary Logic Design

Storage Register

Group of storage elements read/written as a unit

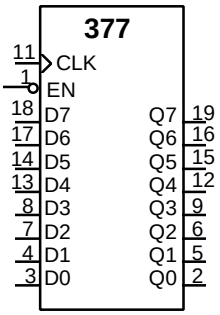


4-bit register constructed from 4 D FFs
Shared clock and clear lines



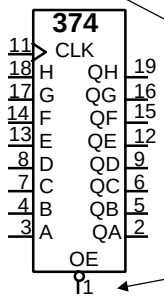
Input/Output Variations

Selective Load Capability
Tri-state or Open Collector Outputs
True and Complementary Outputs



74377 Octal D-type FFs with input enable

EN enabled low and lo-to-hi clock transition to load new data into register



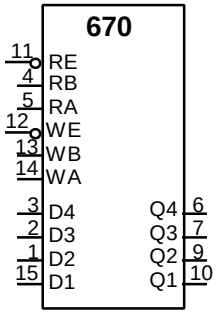
74374 Octal D-type FFs with output enable

OE asserted low presents FF state to output pins; otherwise high impedence

No. 7-3

Register Files

Two dimensional array of flipflops
Address used as index to a particular word
Word contents read or written



Separate Read and Write Enables
Separate Read and Write Address
Data Input, Q Outputs

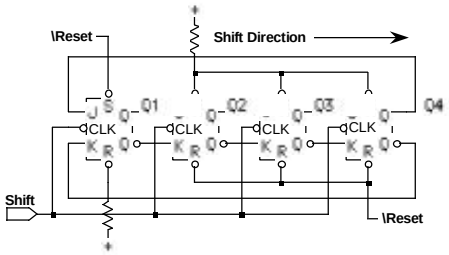
Contains 16 D-ffs, organized as four rows (words) of four elements (bits)

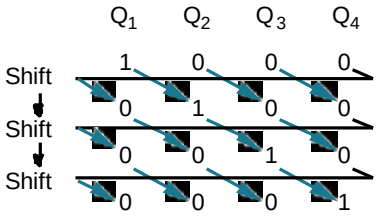
74670 4x4 Register File with Tri-state Outputs

No. 7-4

Shift Registers

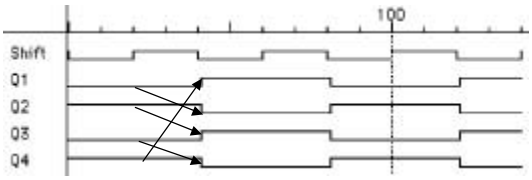
Storage + ability to circulate data among storage elements





Shift from left storage element to right neighbor on every lo-to-hi transition on shift signal

Wrap around from rightmost element to leftmost element

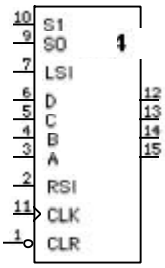


Master Slave FFs: sample inputs while clock is high; change outputs on falling edge

No. 7-5

Shift Register I/O

Serial vs. Parallel Inputs
Serial vs. Parallel Outputs
Shift Direction: Left vs. Right



74194 4-bit Universal Shift Register

Serial Inputs: LSI, RSI
Parallel Inputs: D, C, B, A
Parallel Outputs: QD, QC, QB, QA
Clear Signal
Positive Edge Triggered Devices

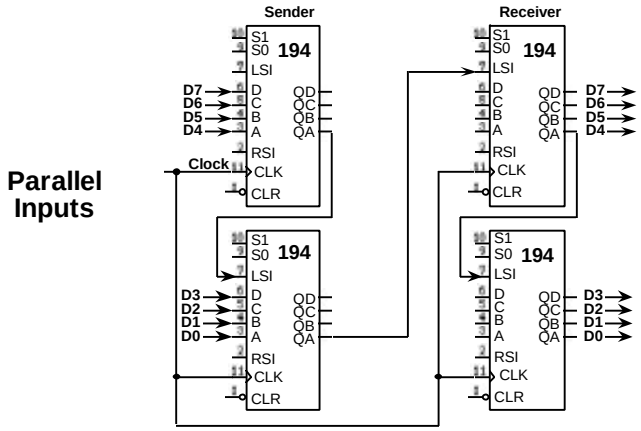
S1,S0 determine the shift function
S1 = 1, S0 = 1: Load on rising clk edge synchronous load
S1 = 1, S0 = 0: shift left on rising clk edge LSI replaces element D
S1 = 0, S0 = 1: shift right on rising clk edge RSI replaces element A
S1 = 0, S0 = 0: hold state

Multiplexing logic on input to each FF!

Shifters well suited for serial-to-parallel conversions, such as terminal to computer communications

No. 7-6

Shift Register Application: Parallel to Serial Conversion



Counters

- Proceed through a well-defined sequence of states in response to count signal

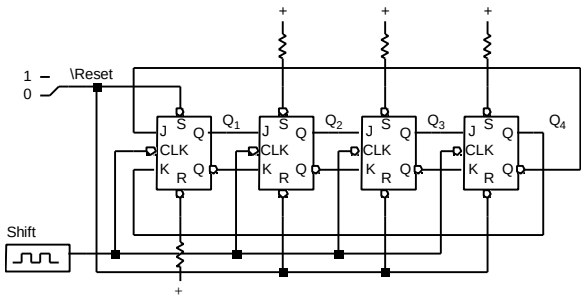
- 3 Bit Up-counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, …

- 3 Bit Down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, …

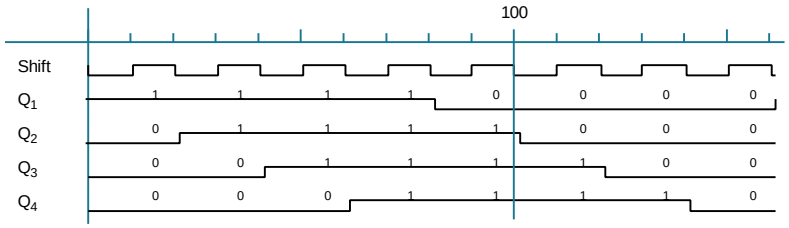
- Binary vs. BCD vs. Gray Code Counters

A counter is a "degenerate" finite state machine/sequential circuit where the state *is* the only output

Johnson (Mobius) Counter



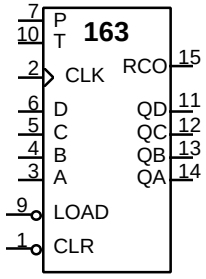
End-Around



8 possible states, single bit change per state, useful for avoiding glitches

No. 7-9

Catalog Counter



74163 Synchronous 4-Bit Upcounter

Synchronous Load and Clear Inputs

Positive Edge Triggered FFs

Parallel Load Data from D, C, B, A

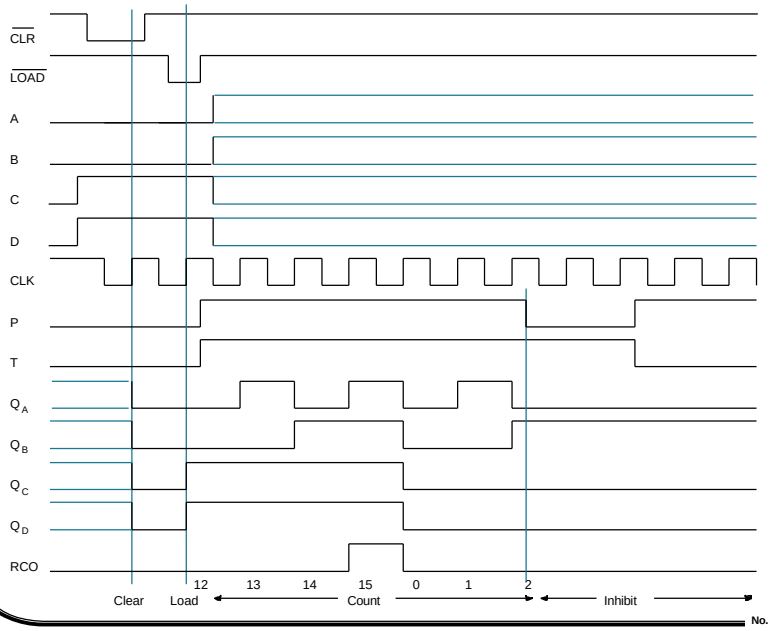
P, T Enable Inputs: both must be asserted to enable counting

RCO: asserted when counter enters its highest state 1111, used for cascading counters *"Ripple Carry Output"*

74161: similar in function, asynchronous load and reset

No. 7-10

74163 Detailed Timing Diagram



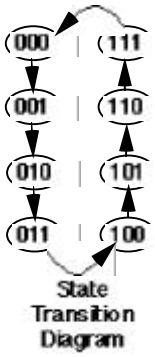
Counter Design Procedure

Introduction

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**

Example: **3-bit Binary Upcounter**



State Transition Diagram

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

State Transition Table **Flipflop Input Table**

Decide to 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"

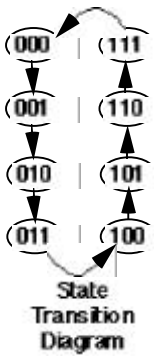
Counter Design Procedure

Introduction

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

Example: **3-bit Binary Upcounter**



State Transition Diagram

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

State Transition Table **Flipflop Input Table**

Decide to 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"

K-maps for Toggle Inputs:

Resulting Logic Circuit:

A \ CB	00	01	11	10
0				
1				

TA =

A \ CB	00	01	11	10
0				
1				

TB =

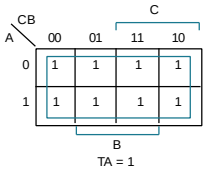
A \ CB	00	01	11	10
0				
1				

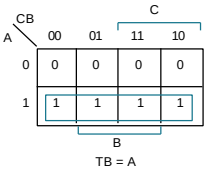
TC =

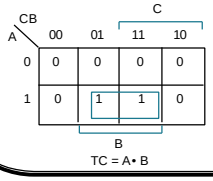
No. 7-15

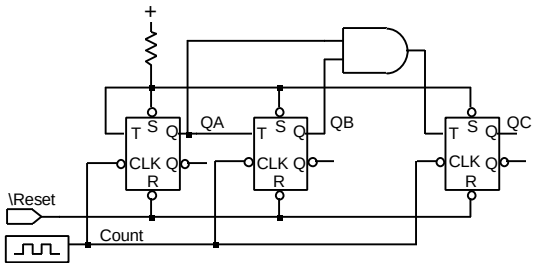
K-maps for Toggle Inputs:

Resulting Logic Circuit:

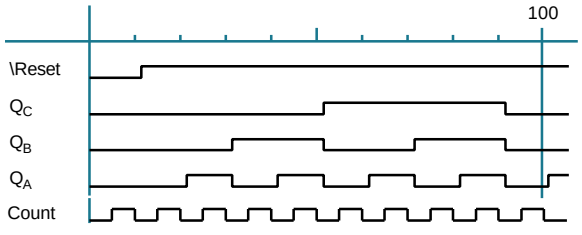








Timing Diagram:

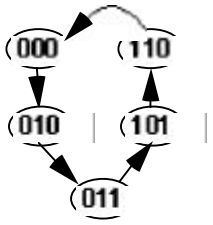


No. 7-16

More Complex Count Sequence

***Step 1*: Derive the State Transition Diagram**

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



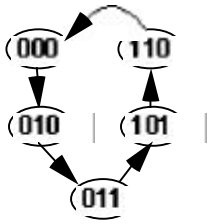
Present State / Next State

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

***Step 2*: State Transition Table**

***Step 1:* Derive the State Transition Diagram**

Sequence: 000, 010, 011, 101, 110



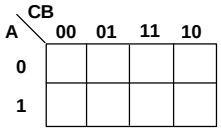
Present State / Next State

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

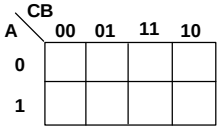
***Step 2:* State Transition Table**

Note the Don't Care conditions

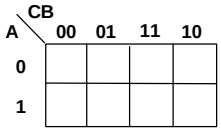
Step 3: K-Maps for Next State Functions



C+ =



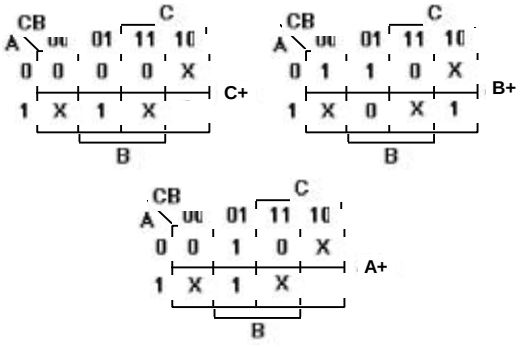
B+ =



A+ =

No. 7-19

Step 3: K-Maps for Next State Functions



No. 7-20

Step 4: 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

Toggle Excitation Table

Present State			Toggle Inputs		
C	B	A	TC	B	A
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Remapped Next State Functions

Counter Design Procedure

More Complex Counter Sequencing

Step 4: **Choose Flipflop Type for Implementation**
Use Excitation Table to Remap Next State Functions

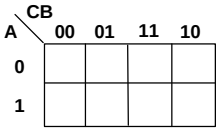
Q	Q+	T
0	0	0
0	1	1
1	0	1
1	1	0

Toggle Excitation Table

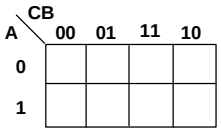
Present State			Toggle Inputs		
C	B	A	TC	B	A
0	0	0	0	0	1
0	0	1	X	1	1
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	X	1	1
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	X	1	1

Remapped Next State Functions

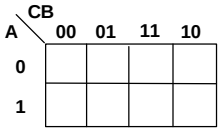
Remapped K-Maps



TC



TB



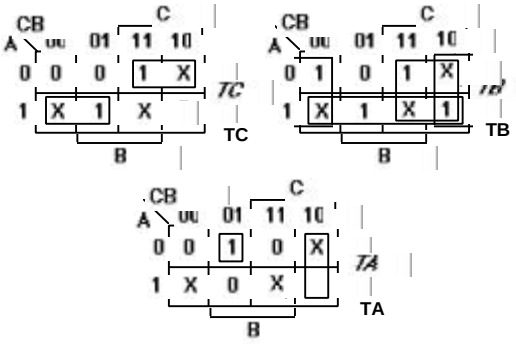
TA

TC =

TB =

TA =

Remapped K-Maps



TC = $\bar{A}$ C + A $\bar{C}$ = A xor C

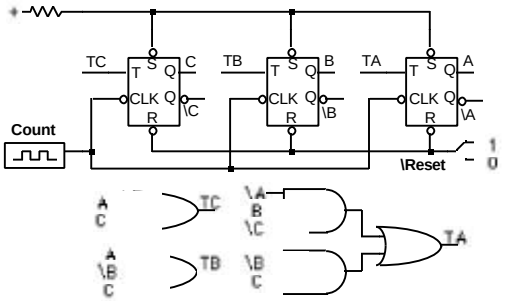
TB = A + $\bar{B}$ + C

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

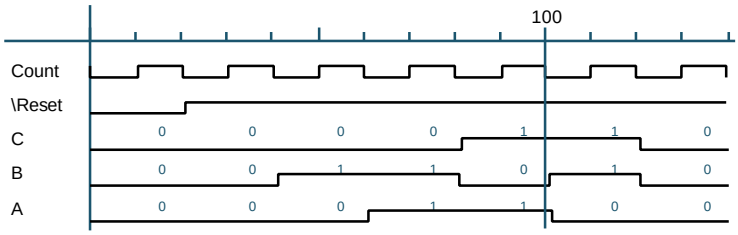
Resulting Logic:

5 Gates

13 Input Literals +
Flipflop connections



Timing Waveform:



No. 7-25

Self-Starting Counters

Start-Up States

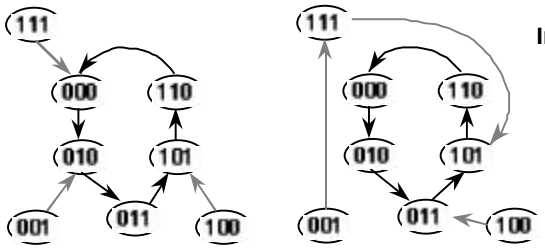
At power-up, counter may be in possible state

Designer must guarantee that it (eventually) enters a valid state

Especially a problem for counters that validly use a subset of states

Self-Starting Solution:

Design counter so that even the invalid states eventually transition to valid state



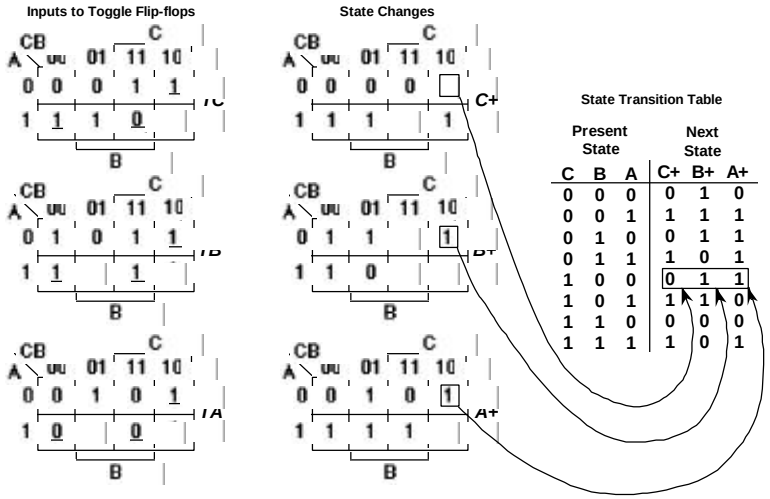
Implementation in Previous Slide!

Two Self-Starting State Transition Diagrams for the Example Counter

No. 7-26

Self-Starting Counters

Deriving State Transition Table from Don't Care Assignment



State Transition Table

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

Implementation with Different Kinds of FFs

R-S Flipflops

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

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

$Q+ = S + R\,\overline{Q}$

RS Exitation Table

Present State			Next State			Remapped Next State					
C	B	A	C+	B+	A+	R	SC	R	SB	R	SA
0	0	0	0	1	0						
0	0	1	X	0	0						
0	1	0	0	1	1						
0	1	1	1	0	1						
1	0	0	X	0	0						
1	0	1	1	1	0						
1	1	0	0	0	0						
1	1	1	X	0	0						

Remapped Next State Functions

Implementation with Different Kinds of FFs

R-S Flipflops

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

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

$Q+ = S + R\,\overline{Q}$

RS Exitation Table

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

Remapped Next State Functions

Implementation with Different Kinds of FFs

RS FFs Continued

A \ CB	00	01	11	10
0				
1				

RC

A \ CB	00	01	11	10
0				
1				

SC

A \ CB	00	01	11	10
0				
1				

RB

A \ CB	00	01	11	10
0				
1				

SB

A \ CB	00	01	11	10
0				
1				

RA

A \ CB	00	01	11	10
0				
1				

SA

RC =

SC =

RB =

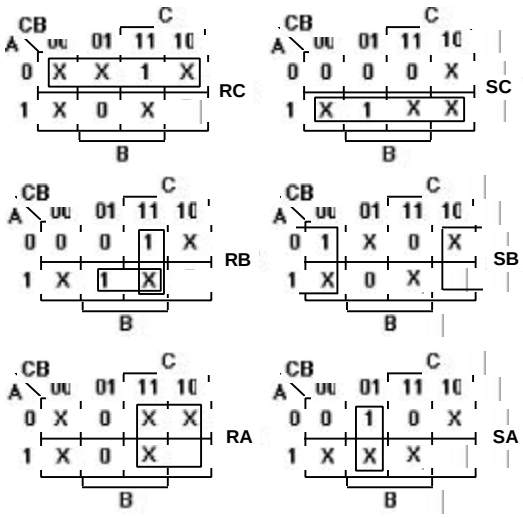
SB =

RA =

SA =

Implementation with Different Kinds of FFs

RS FFs Continued



$RC = \overline{A}$

$SC = A$

$RB = A\,B + B\,C$

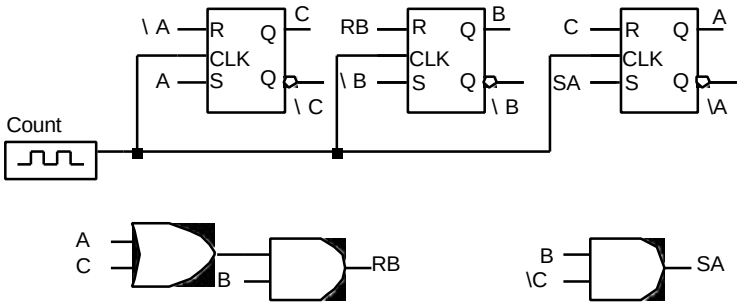
$SB = \overline{B}$

$RA = C$

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

Implementation With Different Kinds of FFs

RS FFs Continued



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

Implementation with Different FF Types

J-K FFs

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

$Q+ = J\,\overline{Q} + \overline{K}\,Q$

J-K Excitation Table

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

Remapped Next State Functions

Implementation with Different FF Types

J-K FFs

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

$Q+ = J\,\overline{Q} + \overline{K}\,Q$

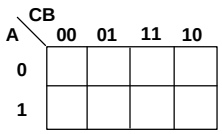
J-K Excitation Table

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

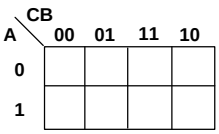
Remapped Next State Functions

Implementation with Different FF Types

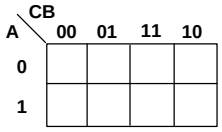
J-K FFs Continued



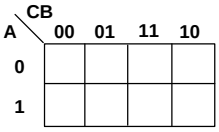
JC



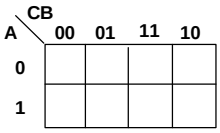
KC



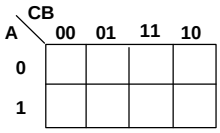
JB



KB



JA



KA

JC =

KC =

JB =

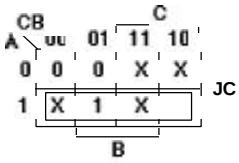
KB =

JA =

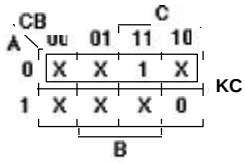
KA =

Implementation with Different FF Types

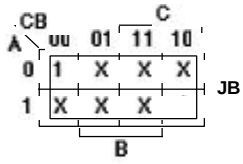
J-K FFs Continued



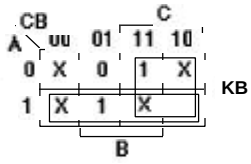
JC



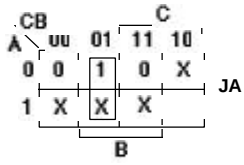
KC



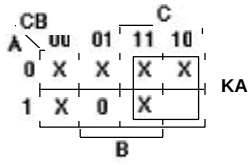
JB



KB



JA



KA

JC = A

KC = $\overline{A}$

JB = 1

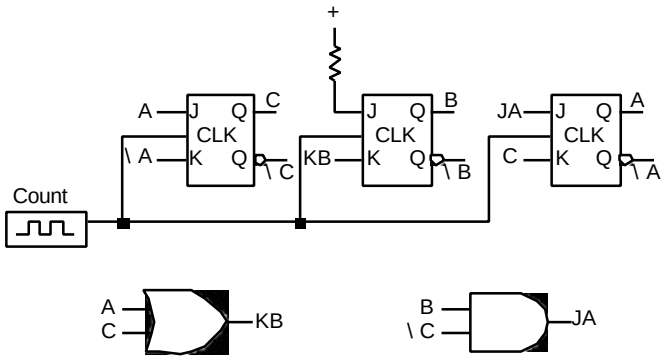
KB = A + C

JA = B $\overline{C}$

KA = C

Implementation with Different FF Types

J-K FFs Continued



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

No. 7-37

Implementation with Different FF Types

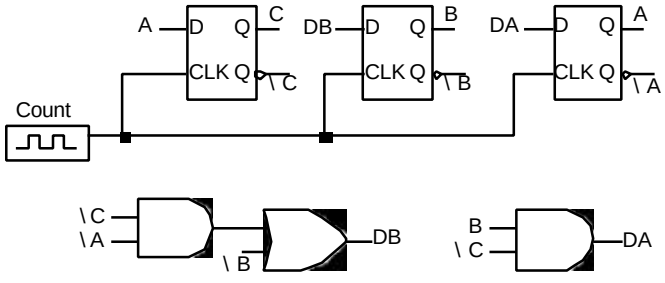
D FFs

Simplest Design Procedure: No remapping needed!

$$DC = A$$

$$DB = \overline{A}\,\overline{C} + \overline{B}$$

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



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

No. 7-38

Implementation with Different FF Types

Comparison

- **T FFs well suited for straightforward binary counters**

  But yielded worst gate and literal count for this example!

- **No reason to choose R-S over J-K FFs: it is a proper subset of J-K**

  R-S FFs don't really exist anyway

  J-K 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

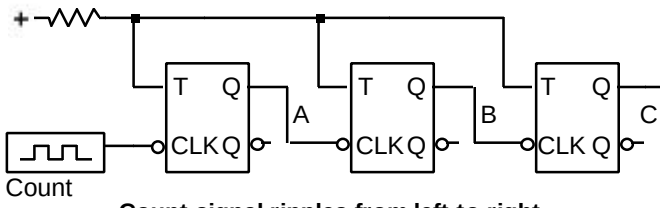
  D storage devices very transistor efficient in VLSI

  Best choice where area/literal count is the key

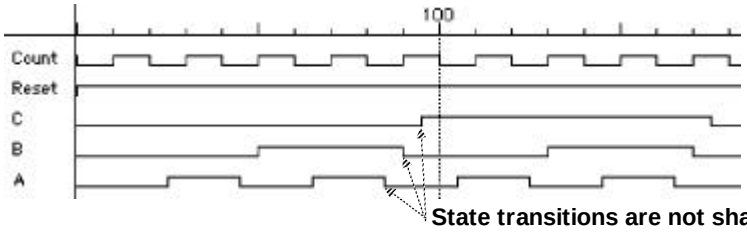
Asynchronous vs. Synchronous Counters

Ripple Counters

Deceptively attractive alternative to synchronous design style



Count signal ripples from left to right

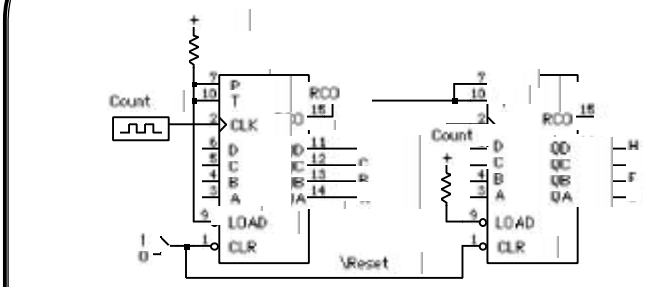


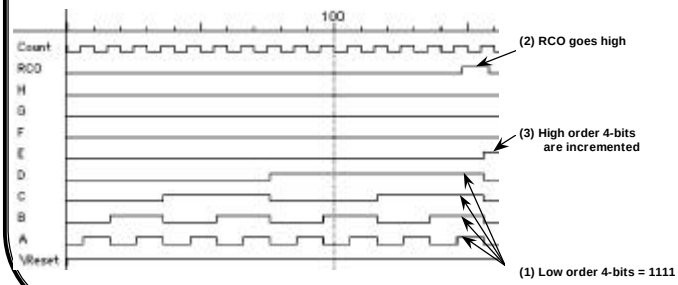
State transitions are not sharp!

Can lead to "spiked outputs" from combinational logic decoding the counter's state

Asynchronous vs. Synchronous Counters

Cascaded Synchronous Counters with Ripple Carry Outputs





First stage RCO enables second stage for counting

RCO asserted soon after stage enters state 1111

also a function of the T Enable

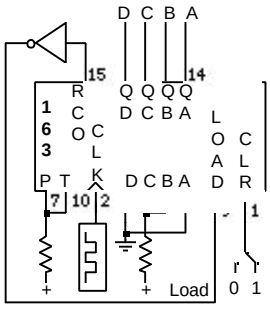
Downstream stages lag in their 1111 to 0000 transitions

Affects Count period and decoding logic

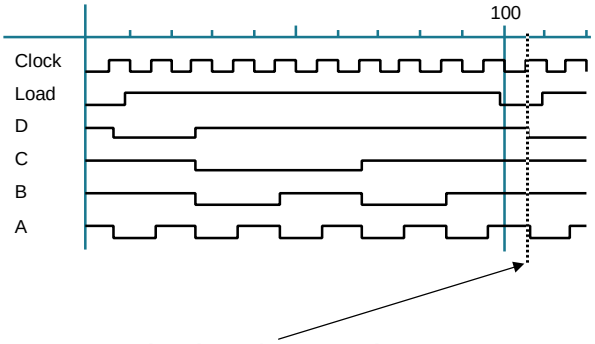
Asynchronous vs. Synchronous Counters

The Power of Synchronous Clear and Load

Starting Offset Counters:
e.g., 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1111, 0110, …



0110 is the state to be loaded



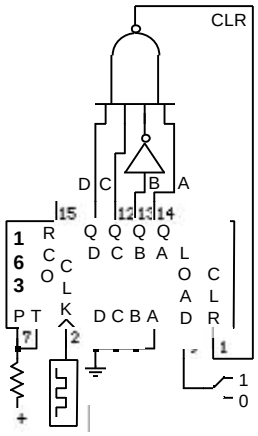
Use RCO signal to trigger Load of a new state

Since 74163 Load is synchronous, state changes only on the next rising clock edge

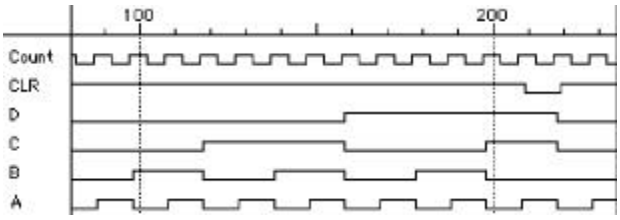
Asynchronous vs. Synchronous Counters

Offset Counters Continued

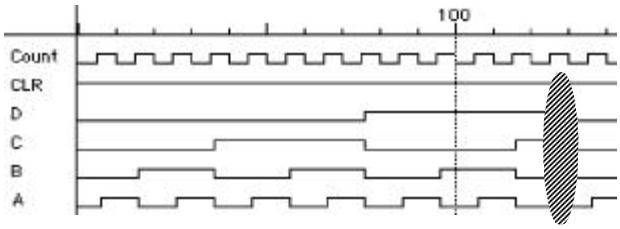
Ending Offset Counter:
e.g., 0000, 0001, 0010, …, 1100, 1101, 0000



Decode state to determine when to reset to 0000



Clear signal takes effect on the rising count edge



Replace '163 with '161, Counter with Async Clear
Clear takes effect immediately!