Motivation

- Flipflops: most primitive "packaged" sequential circuits
- More complex sequential building blocks:

  storage registers, shift registers, counters
  available as components in the TTL catalog

- How to represent and design simple sequential circuits: counters
- Problems and pitfalls when working with counters:

  start-up states
  asynchronous vs. synchronous logic

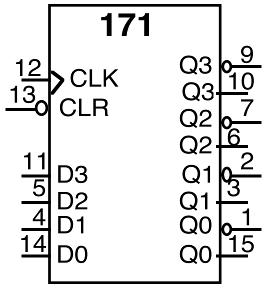
Chapter overview

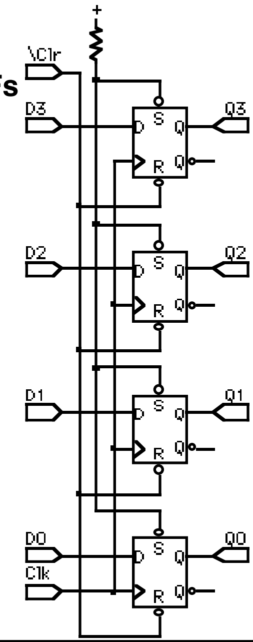
- Examine real sequential logic circuits available as components
  - registers for storage and shifting
  - random access memories
  - counters
- Counter design procedure
  - simple but useful finite state machine
  - state diagram, state transition table, next state functions
  - excitation tables for implementation with alternative flip-flop types
- Synchronous vs. asynchronous counters
  - ripple vs. synchronous counters
  - asynchronous vs. synchronous clears and loads

Kinds of registers and counters

- Storage register – group of storage elements read/written as a unit

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



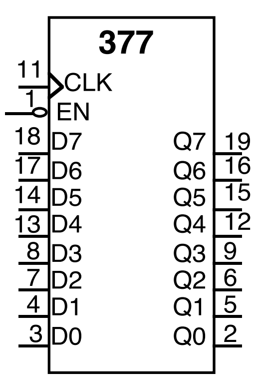


TTL 74171 quad D-FF with clear
(small numbers represent pin #s on package)

Kinds of registers and counters

- Input/output variations
  - selective load capability
  - tri-state or open collector outputs
  - true and complementary outputs

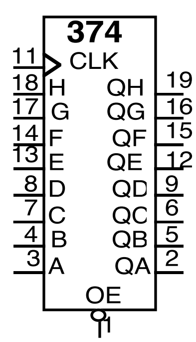
74377 octal D-type FFs
with input enable



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

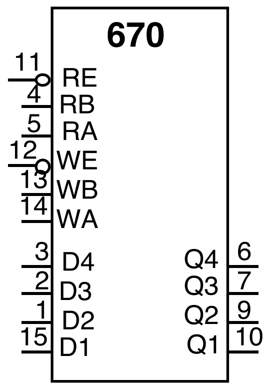
OE asserted low
presents FF state to
output pins; otherwise
high impedence

74374 octal D-type FFs
with output enable



Kinds of registers and counters

- **Register files**
  - 2-dimensional array of flip-flops
  - address used as index to a particular word
  - word contents read or written



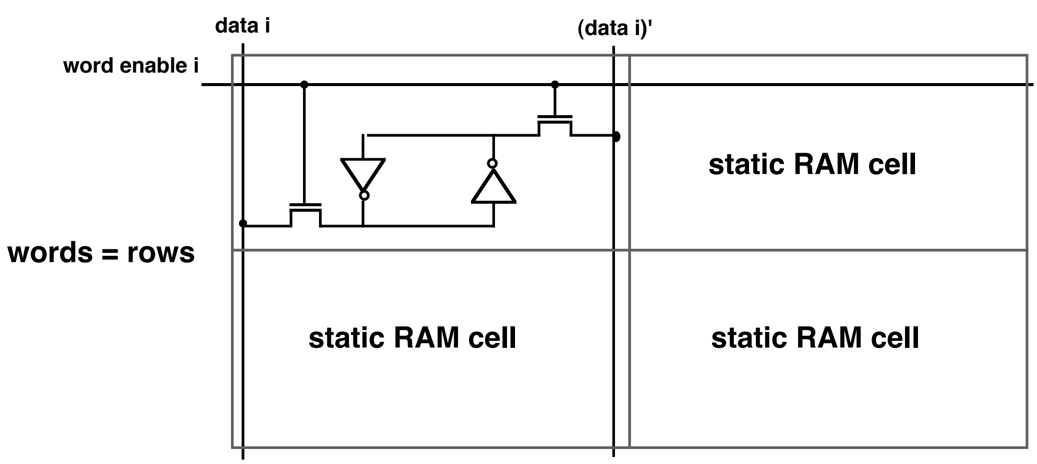
74670 4x4 register file with tri-state outputs

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)

Kinds of registers and counters

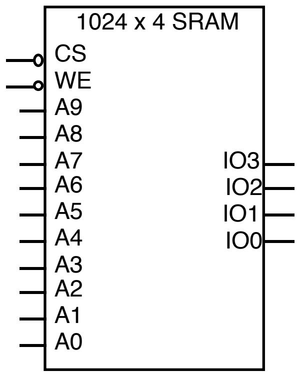
- **Random access memories**
  - transistor efficient methods for implementing storage elements
  - small RAM: 256 words by 4-bit
  - large RAM: 4 million words by 1-bit



columns = bits (double-rail encoded)

Kinds of registers and counters

- RAM organization – for a 1024 by 4-bit part



chip select line (active low)

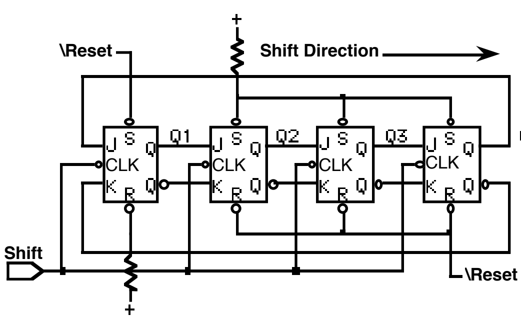
write enable line (active low)

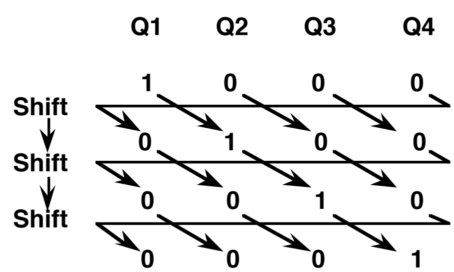
10 address lines

4 bi-directional data lines

Kinds of registers and counters

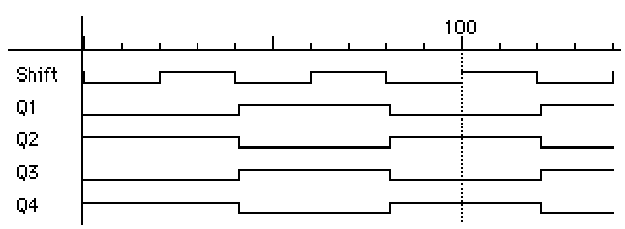
- Shift registers – storage + ability to circulate data among storage elements





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

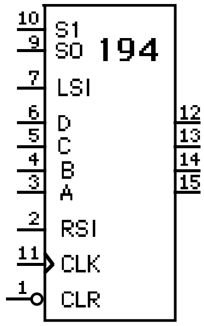
wrap around from rightmost element to leftmost element



master slave FFs: sample inputs while clock is high; change outputs on falling edge

Kinds of registers and counters

- **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

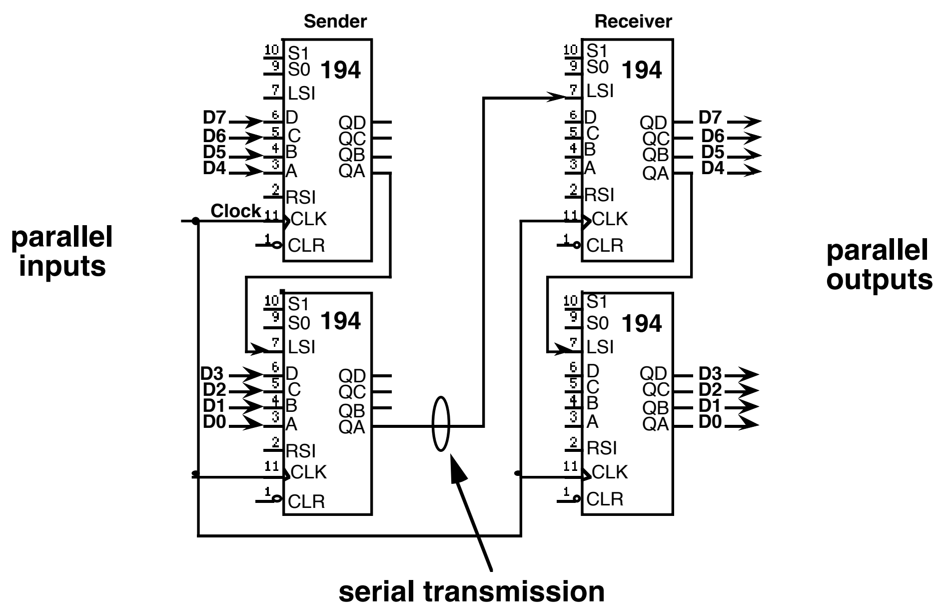
- S1 = 1, S0 = 1: load on rising clk edge
  - synchronous load
- S1 = 1, S0 = 0: shift left on rising clk edge
  - LSI replaces element D
- S1 = 0, S0 = 1: shift right on rising clk edge
  - RSI replaces element A
- S1 = 0, S0 = 0: hold state

view it as multiplexing logic on input to each FF!

shifters are well suited for serial-to-parallel conversions such as terminal to computer communications

Kinds of registers and counters

- **Shift register application – parallel to serial conversion**



Kinds of registers and counters

- Counters – proceed through a well-defined sequence of states
  - in response to a count signal

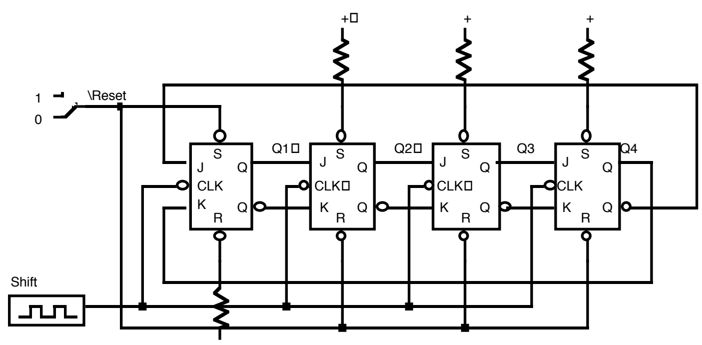
- Examples
  - 3-bit up-counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...
  - 3-bit down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...
  - binary vs. BCD vs. Gray-code counters

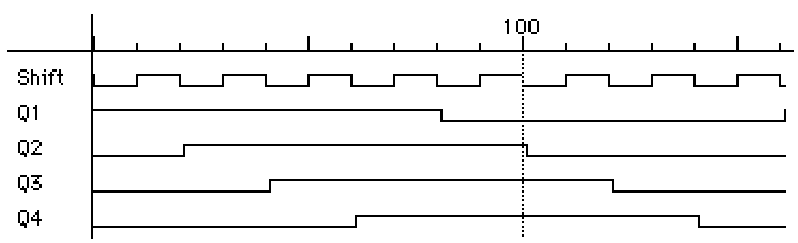
- A counter is a "degenerate" finite state machine (sequential circuit)
  - where the state is the only output

Kinds of registers and counters

- Johnson (Mobius) counter

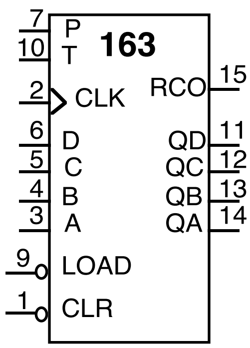
8 possible states
single-bit change per state
useful for avoiding glitches





Kinds of registers and counters

- '163 – TTL catalog counter



74163 synchronous 4-bit up-counter

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 (internal AND gate)

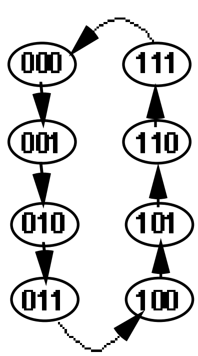
RCO: ripple-carry out – asserted when counter enters its highest state 1111, used for cascading counters

74161: similar in function but with asynchronous load and reset

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



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

State Transition Diagram

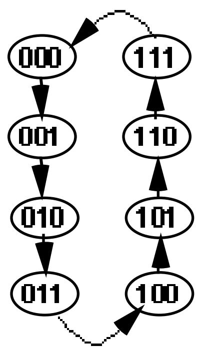
implement with toggle FFs

what inputs must be presented to the T-FFs to get them to change to the desired state bit?

this is called "re-mapping the next state function"

Counter design procedure

- Example: 3-bit binary up-counter
  - □ implement with toggle FFs
  - □ what inputs must be presented to the T-FFs to get them to change
  - □ □ □ to the desired state bit?
  - □ this is called <u>re-mapping the next state function</u>



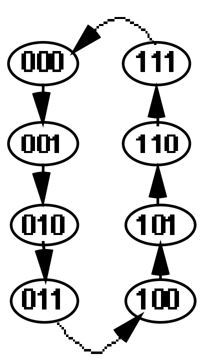
state transition diagram

Present State			Next State			FF inputs		
C	B	A	C+	B+	A+	TC	TB	TA
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

Counter design procedure

- Example: 3-bit binary up-counter
  - □ implement with toggle FFs
  - □ what inputs must be presented to the T-FFs to get them to change
  - □ □ □ to the desired state bit?
  - □ this is called re-mapping the next state function



state transition diagram

Present State			Next State			FF 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

state transition table

Counter design procedure

- **Example (continued)**

A \ CB	00	01	11	10
0	1	1	1	1
1	1	1	1	1

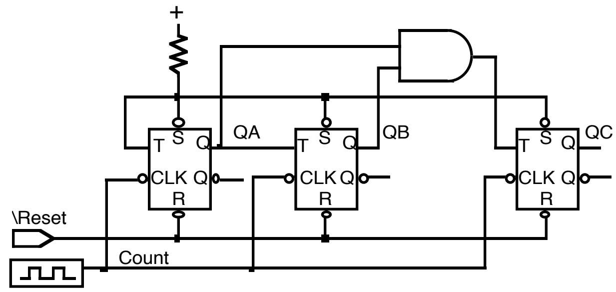
$TA = 1$

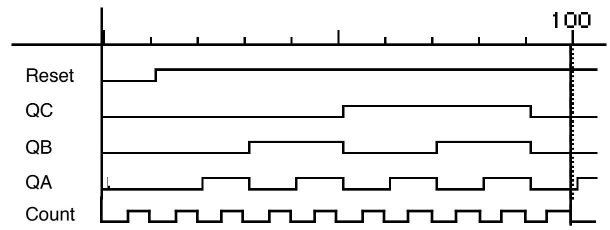
A \ CB	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$TB = A$

A \ CB	00	01	11	10
0	0	0	0	0
1	0	1	1	0

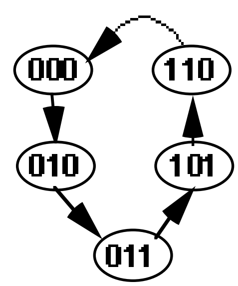
$TC = A \bullet B$





Counter design procedure

- **More complex count sequence example**



- **Step 1: derive the state transition diagram**
  count sequence: 000, 010, 011, 101, 110

- **Step 2: derive the state transition table**
  from the state transition diagram

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

note the don't care conditions that arise from unused states

Counter design procedure

- More complex count sequence example (cont'd)

- Step 3: K-maps for next state functions
  derive input equations for D-FFs

C+

		B	B, C	C
	0	0	0	X
A	X	1	X	1

B+

		B	B, C	C
	1	1	0	X
A	X	0	X	1

A+

		B	B, C	C
	0	1	0	X
A	X	1	X	0

Counter design procedure

- More complex count sequence example (cont'd)

- Step 4: choose FF type for implementation (toggle, T, in this case)
  use excitation table to re-map next state functions

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

toggle excitation table

Present State			Next State			FF inputs		
C	B	A	C+	B+	A+	TC	TB	TA
0	0	0	0	1	0	0	1	0
0	0	1	–	–	–	–	–	–
0	1	0	0	1	1	0	0	1
0	1	1	1	0	1	1	1	0
1	0	0	–	–	–	–	–	–
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	–	–	–	–	–	–

re-mapped next state functions

Counter design procedure

- More complex count sequence example (cont'd)
- Step 5:□K-maps to minimize FF input equations

TC

C

	0□	0□	1□	X
A	X□	1□	X□	0

B

TB

C

	1□	0□	1□	X
A	X□	1□	X□	1

B

TA

C

	0□	1□	0□	X
A	X□	0□	X□	1

B

TC = AC' + A'C = A xor C

TB = A + B' + C

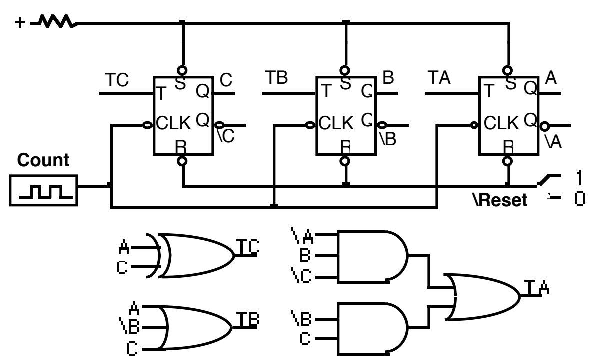
TA = A'BC' + B'C

Counter design procedure

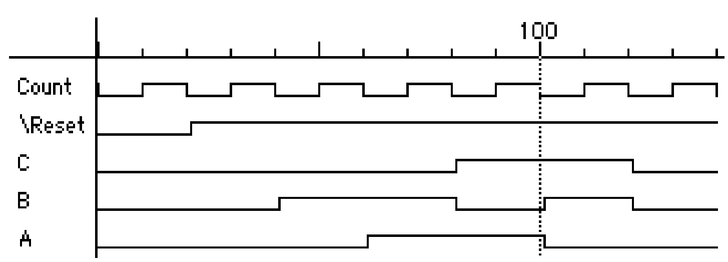
- More complex count sequence example (cont'd)

resulting logic:

□ 5 gates
□ 13 input literals
□ + FF connections



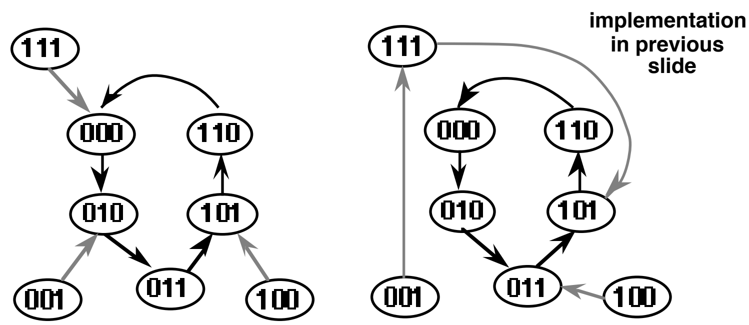
timing waveform:



Self-starting counters

- Start-up states
  - at power-up, counter may be in an used or invalid 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 all the invalid states eventually transition
  - to a valid state

two self-starting state transition diagrams for the example counter



Self-starting counters

- Deriving state transition table from don't care assignment

TC

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

C+

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

TB

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

B+

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

TA

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

A+

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

Present State C	B	A	Next State 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 FF types

- 5-state counter example with R-S FFs

Q	Q+	R	S
0	0	–	0
0	1	0	1
1	0	1	0
1	1	0	–

Q+ = S + R'Q
RS excitation table

Present State			Next State			Re-mapped Next State					
C	B	A	C+	B+	A+	RC	SC	RB	SB	RA	SA
0	0	0	0	1	0	–	0	0	1	–	0
0	0	1	–	–	–	–	–	–	–	–	–
0	1	0	0	1	1	–	0	0	–	0	1
0	1	1	1	0	1	0	1	1	0	0	–
1	0	0	–	–	–	–	–	–	–	–	–
1	0	1	1	1	0	0	–	0	1	1	0
1	1	0	0	0	0	1	0	1	0	–	0
1	1	1	–	–	–	–	–	–	–	–	–

re-mapped next state functions

Implementation with different FF types

- 5-state counter example with R-S FFs (cont'd)

RC

C

	X	X	1	X
A	X	0	X	0

B

RB

C

	0	0	1	X
A	X	1	X	0

B

RA

C

	X	0	X	X
A	X	0	X	1

B

SC

C

	0	0	0	X
A	X	1	X	X

B

SB

C

	1	X	0	X
A	X	0	X	1

B

SA

C

	0	1	0	X
A	X	X	X	0

B

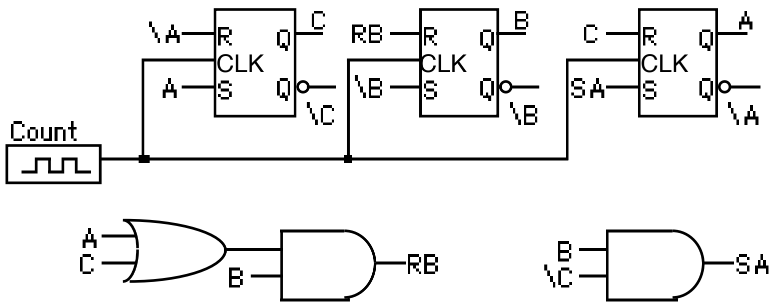
RC = A' **SC = A**

RB = AB + BC **SB = B'**

RA = C **SA = BC'**

Implementation with different FF types

- **5-state counter example with R-S FFs (cont'd)**



resulting logic level implementation:
3 gates, 11 input literals + FF connections

Implementation with different FF types

- **5-state counter example with J-K FFs**

Q	Q+	R	S
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

Q+ = JQ' + K'Q
JK excitation table

Present State			Next State			Re-mapped Next State					
C	B	A	C+	B+	A+	JC	KC	JB	KB	JA	KA
0	0	0	0	1	0	0	–	1	–	0	–
0	0	1	–	–	–	–	–	–	–	–	–
0	1	0	0	1	1	0	–	–	0	1	–
0	1	1	1	0	1	1	–	–	1	–	0
1	0	0	–	–	–	–	–	–	–	–	–
1	0	1	1	1	0	–	0	1	–	–	1
1	1	0	0	0	0	–	1	–	1	0	–
1	1	1	–	–	–	–	–	–	–	–	–

re-mapped next state functions

Implementation with different FF types

• 5-state counter example with J-K FFs (cont'd)

JC

	0□	0□	X□	X
A	X□	1□	X□	X

JB

	1□	X□	X□	X
A	X□	X□	X□	1

JA

	0□	1□	0□	X
A	X□	X□	X□	X

KC

	X□	X□	1□	X
A	X□	X□	X□	0

KB

	X□	0□	1□	X
A	X□	1□	X□	X

KA

	X□	X□	X□	X
A	X□	0□	X□	1

(C over the third and fourth columns, B over the second and third columns of each map)

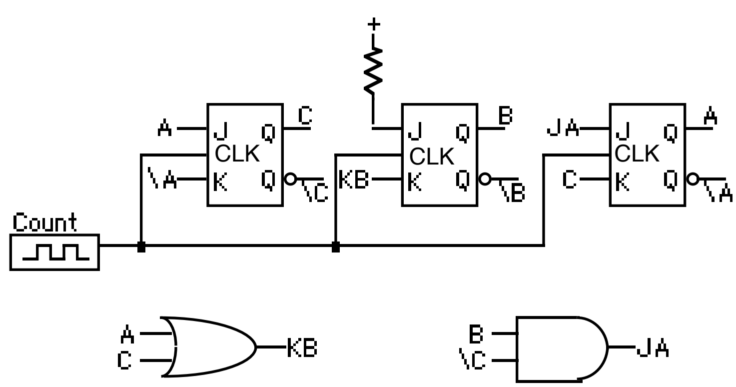
JC = A□ KC = A'

JB = 1□ KB = A + C

JA = BC'□ KA = C

Implementation with different FF types

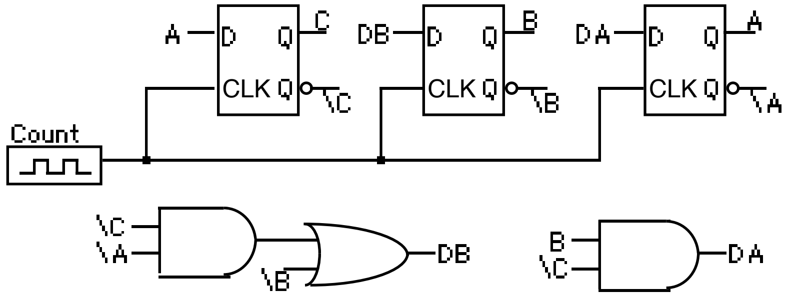
• 5-state counter example with J-K FFs (cont'd)



resulting logic level implementation:
2 gates, 10 input literals + FF connections

Implementation with different FF types

- 5-state counter example with D FFs
  - simplest design procedure – no re-mapping needed



DC = A

DB = A'C' + B'

DA = BC'

resulting logic level implementation:
3 gates, 8 input literals + FF connections

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

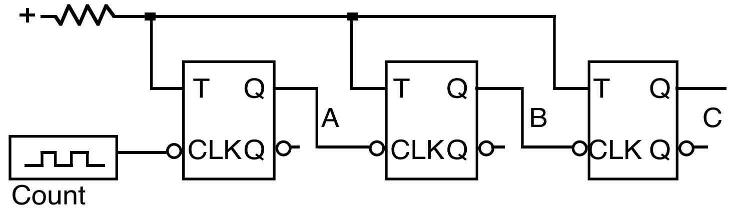
- D FFs have simplest design procedure
  - best literal count
  - D storage devices very transistor efficient in implementation
  - best choice where area/literal count is the key

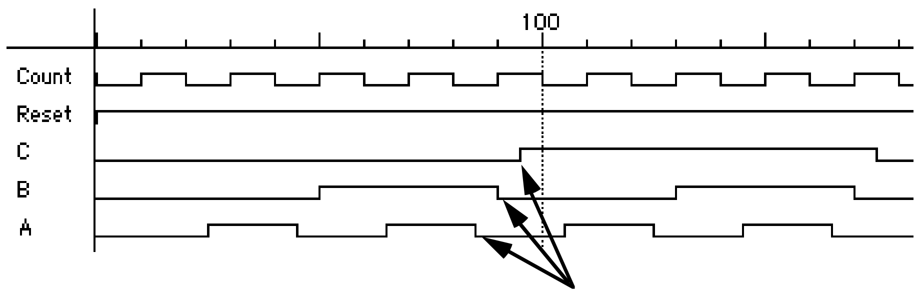
- Which is the best for PALs?

Asynchronous vs. synchronous counters

- **Ripple counters**
  - □ **deceptively attractive alternative to synchronous design style**

count signal ripples from left to right

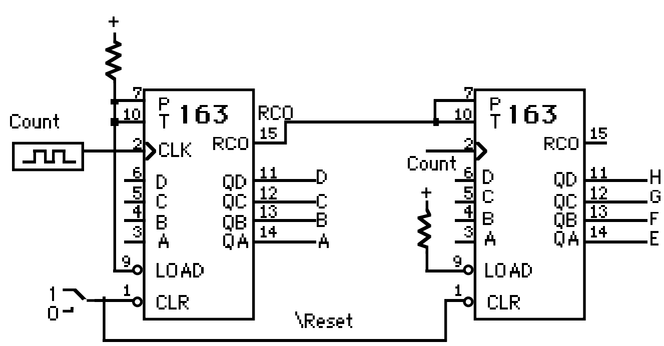




state transitions are not sharp – can lead to "spiked outputs" from combinational logic decoding the counter's state

Asynchronous vs. synchronous counters

- **Cascaded synchronous counters with ripple carry outputs**



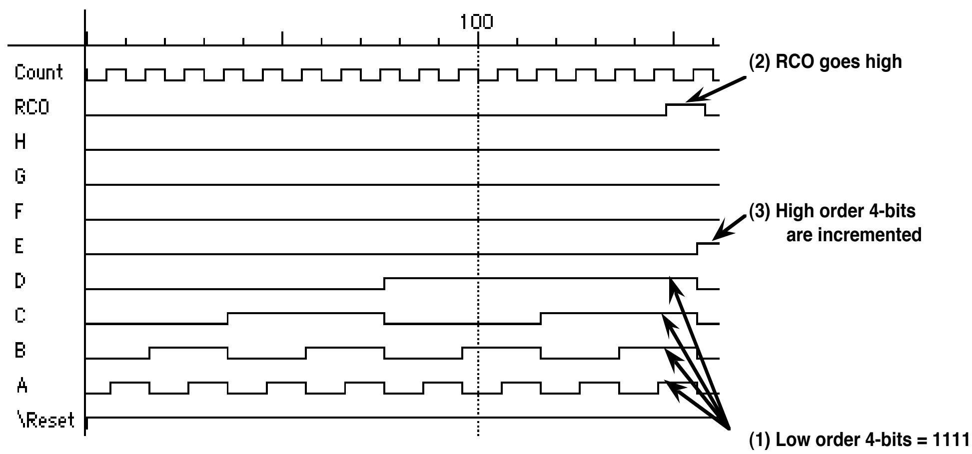
first stage RCO enables second stage for counting

RCO asserted soon after stage enters state 1111

also a function of the T enable so they can be cascaded to form larger counters – downstream stages lag in their 1111 to 0000 transitions

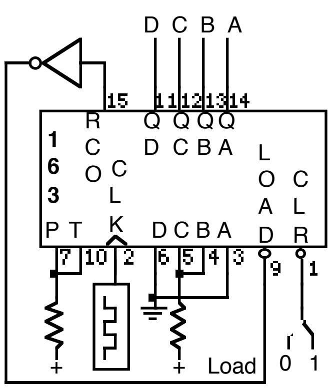
Asynchronous vs. synchronous counters

- Cascaded synchronous counters with ripple carry outputs (cont'd)

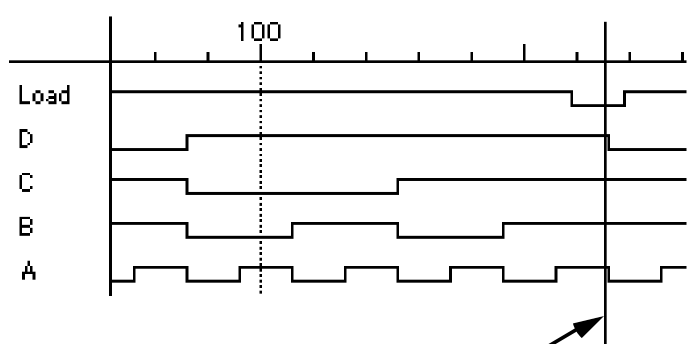


Asynchronous vs. synchronous counters

- Offset counters – the use of synchronous clear and load
  - e.g., 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1111, 0110, . . .



0110
is the state
to be loaded

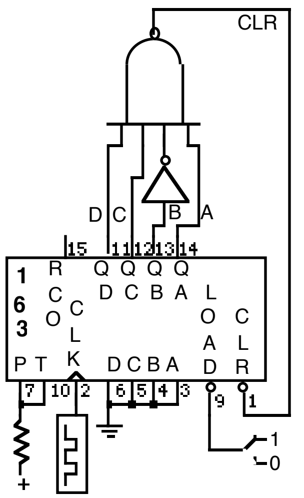


use RCO signal to trigger Load of a new state

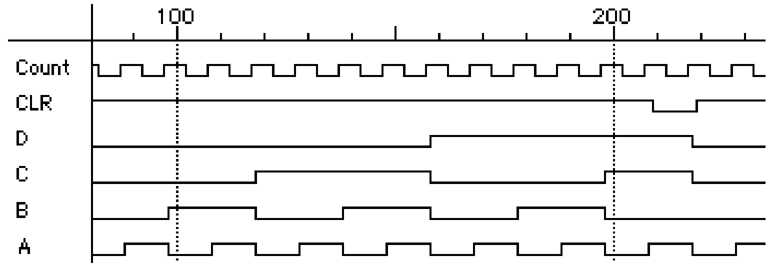
since 74163 load is synchronous, state changes only on the next rising clock edge

Asynchronous vs. synchronous counters

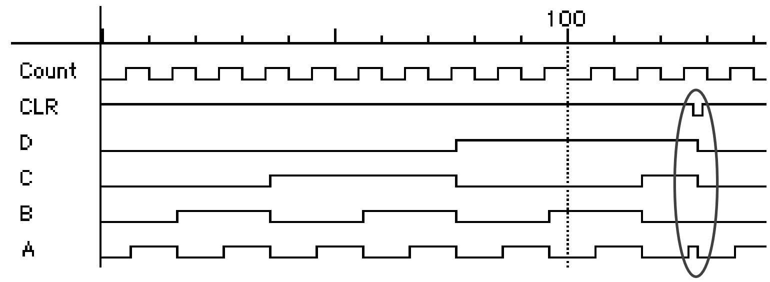
- **Ending offset counter – e.g., 0000, 0001, 0010, ..., 1100, 1101, 0000**



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

Chapter review

- **Sequential circuit packages**
  - registers, shifters, counters, RAMs
- **Counters as simple finite state machines – design procedure**
  - 1. derive state diagram
  - 2. derive state transition table
  - 3. determine next state functions
  - 4. re-map next state for target FF types using excitation tables
  - 5. implement logic
- **Different FF types in counters**
  - J-K best for reducing gate count in packaged logic
  - D is easiest design plus best for reducing wiring and for PALs
- **Asynchronous vs. synchronous counters**
  - avoid ripple counters – state transitions are not sharp
  - beware of potential problems when cascading synchronous counters
  - offset counters: easy to design with synchronous load and clear
  - extreme caution in using asynchronous load and clear