Chapter #7: Sequential Logic Case Studies

Contemporary Logic Design

Randy H. Katz
University of California, Berkeley

June 1993

Motivation

Contemporary Logic Design
Sequential Case Studies

- ***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

Contemporary Logic Design
Sequential Case Studies

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 flipflop types**

Synchronous vs. Asynchronous Counters

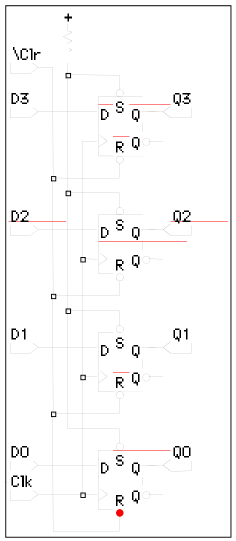
- **Ripple vs. Synchronous Counters**
- **Asynchronous vs. Synchronous Clears and Loads**

Kinds of Registers and Counters

Contemporary Logic Design
Sequential Case Studies

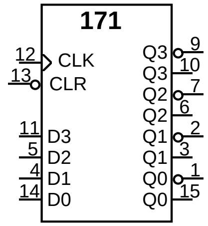
Storage Register

Group of storage elements read/written as a unit



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

Schematic Shape

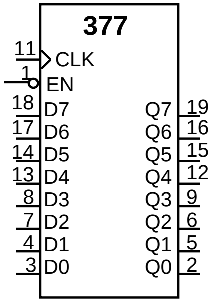


TTL 74171 Quad D-type FF with Clear
(Small numbers represent pin #s on package)

Kinds of Registers and Counters

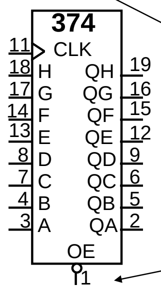
Input/Output Variations

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



74374 Octal D-type FFs with output enable

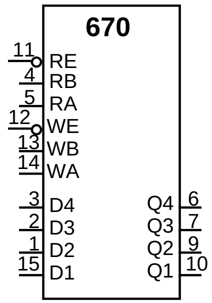
OE asserted low presents FF state to output pins; otherwise high impedence

Contemporary Logic Design
Sequential Case Studies

Kinds of Registers and Counters

Register Files

Two dimensional array of flipflops
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

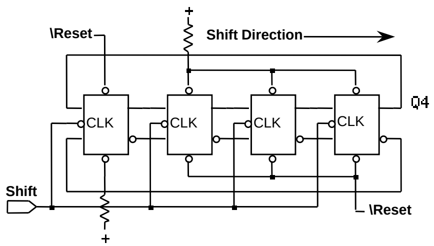
Contains 16 D-ffs, organized as four rows (words) of four elements (bits)

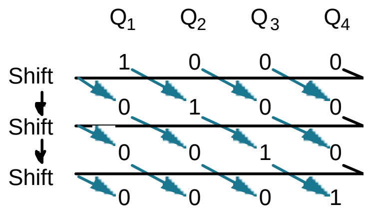
Contemporary Logic Design
Sequential Case Studies

Kinds of Registers and Counters

Shift Registers

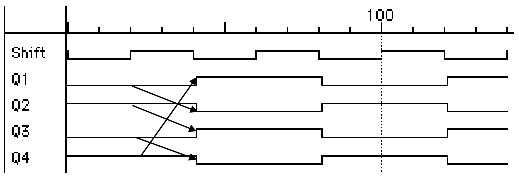
Storage + ability to circulate data among storage elements





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



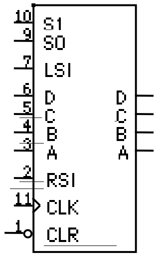
Master Slave FFs: sample inputs while clock is high; change outputs on falling edge

Contemporary Logic Design
Sequential Case Studies

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

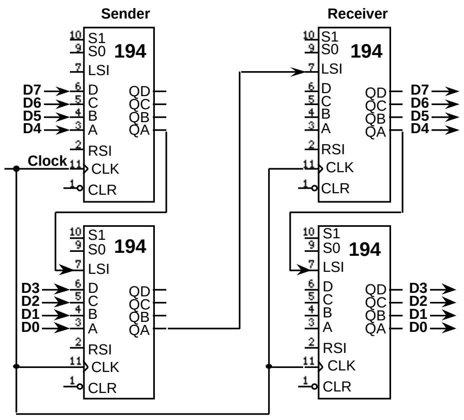
Multiplexing logic on input to each FF!

Shifters 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

Parallel Inputs



Parallel Outputs

Serial transmission

Kinds of Registers and Counters

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, ...

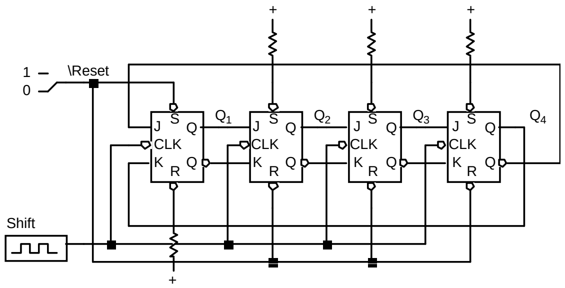
3 Bit Down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...

Binary vs. BCD vs. Gray Code Counters

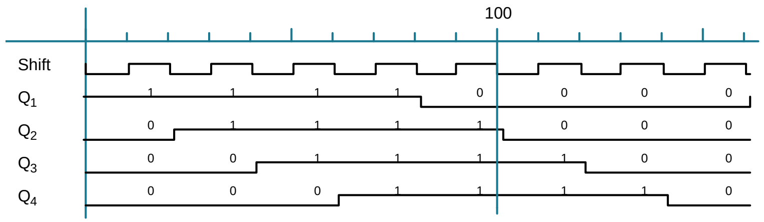
A counter is a "degenerate" finite state machine/sequential circuit where the state *is* the only output

Kinds of Registers and Counters

Johnson (Mobius) Counter



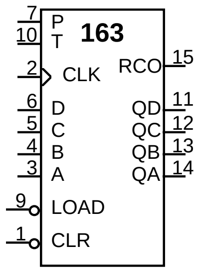
End-Around



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

Kinds of Registers and Counters

Catalog Counter



74163 Synchronous 4-Bit Upcounter

Synchronous Load and Clear Inputs

Positive Edge Triggered FFs

Parallel Load Data from D, C, B, A

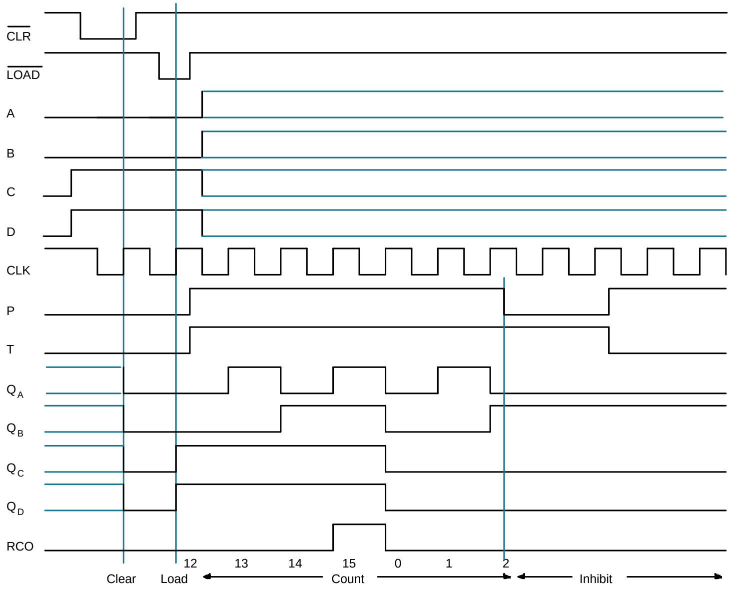
P, T Enable Inputs: both must be asserted to enable counting

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

74161: similar in function, asynchronous load and reset

Kinds of Registers and Counters

74163 Detailed Timing Diagram



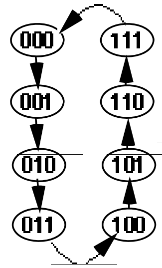
Counter Design Procedure

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



Present State C	B	A	*Next State* C+	A	*Flipflop Inputs* 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"

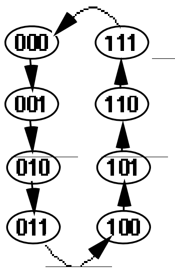
Counter Design Procedure

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



Present State C	B	A	*Next State* C+	A	*Flipflop Inputs* 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

Example Continued

K-maps for Toggle Inputs:

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 =

Resulting Logic Circuit:

Counter Design Procedure

Example Continued

K-maps for Toggle Inputs:

TA = 1

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

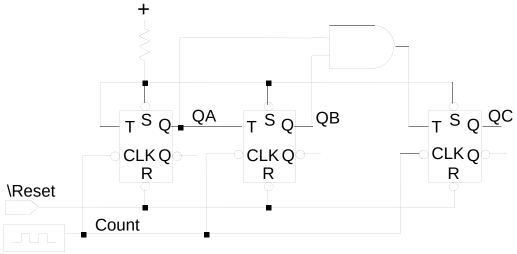
TB = A

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

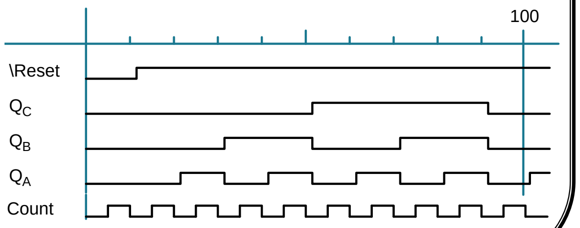
TC = A• B

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

Resulting Logic Circuit:



Timing Diagram:

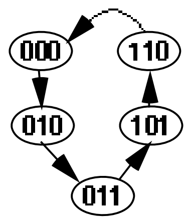


Counter Design Procedure

More Complex Count Sequence

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

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



Step 2: **State Transition Table**

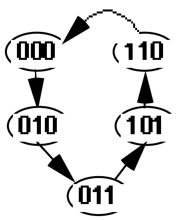
Present State C	B	A	Next State C+	B+	A+
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Counter Design Procedure

More Complex Count Sequence

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

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



Step 2: **State Transition Table**

Present State C	B	A	Next State 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

Note the Don't Care conditions

Counter Design Procedure

More Complex Count Sequence

Step 3: **K-Maps for Next State Functions**

C+

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

B+

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

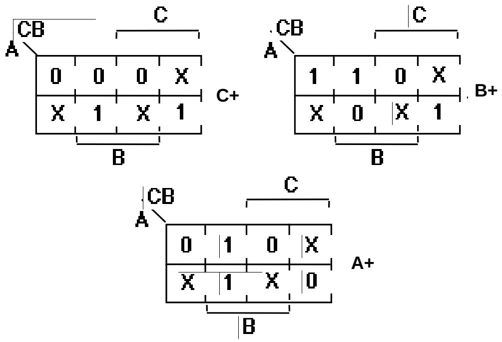
A+

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

Counter Design Procedure

More Complex Count Sequence

Step 3: K-Maps for 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	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

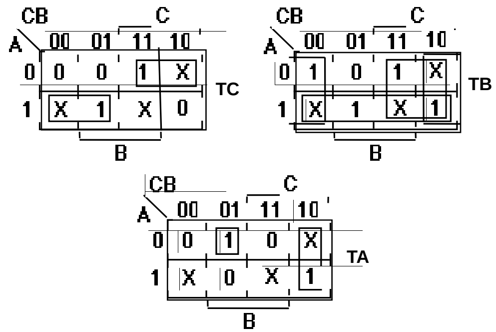
Present State			Toggle Inputs		
C	B	A	TC	TB	TA
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	X	X	X
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	X	X	X

Remapped Next State Functions

Counter Design Procedure

More Complex CounterSequencing

Remapped K-Maps



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

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

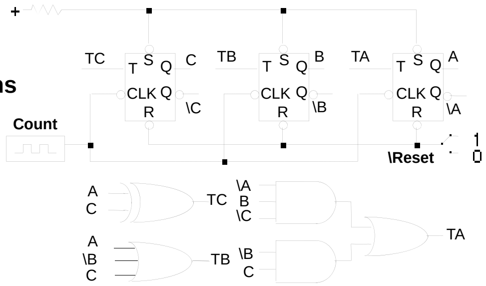
TA = $\bar{A}\,B\,\bar{C} + \bar{B}\,C$

Counter Design Procedure

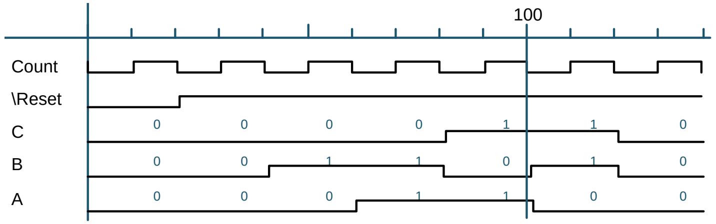
More Complex Counter Sequencing

Resulting Logic:

5 Gates
13 Input Literals +
Flipflop connections



Timing Waveform:



Self-Starting Counters

Start-Up States

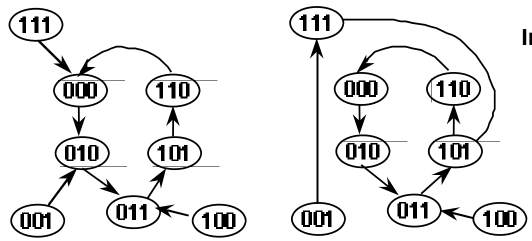
At power-up, counter may be in possible state

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

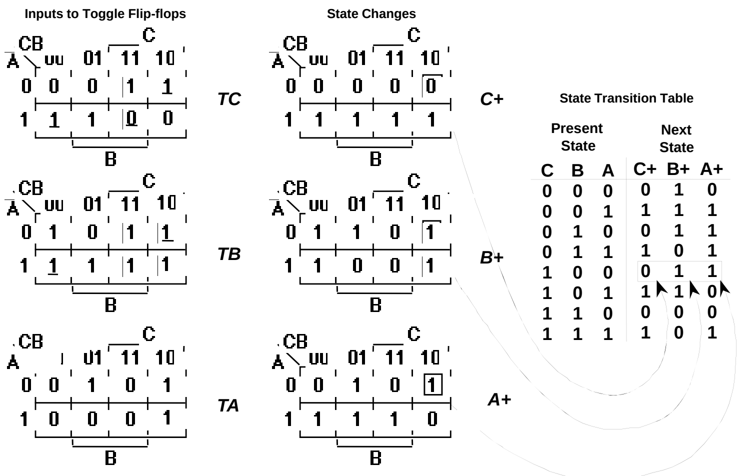


Implementation in Previous Slide!

Two Self-Starting State Transition Diagrams for the Example Counter

Self-Starting Counters

Deriving State Transition Table from Don't Care Assignment



State Transition Table

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 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

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

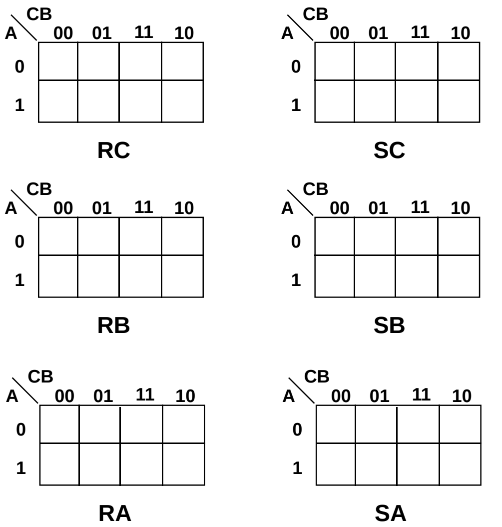
RS Exitation Table

Present State C	B	A	Next State C+	B+	A+	Remapped Next State 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



RC =

SC =

RB =

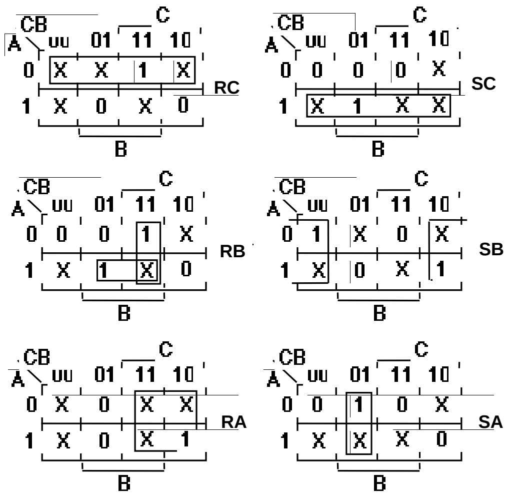
SB =

RA =

SA =

Implementation with Different Kinds of FFs

RS FFs Continued



RC = $\overline{A}$

SC = A

RB = A B + B C

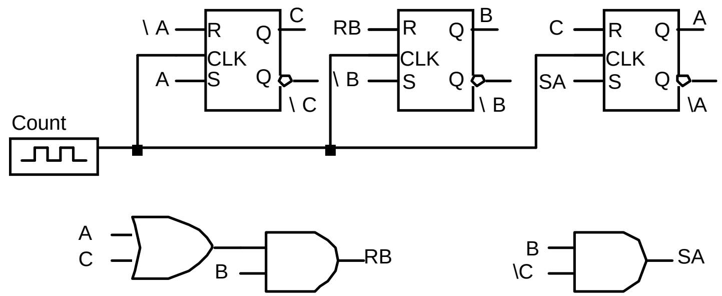
SB = $\overline{B}$

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

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

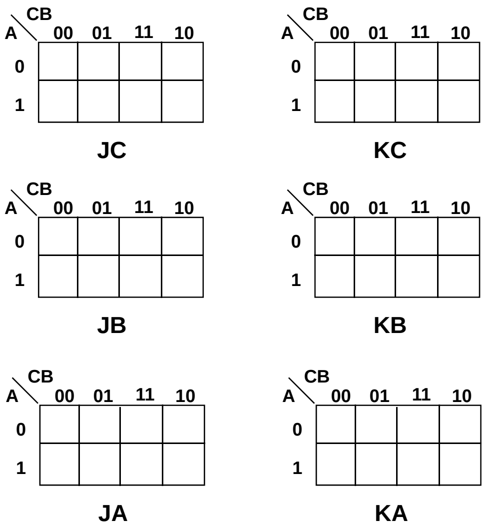
J-K Excitation Table

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

Remapped Next State Functions

Implementation with Different FF Types

J-K FFs Continued



JC =

KC =

JB =

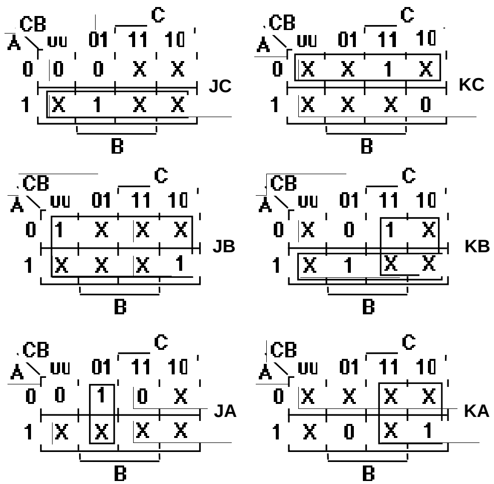
KB =

JA =

KA =

Implementation with Different FF Types

J-K FFs Continued



JC = A

KC = $\overline{A}$

JB = 1

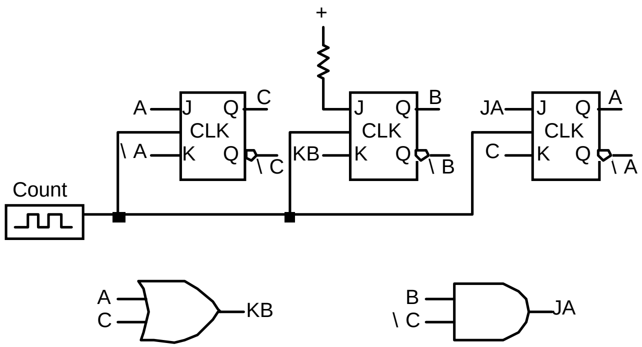
KB = A + C

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

Implementation with Different FF Types

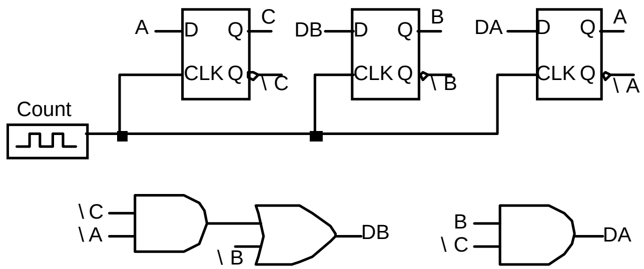
D FFs

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

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

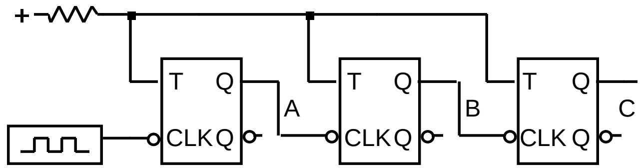
  D storage devices very transistor efficient in VLSI

  Best choice where area/literal count is the key

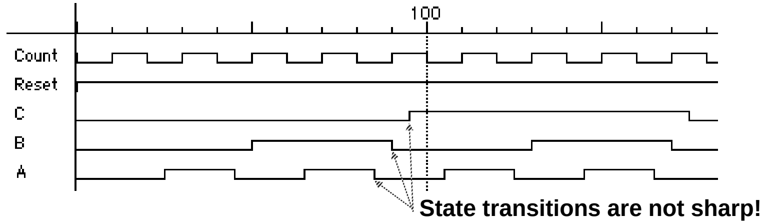
Asynchronous vs. Synchronous Counters

Ripple Counters

Deceptively attractive alternative to synchronous design style



Count signal ripples from left to right

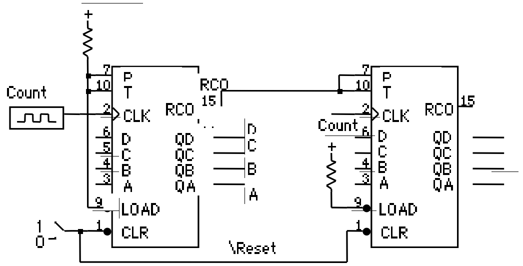


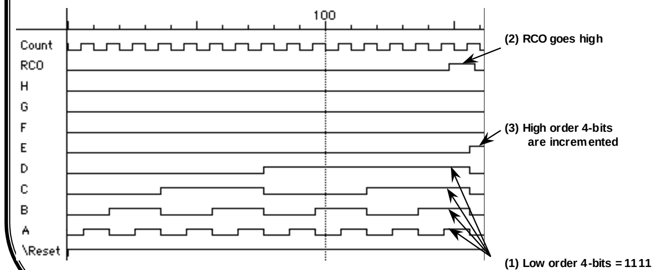
State transitions are not sharp!

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

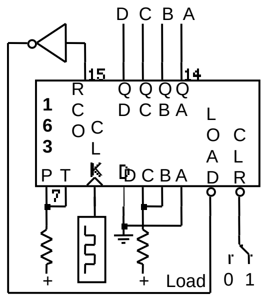
Downstream stages lag in their 1111 to 0000 transitions

Affects Count period and decoding logic

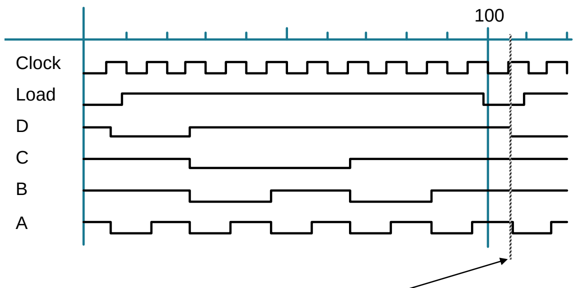
Asynchronous vs. Synchronous Counters

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



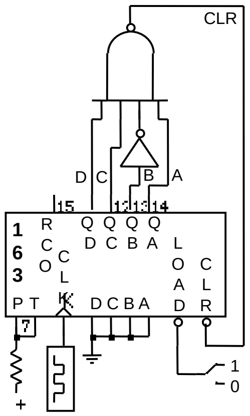
Use RCO signal to trigger Load of a new state

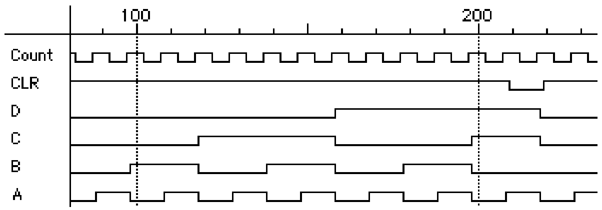
Since 74163 Load is synchronous, state changes only on the next rising clock edge

Asynchronous vs. Synchronous Counters

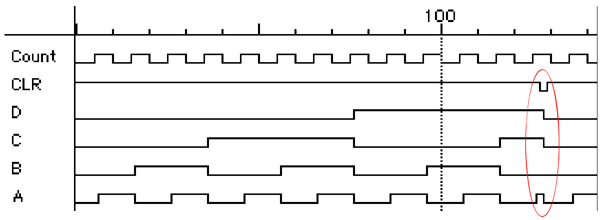
Offset Counters Continued

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





Clear signal takes effect on the rising count edge



Decode state to determine when to reset to 0000

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

Random Access Memories

Static RAM

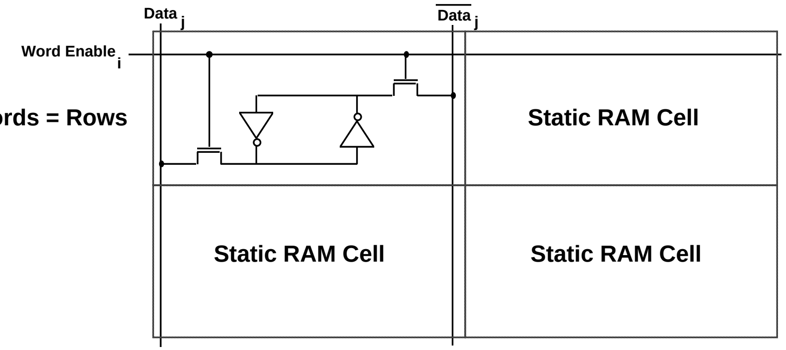
Transistor efficient methods for implementing storage elements

Small RAM: 256 words by 4-bit

Large RAM: 4 million words by 1-bit

We will discuss a 1024 x 4 organization

Words = Rows



Columns = Bits (Double Rail Encoded)

Random Access Memories

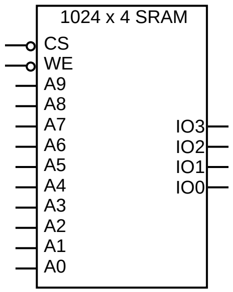
Static RAM Organization

Chip Select Line (active lo)

Write Enable Line (active lo)

10 Address Lines

4 Bidirectional Data Lines



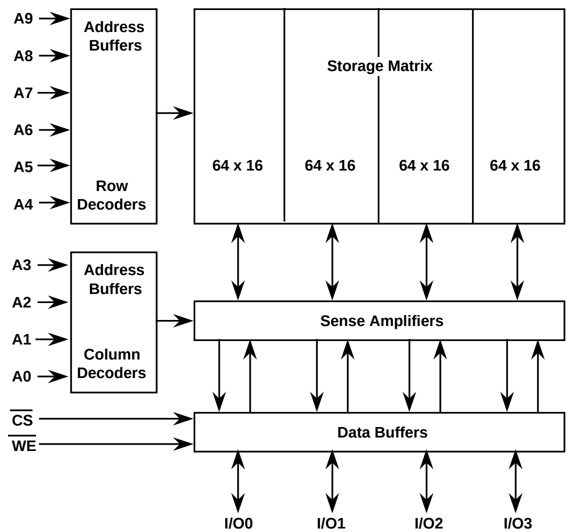
Random Access Memories

RAM Organization

Long thin layouts are not the best organization for a RAM

Some Addr bits select row

Some Addr bits select within row



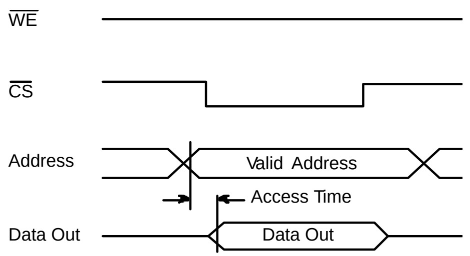
64 x 64 Square Array

Amplifers & Mux/Demux

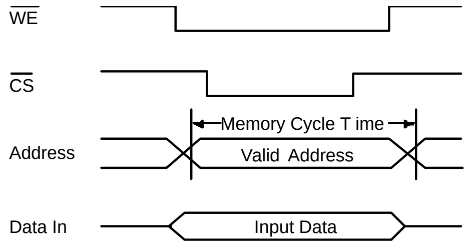
Random Access Memories

RAM Timing

Simplified Read Timing

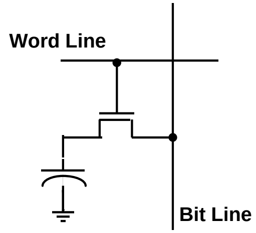


Simplified Write Timing



Random Access Memories

Dynamic RAMs



1 Transistor (+ capacitor) memory element

Read: Assert Word Line, Sense Bit Line

Write: Drive Bit Line, Assert Word Line

Destructive Read-Out

Need for Refresh Cycles: storage decay in ms

Internal circuits read word and write back

Random Access Memories

DRAM Organization

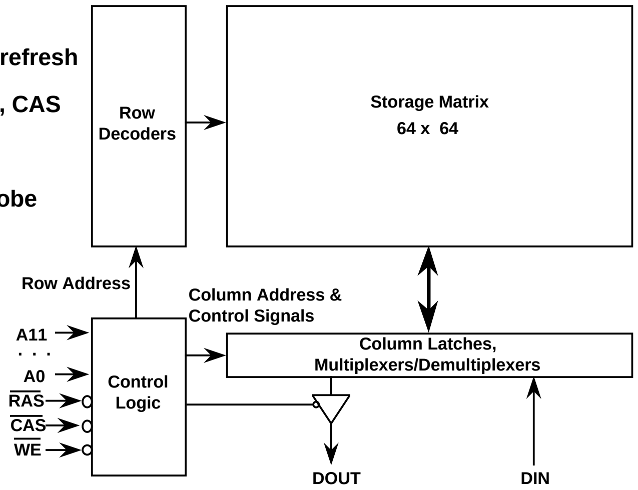
Long rows to simplify refresh

Two new signals: RAS, CAS

Row Address Strobe

Column Address Strobe

replace Chip Select



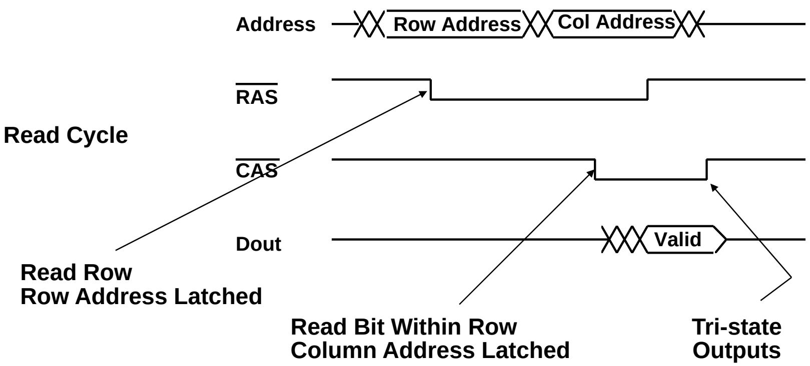
Random Access Memory

RAS, CAS Addressing

Even to read 1 bit, an entire 64-bit row is read!

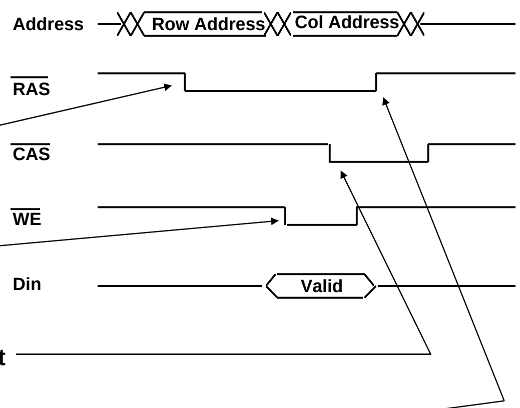
Separate addressing into two cycles: Row Address, Column Address
Saves on package pins, speeds RAM access for sequential bits!

Read Cycle



Random Access Memory

Write Cycle Timing



(1) Latch Row Address
Read Row

(2) WE low

(3) CAS low: replace data bit

(4) RAS high: write back the modified row

(5) CAS high to complete the memory cycle

Random Access Memory

RAM Refresh

Refresh Frequency:

4096 word RAM -- refresh each word once every 4 ms

Assume 120ns memory access cycle

This is one refresh cycle every 976 ns (1 in 8 DRAM accesses)!

But RAM is really organized into 64 rows

This is one refresh cycle every 62.5 µs (1 in 500 DRAM accesses)

Large capacity DRAMs have 256 rows, refresh once every 16 µs

RAS-only Refresh (RAS cycling, no CAS cycling)

External controller remembers last refreshed row

Some memory chips maintain refresh row pointer

CAS before RAS refresh: if CAS goes low before RAS, then refresh

Random Access Memory

DRAM Variations

Page Mode DRAM:

read/write bit within last accessed row without RAS cycle

RAS, CAS, CAS, . . ., CAS, RAS, CAS, ...

New column address for each CAS cycle

Static Column DRAM:

like page mode, except address bit changes signal new cycles rather than CAS cycling

on writes, deselect chip or CAS while address lines are changing

Nibble Mode DRAM:

like page mode, except that CAS cycling implies next column address in sequence -- no need to specify column address after first CAS

Works for 4 bits at a time (hence "nibble")
RAS, CAS, CAS, CAS, CAS, RAS, CAS, CAS, CAS, CAS, . . .

Chapter Summary

- The Variety of Sequential Circuit Packages
  Registers, Shifters, Counters, RAMs
- Counters as Simple Finite State Machines
- Counter Design Procedure
  1. Derive State Diagram
  2. Derive State Transition Table
  3. Determine Next State Functions
  4. Remap Next State Functions for Target FF Types
     Using Excitation Tables; 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 area in VLSI
- 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
  Never use counters with asynchronous clear for this kind of application