Chapter # 4: Programmable and Steering Logic

Chapter Overview

- *PALs and PLAs*

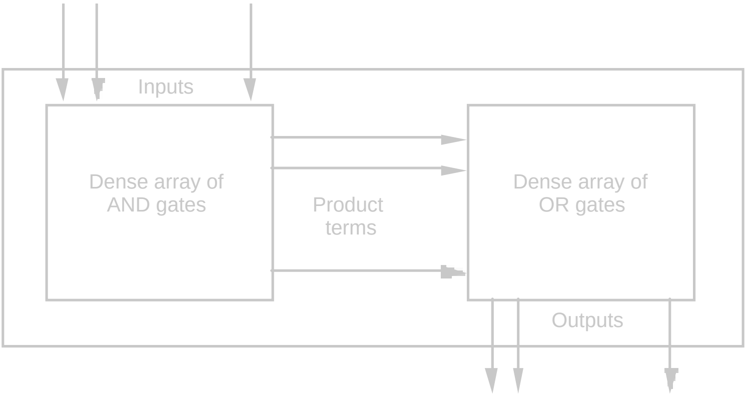
- *Non-Gate Logic*
  Switch Logic
  Multiplexers/Selecters and Decoders
  Tri-State Gates/Open Collector Gates
  ROM

- *Combinational Logic Design Problems*
  Seven Segment Display Decoder
  Process Line Controller
  Logical Function Unit
  Barrel Shifter

PALs and PLAs

Pre-fabricated building block of many AND/OR gates (or NOR, NAND)
"Personalized" by making or breaking connections among the gates

Programmable Array Block Diagram for Sum of Products Form



PALs and PLAs

Key to Success: Shared Product Terms

Equations

Example:

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

Personality Matrix

Product term	Inputs A	B	C	Outputs F_0	F_1	F_2	F_3
A B	1	1	-	0	1	1	0
$\overline{B}$ C	-	0	1	0	0	0	1
A $\overline{C}$	1	-	0	0	1	0	0
$\overline{B}$ $\overline{C}$	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Reuse of terms

Input Side:

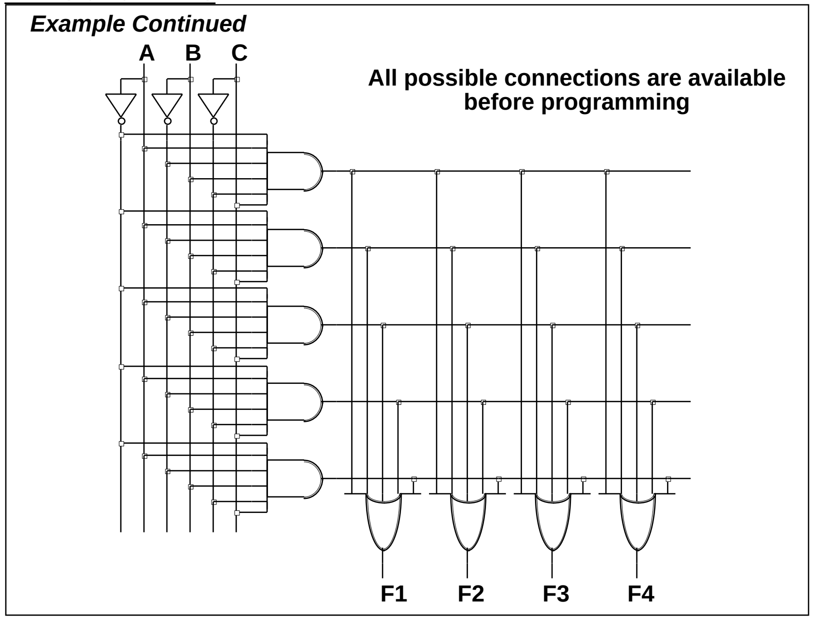
1 = asserted in term
0 = negated in term
- = does not participate

Output Side:

1 = term connected to output
0 = no connection to output

PALs and PLAs

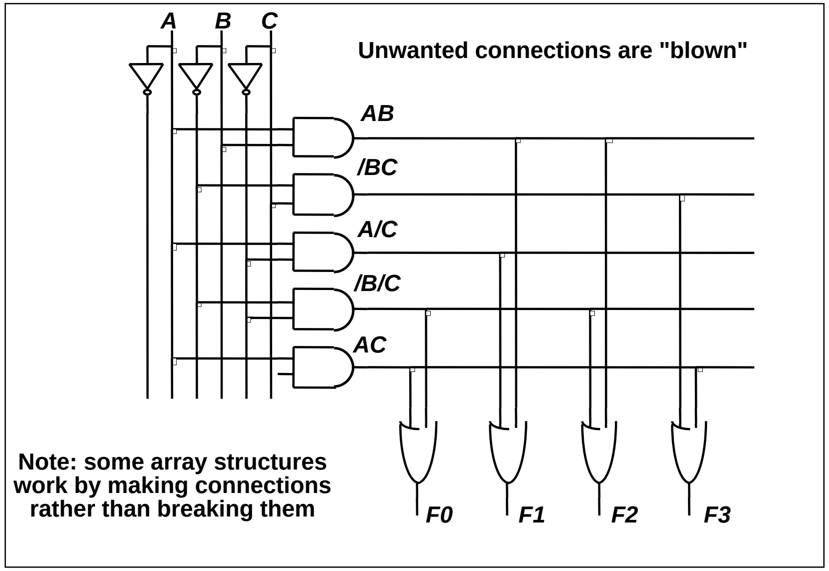
Example Continued



All possible connections are available before programming

PALs and PLAs

Example Continued

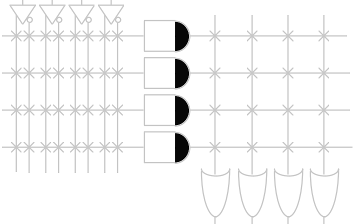


Unwanted connections are "blown"

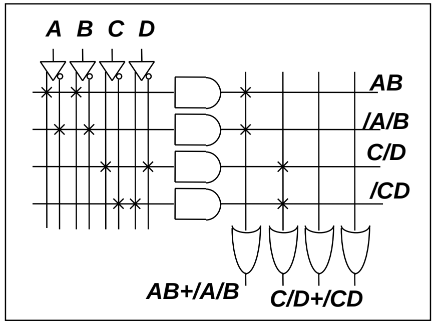
Note: some array structures work by making connections rather than breaking them

PALs and PLAs

Alternative representation for high fan-in structures



Short-hand notation so we don't have to draw all the wires!



Notation for implementing
F0 = A B + A' B'
F1 = C D' + C' D

PALs and PLAs

Design Example

Multiple functions of A, B, C

F1 = A B C

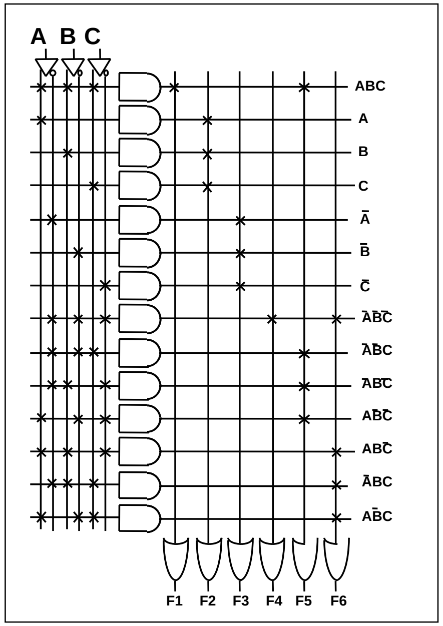
F2 = A + B + C

F3 = $\overline{\text{A B C}}$

F4 = $\overline{\text{A + B + C}}$

F5 = A xor B xor C

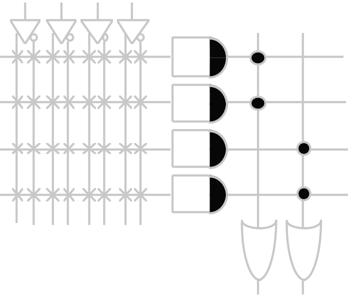
F6 = $\overline{\text{A xor B xor C}}$



PALs and PLAs

What is difference between Programmable Array Logic (PAL) and Programmable Logic Array (PLA)?

PAL concept : implemented by Monolithic Memories
(substrate is active materialsuch as semiconductor silicon)
programmable AND array but constrained topology of the OR Array
connections between product terms are hardwired
the higher the OR gate fanins, the fewer the functional outputs from PAL



A given column of the OR array has access to only a subset of the possible product terms

PLA concept : generalized topologies in AND and OR planes
take advantage of shared product terms
slower

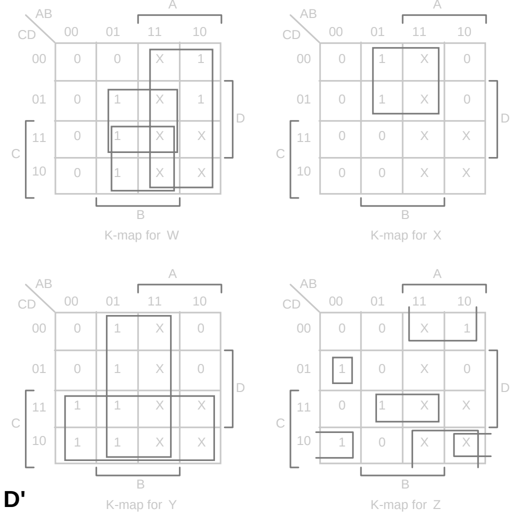
PALs and PLAs

Design Example: BCD to Gray Code Converter

Truth Table

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

K-maps



K-map for W

CD \ AB	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

K-map for X

CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	0	0	X	X
10	0	0	X	X

K-map for Y

CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	1	X	X
10	1	1	X	X

K-map for Z

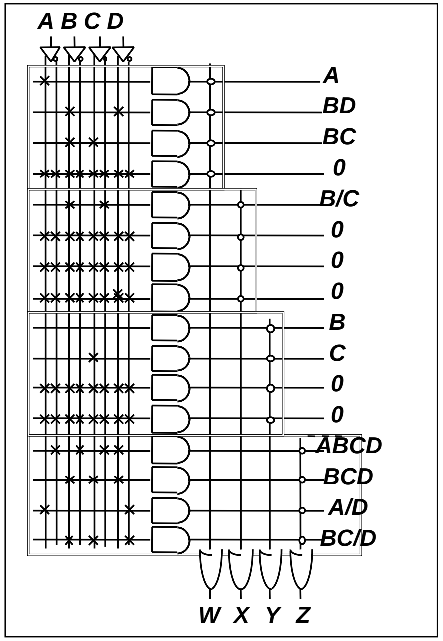
CD \ AB	00	01	11	10
00	0	0	X	1
01	1	0	X	0
11	0	1	X	X
10	1	0	X	X

Minimized Functions:

W = A + B D + B C
X = B C'
Y = B + C
Z = A'B'C'D + B C D + A D' + B' C D'

PALs and PLAs

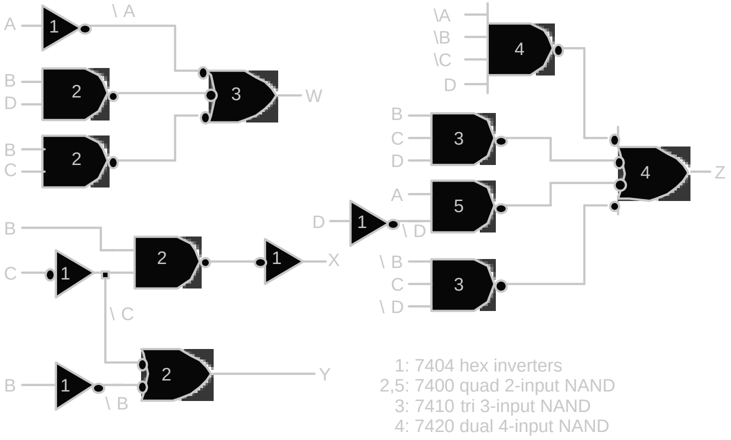
Programmed PAL:



4 product terms per each OR gate

PALs and PLAs

Code Converter Discrete Gate Implementation



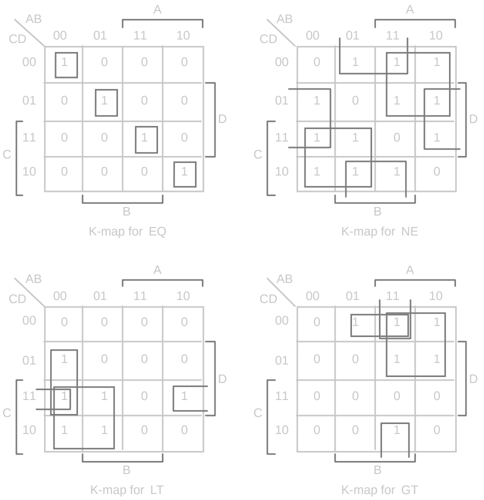
1: 7404 hex inverters
2,5: 7400 quad 2-input NAND
3: 7410 tri 3-input NAND
4: 7420 dual 4-input NAND

4 SSI Packages vs. 1 PLA/PAL Package!

PALs and PLAs

Another Example: Magnitude Comparator

AB=CD AB≠CD AB<CD AB>CD



K-map for EQ

CD \ AB	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

K-map for NE

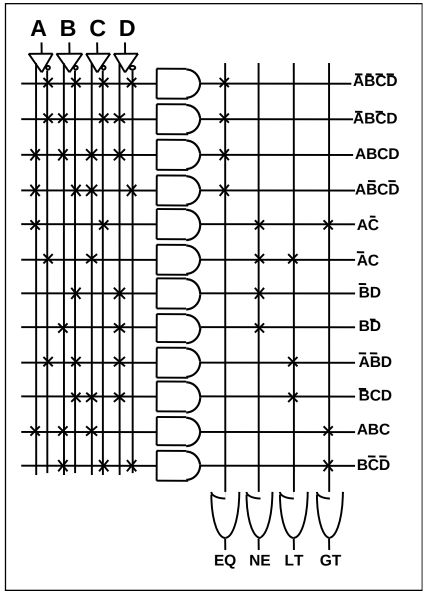
CD \ AB	00	01	11	10
00	0	1	1	1
01	1	0	1	1
11	1	1	0	1
10	1	1	1	0

K-map for LT

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

K-map for GT

CD \ AB	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0



Non-Gate Logic

Introduction

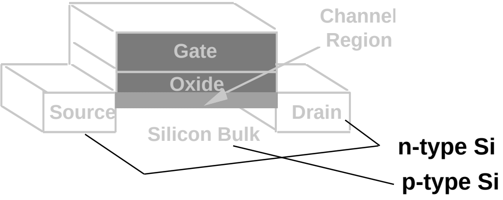
AND-OR-Invert PAL/PLA	Generalized Building Blocks Beyond Simple Gates

Kinds of "Non-gate logic":

- switching circuits built from CMOS transmission gates
- multiplexer/selecter functions
- decoders
- tri-state and open collector gates
- read-only memories

Steering Logic: Switches

Voltage Controlled Switches



"n-Channel MOS"

Metal Gate, Oxide, Silicon Sandwich

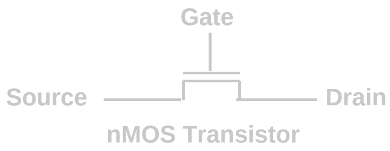
Diffusion regions: negatively charged ions driven into Si surface

Si Bulk: positively charged ions

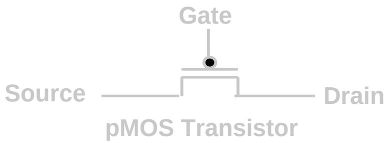
By "pulling" electrons to the surface, a conducting channel is formed

Steering Logic

Voltage Controlled Switches



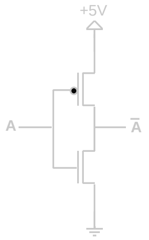
Logic 1 on gate,
Source and Drain connected



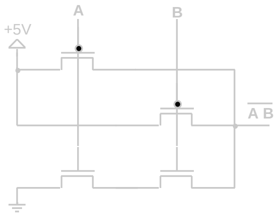
Logic 0 on gate,
Source and Drain connected

Steering Logic

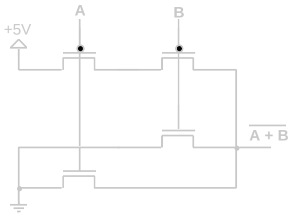
Logic Gates from Switches



Inverter



NAND Gate



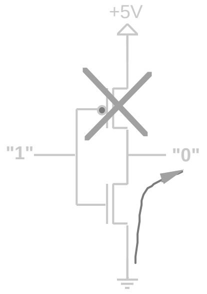
NOR Gate

Pull-up network constructed from pMOS transistors

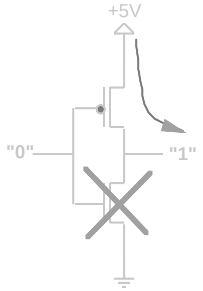
Pull-down network constructed from nMOS transistors

Steering Logic

Inverter Operation



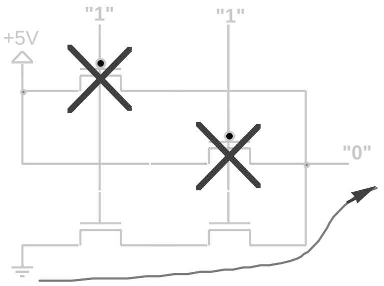
Input is 1
Pull-up does not conduct
Pull-down conducts
Output connected to GND



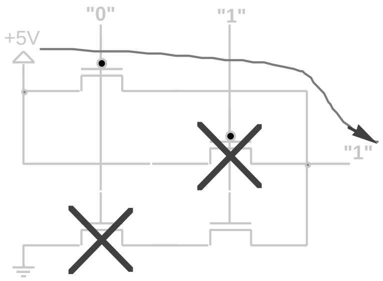
Input is 0
Pull-up conducts
Pull-down does not conduct
Output connected to VDD

Steering Logic

NAND Gate Operation



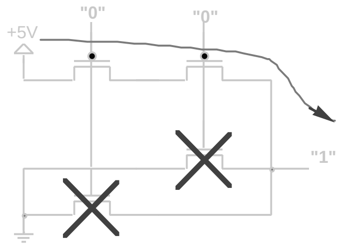
A = 1, B = 1
Pull-up network does not conduct
Pull-down network conducts
Output node connected to GND



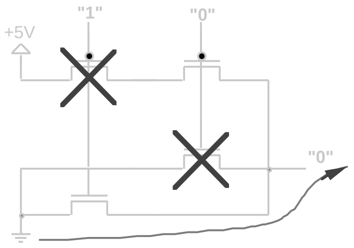
A = 0, B = 1
Pull-up network has path to VDD
Pull-down network path broken
Output node connected to VDD

Steering Logic

NOR Gate Operation



A = 0, B = 0
Pull-up network conducts
Pull-down network broken
Output node at VDD



A = 1, B = 0
Pull-up network broken
Pull-down network conducts
Output node at GND

Steering Logic

CMOS Transmission Gate

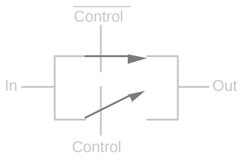
nMOS transistors good at passing 0's but bad at passing 1's

produce weak 1

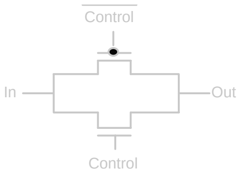
pMOS transistors good at passing 1's but bad at passing 0's

produce weak 0

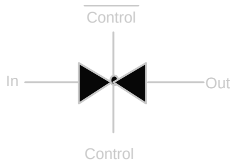
perfect "transmission" gate places these in parallel:



Switches



Transistors

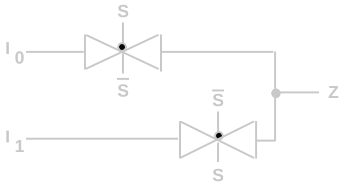


Transmission or "Butterfly" Gate

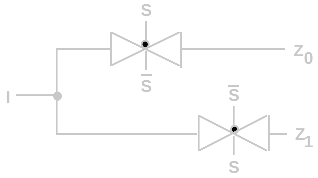
Steering Logic

Selection Function/Demultiplexer Function with Transmission Gates

Selector:
Choose I0 if S = 0
Choose I1 if S = 1

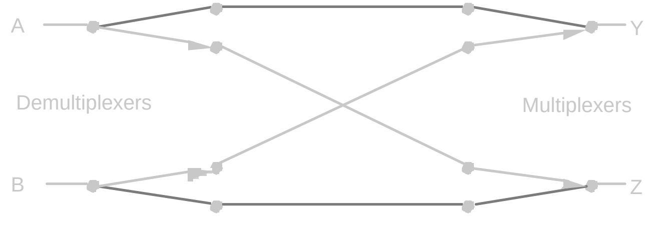


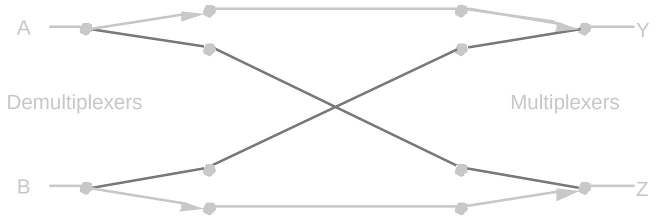
Demultiplexer:
I to Z0 if S = 0
I to Z1 if S = 1



Steering Logic

Use of Multiplexer/Demultiplexer in Digital Systems





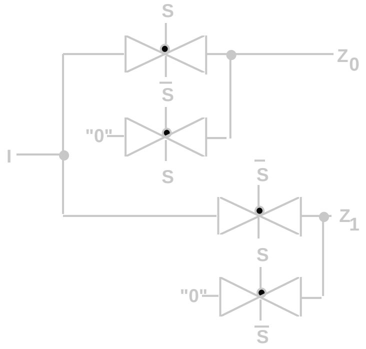
So far, we've only seen point-to-point connections among gates

Mux/Demux used to implement multiple source/multiple destination interconnect

Steering Logic

Well-formed Switching Networks

Problem with the Demux implementation:
multiple outputs, but only one connected to the input!



The fix: additional logic to drive every output to a known value

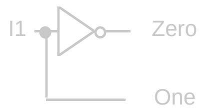
Never allow outputs to "float"

Steering Logic

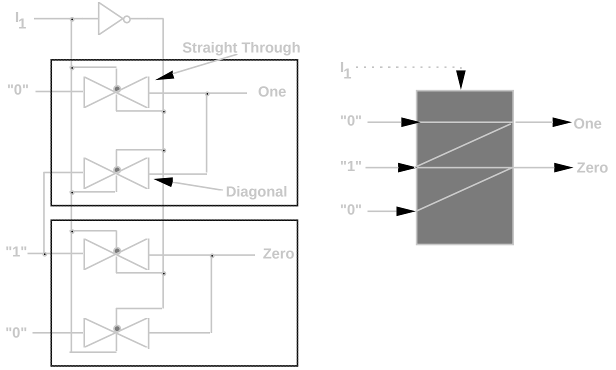
Complex Steering Logic Example

N Input Tally Circuit: count # of 1's in the inputs

I_1	Zero	One
0	1	0
1	0	1



Conventional Logic for 1 Input Tally Function

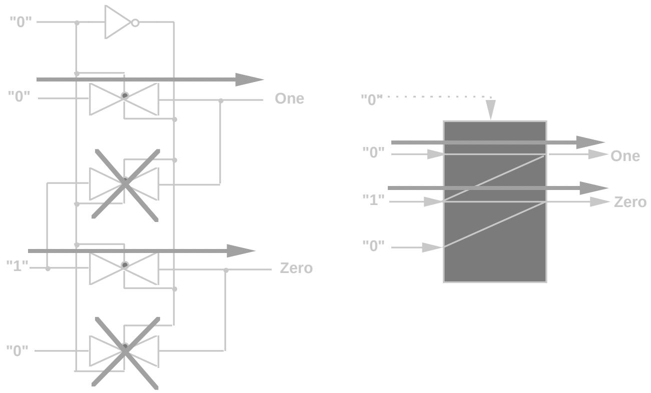


Switch Logic Implementation of Tally Function

Steering Logic

Complex Steering Logic Example

Operation of the 1 Input Tally Circuit

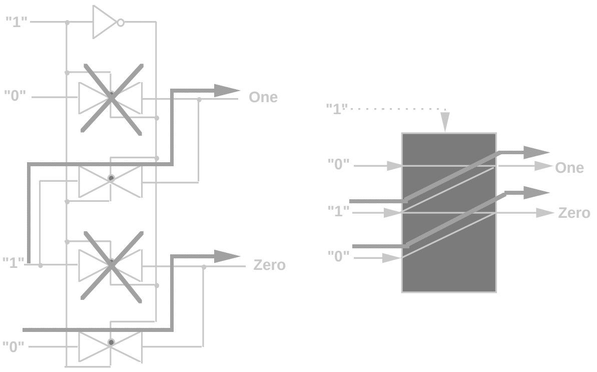


Input is 0, straight through switches enabled

Steering Logic

Complex Steering Logic Example

Operation of 1 input Tally Circuit



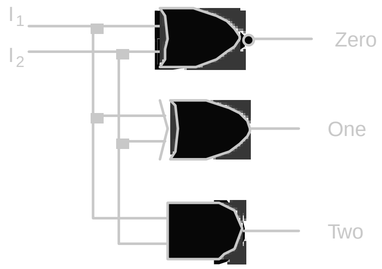
Input = 1, diagonal switches enabled

Steering Logic

Complex Steering Logic Example

Extension to the 2-input case

I_1	I_2	Zero	One	Two
0	0	1	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

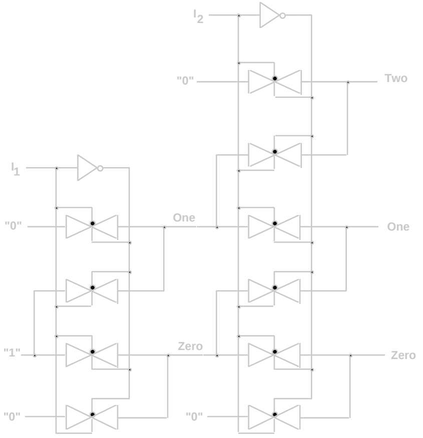


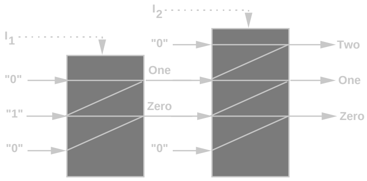
Conventional logic implementation

Steering Logic

Complex Steering Logic Example

Switch Logic Implementation: 2-input Tally Circuit



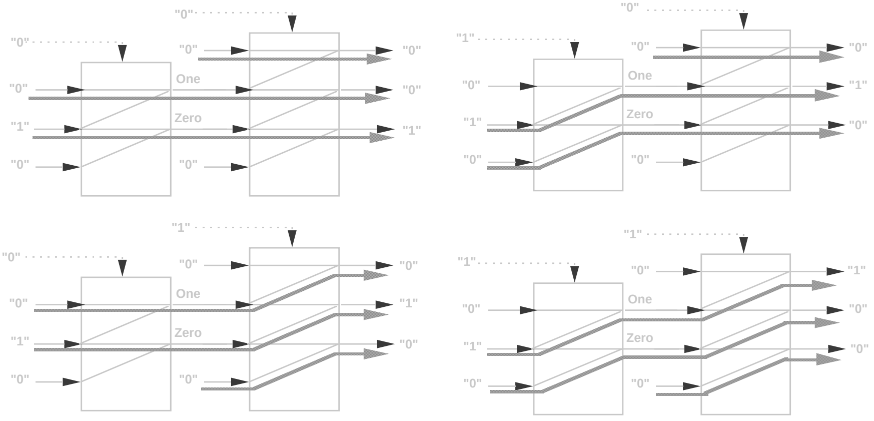


Cascade the 1-input implementation!

Steering Logic

Complex Steering Logic Example

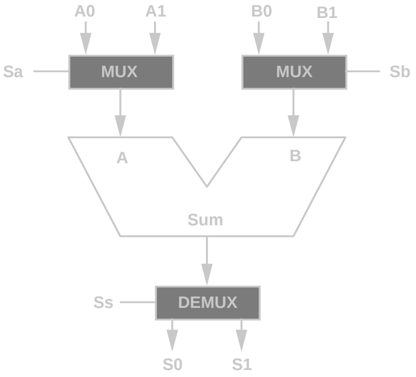
Operation of 2-input implementation



Multiplexers/Selectors

Use of Multiplexers/Selectors

Multi-point connections



Multiple input sources

Multiple output destinations

Multiplexers/Selectors

General Concept

2^n data inputs, n control inputs, 1 output

used to connect 2^n points to a single point

control signal pattern form binary index of input connected to output

$Z = A' I_0 + A I_1$

A	Z
0	I_0
1	I_1

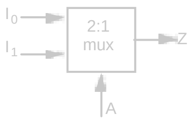
I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Functional form

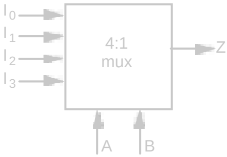
Logical form

Two alternative forms for a 2:1 Mux Truth Table

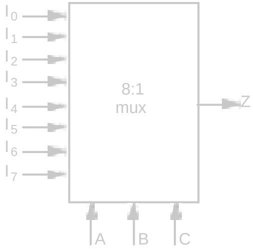
Multiplexers/Selectors



$Z = A' I_0 + A I_1$



Z = A' B' I0 + A' B I1 + A B' I2 + A B I3

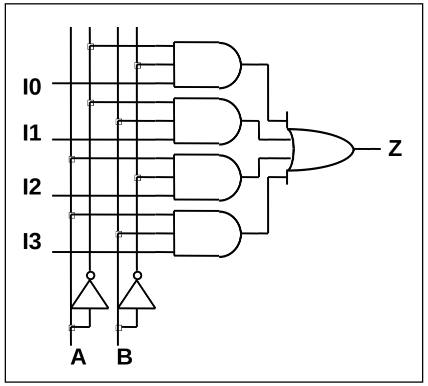


Z = A' B' C' I0 + A' B' C I1 + A' B C' I2 + A' B C I3 +
A B' C' I4 + A B' C I5 + A B C' I6 + A B C I7

In general, $Z = \Sigma_{k=0}^{2^n-1} m_k I_k$
in minterm shorthand form for a 2^n:1 Mux

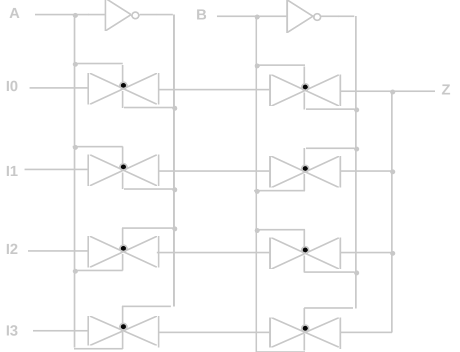
Multiplexers/Selectors

Alternative Implementations



Gate Level Implementation of 4:1 Mux

thirty six transistors

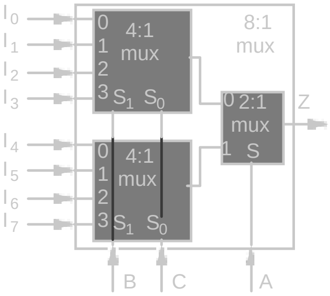


Transmission Gate Implementation of 4:1 Mux

twenty transistors

Multiplexer/Selector

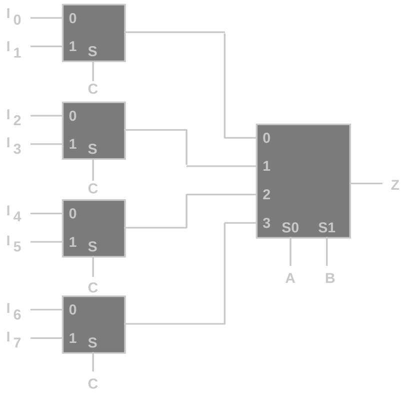
Large multiplexers can be implemented by cascaded smaller ones



Control signals B and C simultaneously choose one of I0-I3 and I4-I7

Control signal A chooses which of the upper or lower MUX's output to gate to Z

Alternative 8:1 Mux Implementation



Multiplexer/Selector

Multiplexers/selectors as a general purpose logic block

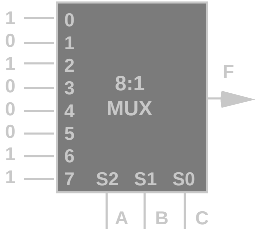
2^{n-1}:1 multiplexer can implement any function of n variables

n-1 control variables; remaining variable is a data input to the mux

Example:

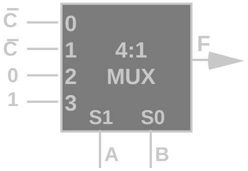
F(A,B,C) = m0 + m2 + m6 + m7
= A' B' C' + A' B C' + A B C' + A B C

= A' B' (C') + A' B (C') + A B' (0) + A B (1)



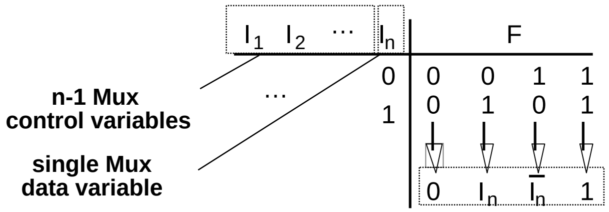
"Lookup Table"

A	B	C	F	
0	0	0	1	$\overline{C}$
0	0	1	0	
0	1	0	1	$\overline{C}$
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



Multiplexer/Selector

Generalization



Four possible configurations of the truth table rows

Can be expressed as a function of In, 0, 1

Example:

G(A,B,C,D) can be implemented by an 8:1 MUX:

CD \ AB	00	01	11	10
00	1	0	1	1
01	1	0	0	0
11	1	1	0	1
10	0	1	1	0

A

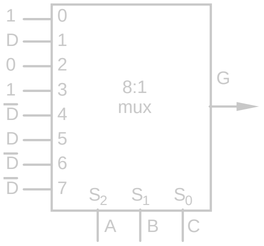
B

C

K-map
Choose A,B,C
as control variables

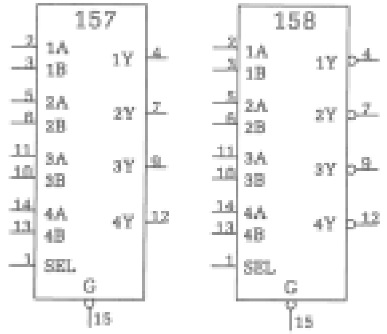
Multiplexer Implementation

TTL package efficient
May be gate inefficient



Multiplexer/Selector

- TTL quad 2:1 multiplexers with enable



Enable G	Select SEL	Input A	Input B	'157 Y	'158 Y
1	X	X	X	0	1
0	0	0	X	0	1
0	0	1	X	1	0
0	1	X	0	0	1
0	1	X	1	1	0

Decoders/Demultiplexers

Decoder: single data input, n control inputs, 2^n outputs

control inputs (called select S) represent Binary index of output to which the input is connected

data input usually called "enable" (G)

1:2 Decoder:

O0 = G • $\overline{S}$; O1 = G • S

2:4 Decoder:

O0 = G • $\overline{S0}$ • $\overline{S1}$

O1 = G • $\overline{S0}$ • S1

O2 = G • S0 • $\overline{S1}$

O3 = G • S0 • S1

3:8 Decoder:

O0 = G • $\overline{S0}$ • $\overline{S1}$ • $\overline{S2}$

O1 = G • $\overline{S0}$ • $\overline{S1}$ • S2

O2 = G • $\overline{S0}$ • S1 •$\overline{S2}$

O3 = G • $\overline{S0}$ • S1 • S2

O4 = G • S0 • $\overline{S1}$ • $\overline{S2}$

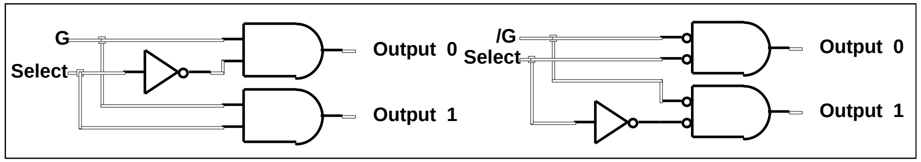
O5 = G • S0 • $\overline{S1}$ • S2

O6 = G • S0 • S1 • $\overline{S2}$

O7 = G • S0 • S1 • S2

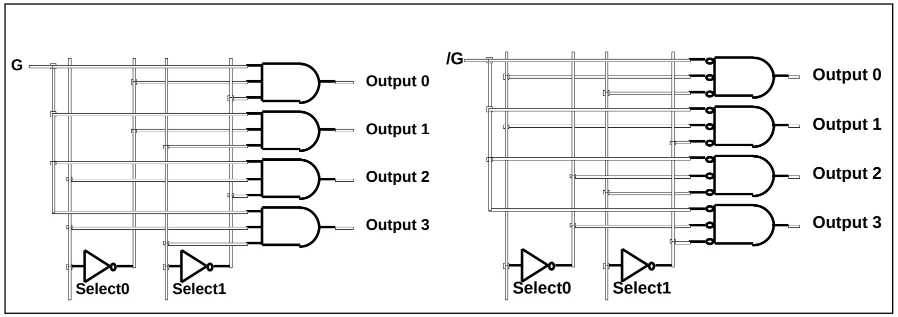
Decoders/Demultiplexers

Alternative Implementations



1:2 Decoder, Active High Enable

1:2 Decoder, Active Low Enable

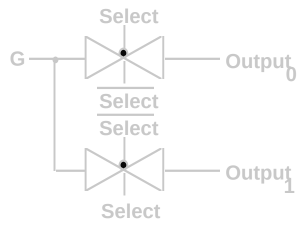


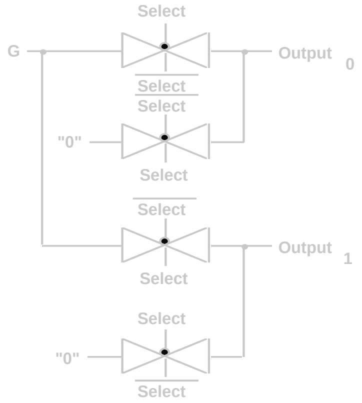
2:4 Decoder, Active High Enable

2:4 Decoder, Active Low Enable

Decoders/Demultiplexers

Switch Logic Implementations




Select
G
Output$_0$
$\overline{\text{Select}}$
Select
"0"
Select
$\overline{\text{Select}}$
Output$_1$
Select
Select
"0"
$\overline{\text{Select}}$

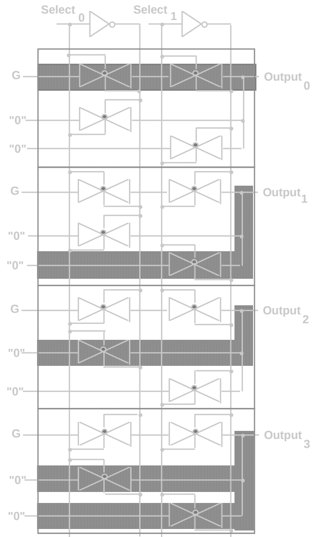
Naive, Incorrect Implementation

All outputs not driven at all times

Correct 1:2 Decoder Implementation

Decoders/Demultiplexers

Switch Implementation of 2:4 Decoder



Operation of 2:4 Decoder

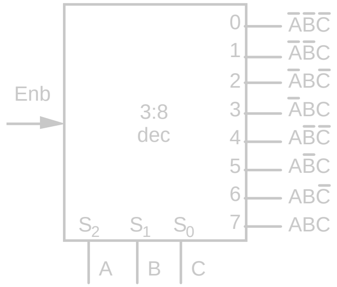
S0 = 0, S1 = 0

one straight thru path

three diagonal paths

Decoder/Demultiplexer

Decoder as a Logic Building Block


Enb
3:8 dec
0 $\overline{A}\,\overline{B}\,\overline{C}$
1 $\overline{A}\,\overline{B}C$
2 $\overline{A}B\overline{C}$
3 $\overline{A}BC$
4 $A\overline{B}\,\overline{C}$
5 $A\overline{B}C$
6 $AB\overline{C}$
7 ABC
S_2 S_1 S_0
A B C

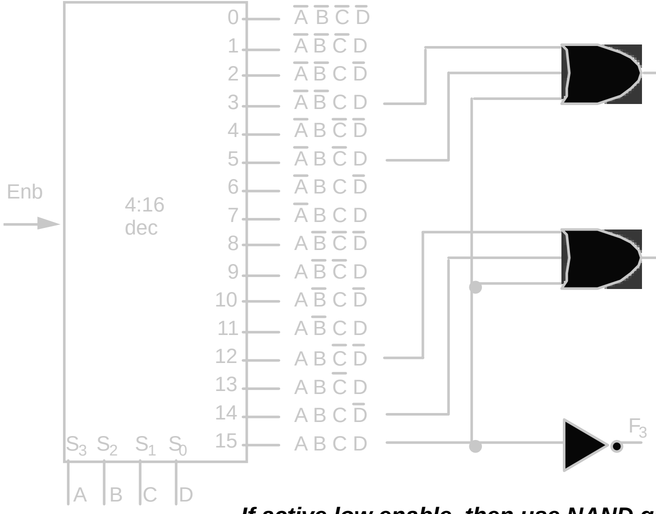
Decoder Generates Appropriate Minterm based on Control Signals

Example Function:

F1 = A' B C' D + A' B' C D + A B C D
F2 = A B C' D' + A B C
F3 = (A' + B' + C' + D')

Decoder/Demultiplexer

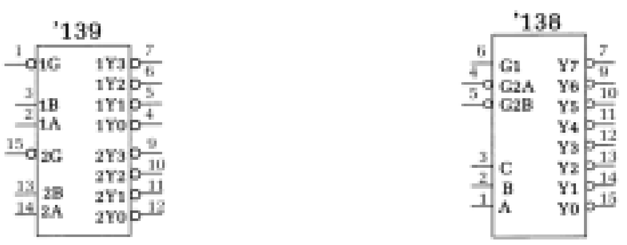
Decoder as a Logic Building Block


Enb
4:16 dec
0 $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
1 $\overline{A}\,\overline{B}\,\overline{C}D$
2 $\overline{A}\,\overline{B}C\overline{D}$
3 $\overline{A}\,\overline{B}CD$
4 $\overline{A}B\overline{C}\,\overline{D}$
5 $\overline{A}B\overline{C}D$
6 $\overline{A}BC\overline{D}$
7 $\overline{A}BCD$
8 $A\overline{B}\,\overline{C}\,\overline{D}$
9 $A\overline{B}\,\overline{C}D$
10 $A\overline{B}C\overline{D}$
11 $A\overline{B}CD$
12 $AB\overline{C}\,\overline{D}$
13 $AB\overline{C}D$
14 $ABC\overline{D}$
15 $ABCD$
S_3 S_2 S_1 S_0
A B C D
F_1
F_2
F_3

If active low enable, then use NAND gates!

Decoder/Demultiplexer

- TTL decoder components

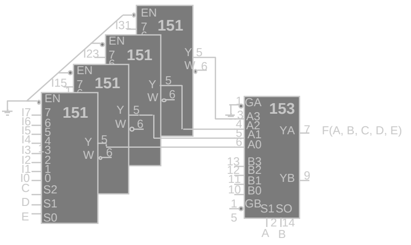


$\overline{G}$	B	A	$\overline{Y0}$	$\overline{Y1}$	$\overline{Y2}$	$\overline{Y3}$
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

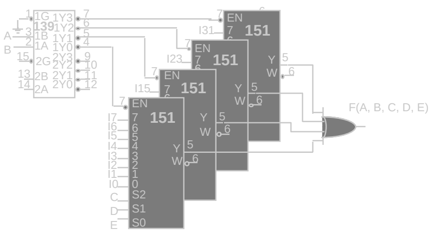
G1	$\overline{G2A} + \overline{G2B}$	C	B	A	$\overline{Y0}$	$\overline{Y1}$	$\overline{Y2}$	$\overline{Y3}$	$\overline{Y4}$	$\overline{Y5}$	$\overline{Y6}$	$\overline{Y7}$
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

Multiplexers/Decoders

Alternative Implementations of 32:1 Mux



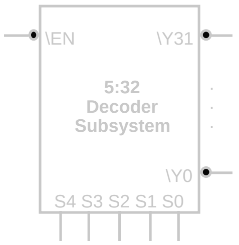
Multiplexer Only

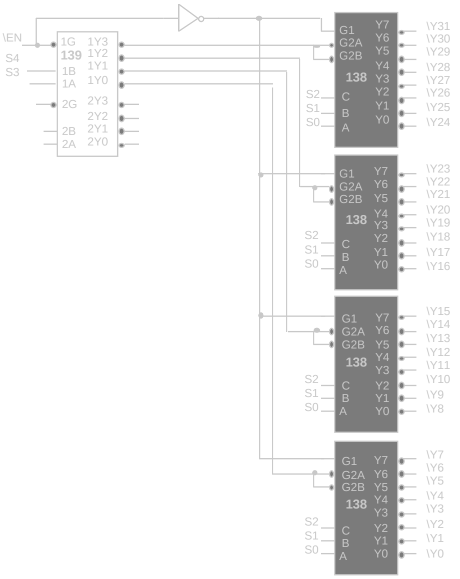


Multiplexer + Decoder

Multiplexers/Decoders

5:32 Decoder





Tri-State and Open-Collector

The Third State

Logic States: "0", "1"
Don't Care/Don't Know State: "X" (must be some value in real circuit!)

Third State: "Z" — high impedance — infinite resistance, no connection

Tri-state gates: output values are "0", "1", and "Z"
additional input: output enable (OE)

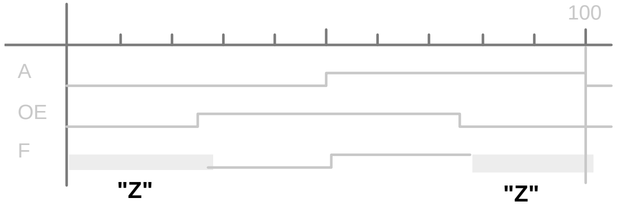
A	OE	F
X	0	Z
0	1	0
1	1	1

When OE is high, this gate is a non-inverting "buffer"

When $\overline{\text{OE}}$ is low, it is as though the gate was *disconnected* from the output!

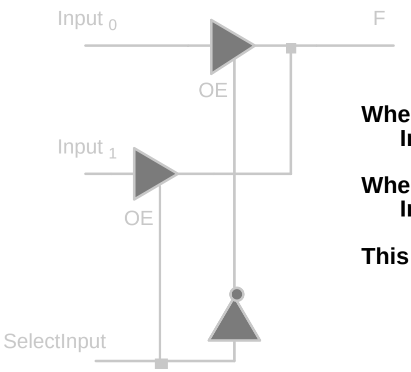
This allows more than one gate to be connected to the same output wire, as long as only one has its output enabled *at the same time*

Non-inverting buffer's timing waveform



Tri-state and Open Collector

Using tri-state gates to implement an economical multiplexer:



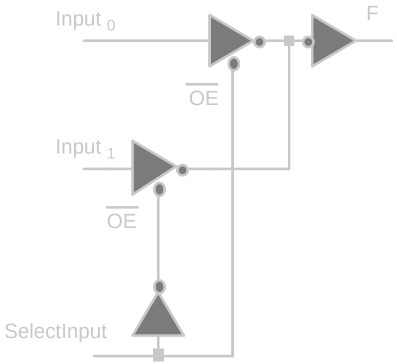
When SelectInput is asserted high Input1 is connected to F

When SelectInput is driven low Input0 is connected to F

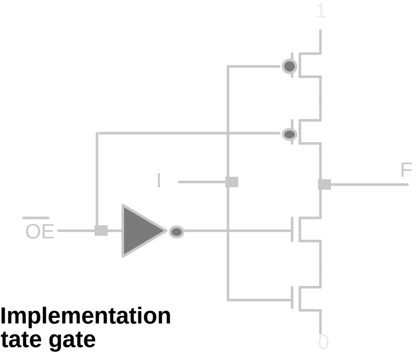
This is essentially a 2:1 Mux

Tri-state and Open Collector

Alternative Tri-state Fragment



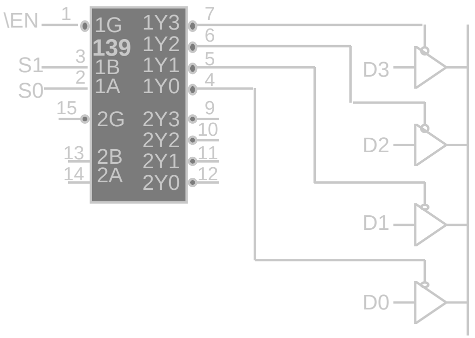
Active low tri-state enables plus inverting tri-state buffers



Switch Level Implementation of tri-state gate

Tri-State and Open Collector

4:1 Multiplexer, Revisited



Decoder + 4 tri-state Gates

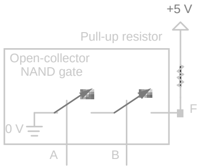
Tri-State and Open Collector

Open Collector

another way to connect multiple gates to the same output wire

gate only has the ability to pull its output low; it cannot actively drive the wire high

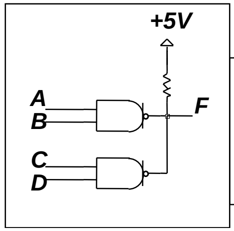
this is done by pulling the wire up to a logic 1 voltage through a resistor

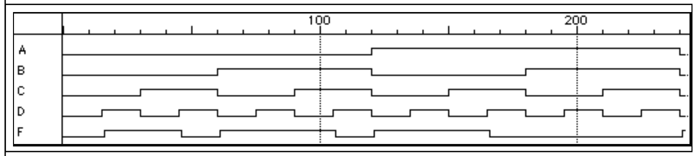


OC NAND gates

Wired AND:

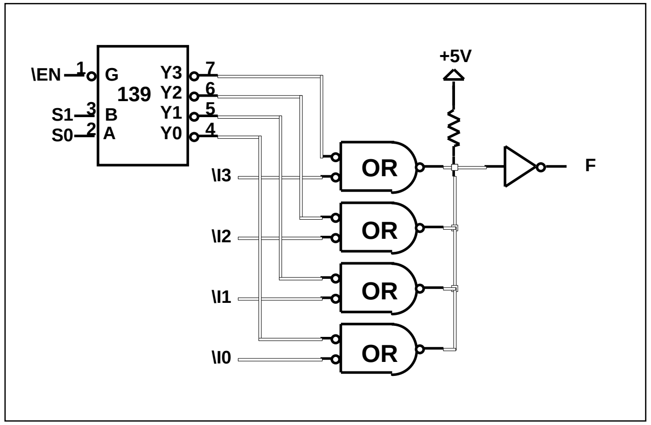
If A and B are "1", output is actively pulled low
if C and D are "1", output is actively pulled low
if one gate is low, the other high, then low wins
if both gates are "1", the output floats, pulled high by resistor
Hence, the two NAND functions are AND'd together!





Tri-State and Open Collector

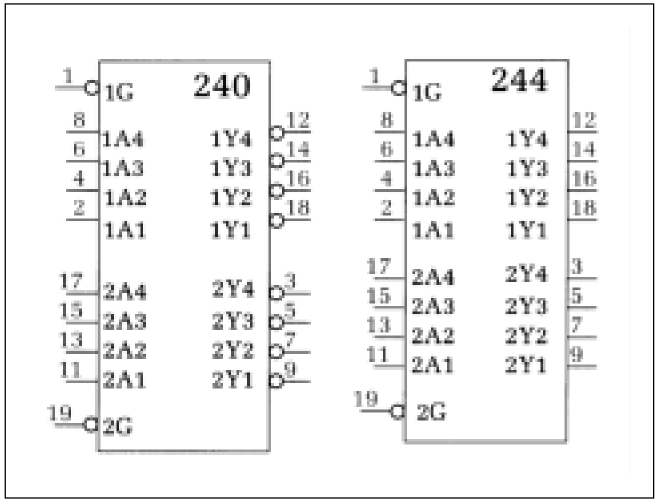
4:1 Multiplexer



Decoder + 4 Open Collector Gates

Tri-State and Open Collector

- **TTL tri-state buffers**



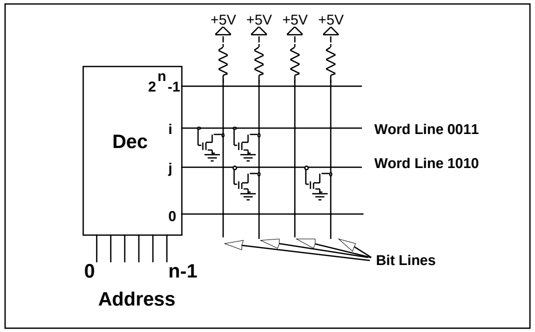
Read-Only Memories

ROM: Two dimensional array of 1's and 0's

Row is called a "word"; index is called an "address"

Width of row is called *bit-width* or *wordsize*

Address is input, selected word is output



Internal Organization

Read-Only Memories

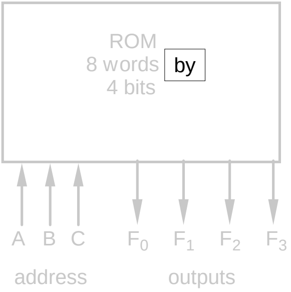
Example: Combination Logic Implementation

F0 = A' B' C + A B' C' + A B' C

F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'



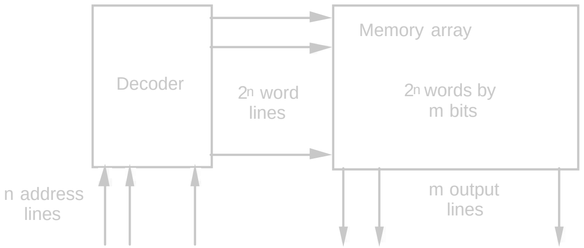
Address

Word Contents

A	B	C	F_0	F_1	F_2	F_3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

Read-Only Memories

Not unlike a PLA structure with a fully decoded AND array!



ROM vs. PLA:

ROM approach advantageous when
(1) design time is short (no need to minimize output functions)
(2) most input combinations are needed (e.g., code converters)
(3) little sharing of product terms among output functions

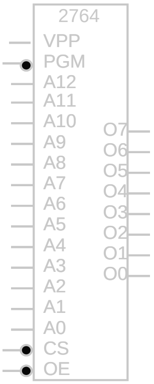
ROM problem: size doubles for each additional input, can't use don't cares

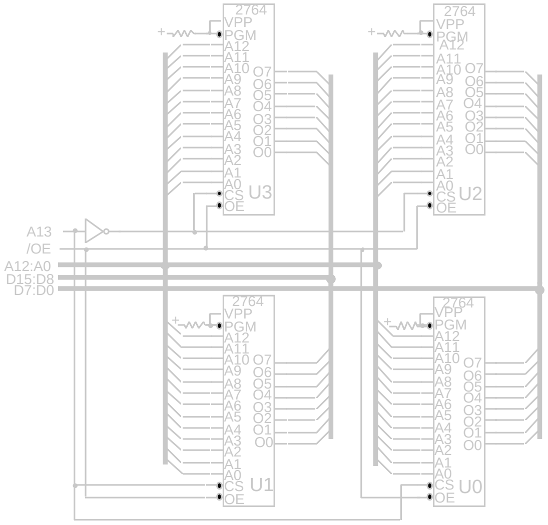
PLA approach advantangeous when
(1) design tool like espresso is available
(2) there are relatively few unique minterm combinations
(3) many minterms are shared among the output functions

PAL problem: constrained fan-ins on OR planes

Read-Only Memories

2764 EPROM
8K x 8





16K x 16
Subsystem

Combinational Logic Word Problems

General Design Procedure

1. Understand the Problem
   what is the circuit supposed to do?
   write down inputs (data, control) and outputs
   draw block diagram or other picture

2. Formulate the Problem in terms of a truth table or other suitable design representation
   truth table or waveform diagram

3. Choose Implementation Target
   ROM, PAL, PLA, Mux, Decoder + OR, Discrete Gates

4. Follow Implementation Procedure
   K-maps, espresso, misII

Combinational Logic Word Problems

Process Line Control Problem

Statement of the Problem

Rods of varying length (+/-10%) travel on conveyor belt
Mechanical arm pushes rods within spec (+/-5%) to one side
Second arm pushes rods too long to other side
Rods too short stay on belt

3 light barriers (light source + photocell) as sensors

Design combinational logic to activate the arms

Understanding the Problem

Inputs are three sensors, outputs are two arm control signals

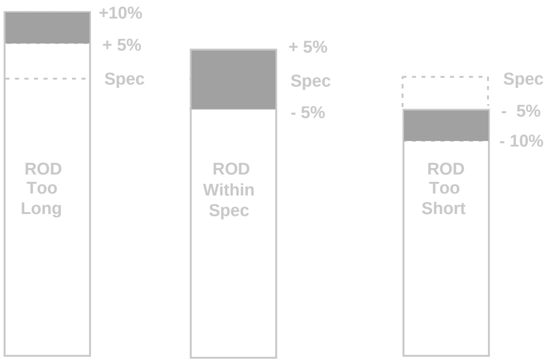
Assume sensor reads "1" when tripped, "0" otherwise

Call sensors A, B, C

Draw a picture!

Combinational Logic Word Problems

Process Control Problem

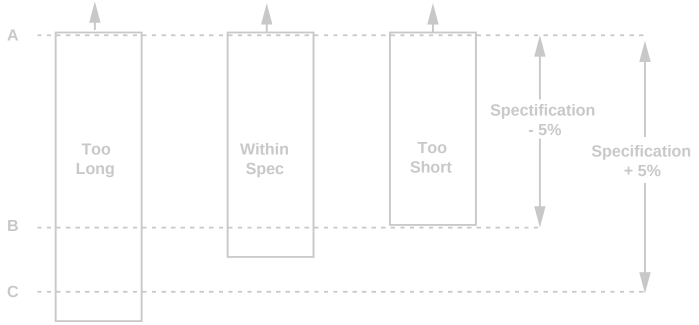


Where to place the light sensors A, B, and C to distinguish among the three cases?

Assume that A detects the leading edge of the rod on the conveyor

Combinational Logic Word Problems

Process Control Problem



A to B distance place apart at specification - 5%

A to C distance placed apart at specification +5%

Combinational Logic Word Problems

Process Control Problem

A	B	C	Function
0	0	0	X
0	0	1	X
0	1	0	X
0	1	1	X
1	0	0	too short
1	0	1	X
1	1	0	in spec
1	1	1	too long

Truth table and logic implementation now straightforward

"too long" = A B C
(all three sensors tripped)

"in spec" = A B C'
(first two sensors tripped)

Combinational Logic Word Problems

BCD to 7 Segment Display Controller

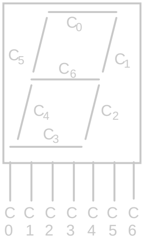
Understanding the problem:

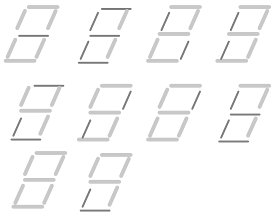
input is a 4 bit bcd digit

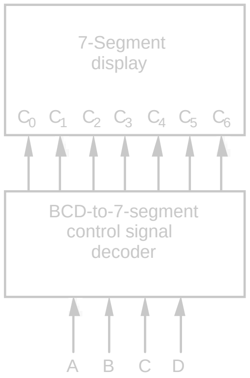
output is the control signals for the display

4 inputs A, B, C, D

7 outputs C0 ~ C6



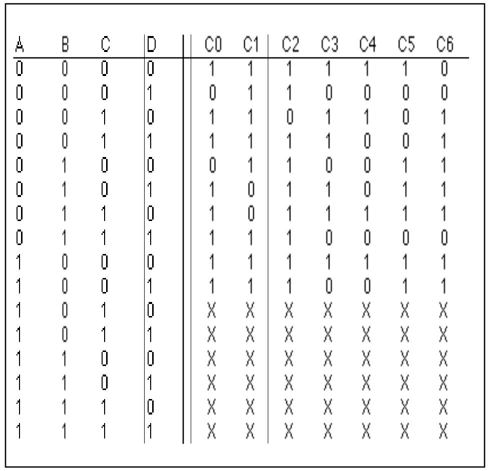




Block Diagram

Combinational Logic Word Problems

BCD to 7 Segment Display Controller



A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

Formulate the problem in terms of a truth table

Choose implementation target:

if ROM, we are done

don't cares imply PAL/PLA may be attractive

Follow implementation procedure:

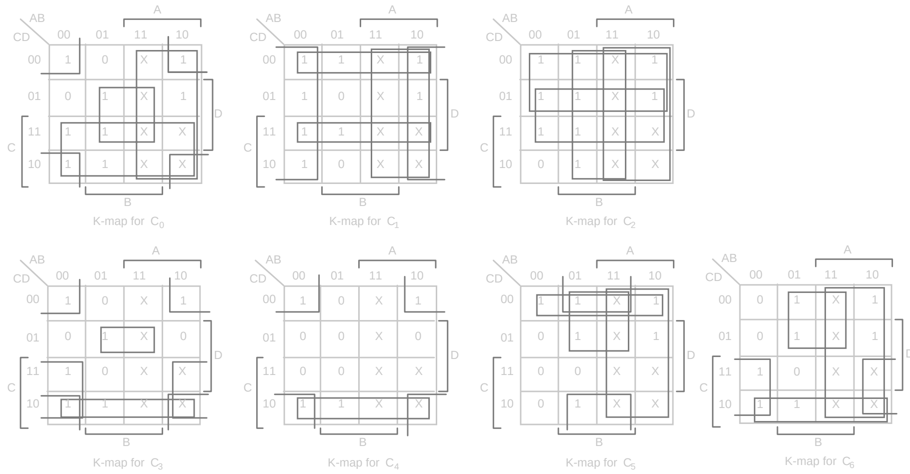
hand reduced K-maps

vs.

espresso

Combinational Logic Word Problems

BCD to 7 Segment Display Controller



K-map for C_0

CD \ AB	00	01	11	10
00	1	0	X	1
01	0	1	X	1
11	1	1	X	X
10	1	1	X	X

K-map for C_1

CD \ AB	00	01	11	10
00	1	1	X	1
01	1	0	X	1
11	1	1	X	X
10	1	0	X	X

K-map for C_2

CD \ AB	00	01	11	10
00	1	1	X	1
01	1	1	X	1
11	1	1	X	X
10	0	1	X	X

K-map for C_3

CD \ AB	00	01	11	10
00	1	0	X	1
01	0	1	X	0
11	1	0	X	X
10	1	1	X	X

K-map for C_4

CD \ AB	00	01	11	10
00	1	0	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

K-map for C_5

CD \ AB	00	01	11	10
00	1	1	X	1
01	0	1	X	1
11	0	0	X	X
10	0	1	X	X

K-map for C_6

CD \ AB	00	01	11	10
00	0	1	X	1
01	0	1	X	1
11	1	0	X	X
10	1	1	X	X

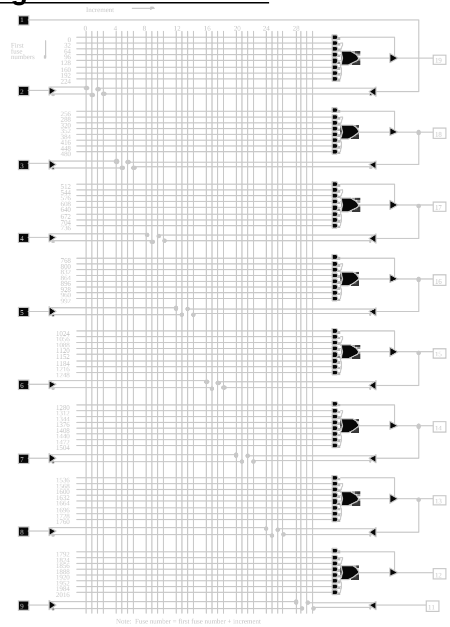
C0 = A + B D + C + B' D'
C1 = A + C' D' + C D + B'
C2 = A + B + C' + D

14 Unique Product Terms

C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

Combinational Logic Word Problems

BCD to 7 Segment Display Controller

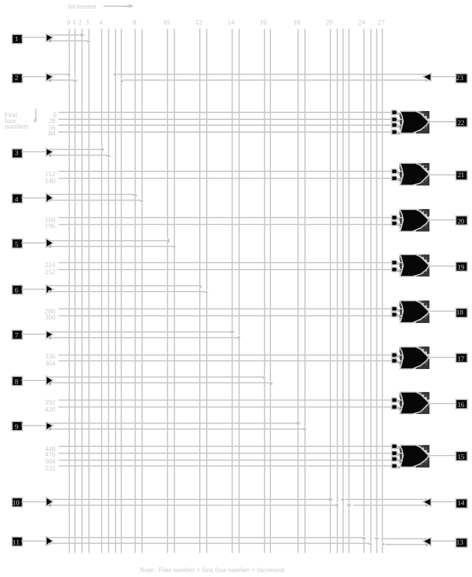


16H8PAL
Can Implement
the function

Combinational Logic Word Problems

BCD to 7 Segment Display Controller

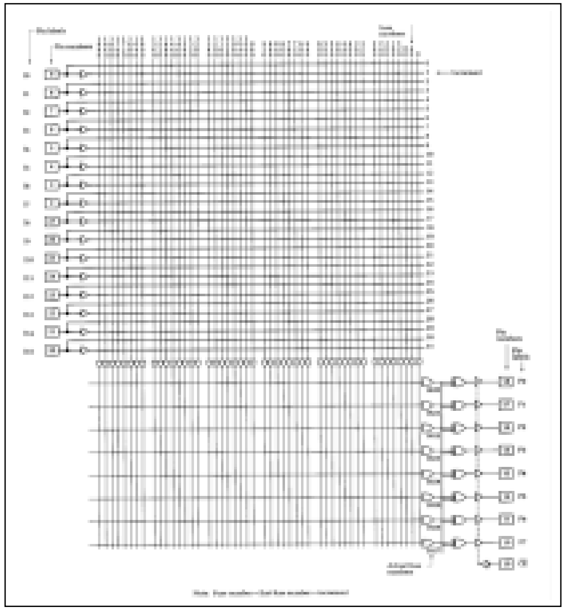
14H8PAL
Cannot Implement
the function



Combinational Logic Word Problems

BCD to 7 Segment Display Controller

F100 PAL programming map



Combinational Logic Word Problems

BCD to7 Segment Display Controller

```
.i 4
.o 7
.ilb a b c d
.ob c0 c1 c2 c3 c4 c5 c6
.p 16
0000 1111110
0001 0110000
0010 1101101
0011 1111001
0100 0110011
0101 1011011
0110 1011111
0111 1110000
1000 1111111
1001 1110011
1010 -------
1011 -------
1100 -------
1101 -------
1110 -------
1111 -------
.e
```

espresso input

```
.i 4
.o 7
.ilb a b c d
.ob c0 c1 c2 c3 c4 c5 c6
.p 9
-10-  0000001
-01-  0001001
-0-1  0110000
-101 1011010
--00  0110010
--11  1110000
-0-0  1101100
1---   1000011
-110 1011111
.e
```

espresso output

C0 = B C' D + C D + B' D' + B C D' + A
C1 = B' D + C' D' + C D + B' D'
C2 = B' D + B C' D + C' D' + C D + B C D'
C3 = B C' D + B' D + B' D' + B C D'
C4 = B' D' + B C D'
C5 = B C' D + C' D' + A + B C D'
C6 = B' C + B C' + B C D' + A

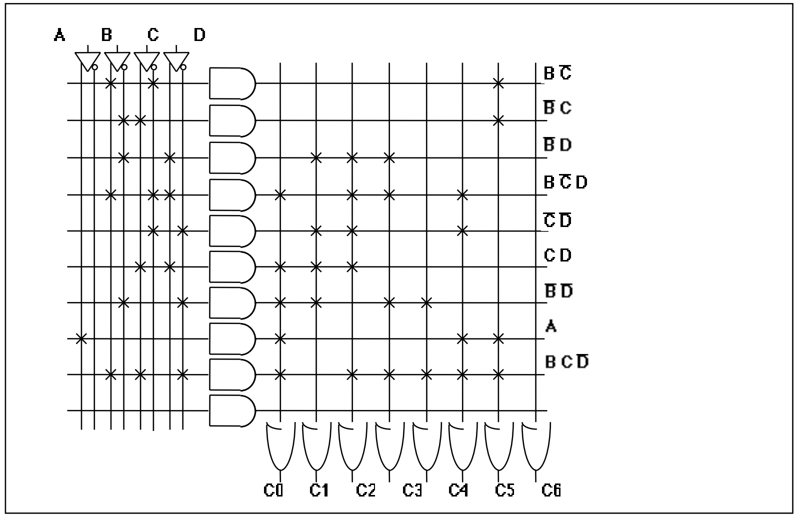
9 Unique Product Terms!

63 Literals, 20 Gates

Combinational Logic Word Problems

BCD to 7 Segment Display Controller

PLA Implementation



Combinational Logic Word Problems

BCD to7 Segment Display Controller

Multilevel Implementation

X = C' + D'

Y = B' C'

C0 = C3 + A' B X' + A D Y

C1 = Y + A' C5' + C' D' C6

C2 = C5 + A' B' D + A' C D

C3 = C4 + B D C5 + A' B' X'

C4 = D' Y + A' C D'

C5 = C' C4 + A Y + A' B X

C6 = A C4 + C C5 + C4' C5 + A' B' C

52 literals

33 gates

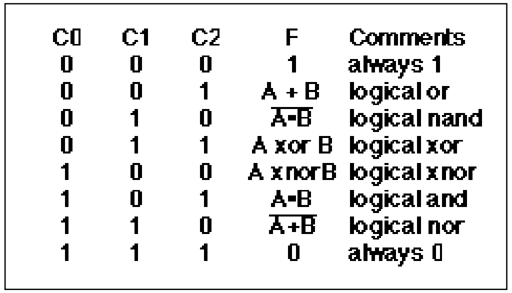
Ineffective use of don't cares

Combinational Logic Word Problems

Logical Function Unit

Statement of the Problem:

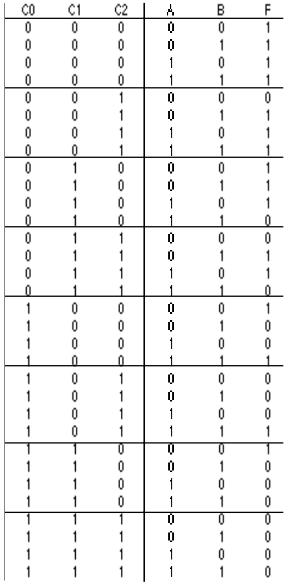
3 control inputs: C0, C1, C2
2 data inputs: A, B
1 output: F



C0	C1	C2	F	Comments
0	0	0	1	always 1
0	0	1	A + B	logical or
0	1	0	$\overline{A \cdot B}$	logical nand
0	1	1	A xor B	logical xor
1	0	0	A xnor B	logical xnor
1	0	1	A•B	logical and
1	1	0	$\overline{A+B}$	logical nor
1	1	1	0	always 0

Combinational Logic Word Problems

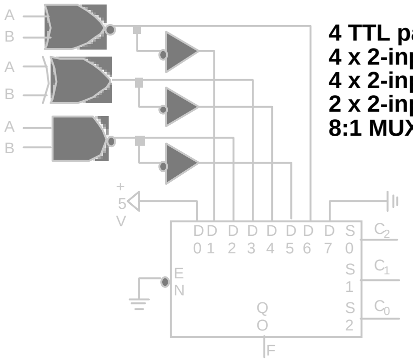
Logical Function Unit



C0	C1	C2	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Formulate as a truth table

Choose implementation technology
5-variable K-map
espresso
multiplexor implementation

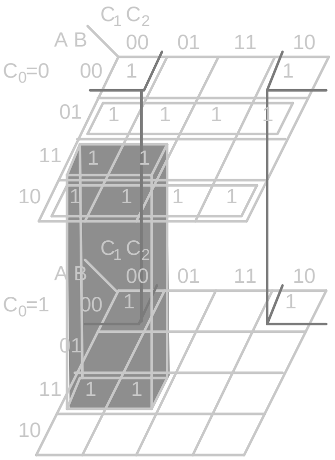


4 TTL packages:
4 x 2-input NAND
4 x 2-input NOR
2 x 2-input XOR
8:1 MUX

Combinational Logic Word Problems

Logical Function Unit

Follow implementation procedure



F = C2' A' B' + C0' A B'
+ C0' A' B + C1' A B

5 gates, 5 inverters

Also four packages:
4 x 3-input NAND
1 x 4-input NAND

Alternative: 32 x 1-bit ROM

single package

Combinational Logic Word Problems

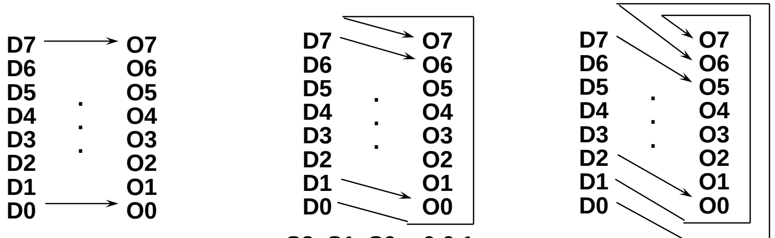
8-Input Barrel Shifter

Specification:

Inputs: D7, D6, ~ D0
Outputs: O7, O6, ~ O0
Control: S2, S1, S0

shift input the specified number of positions to the right

Understand the problem:



S2, S1, S0 = 0 0 0

S2, S1, S0 = 0 0 1

S2, S1, S0 = 0 1 0

Combinational Logic Word Problems

8-Input Barrel Shifter

Function Table

S2	S1	S0	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	D6	D5	D4	D3	D2	D1	D0	D7
0	1	0	D5	D4	D3	D2	D1	D0	D7	D6
0	1	1	D4	D3	D2	D1	D0	D7	D6	D5
1	0	0	D3	D2	D1	D0	D7	D6	D5	D4
1	0	1	D2	D1	D0	D7	D6	D5	D4	D3
1	1	0	D1	D0	D7	D6	D5	D4	D3	D2
1	1	1	D0	D7	D6	D5	D4	D3	D2	D1

Boolean equations

O7 = S2' S1' S0' D7 + S2' S1' S0 D6 + ... + S2 S1 S0 D0

O6 = S2' S1' S0' D6 + S2' S1' S0 D5 + ... + S2 S1 S0 D7

O5 = S2' S1' S0' D5 + S2' S1' S0 D4 + ... + S2 S1 S0 D6

O4 = S2' S1' S0' D4 + S2' S1' S0 D3 + ... + S2 S1 S0 D5

O3 = S2' S1' S0' D3 + S2' S1' S0 D2 + ... + S2 S1 S0 D4

O2 = S2' S1' S0' D2 + S2' S1' S0 D1 + ... + S2 S1 S0 D3

O1 = S2' S1' S0' D1 + S2' S1' S0 D0 + ... + S2 S1 S0 D2

O0 = S2' S1' S0' D0 + S2' S1' S0 D7 + ... + S2 S1 S0 D1

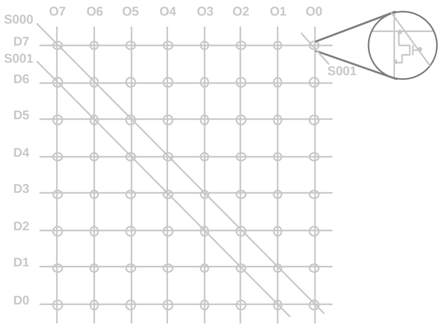
Combinational Logic Word Problems

8-Input Barrel Shifter

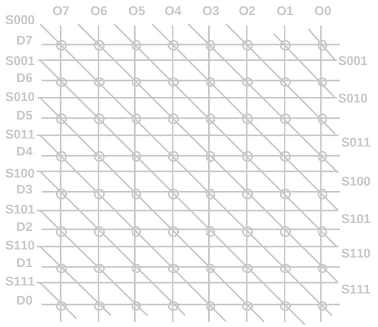
Straightforward gate logic implementation OR

8 by 8:1 multiplexer (wiring mess!) OR

Switch logic



Crosspoint switches



Fully Wired crosspoint switch

Chapter Review

- ***Non-Simple Gate Logic Building Blocks:***

  PALs/PLAs

  Multiplexers/Selecters

  Decoders

  ROMs

  Tri-state, Open Collector

- ***Combinational Word Problems:***

  Understand the Problem

  Formulate in terms of a Truth Table

  Choose implementation technology

  Implement by following the design procedure