Week 3 - T1 2020

Combinational Circuit Design

ELEC2141: Digital Circuit Design

Summary

Standard/canonical form of Boolean expression

Minterms and maxterms

Sum of products/product of sums

Gate input cost

Karnaugh maps

Multi-level optimization

NAND/NOR/XOR gate
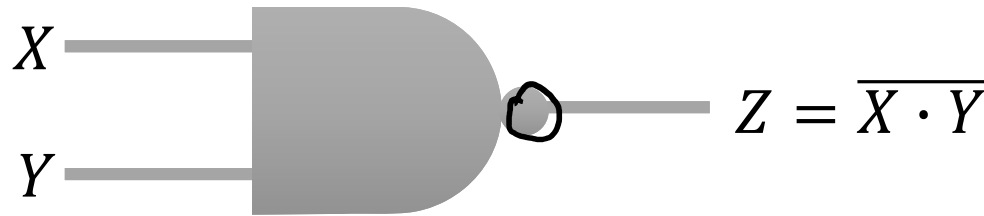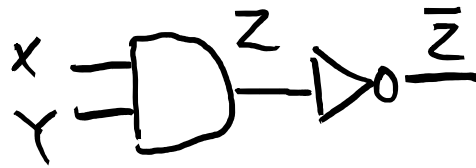
NAND/NOR gate only implementation

3 state buffers

Overview

NAND/NOR gate only implementation (revision)

Mon

Circuit design

Hierarchical design

Thur

Decoders

Encoders

Multiplexers

Reading: Mano - Chapter 3, 3.1-3.7

UNSW
SYDNEY

NAND gate

$$X \overset{Z}{\rightarrow} \overline{Z}$$

$$Z = \overline{X \cdot Y}$$

X	Y	$Z = \overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0

AND
X·Y
0
0
0
1

NAND represents NOT-AND, i.e. the AND function with a NOT applied to the result
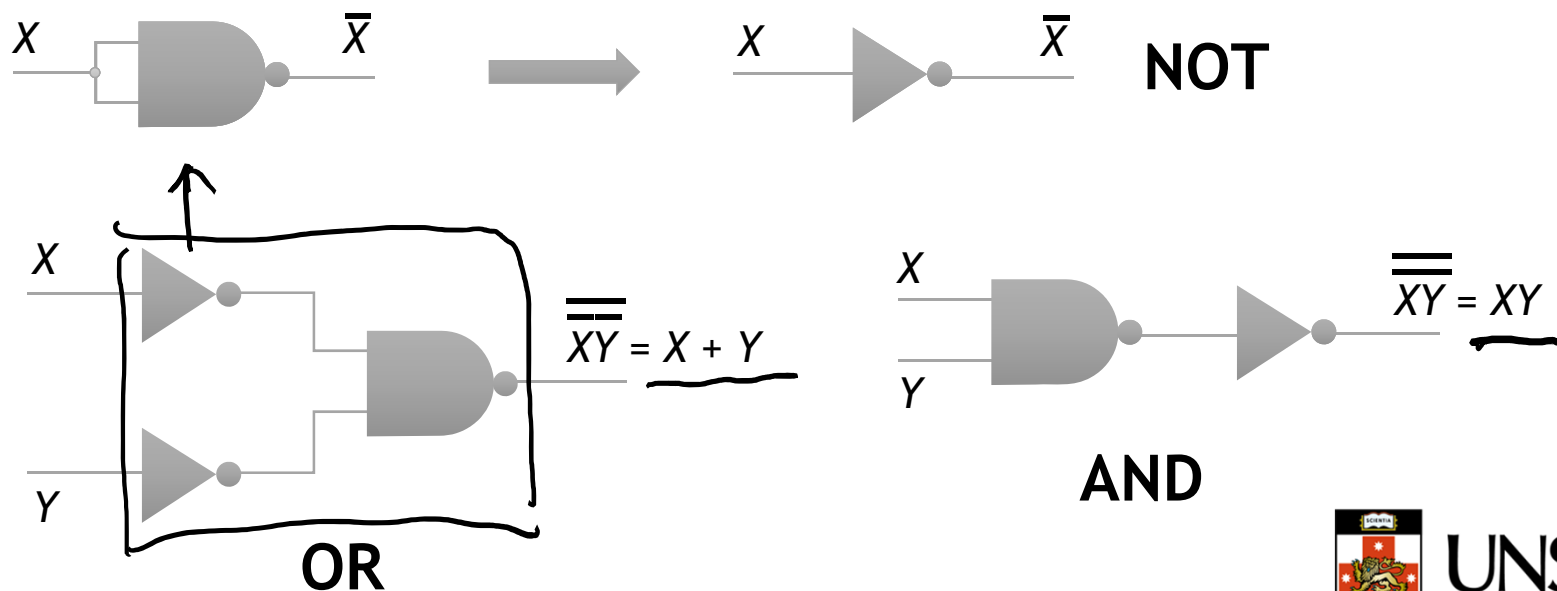
By applying DeMorgan's Theorem, the NAND function can also be expressed as

$$Z = \overline{X \cdot Y} = \overline{X} + \overline{Y}$$

4

NAND as a universal gate

The NAND gate is the natural implementation for CMOS technology in terms of chip area and speed

The NAND gate is a *universal gate* – a gate type that can implement any Boolean function

NOT

OR

AND

$\overline{XY} = X + Y$

$\overline{\overline{XY}} = XY$

5

NAND only implementation

As the NAND gate is a universal gate, all other gates can be replaced with NAND gates to have NAND only implementations

Use the alternative symbols below to change from AND-OR circuit to a NAND circuit

X
Y
Z
$\overline{XYZ}$

(a) AND - NOT

X
Y
Z
$\overline{X} + \overline{Y} + \overline{Z} = \overline{XYZ}$

(b) NOT – OR

X $\overline{X}$
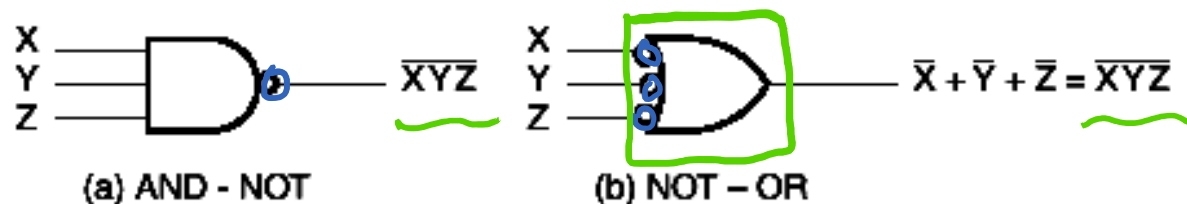
Multi-level NAND circuits

1. Find the simplified SOP circuit

2. Convert all AND gates to NAND gates with AND-NOT graphic symbols

3. Convert all OR gates to NAND gates with NOT-OR graphic symbols

4. Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance
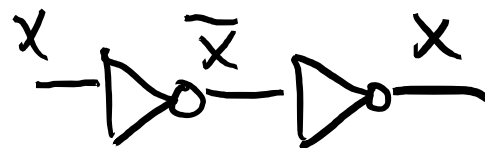
Multi-level NAND circuits

$$F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E$$

NOR gate

X ——
Y ——

$Z = \overline{X + Y}$
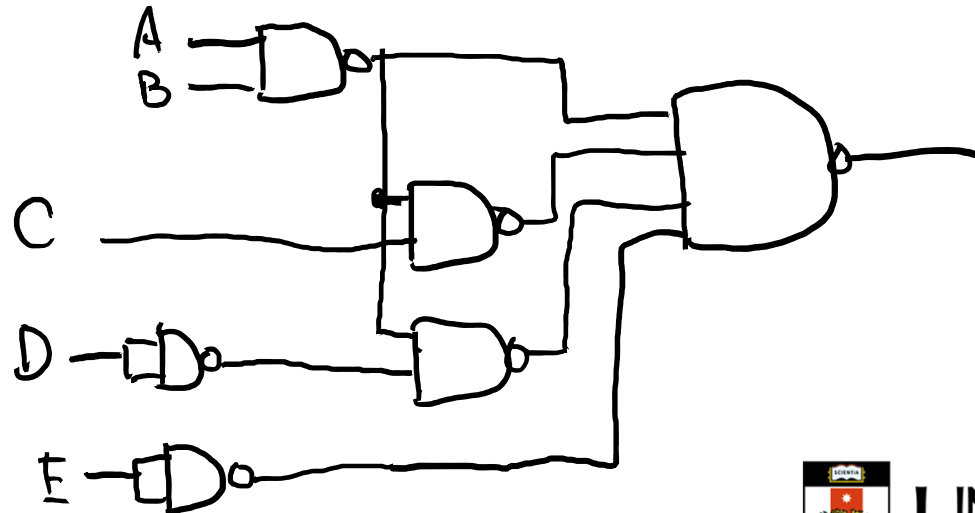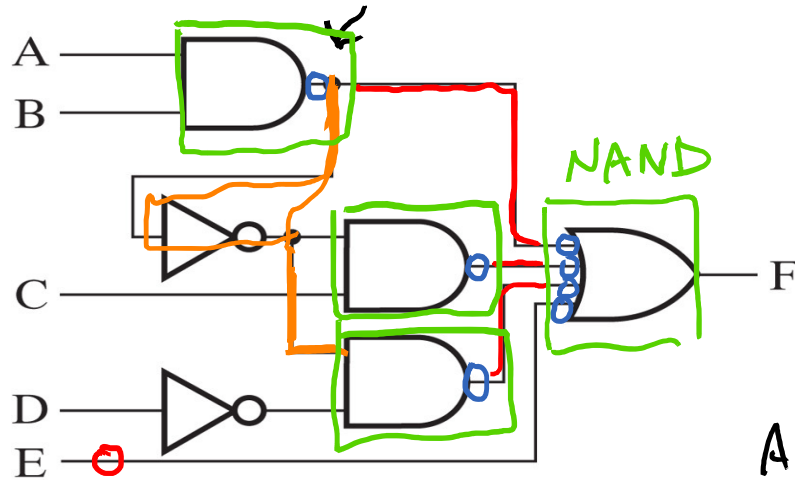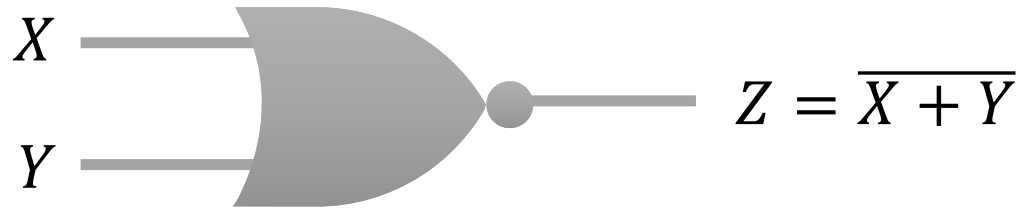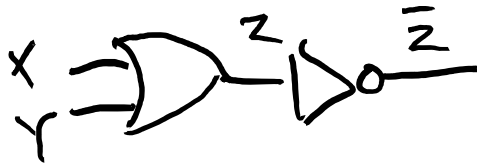
X	Y	$Z = \overline{X+Y}$
0	0	1
0	1	0
1	0	0
1	1	0

$x + y$

0
1
1
1

NOR represents NOT-OR, i.e. the OR function with a NOT applied to the result

The NOR gate is another *universal gate*

By applying DeMorgan's Theorem, the NOR function can also be expressed as

$$Z = \overline{X+Y} = \overline{X}\,\overline{Y}$$

UNSW
SYDNEY

NOR as a universal gate

The NOR gate is a *universal gate* – a gate type
that can implement any Boolean function

X $\overline{X}$ X $\overline{X}$ **NOT**

X
Y
$\overline{\overline{X+Y}} = X+Y$

OR

X
Y
$\overline{\overline{X}+\overline{Y}} = XY$

AND

10
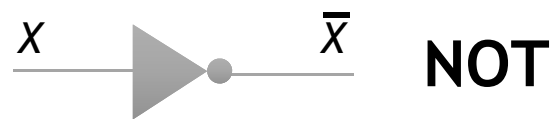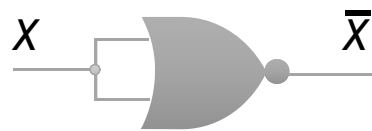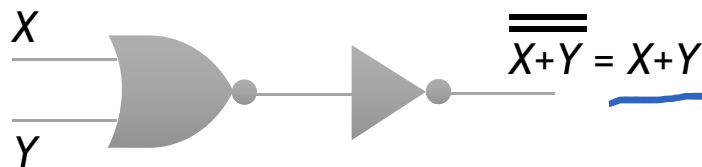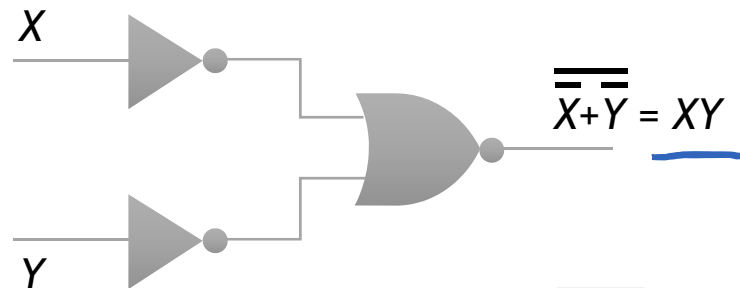
NOR only implementation

Similarly as the NOR gate is a universal gate, all other gates can be replaced with NOR gates to have NOR only implementations

Use the alternate symbols below from AND-OR circuit obtain the NOR only implementation

$$\overline{X+Y+Z}$$

(a) OR – NOT

$$\overline{X}\,\overline{Y}\,\overline{Z} = \overline{X+Y+Z}$$

(b) NOT – AND

11
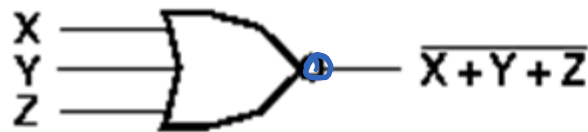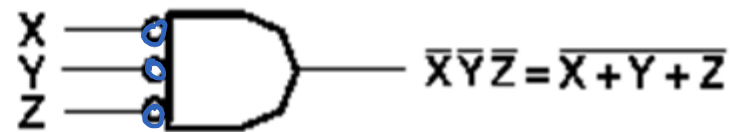
Multi-level NOR circuits

1. Find the simplified SOP circuit

2. Convert all OR gates to NOR gates with OR-NOT graphic symbols

3. Convert all AND gates to NOR gates with NOT-AND graphic symbols

4. Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance

Multi-level NOR circuits

$$F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E$$
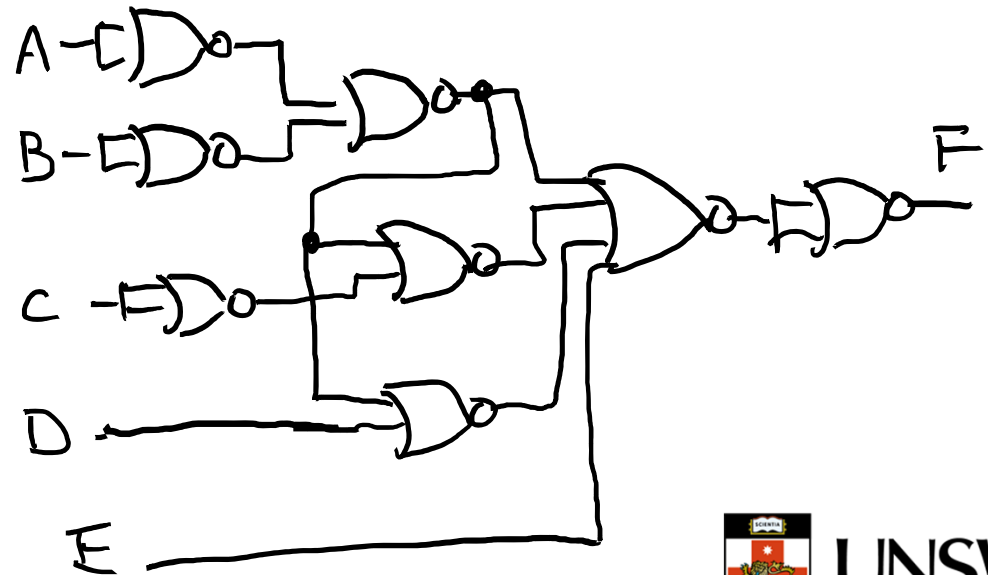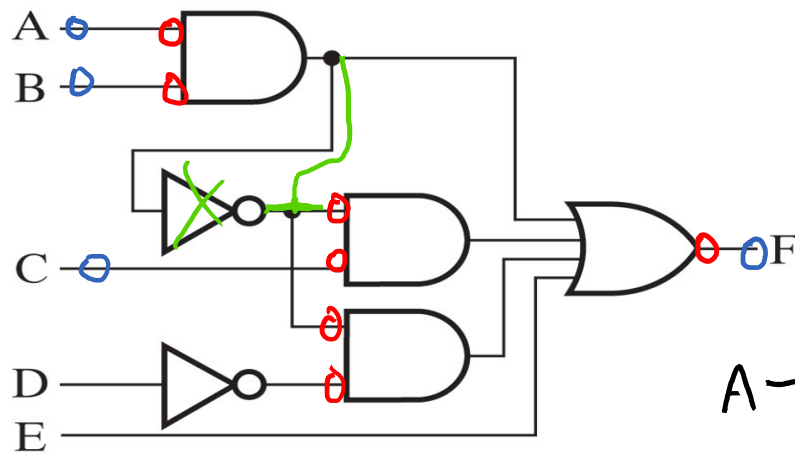
Combinational Logic Design

1) The design of the combinational logic circuits starts from formulating the problem given a set of requirements or specifications *Boolean Expressions*

Once the problem is formulated, the solution is optimised to come up with the simplified to come up with the least cost expression

This expression is then implemented using hardware

UNSW
SYDNEY

Design procedure

1. Specification: produce textual or HDL description of the desired circuit that includes respective symbols or names for inputs and outputs

2. Formulation: convert the specification into a Boolean expression and truth table

3. Optimization: produce a simplified Boolean expression using algebraic manipulation, K-map method, or computer based optimization tools that provides a logic diagram or a netlist using AND, OR and NOT gates

Design procedure

4. **Technology mapping:** transform the logic diagram or netlist based on the available implementation technology

5. **Verification:** to determine the correctness of the design using manual logic analysis and/or computer simulation based logic analysis

Design BCD to excess-3 code converter

MSB A   →   BCD to Excess-3 code converter   →   W MSB

B → X

C → Y

LSB D → Z LSB

Input in BCD   Output in Excess-3

(handwritten annotation)

binary   decimal

0001   1

+0011   +3

0100   4
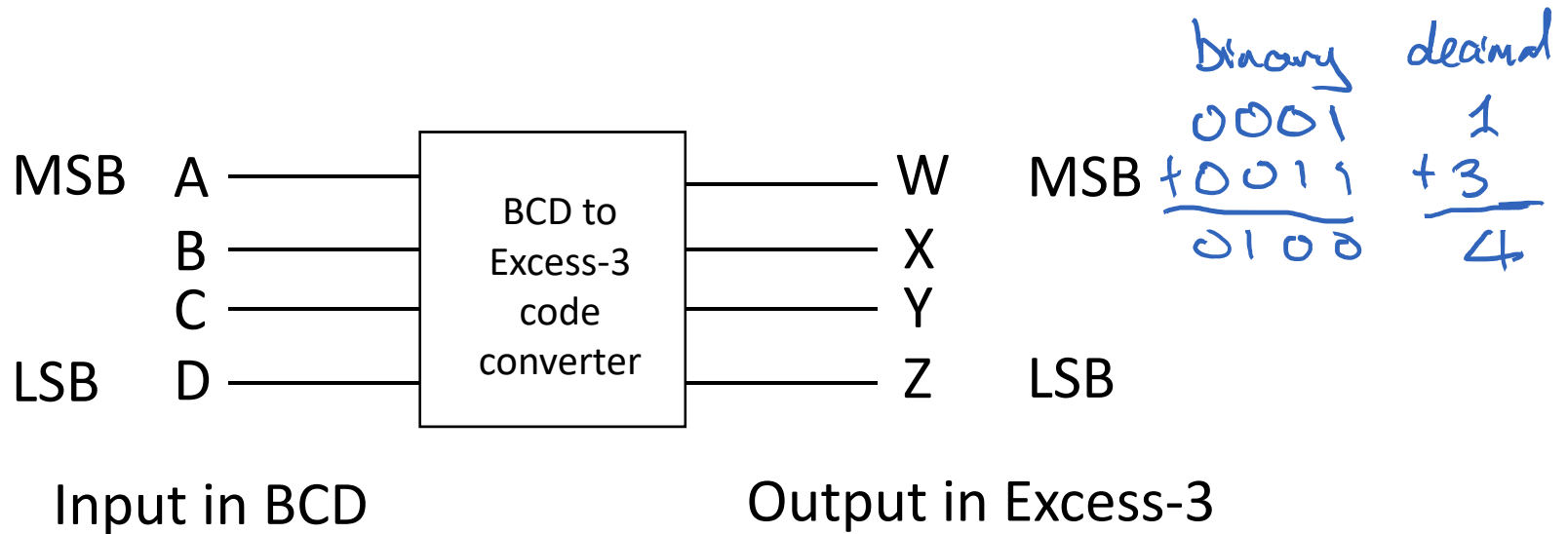
Specification: the excess-3 code for a decimal digit is the binary combination corresponding to the decimal digit plus 3

Design BCD to excess-3 code converter

Formulation

Decimal Digit	Input BCD				Output Excess-3 ↓ ↓ ↓ ↓			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

10~15

X X X X

Design BCD to excess-3 code converter

Optimization for *W*

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$W = A + BC + BD$

Design BCD to excess-3 code converter

Optimization for *X*

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$$X = \overline{B}\,\overline{C}\,\overline{D} + \overline{B}D + \overline{B}C$$

Design BCD to excess-3 code converter

Optimization for Y

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$$Y = CD + C\bar{D}$$

21

Design BCD to excess-3 code converter

Optimization for Z

Decimal Digit	Input BCD				Output Excess-3			↓
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$Z = \bar{D}$

Design BCD to excess-3 code converter

The two level implementation obtained from
the map is thus

$$W = A + BC + BD$$
$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$
$$Y = CD + \bar{C}\bar{D}$$
$$Z = \bar{D}$$

with gate input costs of 26

	literals	terms	comp	total
	5	2	0	7
	7	3	3	10
	4	2	2	6
distinct comp :	B C D			3
				26

UNSW
SYDNEY

Design BCD to excess-3 code converter

Optimization: Multi-level implementation

②

$$T_1 = C + D \qquad \overline{T_1} = \overline{C + D} = \overline{C}\,\overline{D}$$

$$W = A + BC + BD = A + B(C + D) = A + BT_1 \boxed{\begin{array}{l} L:3 \\ T:1 \end{array}}$$

$$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D} = \overline{B}(C + D) + B\overline{C}\,\overline{D}$$

$$= \overline{B}T_1 + B\overline{T_1} \boxed{\begin{array}{l} L:4 \\ T:2 \end{array}}$$

$$Y = CD + \overline{C}\,\overline{D} = CD + \overline{T_1} \boxed{\begin{array}{l} L:3 \\ T:1 \end{array}} \qquad \text{comp. } B, D, T = \boxed{3}$$

$$Z = \overline{D}$$
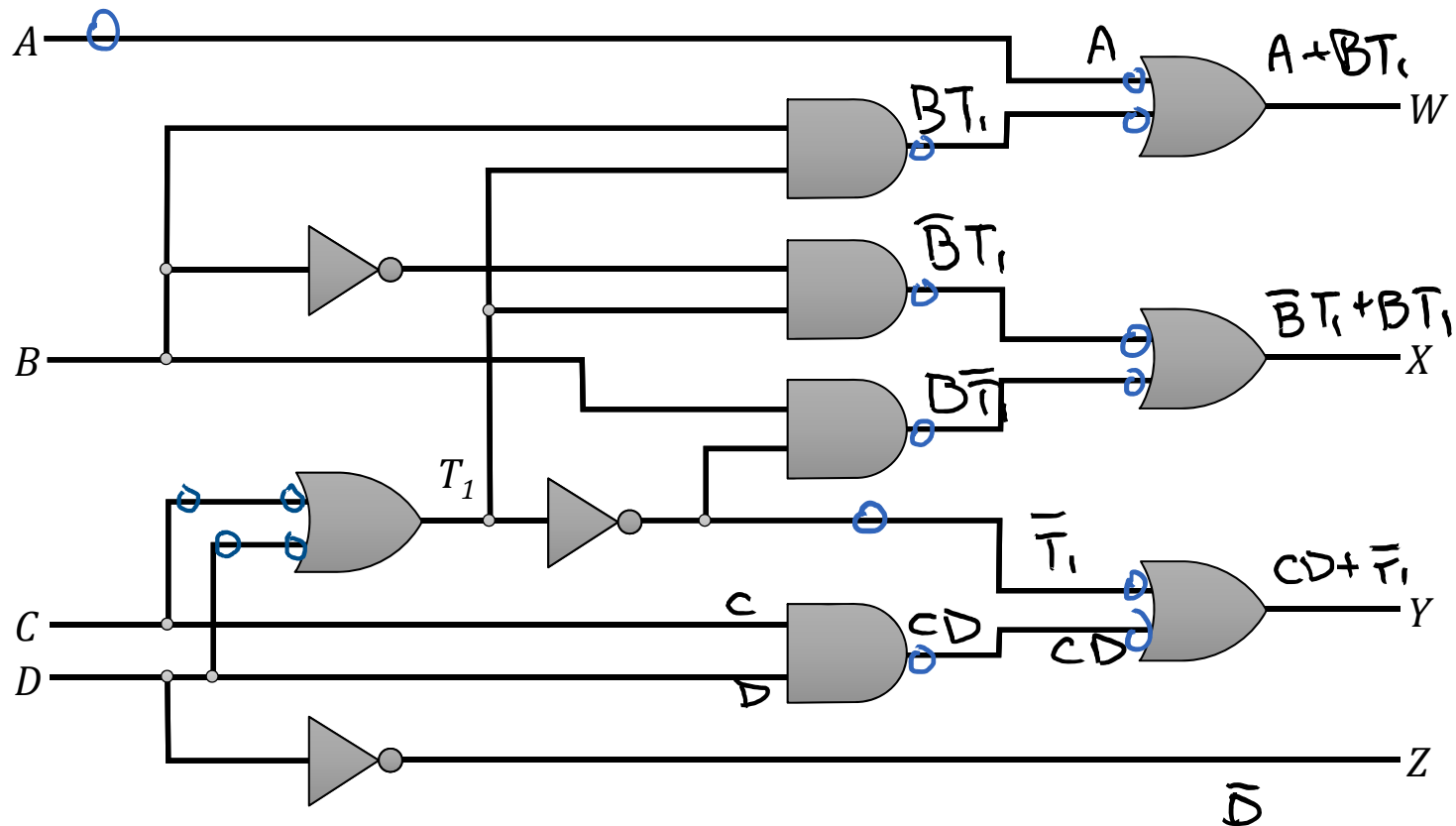
$$2 + 4 + 6 + 4 + 3$$

will reduce gate input costs to 19

Design BCD to excess-3 code converter

Optimization: Logic diagram

$$T_1 = C + D \qquad \overline{T}_1 = \overline{C}\overline{D}$$
$$W = A + BT_1 \qquad Y = CD + \overline{T}_1$$
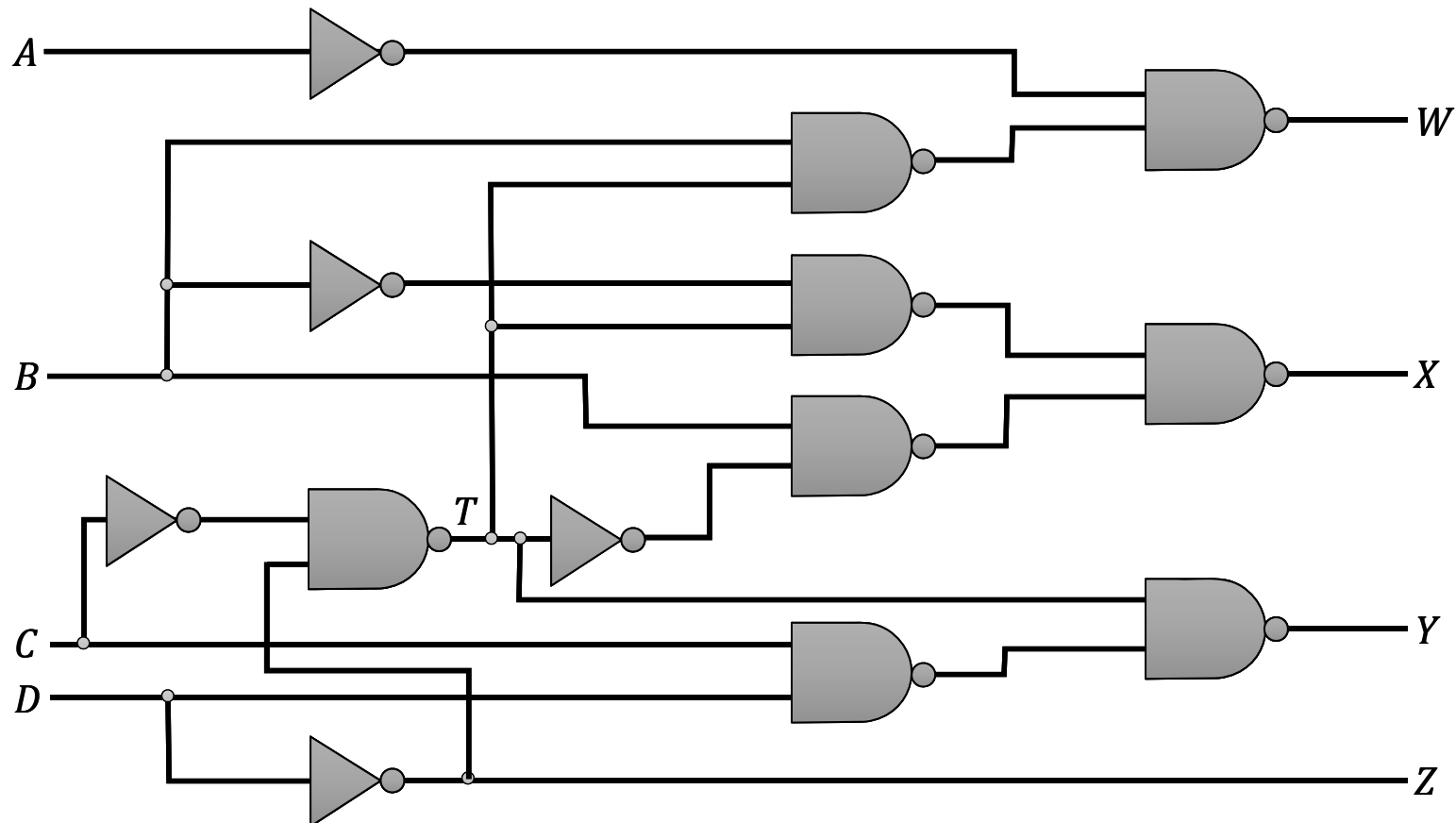$$X = \overline{B}T_1 + B\overline{T}_1 \quad Z = \overline{D}$$

Design BCD to excess-3 code converter

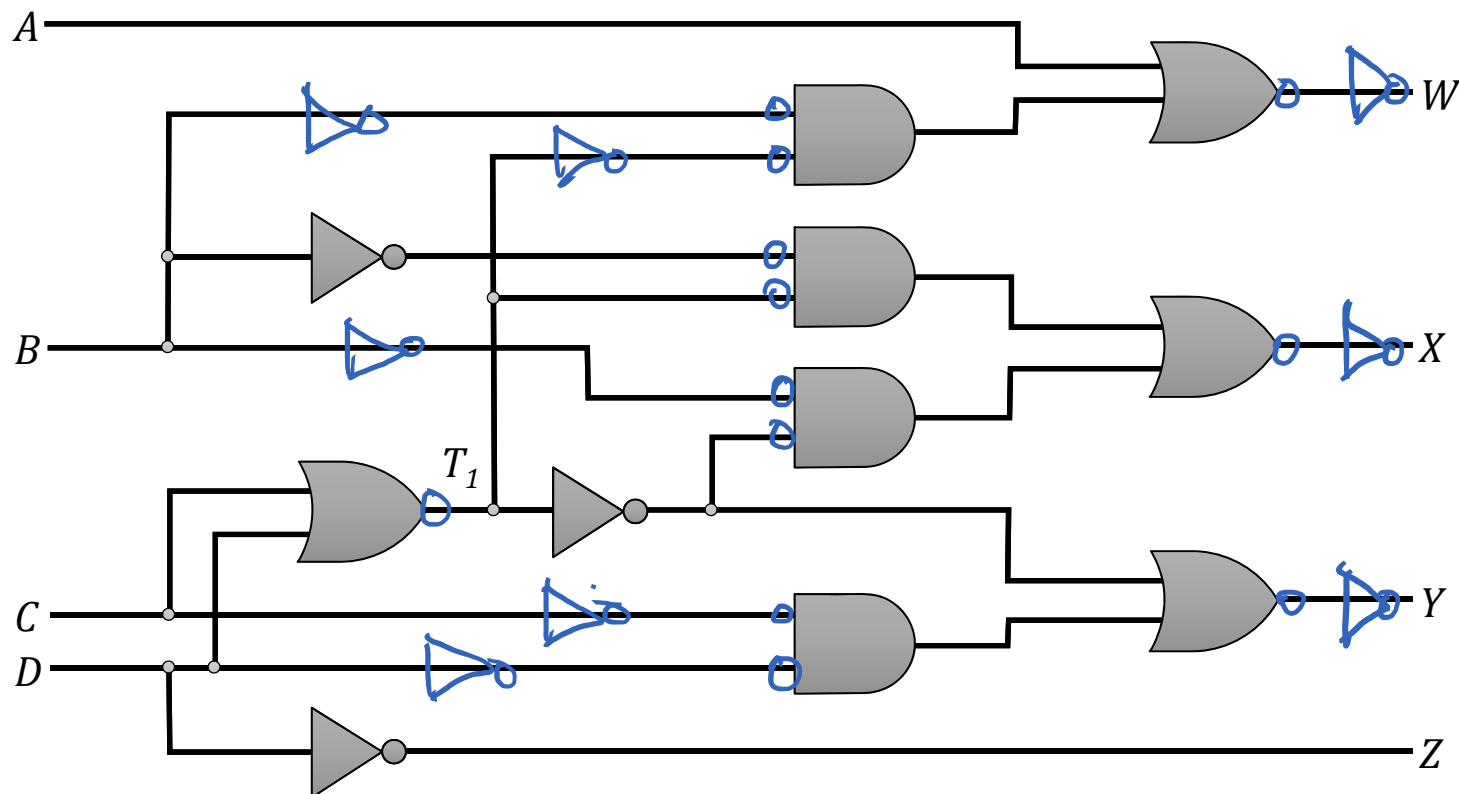Technology Implementation: NAND ONLY implementation
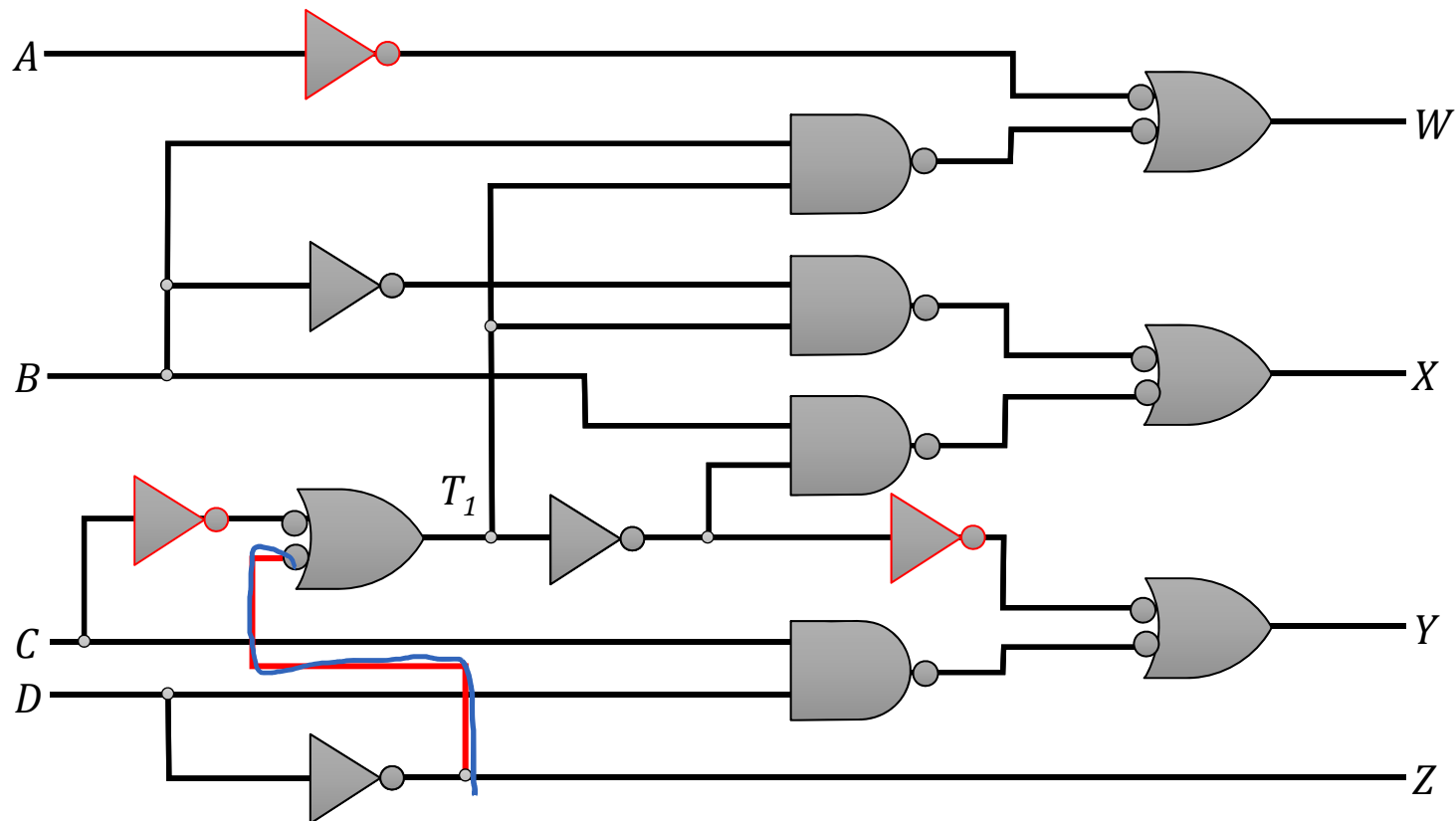
Design BCD to excess-3 code converter
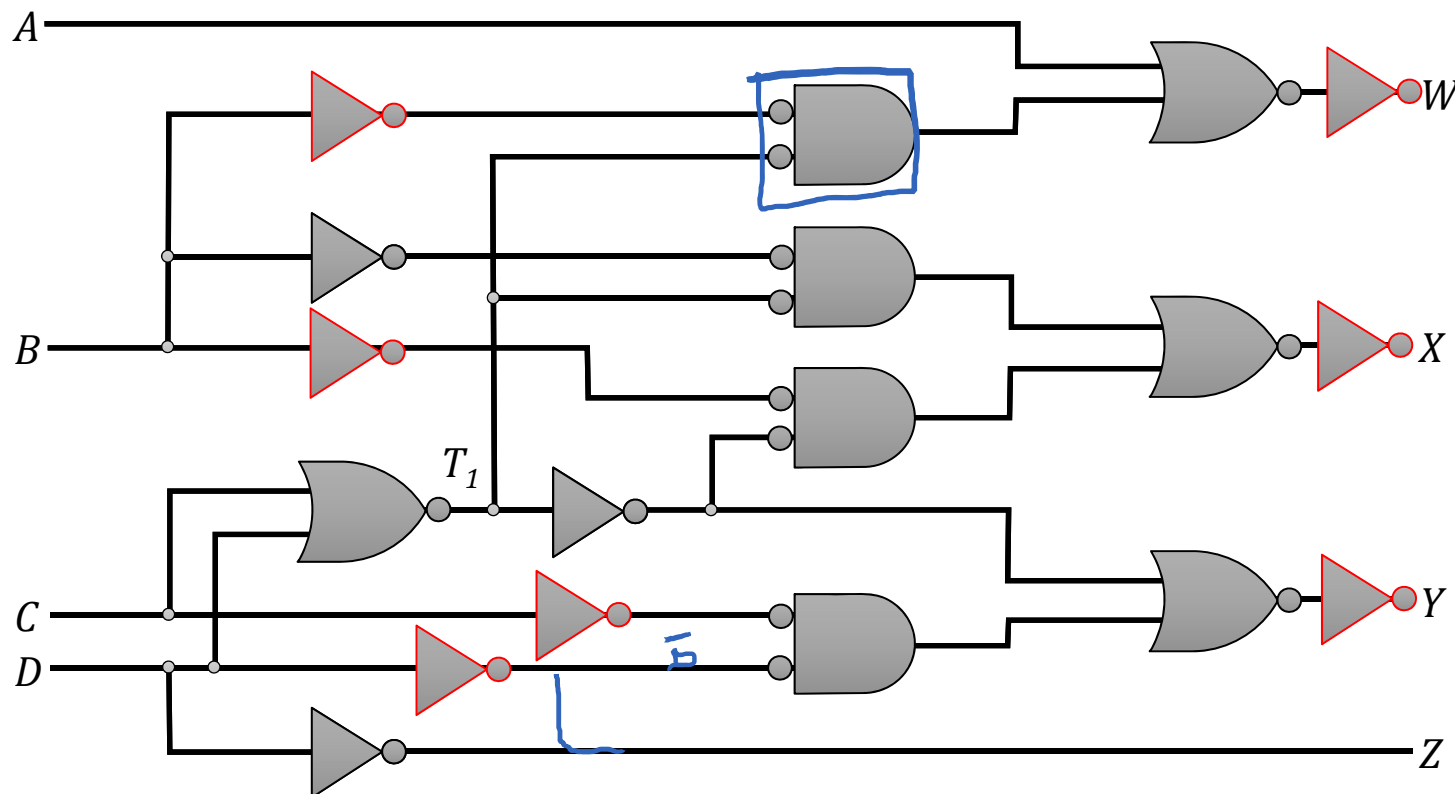
Technology Implementation: NAND ONLY logic diagram

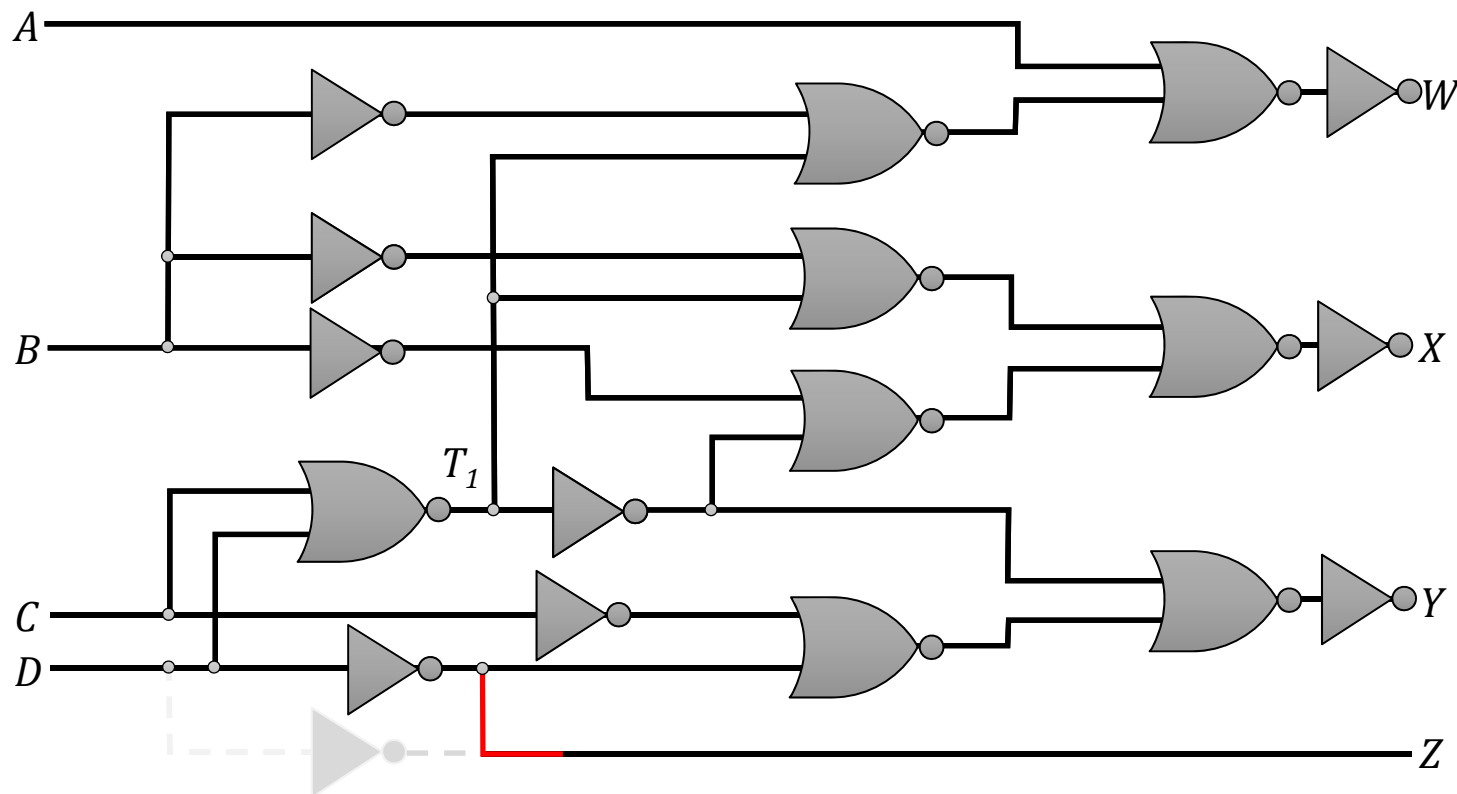# Design BCD to excess-3 code converter

Technology Implementation: NOR ONLY implementation

Design BCD to excess-3 code converter

Technology Implementation: NOR ONLY implementation
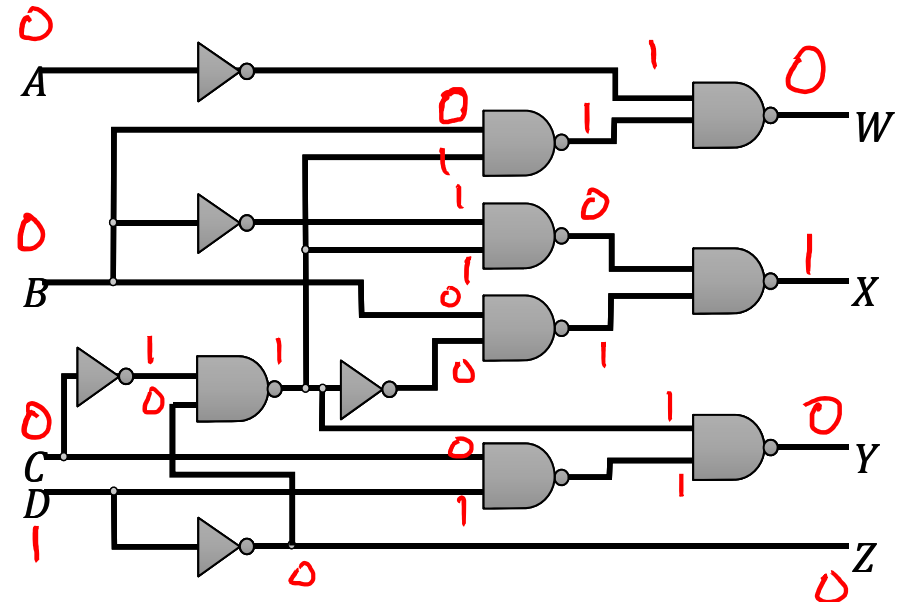
Design BCD to excess-3 code converter

Technology Implementation: NOR ONLY implementation

Design BCD to excess-3 code converter

Verification: Manual

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1				
4	0	1	0	0	0	1	1	1
5	0	1	0	1				
6	0	1	1	0	1	0	0	1
7	0	1	1	1				
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

31

Design BCD to excess-3 code converter

Verification: CAD
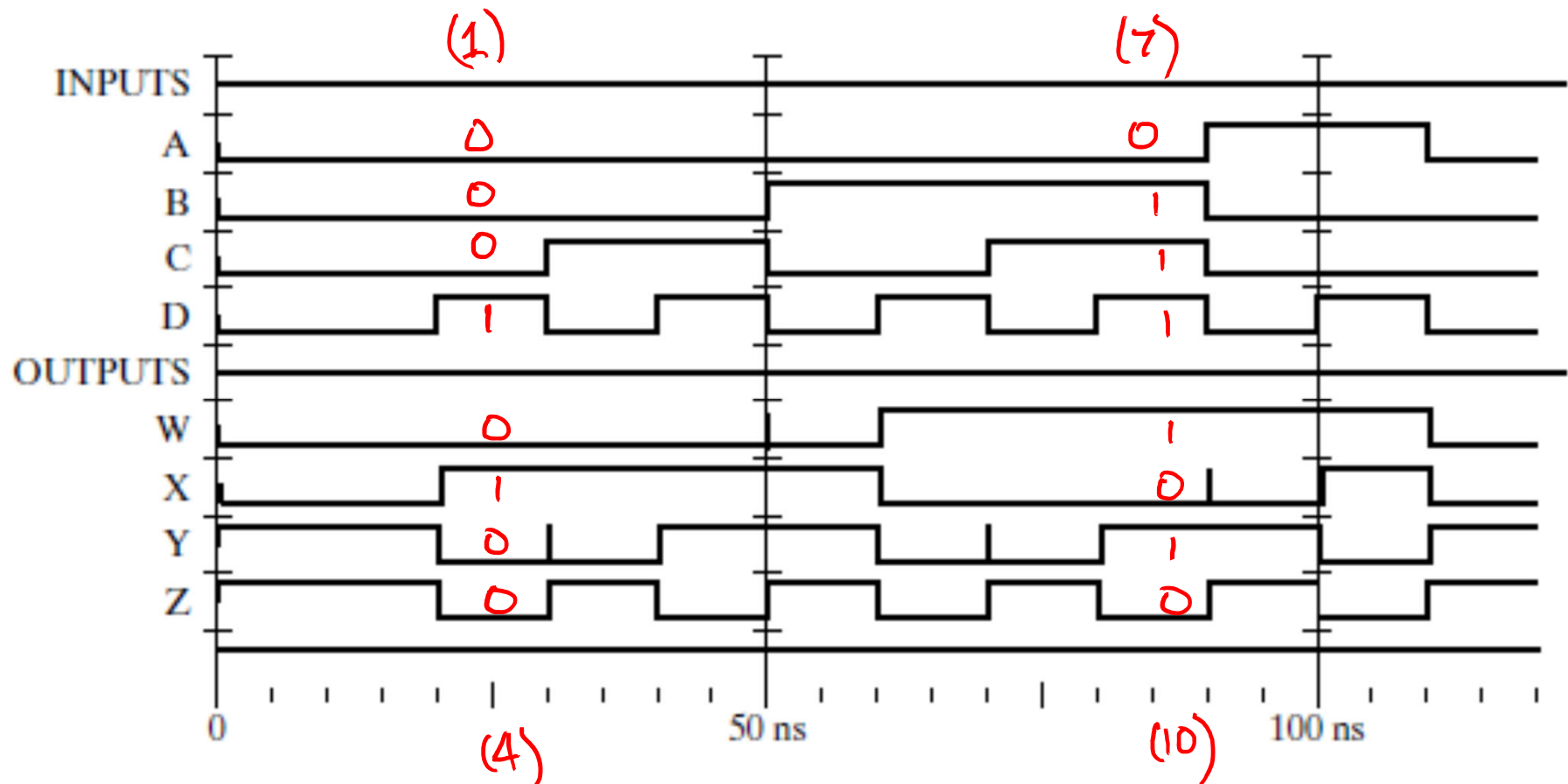
Hierarchical design

Complex or large digital circuits will be broken down into smaller and simpler circuits called blocks.

The blocks are interconnected to form the complex circuit.

The circuit formed by interconnecting the blocks obeys the initial circuit specification.

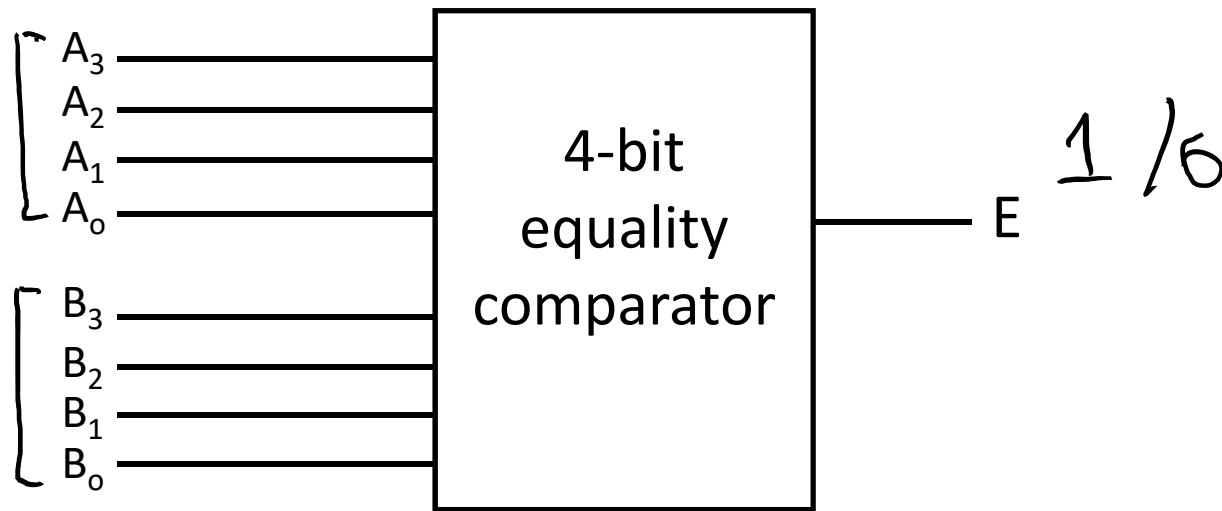A block can further be broken down if found large to be designed as a single entity.

Design a 4-bit equality comparator

Specification: An equality comparator that compares two binary vectors to determine if they are equal or not

Inputs: A(3:0) and B(3:0)

Output of the circuit is a single bit variable E

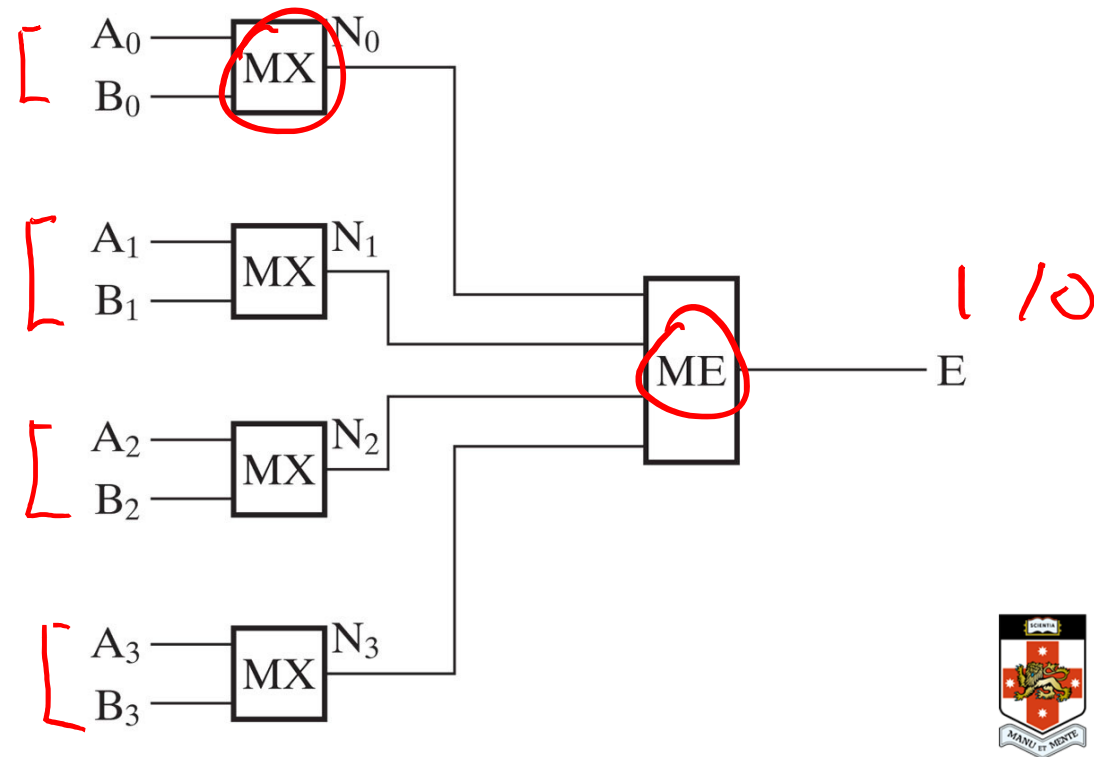E = 1, if A and B are equal, E = 0 if A and B are unequal
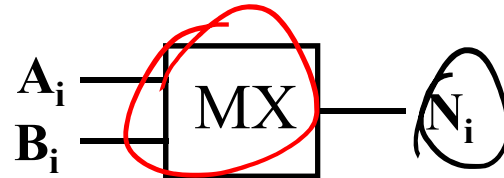
Design a 4-bit equality comparator

Truth tables and K-map are not much of a help because need a truth table with 256 rows and K-map for eight variables

Will use hierarchical approach to break function into four identical blocks of MX, one-bit comparator, and additional circuit ME that combines all four outputs into a single output E

Design a 1-bit equality comparator

A_i —
B_i —
MX — N_i

Specification: A 1 bit Equality comparator compares if two bits are the same or not

Inputs: A_i and B_i

Output: N_i, where $N_i = 1$ if $A_i = B_i$

Formulation: Draw truth table and find Boolean function

A_i	B_i	N_i
0	0	1
0	1	0
1	0	0
1	1	1

UNSW
SYDNEY

Design a 1-bit equality comparator

Optimization: using two variable K-map

N_i

$A \backslash B$	B 0	B 1
0	1	
1		1

$$N_i = \overline{A_i}\,\overline{B_i} + A_i B_i$$

Logic Diagram
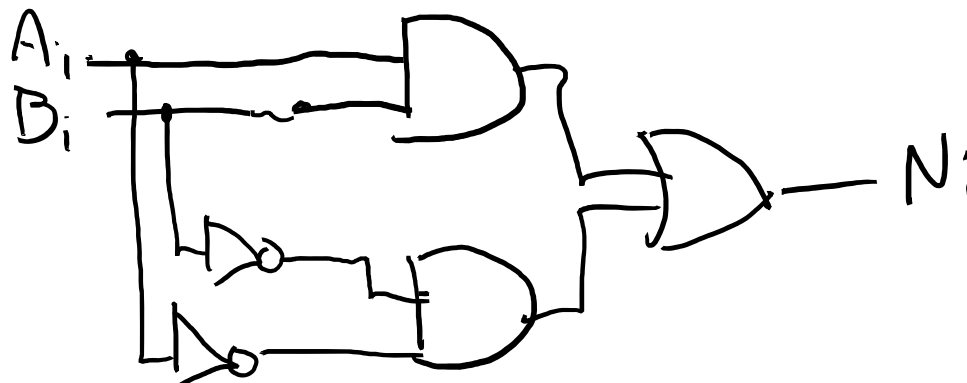
Design ME circuit

ME combines the outputs of four 1-bit comparator

$$A_0 \quad A_1 \quad A_2 \quad A_3$$

$$\| \quad \| \quad \| \quad \|$$

$$B_0 \quad B_1 \quad B_2 \quad B_3$$

$$\| \quad \| \quad \| \quad \|$$
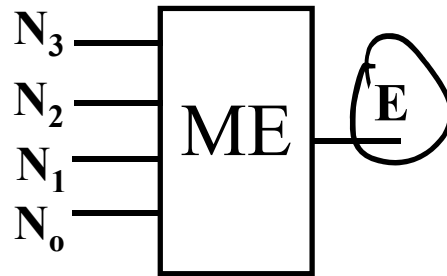
$$1 \quad 1 \quad 1 \quad 1$$

Specification: E is 1 if all four inputs, N_0, N_1, N_2 and N_3 are 1, else E is 0

Formulation: The resultant Boolean function is

$$E = N_0 \, N_1 \, N_2 \, N_3$$

Design ME circuit

Optimization: ME cannot be optimized further as there is only one "1" in a four variable K-map. Hence,
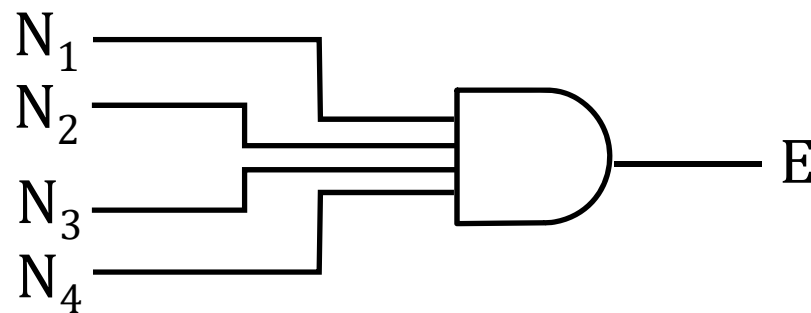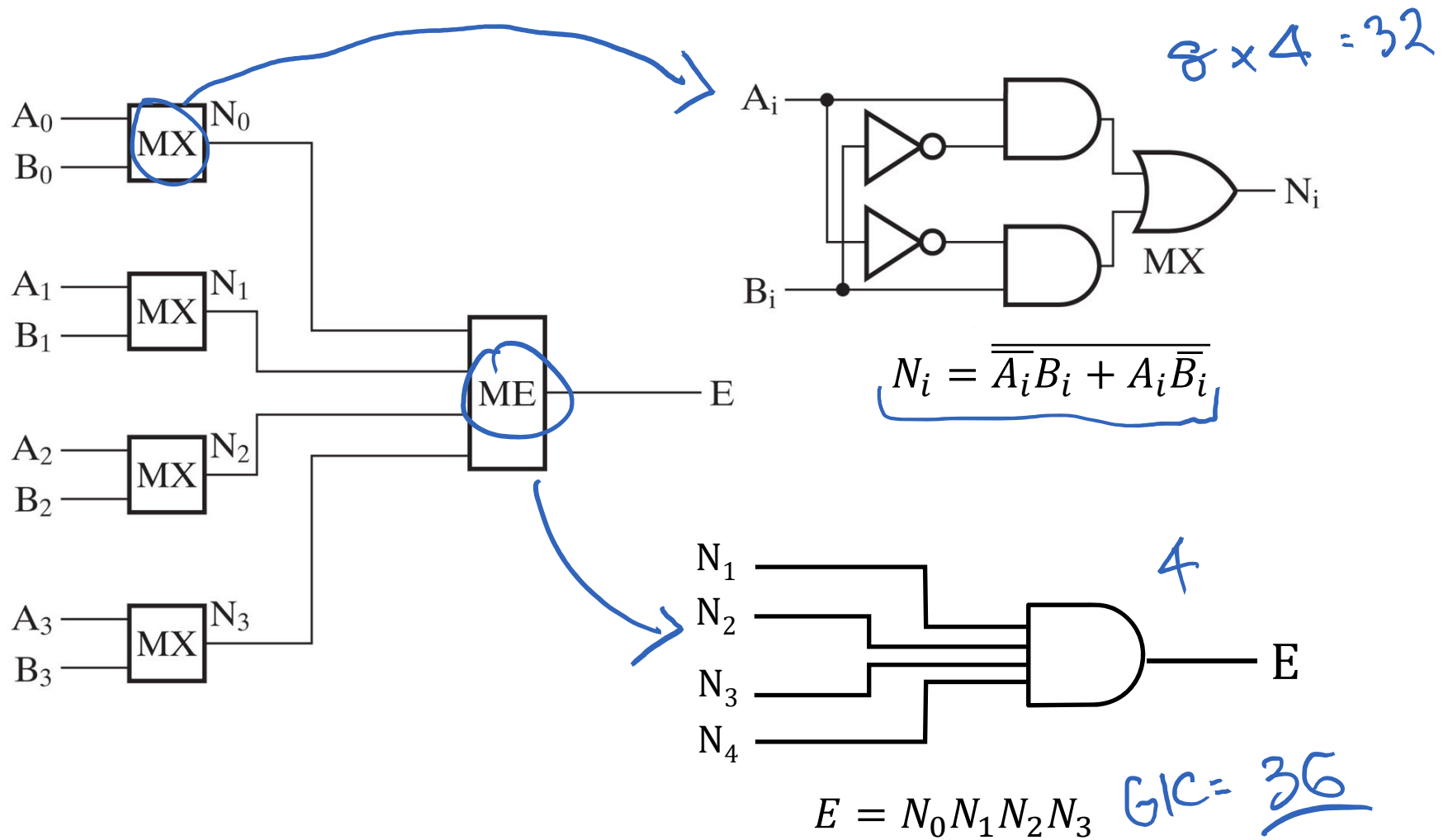
$$E = N_0 N_1 N_2 N_3$$

Logic diagram:

Design a 4-bit equality comparator

$$N_i = \overline{\overline{A_i}B_i + A_i\overline{B_i}}$$

$8 \times 4 = 32$

4

$$E = N_0 N_1 N_2 N_3$$

GIC = 36

Hierarchical design

1. **Reduces the complexity** required to represent the schematic diagram circuit

2. **Allows the reuse of blocks.** In previous example, only one of the four one-bit comparator needs to be designed. A copy of the design can be used for the others

3. Designer needs to look for **regularities** in the circuit. The more regular the designer can abstract a system, the easier it becomes to design as less no. of blocks are required to be designed

   The appearance of a block in a design is called an **instance**

Hierarchical design

Circuits consisting of million gates are designed using hierarchical design using CAD design tools
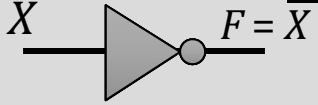
There are predefined reusable blocks called functional blocks are used

They are a predefined collection of interconnected gates

Functional blocks with specific combinational functions are available in most CAD tools as well as discrete ICs, e.g. decoders, encoders, and multiplexers

Rudimentary Logic Functions

For a single variable, X, four different functions are possible:

X	$F = 0$	$F = X$	$F = \overline{X}$	$F = 1$
0	0	0	1	1
1	0	1	0	1
	0-Value Fixing	Transferring	Inverting	1-Value Fixing
	$F = 0$	X —— $F = X$	X —▷∘— $F = \overline{X}$	V_{DD} —— $F = 1$

UNSW SYDNEY

Multi-Bit Variables
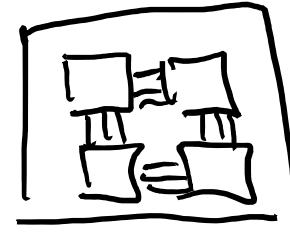
A group of single-bit variables that are used together can be gathered into a *bus* which is a vector signal

Example: the input signals to the 4-bit equality comparator – $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ can be expressed as $A(3:0)$ and $B(3:0)$

Buses are represented graphically using thicker lines and an integer to specify their size:

$A(3:0)$
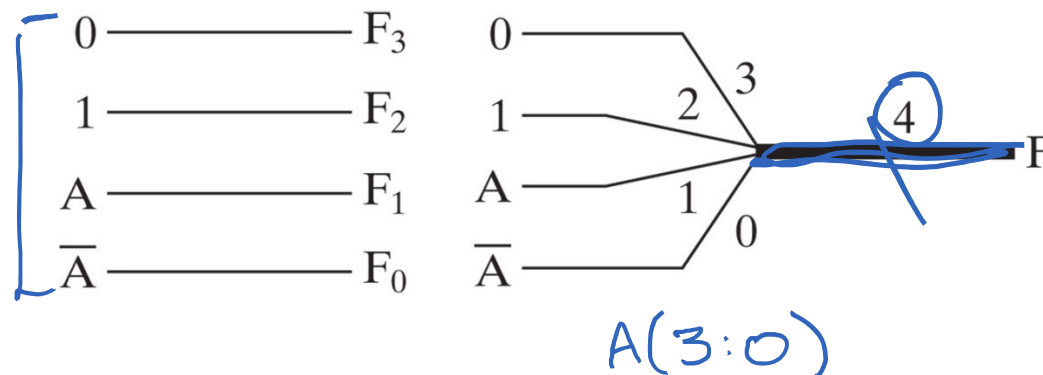
Bus Signals

Buses make circuits clearer to read

They also significantly simplify design and simulation in software

e.g.

A_0 —— 4-Bit Comparator —— E
A_1 ——
A_2 ——
A_3 ——
B_0 ——
B_1 ——
B_2 ——
B_3 ——

A —4— 4-Bit Comparator —— E
B —4—

2 bus lines
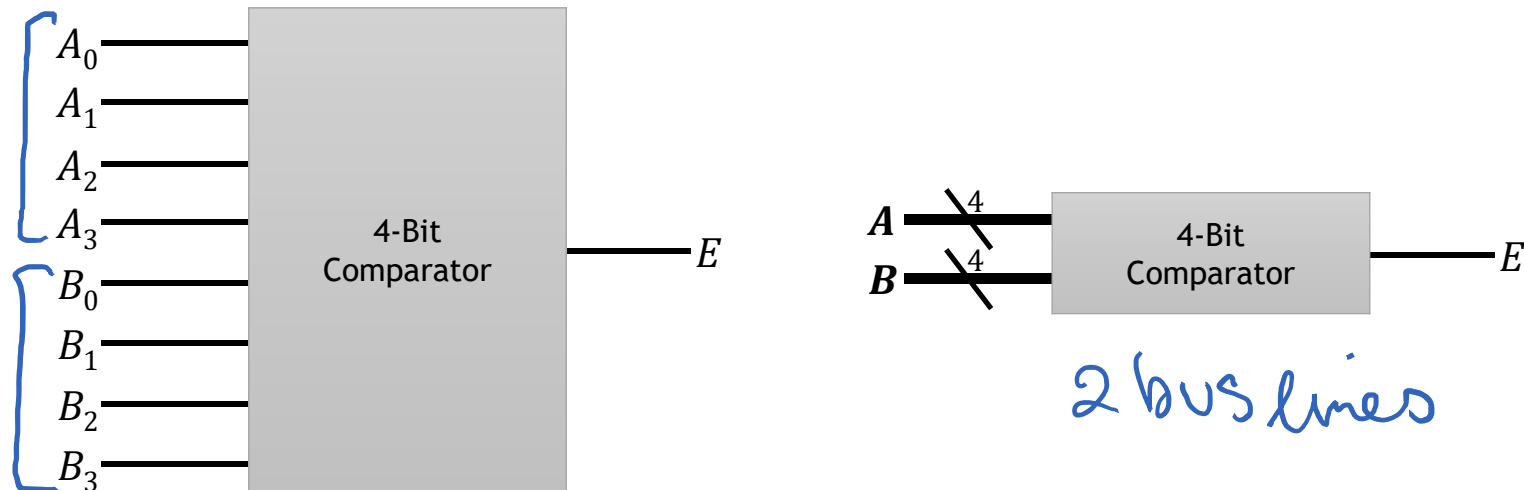
Enabling Circuits

When disabled, output is 0

EN	X	F
0	0	0
0	1	0
1	0	0
1	1	1

EN	F
0	0
1	X

When disabled, output is 1

EN	EN̄	X	F
1	0	0	0
1	0	1	1
0	1	0	1
0	1	1	1

EN	F
0	1
1	X

46
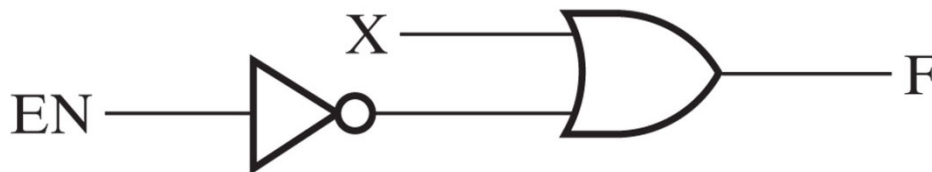
UNSW
SYDNEY

Decoding

Decoders are circuits that convert n-bit input code to m-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code

Functional blocks for decoding are called *n-to-m-line decoders*

The decoder generates *2^n (or fewer) minterms* for the *n input variables*

UNSW
SYDNEY

Decoder Examples

1-to-2-line decoder:

$D_0 = \overline{A}$

$D_1 = A$
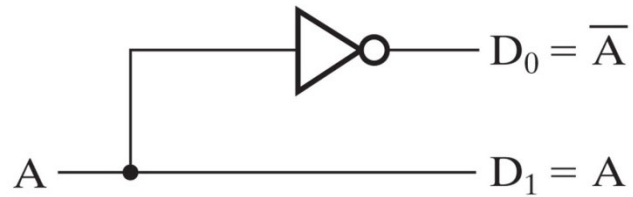
A	D_0	D_1
0	1	0
1	0	1

2-to-4-line decoder:

$D_0 = \overline{A}_1\overline{A}_0$

$D_1 = \overline{A}_1 A_0$

$D_2 = A_1\overline{A}_0$

$D_3 = A_1\overline{A}_0$
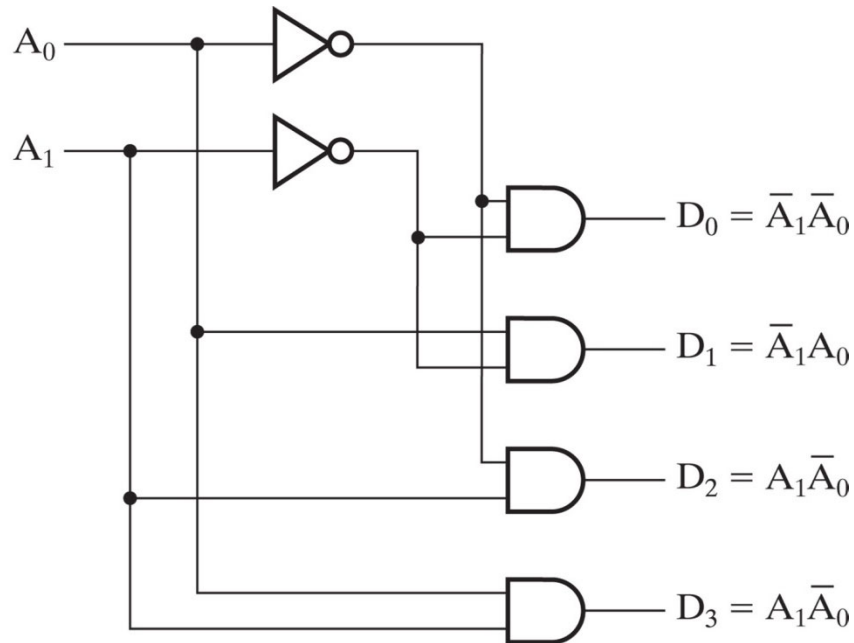
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder Expansion

Decoders can be cascaded together to generate larger decoders using only 2-input AND gates

Example: 3-to-8 line decoder

3-to-8 line decoder
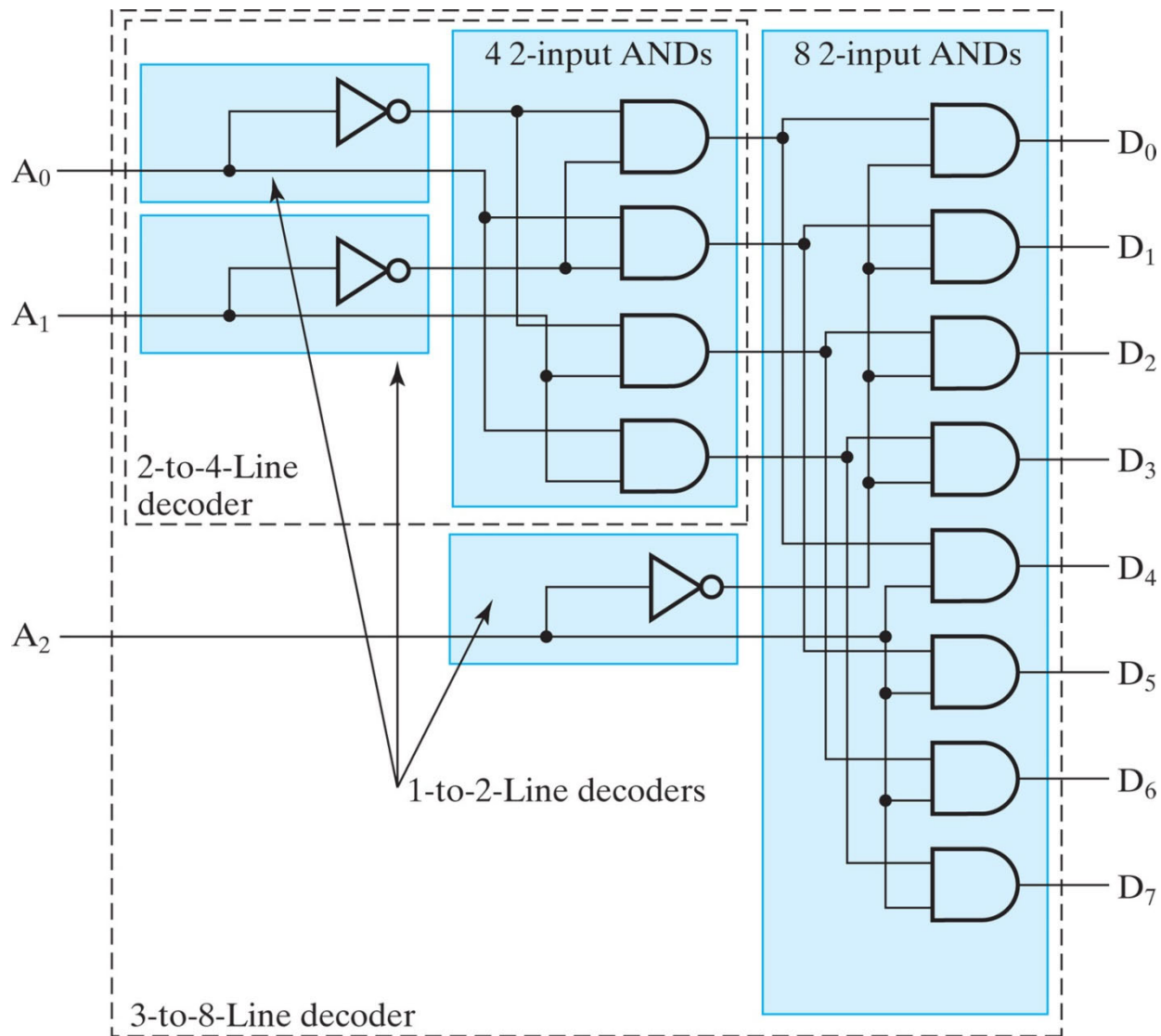
Decoder Expansion Algorithm

The general procedure to construct n-bit decoder using 2-input AND gates only:

1. Let $k = n$

2. If k is **even**, Use 2^k AND gates driven by two decoders of output size $2^{k/2}$

   If k is **odd**, Use 2^k AND gates driven by a decoder of output size $2^{(k+1)/2}$ and a decoder of output size $2^{(k-1)/2}$

3. For each decoder resulting from step 2, repeat step 2 with k equal to the values obtained in step 2 until $k = 1$. For $k = 1$, use a 1-to-2 decoder

6-to-64 Line decoder

k = 6
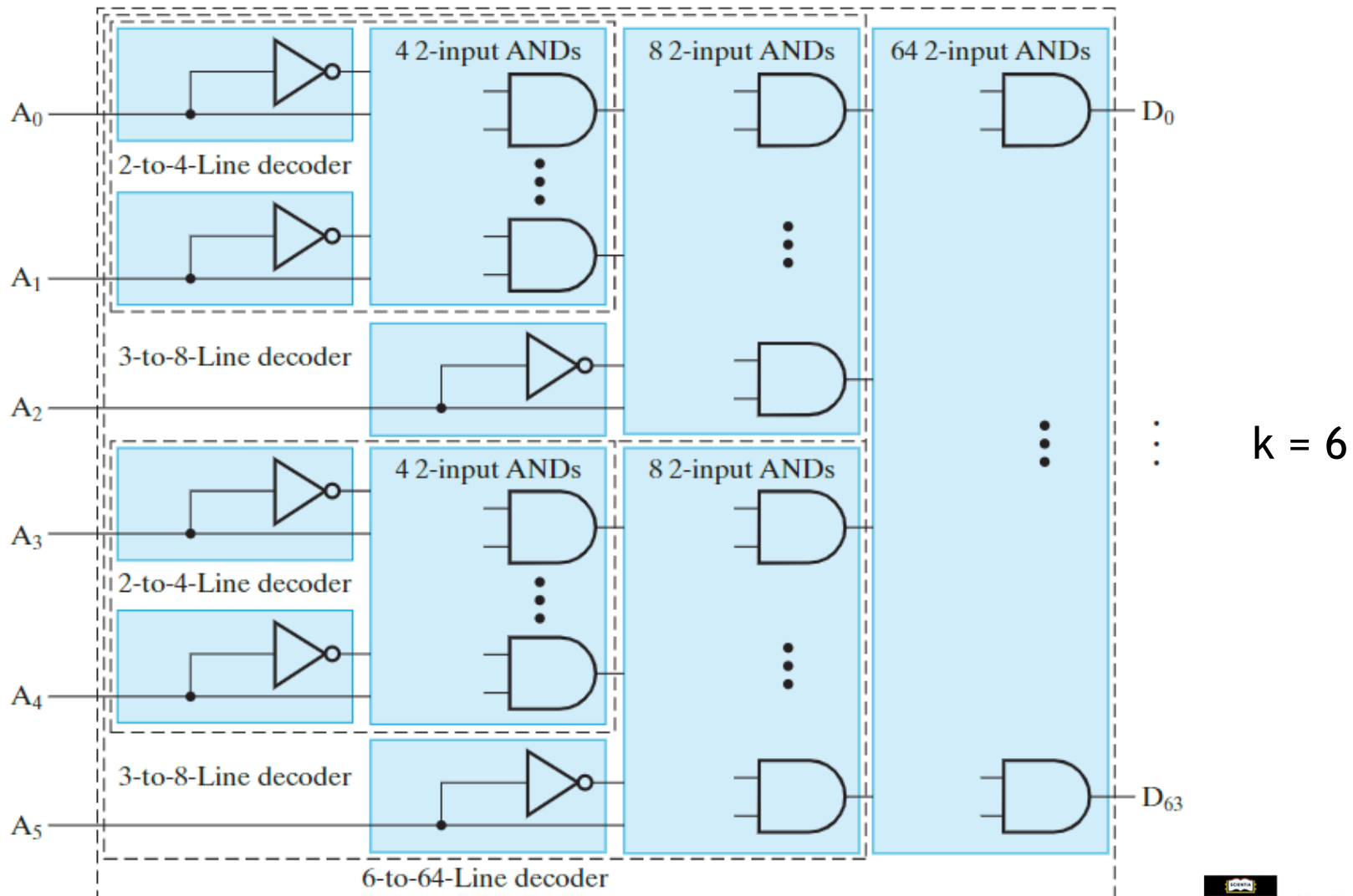
6-to-64 decoder

Using a two input AND approach, the total cost = 6 + 4x2x2 + 8x2x2 + 64x2 = 182

Had the decoder been realized using 6-input AND gates, the total cost would have been 6 + 64x6 = 390 input gates

Substantial gate input cost reduction has been achieved

Decoder with Enable

Add an *Enable* input to the decoder

Allows to set all outputs to 0 when disabled
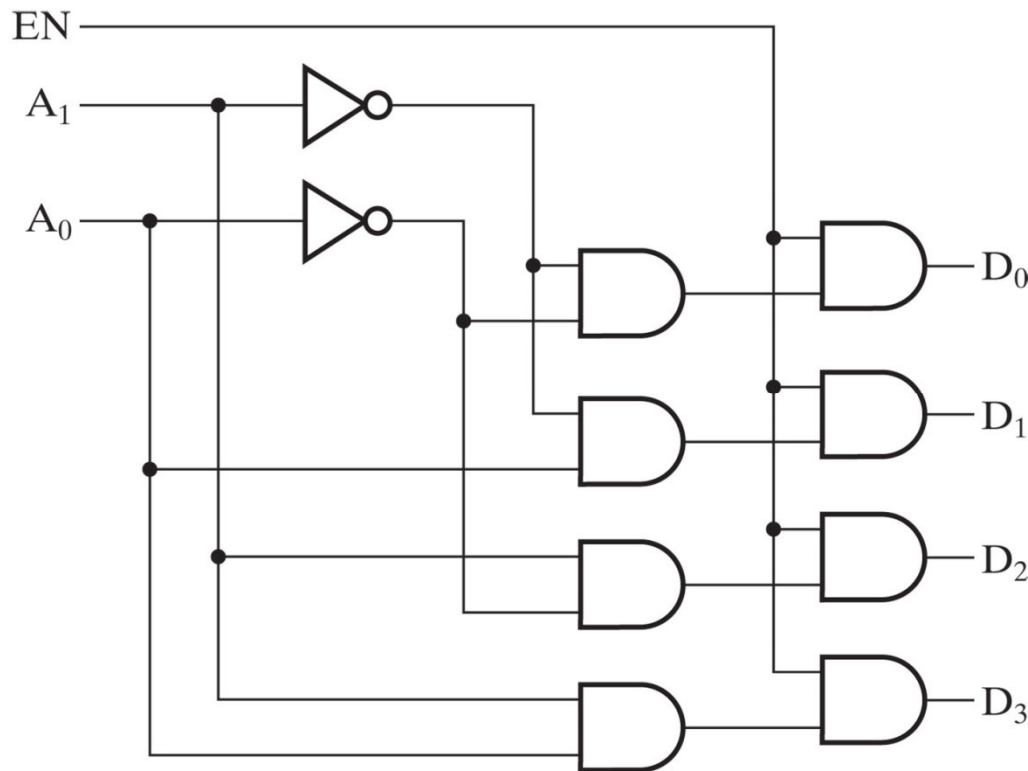
EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Decoder-Based Combinational Circuits

Any Boolean function can be implemented using a decoder and an OR gate

Example – 1-bit binary adder

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S(X, Y, Z) = \sum m(1, 2, 4, 7)$$

$$C(X, Y, Z) = \sum m(3, 5, 6, 7)$$
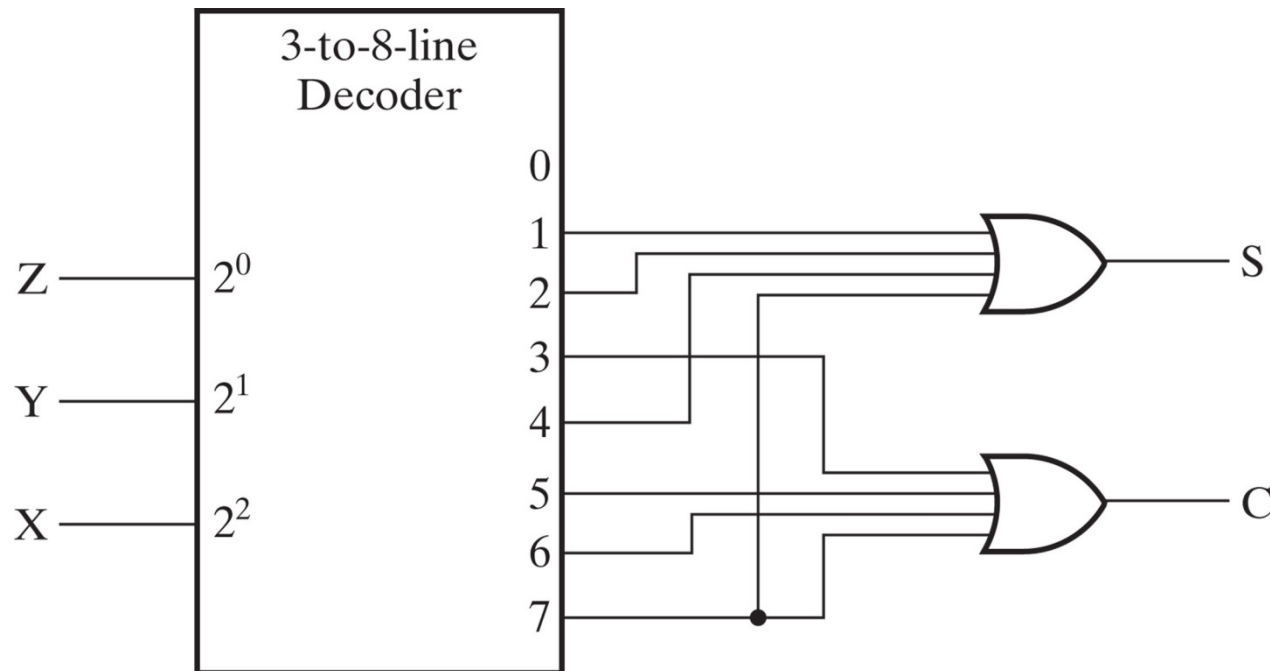
Decoder-Based 1-Bit Binary Adder

$S(X, Y, Z) = \sum m(1, 2, 4, 7)$

$C(X, Y, Z) = \sum m(3, 5, 6, 7)$

Encoding

Encoders perform the inverse operation to decoders

An encoder has 2^n *(or fewer) input lines* and *n output lines*

The output generates the binary code corresponding to the input value

Typically, only one of the inputs can have a value of 1 at any time

Octal-to-Binary Encoder

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$
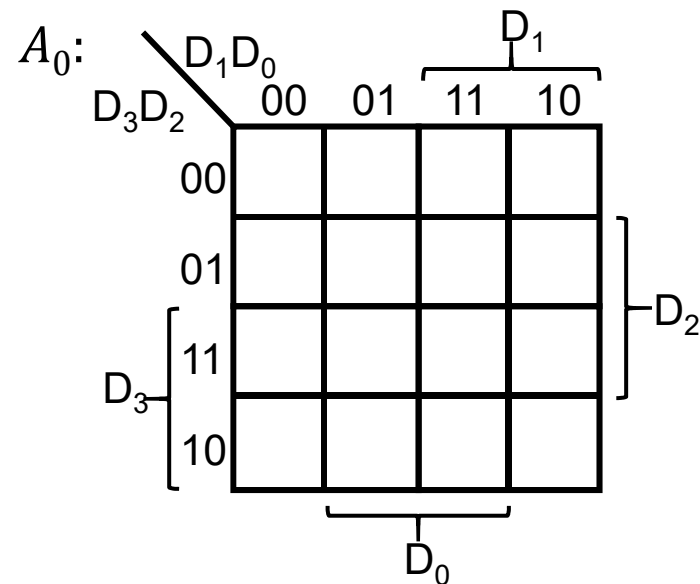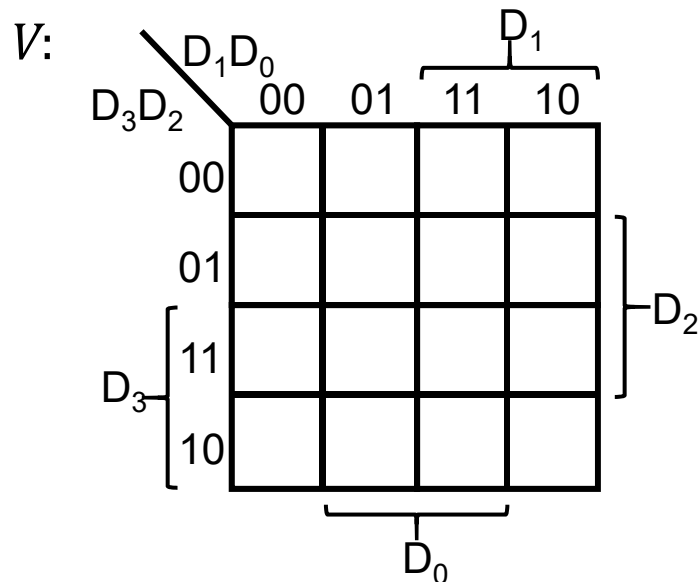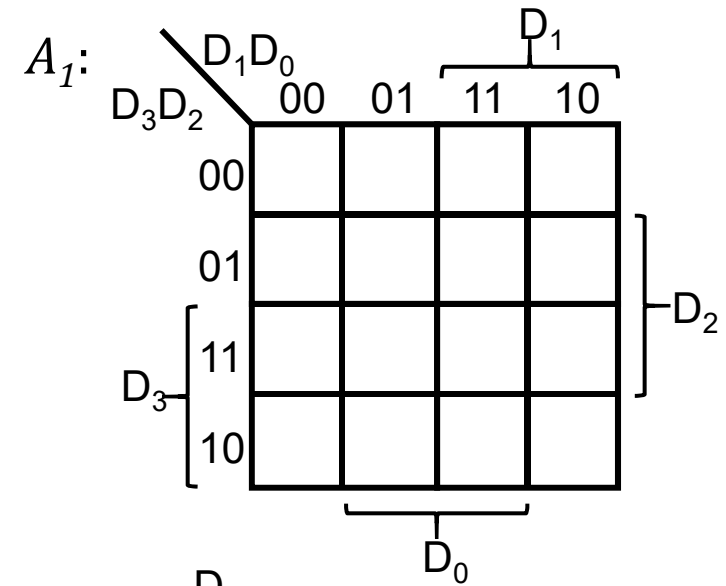
Priority Encoders

Priority Encoders solve the restriction that only one input can be 1 at any time

The priority encoder outputs the binary value corresponding to the most significant input equal to 1

It also solves the ambiguity between all inputs equal 0 and $D_0 = 1$ by adding a *Valid* output

Four-Input Priority Encoder

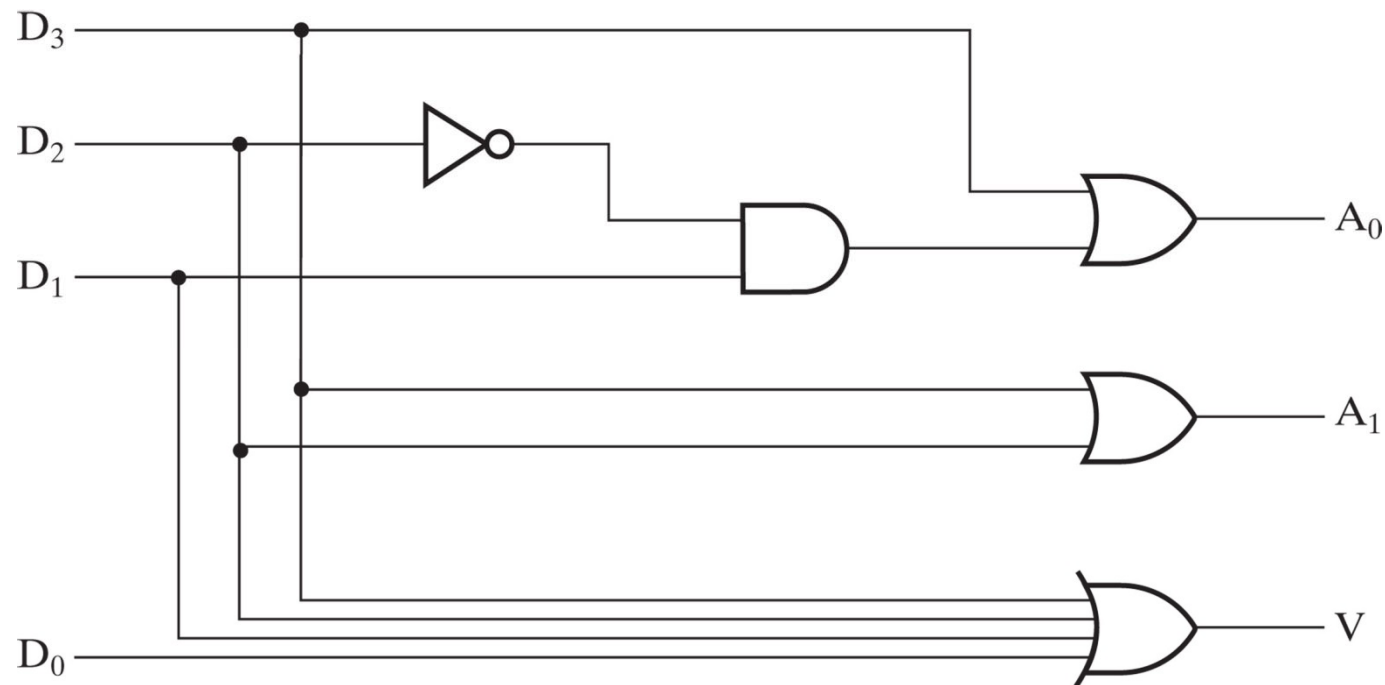| D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Four-Input Priority Encoder

$$A_0 = D_3 + D_1 \overline{D}_2$$

$$A_1 = D_2 + D_3$$

$$V = D_0 + D_1 + D_2 + D_3$$

61

Seven segment displays

Here digits 0-9 and selected alpha characters are displayed by turning on appropriate segments from amongst the 7 segments *abcdefg*

a is high if the segment *a* of the display is required to be on

For example, 0 is displayed with 1111110 because all segments except g need to be on to show the symbol for zero.


Decimal Digits 0-9

0 1 2 3 4 5 6 7 8 9

Select Alpha Characters

A b C d E F H I J
L n o P r S U Y Z
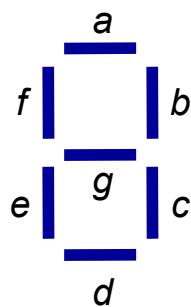
BCD to seven segment decoder

7-segment displays are most commonly used to display numbers coded in BCD

A decoder translates the *four-bit binary-coded decimal word* into *seven-bit words abcdefg*

| w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
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	1	0	1	1
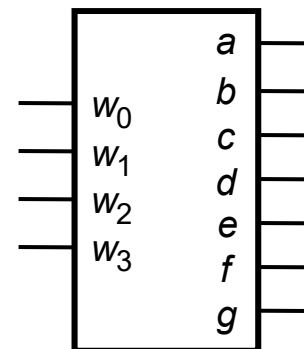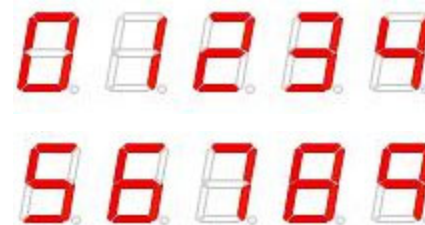
Multiplexers (MUX)

A *multiplexer* selects information from one of many input lines and directs it to an output line

A typical multiplexer has *n selection inputs* (S_{n-1}, ..., S_0) and *2^n information inputs* ($I_{2^n - 1}$, ..., I_0)

Example – 4-to-1-line multiplexer:
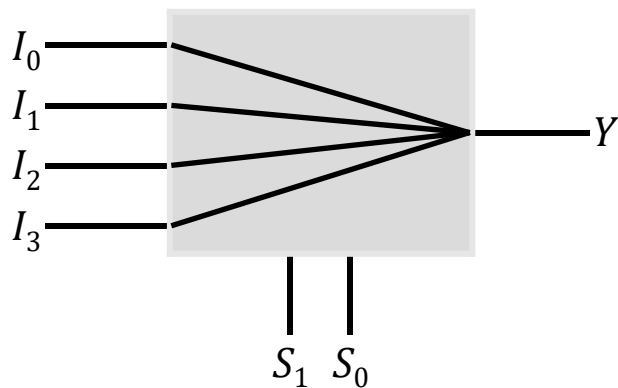
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

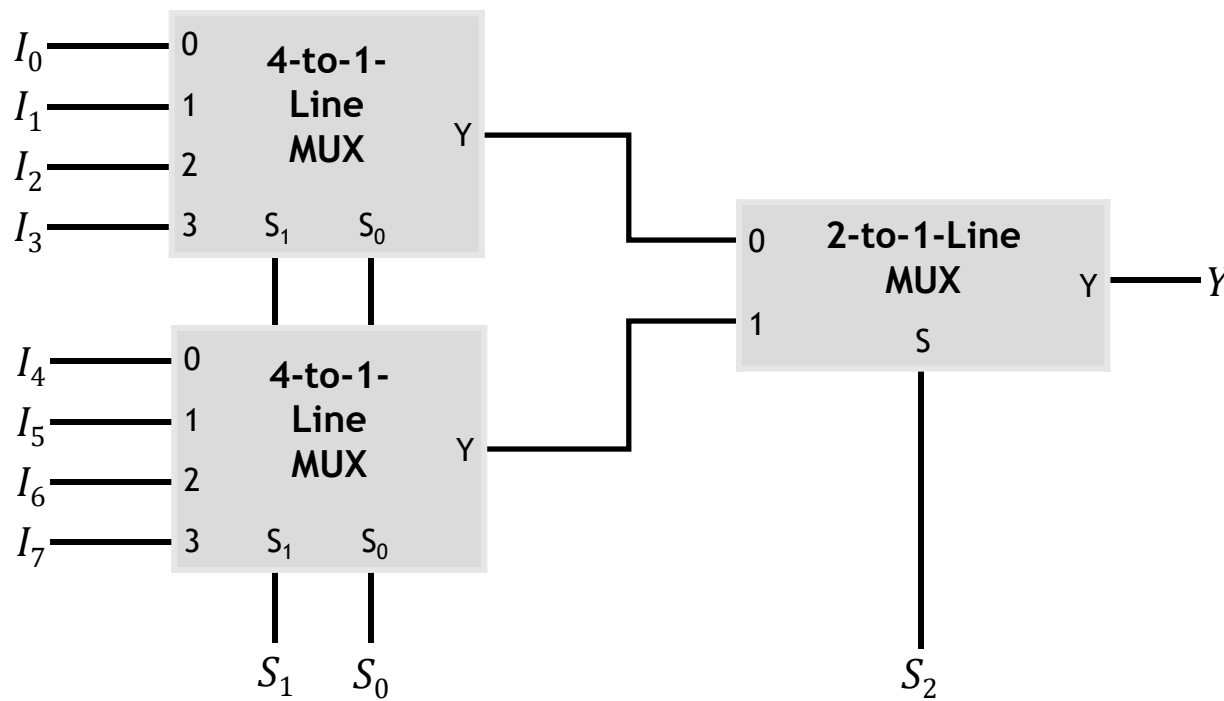# 4-to-1-Line MUX Implementation

A multiplexer can be implemented using a decoder, 2-input AND gates and multiple-input OR gate:
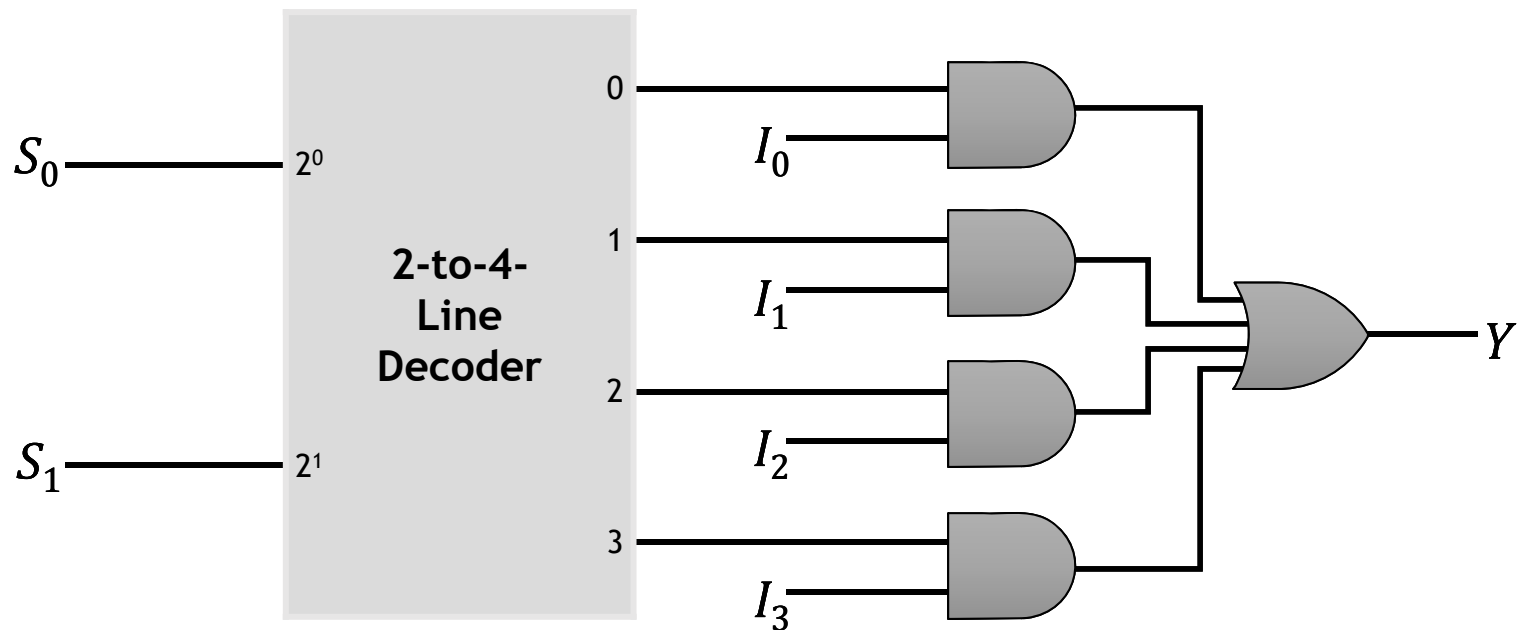
Cascading Multiplexers

Large multiplexers can also be implemented by cascading smaller ones

Example – 8-to-1-line multiplexer:
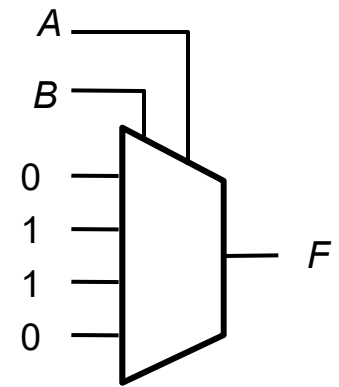
Cascading Multiplexers

Example: 16-to-1 MUX

MUX circuits for Boolean functions

Boolean functions can also be implemented using MUXs

$$F = A \oplus B$$

Easiest way is with 4-to-1 MUX, but not efficient

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

MUX circuits for Boolean functions

Example: Three input majority function (i.e. 2 or more 1's)

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

MUX circuits for Boolean functions

MUX implementations require that a given function be decomposed in terms of variables that can be used as select inputs

This can be done using Shannon's expansion theorem

It states that any Boolean function can be given by

$$F(X_1, X_2, \ldots, X_n)$$

$$= \bar{X}_1 F(0, X_2, \ldots, X_n) + X_1 F(1, X_2, \ldots, X_n)$$

Boolean function implementation

Example: $F = \bar{A}\bar{C} + AB + AC$

Boolean function implementation

Example: $F(X, Y, Z) = \sum m(1,2,6,7)$
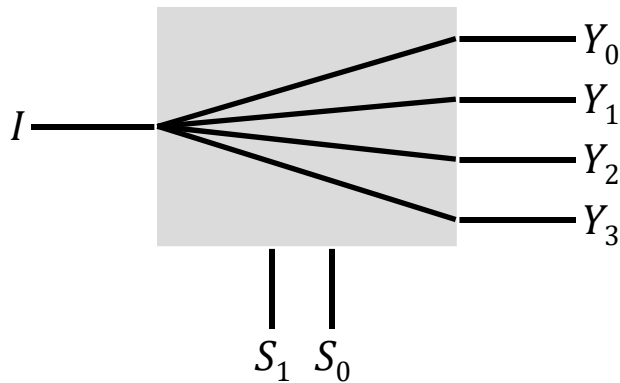
Boolean function implementation

Example:

$$F(A, B, C, D) = \sum m(1,3,4,11,12,13,14,15)$$

Demultiplexers (DeMUX)

A *Demultiplexer* directs the information from *a single input line* to *one of 2^n output lines* and is controlled by *n control lines*

Example – 1-to-4-line demultiplexer:

S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

Demultiplexers and Decoders

Demultiplexers are really the same as decoders with enable

Just swap the signal names:

$$EN \rightarrow I, A \rightarrow S \text{ and } D \rightarrow Y$$

EN	2-to-4-Line Decoder	D_0
		D_1
A_1		D_2
A_0		D_3

$\rightarrow$

I	1-to-4-Line DeMUX	Y_0
		Y_1
		Y_2
$S_1 \quad S_0$		Y_3