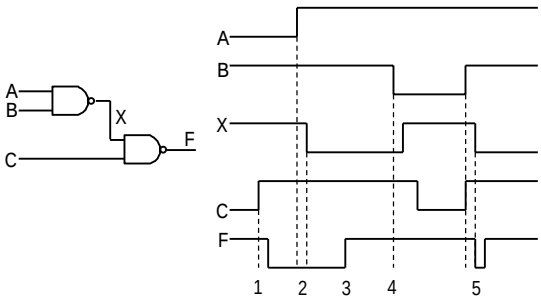
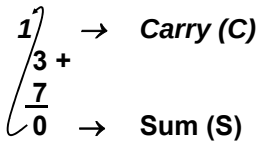
Chapter 4
Solving Larger Problems

- Delay in Combinational Logic Circuits
- Adders and Subtractors
- Decoders and Encoders
- Multiplexers
- Tri-state gates
- Comparators

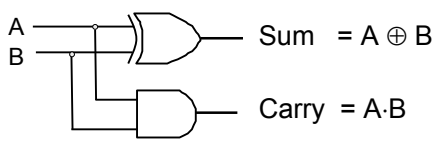
Delay in Combinational Logic Circuits



Half Adder



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



Half Subtractor

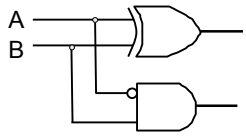
2 *13* → ***Borrower (C)***

3 3 –

7

2 6 → **Difference (D)**

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



Difference = $A \oplus B$

Borrower = $\overline{A} \cdot B$

Full Adder

C_{i-1} ← **1**

5 3 +

2 7

S_i ← **8 0**

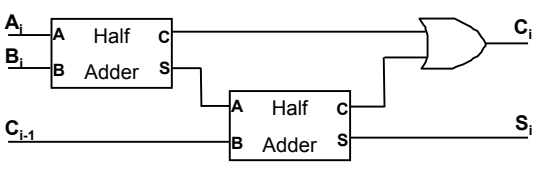
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A_i, B_i, S_i, and C_i are the *i*th order bits of the numbers A, B, Sum and Carry respectively and C_{i-1} is the carry generated from the addition of the *(i-1)*th order bits.

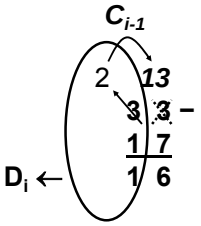
Full Adder (cont.)

$S_i = A_i \oplus B_i \oplus C_{i-1}$

$C_i = A_i \cdot B_i + C_{i-1} \cdot (A_i \oplus B_i)$



Full Subtractor



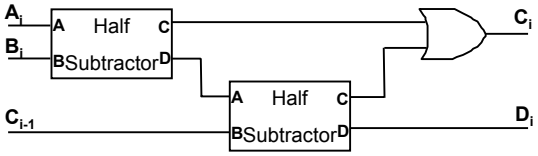
A_i	B_i	C_{i-1}	D_i	C_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

A_i, B_i, D_i, and C_i are the *i*th order bits of the numbers A(Minuend), B(Subtrahend), Difference and Borrow respectively and C_{i-1} is the borrow from the previous stage.

Full Subtractor (cont.)

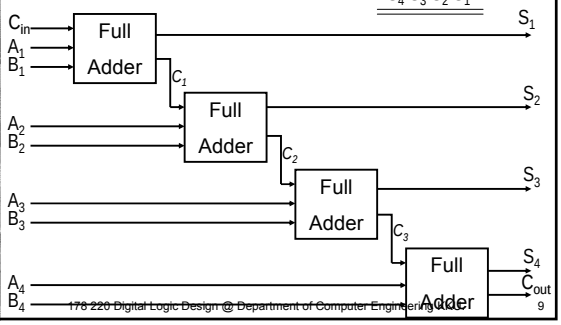
$$D_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = \overline{A}_i \cdot B_i + \overline{(A_i \oplus B_i)} \cdot C_{i-1}$$

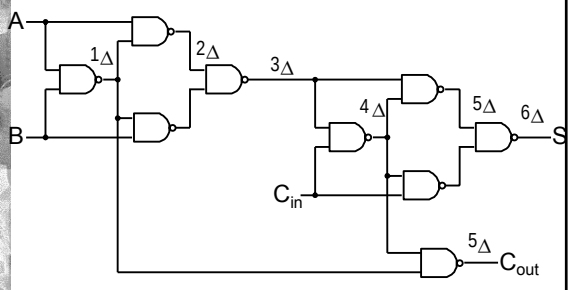


Ripple-carry Adders

C_{out} C_3 C_2 C_1 C_{in}
A_4 A_3 A_2 A_1
+
B_4 B_3 B_2 B_1
S_4 S_3 S_2 S_1



Delay through 1-bit Adder



Carry-look-ahead Adders

It is possible to reduce the delay of the carry chain in the ripple-carry adder by using the carry-look-ahead technique.

If we define carry-generate function $G_i = X_iY_i$

and carry propagate function $P_i = X_i \oplus Y_i$.

The 4 carries $C_{i+1},\ldots, C_{i+4}$ can be expressed as:

$C_{i+1} = G_i + P_iC_i$

$C_{i+2} = G_{i+1} + P_{i+1}C_{i+1}$

$C_{i+3} = G_{i+2} + P_{i+2}C_{i+2}$

$C_{i+4} = G_{i+3} + P_{i+3}C_{i+3}$

Carry-look-ahead Adders (cont.)

After forward substitution:

$C_{i+1} = G_i + P_iC_i$

$C_{i+2} = G_{i+1} + P_{i+1}(G_i + P_iC_i)$

$C_{i+3} = G_{i+2} + P_{i+2}(G_{i+1} + P_{i+1}(G_i + P_iC_i))$

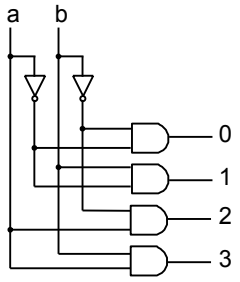
$C_{i+4} = G_{i+3} + P_{i+3}(G_{i+2} + P_{i+2}(G_{i+1} + P_{i+1}(G_i + P_iC_i)))$

These 4 equations show that those 4 carries can be computed directly from input bits and input carry C_i without the ripple effect.

Decoder

= A device, when activated, selects one of several output lines, based on a coded input signal.

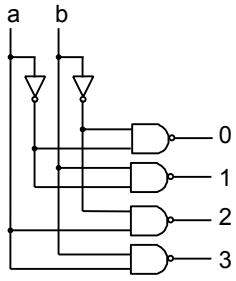
a	b	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



A two-input decoder

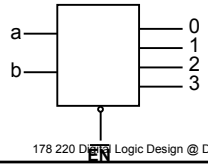
An active low decoder

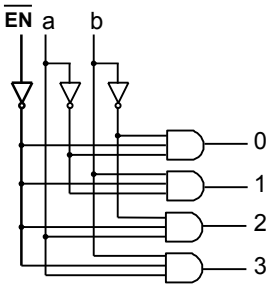
a	b	0	1	2	3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0



Decoder with enable

$\overline{EN}$	a	b	0	1	2	3
1	X	X	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1



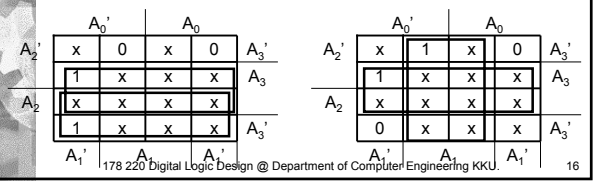


Encoder

= The inverse of a decoder.

A_0	A_1	A_2	A_3	Z_0	Z_1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

$\mathbf{Z_0 = A_2 + A_3}$ $\mathbf{Z_1 = A_1 + A_3}$



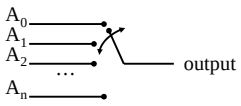
A priority encoder

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Z_0	Z_1	Z_2	NR
0	0	0	0	0	0	0	0	X	X	X	1
X	X	X	X	X	X	X	1	1	1	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0

- $NR = A'_0 A'_1 A'_2 A'_3 A'_4 A'_5 A'_6 A'_7$
- $Z_0 = A_4 + A_5 + A_6 + A_7$
- $Z_1 = A_6 + A_7 + (A_2 + A_3) A'_4 A'_5$
- $Z_2 = A_7 + A_5 A'_6 + A_3 A'_4 A'_6 + A_1 A'_2 A'_4 A'_6$

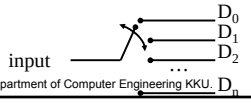
MUX (multiplexer)

- A logic circuit that selects an output from many inputs
- "Data selector"

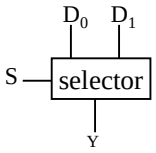


DEMUX (demultiplexer)

- A logic circuit that channels an input into one of many outputs
- "Data distributor"



2-to-1 selector



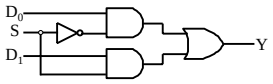
a) Graphical symbol

S	Y
0	D_0
1	D_1

b) Truth table

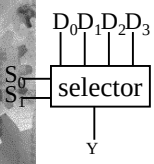
$Y=S'D_0 + SD_1$

c) Boolean expression



d) Logic diagram

4-to-1 selector



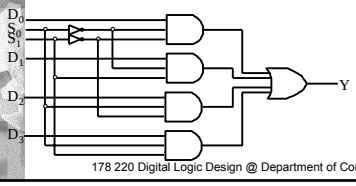
a) Graphical symbol

S_0	S_1	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

b) Truth table

c) Boolean expression

$Y = S_0'S_1'D_0 + S_0'S_1D_1 + S_0S_1'D_2 + S_0S_1D_3$



d) Logic diagram

Comparator

- X = Y ⇒ equal to
- X > Y ⇒ greater than
- X < Y ⇒ less than

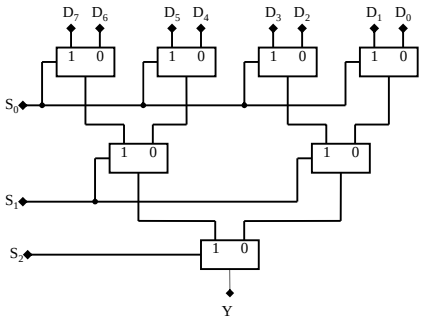
X	Y	G	L
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

Complement operations

- $X \neq Y$ ⇒ not equal to
- $X \leq Y$ ⇒ less than or equal to
- $X \geq Y$ ⇒ greater than or equal to

8-to-1 selector

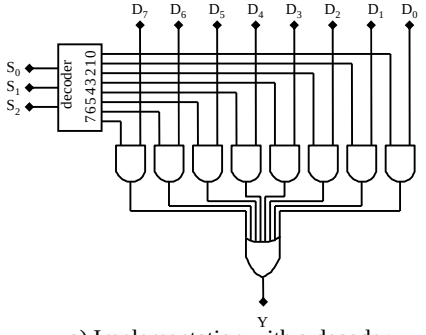
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7



a) Implementation with 2-to-1 selectors

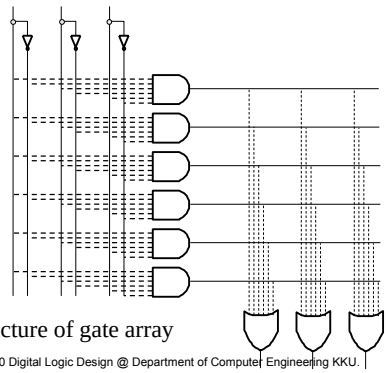
8-to-1 selector (cont.)

S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

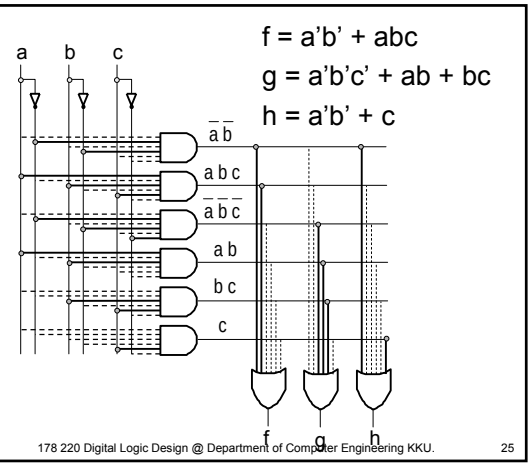


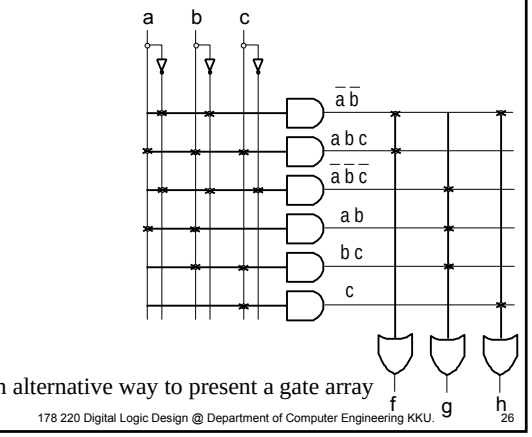
a) Implementation with a decoder

Gate Arrays – ROMs, PLAs and PALs



Structure of gate array



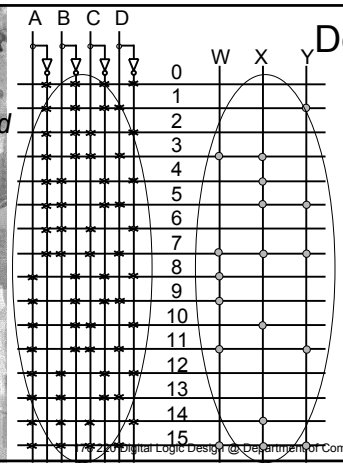


An alternative way to present a gate array

Three common types of combinational logic arrays

- PLAs (Programmable Logic Arrays)
  - Most general type
- ROMs (Read-Only Memories)
  - AND array is fixed
  - Just a decoder consisting of 2^n AND gates
  - User specifies the connections to the OR gate
- PALs (Programmable Array Logic)
  - The connections to the OR gates are specified
  - User can determine the AND gate inputs
  - Each product term can be used only for one of the sums

Designing with ROMs



$W(A, B, C, D) = \Sigma m(3,7,8,9,11,15)$
$X(A, B, C, D) = \Sigma m(3,4,5,7,10,14,15)$
$Y(A, B, C, D) = \Sigma m(1,5,7,11,15)$

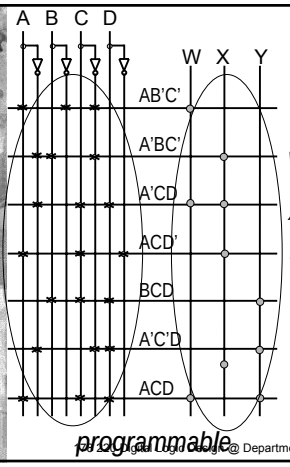
Possible solutions:

$W = AB'C' + CD$
$= AB'C' + A'CD + ACD$

$X = A'BC' + A'CD + ACD' + \{BCD \text{ or } ABC\}$

$Y = A'C'D' + ACD + \{A'BD \text{ or } BCD\}$

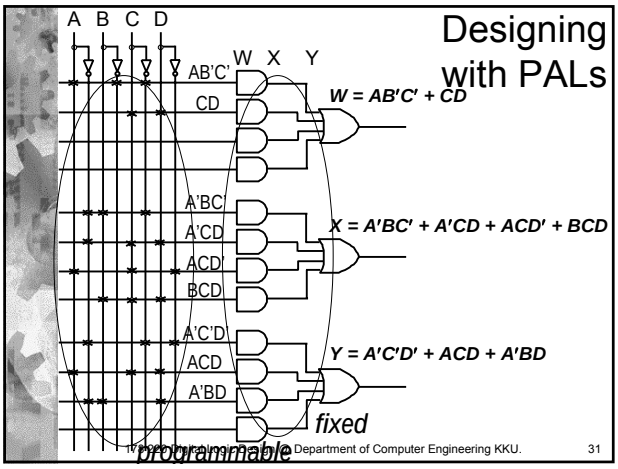
Designing with PLAs



$W = AB'C' + A'CD + ACD$

$X = A'BC' + ACD' + A'CD + BCD$

$Y = A'C'D + ACD + BCD$



Designing with PALs