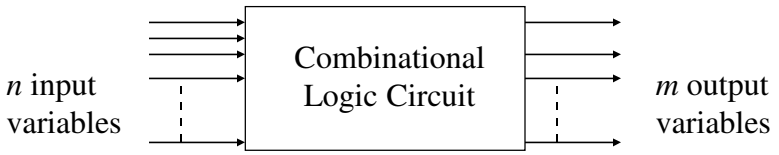
CSC 370 - Computer Architecture and Organization

Lecture 4 - Combinational Logic

Introduction

- A combinational circuit consists of input variables, logic gates, and output variables.
  - The logic gates accept *n* input signals and generate the *m* signals that become output.
- For n input variables, there are 2^n possible combinations of binary input values.
  - For each input combination, there will be one and only possible output combination.
- Each input will have one or two wires.
  - If there is one wire, it will be either in the normal (unprimed) form or the complemented (primed) form.
  - If there are two wires, it will supply both forms.

Block Diagram for a Combinational Circuit



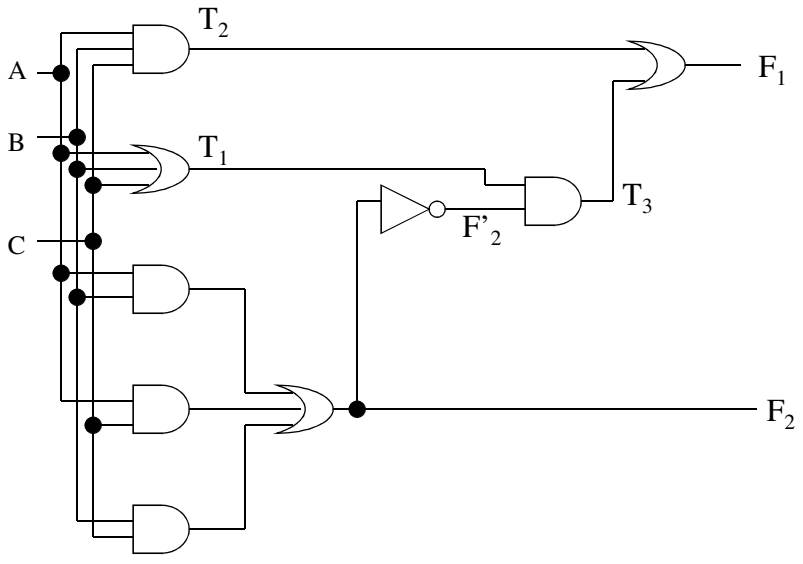
Combinational Circuit Analysis

- Analysis of a combinational circuit requires that we find the function(s) that the circuit implements.
- First we must ensure that the circuit is combinational and ***not*** sequential. (The lack of feedback paths or memory elements ensures that).
- After this, we try to find the logic function or truth table.

Analysis Procedure

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuits are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Analysis Example



Analysis Example – The Outputs

- We initially get:

  $F_2 = AB + AC + BC$

  $T_1 = A + B + C$

  $T_2 = ABC$

- Next, we consider the outputs of gates that are a function of symbol that are already defined:

  $T_3 = F'_2 T_1$

  $F_1 = T_3 + T_2$

Analysis Example – Solving For F_1

$$
\begin{aligned}
F_1 &= T_3 + T_2 = F'_2 T_1 + ABC \\
&= (AB + AC + BC)'(A + B + C) + ABC \\
&= (A'+B')(A'+C')(B'+C')(A + B + C) + ABC \\
&= (A' + B'C')(AB'+AC'+ BC'+ B'C) + ABC \\
&= A'BC' + A'B'C + AB'C' + ABC
\end{aligned}
$$

The Truth Table for F_1

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	**0**	1	0	0	0	**0**
0	0	1	**0**	1	1	0	1	**1**
0	1	0	**0**	1	1	0	1	**1**
0	1	1	**1**	0	1	0	0	**0**
1	0	0	**0**	1	1	0	1	**1**
1	0	1	**1**	0	1	0	0	**0**
1	1	0	**1**	0	1	0	0	**0**
1	1	1	**1**	0	1	1	0	**1**

Design Procedure

- The design procedure starts with the verbal outline of the problem and ends with a logic circuit diagram or a set of Boolean functions from which the circuit diagram can be created.

The Steps in the Design Procedure

The procedure involves these steps:

1. The problem is stated.
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth tables that defines the required relationships between inputs and outputs are derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

The Truth Table

- The truth table consists of input and output columns.
- The 1s and 0s for the input are obtained from the 2^n combinations of n input variables.
- An output could be either 1 or 0 for every valid input combination.
- Some input combinations will not occur; these become ***don't-care*** conditions.

Simplifying the Boolean Functions

- The output functions are simplified by Boolean algebra, Karnaugh maps or tabulation.
- There will usually more than one simplified expression to choose from.
- Which expression we choose may depend on circuit design constraints such as :
  - Minimum number of gates
  - Number of input to a gate
  - Minimum propagation time of the signal through the circuit.
  - Minimum number of interconnections
  - Limitation of the driving capabilities of each gates.

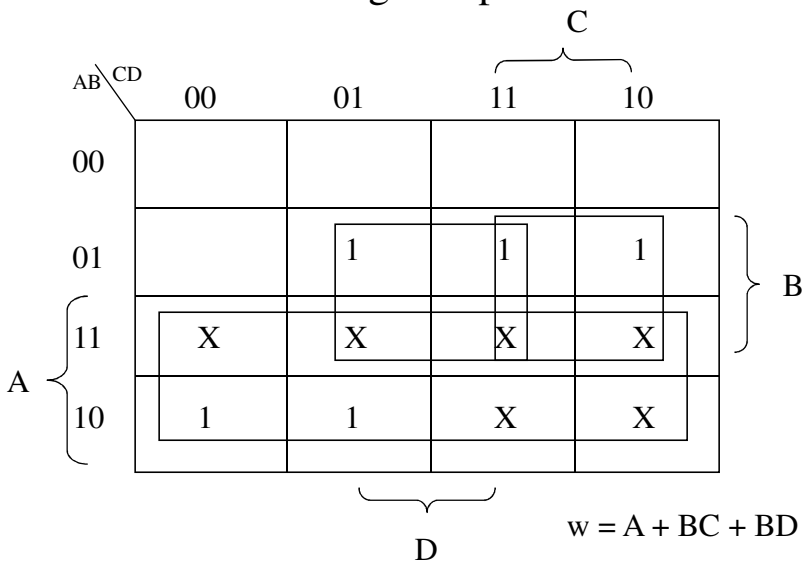
Code Conversion From BCD to Excess-3

- BCD (***B***inary-***C***oded *D*ecimal) and Excess-3 provide two different ways of representing a decimal value in a binary format.
- There will be a one-to-one correspondence between BCD inputs and the corresponding Excess-3 values.
- Not all the BCD minterms are valid values. These will lead to don't-care conditions.

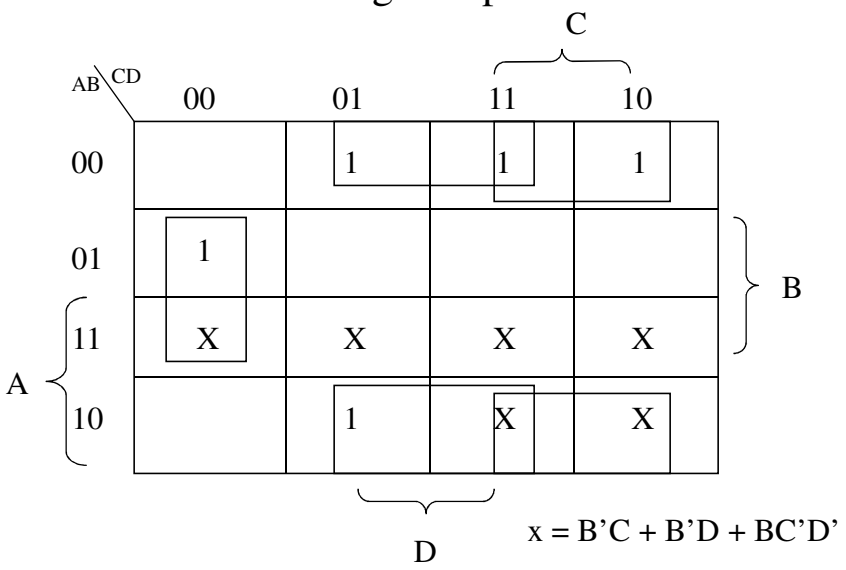
Truth Table for BCD to Excess-3

BCD Inputs				Excess-3 Outputs			
A	**B**	**C**	**D**	**w**	**x**	**y**	**z**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Karnaugh Map for w

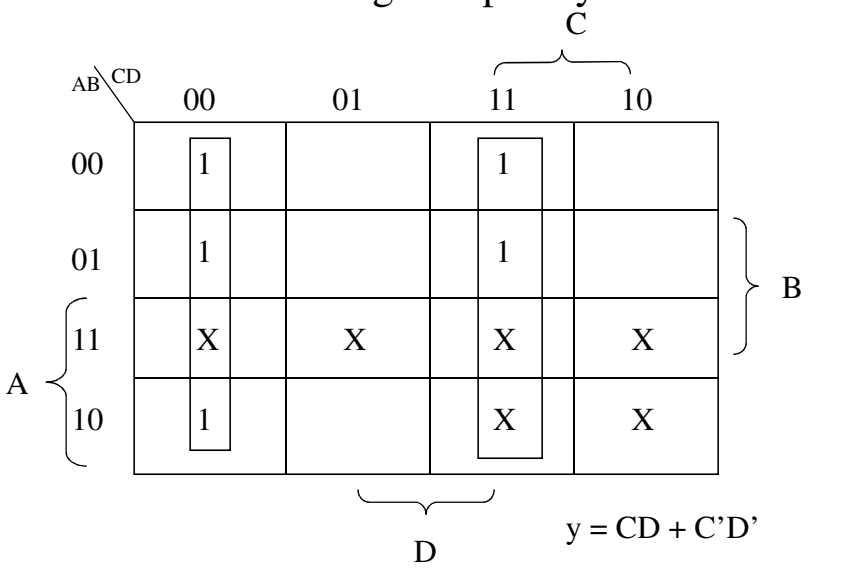


Karnaugh Map for x



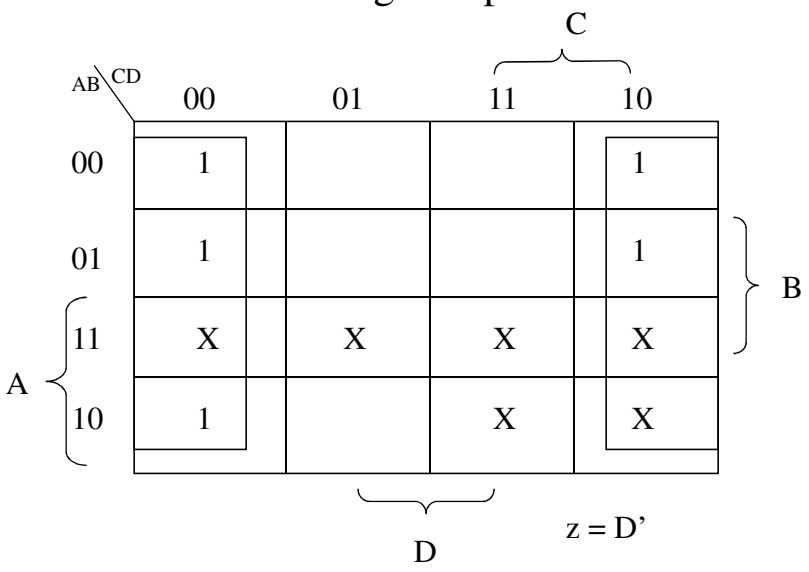
x = B'C + B'D + BC'D'

Karnaugh Map for y

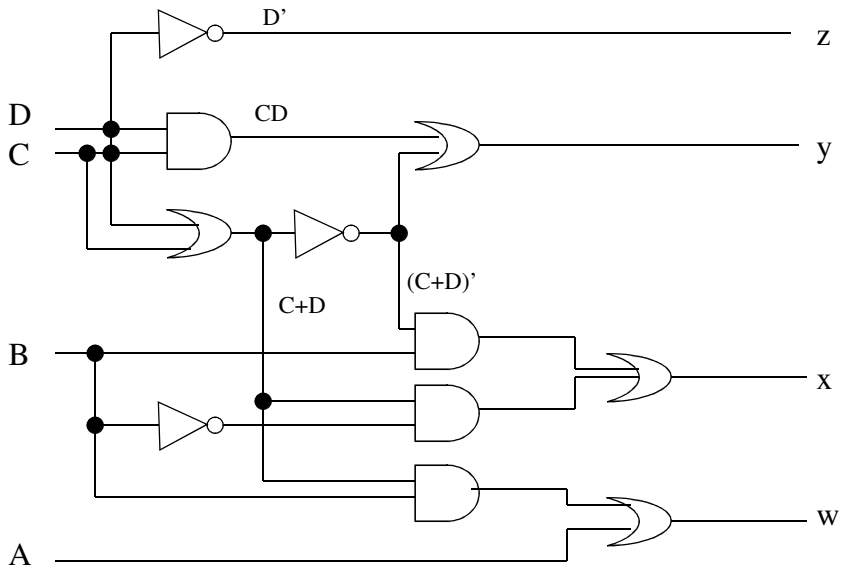


y = CD + C'D'

Karnaugh Map for z



Circuit Diagram for BCD to Excess-3



Half Adder

- The most basic arithmetic operation is the addition of two binary digits.
- We know that:
  - 0 + 0 = 0
  - 0 + 1 = 1 + 0 = 1
  - 1 + 1 = 10
- If both addends are 1, we need a carry bit which will be added to the addend in the next more significant bit.

Half Adder (continued)

- We can summarize this in the form of a truth table:

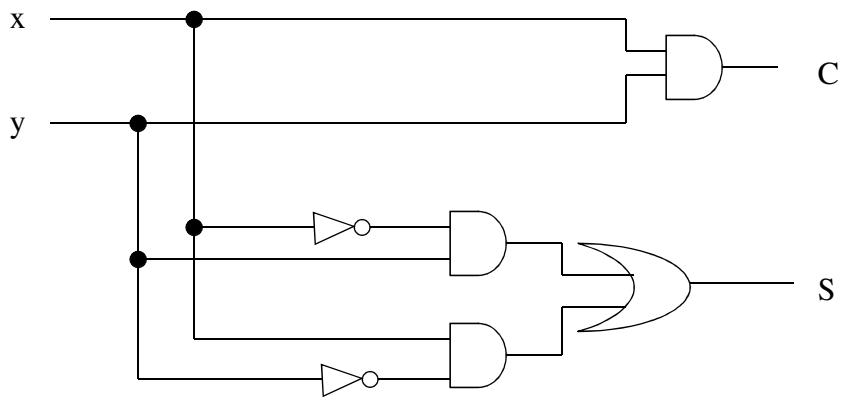
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- From this we learn that:

  $S = x'y + xy' = x \oplus y$

  $C = xy$

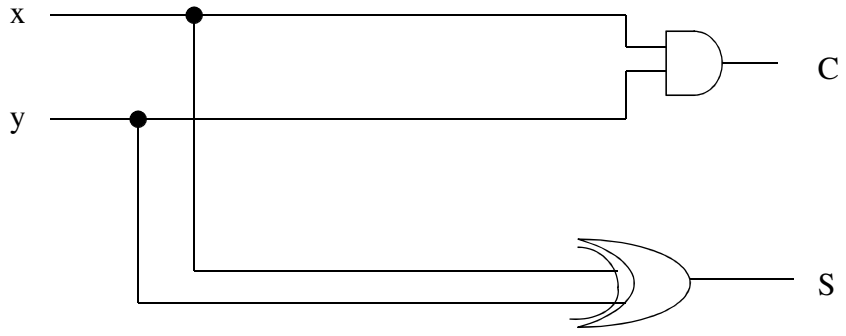
Implementation of Half Adder



S = x'y + xy'

C = xy

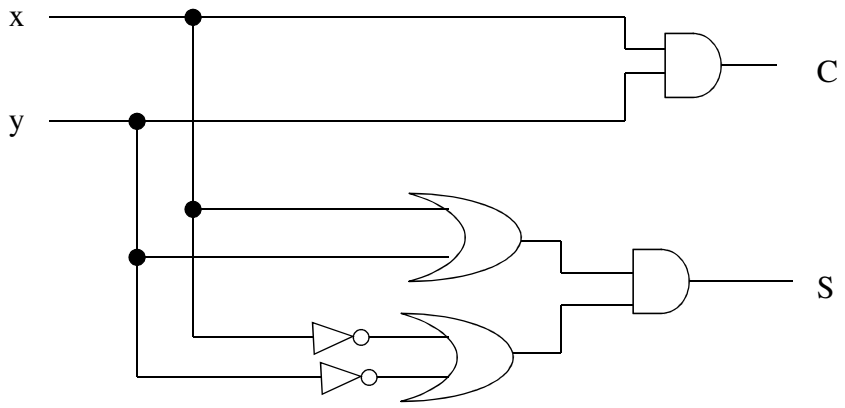
Implementing a Half Adder Using **XOR**



S = x $\oplus$ y

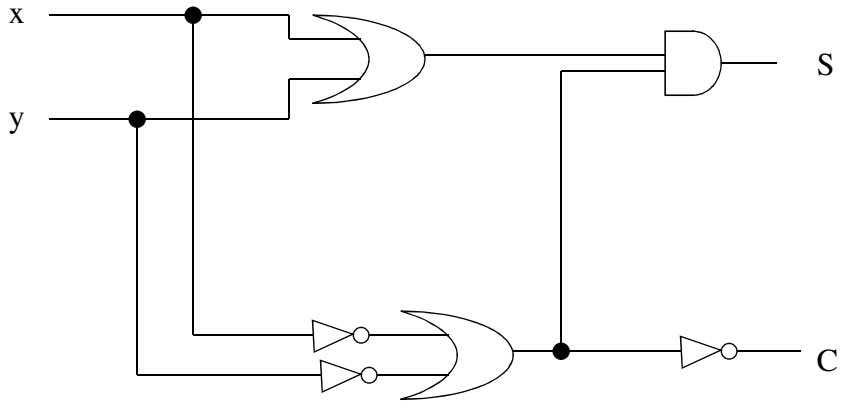
C = xy

Implementing a Half Adder As a Product of Sums



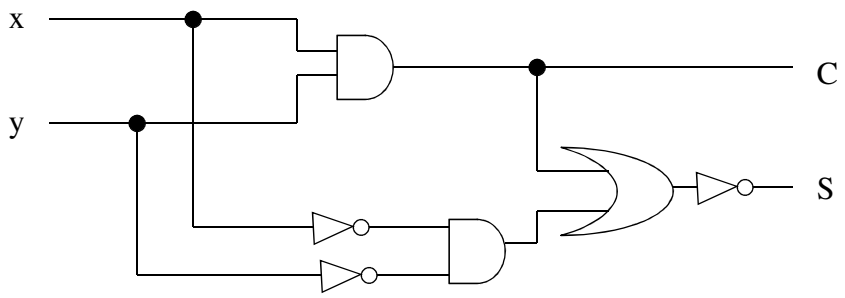
S = (x+y)(x'+y')
C = xy

Implementing a Half Adder As a Product of Sums



S = (x+y)(x'+y')
C = (x'+y')'

Implementing a Half Adder



C = xy
S = (C + x'y')'

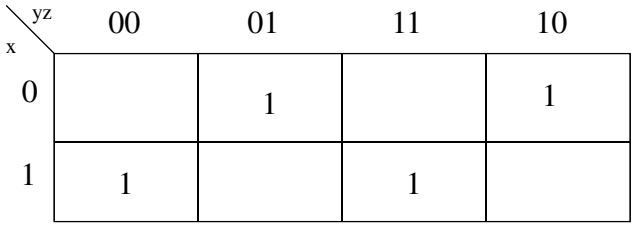
The Full Adder

- A full adder is a combinational circuit that forms the arithmetic sum of three inputs.
  - It consists of 3 inputs and two outputs.
  - Two of the inputs (x and y) are the same as in the half adder.
  - The third input (z) is the carry from the addition of the previous (lesser significance) bits.

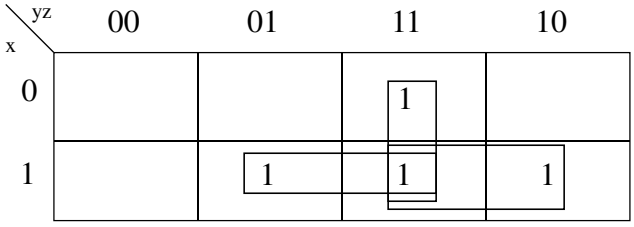
Truth table for a full adder

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

Karnaugh Maps for Full Adder

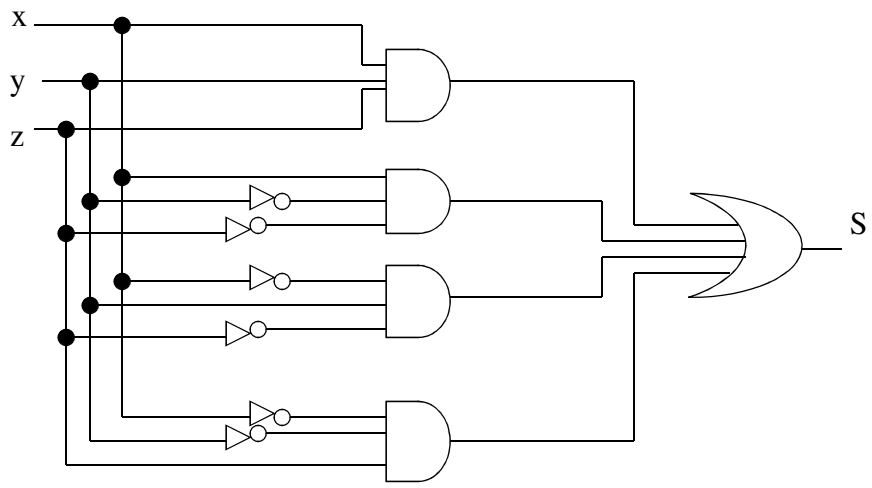


S = x'y'z + x' y z' + x y'z' + xyz

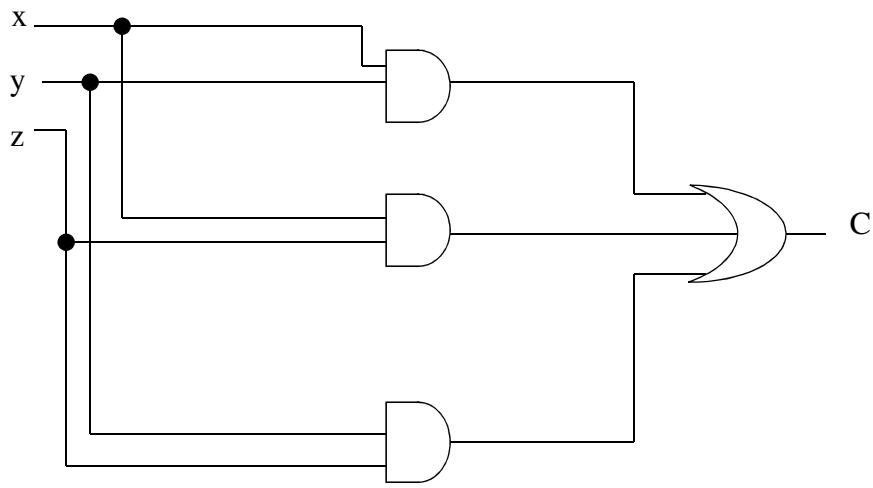


C = xy + xz + yz

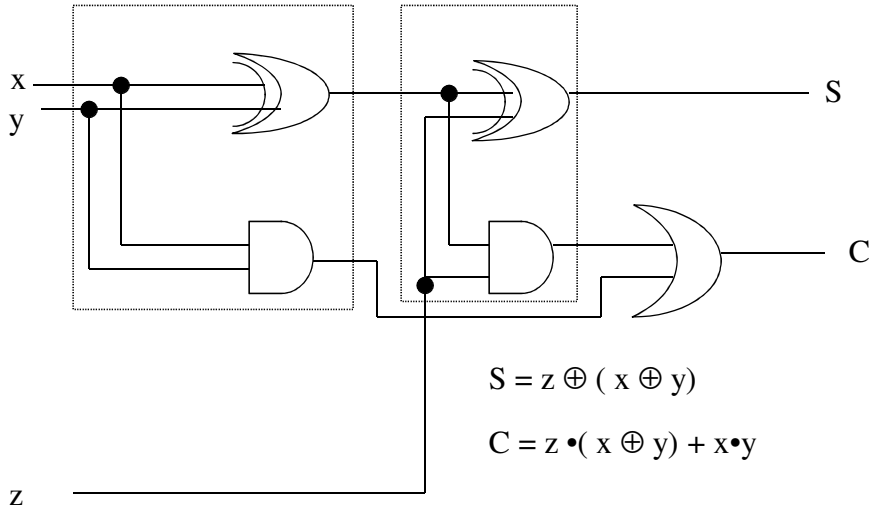
Full Adder Circuit Diagram For the Sum



Full Adder Circuit Diagram For the Carry



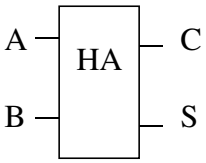
Full Adder Circuit Diagram Using Half Adders



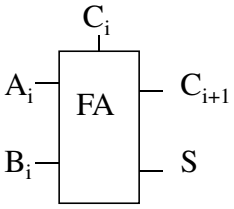
$S = z \oplus (x \oplus y)$

$C = z \bullet (x \oplus y) + x \bullet y$

Block Diagram For Adders



Half Adder

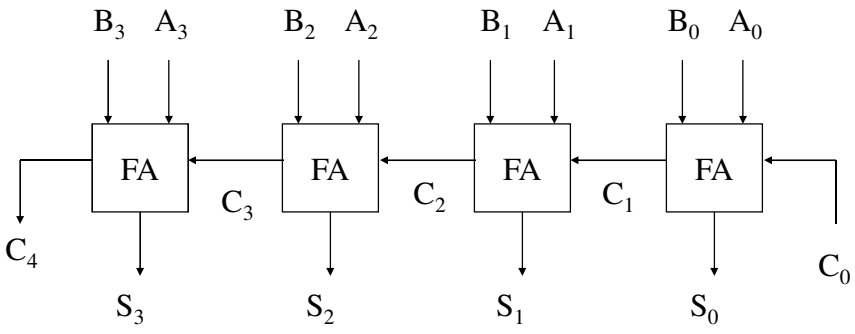


Full Adder

Binary Adder

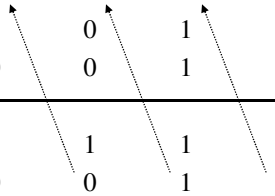
- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed by connecting a series of full adders in cascade.

4 –Bit Adder



4-Bit Adder – An Example

Subscript i	**3**	**2**	**1**	**0**	C_i
Input Carry	0	1	1	0	A_i
Augend	1	0	1	1	B_i
Augend	0	0	1	1	C_i
Sum	1	1	1	0	S_i
Output Carry	0	0	1	1	C_{i+1}



Carry Propagation

- Adding two binary numbers in parallel implies that we have all the bits that we need available at the same time. The cascading of carries seems to belie this assumption.
- If we can generate the necessary bits to determine carries in parallel, then we can actually do the summation without waiting for a carry to cascade through.

Carry Propagation

- We can add two additional terms:
  G_i – Carry Generate
  P_i Carry Propagate
- We can define them as:
  $P_i = A_i \oplus B_i$
  $G_i = A_i B_i$
- The output sum and carry are now:
  $S_i = P_i \oplus G_i$
  $C_{i+1} = G_i + P_i C_i$

Carries in a Carry Lookahead Generator

C_0 = input carry

$C_1 = G_0 + P_0C_0$

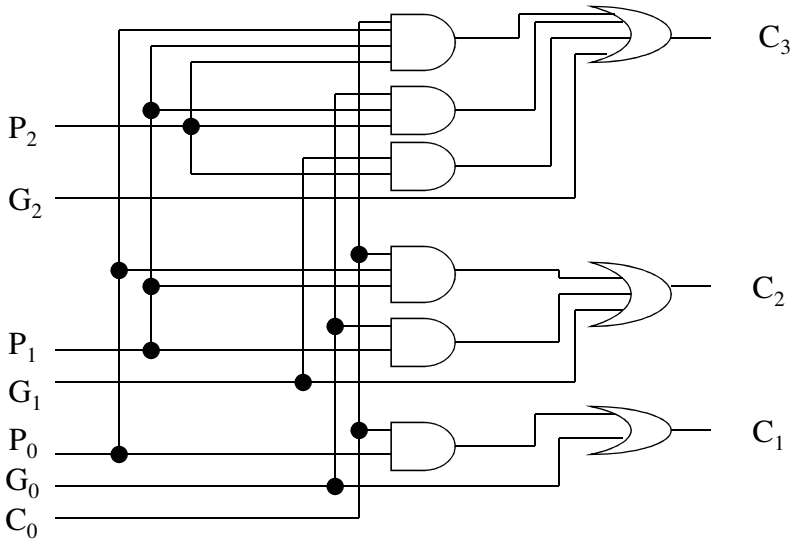
$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)$

$= G_1 + P_1G_0 + P_1P_0C_0$

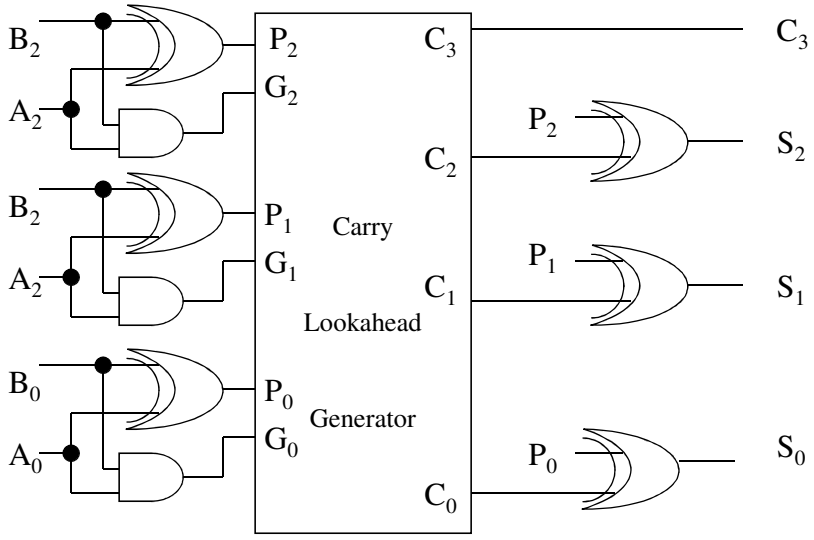
$C_3 = G_2 + P_2C_2$

$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

Carry Lookahead Generator



3-Bit Adder with Carry Generator



Half Subtractor

- A half subtractor subtracts two bits and produces their difference and whether a 1 was borrowed.
- We must remember that:

  0 – 0 = 0; 1 – 0 = 1; and 1 – 1 = 0

  If we have 0 – 1, we must borrow from the next place, so our difference is 1 with a borrow of 1.

Truth Table for the Half Subtractor

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

D = x'y + xy' = x $\oplus$ y
B = x'y

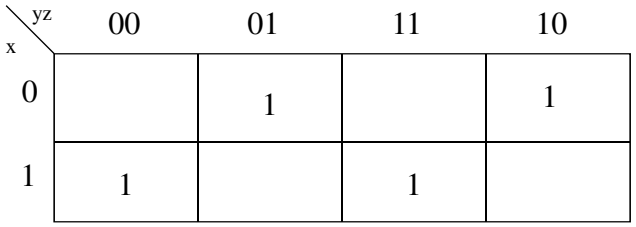
Full Subtractor

- A full subtractor performs subtraction between two bits takinginto account the potential borrow from a lower significance bit.
- A full subtractor's inputs are
  - x, the minuend
  - y, the subtrahend
  - z, the borrow

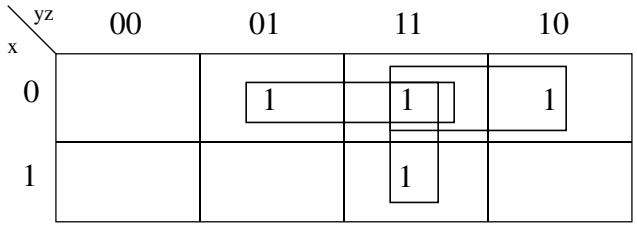
Truth Table for the Full Subtractor

x	**y**	**z**	**B**	**D**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Karnaugh Maps for Full Subtractor



D = x'y'z + x' y z' + x y'z' + xyz

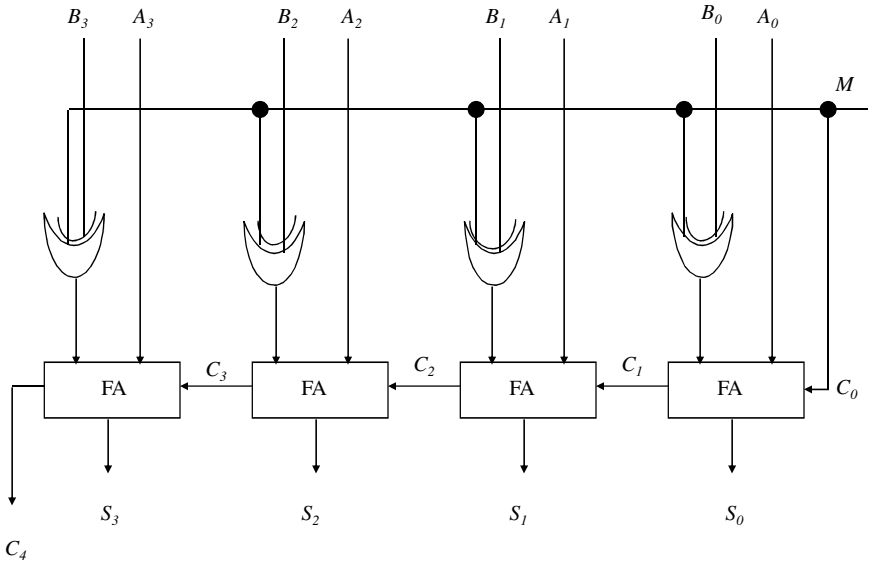


B = x'y + x'z + yz

Adder-Subtracter

- Subtracting ***A – B*** is most easily done by adding ***B'*** to ***A*** and then adding 1.
- This makes it convenient to combine both addition and subtraction into one circuit, called an adder-subtracter.
- M is the mode indicator
  - M = 0 indicates addition (B is left alone and C_0 is 0)
  - M = 1 indicates subtraction (B is complement and C_0 is 1).

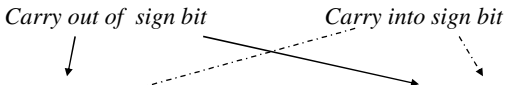
4-Bit Adder-Subtractor



Overflow

- If addition of 2 n-bit augends produces an n+1-bit sum, we say that overflow occurs.
- Overflow is a problem for computers if undetected because the answer that is produced is erroneous.

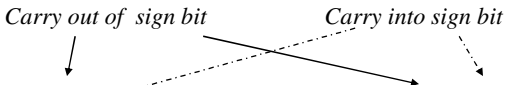
Examples Of No Overflow



Carries:	0	0		0	0
+70		0 1000110		+70	0 1000110
-80		1 0110000		+20	0 0010100
-10		1 1110110		+90	0 1011010

If there is no overflow, the carry into the sign bit matches the carry out of the sign bit.

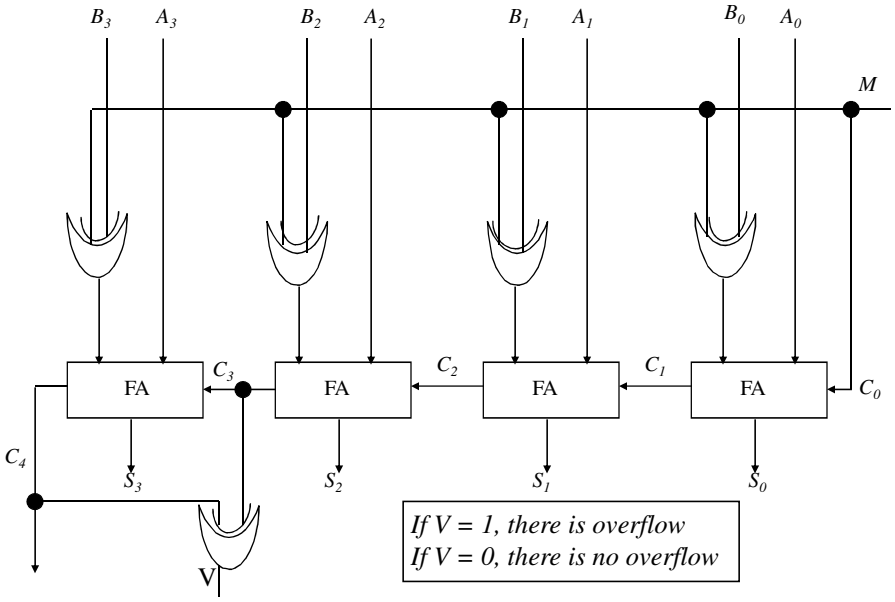
Examples Of Overflow



Carries:	0	1		1	0
+70		0 1000110		-70	1 0111010
+80		0 1010000		-80	1 0110000
+150		1 0010110		-150	0 1101010

If there ***is*** overflow, the carry into the sign bit ***does not*** match the carry out of the sign bit.

4-Bit Adder-Subtractor With Overflow Output



Decimal Adder

- Some systems perform arithmetic on decimal values, which are stored in BCD form.
- A decimal adder requires 9 inputs: 4 bits for each decimal digit of the augend and a carry bit.
- The easiest way to construct a decimal adder is by using a binary adder and then convert the sum to decimal form.
- The five inputs are K (the binary carry), Z_8, Z_4, Z_2 and Z_1.
- The five outputs are C (the decimal carry), S_8, S_4, S_2 and S_1.

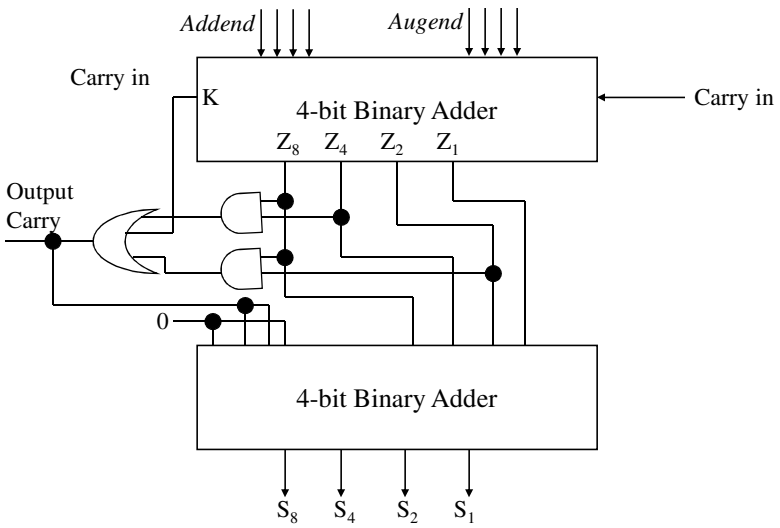
The Truth Table for the BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9

The Truth Table for the BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder

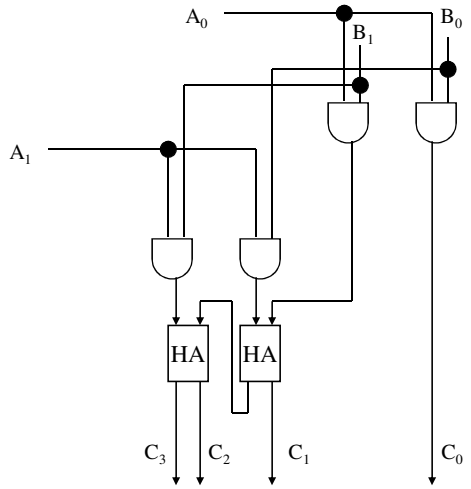


Binary Multiplier

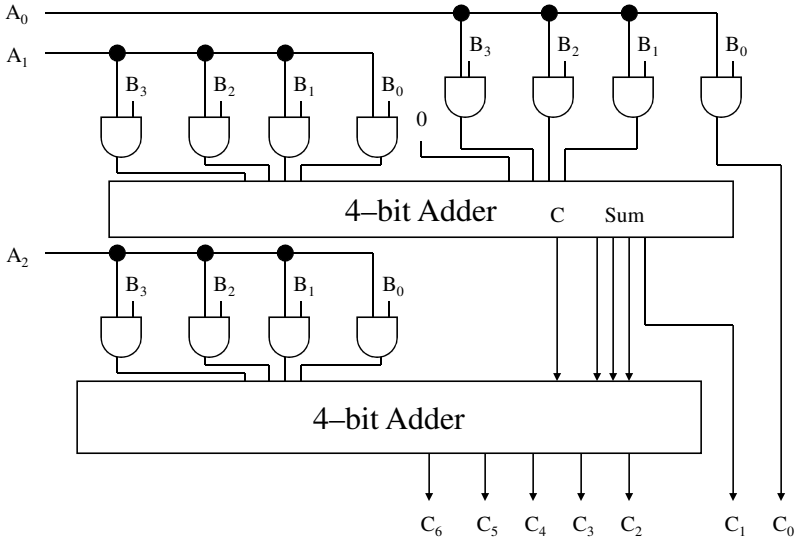
- Binary multiplication is performed the same way as decimal multiplication, except each line is either the multiplicand shifted or all zeros, depending on whether the multiplier bit is 1 or 0.
- Example:

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

2-Bit Multiplier



4-Bit by 3-bit Multiplier



Magnitude Comparator

- A magnitude comparator is a combinational circuit that compares two numbers, A and B and determines their relative magnitudes.
- The output is three variables that indicate whether A = B, A > B or A < B.

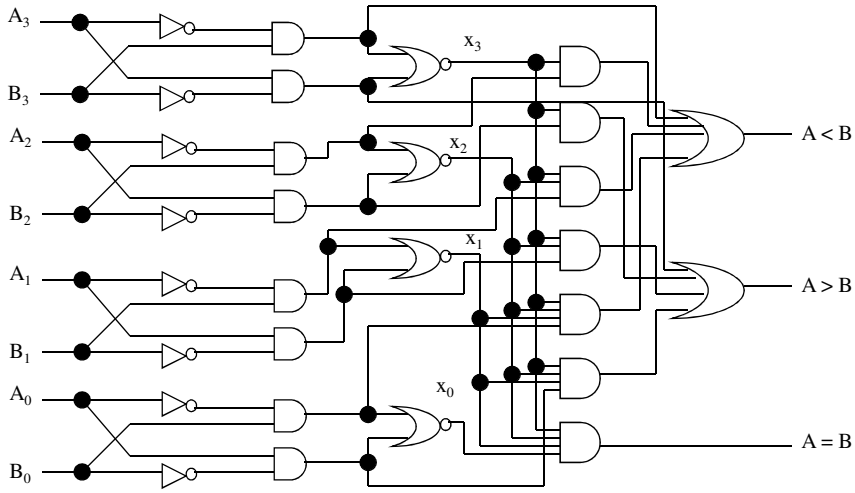
Magnitude Comparator – The Algorithm

- If $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$ then if we define $x_i = A_iB_i + A'_iB'_i$ where i = 0, 1, 2, 3
- A = B when $x_3x_2x_1x_0 = 1$
- (A > B)

  $= A_3B'_3 + x_3A_2B'_2 + x_3x_2A_1B'_1 + x_3x_2x_1A_0B'_0$
- (A < B)

  $= A'_3B_3 + x_3A'_2B_2 + x_3x_2A'_1B_1 + x_3x_2x_1A'_0B_0$

Magnitude Comparator – The Circuit



Decoders

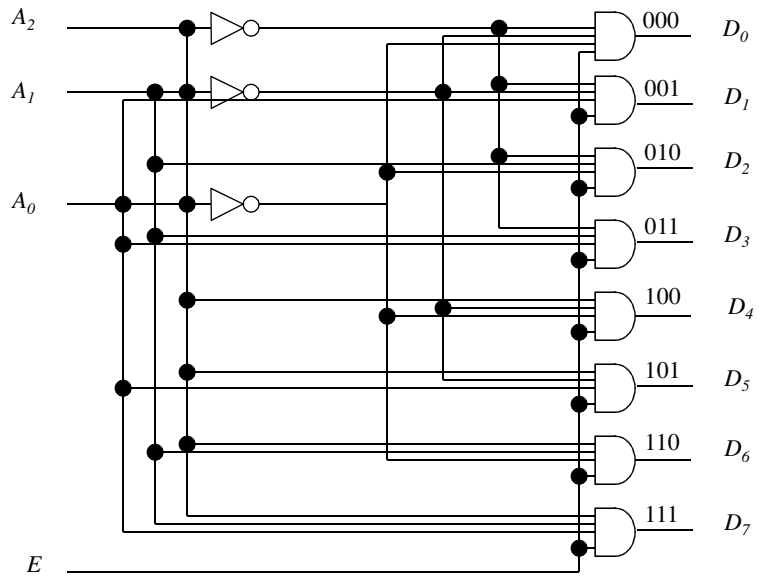
- A decoder is a combinational circuits that converts binary information from the n coded inputs to a maximum of 2^n unique outputs.
- The decoders in which that we are interested are n-to-m-line decoders, where $2^n \geq m$.
- Commercial decoders usually include an enable input, without which there is no response from the decoder.

Truth Table For For a 3-to-8-Line Decoder

Enable | Inputs | Outputs

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

3-to-8 Decoder



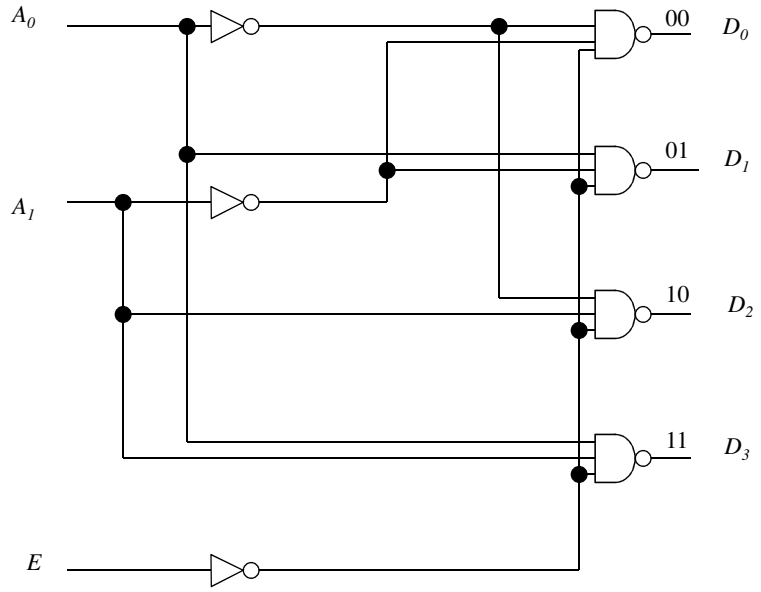
NAND Gate Decoder

- Some decoders are constructed with NAND gates instead of AND gates.
- Since NAND gates invert the outputs, it is more economical to invert the signals, i.e., E has a value 0 to enable the circuit and 1 to disable, and there is only one output D_i with a value of 0.

2-to-4-Line NAND Gate Decoder Truth Table

E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

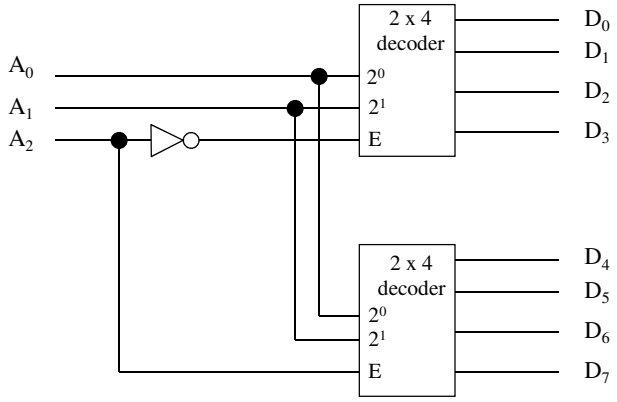
2-to-4 Decoder With NAND Gates



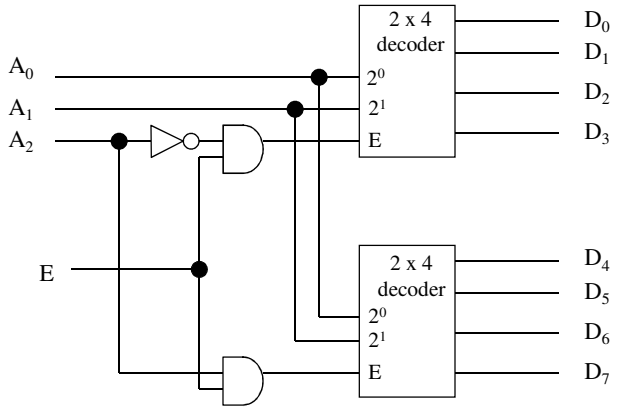
Expanding Decoders

- Sometimes a decoder may be needed but only smaller decoders are available.
- Take the example of using 2-to-4 decoders to build a 3-to-8 decoder:
  - The less significant inputs are attached to both decoders.
  - A_2 is used as E for the lower decoder and E' in the higher decoder.

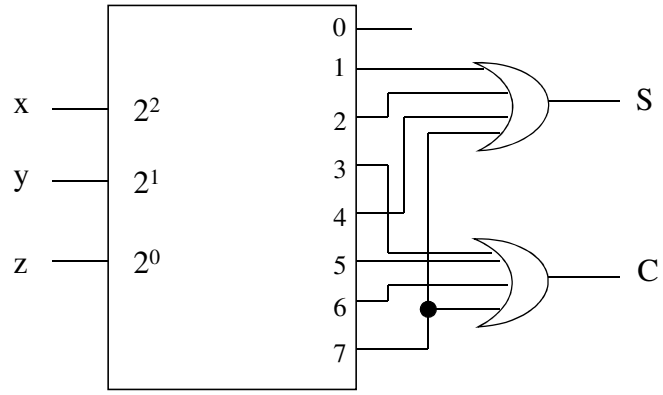
3-to-8 Decoder Constructed With Two 2-to-4 Decoders With Enable



3-to-8 Decoder Constructed With Two 2-to-4 Decoders



Implementing A Full Adder With a Decoder



Encoders

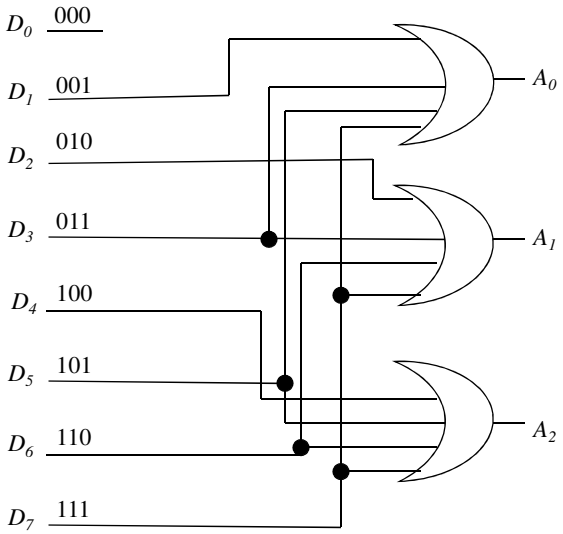
- An encoder does the opposite of a decoder
- An encoder has 2^n (or less) inputs and n outputs.
- An encoder can be implemented using OR gates whose inputs are determined from the truth table.:

  $A_0 = D_1 + D_3 + D_5 + D_7$

  $A_1 = D_2 + D_3 + D_6 + D_7$

  $A_2 = D_4 + D_5 + D_6 + D_7$

Encoders



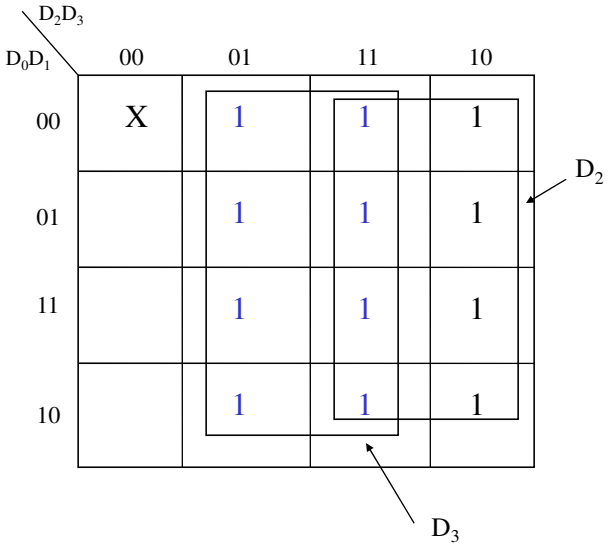
Priority Encoder

- A priority encoder is an encoder with a priority function.
- If two or more inputs are both set, the input with the highest priority takes precedence.
- The outputs x and y indicate the encoded bit; the output V is a valid bit indicator, which is set when one or more bits are set to 1.

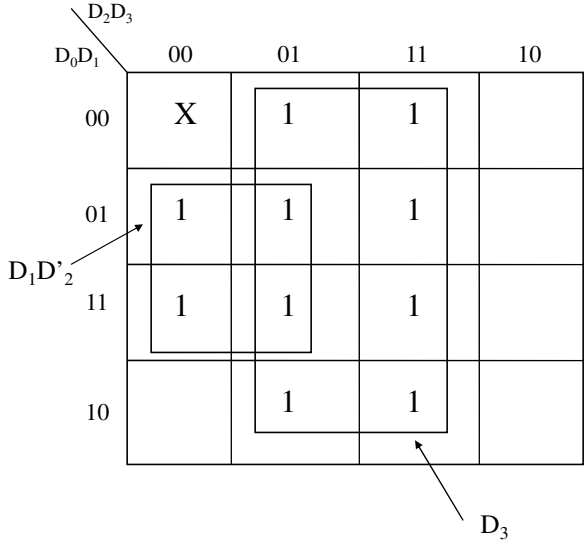
Truth Table For A Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

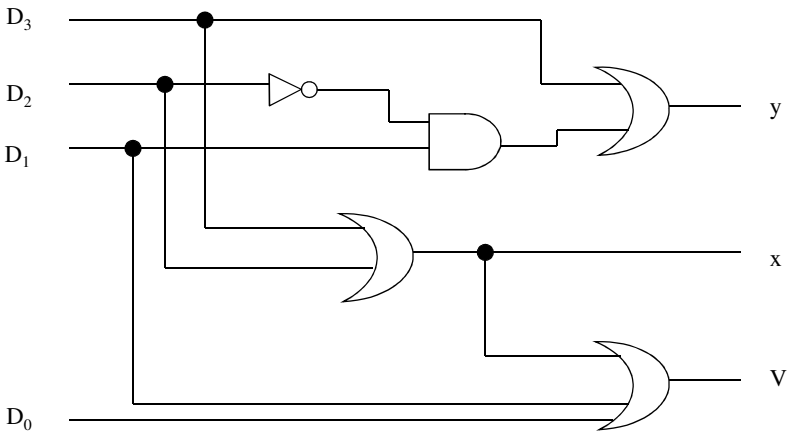
Karnaugh Map for X



Karnaugh Map for Y



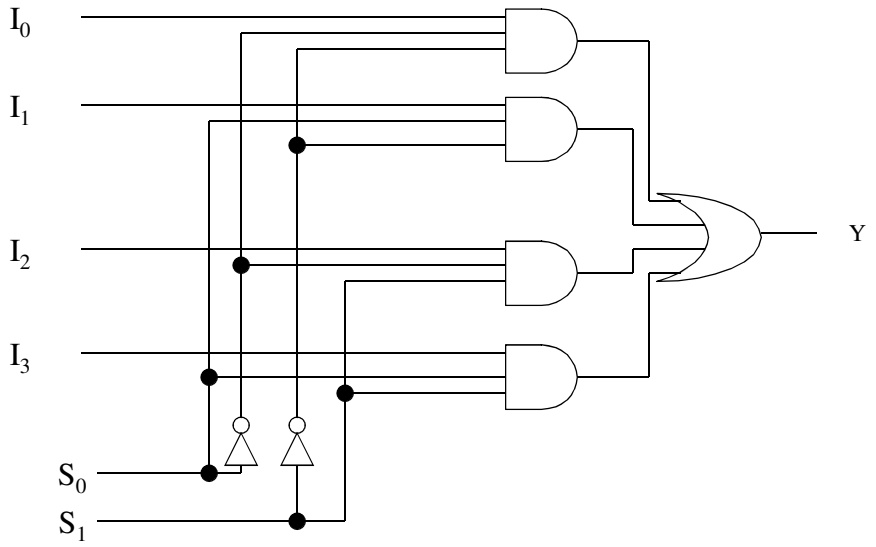
4-input Priority Encoder



Multiplexers

- A multiplexer allows 2^n input lines to share a single output line.
  - The line currently using the common output line is indicated by the select line inputs.
  - The select lines are decoded to determine which input has use of the line.
- A 4-to-1-line multiplexer has six inputs (four data inputs and two select inputs) and one output.
  - The truth table would require 64 lines.
  - It is more efficient and just as informative to use a function table, where we indicate by function what the output will be.

4-to-1-Line Multiplexer



4-to-1-Line Multiplexer Function Table

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Quadruple 2-to-1 Line Multiplexer