Arithmetic and Combinational Circuits

Instructor: Saraju P. Mohanty

- Number representation
- Adder and Circuits
- Multiplier Circuits
- Multiplexers
- Decoders
- Code Converters

Note: The slides are from text or reference book authors or publishers.

Data Representation

- There are basically three types of data:
  - Integer
  - Floating-Point
  - Character
- The Integer type is simple.
- The Floating-Point type is bit complicated (IEEE 754 representation)
- The characters are generally encoded (e.g. ASCII encoded)
- Types of Number Systems
  - Decimal Number System
  - Binary Number System
  - Octal Number System
  - Hexadecimal Number System

Number Systems : General

- A number in base r contains r digits 0, 1, 2, …, r-1, expressed as a power series in r has a general form:

$$A_{n-1}r^{n-1} + A_{n-2}r^{n-2} + \ldots A_1 r^1 + A_0 r^0 + A_{-1}r^{-1} + A_{-2}r^{-2} + \ldots A_{-m+1}r^{-m+1} + A_{-m}r^{-m}$$

- Expressed in positional notation:

$$A_{n-1}A_{n-2}\ldots.A_1A_0 \,.\, A_{-1}A_{-2}\ldots A_{-m+1}A_{-m}$$

- The "." is called the radix point.
- A_{n-1} is referred to as the Most Significant Digit (MSD).
- A_{-m} is referred to as the Least Significant Digit (LSD).
- Uses r distinct digits, hence said to be of base or radix r.
- When m = 0, the LSD $A_{-0} = A_0$.
- To distinguish between number systems enclose the coefficients in parentheses and place subscript after right parenthesis to indicate the base of the number, for example $(23)_{10}$ is a decimal number.

Number Systems : Decimal

- Uses 10 distinct digits, hence said to be of base 10.
- A decimal number (of base 10) with n digits to the left and m to the right of the decimal point is represented as:

  $A_{n-1}A_{n-2}\ldots.A_1A_0 . A_{-1}A_{-2}\ldots A_{-m+1}A_{-m}$

  Each A_i is one of {0, 1,2…,9}, i gives the position of the coefficient, hence, the weight 10^i by which the coefficient must be multiplied.
- Example: $(123.4)_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1}$
- kilo or K – 10^3, mega or M – 10^6, and Giga or G – 10^9

Number Systems : Binary

- Binary number system is a base-2 system with two digits : 0 and 1.
- Most popular representation in digital systems.
- The digits are called bits and the radix point is called the binary point.
- kilo or K – 2^{10}, mega or M – 2^{20}, and Giga or G – 2^{30}
- There are three ways of representing binary numbers :
  - Sign-and-Magnitude
  - 1's Complement
  - 2's Complement

Number Systems : Binary ……

- Both positive and negative numbers can be represented using any of the above three ways.
- In all three methods positive number has identical representations.
- The negative values have different representations.
- In all three ways, if the left most bit is “0” , then the number is positive, whereas if the left most bit is “1” , then number is negative.

Sign-and-Magnitude Binary System

It is simply an ordinary binary number with one extra digit placed in front to represent the sign. If this extra digit is a "1" , it means that the rest of the digits represent a negative number. If the extra digit is a "0" , it means that the number is a positive one.

Example: Let us assume that we have an 8-bit register. This means that we have one bit to represent the sign of the number (the **Sign Bit**) and rest 7-bits to represent a number. The numbers are represented as below.

0	0	1	0	0	1	0	1

The left most "0" bit means that the number is positive. The rest of the digits represent $(37)_{10}$. Thus, the number represented is $(+37)_{10}$.

1	0	1	0	0	1	0	1

The left most "1" bit means that the number is negative. The rest of the digits represent $(37)_{10}$. Thus, the number represented is $(-37)_{10}$.

Binary System : 1's Complement

The negative values are obtained by complementing each bit of the corresponding positive number.

0	0	0	1	1	0	1	1

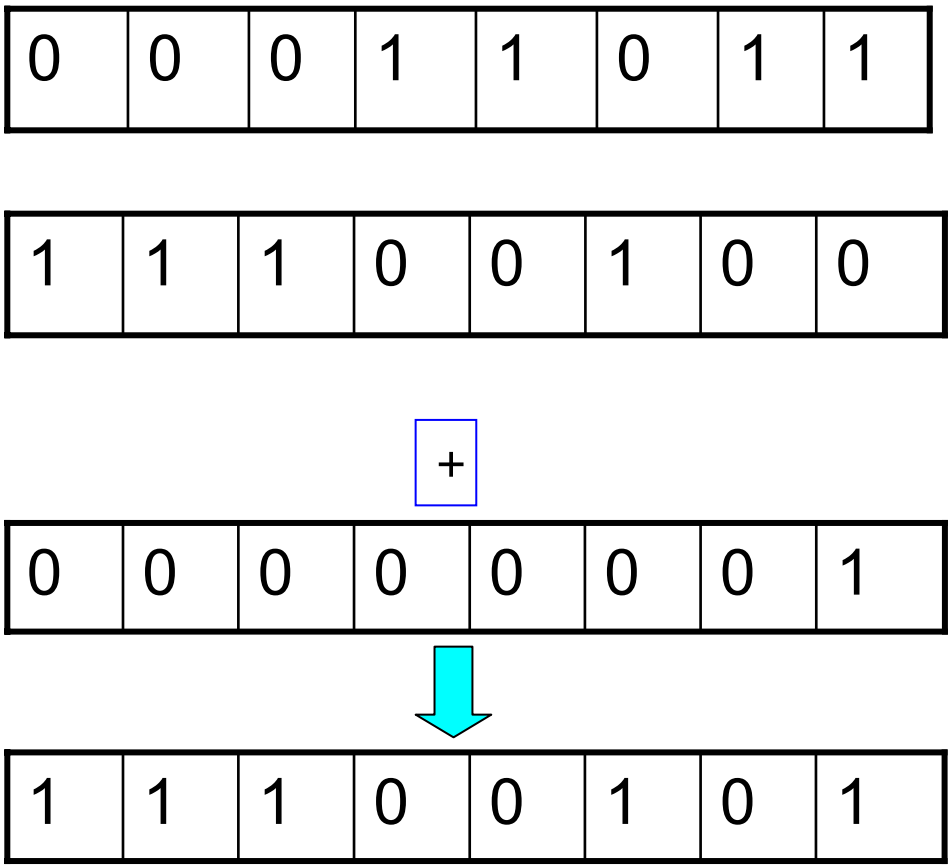
The number represented is $(+27)_{10}$.

1	1	1	0	0	1	0	0

The number represented is $(-27)_{10}$.

Binary System : 2's Complement

2's complement of a number is obtained by adding "1" to its 1's complement of the number.



Binary System : 2's Complement ……..

- Every computer system uses 2's complement binary representation. Subtraction can be performed using adders, since (a-b) = a + (-b) = a + ($b^{'}$+1) = a + $b^{'}$+1, thus reducing complexity of hardware implementation.
- Let us assume a 32-bit number : $(a_{31}a_{30}a_{29}a_{28}$ ……..$a_2a_1a_0)_2$. The decimal value of this number is (a_{31} x $(-2)^{31}$ + a_{30} x 2^{30} + a_{29} x 2^{29} + ……. + a_1 x 2^1 + a_0 x 2^0)$_{10}$.
- Example : Decimal value of the 32-bit 2's complement number shown below is $(-4)_{10}$.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Number Systems : Octal

- Used for compact representation of binary numbers.
- More convenient to people than using a bit strings in binary.
- The octal number system has radix or base of 8 and uses digits 0, 1, 2, 3, 4, 5, 6, 7.
- Example : $(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$.
- Note : The digits 8 and 9 are forbidden.

Number Systems : Hexadecimal

- More compact representation of binary numbers.
- More convenient to people than using a bit strings in binary.
- The hexadecimal number system has radix or base of 16 and uses digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.
- The letters A, B, C, D, E, and F are used for 10, 11, 12, 13, 14, and 15, respectively.
- Example : $(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$.

Ranges of Different Numbers

- The range of numbers that can be represented is based on the number of bits available in the hardware structures that store and process information.
- For a computer processing 16-bit unsigned integers, the range of integers is from 0 to $2^{16} - 1$, i.e. from 0 to 65,535.
- For the above computer the range of fractions that can be represented is from 0 to $(2^{16} - 1)/2^{16}$, i.e. from 0.0 to 0.999984712.
- For a computer using 32-bit 2's complement representation the number $(-2{,}147{,}483{,}648)_{10}$, has no corresponding positive number.

Conversion Among Different Number Systems

- Sign Extension of a Binary Number
- Binary to Octal and vice versa
- Binary to Hexadecimal and vice versa
- Decimal Integer to Binary
- Decimal Integer to Octal
- Decimal fraction to Binary
- Decimal fraction to Octal

Conversion : Sign Extension

• Sign extension is conversion of a binary number in *n* bits to a number with *more than n* bits.

•How to do ? Take the MSB from the smaller quantity and replicate it to the new bits of the larger quantity, and then copy the old bits of the smaller quantity into the right portion of the new word.

•Example :

4 bits Number	8-bit Number
$(0010)_2$	$(0000\ 0010)_2$
$(1010)_2$	$(1111\ 1010)_2$

Conversion : Binary ⟺ Hexadecimal

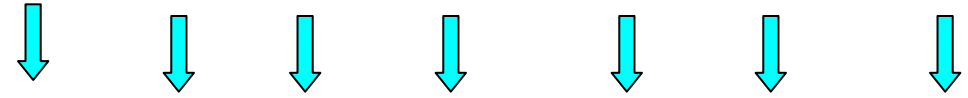
•Partition the binary number into groups of four each starting from the binary point and proceed to the left and to the right.

•Assign hexadecimal digit to each group.

•Example:

$(0010110001101011 . 1111000001 10)_2$ (Binary Number)

$(0010\ 1100\ 0110\ 1011 . 1111\ 0000\ 0110)_2$ (Grouped into 4)



$(2\quad C\quad 6\quad B\quad .\quad F\quad 0\quad 6)_{16}$ (Hexadecimal Digits)

$(2C6B.F06)_{16}$ (Hex Number)

•The above process can be reversed to convert from the hexadecimal numbers to binary numbers.

•Binary ⟺ Octal: Partition the binary number into groups of three each starting from the binary point and proceed to the left and to the right.

Conversion : Some General Rules

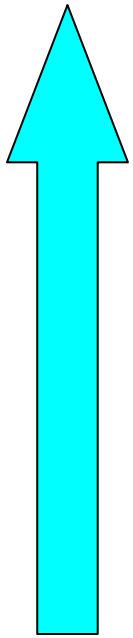
- The conversion of a number in base *r* to a decimal is done by expanding the number in a power series of *r* and adding all the terms.
- The reverse process, i.e. conversion of decimal integer to any base *r* is bit complicated.
- In the number contains a decimal point, then the number has to be divided into integer part and fraction part, and the conversion of the two parts should be done separately.
- The conversion of decimal integer to base *r* is done by dividing the number and the successive coefficients by *r*, and accumulating the remainders.
- The conversion of the decimal fraction to base *r* is done by multiplying the number and the fraction part of the successive products, and accumulating the integer part of the successive products.

Conversion : Decimal Integer to Binary

Let us convert $(1697)_{10}$ to base 2 :

1697	/ 2	=	848	remainder 1
848	/ 2	=	424	remainder 0
424	/ 2	=	212	remainder 0
212	/ 2	=	106	remainder 0
106	/ 2	=	53	remainder 0
53	/ 2	=	26	remainder 1
26	/ 2	=	13	remainder 0
13	/ 2	=	6	remainder 1
6	/ 2	=	3	remainder 0
3	/ 2	=	1	remainder 1
1	/ 2	=	0	remainder 1

(Least Significant Digit)



(Most Significant Digit)

$(1697)_{10} = (11010100001)_2$

Note : The same procedure can be used to convert an decimal integer to base 16 or 8, by dividing the decimal integer by 16 or 8, instead of 2.

Conversion : Decimal Integer to Hexadecimal

Let us convert $(44978)_{10}$ to base 16 :

Division	Quotient	Remainder	Hexadecimal Number
44978 / 16	2811	2	2
2811 / 16	175	11	B2
175 / 16	10	15	FB2
10 / 16	0	10	0AFB2

$$(44978)_{10} = (0AFB2)_{16}$$

Conversion : Decimal Fraction to Binary

Let us convert $(0.375)_{10}$ to binary fraction. Following are the steps to convert this number to binary using repeated multiplication by 2.

Step1 : Multiply 0.375 by 2 to find the MSB. Since our result is less than 1, the MSB in our answer is 0. 0.375*2 = 0.75 Answer: 0 . 0 ? ?

Step2: Take the fractional part of the previous result (0.75) and multiply by 2 again. Now the result is greater than 1, so the next bit of is 1. 0.75*2 = 1.5 Answer: 0 . 0 1 ?

Step3: Take the fractional part of the previous result (0.5) and multiply by 2. This time the result is exactly 1, so the LSB is 1. Since the fractional part of our result is 0, this is the last multiplication needed to find our answer. 0.5*2 = 1.0 Answer: 0 . 0 1 1

$$(0.375)_{10} = (0.011)_2$$

Conversion : Decimal Fraction to Hexadecimal

To convert a decimal fraction to hexadecimal, *multiply* the fraction by 16. Extract everything that appears to the left of the radix point. The first number extracted will be the MSD and will the last number extracted will be the LSD.

Example : Convert decimal $(0.513)_{10}$ to a three digit hexadecimal fraction.

0.513 x 16 = 8.208	Integer = 8	(MSD)
0.208 x 16 = 3.328	= 3	
0.328 x 16 = 0.052	= 0	
0.052 x 16 = 0.832	= 0	(LSD)

$$(0.513)_{10} = (0.830)_{16}$$

Floating Point Numbers

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576×10^9
- Representation:
  - sign, exponent, significand: $(-1)^{sign} \times significand \times 2^{exponent}$
  - more bits for significand gives more accuracy
  - more bits for exponent increases range
- IEEE 754 floating point standard:
  - single precision: 8 bit exponent, 23 bit significand
  - double precision: 11 bit exponent, 52 bit significand

IEEE 754 floating-point standard

- Leading “1” bit of significand is implicit.
- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: $(-1)^{sign} \times (1+significand) \times 2^{exponent - bias}$
- Example:
  - decimal: -.75 = -3/4 = $-3/2^2$
  - binary: -.11 = -1.1×2^{-1}
  - floating point: exponent = 126 = 01111110
  - IEEE single precision: 10111111010000000000000000000000

Why Decimal or Alphanumeric Codes ??

Binary number system is used in computers, but people are more comfortable with decimal system. So, there has to be back and forth conversion between decimal and binary system. The computation is done in binary and human understanding is in decimal. This needs a method to store decimals (and characters) in a computer that can be converted to binary.

Decimal Codes

An n-bit binary code is a group of n-bits that assume up to 2^n distinct combinations of 1's and 0's, with each combination representing one element of the set being coded. The most commonly used for decimal is binary-coded-decimal (BCD).

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Binary-Coded Decimal (BCD)

Alphanumeric Codes

$B_4B_3B_2B_1$	$B_7B_6B_5$ = 000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control Characters:

NULL	NULL	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

American Standard Code for Information Interchange (ASCII)

ASCII is a 7-bit code, but one bit (“0”) is appended as MSB to make it a byte.

Some Arithmetic Terms

	Addition	Subtraction	Multiplication	Division
Numerator	**Augend**	**Minuend**	**Multiplicand**	**Dividend**
Denominator	**Addend**	**Subtrahend**	**Multiplier**	**Divisor**
Result	**Sum**	**Difference**	**Product**	**Quotient**

Arithmetic Operation: Binary addition

- The sum of two numbers is calculated following the same rules as for decimal numbers.
- Unlike the decimal system the sum of two bits has to be only 1 or 0 (in decimal : 0…..to 9).
- The carry occurs if the sum is greater than 1 (decimal carry occurs if sum is greater than 9).
- Example :

Carries : 00000	10110
Augend : 01100	10110
Addend : +10001	+10111
Sum : 11101	101101

- The second addition has a carry.

Arithmetic Operations : Binary subtraction

- The rules of binary subtraction are the same as that of decimal numbers.
- A borrow into a given column adds 2 to the minuend bit (a borrow in the decimal system adds 10 to the minuend digit.
- Example:

Borrows :	00000	00110
Minuend :	10110	10110
Subtrahend:	-10010	-10011
Difference :	00100	00011

- Subtraction can be also performed by adding the 2's complement of the subtrahend to the minuend.

Arithmetic Operations : Binary multiplication

- Example:

  Multiplicand: 1011

  Multiplier: x 101

  1011

  0000

  1011

  Product: 110111

- Observation : The multiplier bits are always 1 or 0, therefore the partial products are equal to either the multiplicand or to 0.
- The above fact has been exploited in various ways, and many time and hardware efficient multiplication algorithms have been developed.
- Booth's multiplier and Wallace-Tree multiplier are two examples.

Overflow handling during arithmetic operations

- Overflow occurs when the result too large for finite computer word (i.e. bit-width of registers and computational units).
- Overflow term is somewhat misleading, it does not mean a carry “overflowed”.
- No overflow when adding a positive and a negative number and when signs are the same for subtraction.
- Effects of Overflow : An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Many hardware supports are available with different computer architectures to handle the overflow.

Binary Adders

A combinatorial circuit that performs the addition of two bits is called a half adder. One that performs the addition of three bits (two significant bits and a previous carry is a full adder.

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table of Half Adder

Inputs			Outputs	
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

Truth Table of Full Adder

Binary Adders: Half adder

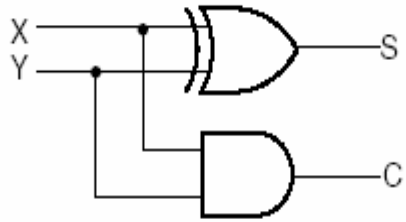
Boolean Functions :

S=X'Y+XY'=X XORY

C=XY

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table of Half Adder



Logic Diagram of Half Adder

Binary Adders : Full adder

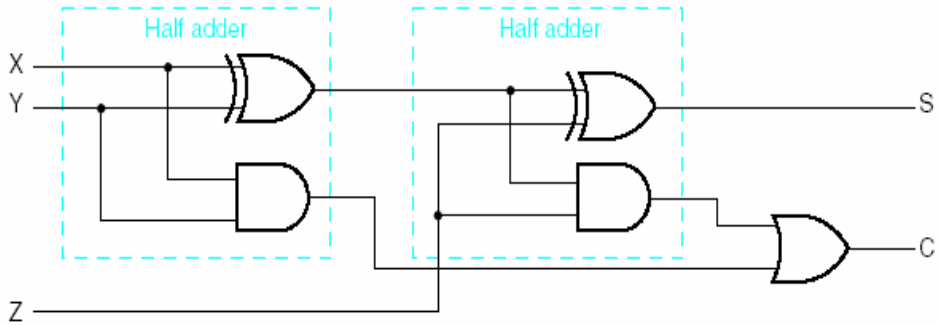
Boolean Functions:

$$S = X'Y'Z+X'YZ'+XY'Z'+XYZ = (X \text{ XOR } Y) \text{ XOR } Z$$

$$C = XY+XZ+YZ = XY+Z(X \text{ XOR } Y)$$

Inputs			Outputs	
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

Truth Table of Full Adder



Logic Diagram of Full Adder

Binary Ripple Carry adder

The full adders are connected in cascade, with the carry output from one full adder connected to the carry input of the next full adder.

Since a 1 carry may appear near the LSB of the adder and yet propagate through many full adders to the MSB → the name **ripple carry adder**. An n-bit ripple carry adder requires n full adders.

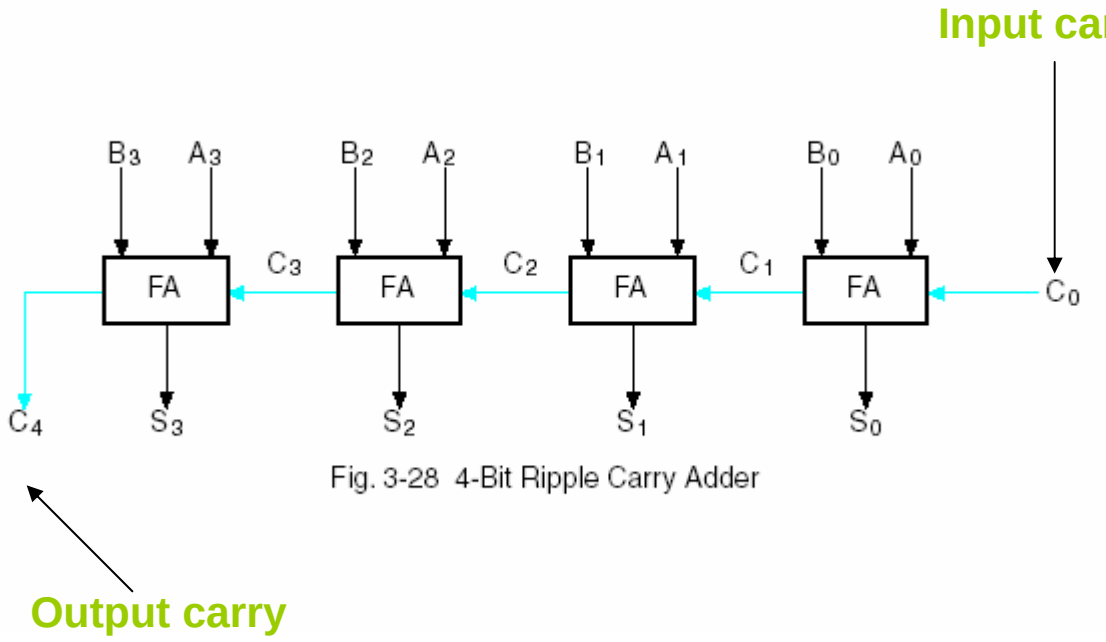


Fig. 3-28 4-Bit Ripple Carry Adder

Input carry:	0110
Augend A:	1011
Addend B:	0011
Sum S:	1110
Output carry:	0011

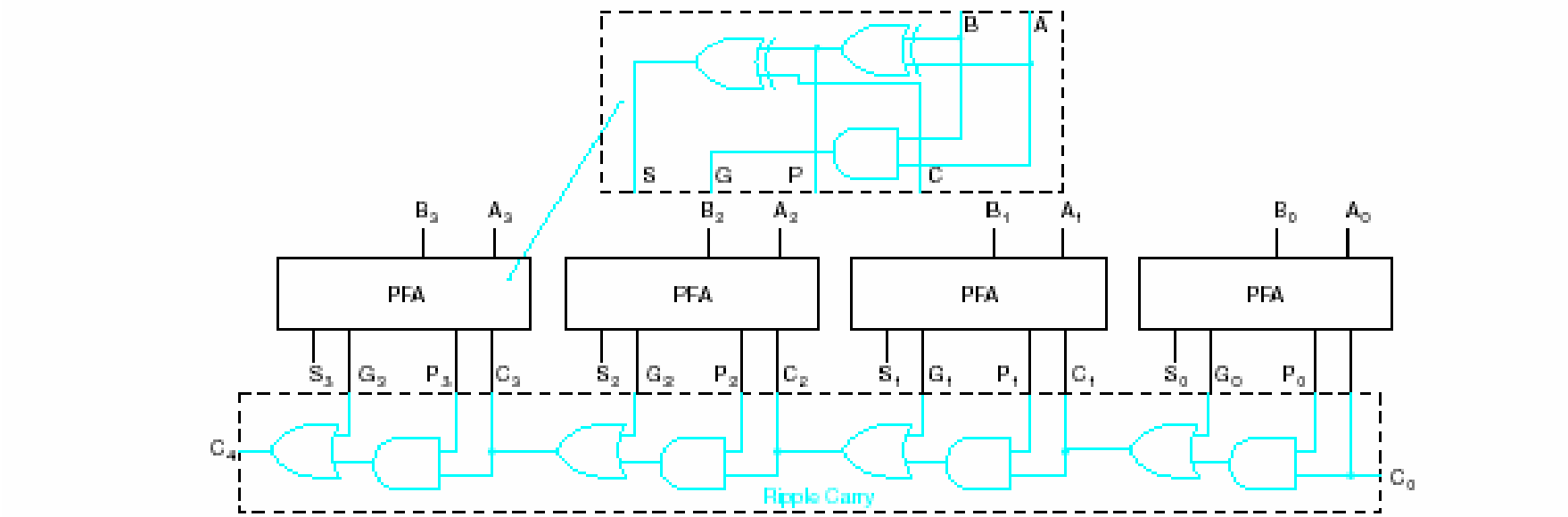
Disadvantage of Ripple Carry Adder

- The design of the 4-bit ripple carry adder with the usual method would require a truth table with 512 entries, since there are nine inputs to the circuit.
- Long circuit delay due to the many gates in the carry path from the LSB to the MSB.
- The longest path delay through an n-bit ripple carry adder is 2n+2 gate delays.
- Carry lookahead adder reduces critical path delay, but there is area penalty involved.

Carry Lookahead Adder

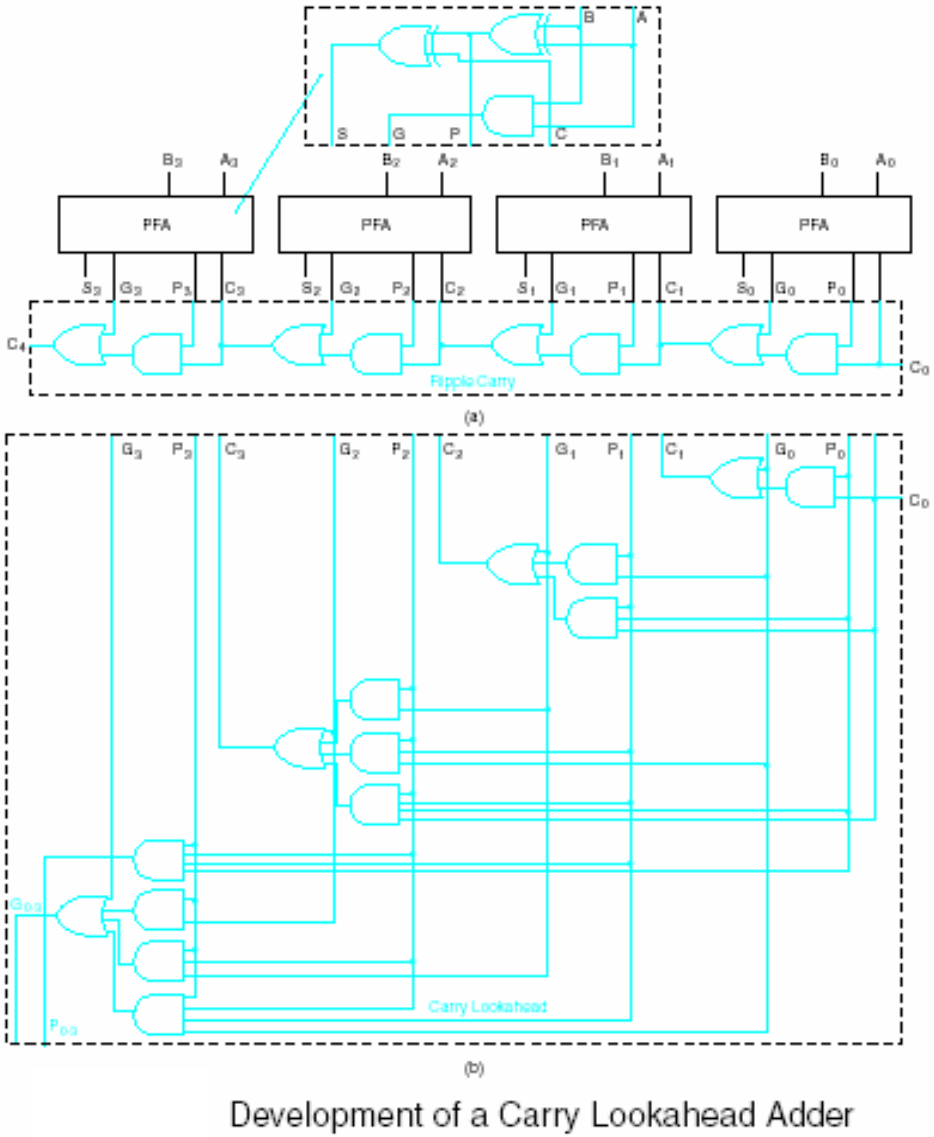
- Reduced delay at the price of more complex hardware.
- The design can be obtained by a transformation of the ripple carry design in which the carry logic over fixed groups of bits of the adder is reduced to two-level logic.
- Construct a new logic hierarchy separating the parts of the full adders not involving the carry propagation path from those containing the path.
- Call the first path of each full adder a **partial full adder** (PFA).

Carry Lookahead Adder: PFA model



- Two outputs: P_i, G_i from each PFA to the ripple carry path
- One input: C_i, the carry input, from the carry path to each PFA.
- Propagate function: $P_i = A_i \; XOR \; B_i$. When it is equal to 1 an incoming carry is propagated through the bit position from C_i, to C_{i+1}; when equal to zero, carry propagation is blocked.
- Generate function: $G_i = A_i * B_i$. Whenever equal to 1 regardless of the P_i value, the carry output from the position is 1, so a carry has been generated in the position. When equal to zero, no carry is generated, so that C_{i+1} is 0 if the carry propagated through the position from C_i is also zero.

Carry Lookahead Adder: PFA model



Development of a Carry Lookahead Adder

Ripple Carry Path

Carry Lookahead circuit will replace the ripple carry path above.

4-bit ripple carry adder has delay of 10 gate delays, for carry lookahead it is 6 gate delays. Assume XOR has 2 OR gate delays.

Adder-Subtractor Circuit

When S = 0 the circuit is an adder, and when S = 1 the circuit is an subtractor.

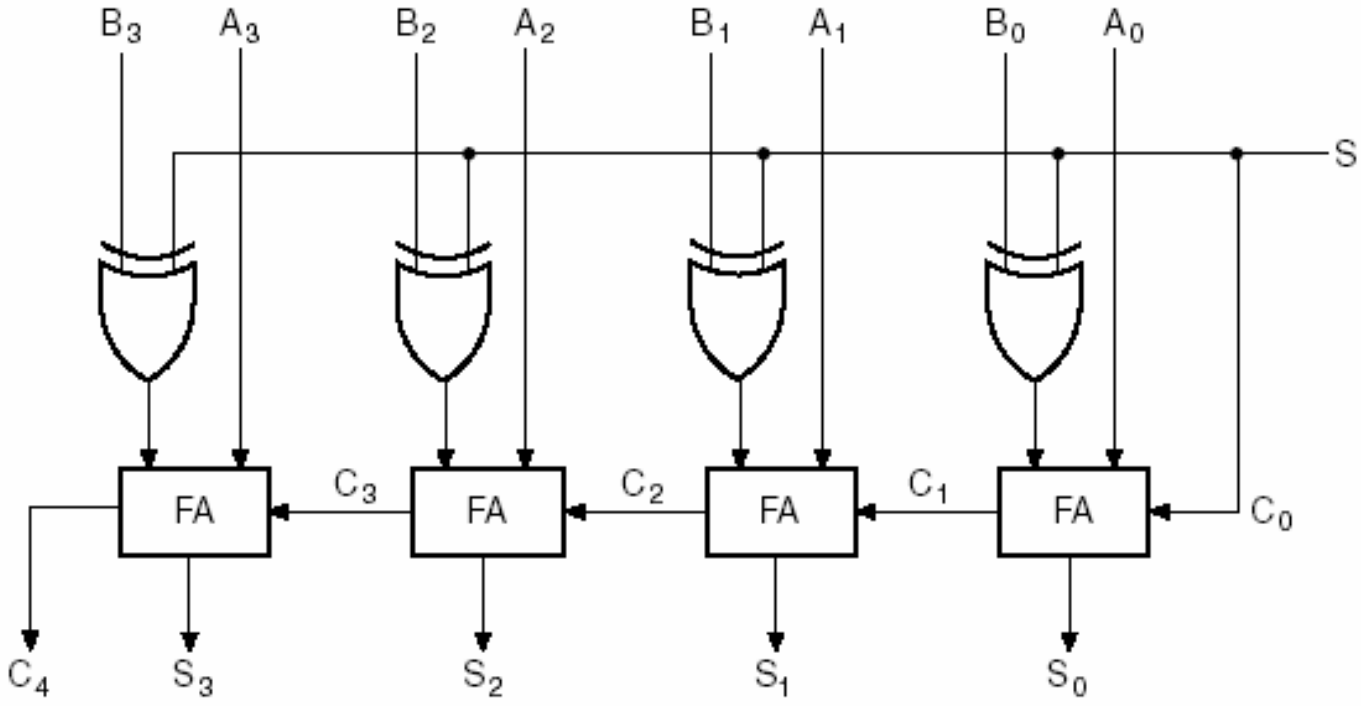
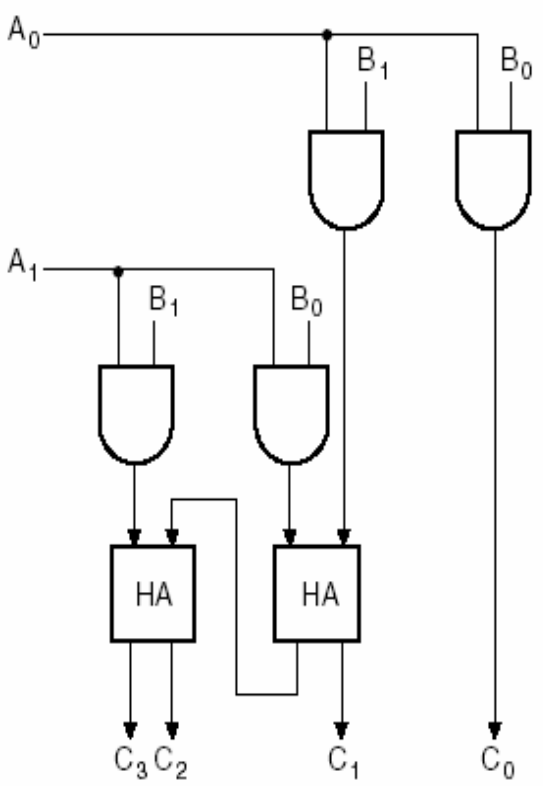


Fig. 3-31 Adder-Subtractor Circuit

Binary Multipliers: A 2-bit example

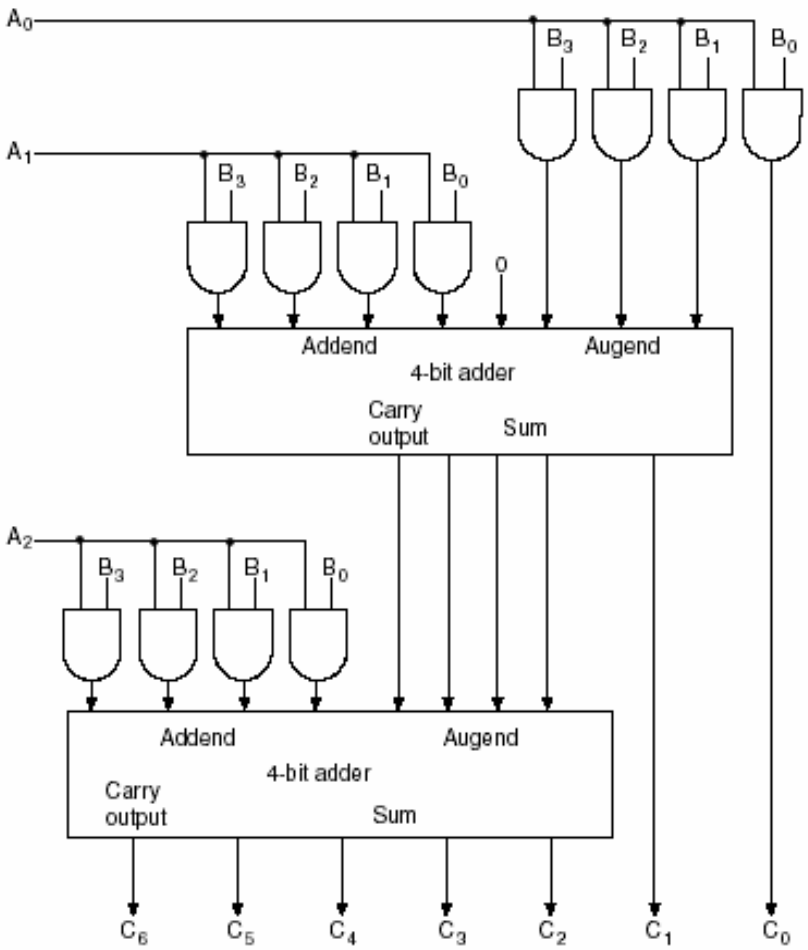
		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



A 2-Bit by 2-Bit Binary Multiplier

Product A_0 and B_0 is 1 if both are 1, else it is 0. Thus, the product is same as AND operation.

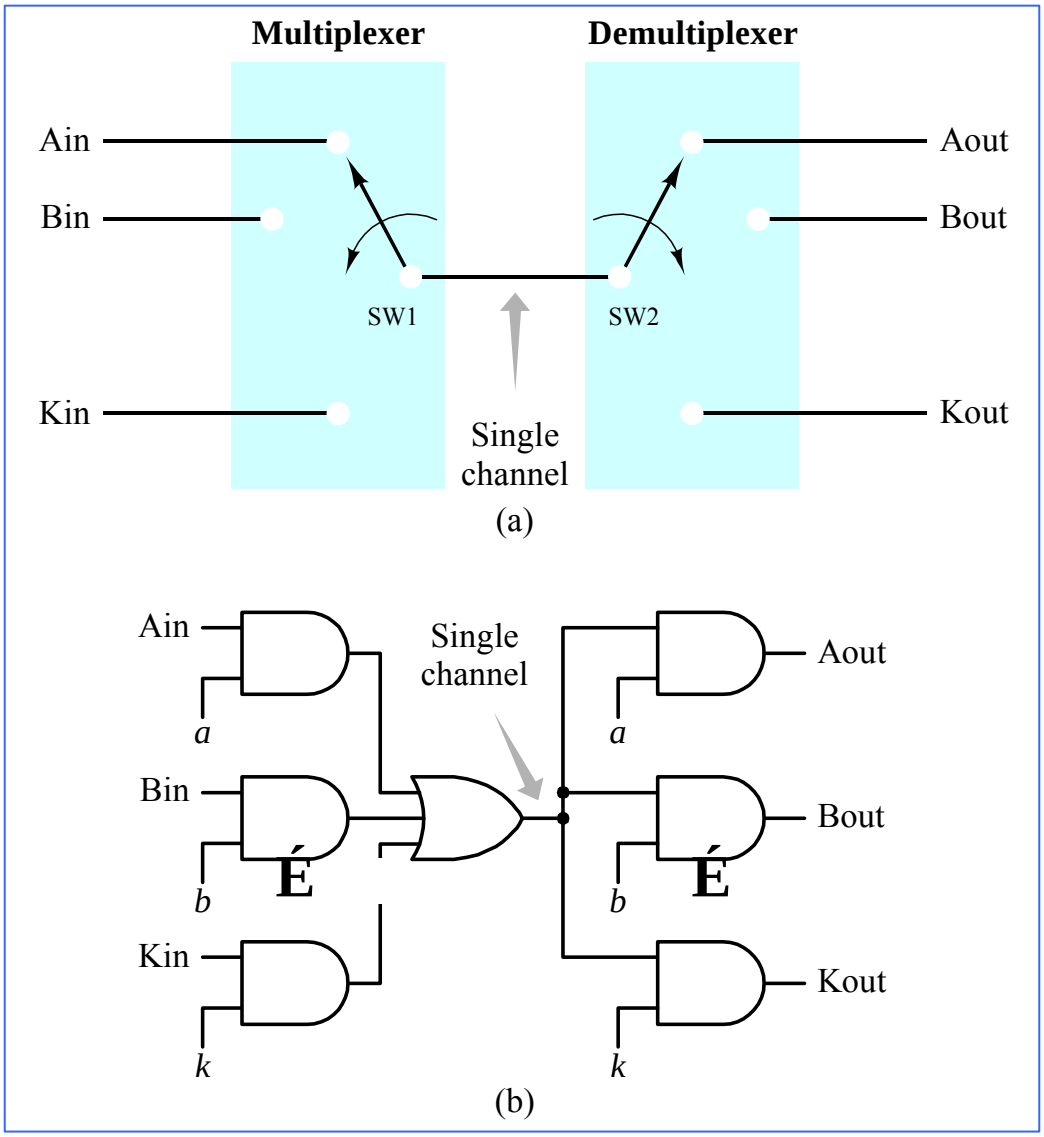
Binary Multipliers: A 4-bit by 3-bit example



A 4-Bit by 3-Bit Binary Multiplier

For J multiplier bits and K multiplicand bits, we need JxK AND gates and (J-1) K-bit adders to produce a product of J+K bits.

K-Channel Multiplexing / Demultiplexing



Multiplexing / Demultiplexing Operation

A Simple Logic Configuration

Multiplexers

•Selects binary information from one of many input lines and directs this information to a single output line.

•The selection of a particular input line is controlled by a set of input variables, called selection inputs. Normally, there are 2^n input lines and n selection inputs whose bit combinations determine which input is selected.

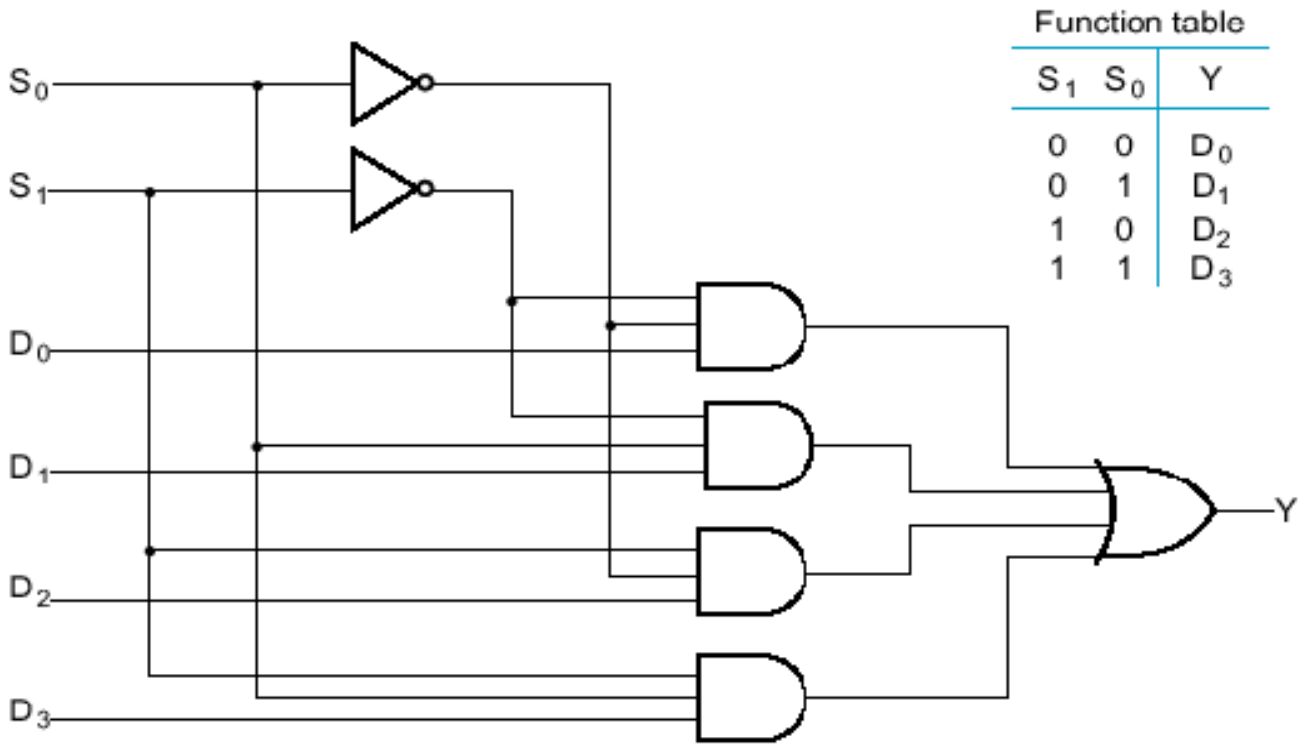
•In general a 2^n-to-1-line multiplexer is constructed from an n-to-2^n decoder by adding 2^n input lines to it, one from each data output.

•The size of the multiplexer is specified by the number 2^n of its data input lines and the single output line.

•As in decoders, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled.

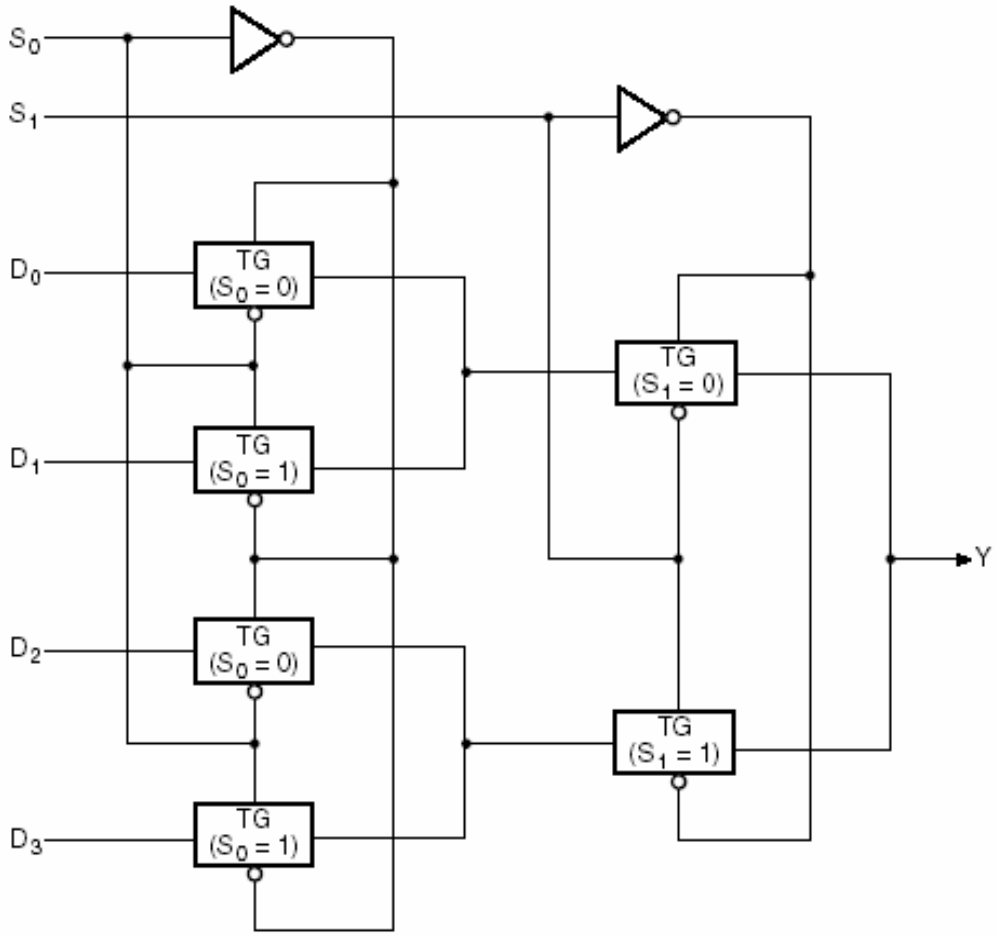
•The enable input is useful for expanding two or more multiplexers into a multiplexer with a larger number of inputs.

Multiplexers: A 4-to-1-line multiplexer



Each input applied to one input of an AND gate. Selection inputs S_0 and S_1 are decoded to select a particular AND gate. AND gate outputs are applied to a single OR gate to provide the 1-line output.

A 4-to-1-line Multiplexer using Trans. Gates



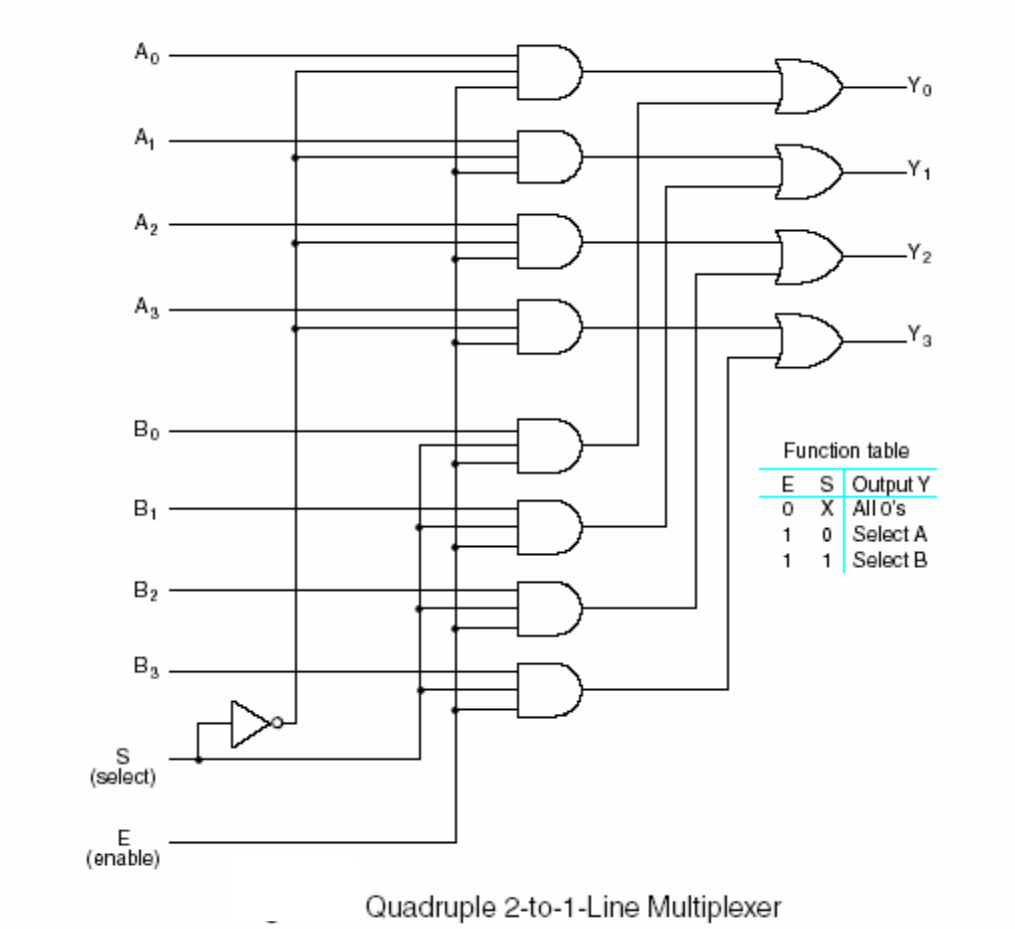
4-to-1-Line Multiplexer with Transmission Gates

Two selection lines S_0 and S_1 control the transmission paths.

For example, S0=0 and S1 =0, then Y = D0.

Quadruple 2-to-1-line Multiplexer

Multiplexer blocks can be combined in parallel with common selection and enable lines to perform selection on multiple-bit quantities.



Quadruple 2-to-1-Line Multiplexer

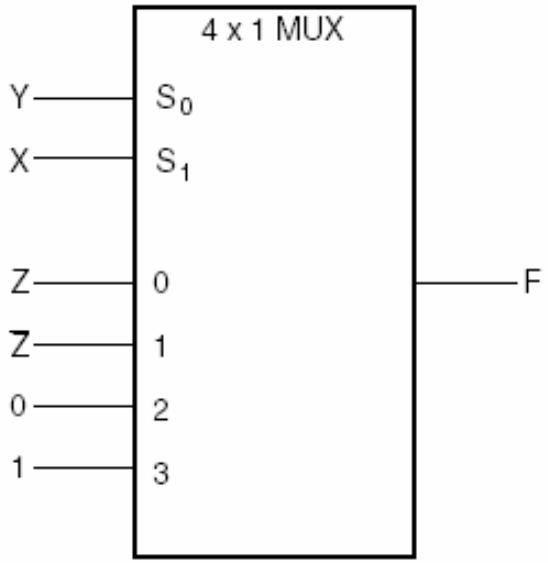
MUX: Combinatorial circuit implementation

- List the Boolean Function in a truth table.
- Apply first n-1 variables as selection inputs.
- For each combination of selection variables evaluate the output as a function of last variable, 0 and 1.

Example: Implementation of the function $F(X,Y,Z) = \Sigma m(1,2,6,7)$

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

(a) Truth table

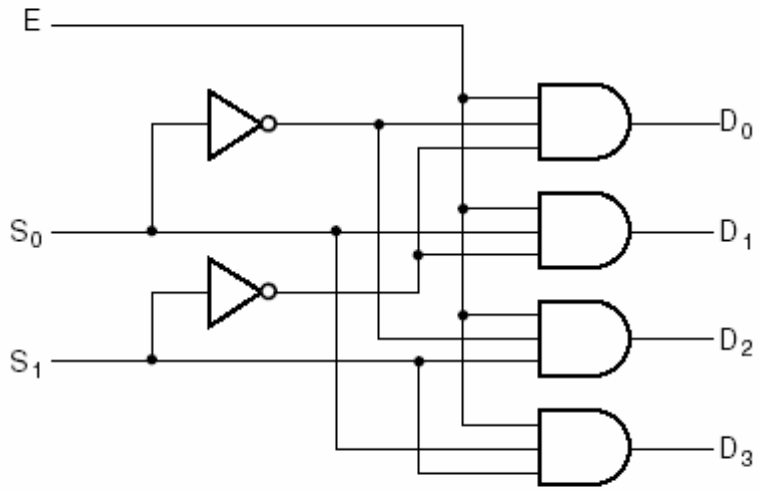


(b) Multiplexer implementation

Implementing a Boolean Function with a Multiplexer

Demultiplexer

Performs the opposite of a multiplexer. Receives information from a single input line and transmit it to one of 2^n possible output lines. The selection of the specific output is controlled by the bit combination of n selection lines.

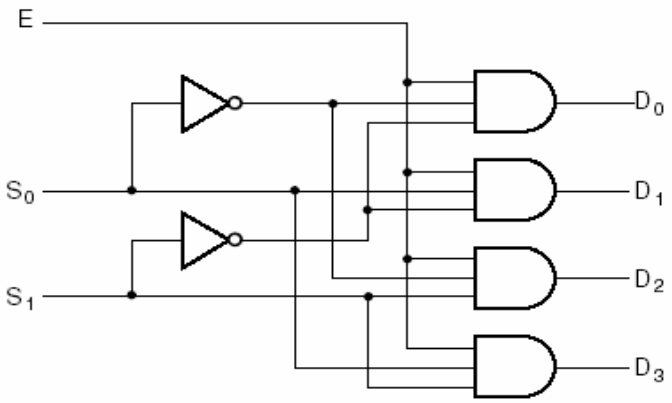


1-to-4-Line Demultiplexer

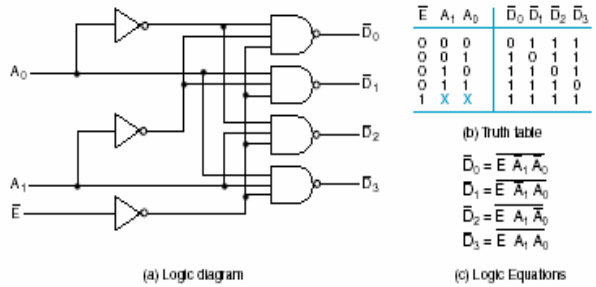
Example:

When $(S_1,S_0) = 10$, the output D_2 is input E

Decoder Vs Demultiplexer



1-to-4-Line Demultiplexer



A 2–to–4-Line Decoder

- Logic circuits are exactly the same
- Applications are different
- Decoder with enable input is referred to as decoder / demultiplexer

Decoders

- A decoder is a combinatorial circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs.
- If the n bit coded information has unused bit combinations, then the decoder has $m \leq 2^n$ outputs.
- Decoders are called n-to-m line decoders. Their purpose is to generate the 2^n or fewer minterms of n input variables.
- Decoder operation may be clarified by its corresponding truth table.
- Some decoders are constructed directly using NAND gates since this is more economical.

Decoder Example: 3-to-8 line decoder

•3 inputs are decoded to 8 outputs.

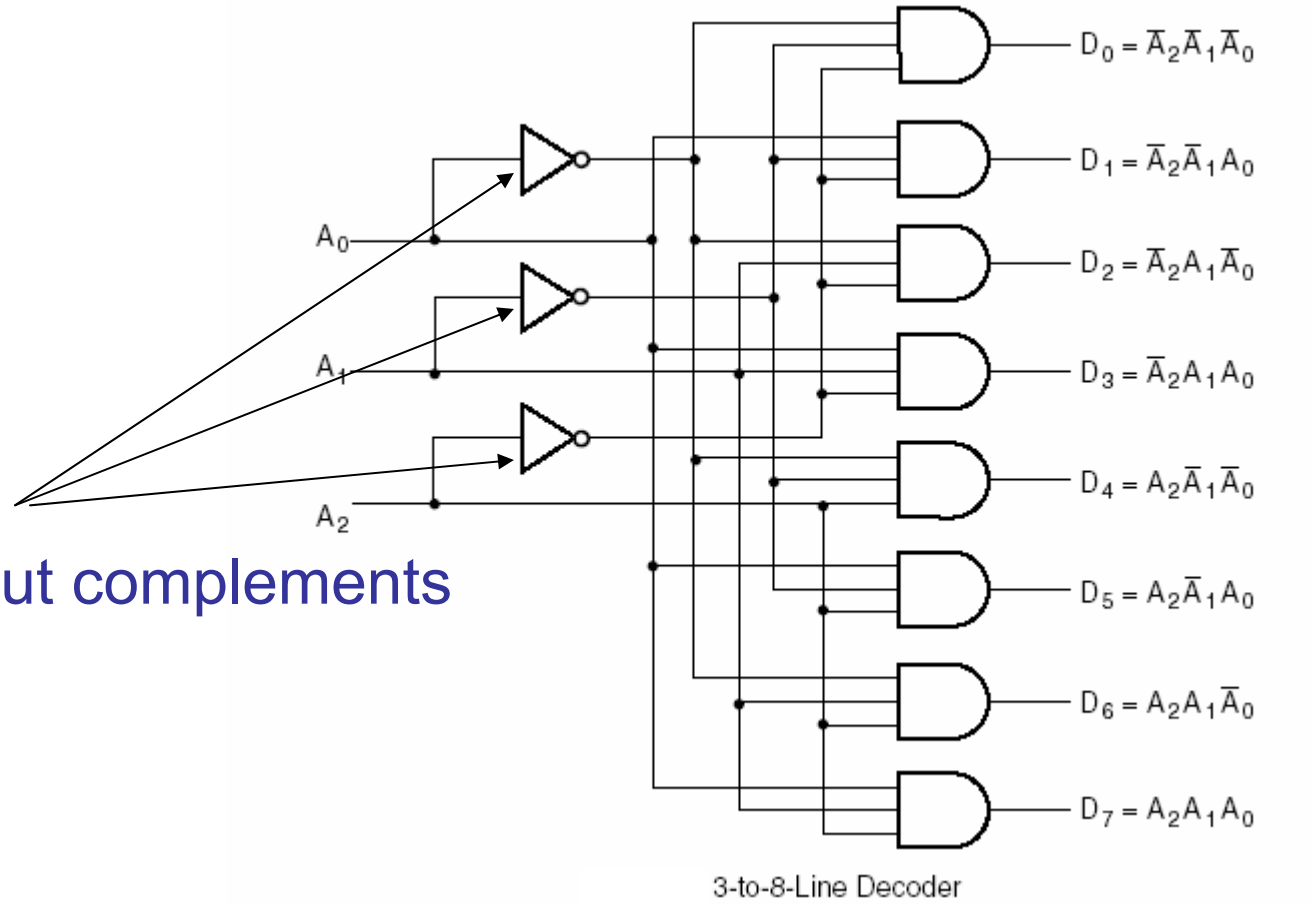
•For each possible input combination, there are 7 outputs that are equal to 0 and only one that is equal to 1.

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

Truth Table for 3–to–8-Line Decoder

Decoder Example: 3-to-8 line decoder ….

- 3 inverters provide complement of the outputs.
- Each one of the eight AND gates generate one minterm.



Input complements

3-to-8-Line Decoder

Decoder to Any Combinatorial Circuit

- Decoder provides the 2^n minterms of n input variables. Since any Boolean function can be expressed as a sum of minterms, one can use a decoder to generate the minterms and an external OR gate to form their logical sum.

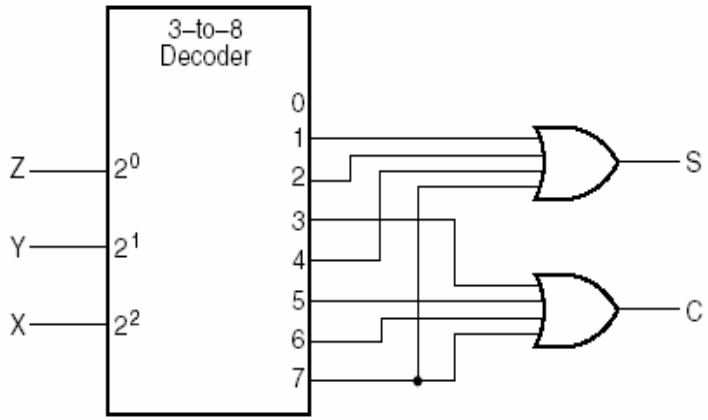
- Thus, any combinatorial circuit with n inputs and m outputs can be implemented with an n-to-2^n-line decoder and m OR gates.

- This procedure for implementing a combinatorial circuit by means of a decoder and OR gates, requires that the Boolean functions be expressed as a sum of minterms. This form can be obtained from the Truth Table or by plotting each function on a map. The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function.

Decoder to A Binary Full Adder

X	Y	Z	C	$\overline{C}$	T_1	T_2	T_3	S
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	1	0	0
1	1	1	1	0	1	1	0	1

Truth Table for Binary Adder



Implementing a Binary Adder Using a Decoder

Full Adder as Sum-of-Minterms :

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

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

Encoders ??

Performs the inverse operation of a decoder. It has 2^n (or fewer) input lines and n output lines.

Example: Octal-to-binary encoder. Eight inputs corresponds to eight octal digits. Only one input has the value 1 at any given time. It can be implemented with 3 4-input OR gates.

Inputs								Outputs		
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

Truth Table for Octal–to–Binary Encoder

Boolean Functions

$A_0 = D_1 + D_3 + D_5 + D_7$

$A_1 = D_2 + D_3 + D_6 + D_7$

$A_2 = D_4 + D_5 + D_6 + D_7$

NOTE: Design of a priority encoder can be considered similarly.