Jiangxi University of Science and Technology

DIGITAL DESIGN

Lecture 5: Coding and Logic gate introduction

Signed Binary Numbers

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Binary Numbers and Binary Coding

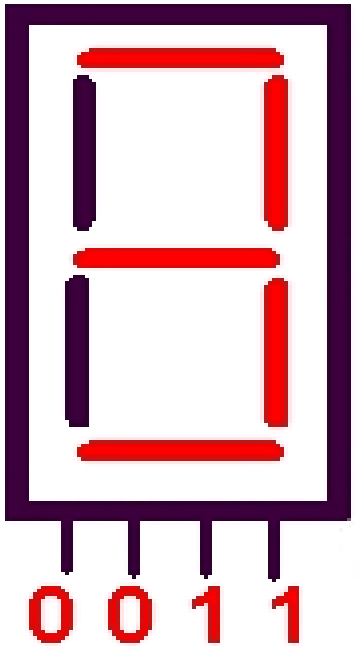
- Flexibility of representation
  - Within constraints below, can assign any binary combination (called a code word) to any data as long as data is uniquely encoded.
- Information Types
  - Numeric
    - Must represent range of data needed
    - Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted
    - Tight relation to binary numbers
  - Non-numeric
    - Greater flexibility since arithmetic operations not applied.
    - Not tied to binary numbers

Non-numeric Binary Codes

- Given *n* binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.
- Example: A binary code for the seven colors of the rainbow
- Code 100 is not used

Color	Binary Number
Red	000
Orange	001
Yellow	010
Green	011
Blue	101
Indigo	110
Violet	111

Binary Coded Decimal



Introduction:

Although binary data is the most efficient storage scheme; every bit pattern represents a unique, valid value.

However, some applications may not be desirable to work with binary data.

For instance, the internal components of digital clocks keep track of the time in binary.

The binary value must be converted to decimal before it can be displayed.

Because a digital clock is preferable to store the value as a series of decimal digits, where each digit is separately represented as its binary equivalent, the most common format used to represent decimal data is called **binary coded decimal**, or **BCD**.

Binary Coded Decimal (BCD)

- The BCD code is the 8,4,2,1 code.
- 8, 4, 2, and 1 are weights
- BCD is a *weighted* code
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.
- Example: 1001 (9) = 1000 (8) + 0001 (1)
- How many “invalid” code words are there?
- What are the “invalid” code words?

BCD Numeric Format

Every four bits represent one decimal digit.

- Use decimal values from **0** to **9**
- 4-bit values above 9 are not used in BCD.

The unused 4-bit values are:

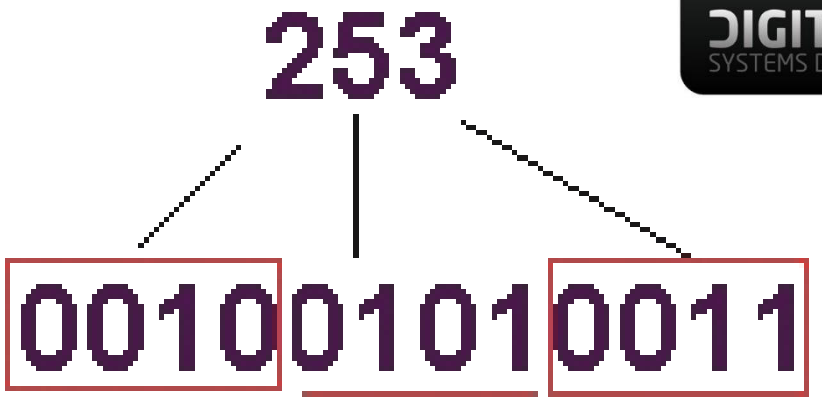
Table 1.4
Binary-Coded Decimal (BCD)

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

BCD	Decimal
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

BCD Numeric Format

Multi-digit decimal numbers are stored as multiple groups of 4 bits per digit.



BCD is a signed notation

- positive or negative.

  For example, +27 as 0(sign) 0010 0111.

  -27 as 1(sign) 0010 0111.

- BCD does not store negative numbers in two's complement.

4221 Code

- Below decimal 5 use the right-most bit representing 2 first
- Above decimal 5 use the left-most bit representing 2 first
- Decimal 5 = 2+2+1 and not 4+1

Decimal	4221	1's complement
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	1000	0111
5	0111	1000
6	1100	0011
7	1101	0010
8	1110	0001
9	1111	0000

Gray Code

- In pure binary coding or 8421 BCD then counting from 7 (0111) to 8 (1000) requires 4 bits to be changed simultaneously.
- Gray coding avoids this since only one bit changes between subsequent numbers

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Gray Code

IF $g_n\ g_{n-1}\cdots g_1\ g_0$ Gray Code → $b_n\ b_{n-1}\ldots b_1\ b_0$ Binary Code

$$g_i = b_i \oplus b_{i+1},\ 0 \le i \le n-1$$
$$g_n = b_n$$

$$b_{n-i} = g_n \oplus g_{n-1} \oplus \ldots \oplus g_{n-i}$$
$$b_n = g_n$$

Reflection of Gray codes

ε	0	0 0	0 00	0 000
	1	0 1	0 01	0 001
		1 1	0 11	0 011
		1 0	0 10	0 010
			1 10	0 110
			1 11	0 111
			1 01	0 101
			1 00	0 100
				1 100
				1 101
				1 111
				1 010
				1 011
				1 001
				1 101
				1 000

Gray Code

- As we count up/down using binary codes, the number of bits that change from one binary value to the next varies

  000 → 001 (1-bit change)
  001 → 010 (2-bit change)
  011 → 100 (3-bit change)

- Gray code: only 1 bit changes as we count up or down
- Binary reflected code
- Gray code can be used in low-power logic circuits that count up or down, because only 1 bit changes per count

Digit	Binary	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Binary –to-Gray Code Conversion

Step1. The left most code digit is the same as the left most binary code

1	0	1	1	0	Binary
↓					
1					Gray

Step2. Add the left most binary code bit to the adjacent one.

1	+	0	1	1	0	Binary
		↓				
1		1				Gray

Step3. Add the next adjacent pair..

1	0	+	1	1	0	Binary
			↓			
1	1		1			Gray

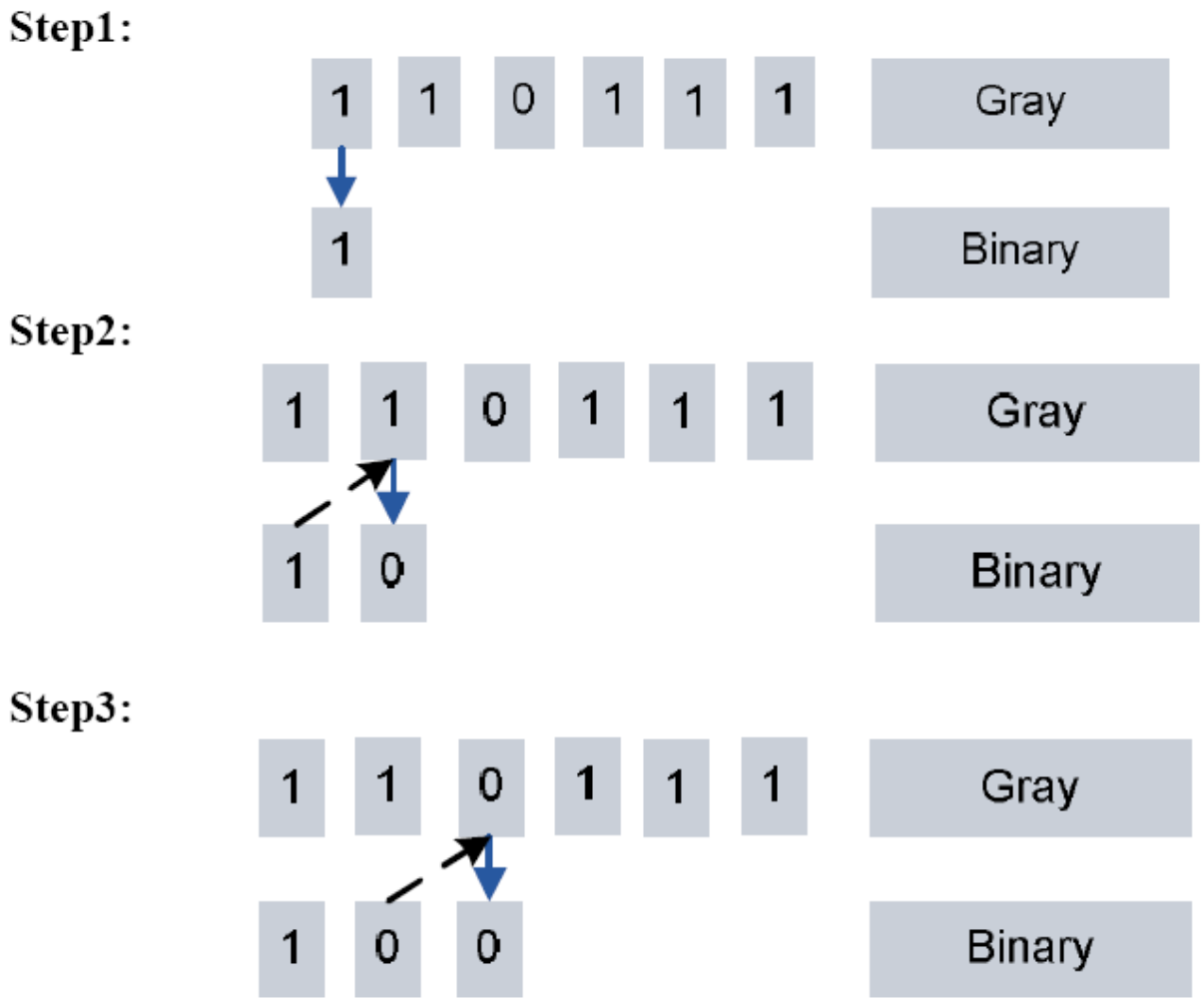
Step4. Add the next adjacent pair.

1	0	1	+	1	0	Binary
				↓		
1	1	1		0		Gray

Step5. Add the next adjacent pair.

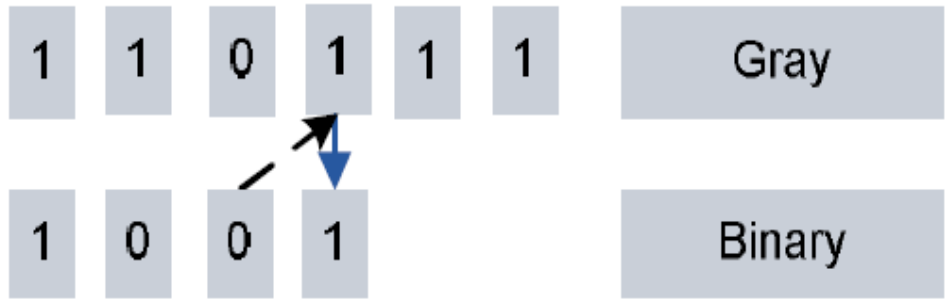
1	0	1	1	+	0	Binary
					↓	
1	1	1			1	Gray

Gray –to-Binary Conversion

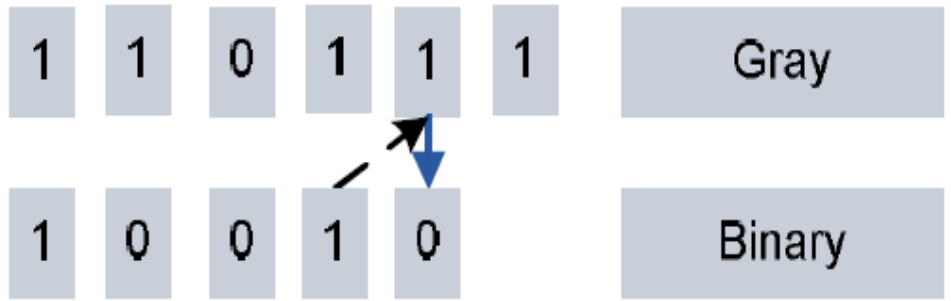


Gray –to-Binary Conversion

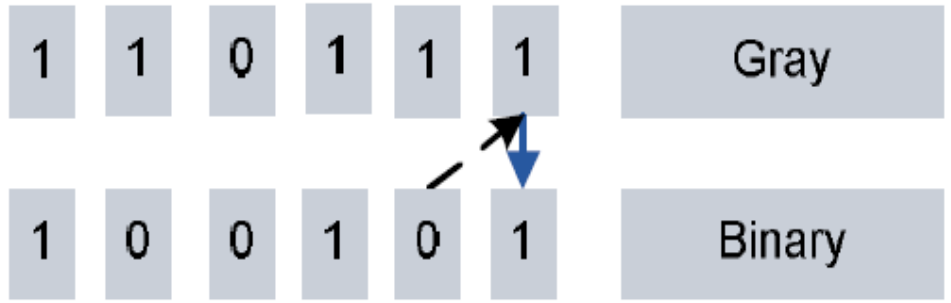
Step4:



Step5:

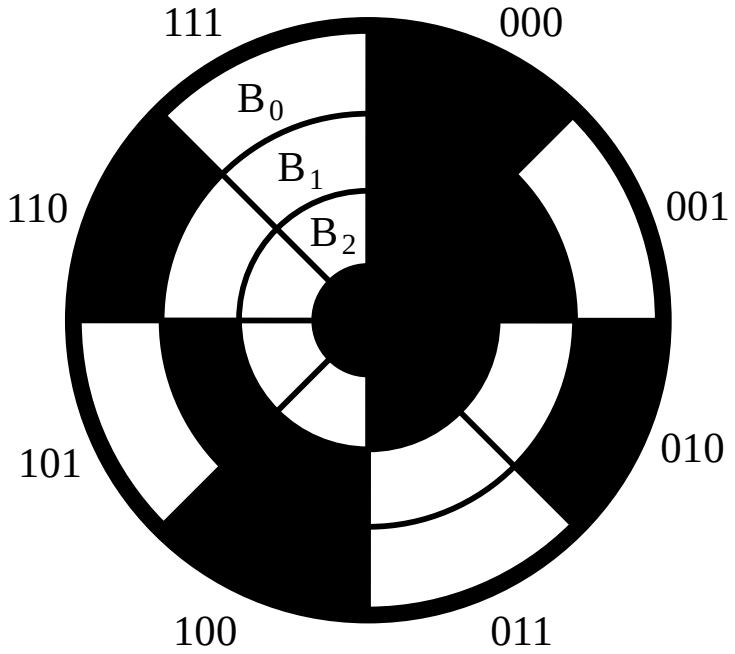


Step6:

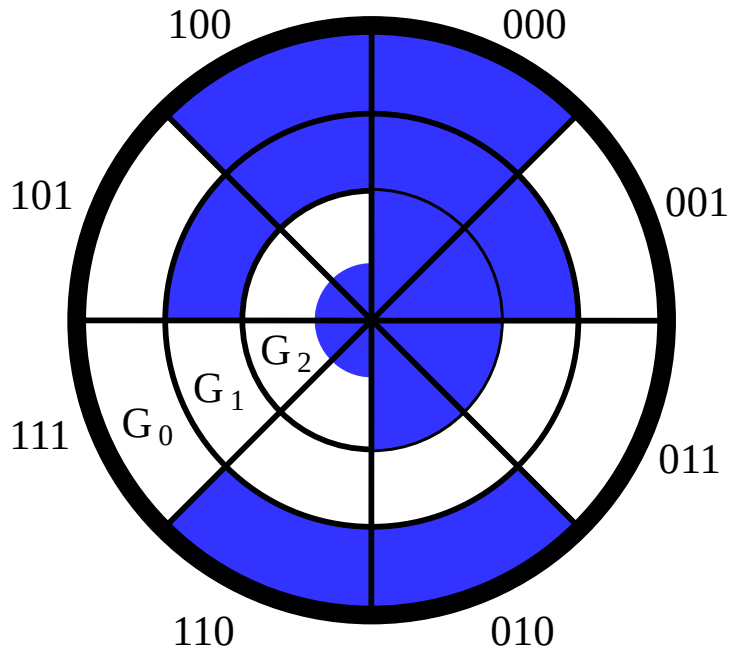


Optical Shaft Encoder

- Does this special Gray code property have any value?
- An Example: Optical Shaft Encoder



(a) Binary Code for Positions 0 through 7



(b) Gray Code for Positions 0 through 7

Shaft Encoder (Continued)

- How does the shaft encoder work?
- For the binary code, what codes may be produced if the shaft position lies between codes for 3 and 4 (011 and 100)?
- Is this a problem?
  - For the Gray code, what codes may be produced if the shaft position lies between codes for 3 and 4 (010 and 110)?
  - Is this a problem?
  - Does the Gray code function correctly for these borderline shaft positions for all cases encountered in octal counting?

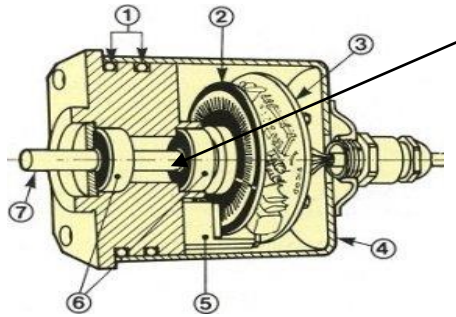
Gray Code:
Advantages for Optical Encoding of Angular Shaft Position

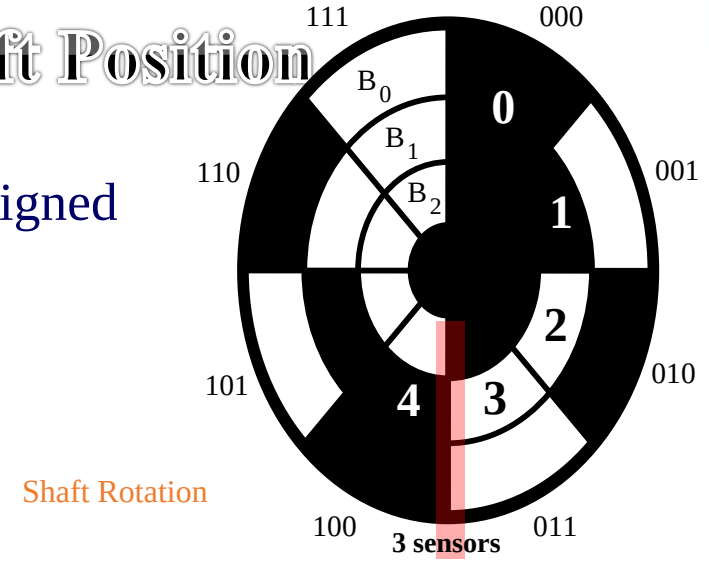
- The 3 optical sources/sensors (or sector marks) may not be perfectly aligned
- Some may change before the others at sector crossings
- For example, at the crossing between positions 3 and 4:

Gray: Only bit G0 changes → code can change only from 111 to 110

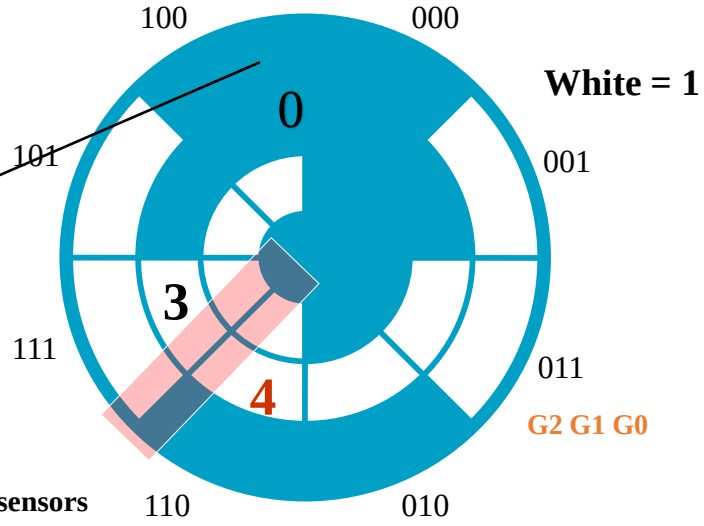
(i.e. correct new code is the only possibility)

Binary: All 3 bits change → several wrong intermediate codes can be generated (e.g. 110). If read, they give wrong information on shaft position





(a) Binary Code for Positions 0 through 7



(b) Gray Code for Positions 0 through 7

Excess-3 BCD Code

- The Excess-3 (XS-3) BCD code does not use the principle of positional weights into consideration while converting the decimal numbers to 4-bit BCD system. Therefore, we can say that this code is a non-weighted BCD code.
- The function of XS-3 code is to transform the decimal numbers into their corresponding 4-bit BCD code.
- In this code, the decimal number is transformed to the 4-bit BCD code by first adding 3 to all the digits of the number and then converting the excess digits, so obtained, into their corresponding 8421 BCD code. Therefore, we can say that the XS-3 code is strongly related with 8421 BCD code in its functioning.

Decimal digits	Excess-3 BCD code
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Excess-3 BCD Code

- Examples 6.7-6.9, p102.
- 6.7: Convert the decimal number 85 to XS-3 BCD code.

Add 3 to each digit of the given decimal number as:

8+3=11

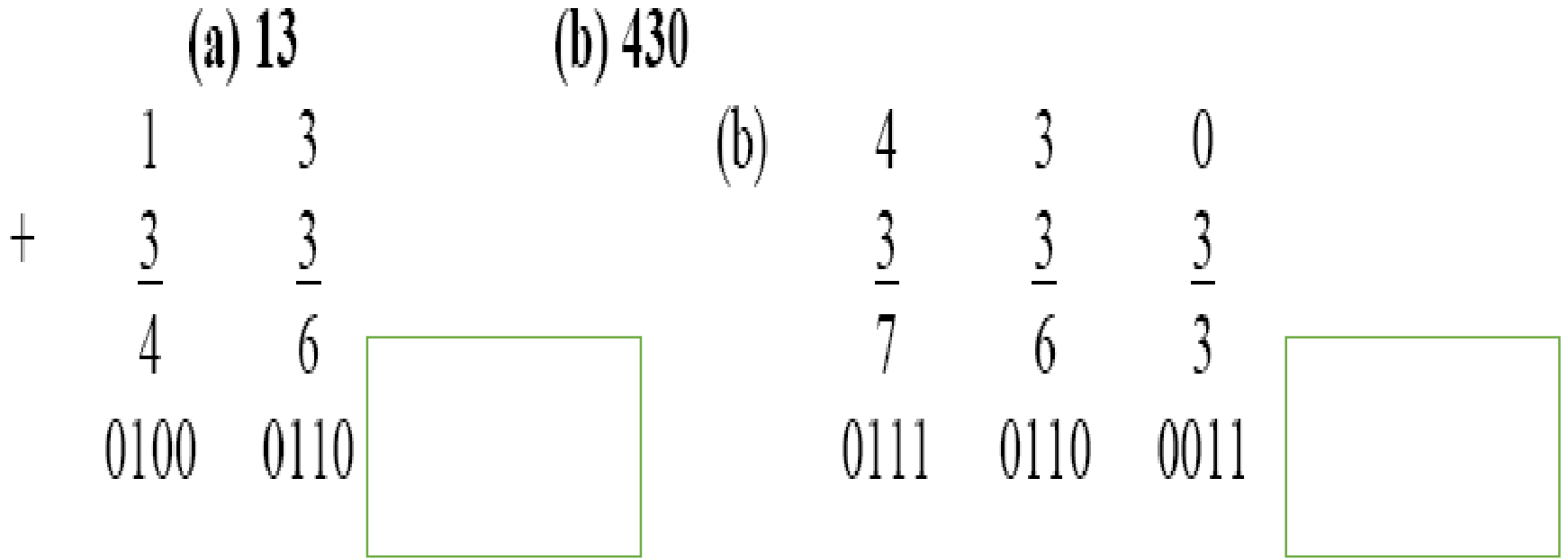
5+3=8

The corresponding 4-bit 8421 BCD representation of the decimal digit 11 is 1011.

The corresponding 4-bit 8421 BCD representation of the decimal digit 8 is 1000.

Therefore, the XS-3 BCD representation of the decimal number 85 is 1011 1000.

The Excess-3- Code



Parity

- The method of parity is widely used as a method of error detection.
  - Extar bit known as parity is added to data word
  - The new data word is then transmitted.
- Two systems are used:
  - Even parity: the number of 1's must be even.
  - Odd parity: the number of 1's must be odd.
    - Example:

	Even Parity	Odd parity
11001	110011	110010
11110	111100	111101
11000	110000	110001

Parity Bit Error-Detection Codes

- Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.

- A simple form of redundancy is parity, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors.

- A code word has even parity if the number of 1's in the code word is even.
- A code word has odd parity if the number of 1's in the code word is odd.

4-Bit Parity Code Example

- Fill in the even and odd parity bits:

Even Parity		Odd Parity	
Message	Parity	Message	Parity
000	_	000	_
001	_	001	_
010	_	010	_
011	_	011	_
100	_	100	_
101	_	101	_
110	_	110	_
111	_	111	_

- The codeword "1111" has even parity and the codeword "1110" has odd parity.
- Both can be used to represent 3-bit data.

DECIMAL CODES – Binary Codes for Decimal Digits

There are over 8,000 ways that you can chose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

Decimal	8,4,2,1	Excess3	8,4,-2,-1	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0100
2	0010	0101	0110	0101
3	0011	0110	0101	0111
4	0100	0111	0100	0110
5	0101	1000	1011	0010
6	0110	1001	1010	0011
7	0111	1010	1001	0001
8	1000	1011	1000	1001
9	1001	1100	1111	1000

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

A code called ASCII

- ASCII stands for American Standard Code for Information Interchange
- The code uses 7 bits to encode 128 unique characters
- Reference the textbook, pg. 27, for a table of the ASCII code
- As a note, formally, work to create this code began in 1960. 1st standard in 1963. Last updated in 1986.

Table 1.7
American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$ \ $b_7b_6b_5$	000	001	010	011	100	101	110	111
0000	NUL	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

ASCII Code

- Represents the numbers
  - All start 011 xxxx and the xxxx is the BCD for the digit
- Represent the characters of the alphabet
  - Start with either 100, 101, 110, or 111
  - A few special characters are in this area
- Start with 010 – space and !”#$%&’()*+.-,/
- Start with 000 or 001 – control char like ESC

ASCII Example

- Encoding of 123
  - 011 0001 011 0010 011 0011
- Encoding of Joanne
  - 100 1010 110 1111 110 0001
  - 110 1110 110 1110 110 0101

- Note that these are 7 bit codes

ALPHANUMERIC CODES - ASCII Character Codes

- American Standard Code for Information Interchange
- This code is a popular code used to represent information sent as character-based data. It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters
- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return)
- Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).

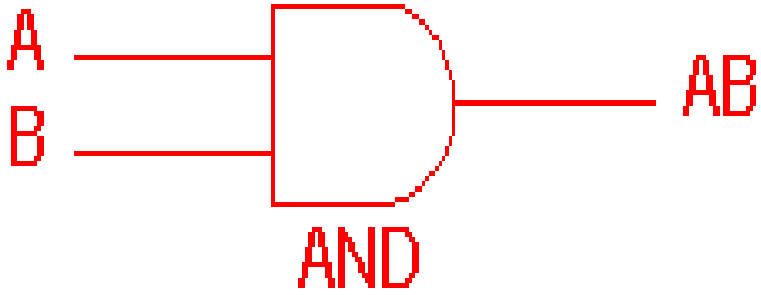
UNICODE

- UNICODE extends ASCII to 65,536 universal characters codes
  - For encoding characters in world languages
  - Available in many modern applications
  - 2 byte (16-bit) code words

Jiangxi University of Science and Technology

Logic Gates

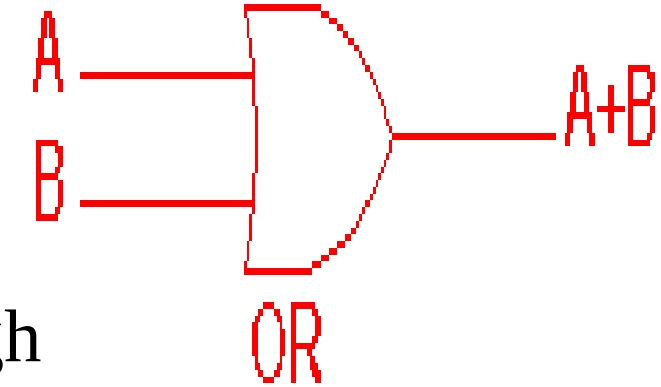
AND gate



The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high.
A dot (.) is used to show the AND operation i.e. A.B.
Bear in mind that this dot is sometimes omitted i.e. AB

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

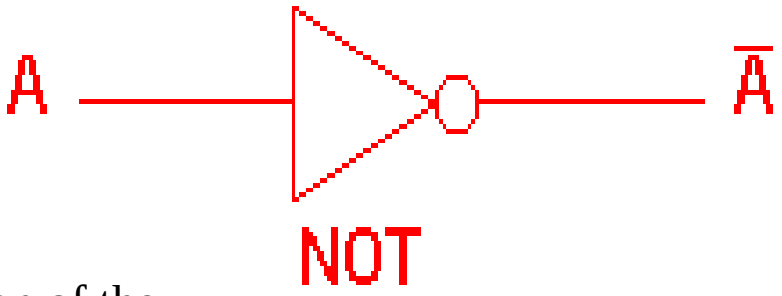
OR gate



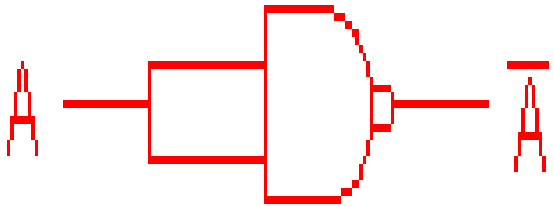
The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high.
A plus (+) is used to show the OR operation.

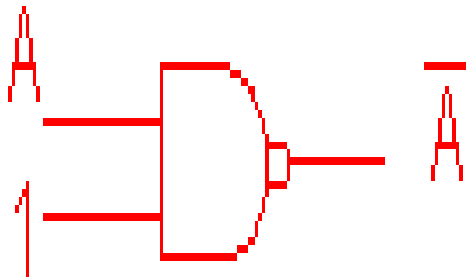
2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate



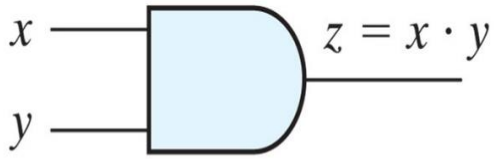
- The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter.
- If the input variable is A, the inverted output is known as NOT A.
- This is also shown as A', or A with a bar over the top, as shown at the outputs.
- The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate.
- It can also be done using NOR logic gates in the same way.



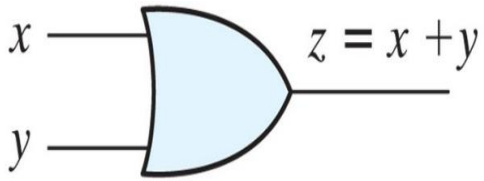


NOT gate	
A	$\bar{A}$
0	1
1	0

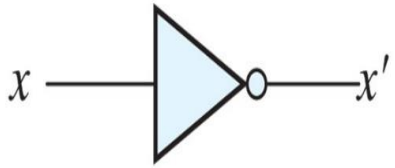
Truth Tables of Logical Operations



(a) Two-input AND gate



(b) Two-input OR gate

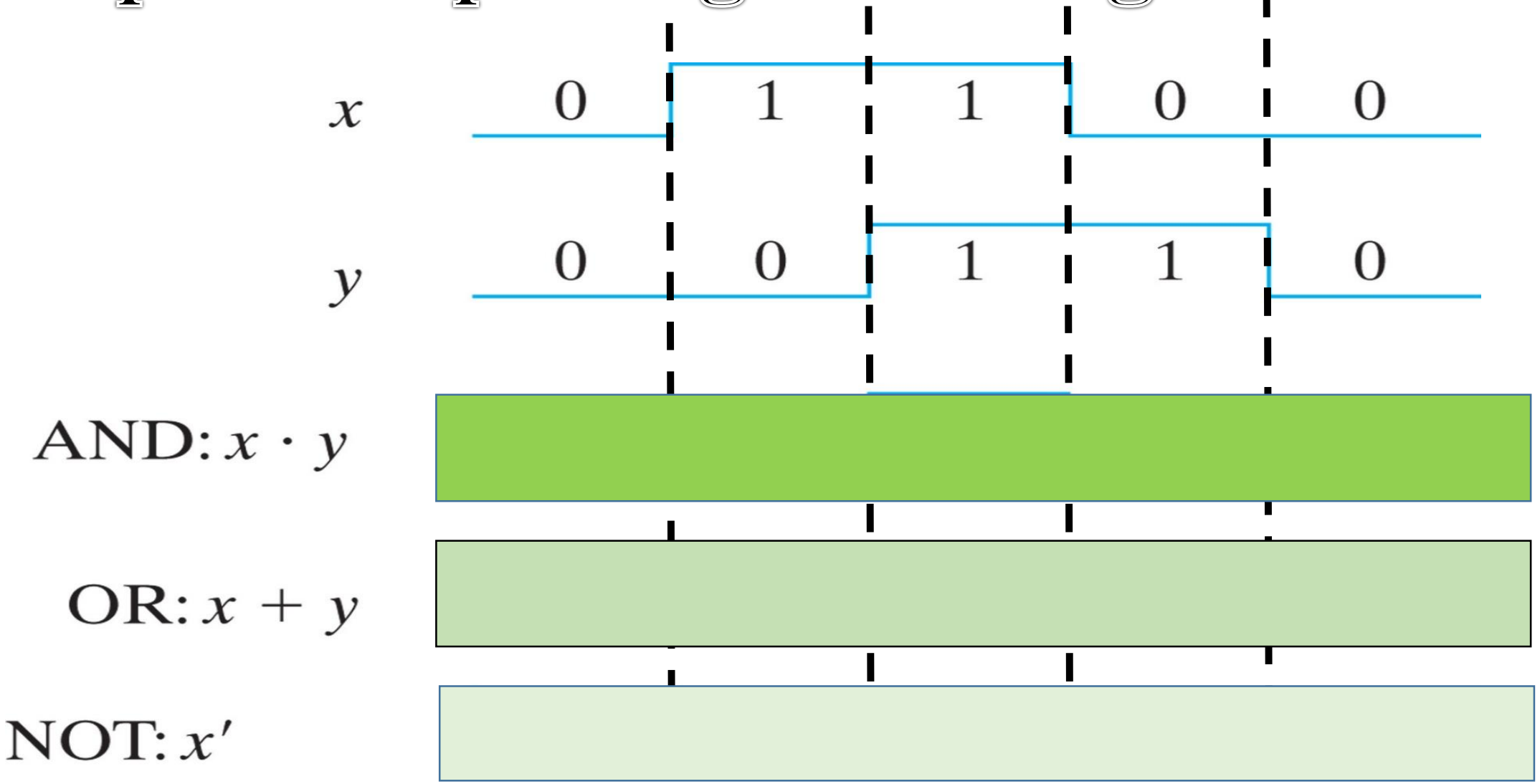


(c) NOT gate or inverter

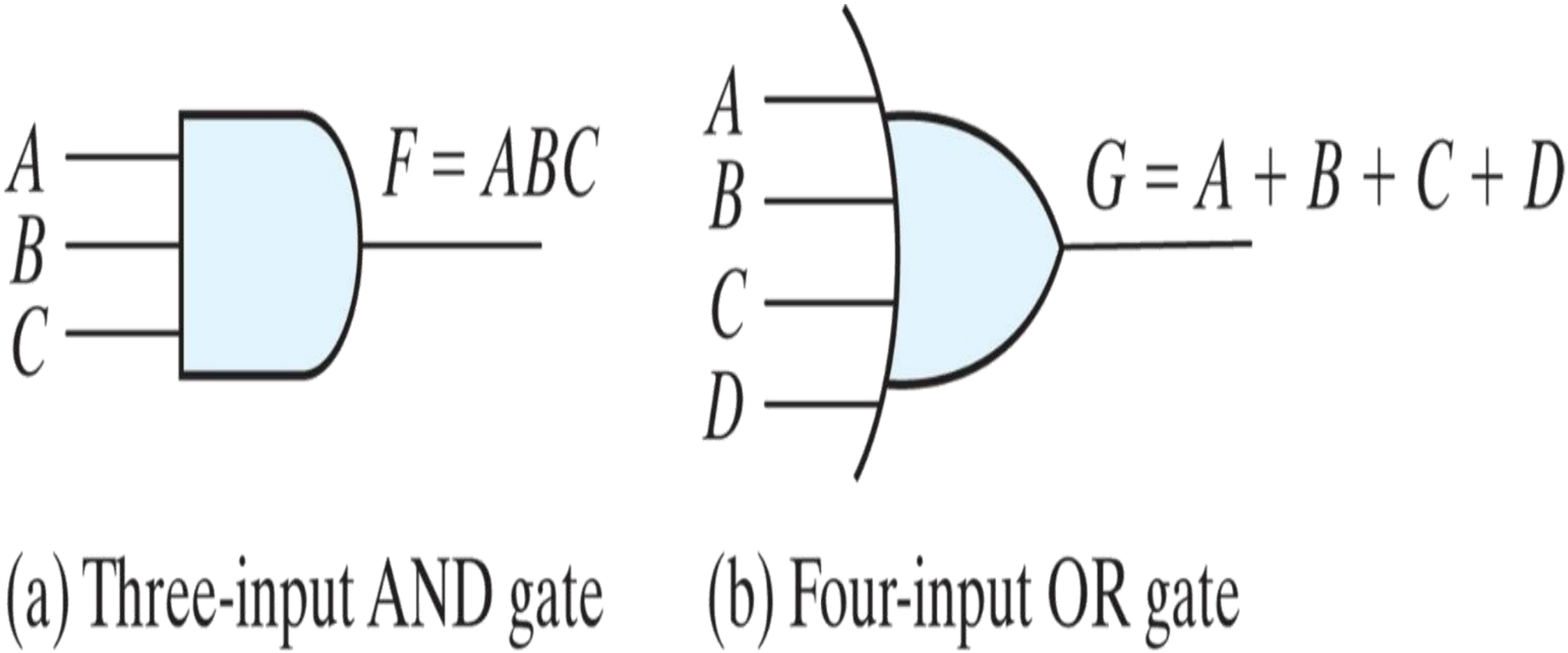
Table 1.8
Truth Tables of Logical Operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Input–output signals for gates

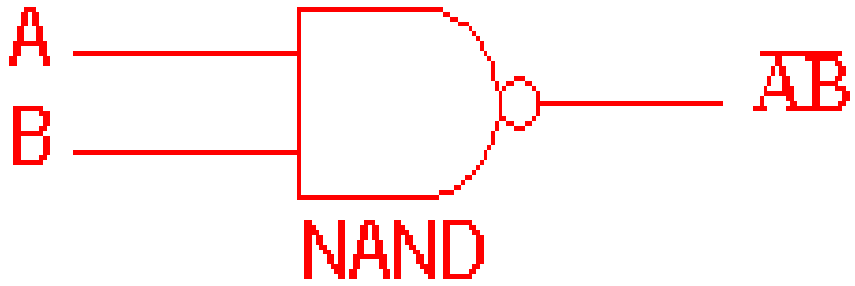


Gates with multiple inputs



(a) Three-input AND gate (b) Four-input OR gate

NAND gate

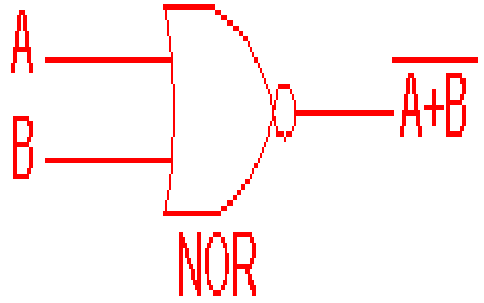


- This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate.
- The outputs of all NAND gates are high if any of the inputs are low.
- The symbol is an AND gate with a small circle on the output.
- The small circle represents inversion.

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

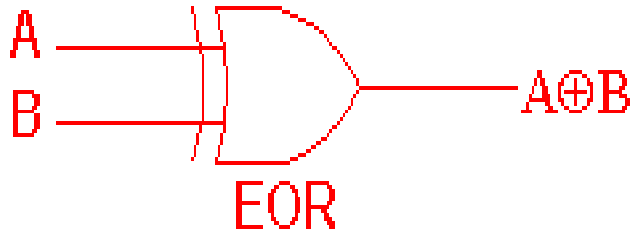
NOR gate

- This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high.
- The symbol is an OR gate with a small circle on the output. The small circle represents inversion.



2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

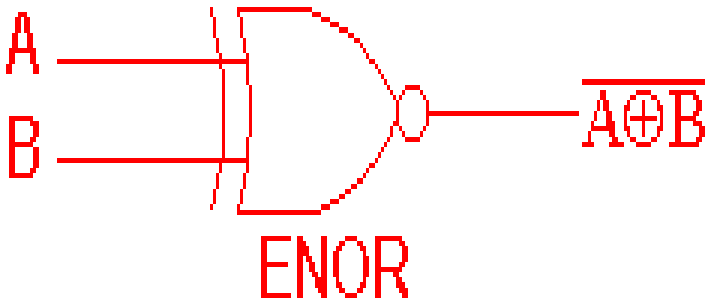
EXOR gate



- The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high.
- An encircled plus sign (⊕) is used to show the EOR operation.
-

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR gate



- The '**Exclusive-NOR**' gate circuit does the opposite to the EOR gate. It will give a low output if **either, but not both**, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Note:

The NAND and NOR gates are called *universal functions* since with either one the AND and OR functions and NOT can be generated.

Note:

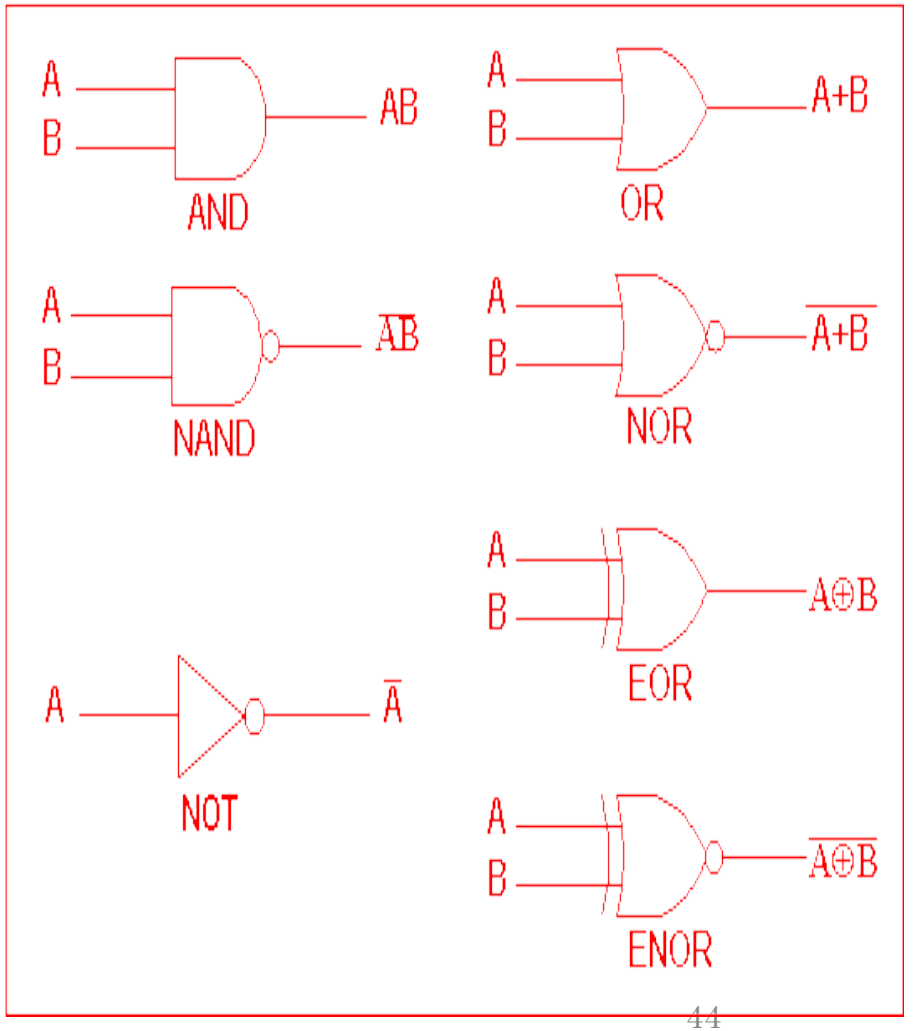
A function in *sum of products* form can be implemented using NAND gates by replacing all AND and OR gates by NAND gates.

A function in *product of sums* form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

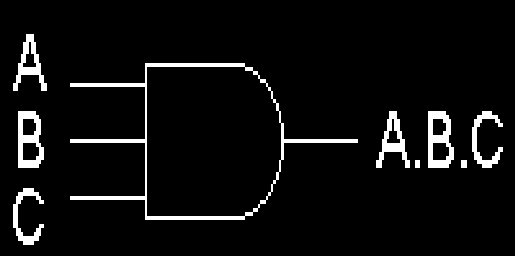
ALL Logic gates

NOT gate	
A	$\bar{A}$
0	1
1	0

INPUTS		OUTPUTS					
A	B	AND	NAND	OR	NOR	EXOR	EXNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1



- Here is an example of a three input AND gate. Notice that the truth table for the three input gate is similar to the truth table for the two input gate. It works on the same principle, this time all three inputs need to be high (1) to get a high output

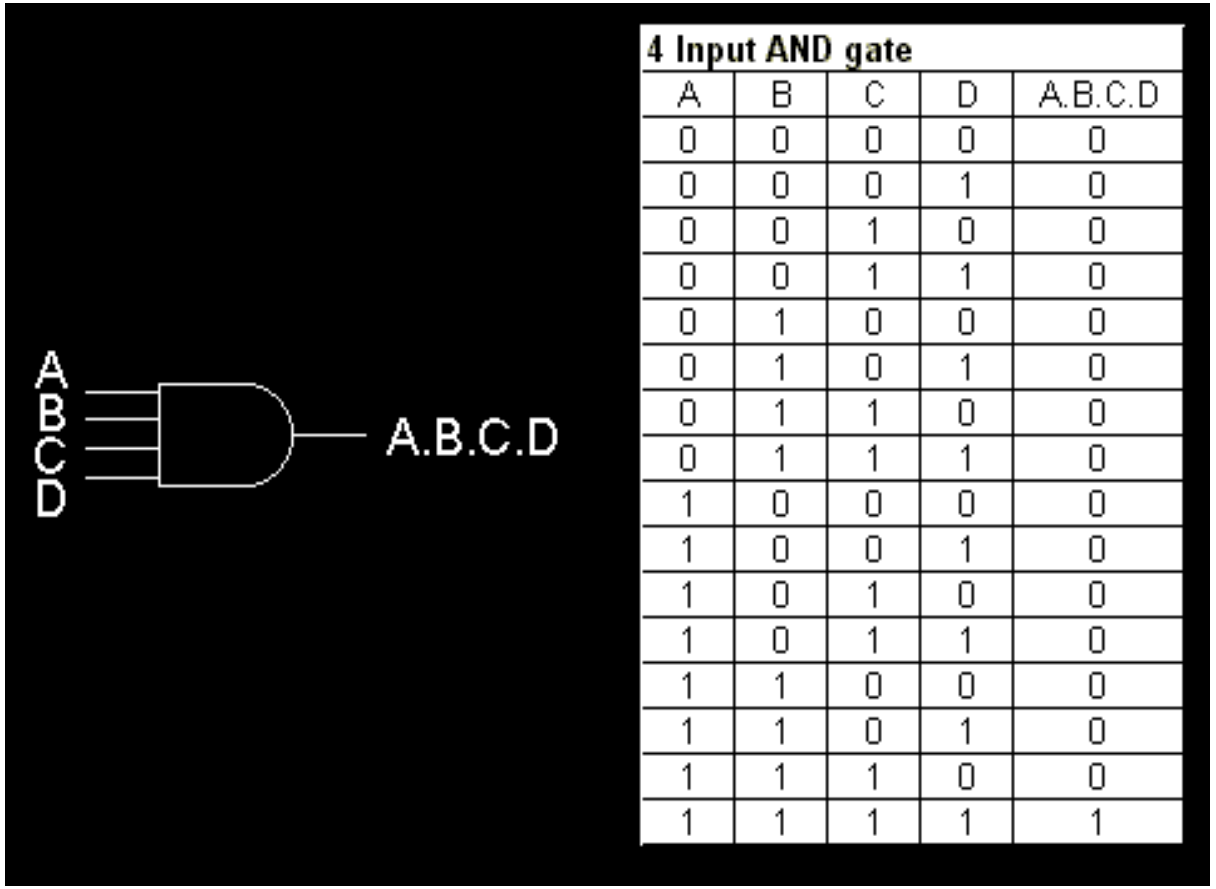


3 Input AND gate

A	B	C	A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Four Input AND Gate

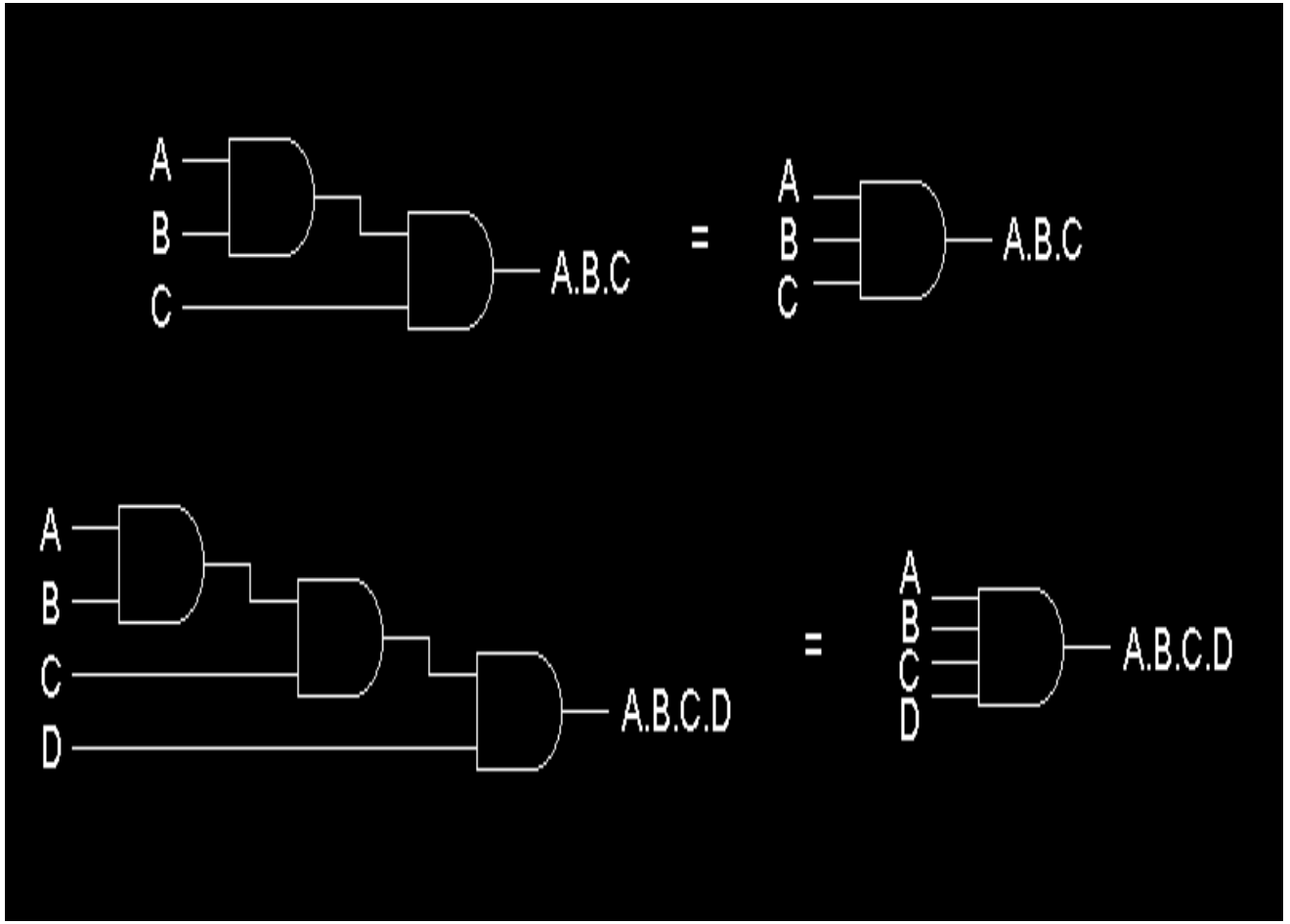
- Here is an example of a four input AND gate. It also works on the same principle, all four inputs need to be high (1) to get a high output. The same principles apply to 5, 6,..., n input gates.



4 Input AND gate				
A	B	C	D	A.B.C.D
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Making Multi Input Gates

- Multi input gates can be made by joining gates of the same type with less inputs.
- The diagrams below shows how a three input AND gate and and a four input AND gate can be made out of two input AND gates.



Reference

http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/