- Re-draw the same Circuit
  - Special Symbol
- Circuit Connection Diagram

4. Digital Arithmatic

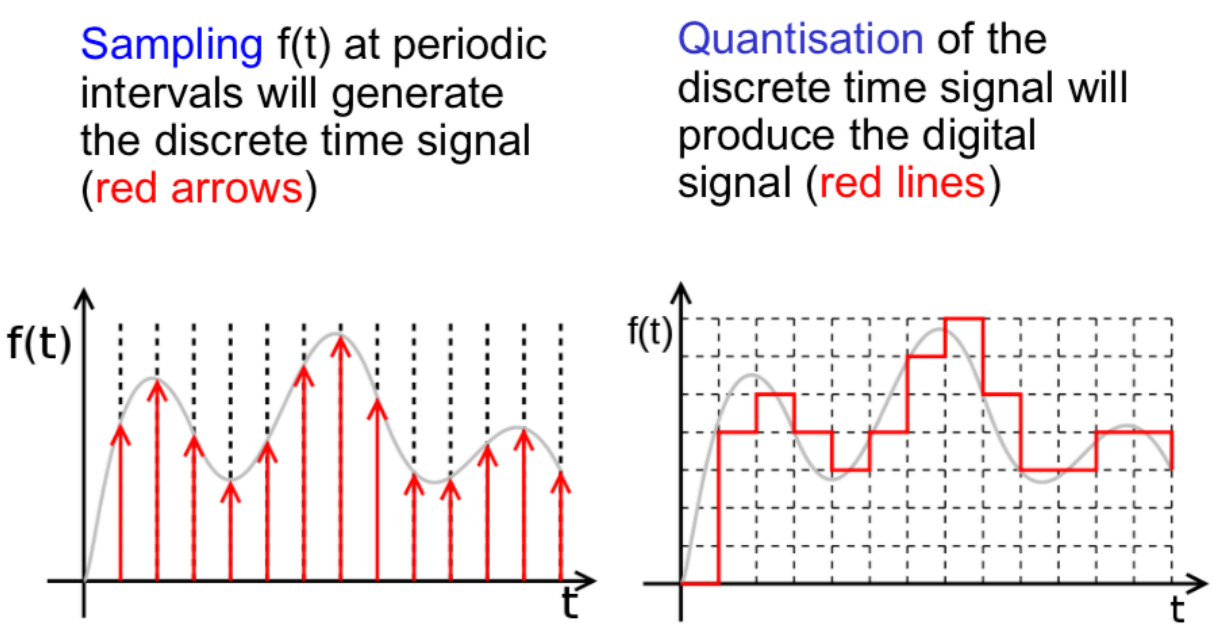
- Simple Digital Arithmetic
  - Unsigned Addition
  - Substraction
    - 16's complement
  - Multiplication
    - Unsigned
    - sign extension (very important)
    - Overflow
      - Overflow detection
  - Division
- Adder
  - Half Adder(HA): addition of 2 bits only
  - Full Adder(FA): addition of 3 bits
    - Circuit Diagram
      - Implementation with HA
      - Propogation Delay
  - Parallel Adder(Ripple Adder)
- Signed Number
  - Signed-Magnitude system
  - 2's Complement System/Representation
    - Observation
    - 2's complement Conversion
      - From 2's complement to Decimal
      - short-cut
      - Exceeding values
      - Understanding: get the decimal value quickly
      - Exception
    - Benefits
    - Sign extension
- Operations in 2's complement system
  - Addition
  - Subtraction
  - Multiplication
- Circuit Implementation
  - Parallel FA with Registers
    - Timing Diagrams
  - BCD Addition

1. Introduction

Analog Versus Digital

- ***Analog***: changes in a continuous manner
- ***Digital***: countable, changes in discrete steps

Analog quantities can be represented in digital format by using **sampling** and **quantisation**(**definition** A range of analog values is lumped together and assigned a representative digital value (many-to-one mapping, with precision lost)



Application: digital camere, because light intensity is analog

Adavantages: easier design, easier information storage, greater accuracy and precision(compared to analog signal), programmability, less susceptible to circuit noise, VLSI techonology

Limitations: needed mutual convertion between digital and analog forms—ADC and DAC

Digital Number Systems

Number system decimal(10), binary(2), octal(8), hexadecimal(16)

hexadecimal is only for human-reading, everything is in binary format

Digital circuits prefer **binary system** to store information, via either **electrical voltage** or **current**

bit(binary digit) has the value 0 or 1

nibble : 4 bits

byte: 8 bits

Electronic aspects and software aspects of digital design

Application:

- CAD(Compute-Aided Design): general name
- schematic entry: for diagrams
- HDL(Hardware Description Language): e.g. Verilog [1]
- Synthesizer
- Simulator
- Test bench [2]

Integrated Logic Circuits

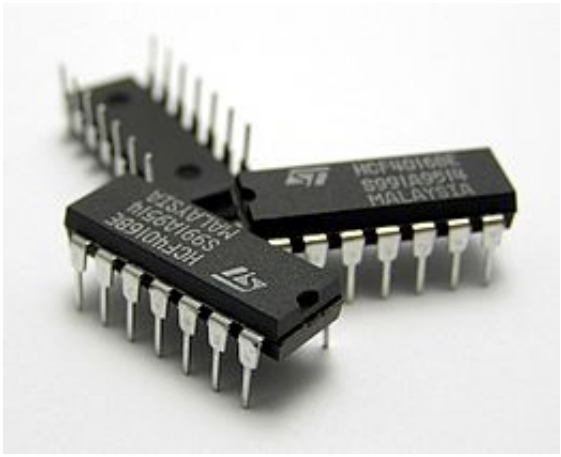
definition usually as Integrated Circuits using various semiconductor technologies

logic circuit descriptions can be in the forms of truth table, logic expression or circuit diagram.

Logic Devices

use standard logic integrated circuits (ICs), application-specific ICs (ASICs), programmable logic devices

Dual-in line

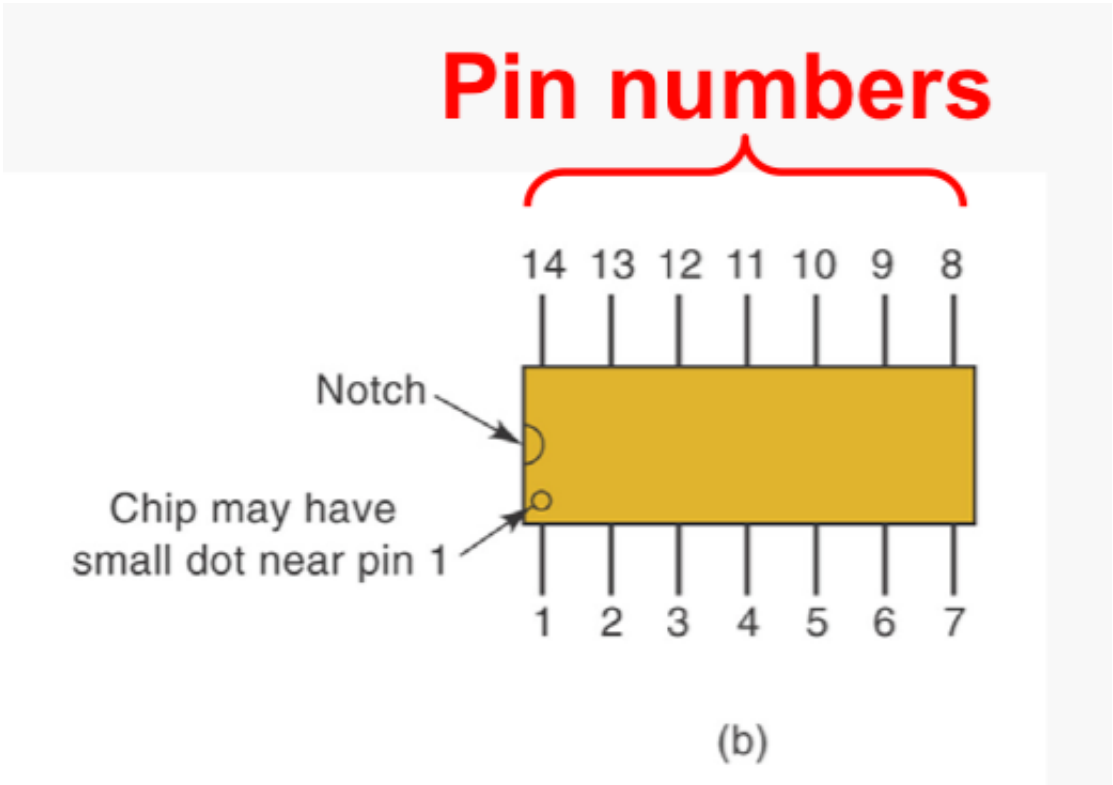


Connection: unidentical metal pins

Volt: 5V commonly

There are also other common IC packaging, like 24-pin to even 96-pin

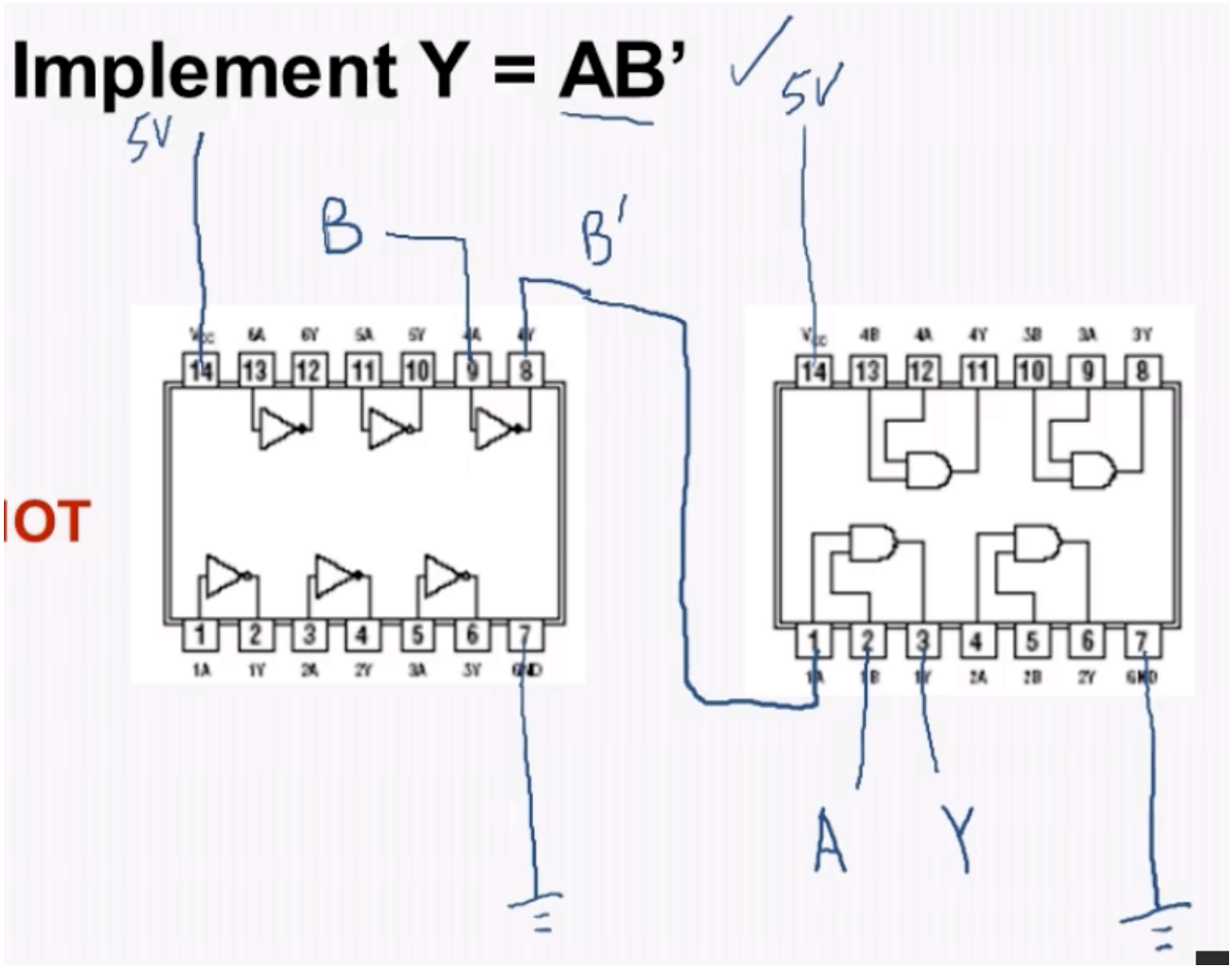
Read pin numbers



seems like counter-clockwise

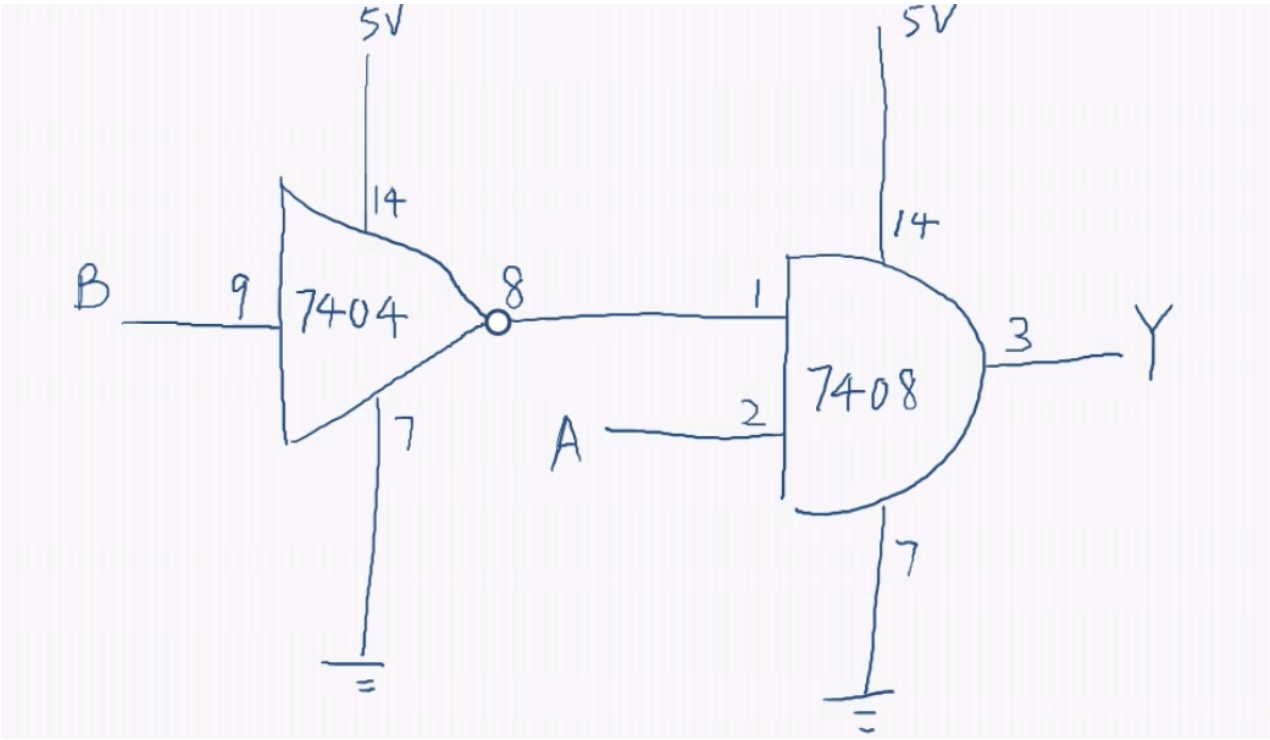
Conncetion

`hex` 6, `quad` 4: e.g. 7408 Quad-AND (Quad: number of components within the unit)



1. pin numbers(every connection)
2. **volt and ground** Each device must have a pair of Vcc and Ground to provide electrical power supply to it.(not all components have the same pin)
3. index of the component

Example



Programmable Logic Devices

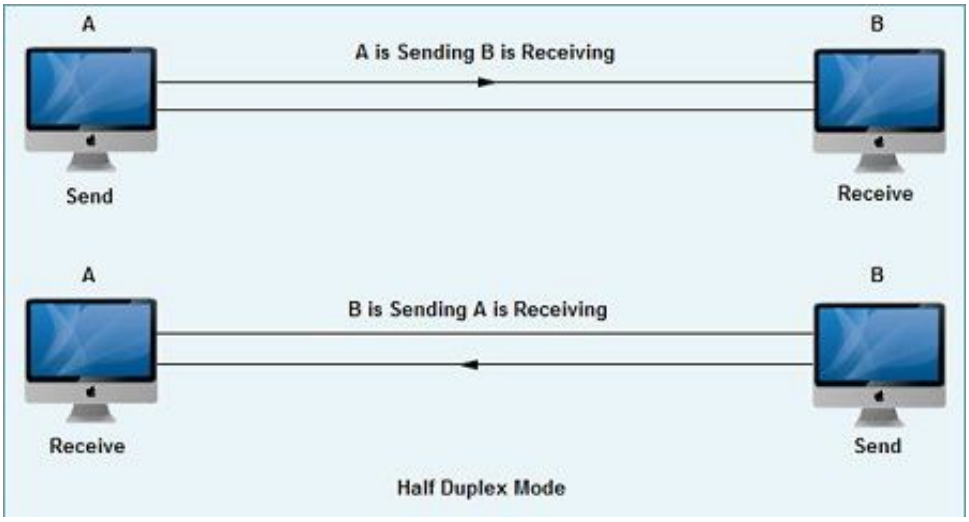
programmable : IC's logic function can be changed easily, without physically replacing or rewiring the device

e.g. FPGA(Fied-Programmble Gate Array)[3]

Digital Data transmission

Data Transmission methods: serial or parallel

- Trade-off: simplicity/cost and speed
- serial: TV, radios, for broadcasting only, serial drives like DVD and hard disk
- ***Half-duplex data transmission*** : two way but one at a time (two way radios)



- ***Duplex data transmission***: two way concurrently
- Parallel: Parrallel-ATA ports and PCI slots in PCs (can be de-synchronised)

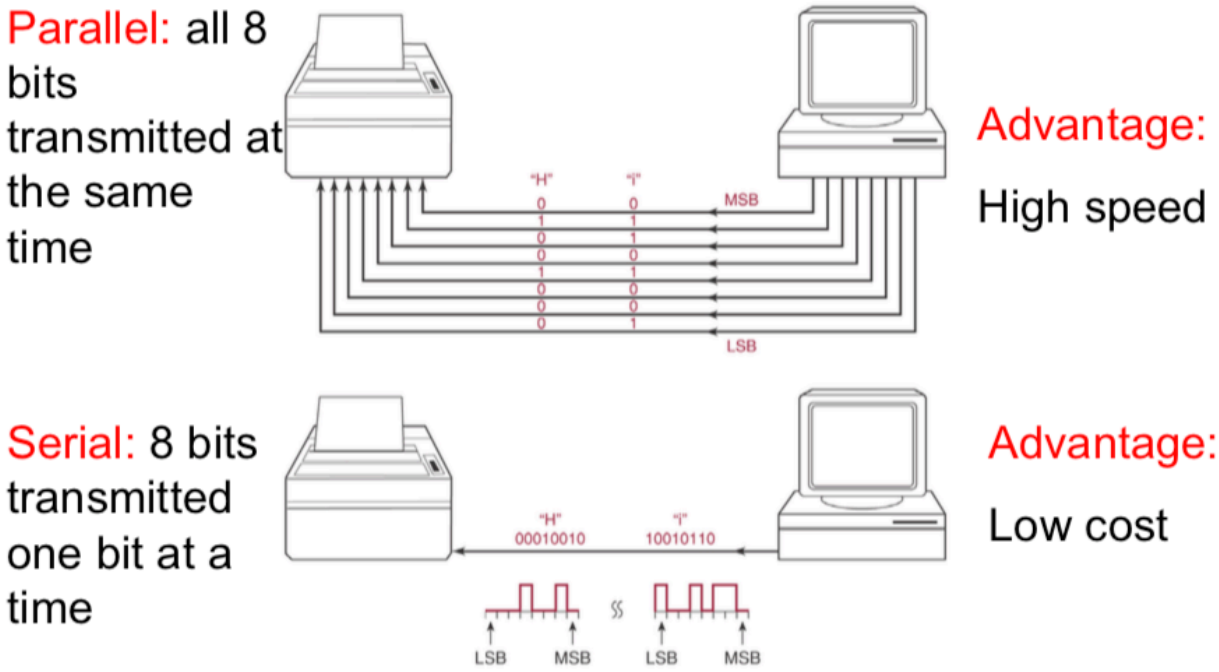


Figure 1.10 (Tocci 10th Ed) Parallel and Serial Transfer

2a. Number Systems

The subscript 10 or 2 shows the ***base or radix***

- $\text{number of base} = \text{number of symbols}$
- the lower the base, the larger number of digits is required to represent a given value

$$1011_2 = 11_{10} = 13_8 = B_{16}$$

4-bit string ranges 0~15, 3-bit string ranges 0~8

`0x` prefix signifies a hex number

`0o` octal

`0b` binary

`NULL` decimal

only decimal numbers can be pronuounced as hundred, thousand, etc.

Position-value system

description each digit carries a **weight**

- LSD(least significant digit): least weight
- MSD(most significant digit): most weight

$$\text{The weight(expressed in decimal)carried by a base-N digit of position } p(p = 0, 1, 2, \dots) \text{ is given by } N^p$$

- corresponding weights of a base-N number: $\boxed{\mathrm{N}^3\mathrm{N}^2\mathrm{N}^1\mathrm{N}^0.\mathrm{N}^{-1}\mathrm{N}^2\mathrm{N}^3}$, note that in the integer part, **the index begins with 0**

Conversion from base-N to base-10

1. Multiply each digit of the base-N number by **its positional weight**, which **starts from 0**
2. **Sum together** the products obtained in step 1

Conversion from base-10 to base-N

1. Divide the base-10 number repeatedly by N, until a quotient of 0 is obtained
2. The first remainder is the LSD; the last remainder is the MSD of the base-N number. Others seuqntially fall between.

Conversion from hex(octal) to binary

replace each hex(octal) digit by the corresponding 4-bit(3-bit) binary equivalent, except the fraction part

Value table

Decimal	*Binary*	*Hex*	*Octal*
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	08	10
09	01001	09	11
10	01010	0A	12
11	01011	0B	13
12	01100	0C	14
13	01101	0D	15
14	01110	0E	16
15	01111	0F	17
16	10000	10	20

Conversion from binary to hex(octal)

Starting from the LSB, replace every 4 bits (3bits) by the corresponding hex(octal) digit, except for the fraction part

Pad MSB with 0's if necessary

short-cut it can be used to convert binary numbers very fast, including the fractional part

Fractional Conversion

$$
\begin{aligned}
&(374.26)_8 = (x)_2 \\
&\mathbf{2 \times 8^{-1} + 6 \times 8^{-2} = 0.34375} \\
&0.34375 \times 2 = 0.6875 \\
\boxed{\text{Answer:}} \quad &0.6875 \times 2 = 1.375 \\
&0.375 \times 2 = 0.75 \\
&0.75 \times 2 = 1.5 \\
&0.5 \times 2 = 1
\end{aligned}
$$

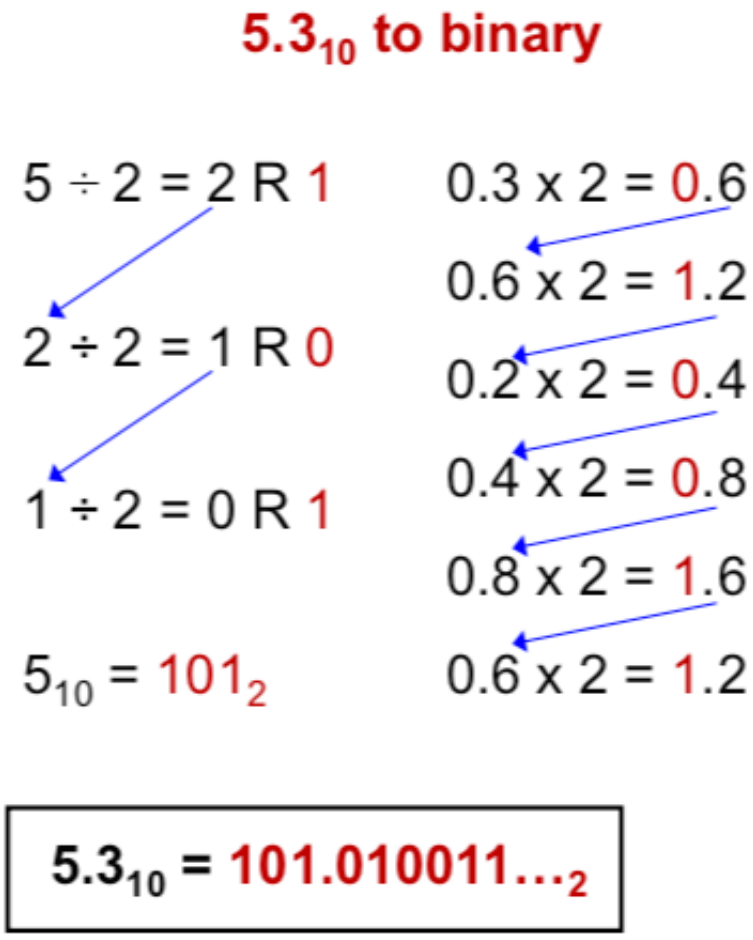
- Try to convert as much as possible

Conversion from hex to oct

- Integer part: divide them and convert to binary
- fraction part: decimal conversion

Case: fractional numbers

$$5.3 = (x)_2$$



Example

$$(13.375)_{10} = (x)_2$$

$$0.375 \times 2 = 0.75$$

$$\boxed{\text{Answer:}} \text{ fraction part: } 0.75 \times 2 = 1.5$$

$$0.5 \times 2 = 1.0$$

Explanation

$$(d_2 \times 10^2) + (d_1 \times 10^1) + (d_0 \times 10^0) = (b_m \times 2^m) + \ldots + (b_1 \times 2^1) + b_0 \times 2^0$$

After divided by 2, $b_1 \times 2^0$ can be derived as a reminder

Concerning the fraction, mutiply by 2, $b_{-1} \times 2^0$ is derived, b_{-1} is the integer

Find bits needed

use inequality, $2^N - 1 \geq \text{number}$

2b. Codes

Encoding Information

code Numbers, letters or words are represented by a special group of symbols. A group of symbols(typically 0 and 1) is called a code.

- There must be a mutual agreement

Straight binary coding

binary/ straight binary

In digital systems, numbers are probably the most common type of information that need to be represented.

straight binary coding/ simply binary coding e.g.
$\text{Note: } 35_{10} = 2\text{^}5 + 2\text{^}1 + 2\text{^}0 = (100011)_2$

Application: LED queue number, light display

BCD(binary-coded-decimal code)

usually, it is binary by default

Confounding with straight binary coding

- to encode decimal numbers, combining features of decimal and binary systems
- **Each digit of** a decimal number is represented by its **4-bit** binary

equivalent
- Large patterns are illegal e.g. 1010, 1011...

Application: BCD is used in digital machine whenever decimal information is either applied ***as inputs or displayed as outputs.***

difference between straight binary coding BCD treats decimals as strings

Example $19.25_{10} = 0001\ 1001.\ 0010\ 0101$ in BCD

Characteristic

- easier to convert (no need of repeated division)
- **Hardware** standpoint - logic circuits perform conversion, all digits can be converted **simutaneously**

BCD is not used in high speed computers:

- BCD requires more bits than binary coding, less efficient therefore
- **arithmetic processes** represented by BCD are more complicated and slower

Gray code

belong to ***minimum-change codes***, that is only 1 bit in the code changed when going from 1 step to the next (in sequence)

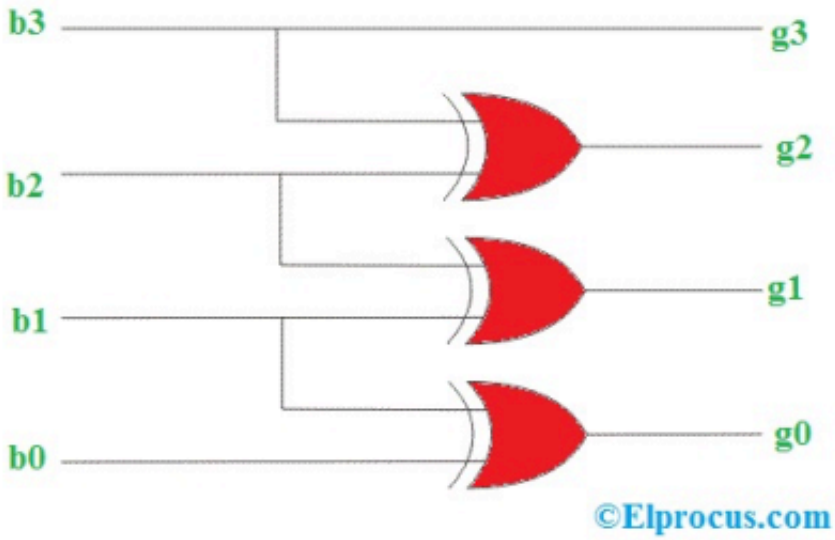
Properties/Requirements

- **unweighted** code
- Usually **cyclical**

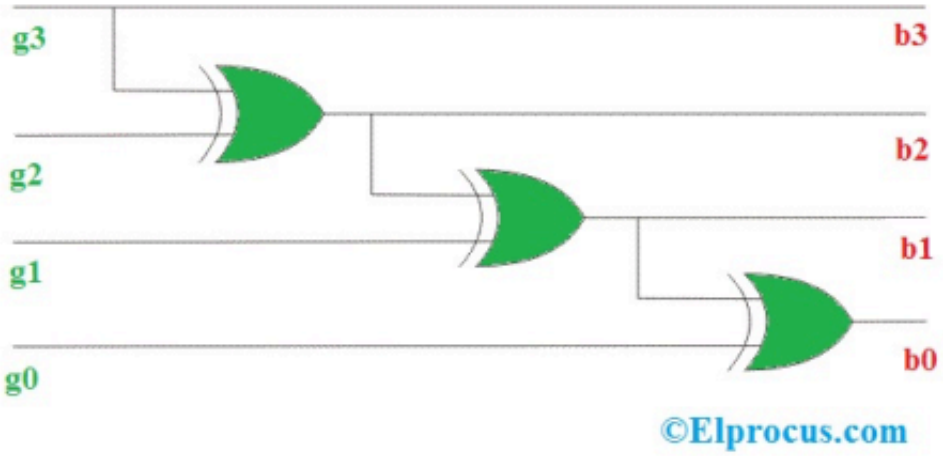
Not suitable for arithmetic operations

Conversion

Binary to Gray



Gray to Binary



Decimal	4-bit Gray code			
0	0	0	0	**0**
1	0	0	**0**	1
2	0	0	1	**1**
3	0	**0**	1	0
4	0	1	1	**0**
5	0	1	**1**	1
6	0	1	0	**1**
7	**0**	1	0	0
8	1	1	0	**0**
9	1	1	**0**	1
10	1	1	1	**1**
11	1	**1**	1	0
12	1	0	1	**0**
13	1	0	**1**	1
14	1	0	0	**1**
15	**1**	0	0	0

Application: Avoid Transition error

During the increment, multiple digits may change, but at different speeds, in different time, which leads to misleading information/error—***Transition error***

Solution Use Gray code, because only one digit changes at a time

especially useful during the manufacturous process

Alphanumeric Codes

They can represent: alphabet, punctuation marks, and special characters as well as numbers

Complete set including:

- 26 lowercase letters (a-z)
- 26 uppercase letters (A-Z)
- 10 numeric digits
- 7 punctuation marks
- 20-40 other characters such as +-/<#%, including `space`

ASCII Code

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

most widely used alphanumeric code

7-bit(128) possible code symbols (extended version: 8-bit)

Often, **hexadecimal digits** are used to represent ASCII codes

Application Transffering alphanumeric data between digital devices; digital computers used it to store alphanumeric characters

padding: with 0 in the front

Parity method for error detection

*Tansferring binary data may be **corrupted** by noise*

Multiple bit errors cannnot be detected by this simple and primitive method, because they are much less likely to happen than single bit errors, due to improved communitive channels

Limitation This method can only detect the single bit error only

Receiver and transmitter must **agree** on odd/even parity scheme

Parity Bit

Parity bit will add extra costs

definition An extra bit(0/1) attached to a code group to be transmitted

Interpretion Here are two ways:

- **Even parity** make total no. of '1' bits even BEFORE transmitting
- **Odd parity** make total no. of '1' bits odd BEFOR transmitting

  Understanding: reflect no. of '1' is odd/even

Commonly Used Prefixes

SI units	**IEC**	**JEDEC**
$k(\text{killo}) = 10^3$	$Ki(\text{kibi}) = 2^{10}$	$K(\text{kilo}) = 2^{10}$
$M(\text{mega}) = 10^6$	$Mi(\text{mebi}) = 2^{20}$	$M(\text{mega}) = 2^{20}$
$G(\text{giga}) = 10^9$	$Gi(\text{gibi}) = 2^{30}$	$G(\text{giga}) = 2^{30}$
	$Ti(\text{tebi}) = 2^{40}$	$T(\text{tera}) = 2^{40}$
	Metric System	
	$m(\text{milli}) = 10^{-3}$	
	$\mu(\text{micro}) = 10^{-6}$	
	$n(\text{nano}) = 10^{-9}$	
	$p(pico) = 10^{-12}$	

3. Logic Gates and Boolean Algebra

Logic Gates the basic building blocks of digital circuits

Boolean Algebra is used to *describe, design and simplify* **digital circuits** (differ from $\mod 2$ at discrete mathematics)

Boolean(logic) Constants (2 values only): True and false, **Logic HIGH and Logic LOW, HI and LO**, 1 and 0, **H and L**

Common Logic-level voltage ranges

TTL a low voltage (0 - 0.8 V) represents logic 0, a high voltage (2 - 5 V) represents logic 1 in TTL circuits

CMOS(74AC)

Notes

1. Be aware of the ranges to make different components work together
2. ***Indeterminate*** : there shouldn't be any voltages, because it is unable to determine H/L

A typical logic circuit

- **1 or more** logic inputs(for the inputs with the same name, they refer to the same signal)
- **1 or more** logic outputs (A logic circuit can have more outputs than inputs.)
    - Generally, outputs should't be conncected together, otherwise the components will be burnt
- outputs are related to inputs by logic functions
- they need power supply

in circuit design, the delay is an important factor

Four ways to describe the logic circuit

- Truth table (behavior)
- Timing Diagram (behaviour)
- Circuit Diagram
- Boolean Expression

Circuit Diagram

intermediate output facilitates thinking and analysis, but it depends on personal preference to be written down or not

- The *tuple means*(for timing diagram) signal change in a short moment(e.g. set as $\triangle t = 1s$)rather than occuring simutaneously

Three ways to draw the circuit diagram

1. Use standard logic symbols only
2. Show clearly how X can go low using appropriate symbols (indicating the logic flow[4]) How to express the logic flow???
    1. to achieve bubble-matching, replace the components, with mismatching bubbles, with its alternate logic gate representation
    2. when expressing the conditions, is logic simplification required? e.g. upper A=0,... lower A=1 ...
3. Show clearly how X can go high using appropriate symbols
    1. relisten(why c is better?)
    2. relisiten(new diagram example: see the photo)

Draw the timing diagram/waveform

There will be occasions where only the simplest form is considered fully correct. In general, we prefer to draw less instead of more.

Note: the same letter represents the same input

- Draw a dot to indicate the connection of wires
- Indicate the inputs and outputs clearly

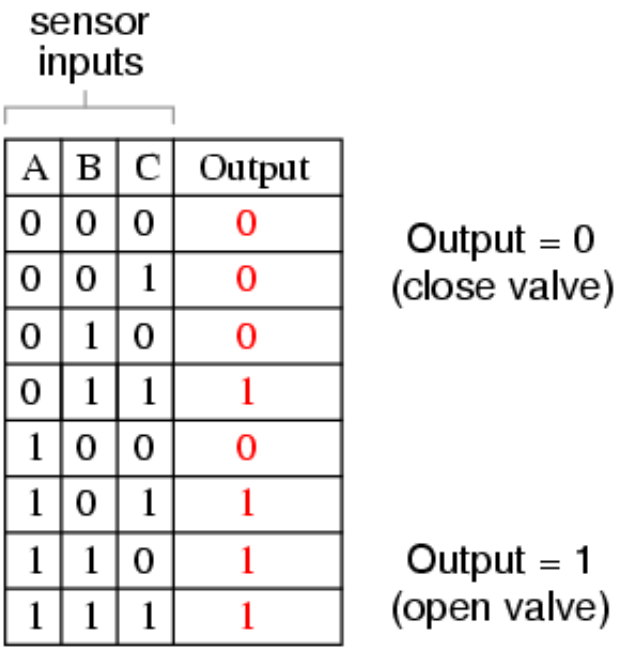
Time driagram may not express the whole truth table

Analysis of the expression may facilitate drawing the waveform, i.e. drawing intermediate $A + B$

Truth Table

*Logic function can be **fully described** by a Truth Table.* One logic circuit can only have one truth table.

- Shows the output respondant to various combinations of logic inputs
- for N inputs, there are 2^N number of input combinations(2^N rows)
- lists all possible input combinations in the **binary counting sequence**
- A logic circuit has 6 inputs and 2 outputs. There are 8 columns in its **truth table**
- Try to find out the pattern/shortcut of output by logic reasoning, saving more time to construct



Example: Find Input pattern

`What input bit pattern will be found on the 20th row of this truth table?`

Hint: Number Conversion, $\text{row number} - 1 = (x)_2$

The 1st bit on the right (i.e. the least significant bit) alternates between 0 and 1 every row.
The 2nd bit from the right alternates between 0 and 1 every 2 rows.
The 3rd bit from the right alternates between 0 and 1 every 4 rows.
The 4th bit from the right alternates between 0 and 1 every 8 rows.
e.g.

a	b	c	d	x
0	0	0	0	
			1	
		1	0	
			1	
	1	0	0	
			1	
		1	0	
			1	
1	0	0	0	
			1	
		1	0	
			1	
	1	0	0	
			1	
		1	0	
			1	

In general, the N-th bit from the right alternates between 0 and 1 every $2^{(N-1)}$ rows.

3 Basic Logic Operations and others

implemented by logic gates

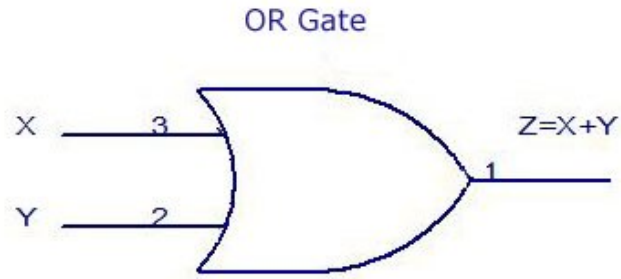
- Logic addition — or
- Logic multiplication — and
- Logic complement or inversion —not

The AND, OR and NOT gates are the basic logic gates from which any logic circuits can be built.

OR

Logic symbol

2 Input OR Gate



TRUTH TABLE

INPUTS		OUTPUT
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

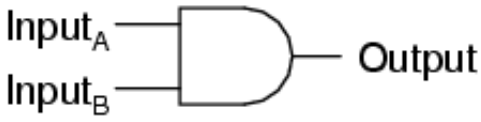
x=1 if at least one input is 1, the same as 3-input OR gate

Logic expression

$$X = A + B$$
$$X = A \text{ OR } B$$
$$X = A + B + C$$

AND

2-input AND gate



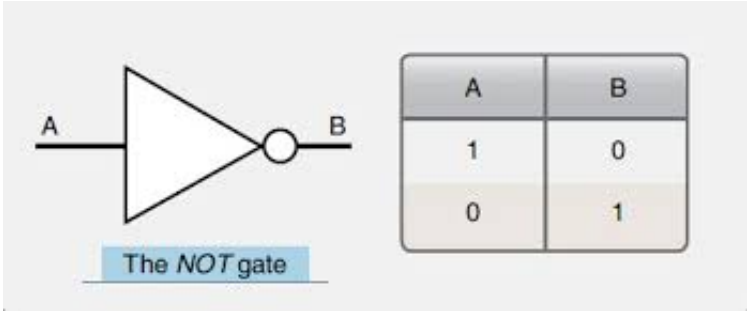
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

$$X = A \cdot B$$
$$X = A \text{ AND } B$$
$$X = AB$$
$$X = ABC$$

The result will be 1 only if all inputs are 1

NOT(inverter)

only have 1 input and it is commonly known as an inverter

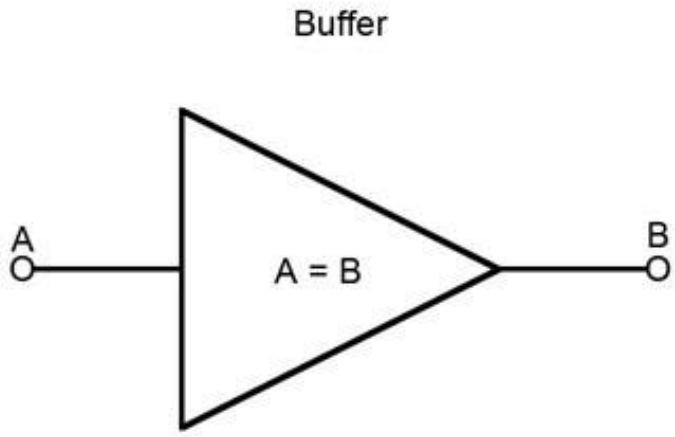


A	B
1	0
0	1

$$X = \bar{A}$$
$$X = A'$$

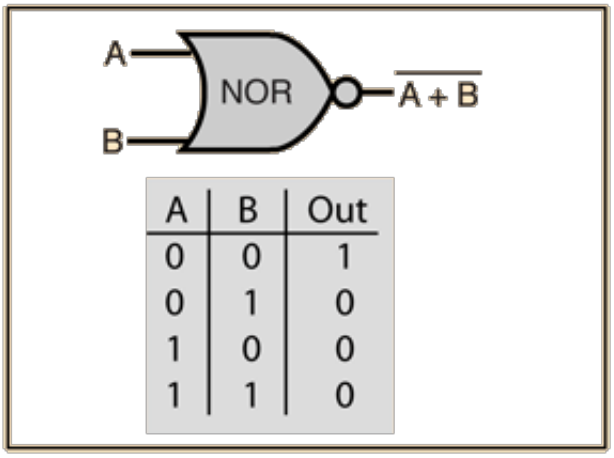
Buffer

no change in logic, but may be useful in electrical requirement



Truth Table	
A	B
0	0
1	1

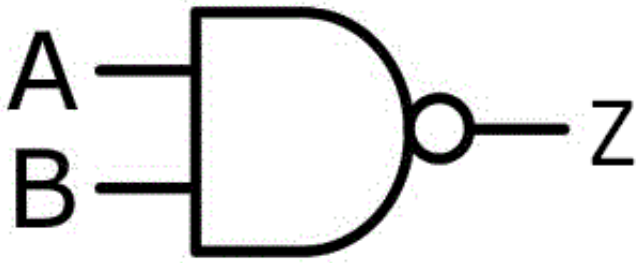
NOR(not with or)



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Ouput x=1 only when all inputs are 0; x=0 when any of the inputs is 1 (opposite to or)

NAND(not with and)



Output x=0 only when all inputs are 1(opposite to and)

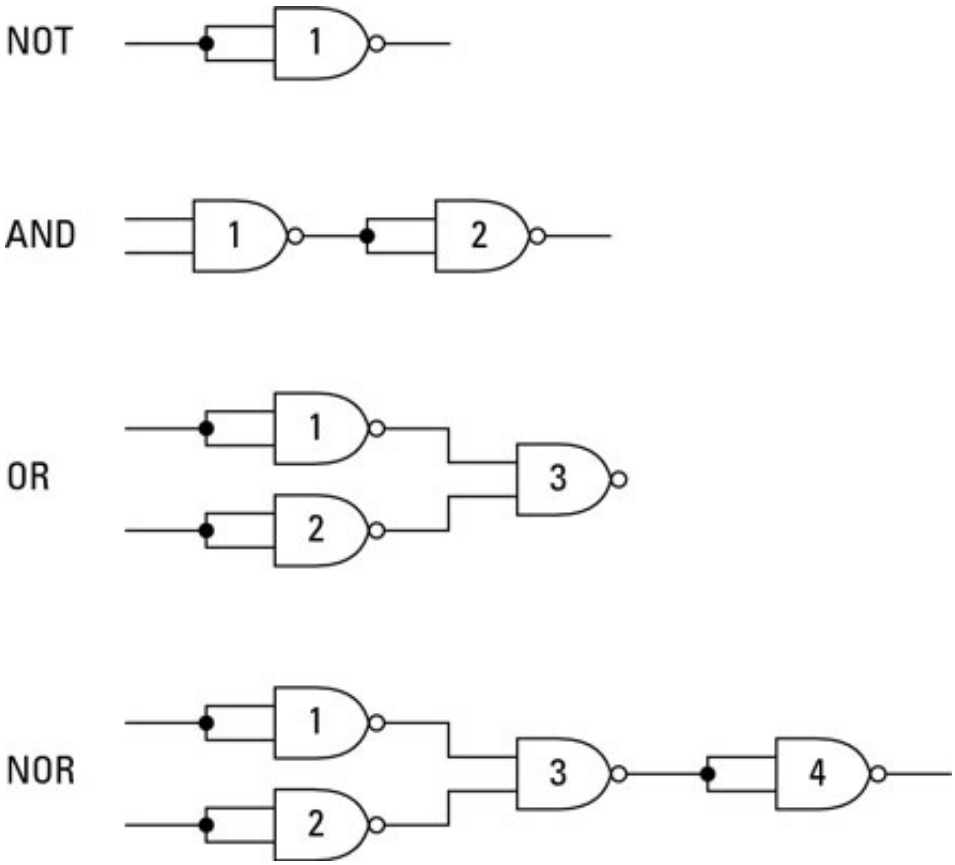
Universality of NAND and NOR[important]

- NAND/NOR gates can be used to form AND, OR and NOT gate (to reduce the manufacture fees)Some expressions are more suited for purely NAND, i.e. use fewer gates; while some others are more suited for purely NOR.
- therefore, they can implement any Boolean function
- proved by demorgan's theorem: **DeMorgan's theorems** can be used to modify a Boolean expression (simplified) into one that contains only NAND gates.
- be careful with the inverse!

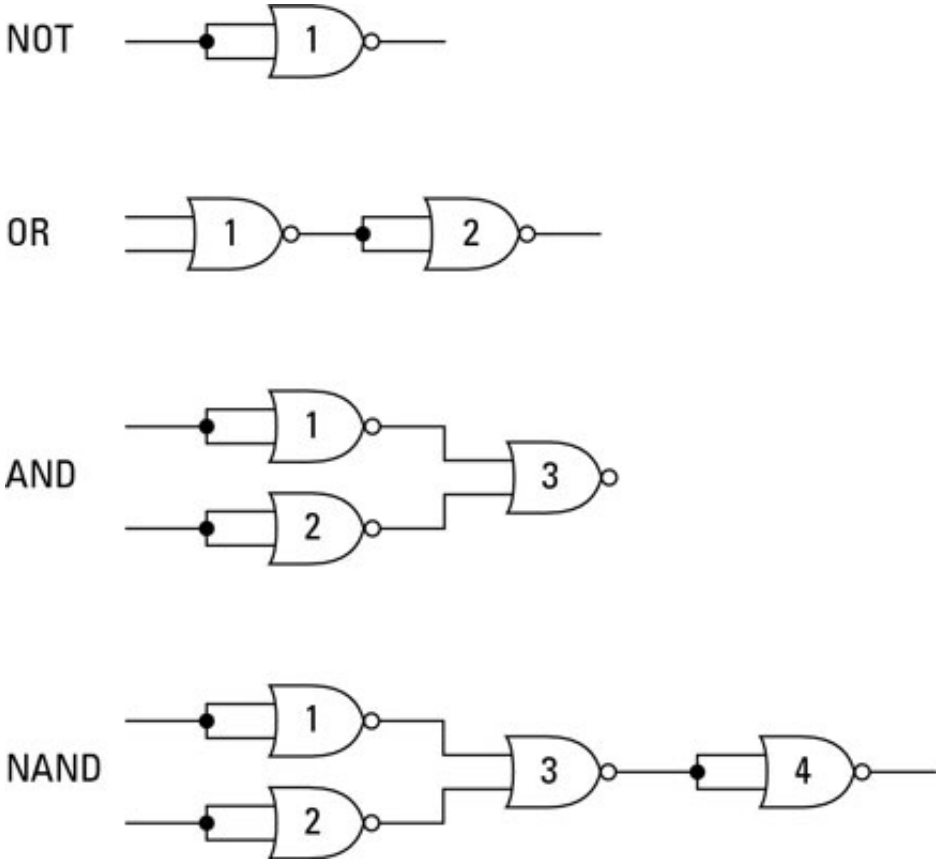
Remember the numbers of NAND/NOR required

Note some question do not restrict to NAND or NOR only

NAND



NOR



Example: Contain only NAND Gates[important]

Convert the **expressions** with only NAND first with $(x')' = x$ and DeMorgan's Theorem

Using the circuit diagram requires simplification

1. * **DeMorgan's theorems** can be used to modify a Boolean expression into one that contains only NAND gates.
Arrange the following steps in the correct order to modify the expression F = AB + A'C as such.

Sort answers in the right order

F = (AB + A'C)' '	(x')' = x	↓
F = [(AB)' (A'C)'] '		↑ ↓
F = [(AB)' ((AA)' C)'] '	NAND gates only	↑

2. * Using ONLY 2-input NAND gates, **how many** such gates are required to implement the following Boolean expression without algebraic simplification?

F = AB + A'C

You may make use of the NAND-only Boolean expression obtained from Q1.

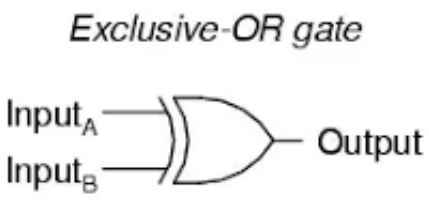
Choose one of the following answers.

- ○ 2x 2-input NAND
- ○ 3x 2-input NAND
- ◉ 4x 2-input NAND
- ○ 5x 2-input NAND

Example above considers only the simplest situation

Ex-OR

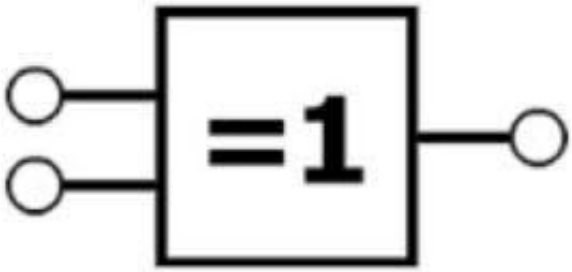
Only consider the two-input XOR



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

$$X = AB' + A'B = A \oplus B$$

IEEE Expression



"=1" means "exactly one input is 1".This implies the other input must be 0.

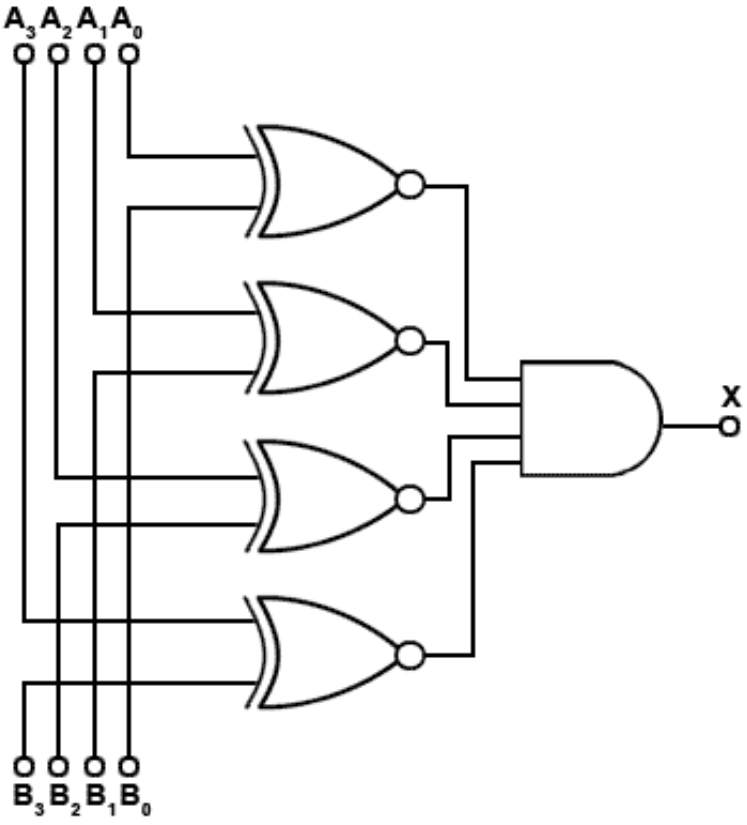
In this course we insist that **each XOR/XNOR gate only has 2 inputs**.The best way is to process 2 inputs at a time.

Make use of the property: A XOR B XOR C = (A XOR B) XOR C = A XOR (B XOR C) = (A XOR C) XOR B

- it can work as an invertor corresponding to one of the inputs

Application

bit-wise comparator output is 0 if the two **multi-bit inputs** are **different** (in the following circuit)



odd-function generator/odd circuit output is **1** if there is an **odd number of 1** among all the inputs

Explanation:$A \oplus B \oplus C = A \oplus (B \oplus C)$

De-Morgan's Theorem: change XOR to XNOR

Parity Circuit

Parity Generator to figure out the parity bit of the data

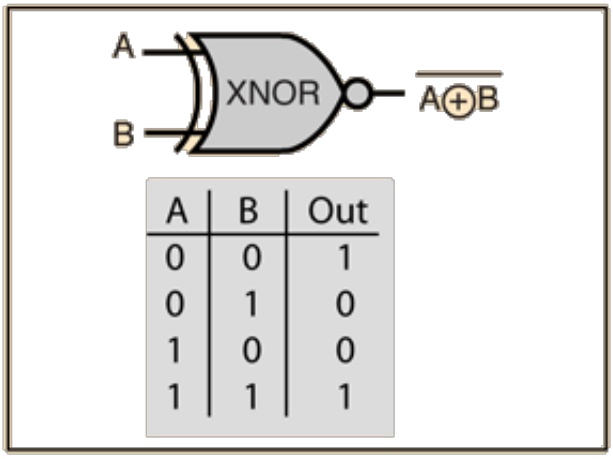
Example generate the parity from an 8-bit data requires 8 XOR, because in this course, XOR has only 2 inputs

Diagram (memorize the structure)

- **Even/odd parity controller** : `Odd/Even #` means 1 for odd, 0 for even

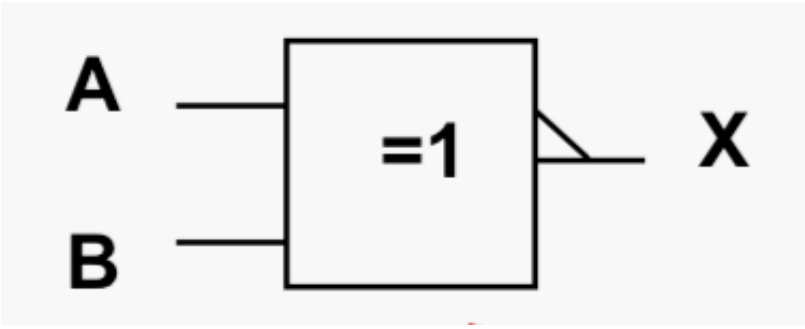
  which leaves flexibility

Ex-NOR



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

IEEE



$$X = AB + A'B' = (A \oplus B)'$$

identical comparator looking for the same

$X = (a_3 XNOR b_3)(a_2 XNOR b_2)(a_1 XNOR b_1)(a_0 XNOR b_0)$, X returns 1 when $a = b$

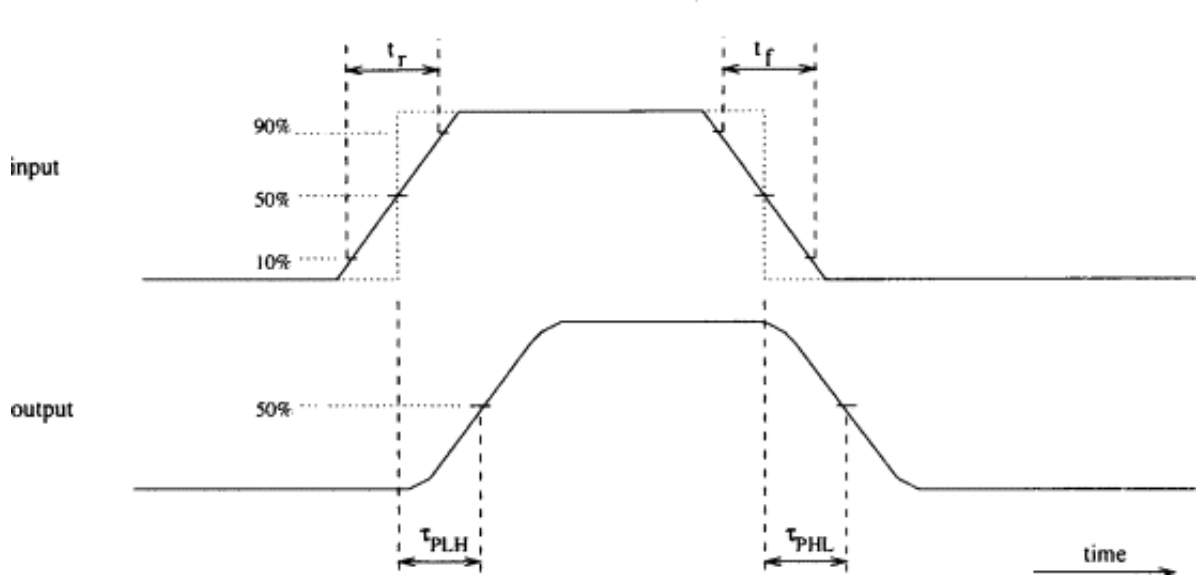
Logic Waveform of X

the value vary according to time

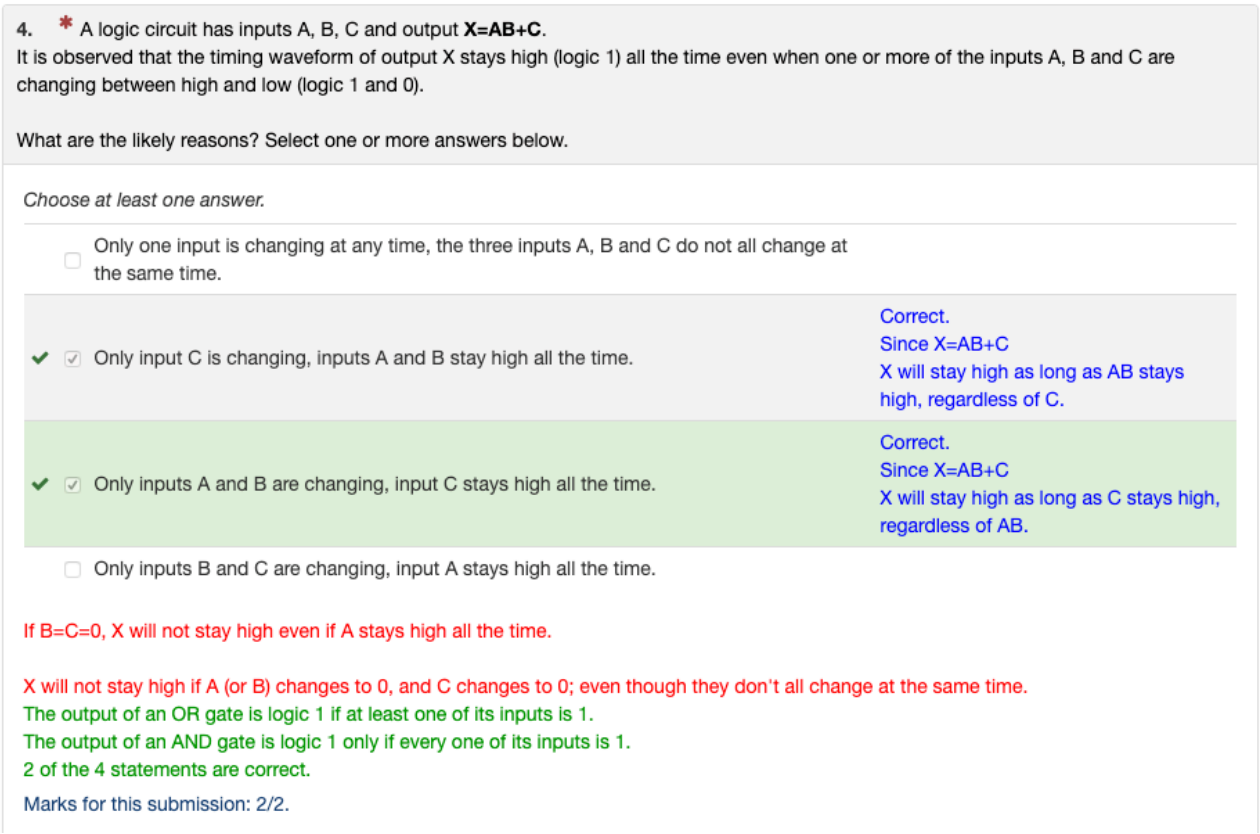
more realistic appearance showing propagation delays, rise time and fall time(commonly ignored when drawing the diagram)- **angled transitions**

- **Propagation delay** is the time taken for the output x to change after input A or input B has changed. (significant factor)
- 3x or 4x Tpd would be too long since gate 1 and gate 2 can **respond simultaneously**.

When we are more interested in the changes of logic levels and less so on timing issues, we can omit drawing rise time, fall time and propagation delays in timing diagrams, but we must bear in mind that they always exist in digital circuits. Propagation delays are drawn only when we want to clearly show its effect on real time behaviour precisely.



Example



4. * A logic circuit has inputs A, B, C and output **X=AB+C**.
It is observed that the timing waveform of output X stays high (logic 1) all the time even when one or more of the inputs A, B and C are changing between high and low (logic 1 and 0).

What are the likely reasons? Select one or more answers below.

Choose at least one answer.

- ☐ Only one input is changing at any time, the three inputs A, B and C do not all change at the same time.
- ✔ ☑ Only input C is changing, inputs A and B stay high all the time. — Correct. Since X=AB+C X will stay high as long as AB stays high, regardless of C.
- ✔ ☑ Only inputs A and B are changing, input C stays high all the time. — Correct. Since X=AB+C X will stay high as long as C stays high, regardless of AB.
- ☐ Only inputs B and C are changing, input A stays high all the time.

If B=C=0, X will not stay high even if A stays high all the time.

X will not stay high if A (or B) changes to 0, and C changes to 0; even though they don't all change at the same time.
The output of an OR gate is logic 1 if at least one of its inputs is 1.
The output of an AND gate is logic 1 only if every one of its inputs is 1.
2 of the 4 statements are correct.
Marks for this submission: 2/2.

Boolean Algebra

simpler, because it doesn't have fraction or negative number, manipulating only with true and false

To evaluate the output value, we need to subsitute the value of inputs. or, directly from the diagram.

boolean variables = logic signals, don't always remain stable

Order of Precedence

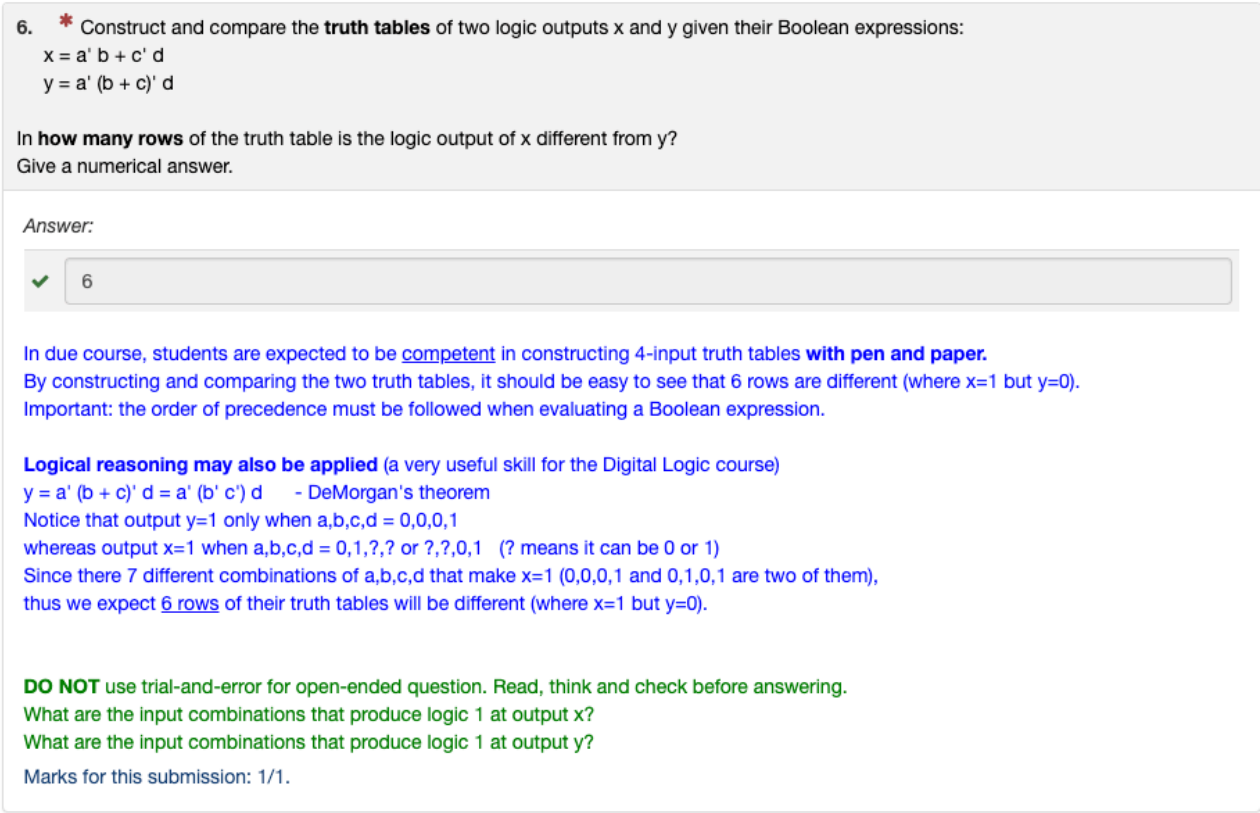
1. Complement(inversion)
2. Parentheses
3. XOR
4. AND
5. OR

   NAND is actually the combination of two operators

Simplify Boolean Algebra

1. reduce the number of components
2. reduce the number of variables

Example: Possibility



6. * Construct and compare the **truth tables** of two logic outputs x and y given their Boolean expressions:
x = a' b + c' d
y = a' (b + c)' d

In **how many rows** of the truth table is the logic output of x different from y?
Give a numerical answer.

Answer:

6

In due course, students are expected to be competent in constructing 4-input truth tables **with pen and paper.**
By constructing and comparing the two truth tables, it should be easy to see that 6 rows are different (where x=1 but y=0).
Important: the order of precedence must be followed when evaluating a Boolean expression.

Logical reasoning may also be applied (a very useful skill for the Digital Logic course)
y = a' (b + c)' d = a' (b' c') d - DeMorgan's theorem
Notice that output y=1 only when a,b,c,d = 0,0,0,1
whereas output x=1 when a,b,c,d = 0,1,?,? or ?,?,0,1 (? means it can be 0 or 1)
Since there 7 different combinations of a,b,c,d that make x=1 (0,0,0,1 and 0,1,0,1 are two of them),
thus we expect 6 rows of their truth tables will be different (where x=1 but y=0).

DO NOT use trial-and-error for open-ended question. Read, think and check before answering.
What are the input combinations that produce logic 1 at output x?
What are the input combinations that produce logic 1 at output y?

Marks for this submission: 1/1.

Boolean Theorems

no need to name theorems

help simplify logic circuits. the same Boolean expression can also be implemented differently.

Axioms

$$X = 0 \text{ if } X \neq 1$$
$$X = 1 \text{ if } X \neq 0$$

Single Variable Theorems

$$\begin{array}{ll} X \cdot 0 = 0 & X + 1 = 1 \\ X \cdot 1 = X & X + 0 = X \\ X \cdot X = X & X + X = X \\ X \cdot X' = 0 & X + X' = 1 \\ & (X')' = X \end{array}$$

DUALITY any theorem or identity in swtiching algebra remains true if 0 and 1 are swapped and $\cdot$ and + are swapped throughtout

$$ABCD + ABC'D' = AB(CD + C'D') \neq AB$$

Duplicate $Y = X = X + X$

Multivariable Theorems

Commutative laws

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$

Associative laws

$$A + (B + C) = (A + B) + C$$
$$A(BC) = (AB)C$$

Distributive laws

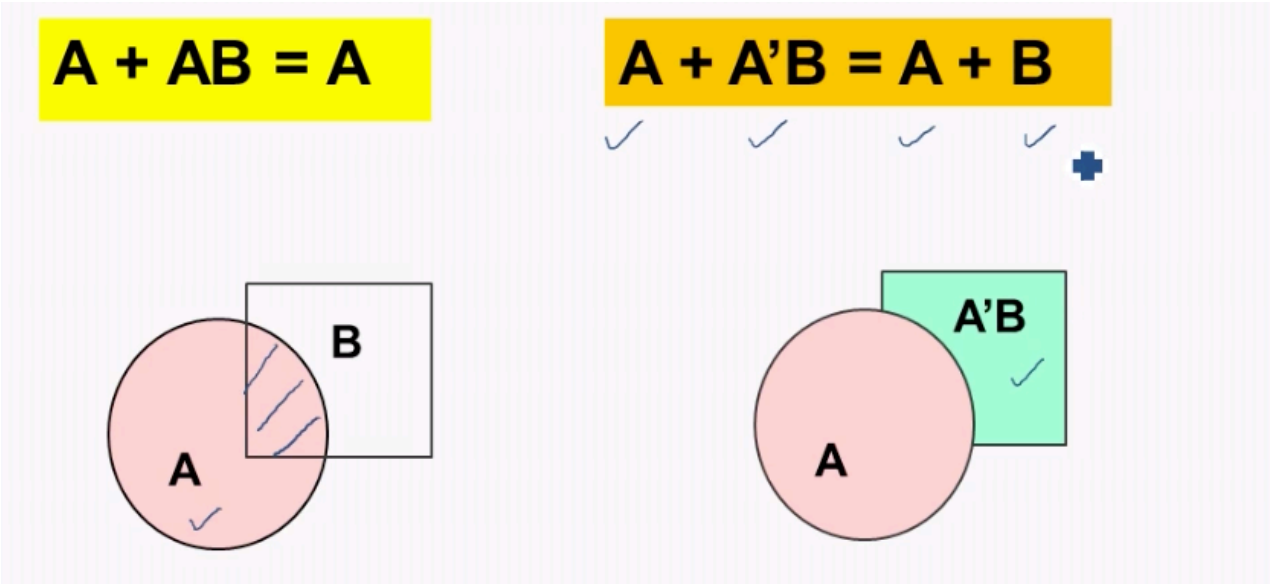
$$A(B + C) = AB + AC$$
$$(A + B)(C + D) = AC + BC + AD + BD$$

Absorption laws

$$A + AB = A$$
$$A + A'B = A + B (*)$$
$$\text{proof of *:} A + A'B = A + AB + A'B = A + B(A + A') = A + B$$

Note $A + A' = 1$ is commonly used in algebraic simplification

Venn diagrams



Consensus Law

$$AB + A'C + BC = AB + A'C$$
$$\text{proof: } BC = ABC + A'BC$$
$$AB + A'C + BC = AB + A'C + ABC + A'BC = AB(1 + C) + A'C(1 + B) = AB + A'C$$

used in 2-input multiplexer

DeMorgan's Theorems

$$(A + B)' = A' \cdot B'$$
$$(AB)' = A' + B'$$
$$(A + B + C + D + \ldots)' = A' \cdot B' \cdot C' \ldots$$
$$(ABCD \ldots)' = A' + B' + C' + \ldots$$

proof: Venn diagram

Example

$$F = (a + b)(ac' + cd)' = (a + b)(ac')'(cd)' = (a + b)(\mathbf{a' + c})(\mathbf{c' + d'})$$

Alternate Logic Gate representations

Make the diagram clearer

Principle

*obtained from the standard symbol by applying **DeMorgan's theorems***

- **adding bubbles to each input and output of the standard symbol**
- **removing bubble from the output of the standard symbol**
- **change the operation symbol from AND to OR and vice versa**
- **NOT symbol shape is not changed**

there is no bubble at the inputs of the standard symbols

For logic flow analysis, matched bubbles can be cancelled out; but the associated gates will be changed. So the "can or cannot" question is always qualified by a "it depends on what you want".

Useage

Interpret Logic Diagram

read the value conditions from the logic component, i.e. with bubble is 1, without bubble is 0

Bubble to Bubble Convention/Bubble Matching no bubble matches with no bubble (as far as possible), including NOT gate

It can be achieved via re-drawing the circuit diagram

Example: Read out when the output=0

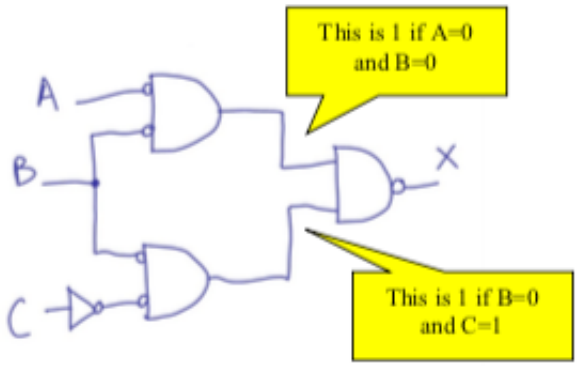


Diagram (b) expresses this relation:

X=0 if
(top path) A=0 and B=0
and
(bottom path) B=0 and C=1

(c)

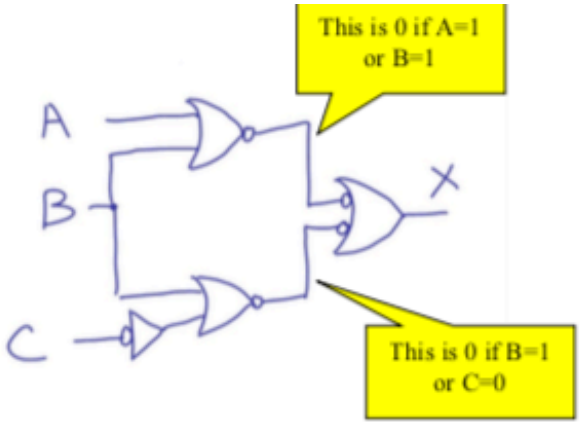


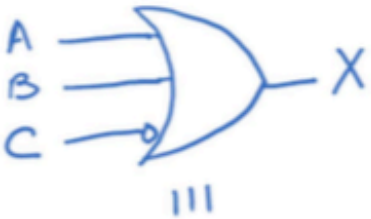
Diagram (c) expresses this relation:

X=1 if
(top path) A=1 or B=1
or
(bottom path) B=1 or C=0

Check against the truth table obtained for the circuit in L1 Practice to verify that both diagrams (b) and (c) are logically correct.

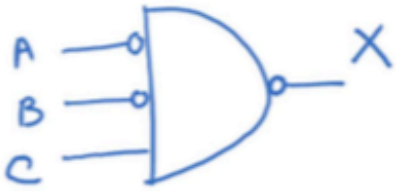
Note that the above diagrams can be made simpler to express the same logic flow:

(c)



III

(b)



Diagrams (b) and (c) are equivalent based on DeMorgan's theorem:

X = A+B+C'
= (A+B+C')''
= (A'B'C'')'

Re-draw the same Circuit

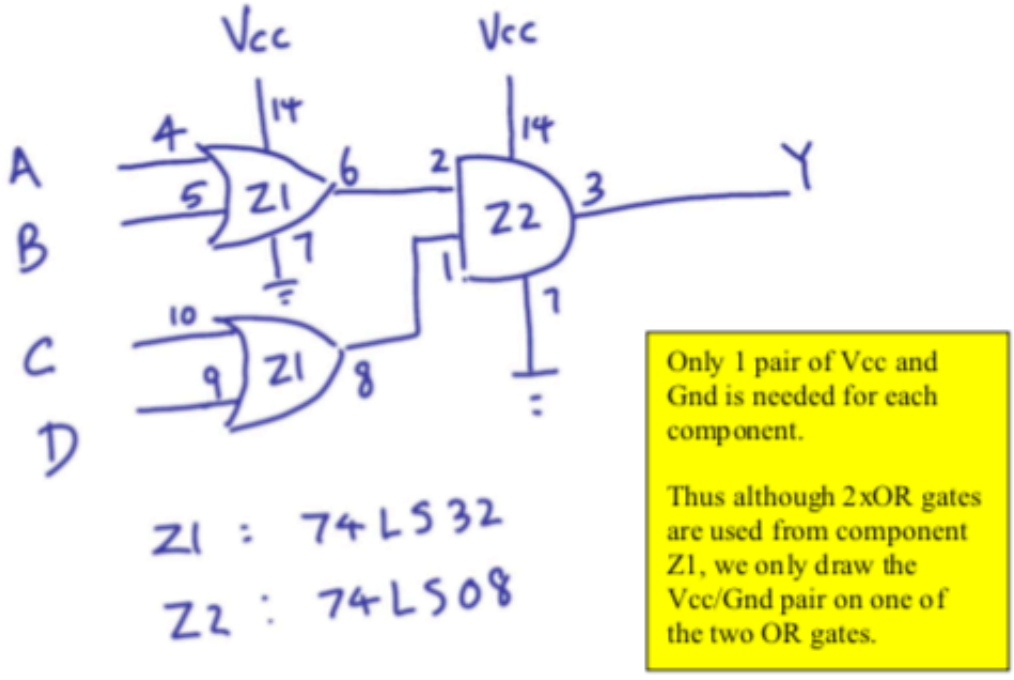
1. Add two inverter in series
2. Simplify the circuit

directly translate the symbol into the alternate one? It's used to satisfy the bubble matching

Special Symbol

X=1 when A=0 and B=1

Circuit Connection Diagram



4. Digital Arithmatic

*In digital circuits, e.g. digital computers and electronic calculators, arithmetic operations are carried out on **binary numbers***

Simple Digital Arithmetic

Binary Addition and Subtraction both begin with the **LSB (least significant bit)**, with the concept of carry and borrow same as decimal arithmatic

The same for the fractional part

Using decimal arithmatic to check the result

logic addition≠ arithmatic addition ≠modulo arithmetic

ultimately, every arithmetic operation relates to addition

In real systems, all operands are likely to have the same number of bits.

Unsigned Addition

keep and write down the carry out of MSB

while for signed addition, because of 2's complement, keeping carry out of MSB doesn't matter

Substraction

using 2's complement numbers

add with 2's complement number

16's complement

Bearing in mind that hexadecimal number is simply a short form for writing binary

16’s complement is a method for representing negative numbers in hexadecimal (base 16).

To form the 16’s complement of **a hex number like 0x1234 you subtract each digit from 0xF** and then add 1 to the whole thing. Subtracting from 0xF (15) forms the 15’s complement **and then adding one to it forms the 16’s complement.**

So 0xFFFF - 0x1234 is 0xEDCB and adding 1 gives you 0xEDCC, which is the 16's complement representation of negative 0x1234

There is a b's complement representation for every base b. The beauty of b's complement is that to calculate A-C in base b, you form the b's complement of C and then ADD rather than subtract.

As for why is it used, well, it isn't. Computers use 2's complement because they are binary, not (usually) hexadecimal. It so happens that if you represent hex numbers in binary, the bit patterns for 16's complement are exactly the same as the bit patterns for 2's complement, but no one says "my computer uses 16's complement"

Multiplication

Unsigned

When $6\ bits \times 4\ bits$, the result is maximum $10\ bits$

- **Left Bitwise Rotation**: In this scenario, the bits are shifted to the left.
- **Right Bitwise Rotation**:In this scenario, the bits are shifted to the right.

Strategy: shift and add

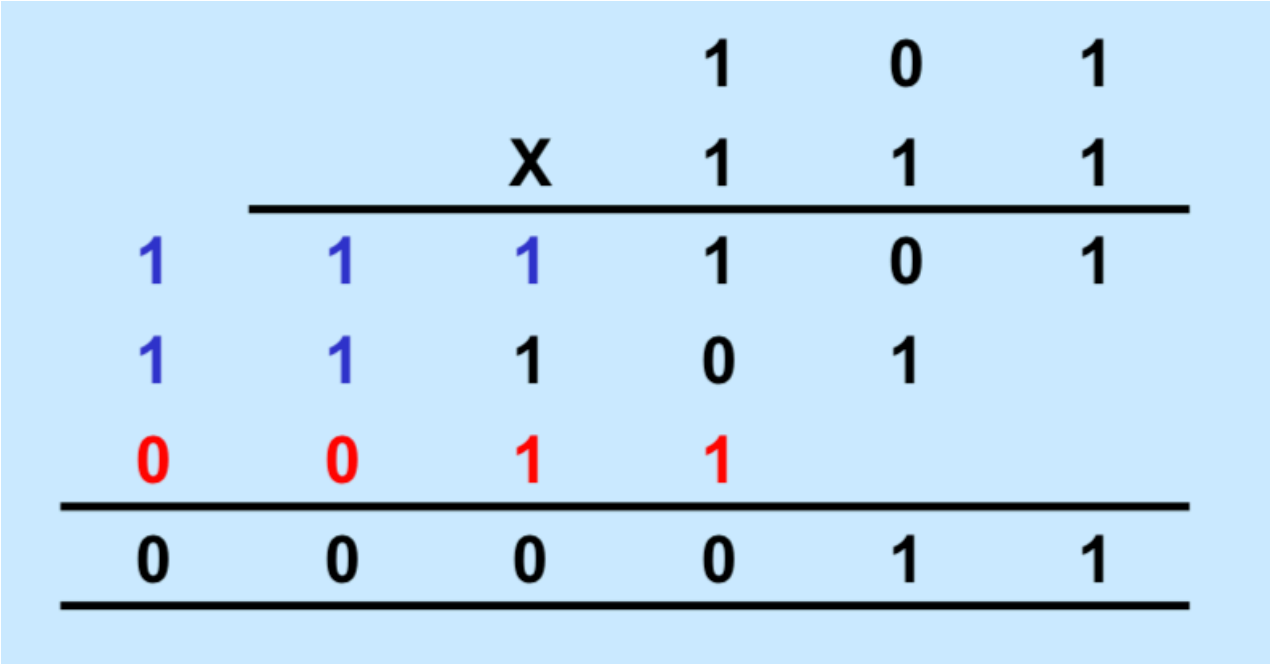
circuit implementation is step by step, which generates many **intermediate results**. Hence, 3-bit adder and registers are required.

be careful of the 0's at the LSB

sign extension (very important)

When $6\ bits \times 4\ bits$, the result is maximum $10\ bits$, similarly.

Then, use the **sign extension**.



Overflow

overflow By **overflow** we mean that the result was a number larger, or smaller, than what is capable of being represented using the given number of bits.

- If we are adding two positive numbers and the left most bit in the result is a **1** then an overflow has occurred.
- If we are adding two negative numbers and the left most bit in the result is a **0** then an overflow has occurred.

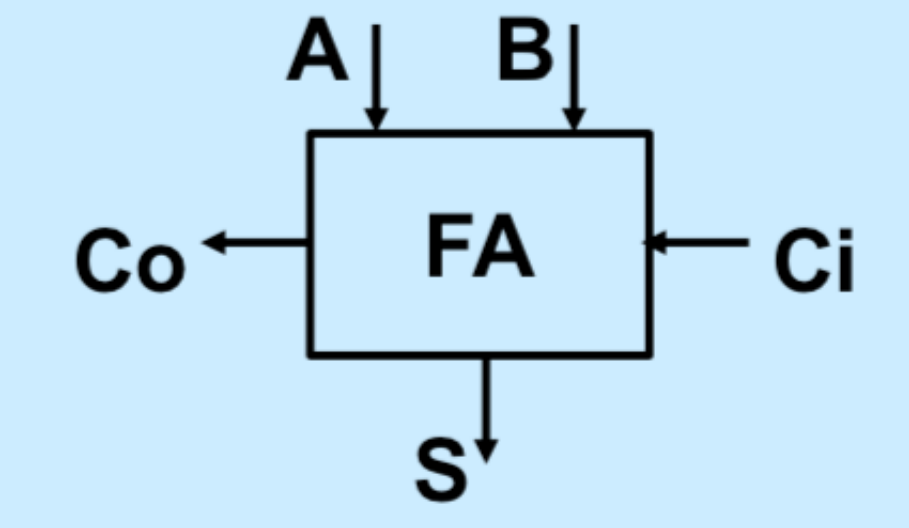
unsigned value doesn't care overflow, instead, they care carry out.

There must be a separate circuit to detect the overflow

occurs when two N-bit operands produce a result that cannot be sufficiently represented by N bits

There is overflow, which cannot mean the circuit is wrong

Overflow detection



1. no overflow when $A \neq B$
2. overflow when $A = B \neq S$, that is

$$\begin{aligned}\mathrm{X}=&(\mathrm{A\ XNOR\ B})\mathrm{AND}(\mathrm{A\ XOR\ S})\\=&\mathrm{Co\ XOR\ Ci}\end{aligned}$$

Division

The easiest is to convert the numbers to **unsigned first,** perform unsigned division, then represent the answer in signed format.

Adder

Half Adder(HA): addition of 2 bits only

$$\begin{aligned}\mathrm{MSD} =\ \mathrm{Carry} &= A \cdot B\\ \mathrm{LSD} =\ \mathrm{Sum} &= A \oplus B\end{aligned}$$

Using table can derive the carry and sum very fast, and they can be expressed in hexdecimal

Full Adder(FA): addition of 3 bits

$$\begin{aligned}\mathrm{MSD} =\ \mathrm{Carry} &= AB + AC_{in} + BC_{in}\\ \mathrm{LSD} =\ \mathrm{Sum} &= A \oplus B \oplus C_{in}\end{aligned}$$

`Cin` is the carry from the previous calculation

Binary number addition (multi-bits): n bits require n FA

Circuit Diagram

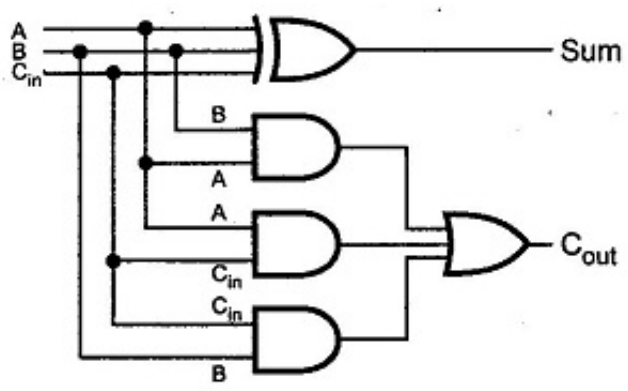
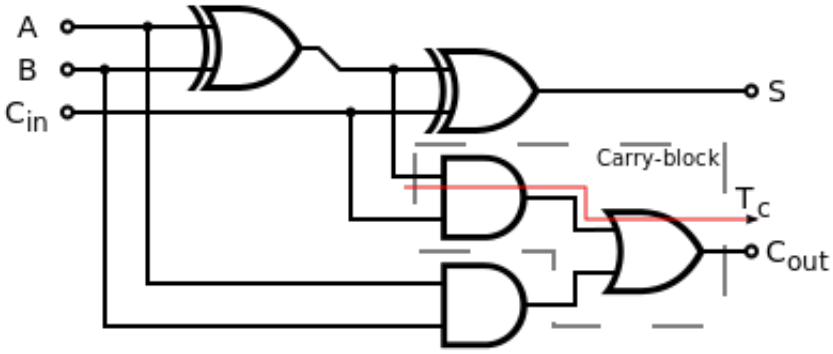
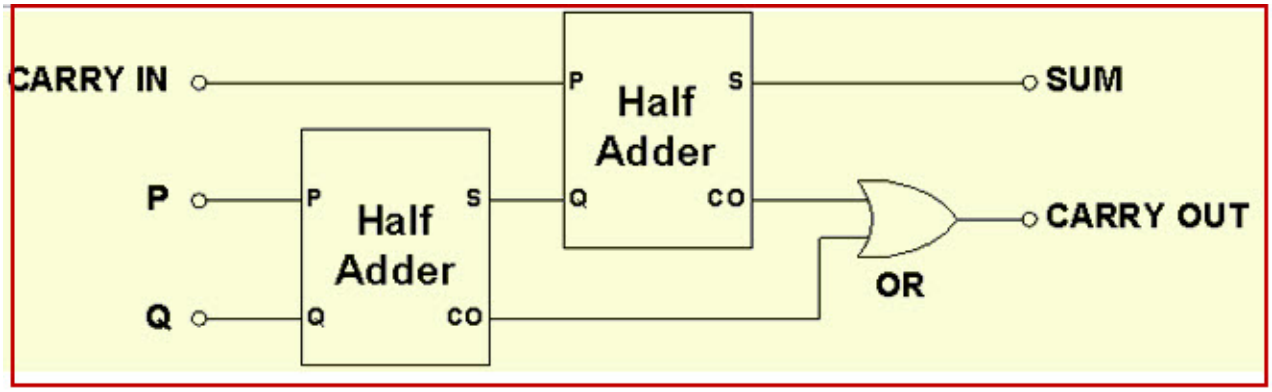


Fig. 3.17 Implementation of full-adder



Implementation with HA



Propogation Delay

For a circuit that comprises multiple gates, consider the total number of logic gates a signal must pass through in the **longest path** from input to output to determine the worst-case propagation delay.

Parallel Adder(Ripple Adder)

What if the digits of the resuilt exceeds the agedness? There should be an overflow.

N full-adders can be cascaded to form an N-bit parallel adder

all the bits of the augend and addend are fed into the adder circuits **simultaneously**

carry propagation addition speed is limited by propagation delays of FAs, e.g. S_n depends on S_{n-1}

*index of digits begins from **0***

Signed Number

two ways: Signed-Magnitude, 2's complement, dependent on the circuit designer

Signed-Magnitude system

Additional bit works as the sign bit, 0 for positive, 1 for negative

Range: $[-(2^{N-1}-1), 2^{N-1}-1]$

0 has two expressions, both negative and positive, which will cause inconvenience

Ungined number = magnitude

2's Complement System/Representation

0 for positive, and it is the same Signed-magnitude system

1 for negative, and it is the 2's complement of the original figure (**Asure MSB is 1**)

Observation

take 4-bit 2's complement number system as example

1. The most negative number is 1000
2. 1000+1=1001; -8+1=-7
3. -1=0b1111
4. The most positive negative number: 0b111111
5. 0b000...00000000 = 0

2's complement Conversion

if the number is positive, things will be very easy

A 2's complement operation changes a positive number to negative and vice-versa, with no change in the magnitude

The direct conversion method(used by computers):

Step 1: invert every bit of a binary number (i.e. perform **1's complement**)

Step 2: add (arithmetic addition) **1** to it

2's complements are reciprocal, which means mutual

Understanding the negative numbers expressed in 2's complement can be considered as the sum of the most negative n-bit value and a positivie value

Conversion Convert numbers into binary format with signed-magnitude system and then conduct 2's complement

sign extension the more significant bits are filled with the sign bit, when there are extra bits

Range $[-(2^{N-1}), 2^{N-1} - 1]$

From 2's complement to Decimal

1. perform 2's complement to convert it to **positive**
2. perform binary-to-decimal conversion on the positive number

short-cut

works on all binary bit patterns

- starting from LSB, copy the bit if it is '0' and repeat process with remaining bits
- copy the bit if it is the first '1', and then invert all the remaining bits
- a sequential process

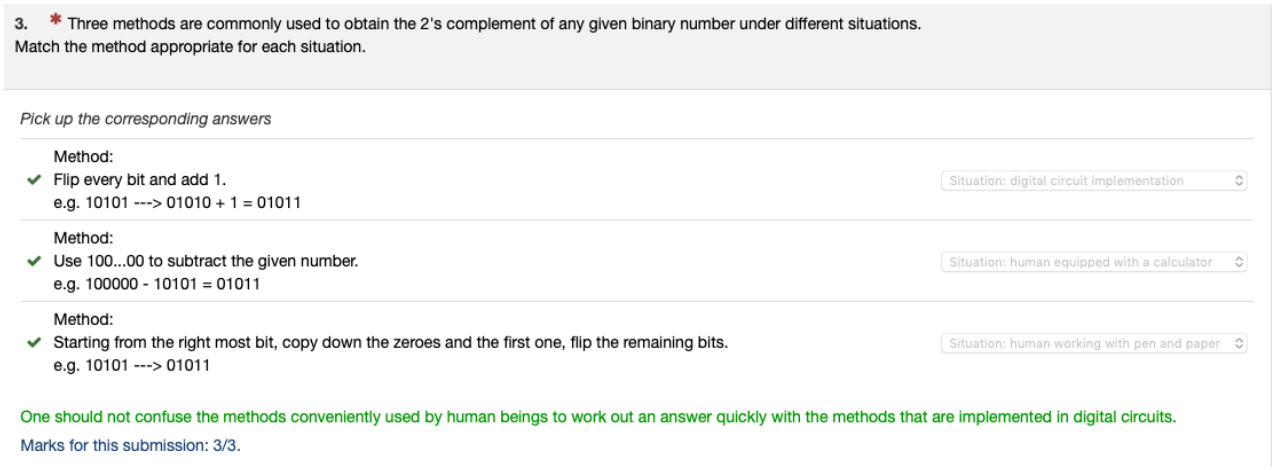
Exceeding values

$128_{10} = 1000\ 0000$

this number exceeds the range of values that can be represented by 8 bits in both sign-magnitude and 2's complement systems.

Thus positive 128 (decimal) cannot be represented in 8 bits. At least 9 bits are required.

Understanding: get the decimal value quickly



Given the same 2's complement representation 11101100 in Q6, the following steps may also be used to determine its signed decimal value:

11101100 is the sum of 10000000 and 01101100

10000000 is the 2's complement representation of negative 128 (decimal)

01101100 is the 2's complement representation of positive 108 (decimal)

-128 + 108 = -20

The signed decimal value is -20

Exception

Dealing with the most negative number that can be represented given a number of bits

Benefits

1. subtraction of numbers can be carried out in the same way as addition
2. the same set of hardware can be used for both subtraction and addition

Sign extension

SM system: Only MSB is sign bit. The rest are plain magnitude.

2's complement representation: **must** apply sign extension

Operations in 2's complement system

The pair of numbers to be added/subtracted must have the same size

And the resulting sum must also be of the same size

ultimately, subtraction and addition are both **addition**

Hence, **carry out is ignored**

Addition

sign bits are added like the other bits

the carry out is ignored

Subtraction

$$-B = 2\text{'s complement of } B$$

Multiplication

ultimately, addition and shift

If the multiplier is negative, need to take care of negative weight of MSB (i.e. the sign bit)

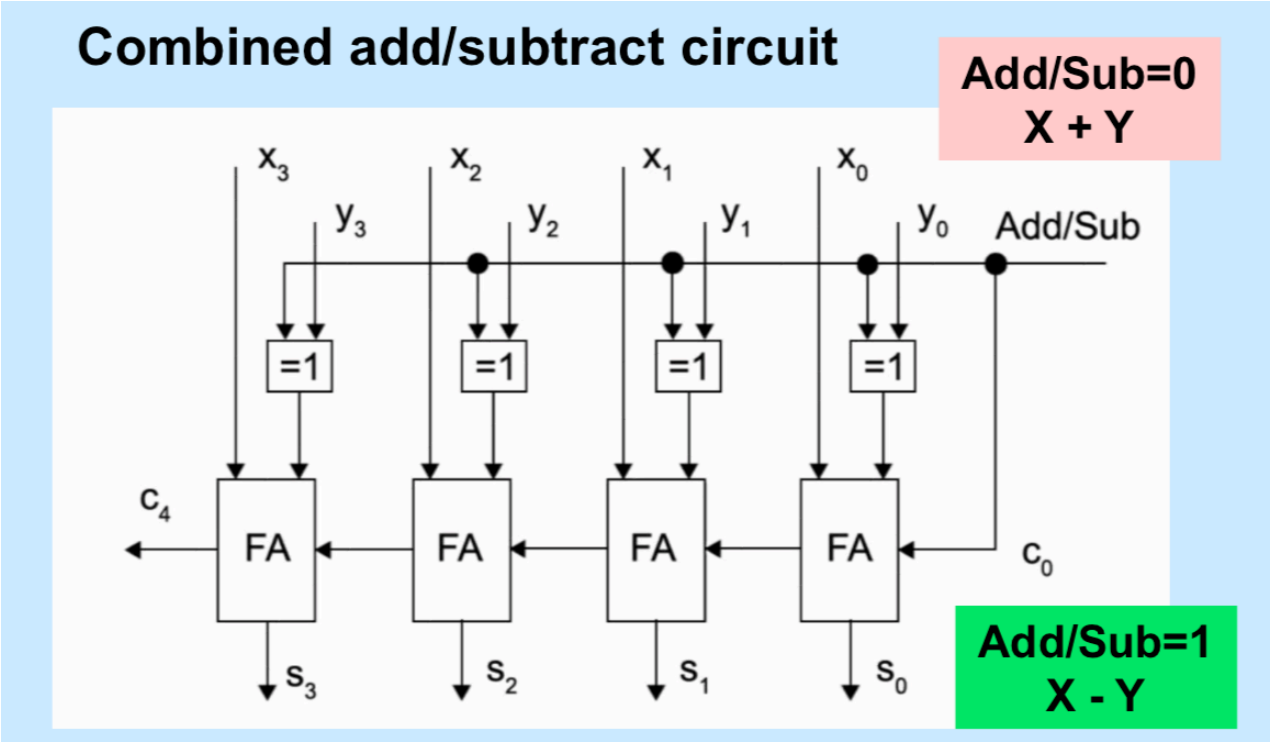
Treat multiplier as a sum of 2 parts:

- MSB – negative part
- Remaining bits – positive part

Circuit Implementation

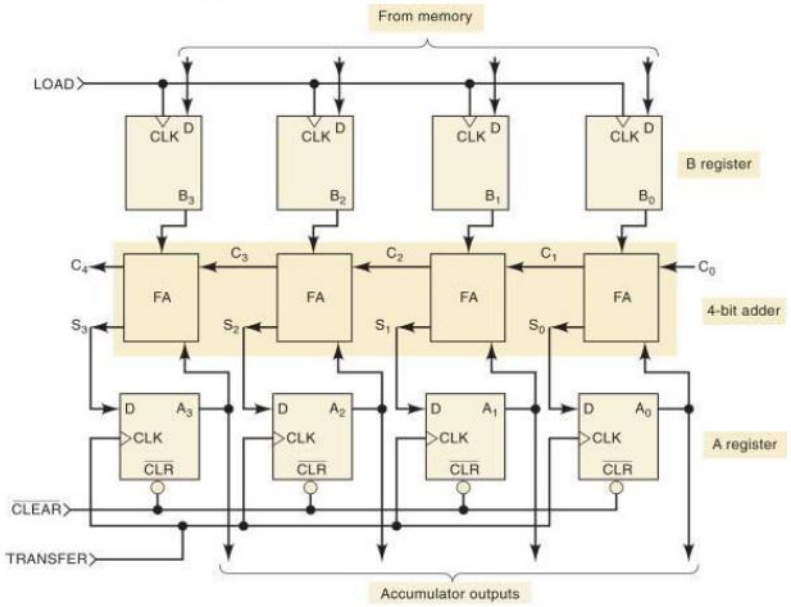
$X - Y = X + (-Y)$

- X, Y can be negative or positive
- 2's complement of Y can be easily obtained by adding 1 to the 1's complement of Y



Parallel FA with Registers

12. ALU - Parallel adder with registers



a) Determine the contents of the *A* register after the following sequence of operations and the carry-out bit C_4 (Assume $C_0 = 0$): [*A*] = 0000, [0110] → [*B*], [*S*] → [*A*], [1110] → [*B*], [*S*] → [*A*].

b) Full adder and half adder design (from truth table to Boolean algebra and circuit diagram).

CLEAR: registers are cleared to 0

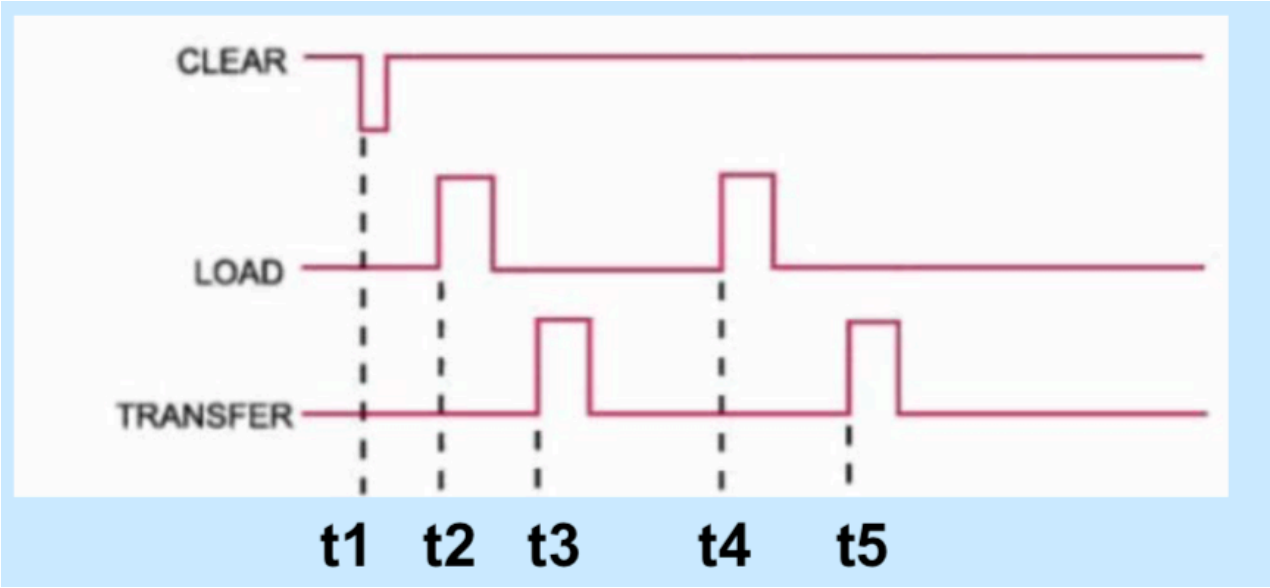
Two registers are used to store the 2 values to be added. One of the two registers also store the result of the sum. Once the sum is obtained, the value is meant to be read by the CPU. Next, a fresh pair of values will be stored into the registers for the next add operation.

Register A: store intermediage results and final results

After the number in Register B transferred to full adders, the value in R_B won't change but wait to be overwritten.

Timing Diagrams

PGT stands for positive-going transitions



What will be the FA's output • shortly before t4? Before t4, register B still contains the value of X. So FA = X+X • **shortly after t5?** Register A has X+Y, register B has Y. So FA = X + Y + Y if it has completed the addition.

FAs are always trying to add the numbers of R_A and R_B automatically, but only the expected value (i.e. X+Y) would be transferred to RA, and after it is transferred, the output of FA may not be expected or relevant (i.e. X+Y+Y).

t1: CLEAR* clears the contents of A register to 0's

t2: PGT of first LOAD pulse transfers operand X from memory into B register

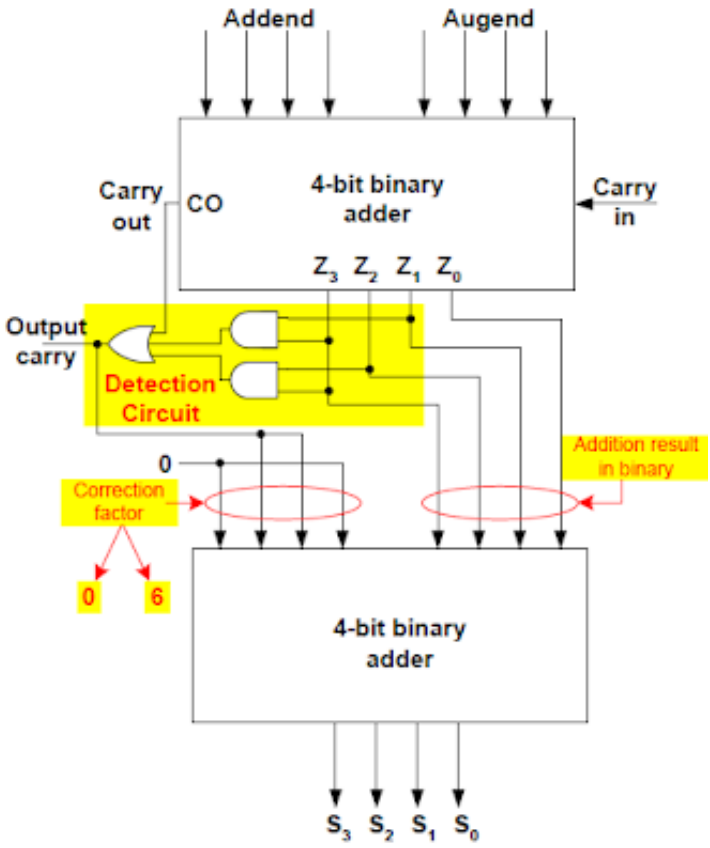
t3: PGT of first TRANSFER pulse transfers FA output (=X) into A register

t4: PGT of second LOAD pulse transfers operand Y from memory into B register

t5: PGT of second TRANSFER pulse transfers FA output (=X+Y) into A register

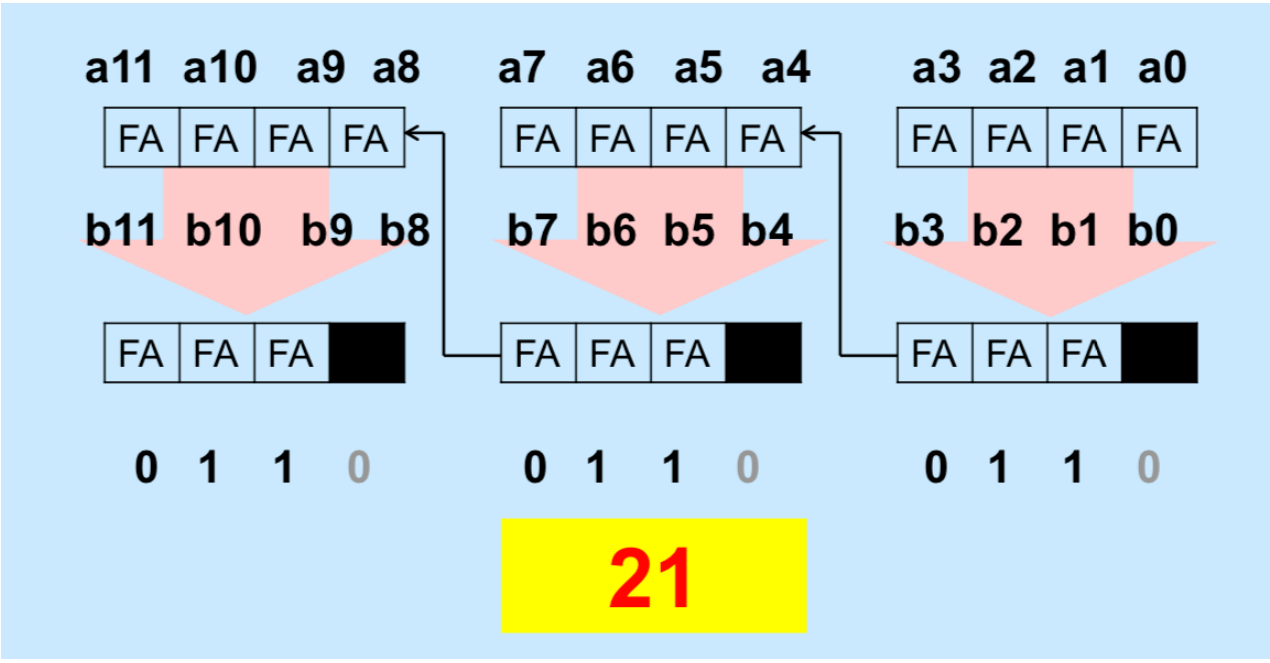
For t_1, $X + 0$ to load X into R_A

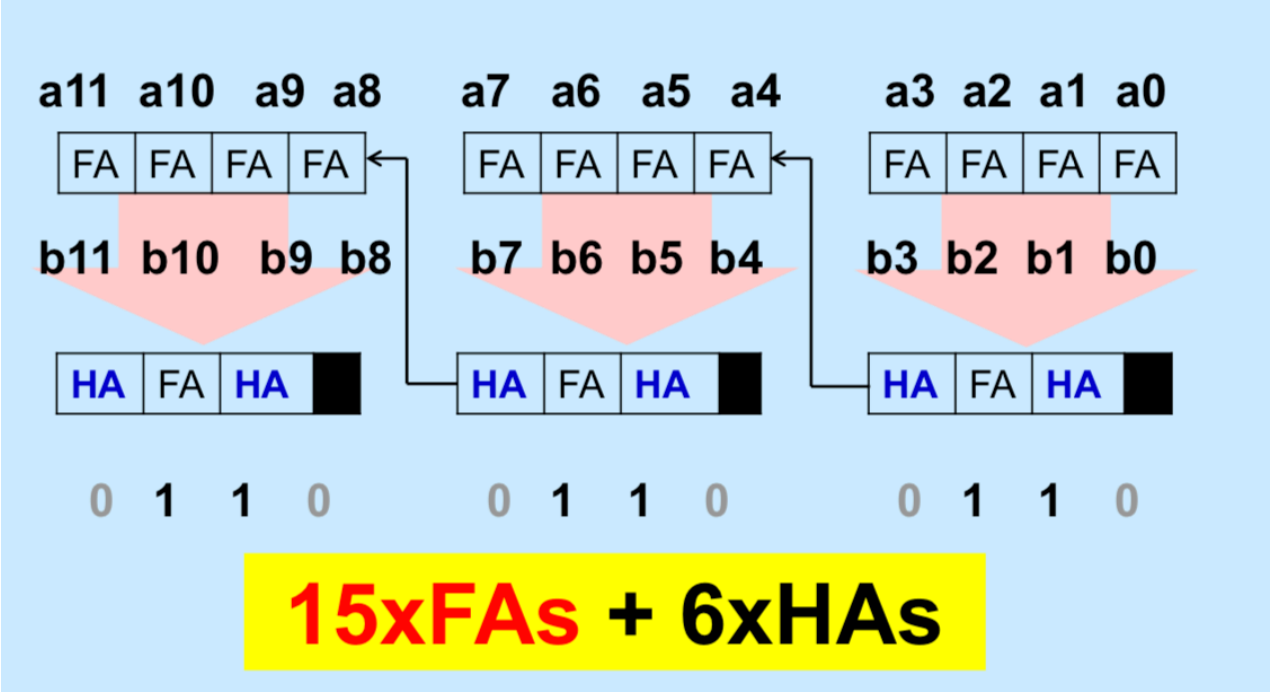
BCD Addition



Total numbers of FAs

the total number of FAs needed for 3-digit BCD addition —24, the carry out as C_{in}





5. Combinational

F = a + b is an SOP (sum-of-product) expression and it is also a POS (product-of-sum) expression.

4. * Given the canonical **product-of-maxterm** expression F(a,b,c,d) = π M (1, 4, 10, 13, 14)

Arrange the following steps in the correct order to algebraically simplify the above and obtain the minimum-cost **product-of-sum** expression.

Sort answers in the right order

- ✔ F = (a+b+c+d')(a+b'+c+d)(a'+b+c'+d)(a'+b'+c+d')(a'+b'+c'+d)
- ✔ F = (a'+c'+d+b)(a'+c'+d+b')(a+b+c+d')(a+b'+c+d)(a'+b'+c+d')
- ✔ F = (a'+c'+d)(a+b+c+d')(a+b'+c+d)(a'+b'+c+d') (minimum-cost POS)

The canonical expression has 5 maxterms. The POS has only 4 sums.
Each maxterm has 4 variables. One of the sum in the POS has only 3 variables (a'+c'+d).
Thus the POS is more simplified than the canonical expression.
After you have learned the Karnaugh map technique in L8, it will be obvious to tell whether or not an expression is most simplified.
For POS, minimum-cost means the expression has the minimum number of sums, and sum has the minimum of variables.
A very common Boolean theorem used to simplify expression by eliminating a variable is this:
(X + Y)(X + Y') = X + XY + XY' + YY' = X (1 + Y + Y') + 0 = X
Marks for this submission: 3/3.

Forming fewer number of loops means fewer products need to be included in an SOP or fewer sums need to be included in a POS expression.

A sum-of-minterm expression doesn't always produce 1 (it could be 0); we choose the term with the output of 1.

Combinational Logic Circuit Design Process

1. Write down the truth table, figuring out the inputs and outputs
2. Express in cononical form
3. Or, with K-map

alternate logic symbol helps designers care more about the relation between outputs and inputs, rather than the specific gates

Active High/Low

active high: does something when it is high

most of the time, it is low. e.g. `load` and `transfer`

SUBTRACT is active high

active low: does something when it is low

most of time, it is high. e.g. `clear` , which always clears something to 0

ADD is active low

Enable/Disable

*make a part of the circuit work or unwork, but **temporarily***

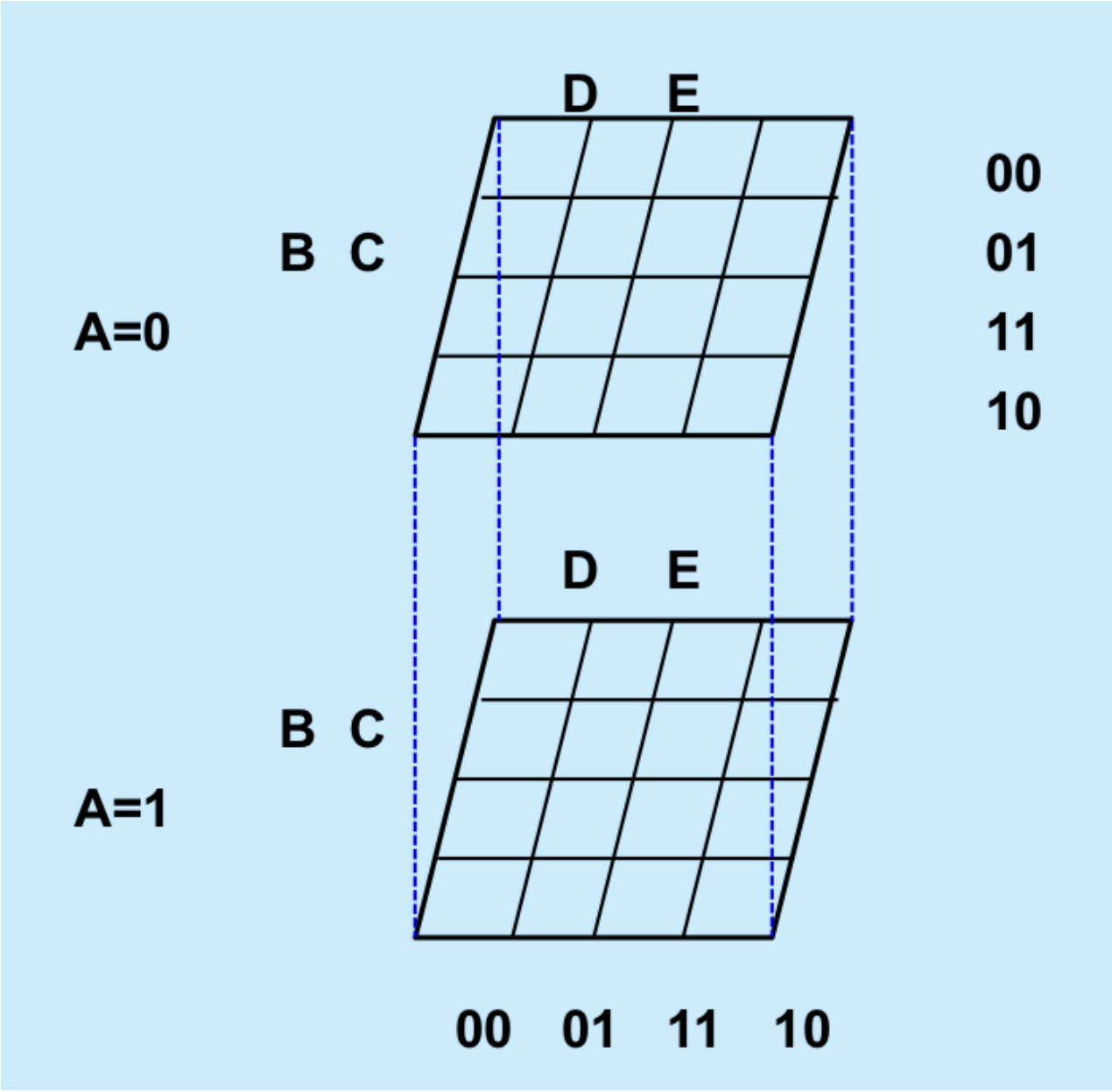
AND is most commonly used gate

or: active low (1)

K-map

In reality, there are at most 6 variables to use K-map

Shown in the picture, the 5-variable Kmap:



1. Verilog, standardized as IEEE 1364, is a hardware description language used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. ↩

2. A test bench or testing workbench is an environment used to verify the correctness or soundness of a design or model. ↩

3. Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. ↩

4. How does the output change with the inputs ↩