Combinational Logic

CS207 Chapter 5

James YU
yujq3@sustech.edu.cn

Department of Computer Science and Engineering
Southern University of Science and Technology

Oct. 12, 2022

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Combinational logic

- Logic circuits for digital systems:
  - combinational logic,
  - sequential logic (next lectures).
- *Combinational*?
  - Output determined by the combination of inputs.
  - Perform an operation specified by a set (combination) of Boolean functions.

only

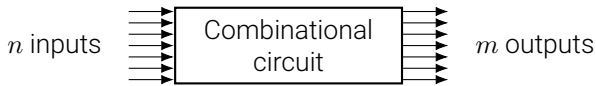
sequencial : have recursive

depends also on previous input

Combinational logic

- An interconnection of logic gates that
  - react to values of input signals,
  - produce output signal values,
  - n inputs from external sources, and
  - m outputs go to external destinations.

input ↕ ⟹ output MAY ↕

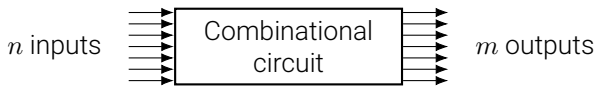


denote combinational circuit

without looping

Combinational logic

- Two representations of a combinational circuit:
  - 2^n possible input combinations: truth table, or
  - m outputs: m Boolean functions, each expressed with the n inputs.



Analysis of combinational logic

- Analysis of a combinational circuit: determine the function of the circuit.
  - Given logic diagram,
  - develop a set of Boolean functions, a truth table, an optional explanation of the circuit operation.

If something is optional, you can choose whether or not you do it or have it.

- If a function name or an explanation is given along the circuit, just verify if the given information is correct.

omit it

Analysis of combinational logic

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

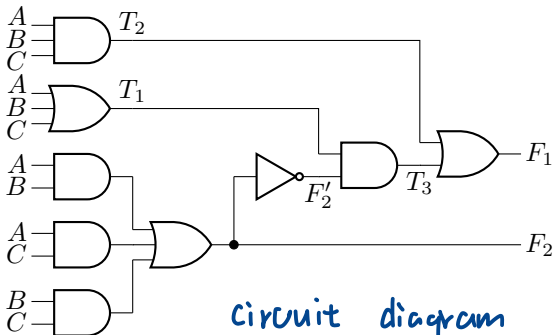
- Obtain the output Boolean functions:
  - Label all gate outputs that are a function of only inputs, no other intermediate variables. Determine their Boolean functions.
  - Label all gates that are a function of inputs and the gates in the previous step. Determine their Boolean functions.
  - List output Boolean functions, squeeze the intermediate variables.

aim

coming between two things in time, place, order, character, etc

Analysis of combinational logic

- Obtain the output Boolean functions:
  - Label all gate outputs that are a function of only inputs, no other intermediate variables. Determine their Boolean functions.



circuit diagram

use F_n to represent functions of output

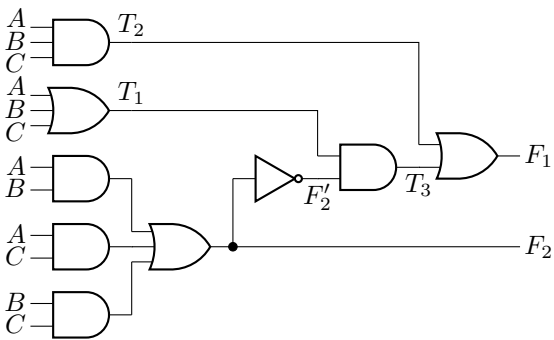
use T_n to represent intermediate functions

- $T_1 = A + B + C.$
- $T_2 = ABC.$
- $F_2 = AB + AC + BC.$

↑ intermediate variables

Analysis of combinational logic

- Obtain the output Boolean functions:
  - Label all gates that are a function of inputs and the gates in the previous step. Determine their Boolean functions.

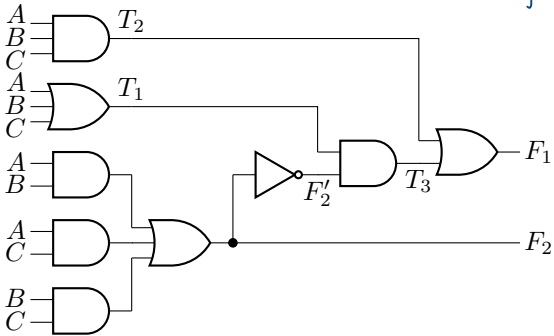


- $T_3 = F_2' + T_1$.
- $F_1 = T_2 + T_3$.

Analysis of combinational logic

- Obtain the output Boolean functions:
  - List output Boolean functions, squeeze the intermediate variables.

use original function



$$
\begin{aligned}
F_1 &= T_2 + T_3 \\
&= ABC + F_2' T_1 \\
&= ABC + \\
&\quad (AB + AC + BC)' \\
&\quad (A + B + C) \\
&= ABC + A'B'C + \\
&\quad AB'C' + A'BC'
\end{aligned}
$$

hard to calculate

Analysis of combinational logic

more straight forward

The demerits of something or someone are their faults or disadvantages.

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Truth table is simple with Boolean function
  - Determine the number of input variables. For n inputs, form the 2^n combinations from 0 to 2^n-1.
  - Label the outputs of the intermediate gates.
  - Obtain the truth table for these outputs.
  - Obtain the truth table for the remaining outputs.

no matter which method is used to analysis the combinational logic The idea is similar. Just like recursion, first we consider the intermediate functions whose input is the real input.

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design of combinational logic

reverse the analysis: purpose ⟶ circuit

- Design of a combinational circuits: develop a logic circuit diagram or a set of Boolean functions.
  - From specification of the design objective
- Involves the following steps:
  - Determine required number of inputs and outputs.
  - Derive the truth table. listing all possible imput
  - Obtain simplified Boolean function for each output.
  - Draw logic diagram and verify the correctness. (optional)

Kmap

Design of combinational logic

inverse of analysis

- Example: Convert from BCD decimal code to excess-3 code.

binary-coded decimal

another code represent o-p

Input BCD				Output Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

- Obviously, four inputs and four outputs
- And we already have the truth table
- Then the Boolean functions. How?
  - Remember K-maps?

Karnaugh

only ten row

what are another 6 ?

another 6 is NOT valid BCD code

They will NOT be provided ⟹ X

Representation [edit]

Biased codes are a way to represent values with a balanced number of positive and negative numbers using a pre-specified number *N* as a biasing value. Biased codes (and Gray codes) are non-weighted codes. In excess-3 code, numbers are represented as decimal digits, and each digit is represented by four bits as the digit value plus 3 (the "excess" amount):

- The smallest binary number represents the smallest value (0 – excess).
- The greatest binary number represents the largest value (2^{N+1} – excess – 1).

Excess-3, and Stibitz code

Decimal	Excess-3	Stibitz	BCD 8-4-2-1	Binary	3-of-6 CCITT extension[13][1]	4-of-8 Hamming extension[1]
–3	0000	pseudo-tetrade	N/A	N/A	N/A	N/A
–2	0001	pseudo-tetrade				
–1	0010	pseudo-tetrade				
0	**0011**	**0011**	0000	0000	…**10**	…**0011**
1	**0100**	**0100**	0001	0001	…**11**	…**1011**
2	**0101**	**0101**	0010	0010	…**10**	…**0101**
3	**0110**	**0110**	0011	0011	…**10**	…**0110**
4	**0111**	**0111**	0100	0100	…**00**	…**1000**
5	**1000**	**1000**	0101	0101	…**11**	…**0111**
6	**1001**	**1001**	0110	0110	…**10**	…**1001**
7	**1010**	**1010**	0111	0111	…**10**	…**1010**
8	**1011**	**1011**	1000	1000	…**00**	…**0100**
9	**1100**	**1100**	1001	1001	…**10**	…**1100**
10	**1101**	pseudo-tetrade	pseudo-tetrade	1010	N/A	N/A
11	**1110**	pseudo-tetrade	pseudo-tetrade	1011		
12	**1111**	pseudo-tetrade	pseudo-tetrade	1100		
13	N/A	N/A	pseudo-tetrade	1101		
14			pseudo-tetrade	1110		
15			pseudo-tetrade	1111		

To encode a number such as 127, one simply encodes each of the decimal digits as above, giving (0100, 0101, 1010).

Excess-3 arithmetic uses different algorithms than normal non-biased BCD or binary positional system numbers. After adding two excess-3 digits, the raw sum is excess-6. For instance, after adding 1 (0100 in excess-3) and 2 (0101 in excess-3), the sum looks like 6 (1001 in excess-3) instead of 3 (0110 in excess-3). In order to correct this problem, after adding two digits, it is necessary to remove the extra bias by subtracting binary 0011 (decimal 3 in unbiased binary) if the resulting digit is less than decimal 10, or subtracting binary 1101 (decimal 13 in unbiased binary) if an overflow (carry) has occurred. (In 4-bit binary, subtracting binary 1101 is equivalent to adding 0011 and vice versa.)

Motivation [edit]

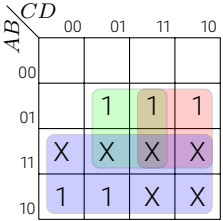
The primary advantage of excess-3 coding over non-biased coding is that a decimal number can be nines' complemented[1] (for subtraction) as easily as a binary number can be ones' complemented: just by inverting all bits.[1] Also, when the sum of two excess-3 digits is greater than 9, the carry bit of a 4-bit adder will be set high. This works because, after adding two digits, an "excess" value of 6 results in the sum. Because a 4-bit integer can only hold values 0 to 15, an excess of 6 means that any sum over 9 will overflow (produce a carry out).

Another advantage is that the codes 0000 and 1111 are not used for any digit. A fault in a memory or basic transmission line may result in these codes. It is also more difficult to write the zero pattern to magnetic media.[1][14][11]

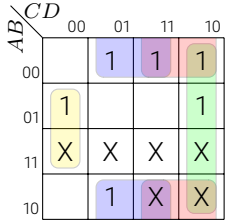
Design of combinational logic

$$w = A + BC + BD$$

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

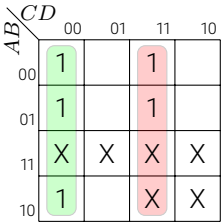


X: don't care.
the combination will not appear

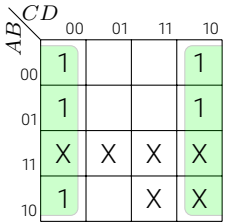


Design of combinational logic

$$y = CD + C'D'$$



$$z = D'$$



Design of combinational logic

- We manipulate these Boolean functions to reuse common gates:
  - $w = A + BC + BD = A + B(C + D)$.
  - $x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$.
  - $y = CD + C'D' = CD + (C + D)'$.
  - $z = D'$.

Scrap here in exam is okay.

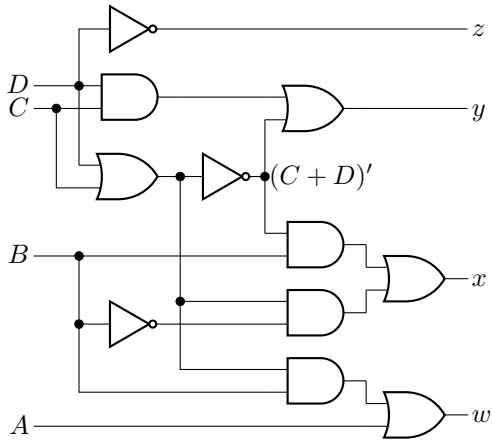
But in manufacturing, we look for the same things

further transformation

⇒ final simplified have same things

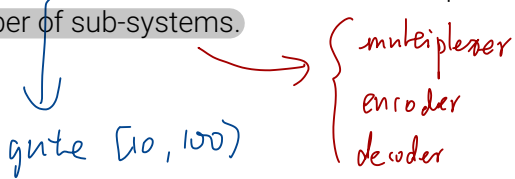
⇒ make final result more be simplified

Design of combinational logic



Combinational logic design with MSI circuits

- Since the introduction of MSI and LSI circuits, the traditional methods of logic design have largely been superseded.
  - Traditionally, the design engineer has developed a Boolean equation as the solution to a particular problem.
  - This function has then been minimised and implemented using SSI circuits.
- In practice, many combinational circuits may have a large number of inputs and outputs.
  - Consequently the use of truth tables in the design of such circuits is impractical.
- The development of MSI circuits has led to the technique of splitting a complex design into a number of sub-systems.



Wiki: integrated circuit

Generations [edit]

See also: List of semiconductor scale examples, MOS integrated circuit, and Transistor count

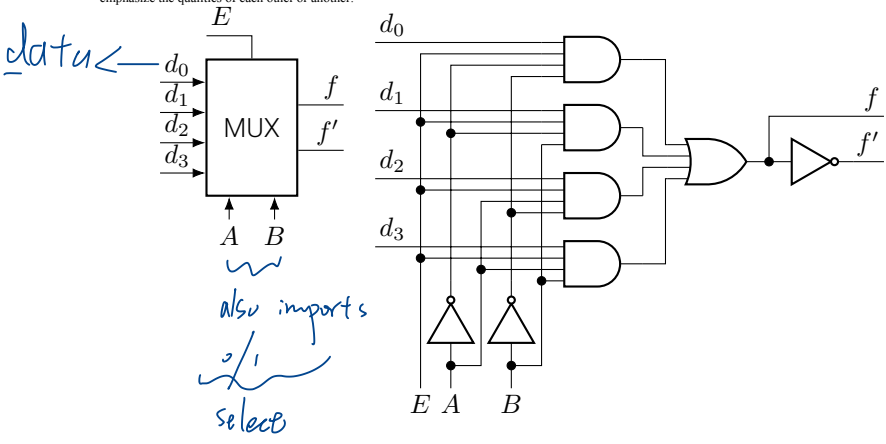
In the early days of simple integrated circuits, the technology's large scale limited each chip to only a few transistors, and the low degree of integration meant the design process was relatively simple. Manufacturing yields were also quite low by today's standards. As metal–oxide–semiconductor (MOS) technology progressed, millions and then billions of MOS transistors could be placed on one chip,[81] and good designs required thorough planning, giving rise to the field of electronic design automation, or EDA. Some SSI and MSI chips, like discrete transistors, are still mass-produced, both to maintain old equipment and build new devices that require only a few gates. The 7400 series of TTL chips, for example, has become a de facto standard and remains in production.

Acronym	Name	Year	Transistor count[82]	Logic gates number[83]
SSI	*small-scale integration*	1964	1 to 10	1 to 12
MSI	*medium-scale integration*	1968	10 to 500	13 to 99
LSI	*large-scale integration*	1971	500 to 20 000	100 to 9999
VLSI	*very large-scale integration*	1980	20 000 to 1 000 000	10 000 to 99 999
ULSI	*ultra-large-scale integration*	1984	1 000 000 and more	100 000 and more

Multiplexer

- A multiplexer (MUX) selects 1-out-of-n lines where n is usually 2, 4, 8, or 16.
- A block diagram of a multiplexer having four input data lines d0, d1, d2, and d3 and complementary outputs f and f'.

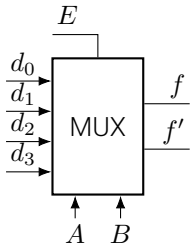
combining in such a way as to enhance or
emphasize the qualities of each other or another:

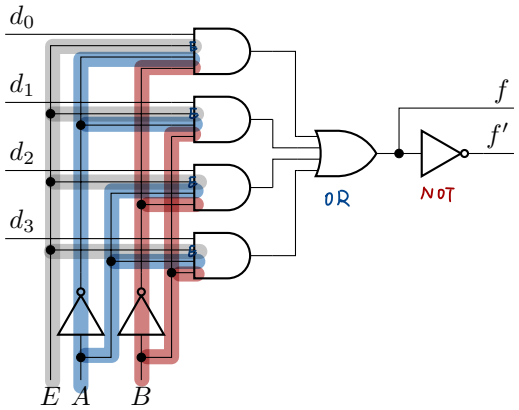


Multiplexer

- The device has two control or selection lines A and B and an enable line E.
- The characteristic equation of the multiplexer is
$$f = A'B'd_0 + A'Bd_1 + AB'd_2 + ABd_3.$$

no matter what value is A.B among AB' & A'B & AB' & AB there is always 1 and only 1 formula equals to 1

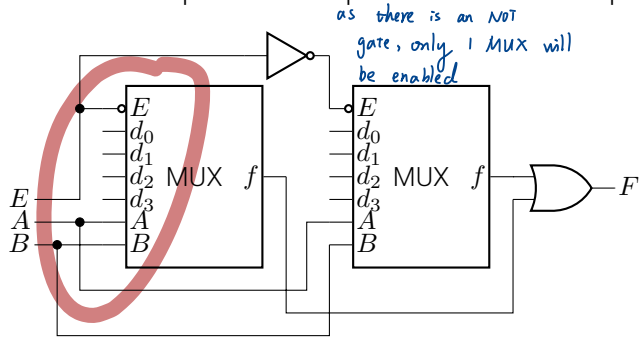




enable line is linked to every AND

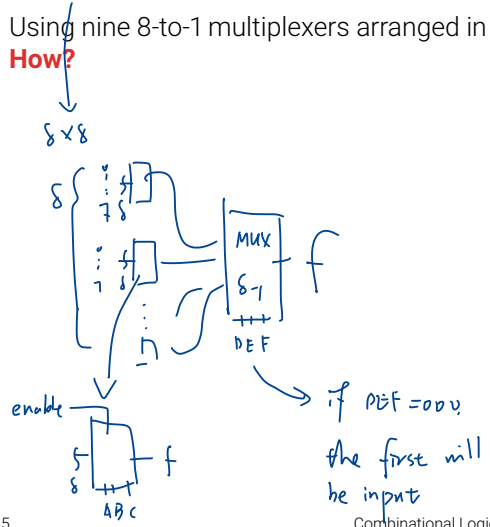
Interconnecting multiplexers

- Data within a digital system is normally processed in parallel form in order to increase the speed of operation.
- If the output of the system has to be transmitted over a relatively long distance then a parallel-to-serial conversion will take place.
- Example: An 8-bit word is presented in parallel at the data inputs.



Interconnecting multiplexers

- The principle of data selection can be extended to allow the selection of 1-out-of-64 lines.
  - Using nine 8-to-1 multiplexers arranged in two levels of multiplexing.
  - **How?**

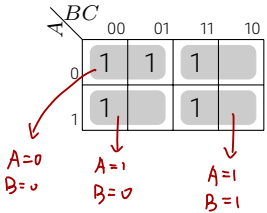


Multiplexer as a Boolean function generator

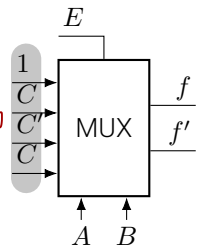
- For a 4-to-1 MUX the characteristic equation is $f = A'B'd_0 + A'Bd_1 + AB'd_2 + ABd_3$.
  - A and B are Boolean variables, applied at the select inputs, which can be factored out of any Boolean function of n variables.
  - The remaining $n - 2$ variables, referred to as the *residue variables*, can be formed into residue functions which can then be applied at the data inputs.

a small amount of something that remains after the main part has gone or been taken or used

- Example: $f(A, B, C) = \sum(0, 1, 3, 4, 7)$.



we group cells in which A & B are the same. and represents the input by C



Multiplexer as a Boolean function generator

- Example: $f(A, B, C, D) = \sum(0, 1, 3, 4, 5, 9, 10, 11, 14, 15)$.

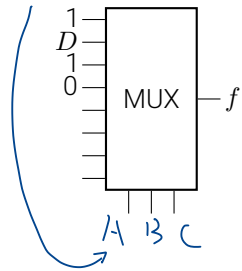
A	B	C	D	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
		...		

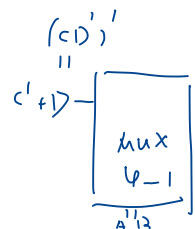
8-1 MUX : use 3 variable

⇑

4 variable

For function with n inputs, we use a 2^{n-1}-1 MUX, letting $n-1$ input be the selection lines.





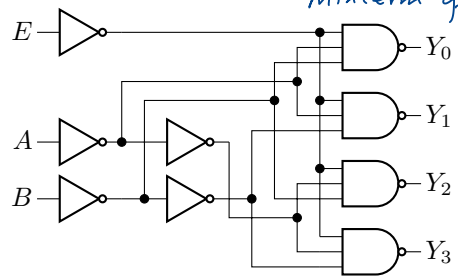
Decoder 译码器 inverse of multiplexer

- The basic function of an MSI decoder having n inputs is to select 1-out-of-2^n output lines.
  - The selected output is identified either by a 1, when all other outputs are 0, or by a 0 when all other outputs are 1.

minterm generator

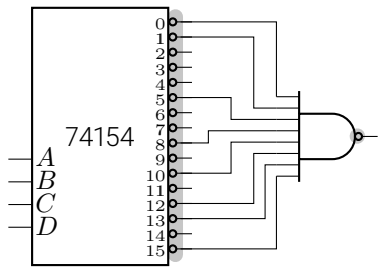
n: the length of code

2^n: the number of different informations that n codes can contain



Decoder

- It will be seen that the logic diagram of the basic decoder is identical to that of the basic demultiplexer.
  n. (多路)信号分离器,多路输出选择器
  - Provided the data line is used to enable the decoder.
- The decoder may also be regarded as a minterm generator. Each output generates one minterm.



INPUT						SELECTED OUTPUT(L)
$\overline{G1}$	$\overline{G2}$	D	C	B	A	
L	L	L	L	L	L	$\overline{Y}0$
L	L	L	L	L	H	$\overline{Y}1$
L	L	L	L	H	L	$\overline{Y}2$
L	L	L	L	H	H	$\overline{Y}3$
L	L	L	H	L	L	$\overline{Y}4$
L	L	L	H	L	H	$\overline{Y}5$
L	L	L	H	H	L	$\overline{Y}6$
L	L	L	H	H	H	$\overline{Y}7$
L	L	H	L	L	L	$\overline{Y}8$
L	L	H	L	L	H	$\overline{Y}9$
L	L	H	L	H	L	$\overline{Y}10$
L	L	H	L	H	H	$\overline{Y}11$
L	L	H	H	L	L	$\overline{Y}12$
L	L	H	H	L	H	$\overline{Y}13$
L	L	H	H	H	L	$\overline{Y}14$
L	L	H	H	H	H	$\overline{Y}15$
X	H	X	X	X	X	NONE
H	X	X	X	X	X	NONE

X : Don't care

Function Table

Inputs						Outputs															
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

H = High Level, L = Low Level, X = Don't Care

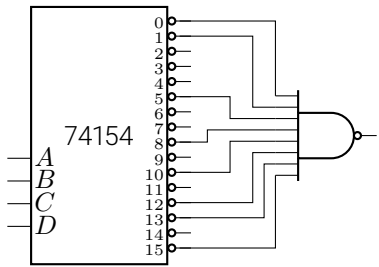
Decoder

- If $A = B = C = D = 0$ the output 0 of the decoder is active low while all other outputs are 1.
  - The decoder can generate the inverse of the 16 minterms.
  - Example: $f(A, B, C, D) = \sum(0, 1, 5, 8, 10, 12, 13, 15)$.

低电平有效；低电平；积极低；低准位有效；

101

111



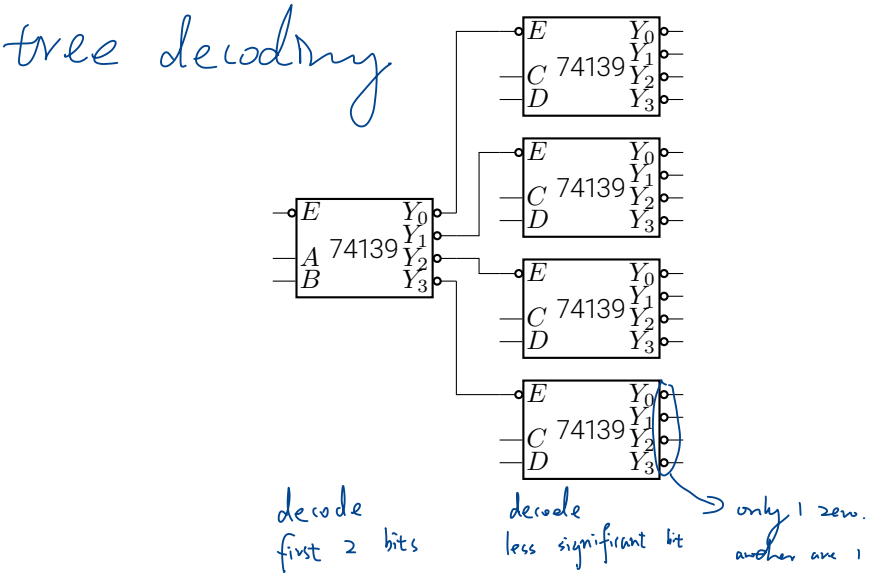
Decoder network

- When a large decoding network is required it cannot be implemented in a single MSI package.
  - Mainly because of the large number of pins needed.

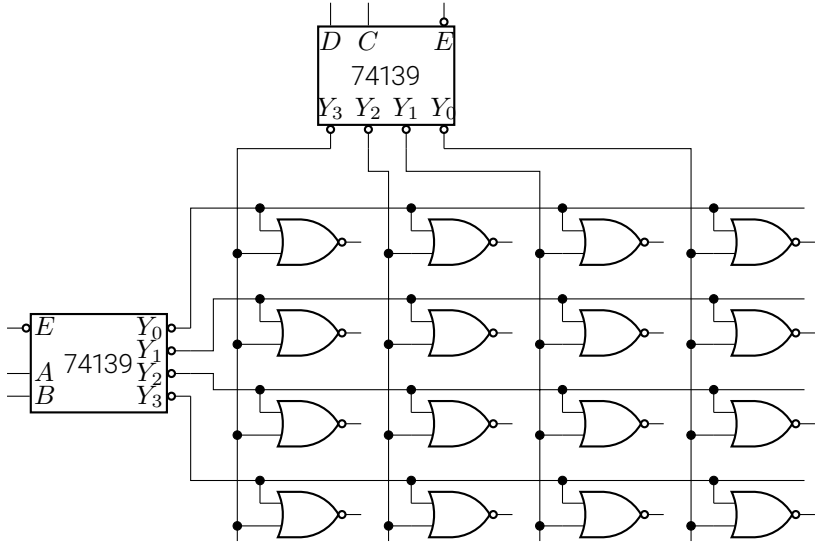
Medium-scale integration

- The decoding range can be extended by interconnecting decoder chips. Two schemes:
  - *Tree decoding*.
  - *Coincident decoding*.

Decoder network



Decoder network



Encoder

中文为编码器 不是译码器

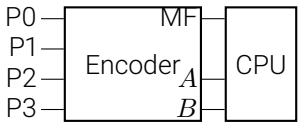
inverse of

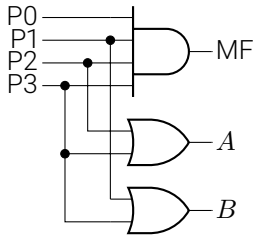
binary → one hot

encoder

multiplexer

- An encoder performs the inverse operation to that of a decoder.
- Example: CPU master flag





- The encoder is designed to identify one, and only one, of the peripherals at any given instant.

one hot

  - In practice, there is nothing to prevent two or more peripherals requesting service at the same time.
  - To deal with this situation a system of priorities can be attached to the peripheral flags.

In digital circuits and machine learning, a one-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0).[1] A similar implementation in which all bits are '1' except one '0' is sometimes called one-cold.[2]

Encoder

- Example: Octal-to-binary encoder.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$xyz = n$

$\Rightarrow D_n = 1$

- Eight inputs and three outputs connected with OR.

Encoder

- $x = D_4 + D_5 + D_6 + D_7$.
- $y = D_2 + D_3 + D_6 + D_7$.
- $z = D_1 + D_3 + D_5 + D_7$.
- Only one input can be active for one time.
  - D_3 and D_6 are 1 simultaneously: outputs $(111)_2 = 7$.
  - Encoder circuit must have priority: *Priority encoder*.

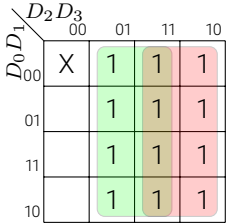
it is depends on the design NOT fixed

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

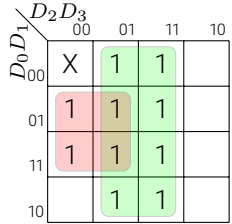
whether there is an input = 1

Encoder

$$x = D_2 + D_3$$



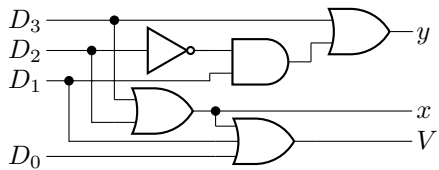
$$y = D_3 + D_1D_2'$$



Encoder

- $x = D_2 + D_3$.
- $y = D_3 + D_1 D_2'$.
- $V = D_0 + D_1 + D_2 + D_3$.

validity



Digital comparator

MSI circuit

just an introduction

- The usual problem for a comparator is the comparison of two multi-digit words such as $A = A_2A_1A_0$ and $B = B_2B_1B_0$.
  - Start from most to least significant bit.
  - $A = B$ if all bits are equal: $A_i = B_i$.
  - $x_i = A_iB_i + A_i'B_i'$.
- $A = B$ if $x_2x_1x_0 = 1$.
- $A > B$ if $A_2B_2' + x_2A_1B_1' + x_2x_1A_0B_0' = 1$.
- $A < B$ if $A_2'B_2 + x_2A_1'B_1 + x_2x_1A_0'B_0 = 1$.

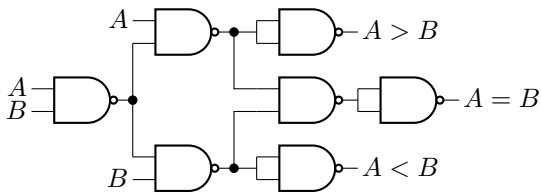
eg. 1 2 3 ⇒ 1 1 0

1 3 4 ⇒ 1 0 1

compare them from MSB to LSB

Digital comparator

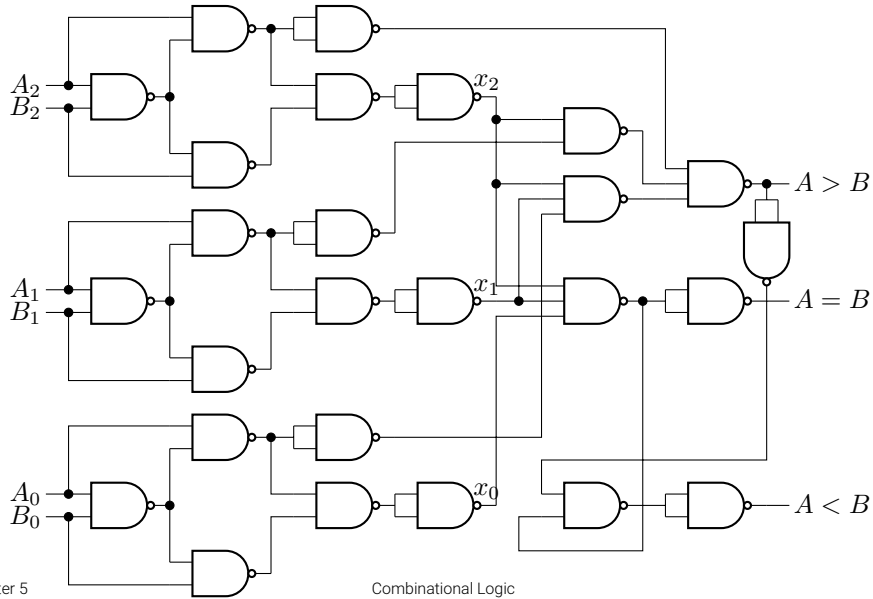
- The implementation of a 3-bit comparator is based on a single bit comparator.



A=B=0/1

- Using the equations developed in the last page, we can have a 3-bit comparator.

Digital comparator

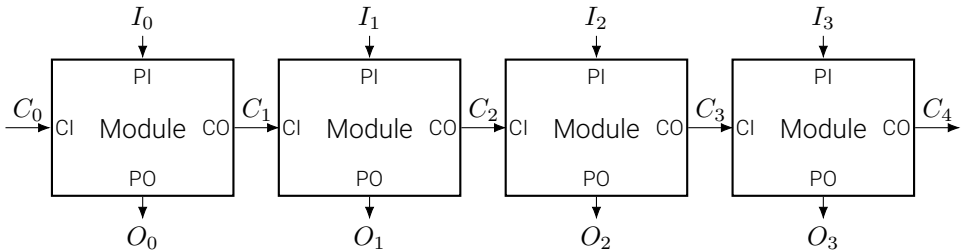


compare 1 bit iterative compare numbers

Iterative combinational circuits

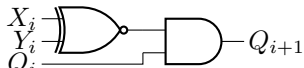
Later NOT in Exam

- General structure: n identical modules for the same functionality.
- For problems that can be solved by an iterative algorithm:
  - Set C_0 to its initial value and set i to 0.
  - While $i < n$, repeat:
    - Use C_i and D_i to determine the values of O_i and C_{i+1}.
    - Increase i.
- Primary inputs/outputs and cascading inputs/outputs.



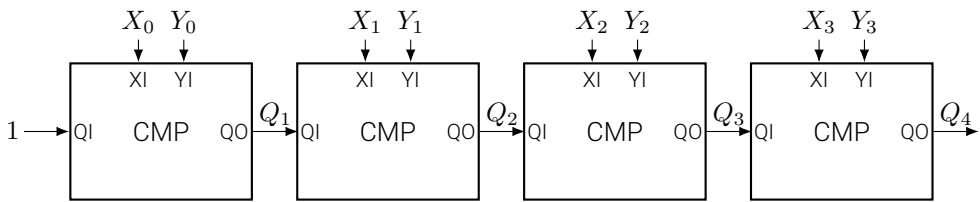
Iterative comparator circuits

- Complete circuit for comparing whether two n-bit values X and Y are the same:
  - Set Q_0 to 1 and set i to 0.
  - While $i < n$, repeat:
    - If Q_i is 1 and X_i equals Y_i, set Q_{i+1} to 1.
    - Increase i.



$$Q_{i+1} = (X_i \oplus Y_i)' Q_i$$

- Module: one-bit comparator.



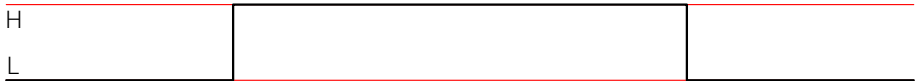
NOT in exam

Gate delays

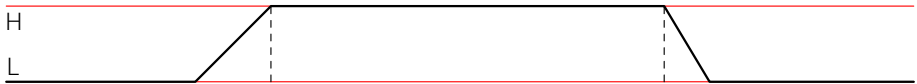
- When the input to a logic gate is changed, the output will not change immediately.
- The switching elements within a gate take a finite time to react to a change (transition) in input.
- As a result the change in the gate output is delayed w.r.t. to the input change.
- Such delay is called the propagation delay of the logic gate.
  - The propagation delay for a 0-to-1 output change may be different from the delay for a 1-to-0 change.

Gate delays

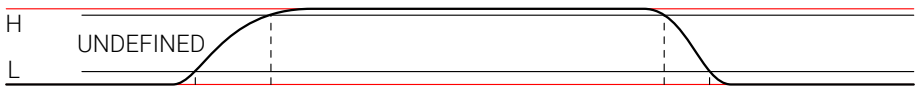
- Ideal case of zero-time switching:



- A more realistic approximation



- Actual timing for rise and fall times:



Effect of gate delays

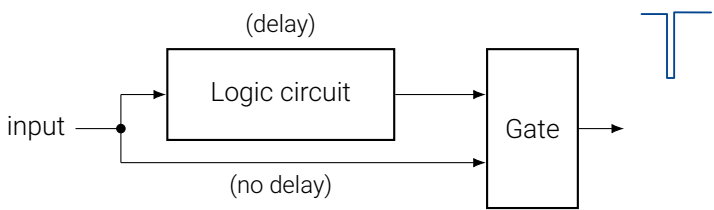
- The analysis of a combinational circuit ignoring delays can predict only its steady-state behavior.
  - Predicts a circuit's output as a function of its inputs assuming that the inputs have been stable for a long time, relative to the delays in the circuit's electronics.
- Because of circuit delays, the transient behavior of a combinational logic circuit may differ from what is predicted by steady-state analysis.
- *Timing hazard*: a circuit's output may produce a short pulse (*glitch*) at a time when steady state analysis predicts that the output should not change.

Effect of gate delays

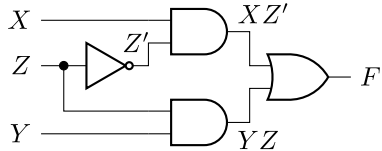
- The analysis of a combinational circuit ignoring delays can predict only its *steady-state behavior*.
  - Predicts a circuit's output as a function of its inputs assuming that the inputs have been stable for a long time, relative to the delays in the circuit's electronics.
- Because of circuit delays, the *transient behavior* of a combinational logic circuit may differ from what is predicted by steady-state analysis.
- *Timing hazard*: a circuit's output may produce a short pulse (*glitch*) at a time when steady state analysis predicts that the output should not change.

Timing hazard

- A gate has measurable response time t_{pLH} and t_{pHL}. Around 10ns per gate.
- Delays through transmission gates can add up and introduce timing hazards.
- t_{pLH} = low-to-high, t_{pHL} = high-to-low propagation times
- *static-1 hazard* is a short **0** glitch when for a changed input, we expect (by logic theorems) the output to remain constant **1**.
- *static-0 hazard* is a short **1** glitch when we expect the output to remain constant **0**.

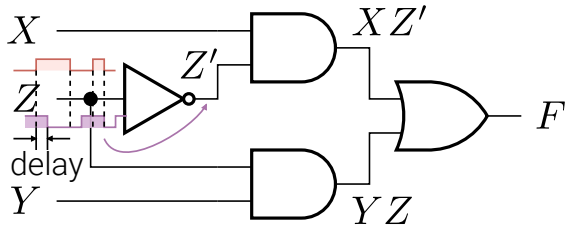


Circuit with a static-1 hazard



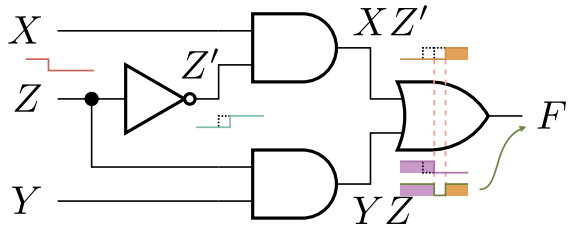
- Assume $XYZ = 111$, consider a transition to $XYZ = 110$.
- What we logically expect:
  - $F = XZ' + YZ$
- Before: $F = 1$, after: $F = 1$.
- **The output does not change!**

Circuit with a static-1 hazard



- But real world gates introduce delays.
- Input signal of each gate is shifted in the output by a constant delay.

Circuit with a static-1 hazard



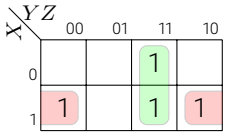
- Change occurs at input Z and propagates to output F along two paths with different delays.

Timing hazards and Karnaugh maps

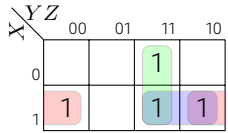
- Recall that in sum-of-products (AND-OR) circuits, AND gates correspond to prime implicants of the Karnaugh map.
- A potential hazard exists wherever two adjacent 1-cells in a Karnaugh map are not covered by a single product term (prime implicant).
- A hazard occurs when there is a transition between adjacent prime implicants.
- To eliminate hazards, find a cover in which all adjacent 1-cells are covered by a prime implicant.
  - Define *consensus* prime implicants, as product terms not covered in F.
  - Therefore, include an extra product term to cover the hazardous input combination

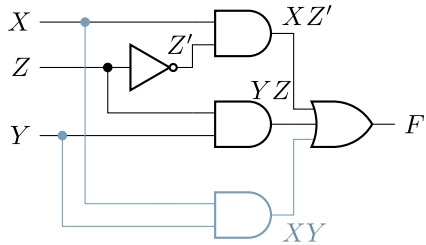
Eliminating the timing hazard

Minimal cost: $F = XZ' + YZ$



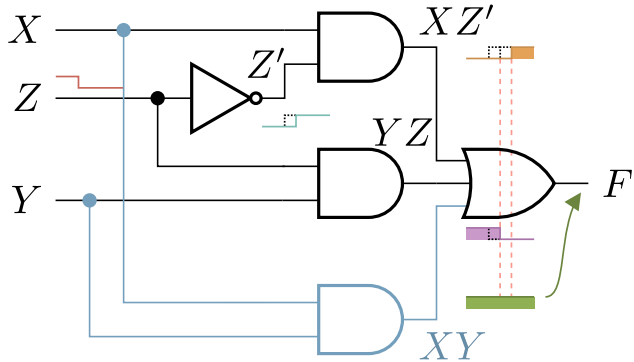
Eliminate hazard: $F = XZ' + YZ + XY$





Eliminating the timing hazard

- So, how does this help?



- Unexpected glitch is eliminated!
  - A change in input variable Z causes a transition between two adjacent 1-cells but now these 1-cells are included in the product term XY.

Dynamic hazards

- Dynamic hazard: Output signal is supposed to change 1-0 or 0-1 but a short oscillation occurs before the output settles to its new logic value.
- Occurs if there are multiple paths with different delays from the changing input to the changing output.
- In practice many input variables, multiple outputs; solved by computer programs.

EDA tool can automatically eliminate it