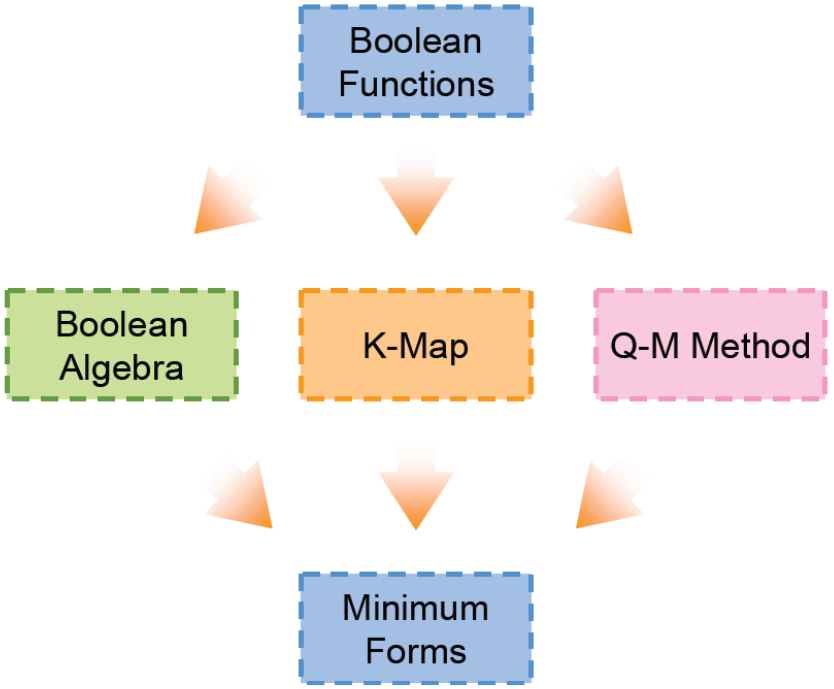
EE2000 Logic Circuit Design

Lecture 3 – Combinational System Design



What will you learn?

3.1 Learn Binary Coded Decimals

3.2 Learn the design procedure of combinational system with examples

3.3 Identify a timing hazard of a combinational system and learn to solve the problem

3.4 Learn various schemes of error detection and correction for binary data transmission

3.1 Binary Coding

- Computers use **Binary Numbers** to perform arithmetic calculations as well as to store information
- A set of n-bit strings in which different bit strings represent different numbers or info is called a code

  At least 4 bits are needed to represent 10 decimal digits

- A particular combination of n-bit values is called a code word
- To develop code for human-computer communications

  Alphanumeric: ASCII Code (American Standard Code for Information Interchange)

  Digits: Binary Coded Decimal (Binary encoding of decimal numbers)

ASCII Code

ASCII Code	Value	
000 0000	NULL	control signals
…	…	
010 0000	Space	symbols
010 0001	! (exclamation mark)	
010 0010	" (double quote)	
…	…	
011 0000	0	numeric characters
011 0001	1	
…	…	
011 1010	: (colon)	symbols
…	…	
100 0001	A	capital letters
…	…	
101 1010	Z	
…	…	symbols
110 0001	a	small letters
…	…	
111 1010	z	
…	…	symbols

The word *Logic* would be coded as:

100 1100 110 1111
L o

110 0111 110 1001 110 0011
g i c

739 would be coded as:

011 0111 011 0011 011 1001
7 3 9

(Please refer to the complete ASCII table in your book)

Binary Coded Decimals

Decimal digit	BCD 8421	2421	Excess-3	Biquinary	1-out-of-10
0	0000	0000	0011	0100001	1000000000
1	0001	0001	0100	0100010	0100000000
2	0010	0010	0101	0100100	0010000000
3	0011	0011	0110	0101000	0001000000
4	0100	0100	0111	0110000	0000100000
5	0101	1011	1000	1000001	0000010000
6	0110	1100	1001	1000010	0000001000
7	0111	1101	1010	1000100	0000000100
8	1000	1110	1011	1001000	0000000010
9	1001	1111	1100	1010000	0000000001
		Unused code words			
	1010	0101	0000	0000000	0000000000
	1011	0110	0001	0000001	0000000011
	1100	0111	0010	0000010	0000000101
	1101	1000	1101	0000011	0000000110
	1110	1001	1110	0000101	0000000111
	1111	1010	1111	...	...

8421 Code

- 4 bits to represent a decimal digit
- Weighted code: (8, 4, 2, 1) are the weights of each bits
- Code: 0110
- Value: (8 × 0) + (4 × 1) + (2 × 1) + (1 × 0) = 6
- Example: 739 would be stored as

  0111 0011 1001

- BCD ≠ Binary conversion
  Binary conversion of $739_{10} = 1011100011_2$

Value (decimal digit)	8421 code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
unused	1010
	1011
	1100
	1101
	1110
	1111

Other Weighted Codes

Value (decimal digit)	8421 code	5421 code	2421 code
0	0000	0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	1000	1011
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111
unused	1010	0101	0101
	1011	0110	0110
	1100	0111	0111
	1101	1101	1000
	1110	1110	1001
	1111	1111	1010

The code are the same for the first 5 digits

(5 x 1) + (4 x 0) + (2 x 1) + (1 x 1)
= 5 + 2 + 1
= 8

(2 x 1) + (4 x 1) + (2 x 1) + (1 x 0)
= 2 + 4 + 2
= 8

4-bit codes have 16 combinations, but we just use 10 of them. The remaining 6 codes are unused and undefined

Properties of 2421 Code

A self-complementing code (9's complement)

- The complement of 0 is 9 (0000 $\Leftrightarrow$ 1111)
- The complement of 1 is 8 (0001 $\Leftrightarrow$ 1110)
- The complement of 2 is 7 (0010 $\Leftrightarrow$ 1101)
- The complement of 3 is 6 (0011 $\Leftrightarrow$ 1100)
- The complement of 4 is 5 (0100 $\Leftrightarrow$ 1011)

Unweighted Code: Excess 3 Code (XS3)

Value (decimal digit)	8421code	Excess 3 code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100
unused	1010	0000
	1011	0001
	1100	0010
	1101	1101
	1110	1110
	1111	1111

Excess 3 code is a shifted 8421 code

XS3's 0 = 8421 code's 3

XS3's 1 = 8421 code's 4

…

9's complement code:
Complement of 0 is 9 (0011 ↔ 1100)
Complement of 1 is 8 (0100 ↔ 1011)
…
Complement of 4 is 5 (0111 ↔ 1000)

Evenly distribution of 1s and 0s

8421 code: the MSB has eight 0s, only two 1s

XS3: each bit has exactly five 1s and five 0s

Gray Code

2-bit gray code	3-bit gray code	4-bit gray code	decimal
00	000	0000	0
01	001	0001	1
11	011	0011	2
10	010	0010	3
	110	0110	4
	111	0111	5
	101	0101	6
	100	0100	7
		1100	8
		1101	9
		1111	10
		1110	11
		1010	12
		1011	13
		1001	14
		1000	15

- Designed by Frank gray to prevent spurious output from mechanical switches (which can only switch one bit at a time)
- One bit difference in the next or adjacent code word independent of the direction taken in the code

Gray Code *vs* Binary Code

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

- Consider a 4-bit digital counter
- In binary code, when change from 3 to 4, 3 bit change

0011 -> 0111 -> 0101 -> 0100

- In Gray code, only 1 bit change

0010 -> 0110

No fake/false intermediate output

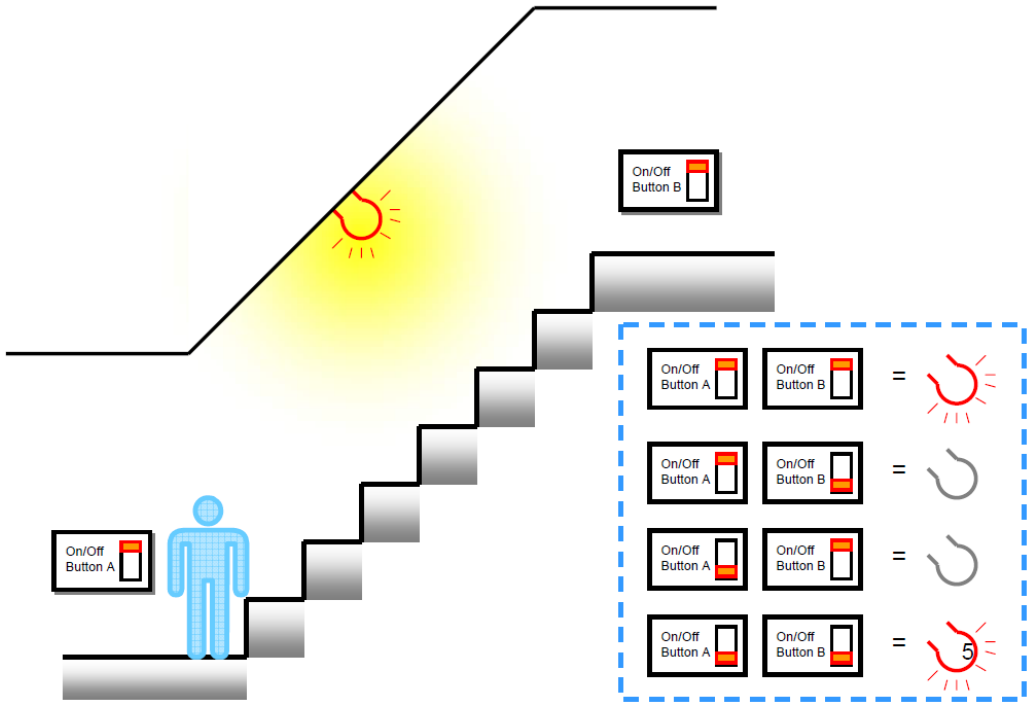
Summary

- Weighted codes
  - 8421, 5421, 2421 codes
- Non-weight codes
  - XS3
- Self-complement
  - 2421, XS3 codes
- Gray code – 1-bit different
- Tables will be given in the test/exam if needed

3.2 Design Procedure

1. State the problem/specification of the design
2. Determine the number of input variables and output variables
3. Formulate truth tables/Boolean functions between inputs and outputs
4. Simplification/minimization of the logic functions
5. Design and draw the logic circuit diagram

Bi-Switch Lighting Controller



State the case

Design a circuit to control the bulb

- The light turns on when both buttons are turned UP / DOWN.
- The light turns off when both buttons swap in different positions.
- ON / OFF light is a binary decision output.
- Button positions are the inputs (variables)

Formulation

Define:
Two variables ***A*** and ***B*** are the button positions.
F is binary decision output of ***A*** and ***B***.

0: button at the UP position.
1: button at the DOWN position.

0: light OFF
1: light ON

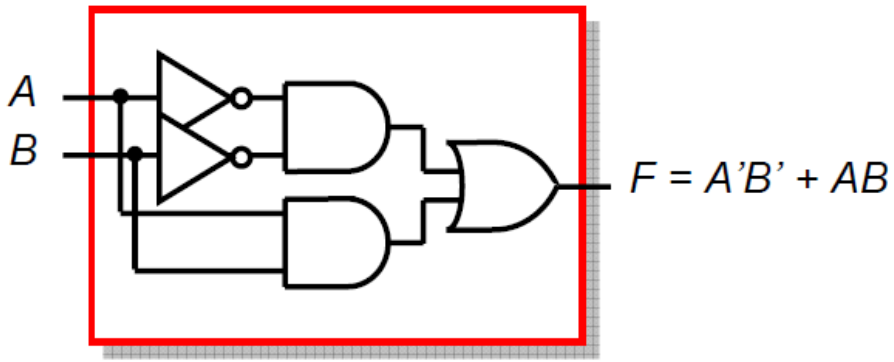
Inputs		Output
A	***B***	***F***
0	0	1
0	1	0
1	0	0
1	1	1

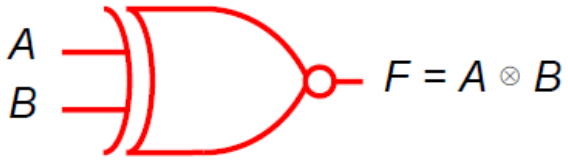
$F(A, B)$ is 1 if (A = 0 AND B = 0) OR (A = 1 AND B = 1)

i.e. $F(A, B) = A'B' + AB = \Sigma m(0, 3)$

- Optimization:
  - From the truth table,
  - $F(A, B) = AB + A'B' (= A \otimes B)$

Logic Circuit Diagram





Exercise

A security gate system has a binary output *Z* that is used to control the gate "open and close" where *Z* = 1 for the gate open. The gate will close when the system detects the first two displayed numbers from a user's ID card {00, 01, 02, …, 15} with the number 06, 07, 11, 12, 13, 14, or 15. The variables *a*, *b*, *c*, and *d* determine a 4-digit binary code to represent the first two decimal numbers shown on the ID card.

Determine the truth table for the above security gate system, and the minimum SOP and POS expressions.

State the case

Exercise

$f(a, b, c, d) = Z = 0$ when ID numbers 06, 07, 11, 12, 13, 14 or 15

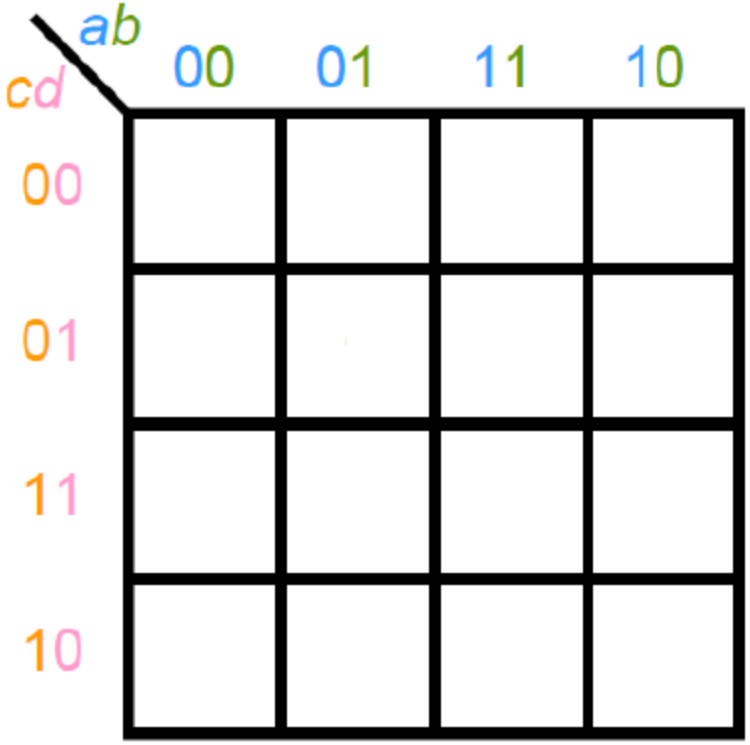
Else $Z = 1$

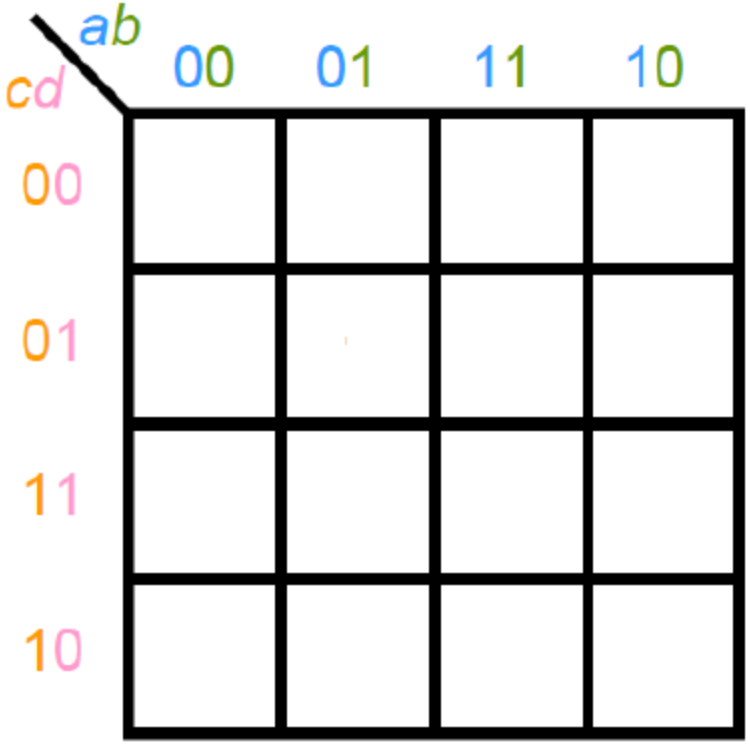
Decimal numbers	Inputs				Output
	a	b	c	d	Z
00					
01					
02					
03					
04					
05					
06					
07					
08					
09					
10					
11					
12					
13					
14					
15					

Exercise

Express the function in SOP and POS forms using K-map approach.

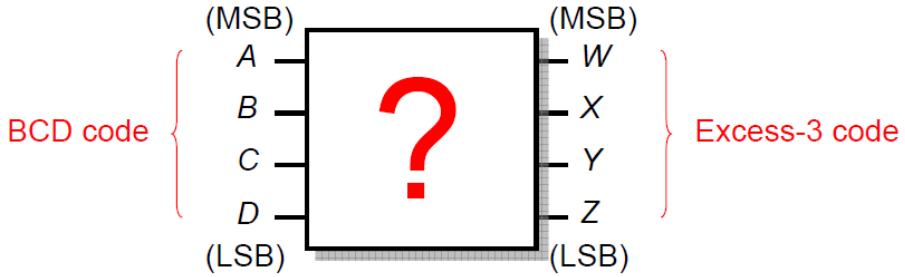
cd \ ab	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}





Code Converter

- Design a logic circuit that perform code conversion



- Input is BCD 8421 code
- Output is Excess-3 code

*Using only Two-input Gates and NOT Gates.

State the case

Design a circuit to convert the BCD 8421 to the Excess-3 code

- *A, B, C, D* are the input of BCD.
- *W, X, Y, Z* are the output of Excess-3 code.
- The output functions are:

  W (*A, B, C, D*)

  X (*A, B, C, D*)

  Y (*A, B, C, D*)

  Z (*A, B, C, D*)

Formulation

Decimal digit	Input (8421 code)				Minterm	Output (Excess-3 code)			
	A	*B*	*C*	*D*		*W*	*X*	*Y*	*Z*
0	0	0	0	0	m_0	0	0	1	1
1	0	0	0	1	m_1	0	1	0	0
2	0	0	1	0	m_2	0	1	0	1
3	0	0	1	1	m_3	0	1	1	0
4	0	1	0	0	m_4	0	1	1	1
5	0	1	0	1	m_5	1	0	0	0
6	0	1	1	0	m_6	1	0	0	1
7	0	1	1	1	m_7	1	0	1	0
8	1	0	0	0	m_8	1	0	1	1
9	1	0	0	1	m_9	1	1	0	0
Unused	X	X	X	X	m_{10}	X	X	X	X
Unused	X	X	X	X	m_{11}	X	X	X	X
Unused	X	X	X	X	m_{12}	X	X	X	X
Unused	X	X	X	X	m_{13}	X	X	X	X
Unused	X	X	X	X	m_{14}	X	X	X	X
Unused	X	X	X	X	m_{15}	X	X	X	X

Unused outputs can consider as DON'T CARE.

Formulation

$$W(A,B,C,D) = \sum m(5,6,7,8,9) + \sum d(10,11,12,13,14,15)$$

$$X(A,B,C,D) = \sum m(1,2,3,4,9) + \sum d(10,11,12,13,14,15)$$

$$Y(A,B,C,D) = \sum m(0,3,4,7,8) + \sum d(10,11,12,13,14,15)$$

$$Z(A,B,C,D) = \sum m(0,2,4,6,8) + \sum d(10,11,12,13,14,15)$$

K-maps

K-map for *W*:

CD \ AB	00	01	11	10
00			x	1
01		1	x	1
11		1	x	x
10		1	x	x

K-map for *X*:

CD \ AB	00	01	11	10
00		1	x	
01	1		x	1
11	1		x	x
10	1		x	x

K-map for *Y*:

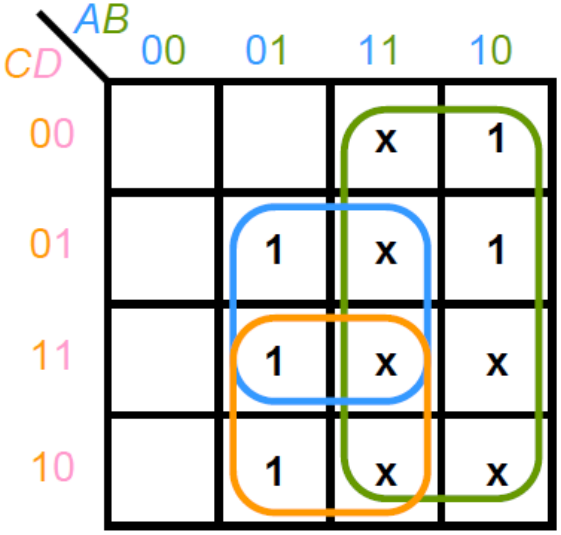
CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11	1	1	x	x
10			x	x

K-map for *Z*:

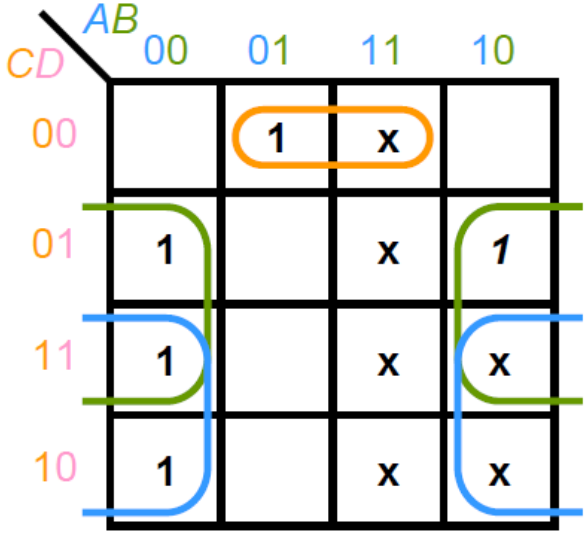
CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11			x	x
10	1	1	x	x

Simplification

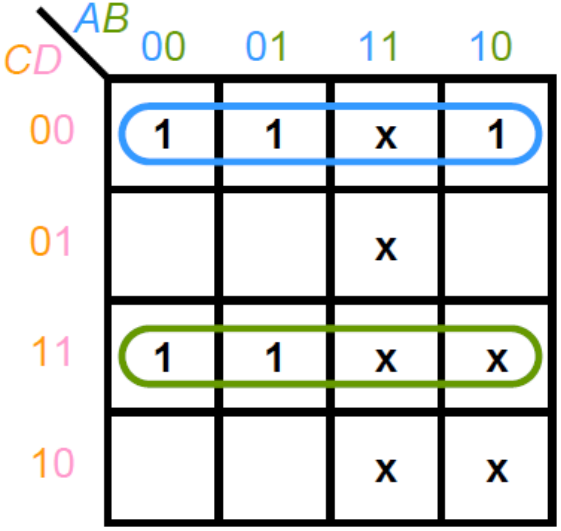
$W = A + BC + BD$



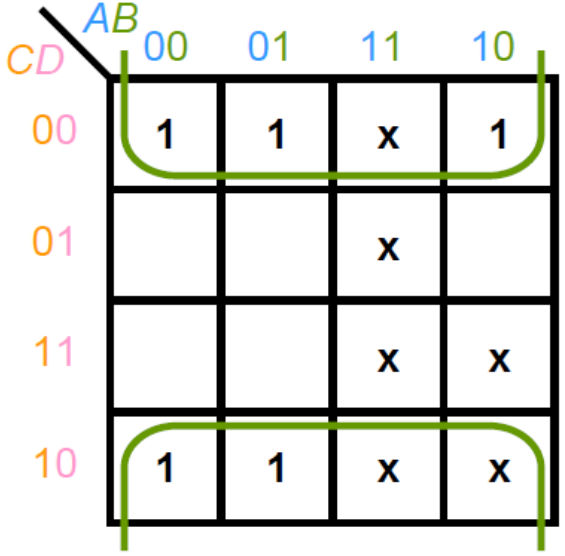
$X = B'C + B'D + BC'D'$



$Y = CD + C'D'$



$Z = D'$



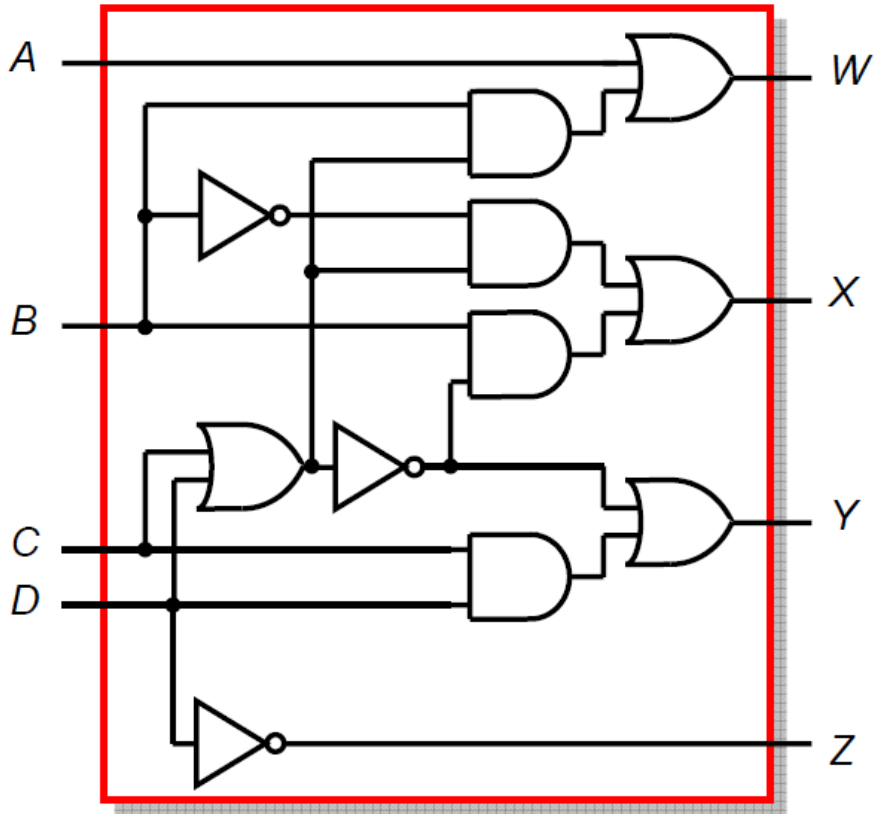
Logic Circuit

$$W = A + BC + BD = A + B(C + D)$$

$$X = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$$

$$Y = CD + C'D' = CD + (C + D)'$$

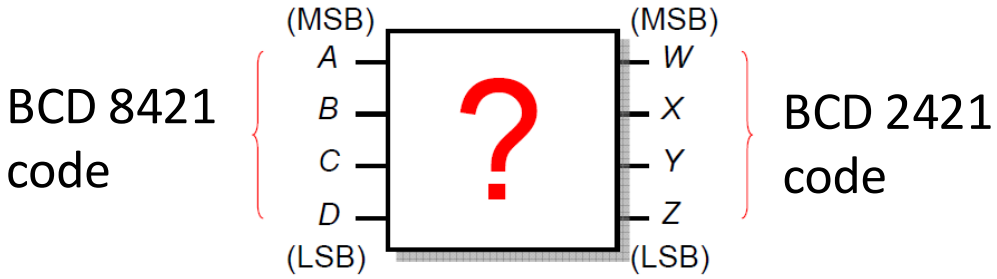
$$Z = D'$$



*Using only Two-input Gates and NOT Gates.

Exercise

- Design a logic circuit that perform code conversion



- Input is BCD 8421 code
- Output is BCD 2421 code

*Using only Two-input Gates and NOT Gates.

State the case

Design a circuit to convert the BCD 2421 to the BCD 8421 code

- *A, B, C, D* are the input.
- *W, X, Y, Z* are the output.
- The output functions are:

  W (*A, B, C, D*)
  X (*A, B, C, D*)
  Y (*A, B, C, D*)
  Z (*A, B, C, D*)

Formulation

Decimal numbers	Minterms	Inputs (2421)				Outputs (8421)			
		A	B	C	D	W	X	Y	Z
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
Unused									
Unused									
Unused									
Unused									
Unused									
Unused									

Simplification

cd \ ab	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

$W(A,B,C,D)$

CD \ AB	00	01	11	10
00				
01				
11				
10				

$X(A,B,C,D)$

CD \ AB	00	01	11	10
00				
01				
11				
10				

$Y(A,B,C,D)$

CD \ AB	00	01	11	10
00				
01				
11				
10				

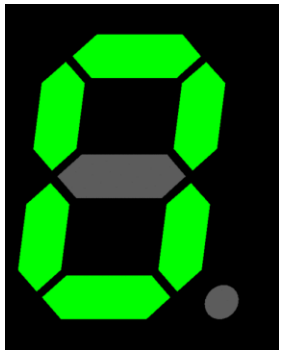
$Z(A,B,C,D)$

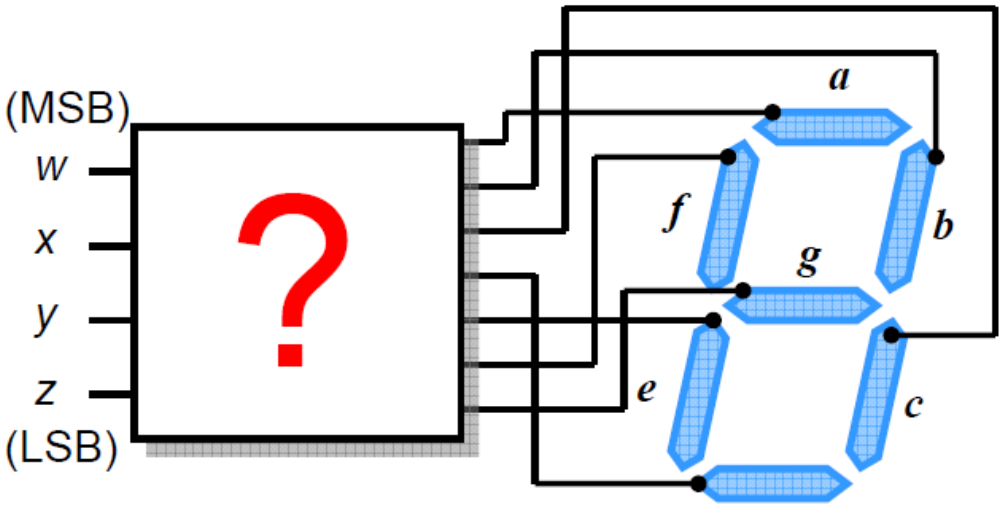
CD \ AB	00	01	11	10
00				
01				
11				
10				

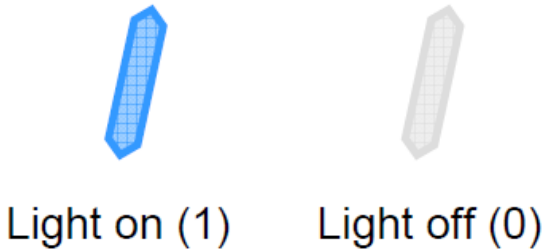
Logic Circuit

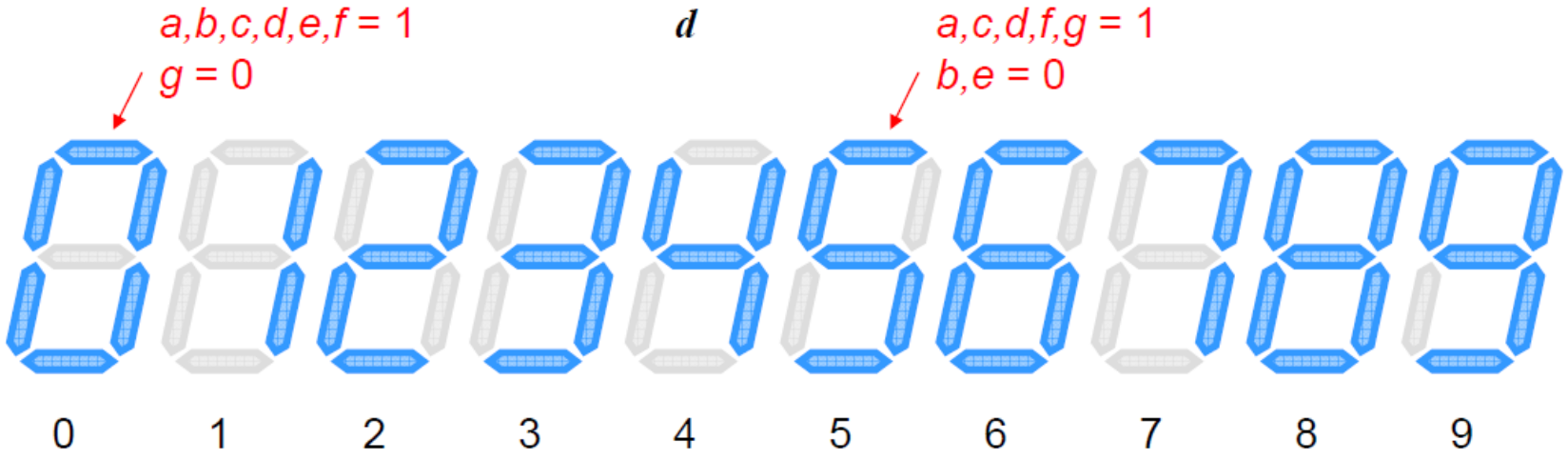
*Using only Two-input Gates and NOT Gates.

7-Segment Display

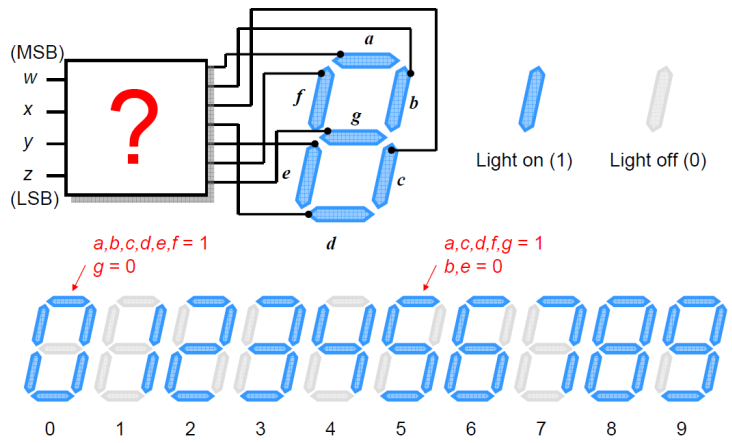








Formulation



State the case

Design a circuit to convert the BCD 8421 to the 7-segment display

- *w, x, y, z* are the input.
- *a, b, c, d, e, f, g* are the output.

numbers	Inputs				7-segment display						
	w	*x*	*y*	*z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	x	x	x	x	x	x	x
11	1	0	1	1	x	x	x	x	x	x	x
12	1	1	0	0	x	x	x	x	x	x	x
13	1	1	0	1	x	x	x	x	x	x	x
14	1	1	1	0	x	x	x	x	x	x	x
15	1	1	1	1	x	x	x	x	x	x	x

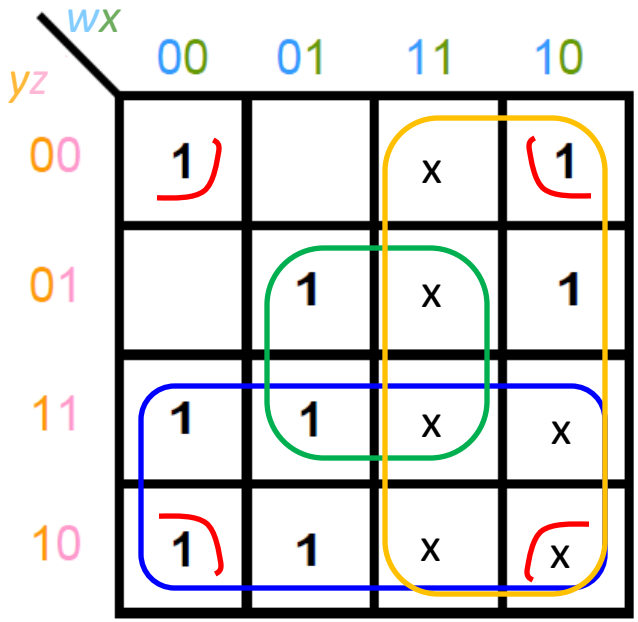
$$a\,(w,x,y,z) =$$

$$\Sigma\, m\,(0,2,3,5,6,7,8,9) + \Sigma d(10,11,12,13,14,15)$$

K-map for segment ‘a’

cd \ ab	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

$$a\,(w,x,y,z) = \Sigma m(0,2,3,5,6,7,8,9) + \Sigma d(10,11,12,13,14,15)$$



$$a(w,x,y,z) = w + y + xz + x'z'$$

K-map for segment ‘c’

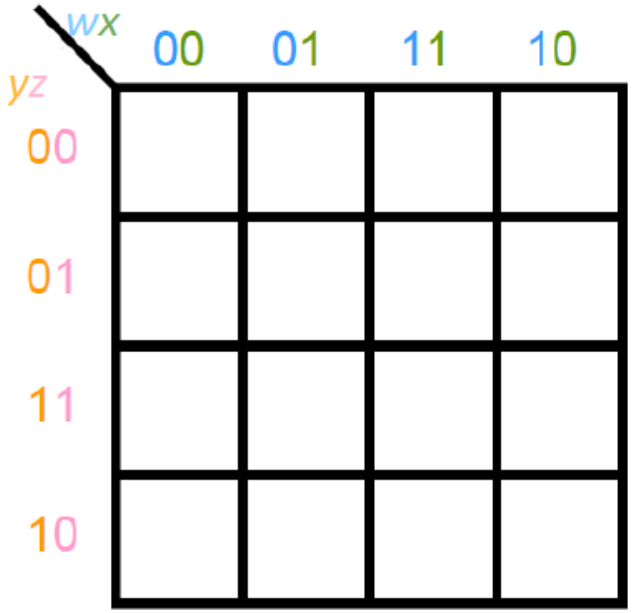
$$c\,(w,x,y,z) = \Sigma m(0,1,3,4,5,6,7,8,9) + \Sigma d(10,11,12,13,14,15)$$

cd \ ab	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

yz \ wx	00	01	11	10
00				
01				
11				
10				

K-map for segment ‘e’

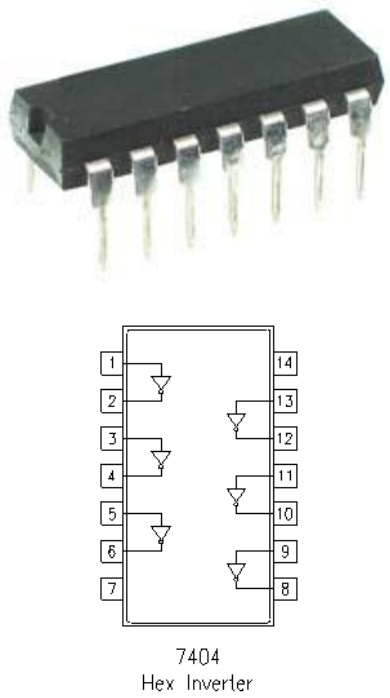
$$e\,(w,x,y,z) = \Sigma m(0,2,6,8) + \Sigma d(10,11,12,13,14,15)$$

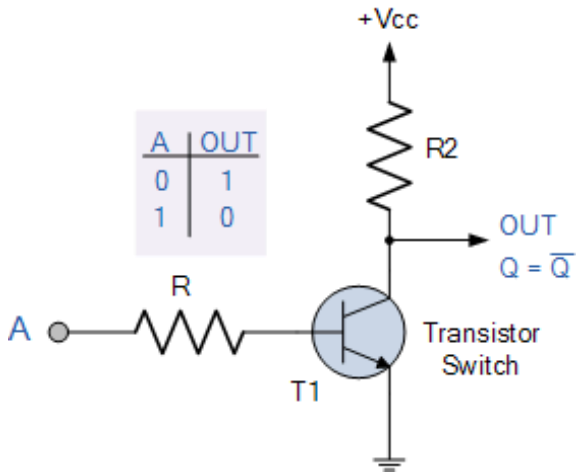


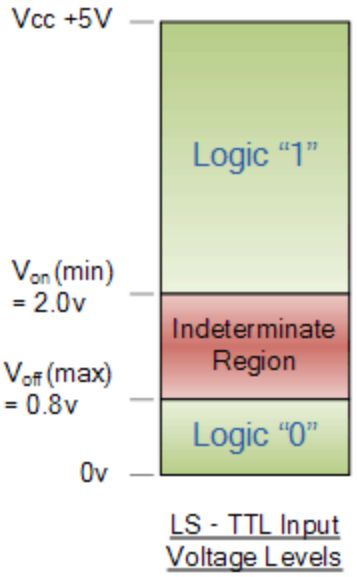
cd \ ab	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

3.3 Timing Hazard

Logic devices (gates or other more complex circuits) are essentially made from semi-conductor



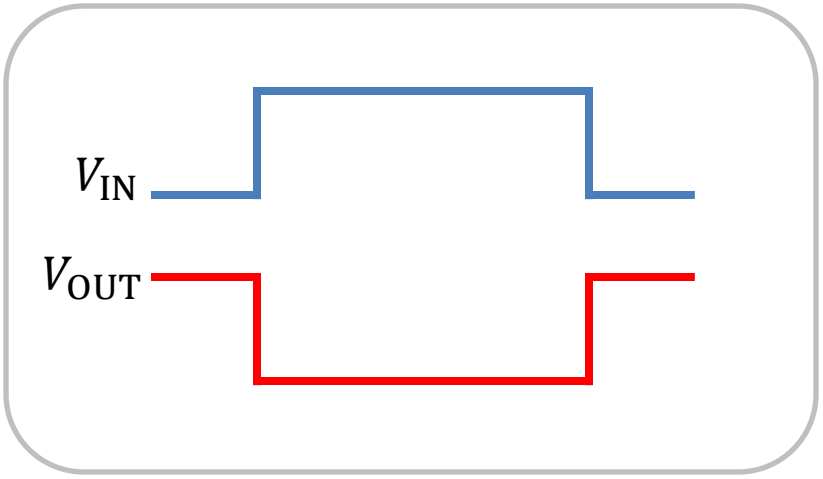




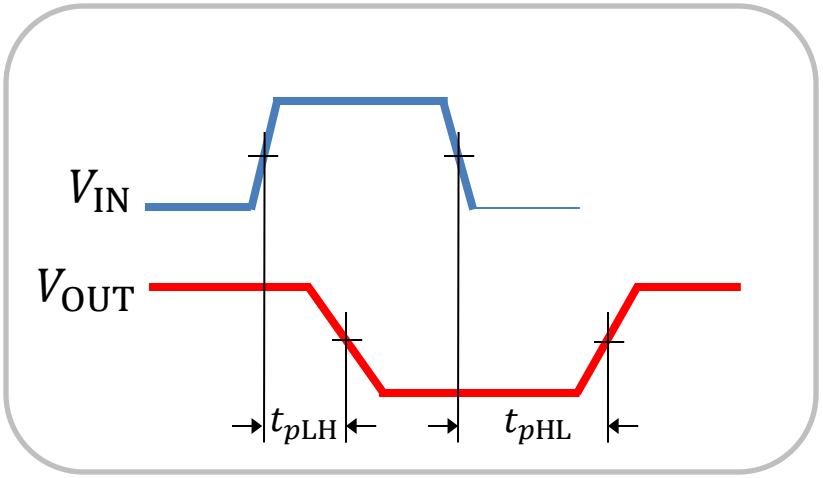
Propagation Delay

- "Real" input and output voltages are not a perfect step function
- Practical logic input and output waveforms exhibit "delay" nature
- Propagation delay of the logic gate (t_p) in ns
- Delay for a 0-to-1 output change ($t_{p\mathrm{LH}}$) might be different from delay for a 1-to-0 change ($t_{p\mathrm{HL}}$)

Ideally



In Reality



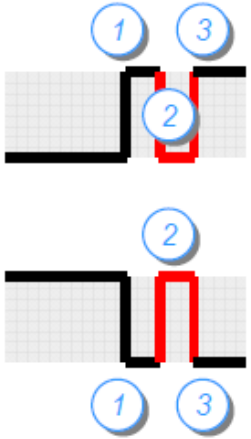
Timing Hazards

- In previous lectures, we only considered the steady-state behavior of a logic circuit
  - Predict the output as a function of its inputs assuming that the inputs have been stable for a long time, relative to the propagation delays
- Due to propagation delay, the transient behavior of the logic circuit might be differed from steady-state analysis
- A circuit's output may produce a glitch (short pulse) at a time when the steady-state analysis predicted that the output should not change
- A timing hazard exists when a circuit has the possibility of producing a glitch

Timing Hazards

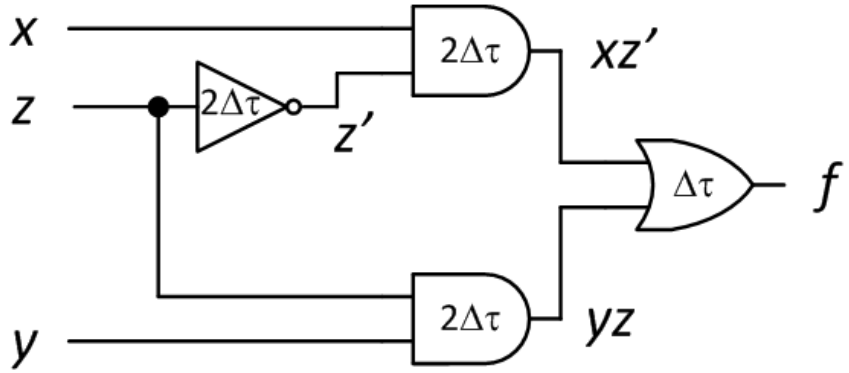
Static-0 hazard: the output may momentarily go to 1 when it should *remain 0*

Static-1 hazard: the output may momentarily go to 0 when it should *remain 1*



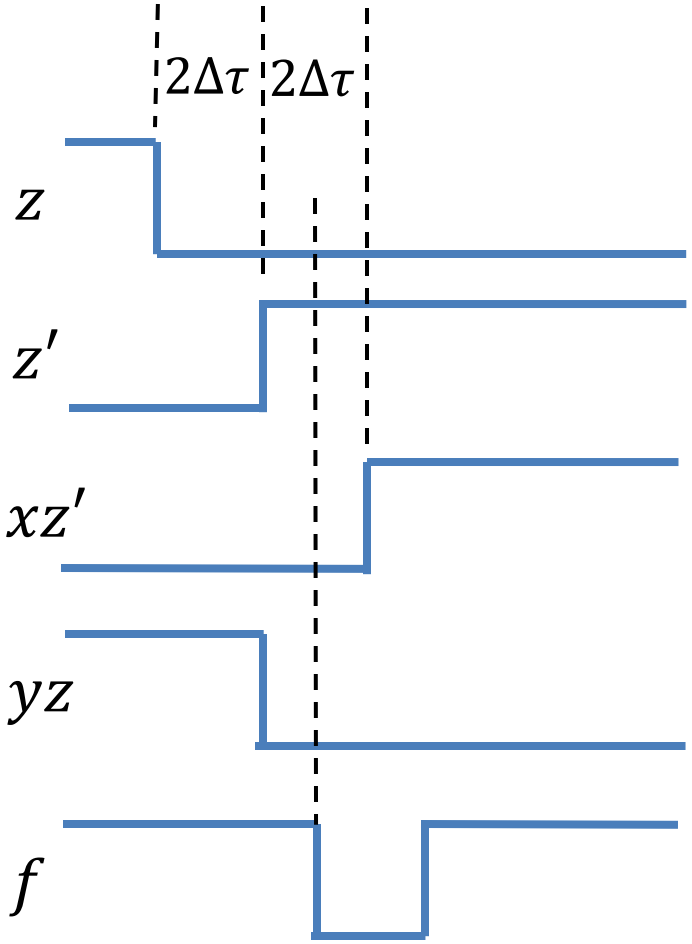
Dynamic hazard: The output changes three or more times when it should change from 1 to 0 or from 0 to 1 *only once*

Example



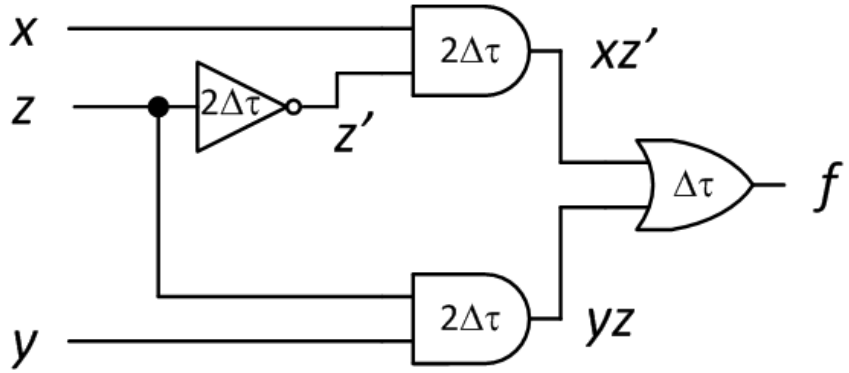
Assume that the propagation delay of each gate are as shown above.

Work out the timing diagram to identify the presence of any timing hazard when the input condition changes from $(x, y, z) = (1,1,1)$ to $(1,1,0)$.



Static-1 hazard!!!

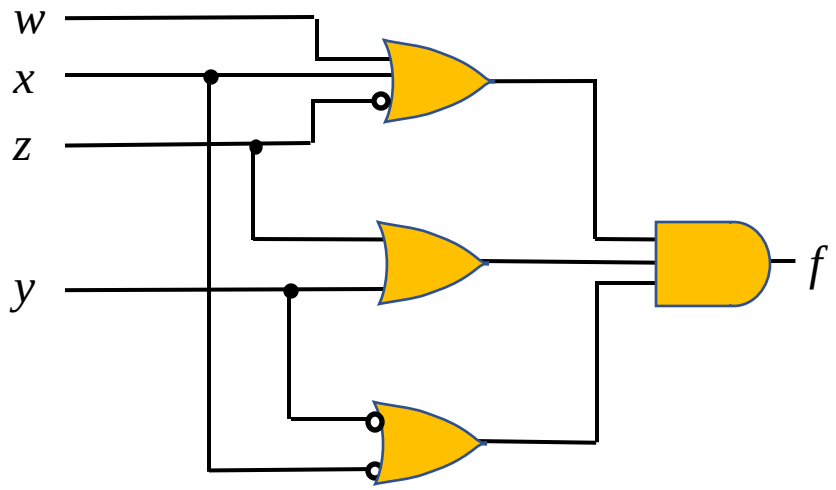
Exercise



Assume that the propagation delay of each gate are as shown above.

Work out the timing diagram to identify the presence of any timing hazard when the input condition changes from $(x, y, z) = (1,1,0)$ to $(1,1,1)$.

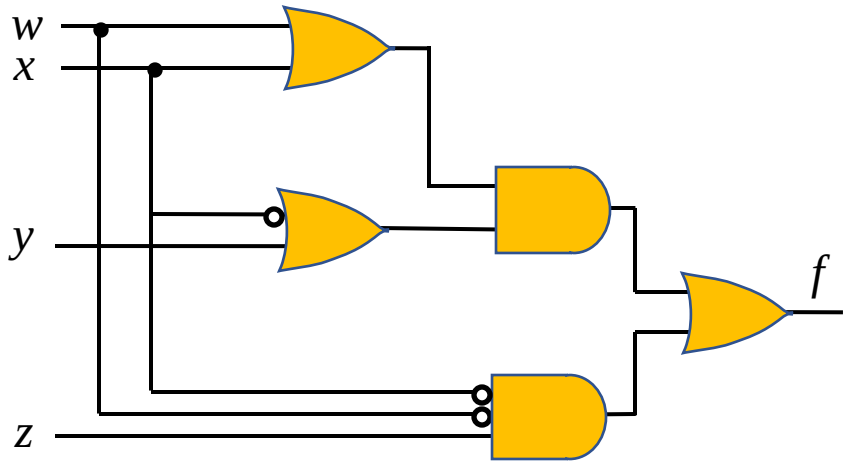
Exercise



Assume that the propagation delay of all gates is $\Delta\tau$.

Work out the timing diagram to identify the presence of any timing hazard when the input condition changes from $(w, x, y, z) = (0,0,0,0)$ to $(0,0,0,1)$.

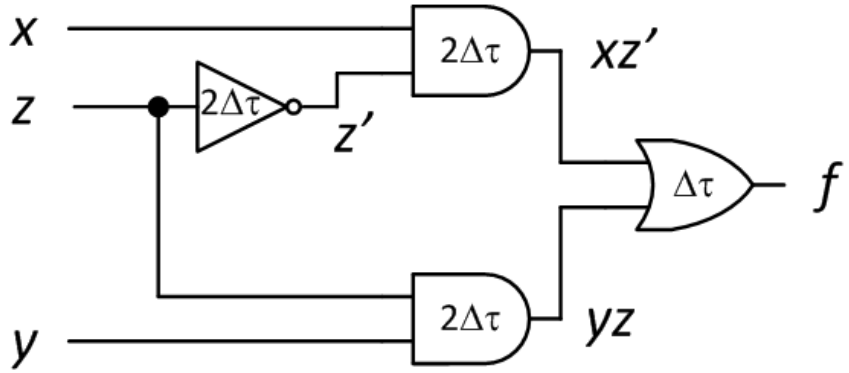
Exercise

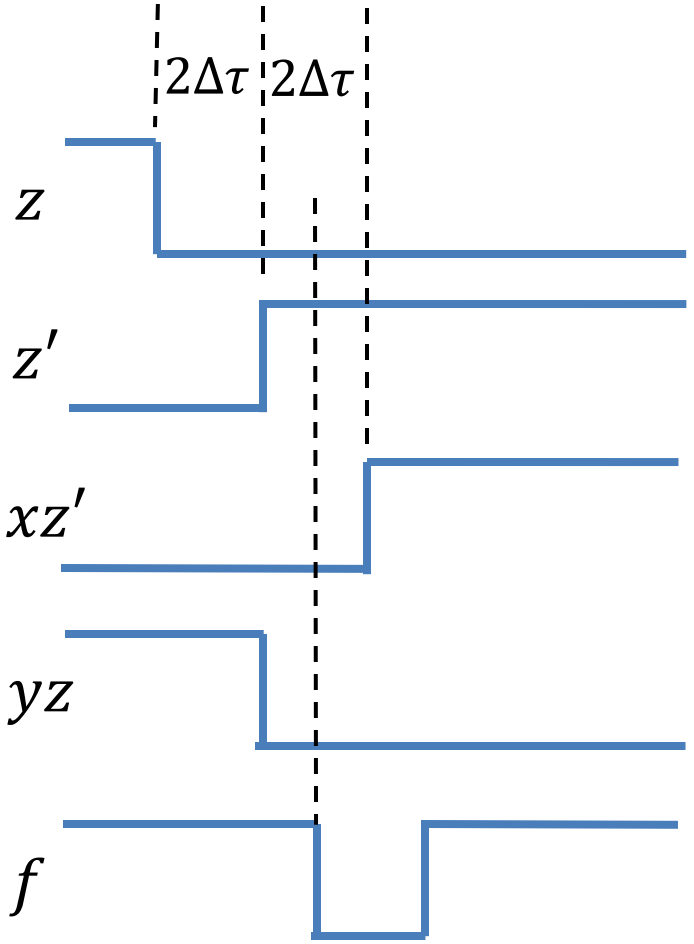


Assume that the propagation delay of NOT gate is $\Delta\tau$ and $2\Delta\tau$ for others.

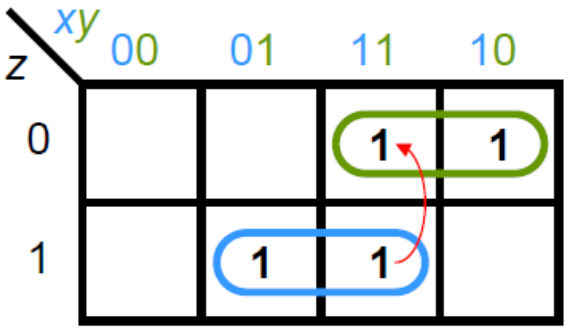
Work out the timing diagram to identify the presence of any timing hazard when the input condition changes from $(w, x, y, z) = (0,0,0,1)$ to $(0,1,0,1)$.

Finding Static Hazards with K-map





- $f(x, y, z) = xz' + yz$

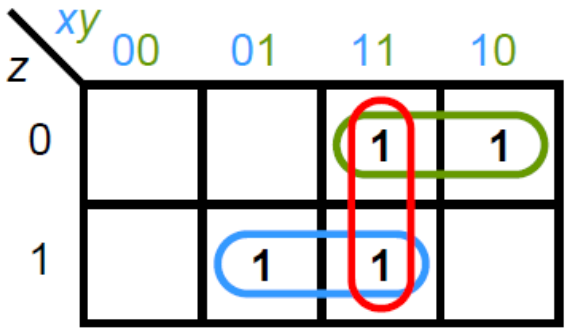


$(x, y, z) = (1,1,1)$ to $(1,1,0)$

Static-1 hazard!!!

Eliminating A Hazard

- Eliminating a hazard is to enclose the two minterms in question with another product term that overlaps both groupings
- Removal of hazards requires the addition of redundant gates to the circuit



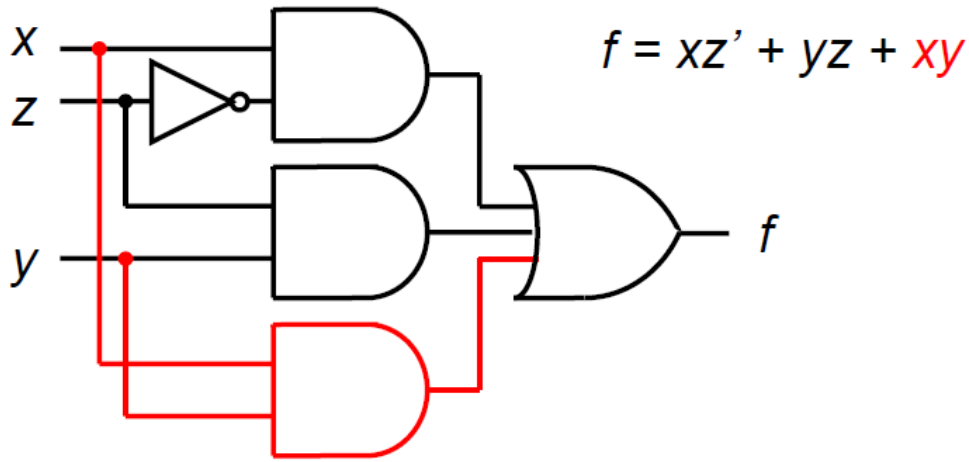
Include an redundant product term

$f = xz' + yz + xy$

Now the hazard is removed!

Eliminating A Hazard

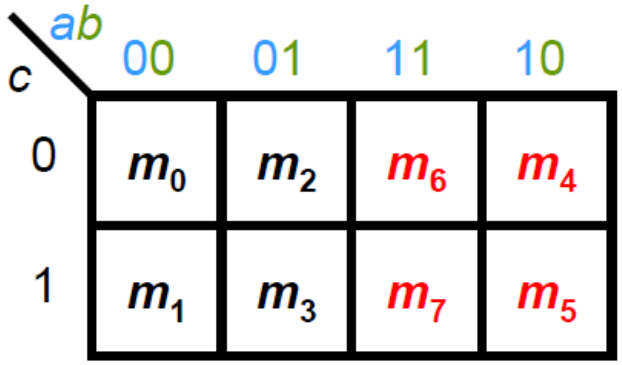
- Removal of hazards requires the addition of redundant gates to the circuit.



Exercise

Given $f(a, b, c) = \Sigma\, m\,(0,2,4,5)$

a) Minimize the function *f*
b) Realize *f to a* hazard-free circuit



3.4 Error Detection and Correction

- Data is transmitted in the form of binary bits (1 or 0)
- Noise might cause an error in the transmitted data (0 to 1 or 1 to 0)
- Error detection codes
  - Constant-weight code, e.g. 2-of-5 code
  - Parity bit
  - Hamming code

Constant-weight Codes (***m***-of-***n*** Code)

- A separable error detection code with a code word length of ***n*** bits, whereby each code word has exactly ***m*** instances of a "one"

Decimal numbers	**3-of-6 code**	
	3 data bits	Appended bits
0	000	111
1	001	110
2	010	110
3	011	100
4	100	110
5	101	100
6	110	100
7	111	000

- 3-of-6 code: 6 bits with 3 "1"s
- Can detect certain errors but not all (Single bit error)
- E.g. Original: 011100

1) 011100 -> Correct
2) 011101 -> Error detected
3) 011000 -> Error detected
4) 101100 -> Incorrect

Parity Code

- The simplest method for error detection is using parity bit
  - An additional bit (LSB) attaches to the original code
  - Two kinds of party bit (even or odd parities)

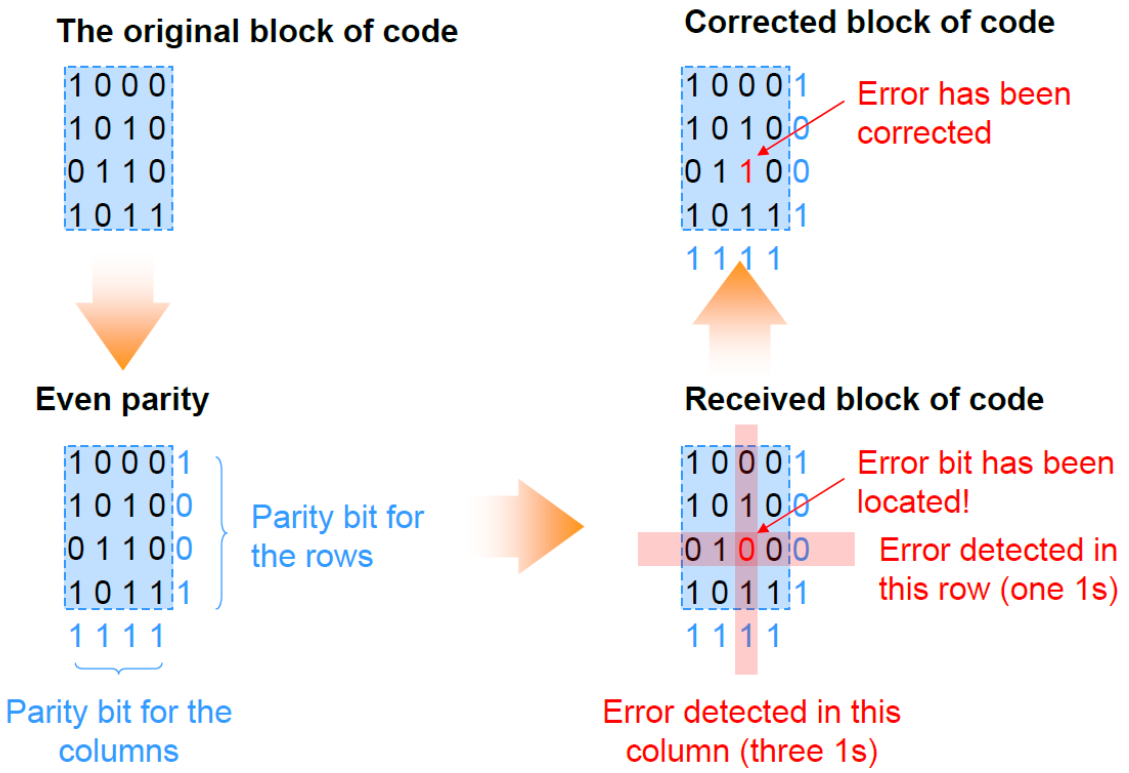
- The value of parity bit is defined by the total no. of "1"s in the resulting codeword either even or odd

Example of Parity Code

Decimal numbers	Binary code	Number of '1'	Even Parity Bit	Even Parity Code	Odd Parity Bit	Odd Parity Code
0	0000	0	0	00000	1	00001
1	0001	1	1	00011	0	00010
2	0010	1	1	00101	0	00100
3	0011	2	0	00110	1	00111
4	0100	1	1	01001	0	01000
5	0101	2	0	01010	1	01011
6	0110	2	0	01100	1	01101
7	0111	3	1	01111	0	01110
8	1000	1	1	10001	0	10000
9	1001	2	0	10010	1	10011
10	1010	2	0	10100	1	10101
11	1011	3	1	10111	0	10110
12	1100	2	0	11000	1	11001
13	1101	3	1	11011	0	11010
14	1110	3	1	11101	0	11100
15	1111	4	0	11110	1	11111

Error Detection and Correction

- Two-dimensional Bit Parity: Detect and Correct Single bit errors



Exercise

The following block of code is received based on an even parity, identify the errors and generate the corrected data.

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

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

Hamming Code

- Insert extra bit at specific positions to enable error detection and correction

Step 1: Calculate extra bit (***k***) needed for a ***n*** bit of code.

$$2^k \geq n + k + 1$$

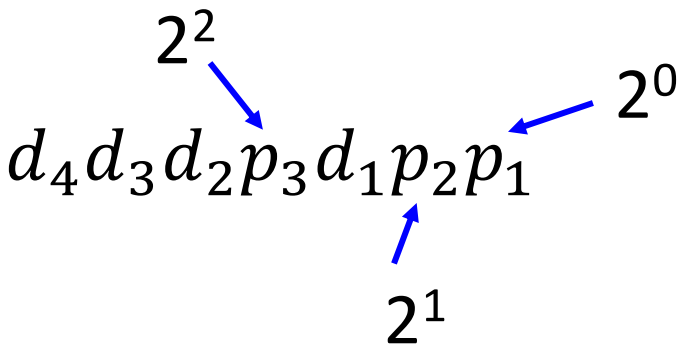
For a 4-bit data $d_4 d_3 d_2 d_1$, $n = 4$

$$2^k \geq 5 + k$$

Therefore, minimum value of *k* is 3. We need **3 parity bits**!

Hamming Code

Step 2: Place Parity Bits in the positions of powers of 2.



Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	**4**	3	**2**	**1**

Hamming Code

Step 3: Calculate each parity bits based on odd or even parity.

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	4	3	2	1
Binary Code	111	110	101	100	011	010	001
p_1	d_4		d_2		d_1		
p_2	d_4	d_3			d_1		
p_3	d_4	d_3	d_2				

p_1: Include all data bits in positions whose binary representation includes a 1 in the least significant position excluding Bit 1.

p_2: Include all data bits in positions whose binary representation includes a 1 in the position 2 from right excluding Bit 2.

p_3: Include all data bits in positions whose binary representation includes a 1 in the position 3 from right excluding Bit 4.

Hamming Code

Example: data $d_4d_3d_2d_1$ = 1000

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	4	3	2	1
Binary Code	111	110	101	100	011	010	001
p_1	1		0		0		
p_2	1	0			0		
p_3	1	0	0				
Even Parity	1	0	0		0		
Odd Parity	1	0	0		0		

Even Parity

$$p_1 = H_7 \oplus H_5 \oplus H_3 = 1 \oplus 0 \oplus 0 = 1$$
$$p_2 = H_7 \oplus H_6 \oplus H_3 = 1 \oplus 0 \oplus 0 = 1$$
$$p_3 = H_7 \oplus H_6 \oplus H_5 = 1 \oplus 0 \oplus 0 = 1$$

Odd Parity

$$p_1 = (H_7 \oplus H_5 \oplus H_3)' = (1 \oplus 0 \oplus 0)' = 0$$
$$p_2 = (H_7 \oplus H_6 \oplus H_3)' = (1 \oplus 0 \oplus 0)' = 0$$
$$p_3 = (H_7 \oplus H_6 \oplus H_5)' = (1 \oplus 0 \oplus 0)' = 0$$

Hamming Code

Example: data $d_4d_3d_2d_1$ = 1000

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	4	3	2	1
Binary Code	111	110	101	100	011	010	001
p_1	1		0		0		
p_2	1	0			0		
p_3	1	0	0				
Even Parity	1	0	0	**1**	0	**1**	**1**
Odd Parity	1	0	0	**0**	0	**0**	**0**

Even Parity

$$p_1 = H_7 \oplus H_5 \oplus H_3 = 1 \oplus 0 \oplus 0 = 1$$
$$p_2 = H_7 \oplus H_6 \oplus H_3 = 1 \oplus 0 \oplus 0 = 1$$
$$p_3 = H_7 \oplus H_6 \oplus H_5 = 1 \oplus 0 \oplus 0 = 1$$

Odd Parity

$$p_1 = (H_7 \oplus H_5 \oplus H_3)' = (1 \oplus 0 \oplus 0)' = 0$$
$$p_2 = (H_7 \oplus H_6 \oplus H_3)' = (1 \oplus 0 \oplus 0)' = 0$$
$$p_3 = (H_7 \oplus H_6 \oplus H_5)' = (1 \oplus 0 \oplus 0)' = 0$$

Exercise

Determine the Hamming code using both odd and even parity bit for a data code of 11100

Step 1: Calculate extra bit (***k***) needed for a ***n*** bit of code.

Step 2: Place Parity Bits in the positions of powers of 2.

Hamming Code	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1
Bit									

Exercise

Step 2: Place Parity Bits in the positions of powers of 2.

data code: 11100

Hamming Code	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1
Bit	9	8	7	6	5	4	3	2	1
Binary Code									
p_1									
p_2									
p_3									
p_4									
Even Parity									
Odd Parity									

Step 3: Calculate the number of '1' in each parity bits

Step 4: Place '1' if odd number of '1' for even parity; else '0'; Place '0' if odd number of '1' for odd parity; else '1'

Error Detection and Correction

Example: data $d_4d_3d_2d_1$ = 1000

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	4	3	2	1
Binary Code	111	110	101	100	011	010	001
p_1	1		0		0		
p_2	1	0			0		
p_3	1	0	0				
Even Parity	1	0	0	1	0	1	1
Odd Parity	1	0	0	0	0	0	0

Consider even parity and if we receive a code of 1001111, check the parity bits

$$c_1 = H_7 \oplus H_5 \oplus H_3 \oplus H_1 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$c_2 = H_7 \oplus H_6 \oplus H_3 \oplus H_2 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$c_3 = H_7 \oplus H_6 \oplus H_5 \oplus H_4 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$$

$$c_3c_2c_1 = (011)_2 = 3$$

Error Detection and Correction

Example: data $d_4d_3d_2d_1$ = 1000

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	4	3	2	1
Binary Code	111	110	101	100	011	010	001
p_1	1		0		0		
p_2	1	0			0		
p_3	1	0	0				
Even Parity	1	0	0	1	0	1	1
Odd Parity	1	0	0	0	0	0	0

Consider even parity and if we receive a code of 1001111, check the parity bits

$$c_1 = (H_7, H_5, H_3, H_1) = (1, 0, 1, 1) = 1$$
$$c_2 = (H_7, H_6, H_3, H_2) = (1, 0, 1, 1) = 1$$
$$c_3 = (H_7, H_6, H_5, H_4) = (1, 0, 0, 1) = 0$$

$$c_3c_2c_1 = (011)_2 = 3$$

Exercise

Example: data $d_4d_3d_2d_1$ = 1000

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Bit	7	6	5	4	3	2	1
Binary Code	111	110	101	100	011	010	001
p_1	1		0		0		
p_2	1	0			0		
p_3	1	0	0				
Even Parity	1	0	0	1	0	1	1
Odd Parity	1	0	0	0	0	0	0

Consider odd parity and if we receive a code of 1001000, check the parity bits

$$c_1 = (H_7, H_5, H_3, H_1)$$
$$c_2 = (H_7, H_6, H_3, H_2)$$
$$c_3 = (H_7, H_6, H_5, H_4)$$
$$c_3c_2c_1 =$$

Limitation?

Example: data $d_4d_3d_2d_1$ = 1000

Hamming Code	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	d_4	d_3	d_2	p_3	d_1	p_2	p_1
Odd Parity	1	0	0	0	0	0	0

If we receive a code of 10011100, check the parity bits

$$c_3c_2c_1 = (111)_2 = 7?$$

If we receive a code of 10111100, check the parity bits

$$c_3c_2c_1 = (010)_2 = 2?$$