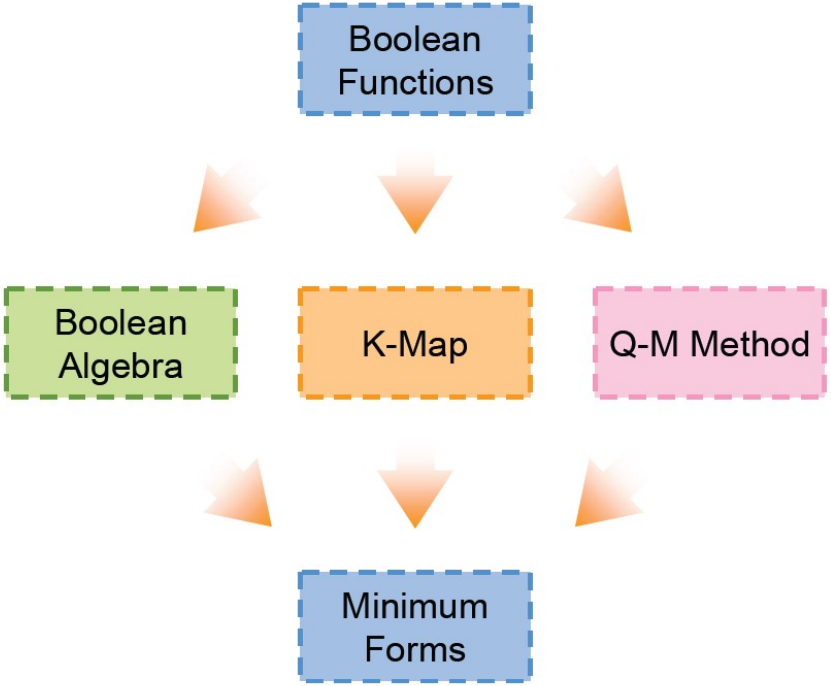
EE2000 Logic Circuit Design

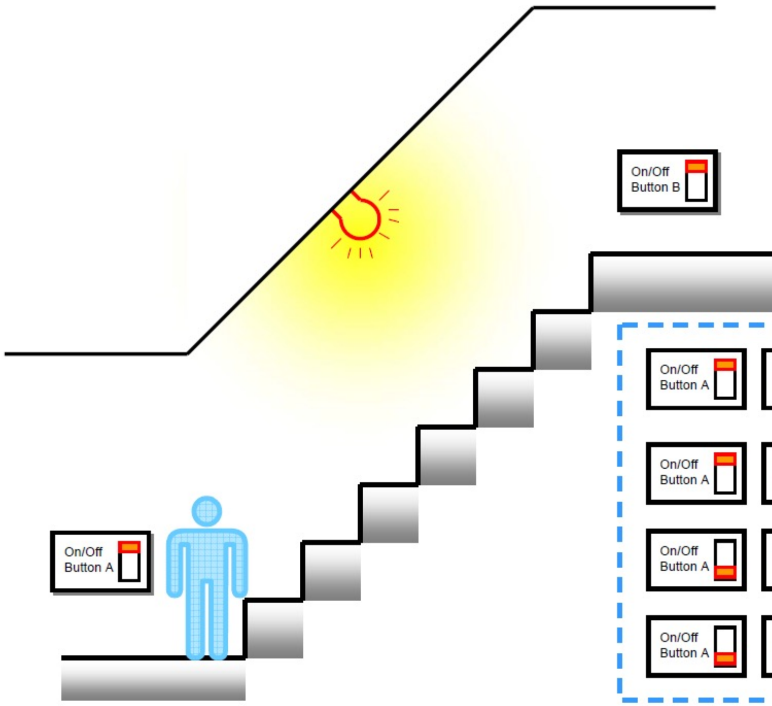
Lecture 3 – Combinational System Design

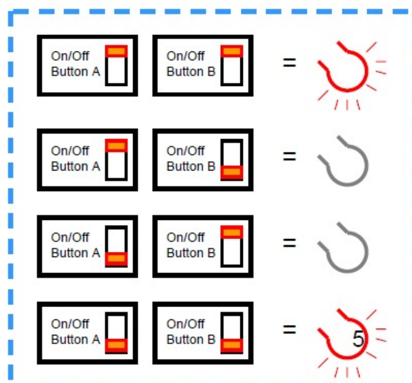


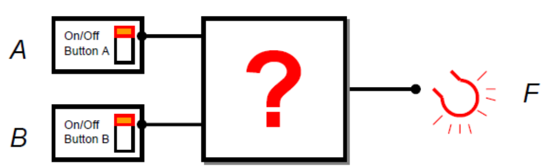
3.2 Design Procedure

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





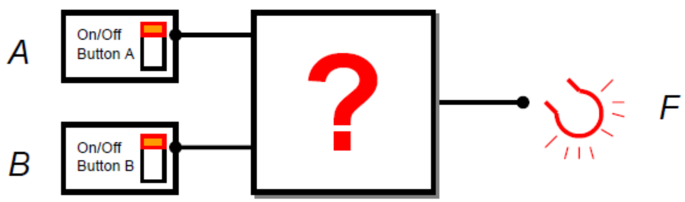


<u>State the case</u>

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

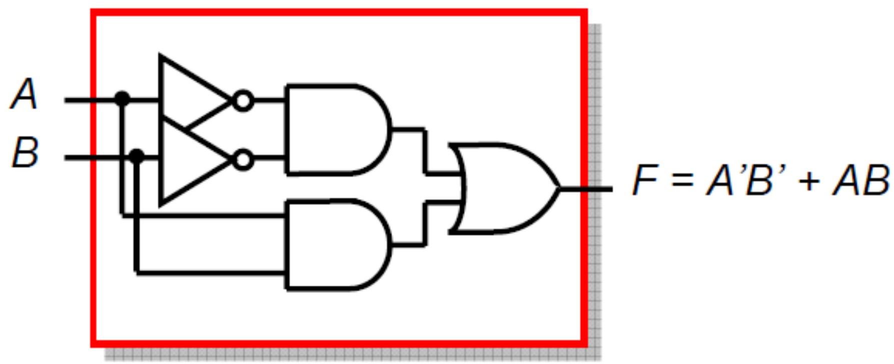
$F(A, B)$ is 1 if (A = 0 AND B = 0) OR

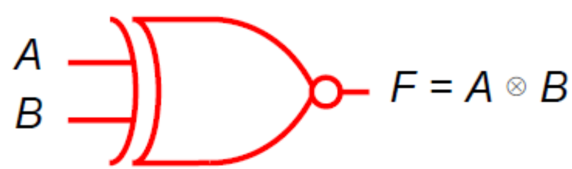
(A = 1 AND B = 1)

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

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

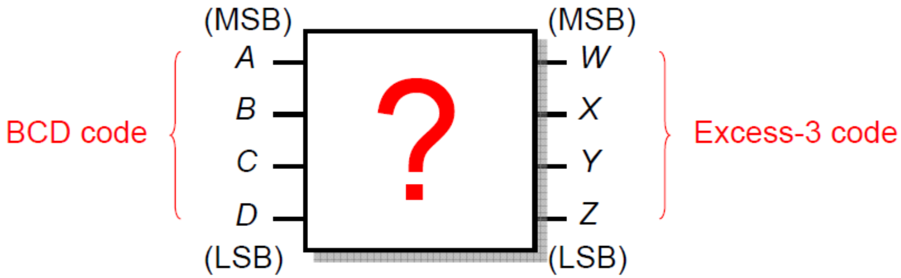
Logic Circuit Diagram





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.

<u>State the case</u>

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)				Output (Excess-3 code)			
	A	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	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

$W = A + BC + BD$

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

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

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

$Y = CD + C'D'$

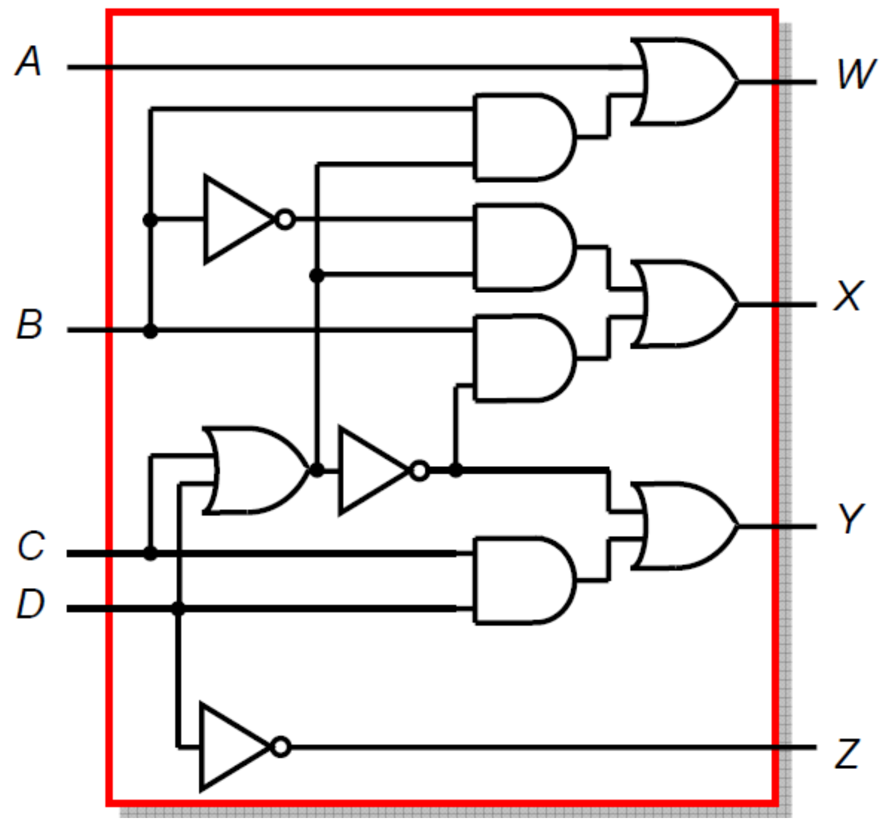
CD \ AB	00	01	11	10
00	1	1	x	1
01			x	
11	1	1	x	x
10			x	x

$Z = D'$

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

Logic Circuit

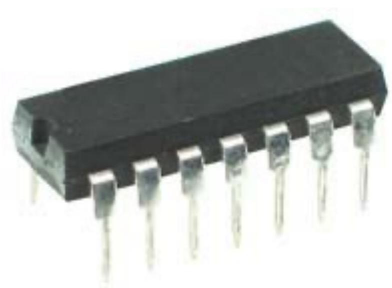
$$W = A + BC + BD = A + B(C + D)$$
$$X = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$$
$$Y = CD + C'D' = CD + (C + D)'$$
$$Z = D'$$

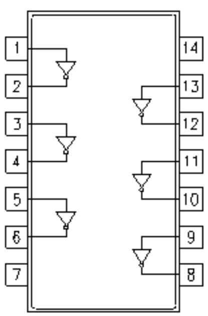


*Using only Two-input Gates and NOT Gates.

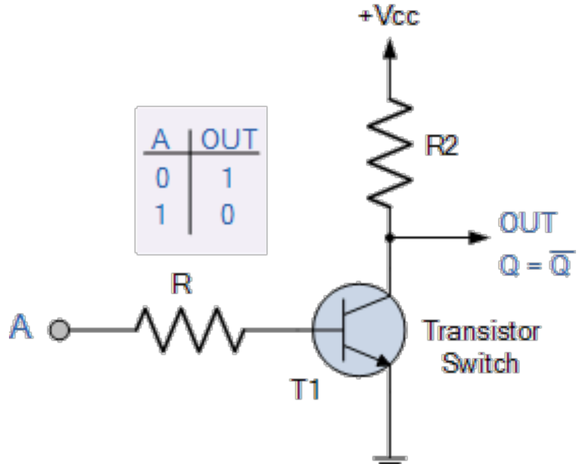
3.3 Timing Hazard

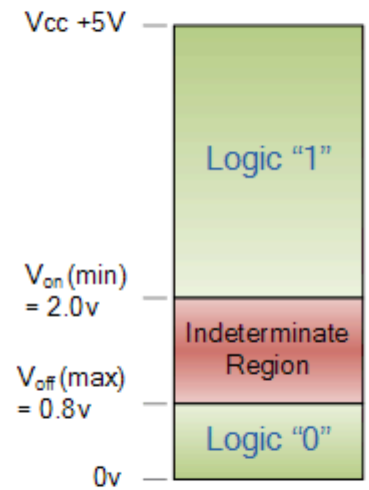
Logic devices (gates or other more complex circuits) are essentially made from semi-conductor





7404
Hex Inverter



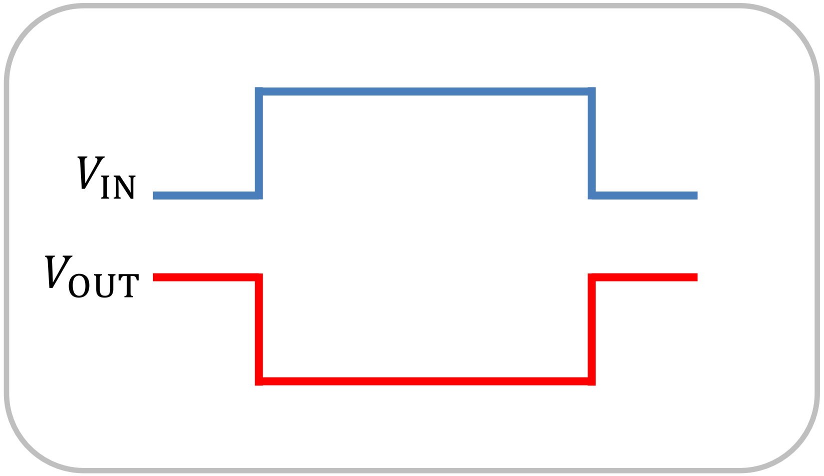


LS - TTL Input
Voltage Levels

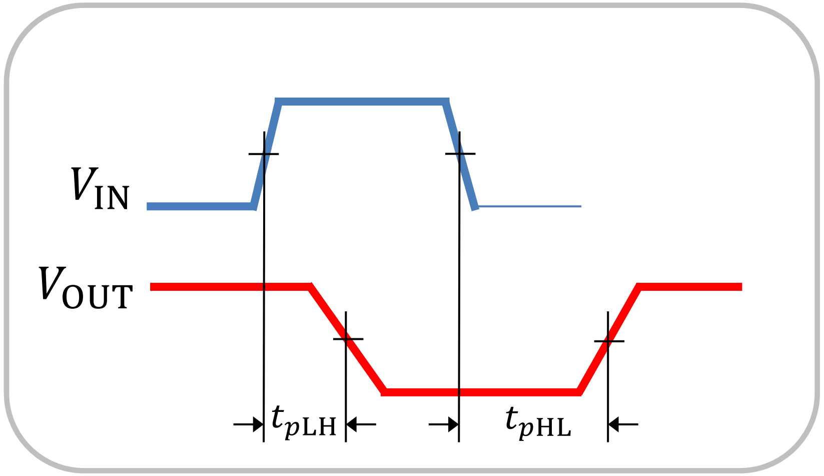
Propagation Delay

- "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



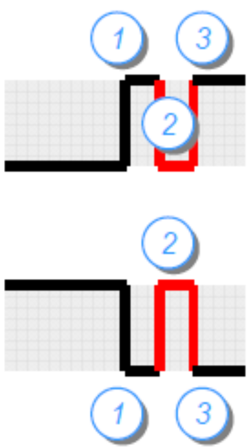
In Reality



Timing Hazards

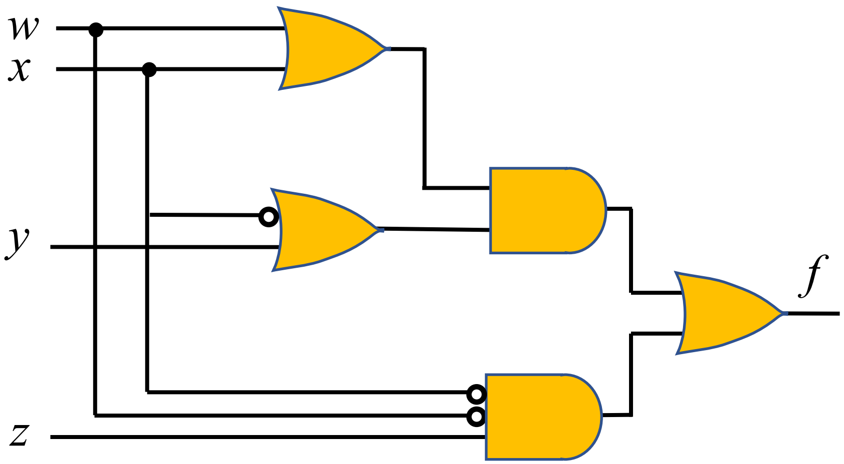
Static-0 hazard: the output may momentarily go to 1 when it should *remain 0*

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*

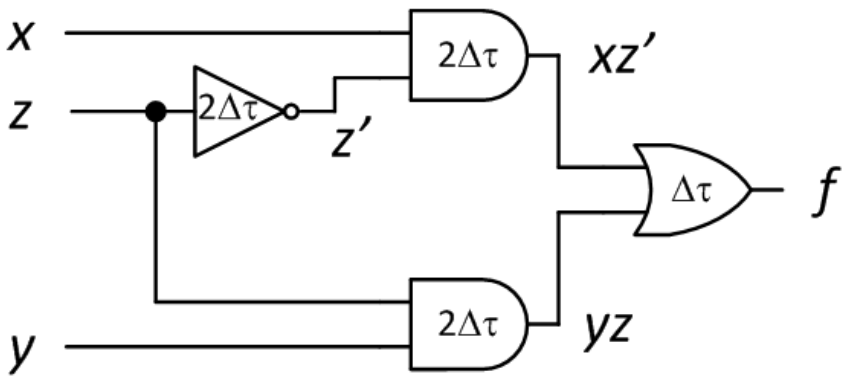
Exercise

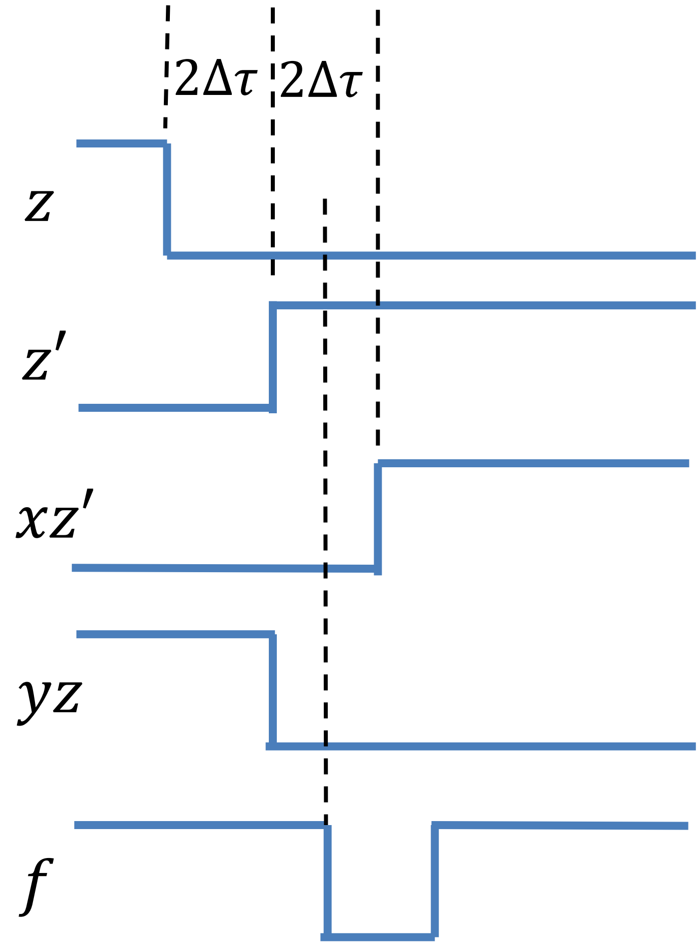


Assume that the propagation delay of NOT gate is $\Delta\tau$ and $2\Delta\tau$ for others .

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





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

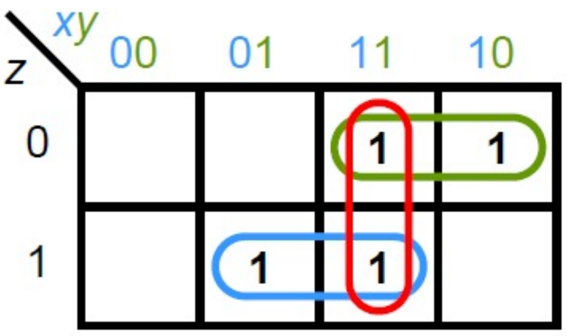
z \ xy	00	01	11	10
0			1	1
1		1	1	

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



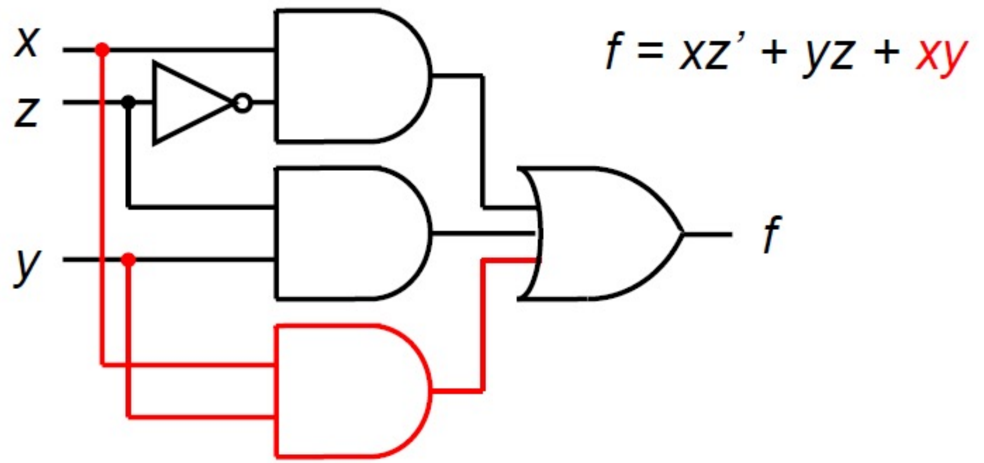
Include an redundant product term

$f = xz' + yz + xy$

Now the hazard is removed!

Eliminating A Hazard

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



Exercise

$c \backslash ab$	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

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

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
  - Gray 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

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

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” in the resulting codeword either even or odd

Example of Parity Code

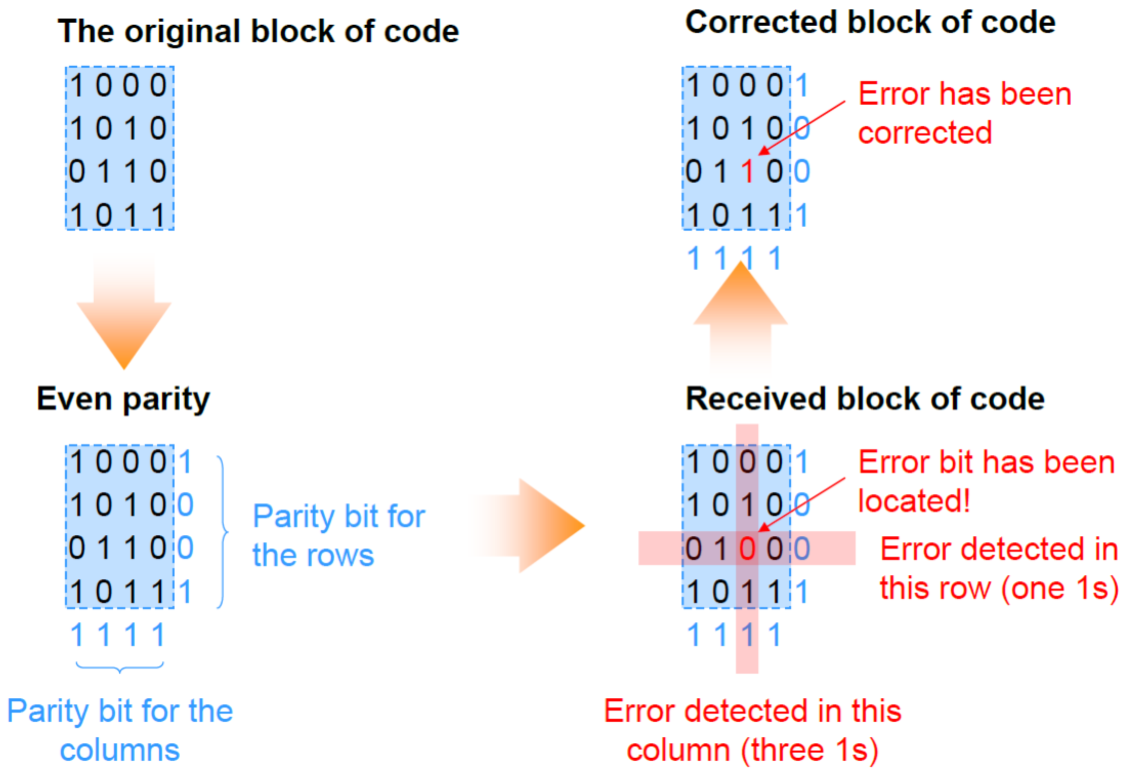
Decimal numbers	Binary code	Number of '1'	Even Parity Bit	Even Parity Code	Odd Parity Bit	Odd Parity Code
0	0000	0	0	00000	1	00001
1	0001	1	1	00011	0	00010
2	0010	1	1	00101	0	00100
3	0011	2	0	00110	1	00111
4	0100	1	1	01001	0	01000
5	0101	2	0	01010	1	01011
6	0110	2	0	01100	1	01101
7	0111	3	1	01111	0	01110
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

- Single Bit Parity: Detect single bit errors

  e.g. 111011 -> 111010

- 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.

$$d_4 d_3 d_2 \overset{2^2}{p_3} d_1 \underset{2^1}{p_2} \overset{2^0}{p_1}$$

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

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

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

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

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