FACULTAD DE INGENIERIA
Universidad de Buenos Aires

86.01 Técnica Digital

Códigos

Ing. Jorge H. Fuchs

Introducción

Objetivos de la clase:

Analizar las distintas posibilidades de representar información numérica y alfabética.

Conocer las características principales que puede presentar un código numérico BCD.

Estudiar las características que necesitan los códigos para la seguridad de la información (detectores y correctores de errores).

Evaluar el método de Hamming para detección y corrección de errores.

Definición

Para representar información se utilizan códigos de distintos tipos, primero analizaremos los códigos **numéricos**, particularmente los **BCD** (decimal codificado en binario), que me permiten representar dígitos decimales (**0 a 9**).

Características

- Pesado, posicional o ponderado
- Autocomplementado
- Progresivo
- Cerrado
- Reflejado
- Distancia mínima

Códigos BCD pesados

Sin peso		Pesado	
			8421
0	0000	0	0000
1	0101	1	0001
2	0010	2	0010
3	0100	3	0011
4	1111	4	0100
5	0110	5	0101
6	1011	6	0110
7	1101	7	0111
8	1010	8	1000
9	0011	9	1001

Códigos BCD pesados

	6311			5211	
0	0000		0	0000	
1	0001	(0010)	1	0001	(0010)
2	0011		2	0100	(0011)
3	0100		3	0101	(0110)
4	0101	(0110)	4	0111	
5	0111		5	1000	
6	1000		6	1010	(1001)
7	1001	(1010)	7	1011	(1100)
8	1011		8	1110	(1101)
9	1100		9	1111	

Códigos BCD autocomplementados

84(-2)(-1)		XS-3 (8421)	
0	0000	0	0011
1	0111	1	0100
2	0110	2	0101
3	0101	3	0110
4	0100	4	0111
5	1011	5	1000
6	1010	6	1001
7	1001	7	1010
8	1000	8	1011
9	1111	9	1100

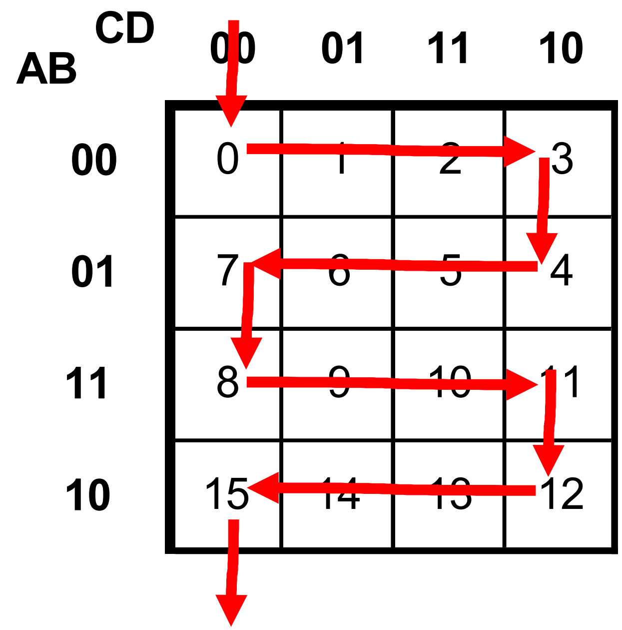
Código reflejado de Gray de 4 bits

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

Reflejado (por construcción)

Progresivo (solo cambia 1 bit)

Cerrado (solo cambia 1 bit del **15** al **0**)



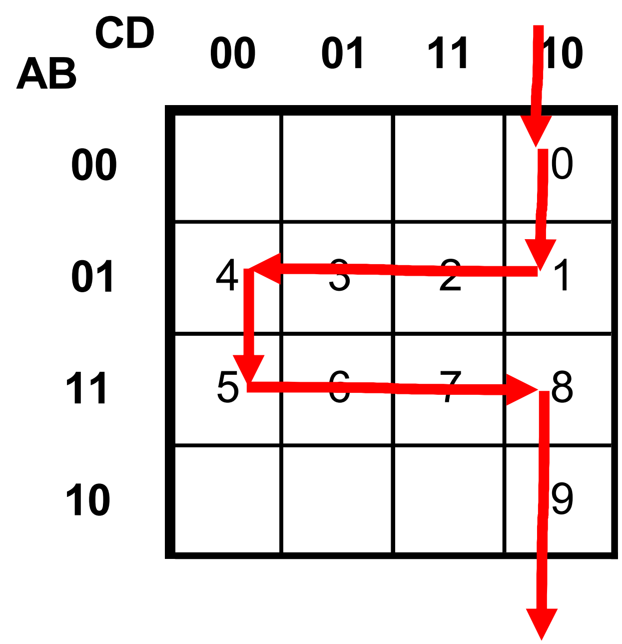
Código de Gray BCD XS-3

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

Reflejado (por construcción)

Progresivo (solo cambia 1 bit)

Cerrado (solo cambia 1 bit del **9** al **0**)



Representación de números en BCD

FACULTAD DE INGENIERIA Universidad de Buenos Aires

Puedo representar al número decimal 739 en binario, pero también mediante códigos **BCD** como los vistos:

	7	**3**	**9**	
Binario		$1011100011_{\|2}$		(10 bits)
BCD 8421	0111	0011	1001	(12 bits)
BCD 5211	1011	0101	1111	
BCD XS-3	1010	0110	1100	
BCD Gray XS-3	1111	0101	1010	

Distancia mínima de un código

Interesa estudiar la **distancia mínima** de un código:

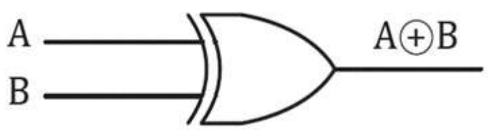
	8421
	ABCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

AB \ CD	**00**	**01**	**11**	**10**
00	0	1	3	2
01	4	5	7	6
11				
10	8	9		

dmin = 1 >>> no detecta errores de 1 bit

Distancia mínima de un código

Para aumentar **dmin** puedo agregar un **bit de paridad**:



A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

8421-P

	ABCD	**P**
0	0000	0
1	0001	1
2	0010	1
3	0011	0
4	0100	1
5	0101	0
6	0110	0
7	0111	1
8	1000	1
9	1001	0

P = 0

AB \ CD	00	01	11	10
00	0	x	3	x
01	x	5	x	6
11		x		x
10	x	9	x	

P = 1

AB \ CD	00	01	11	10
00	x	1	x	2
01	4	x	7	x
11	x		x	
10	8	x		x

dmin = 2 >>> detecta errores de 1 bit (no corrige)

Distancia mínima de un código

Supongamos un código con **dmin = 3**, analicemos parte del mapa K:

Si asumimos que solo puede haber 1 error:

dmin = 3 >>> detecta 1 error y lo corrige.

AB \ CD	00	01	11	10
00	1	1	2	1
01	1	2	2	2
11	x	3	2	x
10	1	x	x	x

Pero si asumimos que puede haber 2 errores:

dmin = 3 >>> detecta 2 errores sin corregirlos.

AB \ CD	00	01	11	10
00	1	x	x	x
01	x	3	2	x
11	3	3	3	x
10	x	3	x	

Distancia mínima de un código

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Ahora supongamos un código con **dmin = 4**, analicemos parte del mapa K:

dmin = 4 >>> detecta 2 errores o corrige 1.

Podemos establecer:

dmin = d + c + 1

1 = 0 + 0 + 1
2 = 1 + 0 + 1
3 = 1 + 1 + 1
3 = 2 + 0 + 1
4 = 2 + 1 + 1
4 = 3 + 0 + 1

AB \ CD	00	01	11	10
00	1	1	1/2	1
01	1	1/2	2	1/2
11	1/2	2	2	2
10	1	1/2	2	1/2

AB \ CD	00	01	11	10
00	1	1/2		1/2
01	1/2		1/2	
11		1/2	2	1/2
10	1/2		1/2	

Códigos redundantes

2 entre 5

	7421P
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

Biquinario

	50 43210
0	01 00001
1	01 00010
2	01 00100
3	01 01000
4	01 10000
5	10 00001
6	10 00010
7	10 00100
8	10 01000
9	10 10000

Johnson

0	00000
1	00001
2	00011
3	00111
4	01111
5	11111
6	11110
7	11100
8	11000
9	10000

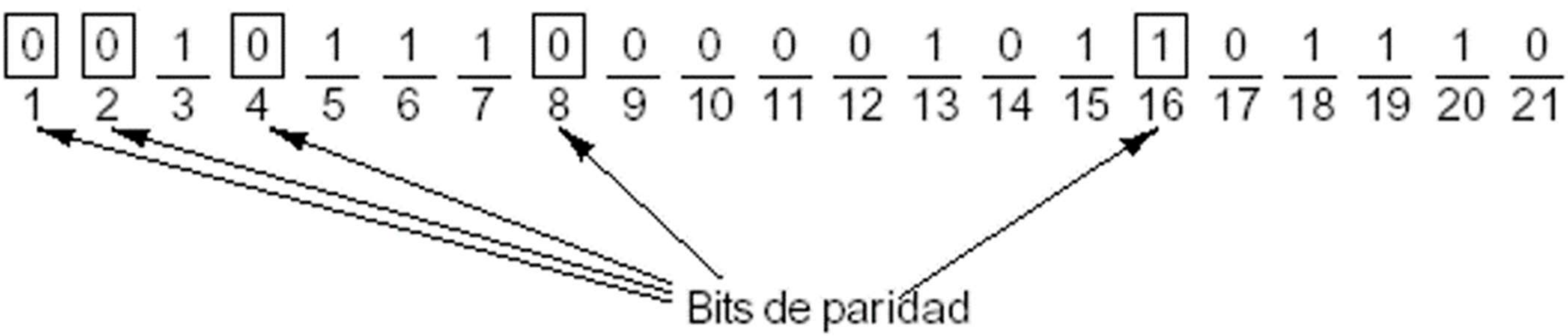
Método de Hamming

Hamming **no es un código** en sí mismo. El **método** agrega **bits de paridad** a palabras pertenecientes a cualquier código. Lo hace en posiciones determinadas para poder detectar y corregir errores.

k_i = bit de paridad

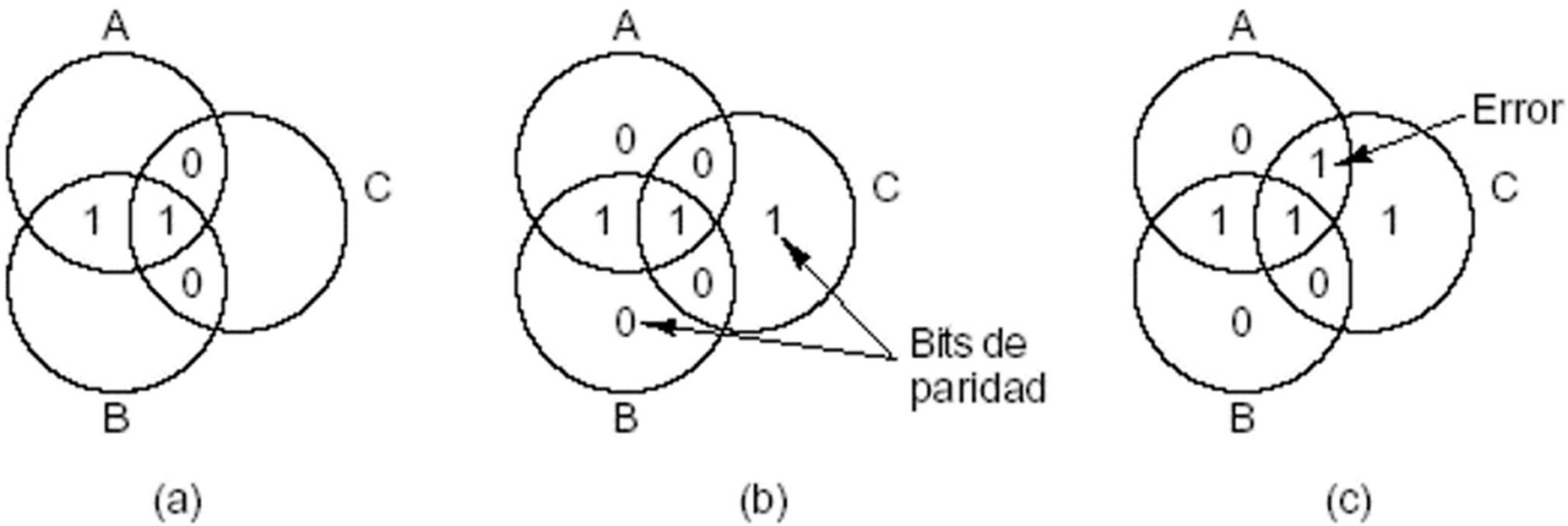
m_i = bit de mensaje

Los bits de paridad ocupan las posiciones que son potencias de 2: k_1, k_2, k_4, k_8, k_{16}, etc.



Método de Hamming

Para 4 bits de mensaje tendremos 3 bits de paridad: $\mathbf{k_1k_2m_3k_4m_5m_6m_7}$



k_i será la paridad par de los bits de mensaje que tengan en su posición en binario el peso i en 1:

k_1 >>> m_3 (011), m_5 (101) y m_7 (111)

k_2 >>> m_3 (011), m_6 (110) y m_7 (111)

k_4 >>> m_5 (101), m_6 (110) y m_7 (111)

Método de Hamming

Calculamos los k_i:

$$\mathbf{k_1} = m_3 \oplus m_5 \oplus m_7$$
$$\mathbf{k_2} = m_3 \oplus m_6 \oplus m_7$$
$$\mathbf{k_4} = m_5 \oplus m_6 \oplus m_7$$

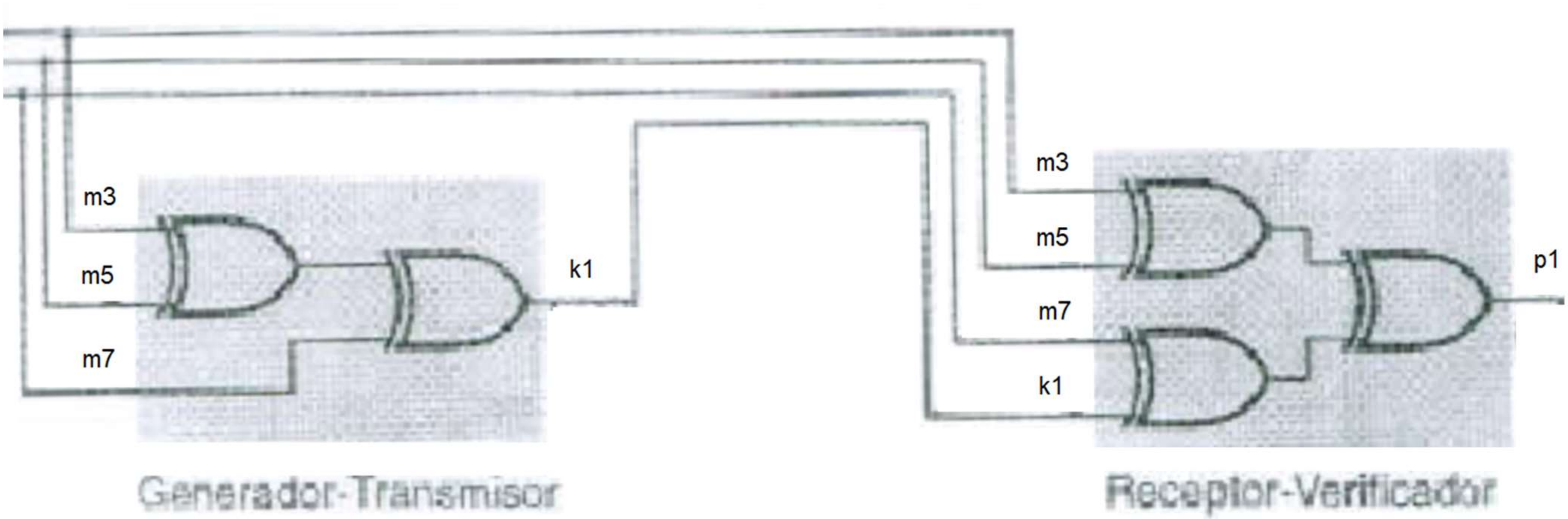
Los calculamos para $\mathbf{m_3m_5m_6m_7 = 1100}$

$$\mathbf{k_1} = 1 \oplus 1 \oplus 0 = \mathbf{0}$$
$$\mathbf{k_2} = 1 \oplus 0 \oplus 0 = \mathbf{1}$$
$$\mathbf{k_4} = 1 \oplus 0 \oplus 0 = \mathbf{1}$$

Por lo tanto enviamos: $\mathbf{k_1k_2m_3k_4m_5m_6m_7 = 0111100}$

Método de Hamming

Los circuitos **generador** de paridad y **verificador** quedan:



Debo verificar todas las paridades en el receptor (P_i), resultando **todas 0** si **no hay errores**.

Método de Hamming

En el receptor calculamos los P_i:

$$\mathbf{P_1} = k_1 \oplus m_3 \oplus m_5 \oplus m_7$$
$$\mathbf{P_2} = k_2 \oplus m_3 \oplus m_6 \oplus m_7$$
$$\mathbf{P_4} = k_4 \oplus m_5 \oplus m_6 \oplus m_7$$

Los calculamos para nuestro ejemplo: $\mathbf{k_1k_2m_3k_4m_5m_6m_7 \;=\; 0111100}$

$$\mathbf{P_1} = 0 \oplus 1 \oplus 1 \oplus 0 = \mathbf{0}$$
$$\mathbf{P_2} = 1 \oplus 1 \oplus 0 \oplus 0 = \mathbf{0}$$
$$\mathbf{P_4} = 1 \oplus 1 \oplus 0 \oplus 0 = \mathbf{0}$$

Por lo tanto **no hubo error**.

Método de Hamming

Si recibimos un **error** en $\mathbf{m_6}$: $\mathbf{k_1k_2m_3k_4m_5m_6m_7}$ = **0111110**

$$P_1 = 0 \oplus 1 \oplus 1 \oplus 0 = \mathbf{0}$$
$$P_2 = 1 \oplus 1 \oplus 1 \oplus 0 = \mathbf{1}$$
$$P_4 = 1 \oplus 1 \oplus 1 \oplus 0 = \mathbf{1}$$

Por obtener paridades distintas de 0 (impares), estoy en presencia de un **error**.

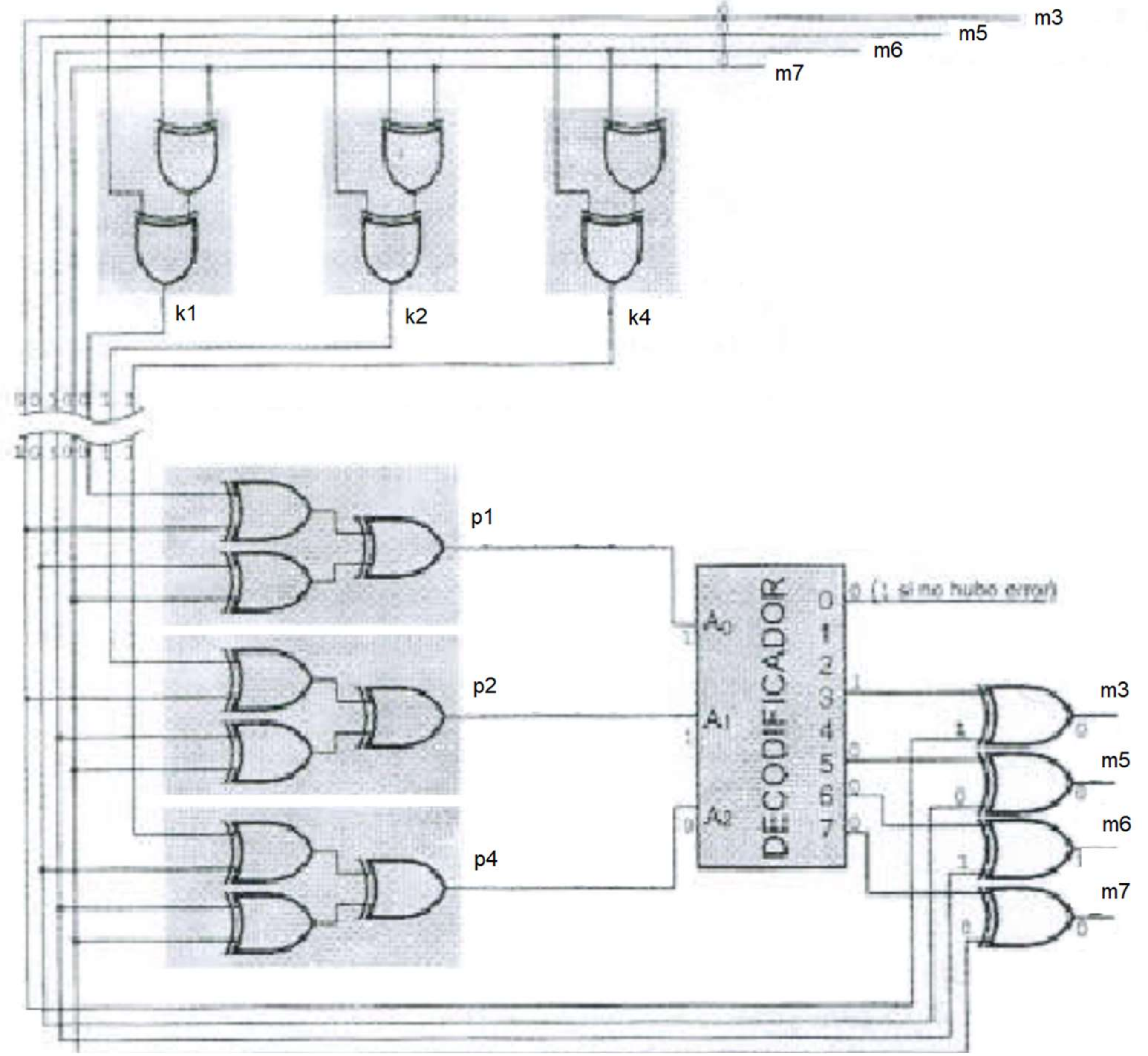
Para determinar en qué posición analizo:

$\mathbf{P_4P_2P_1}$ = **110** >>> el **error** está en el **bit 6**

Por lo tanto lo corrijo invirtiéndolo para obtener el mensaje original, y desecho los bits de paridad:

0111110 >>> **0111100** >>> $\mathbf{m_3m_5m_6m_7}$ **= 1100**

Método de Hamming



Método de Hamming

Para 4 bits de mensaje agregué 3 bits de paridad, 75 % de gasto adicional, sin embargo, por **cada bit** de paridad agregado **duplico** la cantidad de bits de mensaje, resultando apropiado para palabras largas:

Word size	Check bits	Total size	% de gasto extra
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

Códigos alfanuméricos

Los códigos **alfanuméricos**, tal como su nombre lo indica, permite representar letras, números, caracteres especiales y comandos.
Actualmente uno de los más utilizados es el ASCII (American Standard Code for Information Interchange). El estándar **ASCII** utiliza **7 bits** para representar los caracteres del alfabeto latino (inglés).
Los primeros 32 caracteres del código ASCII son conocidos como "**comandos** de control", estos sirven para configurar algunas opciones del texto sin imprimir ningún caracter en pantalla. Originalmente fue usado para enviar información a dispositivos de texto, como teleimpresoras, por lo que actualmente solo se utilizan unos pocos comandos.
El resto son caracteres imprimibles que incluyen números, letras mayúsculas y minúsculas (inglés), símbolos y caracteres especiales, etc.
Existe también el **ASCII extendido** o de **8 bits** (1 byte) que permite representar otros caracteres adicionales, como por ejemplo letras pertenecientes a otros idiomas, letras con tildes, etc.

Códigos alfanuméricos

Comandos ASCII

Comando ASCII	Código hexadecimal	Función
NUL	0x00	Carácter nulo
SOH	0x01	Inicio de encabezado
STX	0x02	Inicio de texto
ETX	0x03	Fin de texto
EOT	0x04	Fin de transmisión
ENQ	0x05	Consulta
ACK	0x06	Acuse de recibo
BEL	0x07	Timbre
BS	0x08	Retroceso *
HT	0x09	Tabulación horizontal
LF	0x0A	Salto de línea *
VT	0x0B	Tabulación vertical
FF	0x0C	De avance
CR	0x0D	Retorno de carro *
SO	0x0E	Mayúsculas fuera
SI	0x0F	En mayúsculas
DEL	0x10	Enlace de datos/Escape
DC1	0x11	Dispositivo de control 1
DC2	0x12	Dispositivo de control 2
DC3	0x13	Dispositivo de control 3
DC4	0x14	Dispositivo de control 4
NAK	0x15	Confirmación negativa
SYN	0x16	Síncrono en espera
ETB	0x17	Fin de transmisión de bloque
CAN	0x18	Cancelar

Códigos alfanuméricos

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Caracteres imprimibles ASCII 7 bits

Código hexadecimal	Símbolo ASCII	Código hexadecimal	Símbolo ASCII	Código hexadecimal	Símbolo ASCII
0x20	Espacio ‘ ‘	0x40	@	0x60	`
0x21	!	0x41	A	0x61	a
0x22	“	0x42	B	0x62	b
0x23	#	0x43	C	0x63	c
0x24	$	0x44	D	0x64	d
0x25	%	0x45	E	0x65	e
0x26	&	0x46	F	0x66	f
0x27	‘	0x47	G	0x67	g
0x28	(	0x48	H	0x68	h
0x29	)	0x49	I	0x69	i
0x2A	*	0x4A	J	0x6A	j
0x2B	+	0x4B	K	0x6B	k
0x2C	,	0x4C	L	0x6C	l
0x2D	-	0x4D	M	0x6D	m
0x2E	.	0x4E	N	0x6E	n
0x2F	/	0x4F	O	0x6F	o
0x30	0	0x50	P	0x70	p
0x31	1	0x51	Q	0x71	q
0x32	2	0x52	R	0x72	r
0x33	3	0x53	S	0x73	s
0x34	4	0x54	T	0x74	t

Códigos alfanuméricos

Caracteres imprimibles ASCII 7 bits (continuación)

0x35	5	0x55	U	0x75	u
0x36	6	0x56	V	0x76	v
0x37	7	0x57	W	0x77	w
0x38	8	0x58	X	0x78	x
0x39	9	0x59	Y	0x79	y
0x3A	:	0x5A	Z	0x7A	z
0x3B	;	0x5B	[	0x7B	{
0x3C	<	0x5C	\	0x7C	|
0x3D	=	0x5D	]	0x7D	}
0x3E	>	0x5E	^	0x7E	~
0x3F	?	0x5F	_