Universidad de Granada

decsai.ugr.es

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

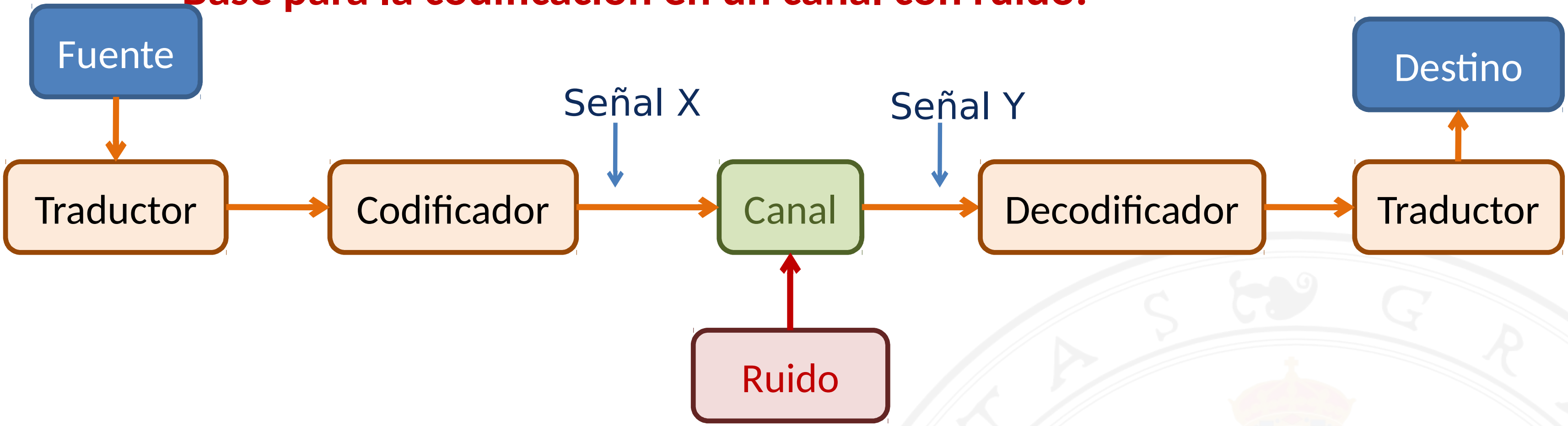
Seminario 4.- Códigos detectores y códigos correctores.

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

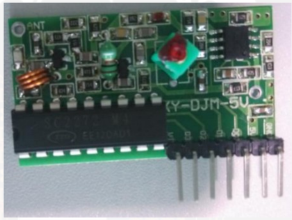
1. Códigos de Hamming para detección
2. Códigos de Hamming para corrección
3. Códigos lineales para detección y corrección

– **Base para la codificación en un canal con ruido:**



– En nuestro sistema, la **fuente** será el mando RF **2262**.
– Codificará los datos de los botones leídos y los enviará al receptor de radiofrecuencia **PT2272**.

Emisor PT 2262



Receptor RF 2272

- **Haremos uso del ejemplo de recepción de datos para los módulos 2262/2272 visto en el seminario anterior.**
- La función **LeerRF2272** devuelve un código de Hamming **(7,4)**

```
/**
 * Realiza una lectura de los bits de los códigos de los botones pulsados en el emisor RF2262 y recibidos en el RF2272.
 * @return El código leído, que se corresponde con un código de Hamming (7,4) con las siguiente distribución:
 *   - Bit 0: Paridad de Hamming 1
 *   - Bit 1: Paridad de Hamming 2
 *   - Bit 2: Bit de datos. Botón A pulsado (1) o no pulsado (0)
 *   - Bit 3: Paridad de Hamming 3
 *   - Bit 4: Bit de datos. Botón B pulsado (1) o no pulsado (0)
 *   - Bit 5: Bit de datos. Botón C pulsado (1) o no pulsado (0)
 *   - Bit 6: Bit de datos. Botón D pulsado (1) o no pulsado (0)
 */
uint8_t leerRF2272();
```

- **Un código de Hamming (7,4) es un código de bloque, que codifica en n=7 bits códigos de k=4 bits.**

- **Los bits de las posiciones 0, 1 y 3 son bits de paridad**
- Los bits de las posiciones 2, 4, 5 y 6 son los bits de datos

DDDPDPP

- Los códigos de Hamming **(7,4) aseguran una distancia de Hamming igual a 3 → Se pueden detectar 2 errores de 1 bit en un mensaje.**

- **Cálculo de la paridad:** bits 0, 1, 2, 3, 4, 5, 6 (7 bits)
  - **Es paridad par en todos los casos**

- **Bit de paridad 0:** Se consigue haciendo que la suma módulo 2 de los bits 0, 2, 4, 6 sea par

D D D P D P P

- **Se calcula fácilmente con operadores XOR:** bit0= bit2 XOR bit4 XOR bit6

- **Bit de paridad 1:** Se consigue haciendo que la suma módulo 2 de los bits 1, 2, 5, 6 sea par

D D D P D P P

- **Se calcula fácilmente con operadores XOR:** bit1= bit2 XOR bit5 XOR bit6

- **Bit de paridad 3:** Se consigue haciendo que la suma módulo 2 de los bits 3, 5, 5, 6 sea par

D D D P D P P

- **Se calcula fácilmente con operadores XOR:** bit3= bit4 XOR bit5 XOR bit6

- **Cuando recibimos un código, ¿cómo detectamos errores?**
- Hay que calcular la paridad par para los 3 bits de paridad.
- **En caso de que alguno no salga con paridad par, hay un error al menos.**

  - **Error en bit 0:** bit0 XOR bit2 XOR bit4 XOR bit6
  - **Error en bit 1:** bit1 XOR bit2 XOR bit5 XOR bit6
  - **Error en bit 3:** bit3 XOR bit4 XOR bit5 XOR bit6

- **Si alguno de los errores anteriores es 1, entonces se ha detectado un error al menos.**

Tarea principal a realizar en la práctica: Función ***decodificar***:

```
bool decodificar(const uint8_t codigo, bool pulsados[4]) {

    pulsados[0]= ((codigo&(1<<2))>0);
    pulsados[1]= ((codigo&(1<<4))>0);
    pulsados[2]= ((codigo&(1<<5))>0);
    pulsados[3]= ((codigo&(1<<6))>0);

    return true;
}
```

Partiendo de la base de la función anterior, se debe modificar su implementación para que devuelva:

- True: Se ha decodificado correctamente. Los valores de ***pulsados*** valdrán true/false dependiendo de si se han pulsado o no los botones **A, B, C, D** del mando, respectivamente.
- False: Se han detectado errores. Se debe devolver ***pulsados*** con todas las componentes a ***false***.

DECSAI

- Un código de Hamming **(7,4)** **es también un código lineal**.

- **Calculando el síndrome, se nos indica en qué bit se encuentra el error.**
- **Un código de Hamming (7,4) es un código de bloque, que codifica en n=7 bits códigos de k=4 bits.**

- **Un código de Hamming (7,4) asegura una distancia de Hamming igual a 3. Por tanto, se puede corregir un error de 1 bit.**

DDDPDPP

- **Generación de un código de Hamming**
  - **¿Cómo calcular la matriz del código? Ejemplo para el código de Hamming (7,4).**

Las posiciones de los bits 3º, 5º, 6º y 7º son los bits del código, por lo que podemos diseñar parcialmente la matriz G:

$$(x_4 x_3 x_2 x_1)\cdot G = (x_4 x_3 x_2 p_3 x_1 p_2 p_1) \Rightarrow G = \begin{pmatrix} 1 & 0 & 0 & ? & 0 & ? & ? \\ 0 & 1 & 0 & ? & 0 & ? & ? \\ 0 & 0 & 1 & ? & 0 & ? & ? \\ 0 & 0 & 0 & ? & 1 & ? & ? \end{pmatrix}$$

– **Generación de un código de Hamming**

– **¿Cómo calcular la matriz del código? Ejemplo para el código de Hamming (7,4).**

Para el bit de paridad p_3, sabemos que debe tomar paridad par con los bits desde el 4º hasta el 7º. Es decir, consigo mismo y con los bits del código x_4, x_3 y x_2.

Podemos rellenar la columna de la matriz para calcular p_3:

$$p_3 = x_4 + x_3 + x_2$$

$$(x_4 x_3 x_2 x_1) \cdot G = (x_4 x_3 x_2 p_3 x_1 p_2 p_1) \Rightarrow G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & ? & ? \\ 0 & 1 & 0 & 1 & 0 & ? & ? \\ 0 & 0 & 1 & 1 & 0 & ? & ? \\ 0 & 0 & 0 & 0 & 1 & ? & ? \end{pmatrix}$$

– **Generación de un código de Hamming**

– **¿Cómo calcular la matriz del código? Ejemplo para el código de Hamming (7,4).**

Para el bit de paridad p_2, sabemos que debe tomar paridad par con los bits 2º (él mismo), 3º, 6º y 7º. Es decir, con los bits del código x_1, x_3 y x_4.

Podemos rellenar la columna de la matriz para calcular p_2:

$$p_2 = x_4 + x_3 + x_1$$

$$(x_4 x_3 x_2 x_1) \cdot G = (x_4 x_3 x_2 p_3 x_1 p_2 p_1) \Rightarrow G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & ? \\ 0 & 1 & 0 & 1 & 0 & 1 & ? \\ 0 & 0 & 1 & 1 & 0 & 0 & ? \\ 0 & 0 & 0 & 0 & 1 & 1 & ? \end{pmatrix}$$

– **Generación de un código de Hamming**

– **¿Cómo calcular la matriz del código? Ejemplo para el código de Hamming (7,4).**

Para el bit de paridad p_1, sabemos que debe tomar paridad par con los bits 1º (él mismo), 3º, 5º y 7º. Es decir, con los bits del código x_1, x_2 y x_4.

Podemos rellenar la columna de la matriz para calcular p_1:

$$p_1 = x_4 + x_2 + x_1$$

$$(x_4 x_3 x_2 x_1) \cdot G = (x_4 x_3 x_2 p_3 x_1 p_2 p_1) \Rightarrow G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

- **Ejemplo para calcular un código de Hamming**

  - **Codificación utilizando la matriz generadora del código.**

  Ejemplo: x=(1101) → Pulsados A, C, D

  c= x·G= (1101)·G= (1 1 0 0 1 1 0)

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

– **Cálculo del síndrome**

– Cálculo de la matriz de control de paridad H(C):
  – Muy sencillo:
    1. Basta con incluir tantas columnas como bits de paridad.
    2. Tantas filas como longitud del código (**n**).
    3. Cada columna va asignada a un bit de paridad. **Se asignan de izquierda a derecha de menor a mayor:**

Ejemplo del código (7,4):

$$H(C) = \begin{array}{c} \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} \\ \begin{array}{ccc} p_3 & p_2 & p_1 \end{array} \end{array}$$

- **Cálculo del síndrome**

  - Cálculo de la matriz de control de paridad H(C):

    4. **Por último, se rellenan las filas con los bits asignados para cada paridad, según el código diseñado.**

Ejemplo del código (7,4):

$$H(C) = \begin{array}{c} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ \\ \end{array} \begin{array}{c} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ \begin{array}{ccc} p_3 & p_2 & p_1 \end{array} \end{array}$$

- **Cálculo del síndrome:**
  - El síndrome se calcula multiplicando el código recibido **c’** por la matriz: **s(c’)= c’·H(C)**

Ejemplo del código (7,4):

Se recibe c’= (1100110)

$$c'\cdot H(C) = (1100110)\cdot H(C) = (1100110)\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (000)$$

Síndrome (000)= No hay errores

- **Cálculo del error:**
  - **En códigos de Hamming**, por cómo está diseñada H(C), el síndrome proporciona una cadena de bits que se pueden interpretar como un número entero.
  - Este número indica el bit en el que se recibe el error.
  - **¡No necesitamos tablas de síndromes!**

Ejemplo del código (7,4):
Se envía c = (1100110)
Se recibe c'=(1110110)

$$c'\cdot H(C) = (1110110)\cdot H(C) = (1110110)\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (101)$$

Según el síndrome, el error está en el 5° bit
Error e=(0010000)
Corrección: c=c'-e=(1110110)-(0010000)= (1100110)

Tarea principal a realizar en la práctica: Función ***decodificar***:

```
bool decodificar(const uint8_t codigo, bool pulsados[4]) {

    pulsados[0]= ((codigo&(1<<2))>0);
    pulsados[1]= ((codigo&(1<<4))>0);
    pulsados[2]= ((codigo&(1<<5))>0);
    pulsados[3]= ((codigo&(1<<6))>0);

    return true;
}
```

Partiendo de la base de la función anterior, se debe modificar su implementación para que devuelva:

- True: Se ha decodificado correctamente. Los valores de ***pulsados*** valdrán true/false dependiendo de si se han pulsado o no los botones **A, B, C, D** del mando, respectivamente.
- False: Se han detectado errores y se han corregido. Los valores de ***pulsados*** valdrán true/false dependiendo de si se han pulsado o no los botones **A, B, C, D** del mando, respectivamente.

- **Para esta parte de la práctica, utilizaremos el tarjetero MF RC522.**

- **Crearemos un código uniforme de tamaño k=5, que contenga palabras para representar el alfabeto de la fuente siguiente:**
  - Todos los símbolos del alfabeto en mayúsculas, salvo la ñ
  - Los signos de puntuación punto (.), coma (,), punto y coma (;), espacio ( ), dos puntos (:) y el carácter de terminación de cadena '\0'.
  - En total: 32 símbolos

- **Puede reutilizarse y modificarse el código uniforme de la práctica 1**

- **Por tanto, cada palabra del código ocupará k=5 bits.**

- El alumno deberá diseñar un **código lineal de 12 bits** que permita corregir el máximo número de errores posible.

- Se deberá diseñar, en particular:
  - La matriz de codificación (matriz generadora),
  - La matriz de cálculo de síndromes
  - La tabla de síndromes y sus errores asociados

- Además, se deberán implementar:
  - Métodos para codificar una palabra del código uniforme en una palabra del código lineal.
  - Métodos para decodificar una palabra del código lineal en el código uniforme.
  - Métodos para codificar y decodificar los símbolos del alfabeto de la fuente en el código uniforme (modificación de la implementación de la práctica 1).

– **Estructura de la matriz generadora del código lineal**

– Se diseñará la matriz generadora **M(C)** como la concatenación de la matriz de paridad con la matriz identidad.

– Como el código uniforme original tiene k=5 bits, la matriz deberá tener 5 filas.

$$M(C)=[P_{5,7}|I_{5,5}]$$

– Como el código lineal a diseñar tiene 12 bits, la matriz deberá tener 12 columnas.

$$M(C)=\begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} & p_{1,5} & p_{1,6} & p_{1,7} & 1 & 0 & 0 & 0 & 0 \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} & p_{2,5} & p_{2,6} & p_{2,7} & 0 & 1 & 0 & 0 & 0 \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} & p_{3,5} & p_{3,6} & p_{3,7} & 0 & 0 & 1 & 0 & 0 \\ p_{4,1} & p_{4,2} & p_{4,3} & p_{4,4} & p_{4,5} & p_{4,6} & p_{4,7} & 0 & 0 & 0 & 1 & 0 \\ p_{5,1} & p_{5,2} & p_{5,3} & p_{5,4} & p_{5,5} & p_{5,6} & p_{5,7} & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Se deberán diseñar los valores de la matriz de paridad para poder detectar y corregir el máximo número de errores posible.

– **No olvidar que las filas deben ser linealmente independientes.**

– Estructura de la matriz del cálculo de síndromes

– Se diseñará la matriz de comprobación de errores **H(C)** (cálculo de síndromes) como la concatenación de la matriz identidad con la traspuesta de la matriz de paridad diseñada para la matriz generadora del código.

– Como el código lineal tiene n=12 bits, la matriz debe tener 12 filas.

– Como el código inicial uniforme tiene k=5 bits, los síndromes, por tanto, serán vectores de tamaño n-k=12-5=7

$$H(C)=[I_{n-k}|P^t_{k,n-k}]=[I_{7,7}|P^t_{5,7}]$$

$$H(C)=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & p_{1,1} & p_{2,1} & p_{3,1} & p_{4,1} & p_{5,1} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & p_{1,2} & p_{2,2} & p_{3,2} & p_{4,2} & p_{5,2} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & p_{1,3} & p_{2,3} & p_{3,3} & p_{4,3} & p_{5,3} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & p_{1,4} & p_{2,4} & p_{3,4} & p_{4,4} & p_{5,4} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & p_{1,5} & p_{2,5} & p_{3,5} & p_{4,5} & p_{5,5} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & p_{1,6} & p_{2,6} & p_{3,6} & p_{4,6} & p_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & p_{1,7} & p_{2,7} & p_{3,7} & p_{4,7} & p_{5,7} \end{pmatrix}$$

– Estructura de la tabla de síndromes

– Se diseñará una tabla donde, a cada síndrome (vector de longitud 7), se le asigne un error (vector de longitud 12).
– El cálculo de la tabla de síndromes consiste en iterar, para cada posible ocurrencia de errores **e** que se puedan corregir (1 error, 2 errores, etc.), y calcular su síndrome asociado $\mathbf{s_e}$.
– **Ejemplo:** Todas las posibles ocurrencias de errores de 1 bit, todas las posibles ocurrencias de errores de 2 bits, etc.

Error e_1=(000000000001) → $e_1 * H^t(C) = s_{e1} = (s^1{}_1, s^1{}_2, s^1{}_3, s^1{}_4, s^1{}_5, s^1{}_6, s^1{}_7)$
Error e_2=(000000000010) → $e_2 * H^t(C) = s_{e2} = (s^1{}_1, s^1{}_2, s^1{}_3, s^1{}_4, s^1{}_5, s^1{}_6, s^1{}_7)$
…
Error e_{12}=(100000000000) → $e_{12} * H^t(C) = s_{e12} = (s^{12}{}_1, s^{12}{}_2, s^{12}{}_3, s^{12}{}_4, s^{12}{}_5, s^{12}{}_6, s^{12}{}_7)$
Error e_{13}=(000000000011) → $e_{12} * H^t(C) = s_{e12} = (s^{12}{}_1, s^{12}{}_2, s^{12}{}_3, s^{12}{}_4, s^{12}{}_5, s^{12}{}_6, s^{12}{}_7)$
…
Error e_{xx}=(001000010000) → $e_{xx} * H^t(C) = s_{exx} = (s^{xx}{}_1, s^{xx}{}_2, s^{xx}{}_3, s^{xx}{}_4, s^{xx}{}_5, s^{xx}{}_6, s^{xx}{}_7)$
…
Error e_{yy}=(110000000000) → $e_{yy} * H^t(C) = s_{eyy} = (s^{yy}{}_1, s^{yy}{}_2, s^{yy}{}_3, s^{yy}{}_4, s^{yy}{}_5, s^{yy}{}_6, s^{yy}{}_7)$

- **Método para codificar**

- Para cada símbolo del alfabeto de la fuente existente en el mensaje, se deberá calcular en primer lugar su palabra del código uniforme asociado.
- La palabra del código uniforme deberá multiplicarse por la matriz generadora, para calcular la palabra de 12 bits del código lineal

- La palabra resultante se codificará en un tipo **uint16_t**. De este modo, como podemos guardar 64B en la tarjeta, sólo podremos guardar 32 palabras del código.

- Las palabras del código lineal codificadas se escribirán en la tarjeta usando la API proporcionada de la biblioteca ***Tarjetero***.

- **Método para decodificar**

- Se leerán 64 bytes de la tarjeta, con la API ***Tarjetero*** proporcionada en la práctica. Esos bytes se leerán como 32 valores de tipo ***uint16_t***.
- Cada valor ***uint16_t*** contendrá, en los 12 primeros bits, el valor del código lineal.
- Se deberá calcular el síndrome del valor, muntiplicándolo por $H^t(C)$.
- Se deberá acudir a la tabla de síndromes para conocer qué vector de error tiene asociado.
- Se corregirá el error, cambiando el valor de los bits afectados.
- Se decodificará (extraerá) el código uniforme inicial desde el valor con el error corregido.
- Se decodificará el código uniforme en el símbolo de la fuente correspondiente.

Universidad de Granada

decsai.ugr.es

Teoría de la Información y la Codificación

Grado en Ingeniería Informática

Seminario 4.- Códigos detectores y códigos correctores.

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial