



UNIVERSIDAD  
DE GRANADA

*Este documento está protegido por la Ley de Propiedad Intelectual ([Real Decreto Ley 1/1996 de 12 de abril](#)).  
Queda expresamente prohibido su uso o distribución sin autorización del autor.*

# Teoría de la Información y la Codificación

4º Grado en Ingeniería Informática

## Guión de prácticas Práctica 3

Canales con ruido.  
Códigos detectores y códigos correctores

1. Requisitos.....	2
2. Contenido.....	2
3. Metodología de trabajo.....	2
4. Sesión 1: Plataformas reales y Arduino.....	4
5. Sesión 2: Detección y corrección de errores con códigos de Hamming.....	13
6. Sesión 3: Detección y corrección de errores con códigos lineales.....	18
7. Entrega y evaluación de la práctica.....	19
8. Anexo: Cuestionario de evaluación.....	20

© Prof. Manuel Pegalajar Cuéllar  
Dpto. Ciencias de la Computación e I. A.  
Universidad de Granada



**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**

# Canales con ruido. Códigos detectores y códigos correctores.

## 1. Requisitos

Para la realización de esta práctica es necesario:

- Haber finalizado la práctica 1.
- Haber realizado el "*Seminario 3: Plataformas reales y Arduino*" y el "*Seminario 4: Códigos detectores y códigos correctores*".

## 2. Contenido

Este documento contiene los trabajos a realizar en las sesiones de laboratorio correspondientes a la práctica 3, junto con las preguntas que el alumno debe saber contestar tras la finalización de las sesiones. En este guión de prácticas se desarrollan habilidades relacionadas con la interacción con Arduino a través de sensores externos, tales como receptores de señales de radiofrecuencia, y creación de códigos de Hamming y lineales para detección y corrección de errores. Se continúan desarrollando habilidades introducidas en la práctica 1, tales como el diseño de códigos uniformes, construcción, compilación y envío de programas, transmisión de datos por puerto serie, y se amplían con la construcción y montaje de plataformas para recepción y envío de datos por canales de radio.

Cada una de las secciones siguientes se corresponde con las diferentes sesiones de trabajo en el laboratorio (1 sesión= 2 horas presenciales), y guiarán al alumno en la construcción de códigos detectores y códigos correctores de errores, su implementación en un programa Arduino y su utilización para codificación en plataformas de recepción de datos por radio mediante mandos a distancia, y escritores/lectores de tarjetas RFID. Finalmente, las últimas secciones contienen un conjunto de preguntas que el alumno deberá entregar al profesor, junto con la descripción del método de evaluación.

## 3. Metodología de trabajo

Las prácticas deberán ser realizadas por parejas de estudiantes. Con carácter previo a la realización de este cuaderno de prácticas en el laboratorio, cada estudiante deberá haber estudiado **antes de asistir a clase** el Seminario o los Seminarios asociados al cuaderno de prácticas y, preferiblemente, también las explicaciones disponibles en este documento para la sesión de prácticas correspondiente. Si es posible, también es recomendable haber realizado previamente esquemas, diseños e incluso desarrollos que favorezcan realizar las prácticas de laboratorio de forma más fluida. **No es recomendable asistir a clase de prácticas sin haber estudiado previamente el presente cuaderno de prácticas ni el seminario asociado**, pues se corre el riesgo de no poder terminar a tiempo los trabajos requeridos. Con carácter general, **el estudiante debe tener todo el material preparado para poder dedicar las clases de prácticas íntegramente a implementación de los programas en la plataforma Arduino**.

**Se recomienda seguir la planificación que se ilustra a continuación para asegurar el éxito en la realización de la práctica:**

<b>TAREAS A REALIZAR PARA LA ELABORACIÓN DE LA PRÁCTICA</b>	
<b>Antes de la Sesión 1 en el laboratorio</b>	<ul style="list-style-type: none"> <li>• Estudiar el Seminario 3, <i>Plataformas Reales y Arduino</i>.</li> <li>• Estudiar el Tema 4 de teoría, <i>Codificación en canales con ruido</i>. En especial, la detección de errores por el método de codificación de Hamming.</li> <li>• Estudiar la Sección 4 de este documento.</li> <li>• Realizar pseudocódigo, diagramas o diseños y, si es posible, código fuente, de la tarea final de la sesión.</li> </ul>
<b>Sesión 1</b>	<ul style="list-style-type: none"> <li>• Realización de los montajes y los programas de ejemplo de los módulos PT 2272 y MFRC522 descritos en la Sección 4.</li> </ul>
<b>Antes de la Sesión 2 en el laboratorio</b>	<ul style="list-style-type: none"> <li>• Estudiar el Seminario 4, <i>códigos detectores y correctores</i>. En particular, las secciones 1) <i>Códigos de Hamming para detección</i>; y 2) <i>Códigos de Hamming para corrección</i>.</li> <li>• Estudiar los temas 4 y 5 de teoría. En especial, las secciones dedicadas al diseño de códigos de Hamming.</li> <li>• Estudiar la Sección 5 de este documento</li> <li>• Realizar pseudocódigo, diagramas o diseños y, si es posible, código fuente, de las tareas de la sesión.</li> </ul>
<b>Sesión 2</b>	<ul style="list-style-type: none"> <li>• Realización de los montajes de los módulos PT 2272 y RF2262 descritos en la Sección 4.</li> <li>• Implementar las tareas de la sesión.</li> </ul>
<b>Antes de la Sesión 2 en el laboratorio</b>	<ul style="list-style-type: none"> <li>• Estudiar el Seminario 4, <i>códigos detectores y correctores</i>. En particular, la sección 3) <i>Códigos lineales para detección y corrección</i>.</li> <li>• Estudiar el Tema 5 de teoría. En especial, las secciones dedicadas al diseño de códigos lineales.</li> </ul>
<b>Sesión 2</b>	<ul style="list-style-type: none"> <li>• Realización de los montajes del módulo MF RC522 descrito en la Sección 4.</li> <li>• Implementar el trabajo final de la sesión.</li> </ul>
<b>Antes de la finalización de la entrega de la práctica</b>	<ul style="list-style-type: none"> <li>• Estudiar los temas 4 y 5 de teoría.</li> </ul>

## 4. Sesión 1: Plataformas reales y Arduino

### 4.1. Prerrequisitos

Para elaborar esta sesión es necesario que el alumno haya estudiado previamente todos los apartados del **Seminario 2: Plataformas reales y Arduino**.

Se asume también que el estudiante ha instalado el entorno IDE de Arduino en su PC y ha finalizado la práctica 1 con éxito.

### 4.2. Material necesario

- 2 PCs con IDE Arduino instalado y puerto USB libre.
- 2 placas Arduino Uno (1 placa para el montaje del mando, 1 placa para el montaje del lector/escritor de tarjetas RFID).
- 2 cables USB de conexión Arduino - PC (1 cable para cada Arduino).
- 1 pack de módulos RF2262 / PT 2272 (mando a distancia y receptor de radio).
- 1 pack de módulo MiFare RC522 (lector/escritor de tarjetas RFID) + tarjeta RFID + llavero RFID.
- 7 cables conectores macho/hembra (montaje del receptor del mando a distancia PT2272).
- 7 cables conectores macho/hembra (montaje del lector/escritor de tarjetas RFID).

### 4.3. Instalación de bibliotecas

Se proporcionan dos bibliotecas externas con la API para acceder a los datos de los módulos PT 2272 (fichero **RF2272.zip**) y MF RC522 (**Tarjetero.zip**), cuya instalación es necesaria para la elaboración de la práctica. Para instalar una biblioteca en Arduino desde un fichero .zip hay que seguir las siguientes instrucciones:

1. Abrir el IDE de Arduino
2. Ir al menú **Programa → Incluir Librería → Añadir biblioteca .ZIP**
3. Seleccionar el fichero .zip correspondiente a una biblioteca
4. Pulsar sobre **Aceptar**

Realice esta operación con los dos ficheros de bibliotecas proporcionados en la práctica.

## 4.4. Recepción de datos por radio

En este apartado se realizará el montaje del receptor de radio PT 2272, para obtener la información sobre los botones pulsados en el mando a distancia que contiene el emisor RF2262. Finalmente, se implementará un programa de ejemplo para que el alumno se familiarice con la biblioteca **RF2272.zip** y la adquisición de datos.

### 4.4.1. Montaje

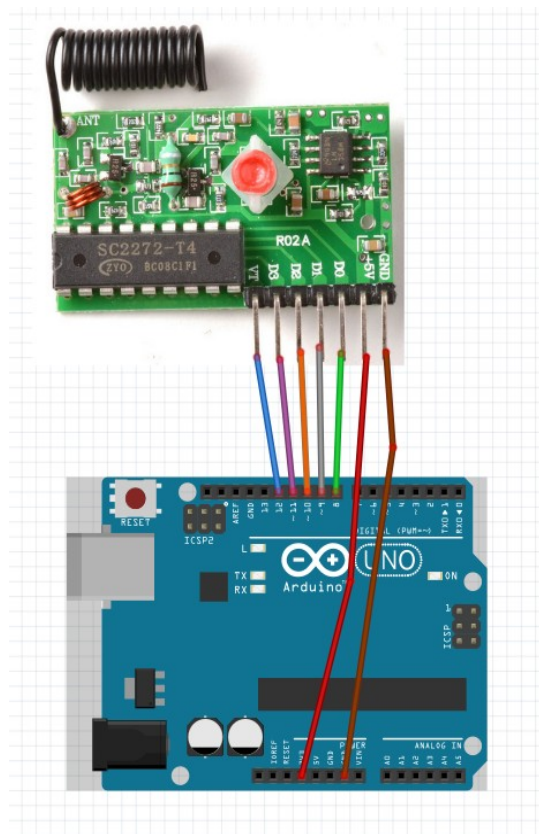
**Antes de comenzar, cree un programa vacío en el IDE Arduino y envíelo a la placa Arduino para eliminar cualquier programa anterior existente, y así evitar dañar las componentes electrónicas del montaje.**

**Posteriormente, desconecte el cable USB de Arduino (en caso de estar conectado) para realizar el montaje.**

Seleccione una placa Arduino, 7 cables de conexión macho/hembra, y un módulo PT2272. Conéctelos siguiendo el esquema siguiente:

- Pin **VT** del módulo PT2272 -> Pin **12** de Arduino
- Pin **D3** del módulo PT2272 -> Pin **11** de Arduino
- Pin **D2** del módulo PT2272 -> Pin **10** de Arduino
- Pin **D1** del módulo PT2272 -> Pin **9** de Arduino
- Pin **D0** del módulo PT2272 -> Pin **8** de Arduino
- Pin **5V** del módulo PT2272 -> Pin **3.3V** o Pin **5V** de Arduino
- Pin **GND** del módulo PT2272 -> Pin **GND** de Arduino

La Figura 1 muestra gráficamente el esquema de montaje, conectando el Pin **5V** del módulo PT2272 al Pin **3.3V** de Arduino.



*Figura 1: Montaje del módulo PT2272*

#### 4.4.2. Biblioteca RF2272

La biblioteca **RF2272** proporcionada en la práctica permite abstraer las comunicaciones con el módulo receptor PT2272, mediante el acceso a una API. Esta API se debe incluir en el programa Arduino mediante el código siguiente:

```
#include "RF2272.h"
```

La biblioteca proporciona las siguientes funcionalidades:


- Función **void IniciarRF2272()**: Inicialización del módulo PT2272. Se debe ejecutar en la función setup() para configurar el módulo receptor.
- Función **uint8\_t leerRF2272()**: Realiza una lectura del receptor de radio PT2272. Devuelve valor 0 si no se detectan datos, o un código de 7 bits de Hamming (7,4) con la siguiente nomenclatura:

Bit	6	5	4	3	2	1	0
Valor	0	0	0	0	0	0	0

Los bits con los valores **A, B, C, D** se corresponden con los botones del mando. Dichos valores tendrán valor 1 si el botón correspondiente está pulsado, y valor 0 en otro caso. Los valores  $P_1$ ,  $P_2$ ,  $P_3$  se corresponden con la paridad par de Hamming. Tendrán valores 0 ó 1 dependiendo del valor de paridad de cada bit.

### 4.4.3. Programa de ejemplo

La Figura 2 muestra un programa de ejemplo de uso de la biblioteca **RF2272**. En particular, la funcionalidad del programa consiste en leer constantemente desde el módulo PT2272 valores recibidos, transformar la cadena de bits en una cadena de caracteres conteniendo "0"s y "1"s, y enviar dicha cadena por el puerto serie al PC.



```
#include "RF2272.h"

void setup() {
  Serial.begin(9600);
  IniciarRF2272();
}

void loop() {

  int v= leerRF2272();
  int bit;
  String codigo= "";

  for (int i=6; i>=0; i--) {
    bit=(1<<i);
    if ((v & bit) > 0) {
      codigo= codigo+'1';
    } else {
      codigo= codigo+'0';
    }
  }

  Serial.println(codigo);
  delay(50);
}
```

*Figura 2: Ejemplo de uso de la biblioteca RF2272*

## 4.5. Lectura/escritura de tarjetas RFID

En este apartado se realizará el montaje del lector/escritor de tarjetas RFID MiFare RC522. Finalmente, se implementará un programa de ejemplo para que el alumno se familiarice con la biblioteca **Tarjetero.zip** y la escritura/adquisición de datos.

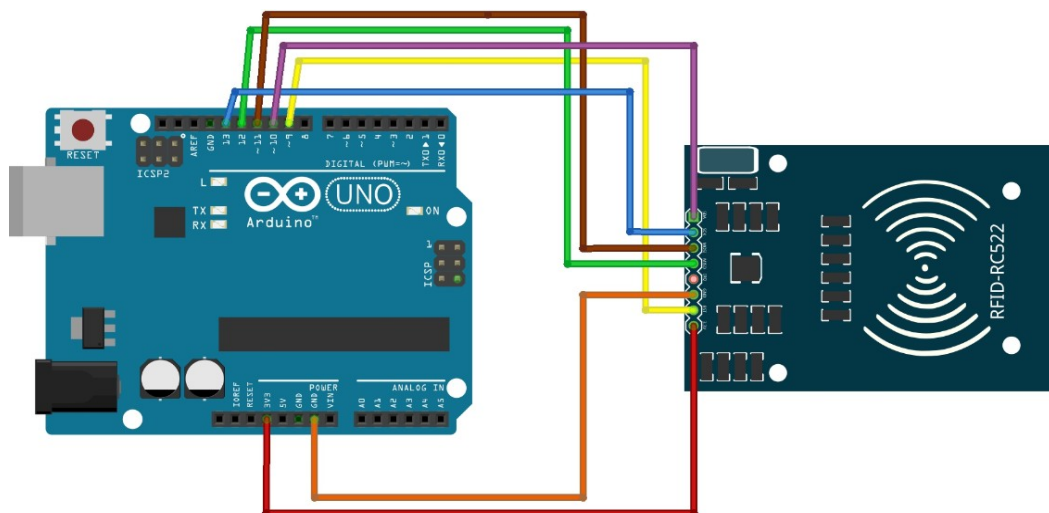
### 4.5.1. Montaje

**Antes de comenzar, cree un programa vacío en el IDE Arduino y envíelo a la placa Arduino para eliminar cualquier programa anterior existente, y así evitar dañar las componentes electrónicas del montaje.**

**Posteriormente, desconecte el cable USB de Arduino (en caso de estar conectado) para realizar el montaje.**

Seleccione una placa Arduino, 7 cables de conexión macho/hembra, y un módulo MFRC522. Conéctelos siguiendo el esquema siguiente:

- Pin **SDA** del módulo MFRC522 -> Pin **10** de Arduino
- Pin **SCK** del módulo MFRC522 -> Pin **13** de Arduino
- Pin **MOSI** del módulo MFRC522 -> Pin **11** de Arduino
- Pin **MISO** del módulo MFRC522 -> Pin **12** de Arduino
- Pin **GND** del módulo MFRC522 -> Pin **GND** de Arduino
- Pin **RST** del módulo MFRC522 -> Pin **9** de Arduino
- Pin **3.3V** del módulo MFRC522 -> Pin **3.3V** de Arduino



*Figura 3: Montaje del módulo MiFare RC522*



La Figura 3 muestra gráficamente la conexión a realizar.

### 4.5.2. Biblioteca Tarjetero

La biblioteca **Tarjetero** proporcionada en la práctica permite abstraer las comunicaciones con el módulo lector/escritor de tarjetas RFID MFRC522, mediante el acceso a una API. Esta API se debe incluir en el programa Arduino mediante el código siguiente:

```
#include "Tarjetero.h"
```

La biblioteca proporciona las siguientes funcionalidades:

- Función **void initMFRC522()**: Inicializa el módulo MF RC522. Se debe usar al comienzo del programa Arduino, en la función setup(), para configurar e iniciar las comunicaciones con el tarjetero.
- Función **bool isCardDetected()**: Devuelve true si detecta una nueva tarjeta cerca y false en otro caso. Se debe usar cuando se quiere acceder a alguna tarjeta. Esta función no sólo comprueba si hay alguna tarjeta cerca, sino que también inicia las comunicaciones con la misma. Siempre debe ser utilizada antes de iniciar operaciones de lectura/escritura.
- Función **bool readUIDCard(char \*uid)**: Lee el UID de la tarjeta cercana al lector y lo devuelve en el parámetro de entrada como cadena (máximo 32 bytes). Devuelve false y cadena vacía "" si no se leyó el UID.
- Función **bool readCard(uint16\_t data[32])**: Lee 64 bytes de la tarjeta (si puede), y los devuelve como un vector de 32 enteros sin signo de 16 bits. En caso de no poder acceder a la tarjeta (corte en las comunicaciones, tarjeta en mal estado, clave de acceso no válida, etc.), devuelve false. Esta función sólo puede ser operativa después de iniciar las comunicaciones con **isCardDetected()** y de haber accedido a su UID mediante **readUIDCard()**.
- Función **bool writeCard(uint16\_t data[32])**: Escribe 64 bytes en la tarjeta (si puede), dados como entrada en un vector de 32 enteros sin signo de 16 bits. En caso de no poder acceder a la tarjeta (corte en las comunicaciones, tarjeta en mal estado, clave de acceso no válida, etc.), devuelve false. Esta función sólo puede ser operativa después de iniciar las comunicaciones con **isCardDetected()** y de haber accedido a su UID mediante **readUIDCard()**.
- Función **closeCard()**: Cierra las comunicaciones con la tarjeta actual activa. Se debe usar cuando se quiere cerrar las comunicaciones con una tarjeta activa con **isCardDetected()** y cuyo UID ha sido obtenido con **readUIDCard()**.

### 4.5.3. Programa de ejemplo

Las ilustraciones Figura 4, Figura 5 y Figura 6 muestran un programa de ejemplo de uso de la biblioteca **Tarjetero**. En particular, la funcionalidad del programa consiste en:

- Esperar a que se inicie el programa de Monitor Serie.

- Pedir al usuario introducir "L" (lectura) o "E" (escritura) en la tarjeta.
- La opción "L" esperará a que se acerque una tarjeta al tarjetero. Se leerá su UID y los datos existentes en la misma. Se enviarán al PC por comunicaciones en serie.
- La opción "E" pedirá al usuario que envíe por el Monitor Serie un número de caracteres no superior a 64. Posteriormente, guardará los datos en todas las tarjetas que se acerquen al tarjetero.

```
#include "Tarjetero.h"
#include <string.h>

// UID de la tarjeta a leer
char UID[32];

// Buffer para escribir/leer datos en/de la tarjeta
char buffer[65];

char opcion; // Opcion a realizar (L=leer/E=escribir)

void setup() {

    // Inicialización del tarjetero
    initMFRC522();

    // Finalización del buffer para poder enviarse por serial
    buffer[64]= '\0';
    opcion= ' ';

    Serial.begin(9600);
    while (!Serial); // Esperamos a que se abra el Monitor Serie
```

*Figura 4: Ejemplo de uso de la biblioteca Tarjetero (I)*

```
Serial.println("Escoja una opción (L=leer/E=escribir)...");
do {
    if (Serial.available()) {
        opcion= Serial.read();
        if (opcion != 'L' && opcion != 'E') {
            Serial.println("Opción no válida (L/E).");
        }
    }
} while (opcion!='L' && opcion != 'E');

if (opcion == 'E') {
    Serial.print("\nIntroduzca el mensaje a escribir (max. 64)...\\n");
    while (!Serial.available());
    Serial.readBytes(buffer, 64);
    Serial.println("Esperando a tarjeta para escribir...");
} else {
    Serial.println("Esperando a tarjeta para leer...");
}
}
```

Figura 5: Ejemplo de uso de la biblioteca Tarjetero (II)

```
void loop() {
    if ( ! isCardDetected())return;

    Serial.print("Tarjeta detectada. ");
    if ( !readUIDCard(UID) )
        return;

    Serial.print(F("UID:"));
    Serial.println(UID);

    if (opcion == 'L') { // Lectura de la tarjeta
        if (!readCard((uint16_t *)buffer)) {
            Serial.println("Error en lectura de tarjeta.");
            closeCard();
            return;
        }
        Serial.print("Datos en la tarjeta: ");
        Serial.println(buffer);
    } else { // Escritura en tarjeta
        if (!writeCard((uint16_t *)buffer)) {
            Serial.println("Error en escritura de tarjeta.");
            closeCard();
            return;
        }
        Serial.println("Datos escritos en la tarjeta.");
    }
    closeCard();
}
```

Figura 6: Ejemplo de uso de la biblioteca Tarjetero (III)

## 4.6. Trabajo final de sesión

### 4.6.1. Material necesario

- Montajes realizados en la sesión.

### 4.6.2. Tarea a realizar

Escriba dos programas Arduino, **Mando.ino** y **LectorTarjetas.ino**, que implementen las funcionalidades descritas en los ejemplos mostrados en los apartados 4.4 y 4.5 de este documento, comentando cada línea de código en la que se haga uso de una de las funciones de las bibliotecas proporcionadas en la práctica. Estos comentarios deberán indicar detalladamente qué realiza cada llamada a función, y cuál es el resultado esperado. Pruebe las funcionalidades en clase con los montajes de cada apartado.

## 5. Sesión 2: Detección y corrección de errores con códigos de Hamming.

### 5.1. Prerrequisitos

Para elaborar esta sesión es necesario que el alumno haya estudiado previamente el **Seminario 4: Códigos detectores y códigos correctores**, apartados:

1. Códigos de Hamming para detección
2. Códigos de Hamming para corrección

Se debe haber finalizado con éxito la Sesión 1 de esta práctica.

### 5.2. Material necesario

- 1 PC con IDE Arduino instalado y puerto USB libre.
- 1 placa Arduino Uno.
- 1 cables USB de conexión Arduino – PC
- 1 pack de módulos RF2262 / PT 2272 (mando a distancia y receptor de radio).
- 7 cables conectores macho/hembra.

### 5.3. Montaje del módulo de recepción de radio PT 2272

**Antes de comenzar, cree un programa vacío en el IDE Arduino y envíelo a la placa Arduino para eliminar cualquier programa anterior existente, y así evitar dañar las componentes electrónicas del montaje.**

**Posteriormente, desconecte el cable USB de Arduino (en caso de estar conectado) para realizar el montaje.**

Siga las instrucciones del apartado 4.4.1 de este documento.

## 5.4. Detección de errores en la recepción de datos transmitidos por radio por el módulo RF2262

### 5.4.1. Cálculo de la detección de errores

En la práctica, se deberá utilizar las funciones **`void initRF2272()`** y **`uint8_t leerRF2272()`** de la API de la biblioteca **`RF2272.zip`** proporcionada, para leer la información recibida por el módulo PT2272 y emitida por el mando a distancia, conteniendo los datos sobre los botones pulsados. La función **`leerRF2272()`** devuelve 1 byte con esta información, como una palabra de un código de Hamming (7,4) organizada en bits como se indica en la siguiente tabla:

Bit	6	5	4	3	2	1	0
Valor	0	0	0	0	0	0	0

Los bits **A**, **B**, **C**, **D** tendrán valor 1 si se ha pulsado el correspondiente botón, y false en caso contrario. Los bits **p<sub>1</sub>**, **p<sub>2</sub>**, **p<sub>3</sub>** son los bits de paridad del código de Hamming. Se puede calcular el valor de cada bit utilizando un operador AND lógico (**&**) entre el código devuelto por la función **`leerRF2272()`** y el bit en cuestión:

- Valor del bit 0: 1 si **`(leerRF2272() & 1) > 0`**; valor 0 en otro caso
- Valor del bit 1: 1 si **`(leerRF2272() & 2) > 0`**; valor 0 en otro caso
- Valor del bit 2: 1 si **`(leerRF2272() & 4) > 0`**; valor 0 en otro caso
- Valor del bit 3: 1 si **`(leerRF2272() & 8) > 0`**; valor 0 en otro caso
- Valor del bit 4: 1 si **`(leerRF2272() & 16) > 0`**; valor 0 en otro caso
- Valor del bit 5: 1 si **`(leerRF2272() & 32) > 0`**; valor 0 en otro caso
- Valor del bit 6: 1 si **`(leerRF2272() & 64) > 0`**; valor 0 en otro caso

Tal y como se explica en el Seminario 4 y en el Tema 4 de teoría, el cálculo de los errores en los bits de paridad puede ser realizado de forma simple mediante operaciones XOR (**⊕**):

- **Error en paridad p<sub>1</sub>**: bit0 ⊕ bit2 ⊕ bit4 ⊕ bit6
- **Error en paridad p<sub>2</sub>**: bit1 ⊕ bit2 ⊕ bit5 ⊕ bit6
- **Error en paridad p<sub>3</sub>**: bit3 ⊕ bit4 ⊕ bit5 ⊕ bit6

Cualquier valor a 1 de los errores anteriores indicará que se ha detectado error en 1 ó 2 bits.

### 5.4.2. Tarea a realizar

Se proporciona al estudiante el esqueleto de la función **`decodificar`** mostrada en la Figura 7. Se debe modificar esta función para que realice la siguiente funcionalidad:

- Calcule errores en los bits de paridad par p<sub>1</sub>, p<sub>2</sub>, p<sub>3</sub>.
- Devuelva true si al decodificar no hubo errores. En tal caso:

- **pulsados[0]** valdrá true si el botón A está pulsado, false en caso contrario.
- **pulsados[1]** valdrá true si el botón A está pulsado, false en caso contrario.
- **pulsados[2]** valdrá true si el botón A está pulsado, false en caso contrario.
- **pulsados[3]** valdrá true si el botón A está pulsado, false en caso contrario.
- Devuelva false si al decodificar se detectaron errores. En tal caso, **pulsados[i]=false** para todo **i**.

```
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;
    |
}
```

*Figura 7: Función de soporte para cálculo de errores en el módulo PT2272*

Una vez implementada correctamente la funcionalidad de la función **decodificar**, la tarea final de esta sección consiste en:

- Realizar un programa de nombre **Deteccion.ino**
- El programa deberá configurar, al inicio, el puerto serie y el módulo PT2272.
- Cuando se presione algún botón, el programa deberá leer el código devuelto por la función **leerRF2272** de la biblioteca RF2272.
- Se deberá llamar a la función **decodificar** desarrollada, con este código como entrada, que devolverá los botones pulsados.
- Finalmente, el programa deberá enviar por puerto serie una cadena con los bits recibidos, seguido de una cadena con los botones pulsados y un mensaje indicando si se detectó error o si no se detectó.
- El comportamiento anterior deberá poder realizarse indefinidamente sin necesidad de reiniciar la plataforma Arduino.

## 5.5. Corrección de errores en la recepción de datos transmitidos por radio por el módulo RF2262

### 5.5.1. Cálculo de la corrección de errores

Los códigos de Hamming permiten corregir errores en 1 bit. Al ser un código lineal, un código de Hamming puede corregirse conociendo el síndrome asociado a una palabra del código recibida. Siendo la matriz del cálculo del síndrome de un código de Hamming (7,4) la mostrada en

la Figura 8, y  $\mathbf{c}=(\mathbf{c}_6, \mathbf{c}_5, \mathbf{c}_4, \mathbf{c}_3, \mathbf{c}_2, \mathbf{c}_1)$  la palabra del código recibida mediante la función **leerRF2272()**, el cálculo del síndrome de la palabra  $\mathbf{c}$ ,  $\mathbf{s}(\mathbf{c})=(\mathbf{p}_3, \mathbf{p}_2, \mathbf{p}_1)$ , se realiza como sigue:

$$\mathbf{s}(\mathbf{c})= \mathbf{c} \cdot \mathbf{H}(\mathbf{C})$$

$$H(C) = \begin{matrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \end{matrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$p_3 \quad p_2 \quad p_1$

*Figura 8: Cálculo del síndrome de un código de Hamming (7,4)*

El valor  $(\mathbf{p}_3, \mathbf{p}_2, \mathbf{p}_1)$  valdrá 0 si no hay error, e indicará el bit (como número binario) donde se encuentra el error en caso contrario. Por ejemplo, un síndrome  $\mathbf{s}(\mathbf{c})=(\mathbf{p}_3, \mathbf{p}_2, \mathbf{p}_1)=(\mathbf{1}, \mathbf{1}, \mathbf{0})$  indicará que se ha detectado un error en el sexto bit, el cual habrá que corregir cambiando su valor de 0 a 1 o de 1 a 0, según corresponda.

### 5.5.2. Tarea a realizar

Se proporciona al estudiante el esqueleto de la función **decodificar** mostrada en la Figura 7. Se debe modificar esta función para que realice la siguiente funcionalidad:

- Calcule el síndrome de una palabra del código recibida como entrada en el argumento **codigo**. **Si el síndrome es distinto de 0, deberá corregirse el error.**
- Devuelva true si al decodificar no hubo errores, o false si se detectaron (y, por tanto, también se corrigieron).
- Tanto si no hubo errores como si hubo y se corrigieron:
  - **pulsados[0]** valdrá true si el botón A está pulsado, false en caso contrario.
  - **pulsados[1]** valdrá true si el botón A está pulsado, false en caso contrario.
  - **pulsados[2]** valdrá true si el botón A está pulsado, false en caso contrario.
  - **pulsados[3]** valdrá true si el botón A está pulsado, false en caso contrario.

Se recomienda, para facilitar las operaciones de esta parte de la práctica y de sesiones posteriores, implementar una función que realice la multiplicación (módulo 2) de un vector por una matriz. El prototipo de esta función podría ser el siguiente:



**`void MultiplicaZ2(int *palabra, int palabraSize,  
int Matriz[7][3], int ColMatriz,  
int *salida);`**

donde:

- **palabra** es la palabra del código recibida, con valores 0/1.
- **PalabraSize** es la longitud de la palabra.
- **Matriz** es una matriz de cálculo de síndromes (en este ejercicio, de tamaño 7 filas y 3 columnas)
- **ColMatriz** es el número de columnas de la matriz.
- salida es el síndrome, de longitud **ColMatriz**.

Una vez implementada correctamente la funcionalidad de la función **decodificar**, la tarea final de esta sección consiste en:

- Realizar un programa de nombre **Correccion.ino**
- El programa deberá configurar, al inicio, el puerto serie y el módulo PT2272.
- Cuando se presione algún botón, el programa deberá leer el código devuelto por la función **leerRF2272** de la biblioteca RF2272.
- Se deberá llamar a la función **decodificar** desarrollada, con este código como entrada, que devolverá los botones pulsados con los errores corregidos, en su caso.
- Finalmente, el programa deberá enviar por puerto serie una cadena con los bits recibidos, seguido de una cadena con los botones pulsados y un mensaje indicando si se detectó y corrigió algún error, o si no se detectó ninguno.
- El comportamiento anterior deberá poder realizarse indefinidamente sin necesidad de reiniciar la plataforma Arduino.

## 6. Sesión 3: Detección y corrección de errores con códigos lineales.

### 6.1. Prerrequisitos

Para elaborar esta sesión es necesario que el alumno haya estudiado previamente el **Seminario 4: Códigos detectores y códigos correctores**, apartado "Códigos lineales para detección y corrección".

Se recomienda haber finalizado con éxito la práctica 1.

Se debe haber finalizado con éxito la Sesión 1 de esta práctica.

### 6.2. Códigos lineales

En la práctica se deberá implementar un código lineal capaz de codificar mensajes procedentes de un alfabeto de la fuente compuesto por 32 símbolos diferentes, de modo que sea capaz de detectar y corregir el máximo número de errores de 1 bit posible. La longitud máxima del código resultante será de 12 bits. Para realizar esta tarea, se deberá:

- Diseñar un código binario uniforme inicial capaz de representar los 32 símbolos del alfabeto de la fuente (total:  $k=5$  bits/palabra del código uniforme).
- Diseñar e implementar la matriz de codificación (matriz generadora), que transforme palabras del código binario uniforme a palabras del código lineal diseñado, con palabras de longitud 12 bits. Por tanto, dicha matriz tendrá un tamaño de 5 filas y 12 columnas.
- Diseñar e implementar la matriz de cálculo de síndromes.
- Diseñar e implementar la tabla de síndromes y sus errores asociados, para un máximo de errores en 2 bits por palabra.

Por tanto, la matriz generadora tendrá la siguiente forma:

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

Dada una palabra del código uniforme inicial  $\mathbf{c}=(\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5)$ , la transformación de esta palabra a su palabra asociada en el código lineal se realizaría mediante la multiplicación  $\mathbf{c} \cdot \mathbf{M}(\mathbf{C})$ .

Además, el alumno deberá encontrar una matriz de paridad de modo que se permita corregir el máximo de errores posible. Con respecto al cálculo de síndromes, la matriz de comprobación de errores tendrá la siguiente forma:

$$H(C)=[I_{7,7}|P_{5,7}^t]=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & p_{1,1} & p_{2,1} & p_{3,1} & p_{4,1} & p_{5,1} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & p_{1,2} & p_{2,2} & p_{3,2} & p_{4,2} & p_{5,2} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & p_{1,3} & p_{2,3} & p_{3,3} & p_{4,3} & p_{5,3} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & p_{1,4} & p_{2,4} & p_{3,4} & p_{4,4} & p_{5,4} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & p_{1,5} & p_{2,5} & p_{3,5} & p_{4,5} & p_{5,5} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & p_{1,6} & p_{2,6} & p_{3,6} & p_{4,6} & p_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & p_{1,7} & p_{2,7} & p_{3,7} & p_{4,7} & p_{5,7} \end{pmatrix}$$

Esta matriz se utilizará para el cálculo de la tabla de síndromes y para el cálculo de la detección de errores tras recibir una palabra del código lineal. En particular, se deberá construir la tabla de síndromes y sus errores asociados como mínimo para detectar y corregir errores en 2 bits. Para ello, se generarán vectores de error  $\mathbf{e}_x$  de 12 bits para 1, 2 errores (como mínimo), y se calculará su síndrome con la multiplicación  $\mathbf{e}_x \cdot \mathbf{H}^t(\mathbf{C})$ , para cada posible error de 1 bit, 2 bits, etc., que el código sea capaz de corregir:

$$\text{Error } \mathbf{e}_1=(000000000001) \rightarrow \mathbf{e}_1 \cdot \mathbf{H}^t(\mathbf{C})=\mathbf{s}_{\mathbf{e}_1}=(s^1_1, s^1_2, s^1_3, s^1_4, s^1_5, s^1_6, s^1_7)$$

$$\text{Error } \mathbf{e}_2=(000000000010) \rightarrow \mathbf{e}_2 \cdot \mathbf{H}^t(\mathbf{C})=\mathbf{s}_{\mathbf{e}_2}=(s^1_1, s^1_2, s^1_3, s^1_4, s^1_5, s^1_6, s^1_7)$$

...

$$\text{Error } \mathbf{e}_{12}=(100000000000) \rightarrow \mathbf{e}_{12} \cdot \mathbf{H}^t(\mathbf{C})=\mathbf{s}_{\mathbf{e}_{12}}=(s^{12}_1, s^{12}_2, s^{12}_3, s^{12}_4, s^{12}_5, s^{12}_6, s^{12}_7)$$

$$\text{Error } \mathbf{e}_{13}=(000000000011) \rightarrow \mathbf{e}_{13} \cdot \mathbf{H}^t(\mathbf{C})=\mathbf{s}_{\mathbf{e}_{13}}=(s^{12}_1, s^{12}_2, s^{12}_3, s^{12}_4, s^{12}_5, s^{12}_6, s^{12}_7)$$

...

$$\text{Error } \mathbf{e}_{xx}=(001000010000) \rightarrow \mathbf{e}_{xx} \cdot \mathbf{H}^t(\mathbf{C})=\mathbf{s}_{\mathbf{e}_{xx}}=(s^{xx}_1, s^{xx}_2, s^{xx}_3, s^{xx}_4, s^{xx}_5, s^{xx}_6, s^{xx}_7)$$

...

$$\text{Error } \mathbf{e}_{yy}=(110000000000) \rightarrow \mathbf{e}_{yy} \cdot \mathbf{H}^t(\mathbf{C})=\mathbf{s}_{\mathbf{e}_{yy}}=(s^{yy}_1, s^{yy}_2, s^{yy}_3, s^{yy}_4, s^{yy}_5, s^{yy}_6, s^{yy}_7)$$

Las parejas  $(\mathbf{e}_1, \mathbf{s}_{\mathbf{e}_1}), (\mathbf{e}_2, \mathbf{s}_{\mathbf{e}_2}), \dots, (\mathbf{e}_{yy}, \mathbf{s}_{\mathbf{e}_{yy}})$  se guardarán para la posterior corrección de errores.

La corrección de errores se realizará de la siguiente forma:

- Sea una palabra del código lineal recibida  $\mathbf{l}=(\mathbf{l}_1, \mathbf{l}_2, \dots, \mathbf{l}_{12})$ , entonces se multiplicará  $\mathbf{l} \cdot \mathbf{H}^t(\mathbf{C})$  para calcular el síndrome  $\mathbf{s}$  asociado a la palabra  $\mathbf{l}$ .

- En la tabla de síndromes, se buscará cuál es el síndrome  $\mathbf{s}_{ei}=\mathbf{s}$ , obteniendo así el error asociado al síndrome,  $\mathbf{e}_i$ .
- Se corregirá el error calculando la palabra corregida como  $\mathbf{l}'=\mathbf{l}+\mathbf{e}_i$ .

## 6.3. Tarea final de sesión

### 6.3.1. Esquema del sistema a implementar

Se desea implementar un programa en Arduino que escriba y lea información de tarjetas RFID usando el módulo MF RC522, usando la biblioteca **Tarjetero.zip** facilitada en la práctica. Por tanto, tendrá dos funcionalidades específicas:

- Escritura de datos:
  - El programa Arduino esperará a que se le envíe por puerto serie (**Monitor Serie**) una secuencia de caracteres (máximo 32 caracteres), de un alfabeto de 32 símbolos formados por:
    - 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'.
  - Si el mensaje fuese inferior a 32 caracteres, se rellenarán todos los símbolos restantes con el símbolo '\0'.
  - El programa Arduino codificará los 32 caracteres anteriores en una secuencia de palabras de un código uniforme de 5 bits. Este código uniforme, así como la función para codificar un símbolo en una palabra del código uniforme, puede basarse en los desarrollos realizados en la práctica 1.
  - A continuación, el programa Arduino deberá codificar cada palabra del código uniforme en el código lineal diseñado, de 12 bits, que sea capaz de corregir un mínimo de 2 errores por palabra.
  - Estas palabras se codificarán en un vector de 32 datos de tipo **uint16\_t**, ocupando los bits menos significativos. El resto de bits sin uso (los 4 más significativos) se deberán dejar a 0.
  - Finalmente, el programa Arduino deberá quedar a la espera de captar tarjetas RFID en su cercanía, y de guardar el vector de datos codificados en todas las tarjetas a su alcance.
- Lectura de datos:
  - El programa Arduino quedará a la espera de captar tarjetas RFID a su alcance. Cuando detecte una tarjeta, leerá un conjunto de 64 bytes, devolviéndolo como un vector de 32 datos de tipo **uint16\_t** mediante la función **readCard** de la biblioteca **Tarjetero.zip**.
  - Los 12 bits menos significativos de cada componente del vector obtenido en el paso anterior serán tratados como una palabra del código uniforme de 12 bits diseñado. Para cada componente del vector, se deberán extraer los 12 bits menos significativos, y calcular el síndrome asociado a la palabra del código.
  - Si el síndrome es distinto de 0, se deberá calcular el error asociado y corregirlo.
  - La palabra se decodificará en el código uniforme inicial de 5 bits, obteniendo los 5 bits

menos significativos de la palabra, según se ha diseñado con la matriz generadora del código.

- Los 5 bits extraídos del código lineal se decodificarán en un símbolo del alfabeto de la fuente. Para decodificar cada palabra, se puede inspirar en los resultados de la práctica 1.
- Todos los símbolos de la fuente decodificados serán enviados a PC mediante puerto Serie, de modo que se pueda visualizar mediante el **Monitor Serie** de Arduino.

Se podrá cambiar entre funcionalidades de escritura/lectura pulsando el botón **reset** de la placa de Arduino, de forma similar a como se ha expuesto en la Sesión 1 de esta práctica.

### 6.3.2. Material necesario

- 1 PC con IDE Arduino instalado y puerto USB libre.
- 1 placa Arduino Uno.
- 1 cables USB de conexión Arduino - PC
- 1 pack de módulo MiFare RC522 (lector/escritor de tarjetas RFID) + tarjeta RFID + llavero RFID.
- 7 cables conectores macho/hembra (montaje del lector/escritor de tarjetas RFID).

### 6.3.3. Montaje

**Antes de comenzar, cree un programa vacío en el IDE Arduino y envíelo a la placa Arduino para eliminar cualquier programa anterior existente, y así evitar dañar las componentes electrónicas del montaje.**

**Posteriormente, desconecte el cable USB de Arduino (en caso de estar conectado) para realizar el montaje.**

Siga las instrucciones del apartado 4.5.1 de este documento para realizar el montaje del lector/escritor de tarjetas RFID MF RC522.

### 6.3.4. Tareas a realizar

Realice un programa Arduino que, en la función **setup()**, inicialice el puerto serie para comunicaciones por el PC mediante **Monitor Serie** y el lector de tarjetas MF RC522. Posteriormente, se pedirá que se introduzca por el **Monitor Serie** la opción a escoger entre "Escribir Tarjeta" o "Leer Tarjeta". El código de la función **loop()** deberá implementar la funcionalidad descrita en el apartado 6.3.1 de este documento.

Para la realización de la práctica, se recomienda modularizar el código en las siguientes funciones:

- Con respecto al escritor:
  - Función **codifica**, que tenga como entrada un símbolo del alfabeto de la fuente y, como salida, una cadena de "0"s y "1"s con la palabra correspondiente del código uniforme. Se puede adaptar la función **codifica** de la práctica 1 al alfabeto de la fuente descrito en este documento.
  - Función **codificaLineal**, que tenga como entrada una palabra del código uniforme, resultante de la función **codifica**, y proporcione como salida la palabra del código lineal diseñado.
  - Función **BytesToUint16**, que tenga como entrada una palabra del código lineal, resultante de la función **codificaLineal**, y codifique dicha palabra a nivel de bits en un dato de tipo **uint16\_t** como salida, en los 12 bits menos significativos.
- Con respecto al lector
  - Función **Uint16ToBytes**, que tenga como entrada un dato de tipo **uint16\_t**, y pase a cadena de "0"s y "1"s los 12 bits menos significativos del dato de entrada. Esta función debe realizar la operación inversa de la función **BytesToUint16**.
  - Función **decodificaLineal**, que tenga como entrada una palabra del código lineal creado, y proporcione como salida una cadena de "0"s y "1"s con la palabra del código uniforme inicial correspondiente. Esta función debe realizar la operación inversa de la función **codificaLineal**.
  - Función **decodifica**, que tenga como entrada una cadena de "0"s y "1"s con una palabra de un código uniforme, y proporcione como salida un carácter del alfabeto de la fuente. Esta función debe realizar la operación inversa de la función **codifica**. Se puede adaptar la función **decodifica** de la práctica 1 al alfabeto de la fuente descrito en este documento.
  - Función **CalculaSindrome**, que tenga como entrada una cadena de "0"s y "1"s conteniendo una palabra del código lineal y, como salida, proporcione su síndrome asociado.

Como idea general, la función **CalcularSindrome** podría ser llamada desde el interior de la función **decodificaLineal**, para detectar y corregir los errores de en la palabra del código lineal de entrada a dicha función.

Como ayuda, se proporciona con la práctica el código del proyecto Arduino **LectorTarjetas**, con gran parte del esqueleto del programa implementado, para que el alumno pueda centrarse en la construcción de los códigos y la codificación/decodificación de palabras.

## 7. Entrega y evaluación de la práctica

La práctica debe ser resuelta **por parejas, y contribuirá con 1.5 puntos sobre 5 (prácticas) a la calificación global de la asignatura.** La evaluación principal será continua, y se realizará por sesiones o al finalizar la práctica. Se evaluarán los siguientes items:

- Funcionamiento del ejercicio final de la sesión 2: 4 puntos.
- Funcionamiento del ejercicio final de la sesión 3: 3 puntos.

El funcionamiento de los ejercicios anteriores se evaluará en el laboratorio. Consistirá en una inspección y entrevista del profesor con los estudiantes, donde se tratarán y se deberán resolver las siguientes cuestiones:

- Prueba del correcto funcionamiento esperado del sistema (mitad de la puntuación).
- Explicación de cómo se ha implementado cada parte (mitad de la puntuación).

Adicionalmente, al finalizar la práctica se deberá presentar al profesor, mediante la entrega habilitada en la plataforma docente web de la asignatura, la siguiente documentación:

- Código fuente final de cada una de las sesiones (salvo la primera) de la práctica. El código fuente se organizará por carpetas, denominadas "S2" (código de la sesión 2), "S3" (código de la sesión 3). **La no entrega del código supondrá la calificación numérica 0 en el apartado correspondiente, independientemente de la defensa realizada en clase de laboratorio.**
- **Entrega del cuestionario del anexo de este documento**, en formato PDF. Todas las preguntas del cuestionario se valorarán con la misma calificación. La calificación total del cuestionario en la práctica es de 3 puntos.

## 8. Anexo: Cuestionario de evaluación

<b>Nombre, apellidos y email del estudiante 1:</b>	
<b>Nombre, apellidos y email del estudiante 2:</b>	
<b>CUESTIONARIO</b>	
<b>Explique qué es un código de Hamming (7,4), cuántos errores puede detectar y cuántos errores puede corregir, justificando su respuesta.</b>	
<b>Explique cómo detectar errores en un código de Hamming (7,4)</b>	
<b>Explique cómo corregir errores en un código de hamming (7,4)</b>	



**El código lineal diseñado en la parte práctica de la sesión 3 es un código  $(n, M, d)$ . ¿Qué valores  $n$ ,  $M$  y  $d$  tiene? ¿Qué significa cada valor?**

**Escriba la matriz de generación del código lineal diseñado en la sesión 3**

**Basándose en la matriz del código lineal diseñado en la sesión 3, indique y justifique cuántos errores se pueden detectar y corregir con el código generado por la matriz**

**Exponga la matriz de cálculo de errores diseñada en la sesión 3**

**Indique cómo se ha construido la tabla de síndromes para el cálculo y la corrección de errores en la sesión 3**

**Exponga un ejemplo de palabra del código lineal con errores en 1 bit. Explique, con dicho ejemplo, cómo se calcula el síndrome, su error asociado, y cómo se corregiría dicho error.**