



Universidad de Granada

[decsai.ugr.es](http://decsai.ugr.es)

# Teoría de la Información y la Codificación

## Grado en Ingeniería Informática

Seminario 3.- Plataformas reales y Arduino.



**DECSAI**

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



UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

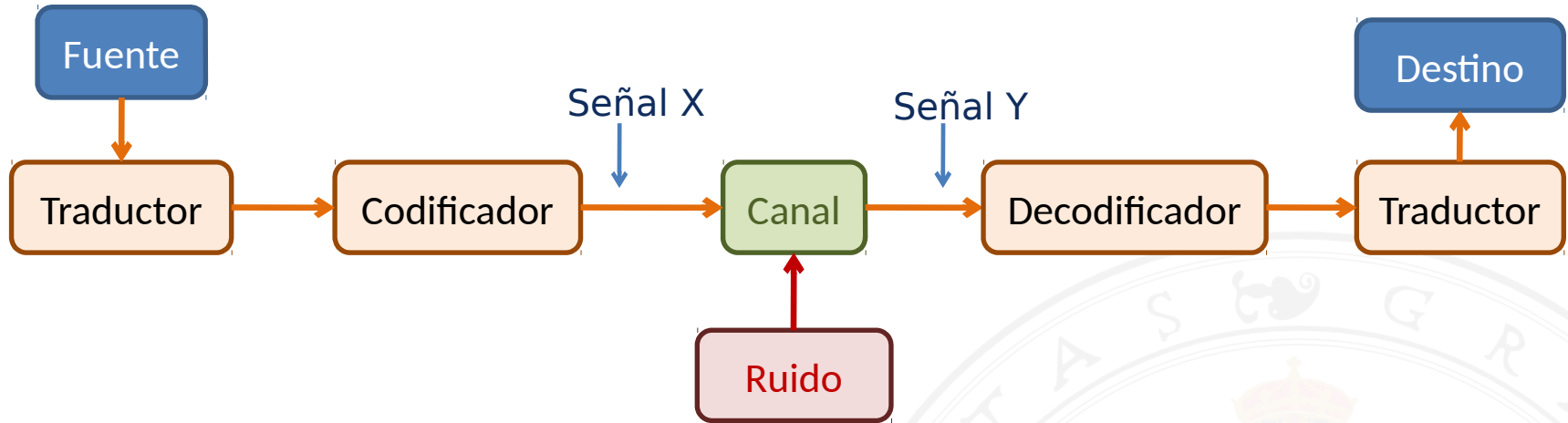


1. Codificación en canales con ruido
2. Módulos RF2262 / 2272
3. Módulo MFRC522



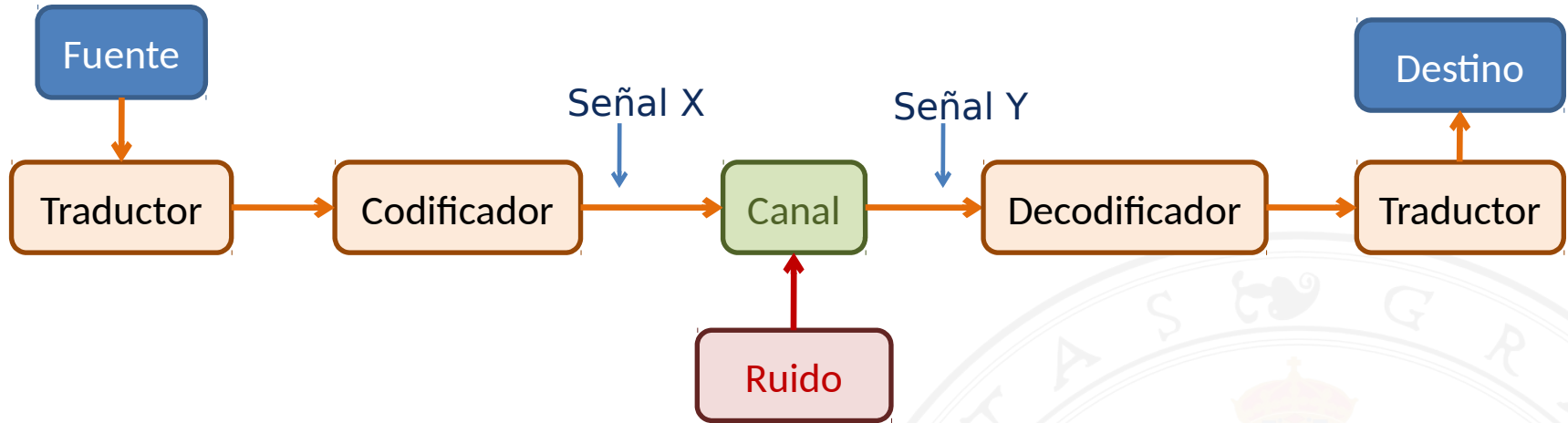
DECSAI

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



- En nuestro sistema, la **fuente** será un dispositivo que emite información (mando a distancia, tarjetas RFID...).
- El **traductor y el codificador** Estarán en el dispositivo, y se encargarán de crear los códigos y transmitirlos.
- El **Decodificador y el traductor del respecto** será un Arduino, equipado con un dispositivo sensor. Se encargará de leer la información recibida y pasarla a Arduino mediante protocolos de comunicaciones.
- El **destino** será un PC, que recibe por USB los códigos dados por Arduino.

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



- El ruido dará como resultado la modificación aleatoria en uno o varios bits durante la recepción de datos y su transmisión entre la fuente y Arduino.

## – ¿Qué es un código de bloque?

- Son códigos uniformes de  $n$  bits.
- Se generan a partir de otros códigos uniformes de longitud menor  $k$ .
- A un código uniforme de longitud  $k$ , se le añaden  $r$  bits adicionales de redundancia, de modo que:

$$n = k + r$$

- Las plataformas a usar en esta práctica trabajarán sobre códigos de bloque, que incluyan redundancia al código original para que se puedan detectar y corregir errores en el receptor.

### – Ejemplos cotidianos de aplicaciones de transmisión en canales con ruido





UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. Codificación en canales con ruido
- » 2. Módulos RF2262 / 2272
3. Módulo MFRC522

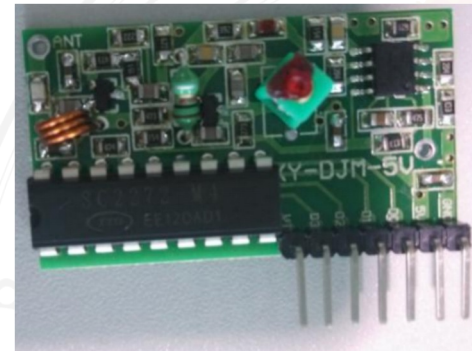


DECSAI

- Los módulos **RF 2262/2272** son un emisor (RF 2262) y receptor (PT 2272) de señales de radio.
- El emisor es capaz de enviar señales a 315MHz, usando 4 canales.
- El receptor lleva un microchip que decodifica las señales de radio y las traduce en los botones que se han pulsado.



**Emisor PT 2262**



**Receptor RF 2272**

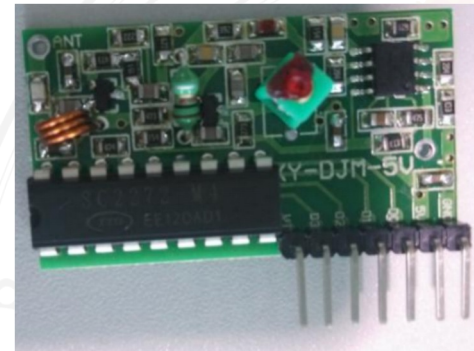


## – Funcionamiento:

- Al pulsar uno de los botones en el módulo PT2262, se envía una señal de radio a 315MHz
- El receptor está muestreando la frecuencia de radio de 315MHz. Cuando recibe un paquete de datos del emisor, se decodifica el paquete y se traduce la señal a un conjunto de bits que indican qué botones se están pulsando.

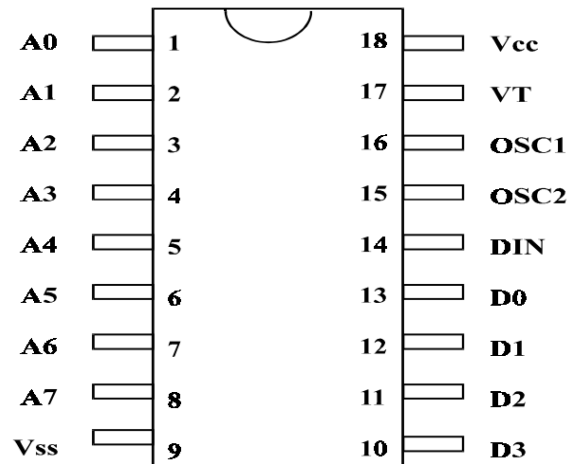
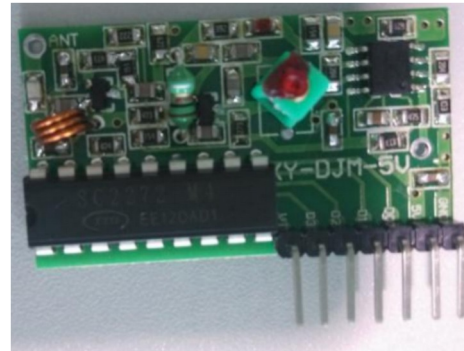


**Emisor PT 2262**

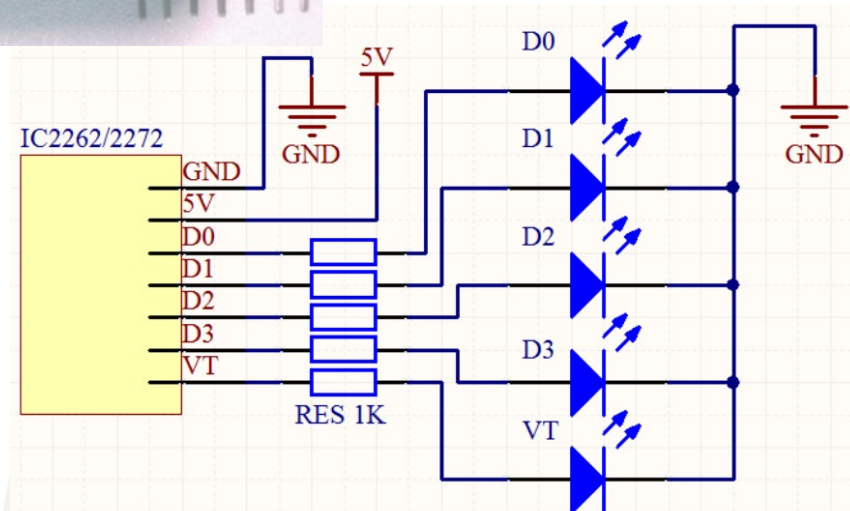


**Receptor RF 2272**

- El receptor tiene un esquema de patillaje como el siguiente:



**PT 2272 - M4/L4**

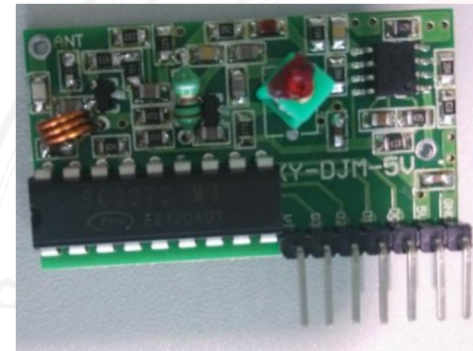


- Pin VT: Entrada a Arduino. A HIGH si está recibiendo datos
- Pines D3-D0: Entrada a Arduino. A HIGH si los botones D, C, B, A han sido pulsados
- 5V: Conexión de corriente a 5V
- GND: Conexión de corriente a tierra

- Programar directamente el microchip del receptor requiere:
  - Conocimientos de procesamiento de señales analógicas.
  - Protocolo de comunicaciones usado a nivel interno por los dispositivos.
- En su lugar, programaremos directamente Arduino con los módulos RF2262 y PT2272 que ya se obtienen montados en el mercado.

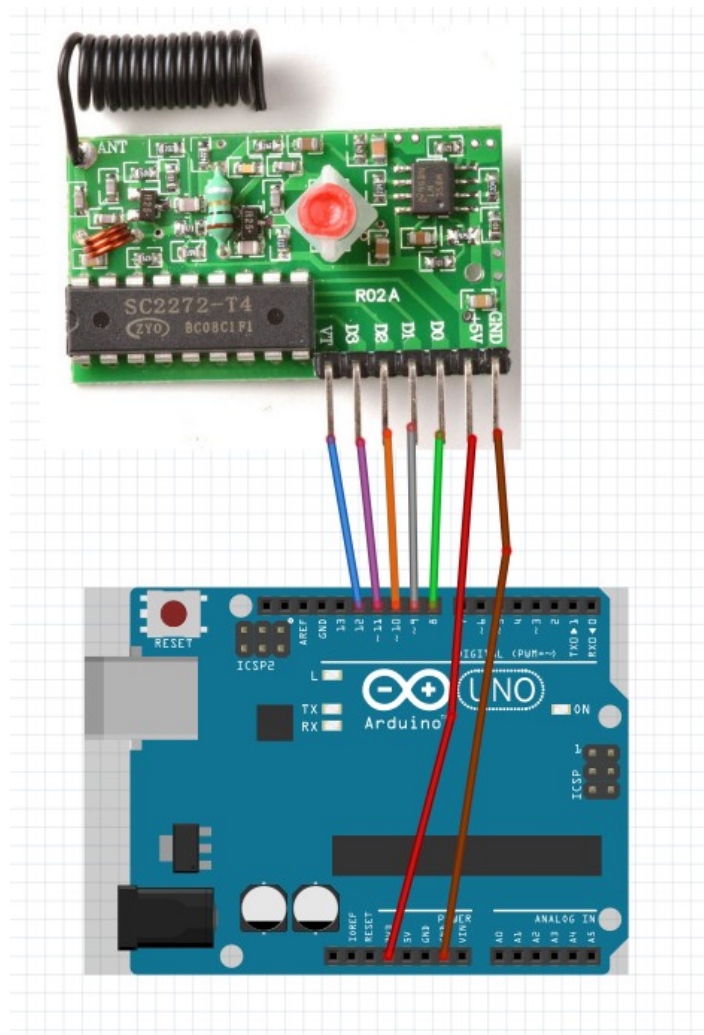


**Emisor PT 2262**



**Receptor RF 2272**

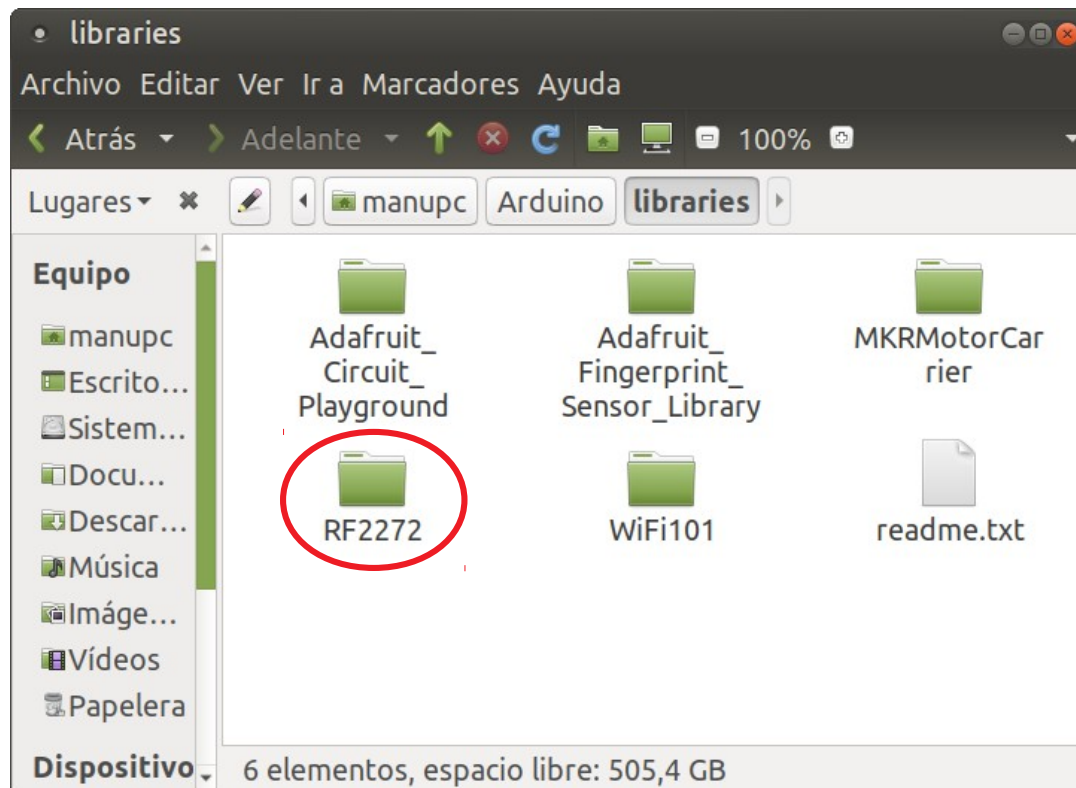
### — Esquema de montaje:



- La biblioteca **RF2272.zip** proporcionada en prácticas ha sido adaptada para poder trabajar con códigos detectores y correctores de errores, utilizando los módulos 2262 y 2272.
- 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 **RF2272.zip** proporcionado en la práctica.
  - 4) Pulsar sobre **Aceptar**



- Podemos ver las bibliotecas que tenemos instaladas en nuestro sistema en la carpeta dedicada de **Arduino**, subcarpeta **libraries**.



- La biblioteca **RF2272** proporcionada en prácticas ha sido adaptada para poder trabajar con códigos detectores y correctores de errores, utilizando los módulos 2262 y 2272:

**#include "RF2272.h"**

La probabilidad de detección de errores es muy baja. Para poder realizar las prácticas, la biblioteca permite la simulación de errores, controlando la probabilidad de que una trama de datos recibida contenga error. Por defecto, genera errores en 1 bit con probabilidad de 0.1.

- Echemos un vistazo al fichero **RF2272.h**:

```
void IniciarRF2272( int _pinD0= PIN_D0, int _pinD1= PIN_D1,  
                  int _pinD2= PIN_D2, int _pinD3= PIN_D3,  
                  int _pinVT= PIN_VT, float pError=0.1);
```

Función ***IniciarRF2272***: Inicialización del módulo PT2262. Se debe ejecutar en la función ***setup()*** para configurar el módulo receptor.

**Los valores por defecto deben no ser alterados, pues están configurados para el montaje diseñado en las prácticas.**



— Echemos un vistazo al fichero **RF2272.h**:

/\*\*

\* 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 la 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)

\*/

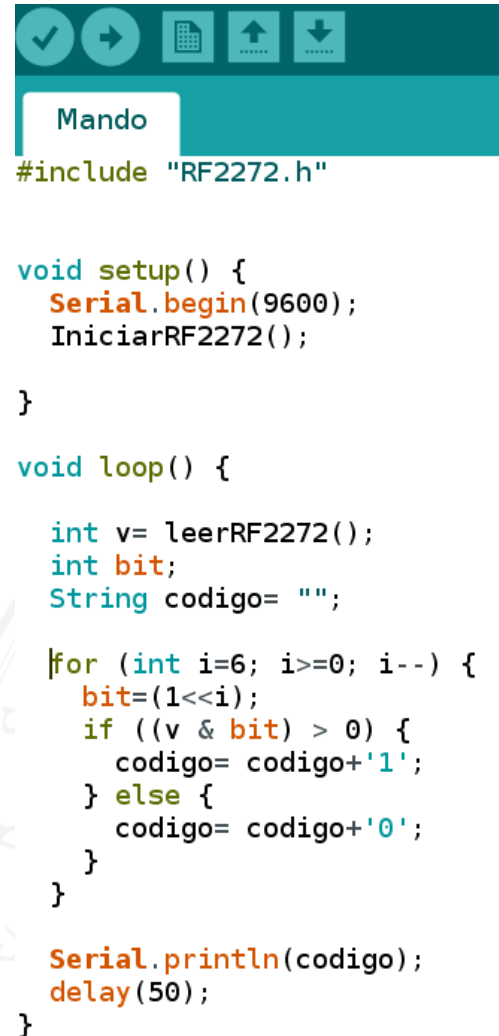
**uint8\_t leerRF2272();**

- Los bits leídos por *leerRF2272()* se corresponden con una palabra de un código de Hamming(7,4), con el siguiente formato:

Bit	6	5	4	3	2	1	0
Valor	D	C	B	$P_3$	A	$P_2$	$P_1$

- 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.
- El ejemplo de la diapositiva siguiente muestra cómo acceder al valor de cada bit.**

- Un ejemplo de su uso: Programa que lee una palabra del código desde el módulo PT2272, la transforma a cadena de caracteres y la envía por puerto serie:



```

Mando
#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);
}
  
```

Ejemplo de función que tiene como entrada un **codigo**, lectura de la función **leerRF2272()**, y proporciona como salida un vector de tipo **bool** indicando si se ha pulsado o no el botón **A** (pulsados[0]), **B** (pulsados[1]), **C** (pulsados[2]) o **D** (pulsados[3]):

```
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;
}
```

En las prácticas, habrá que modificar esta función para que devuelva **true** si no hay errores, o **false** en caso contrario.



UNIVERSIDAD  
DE GRANADA

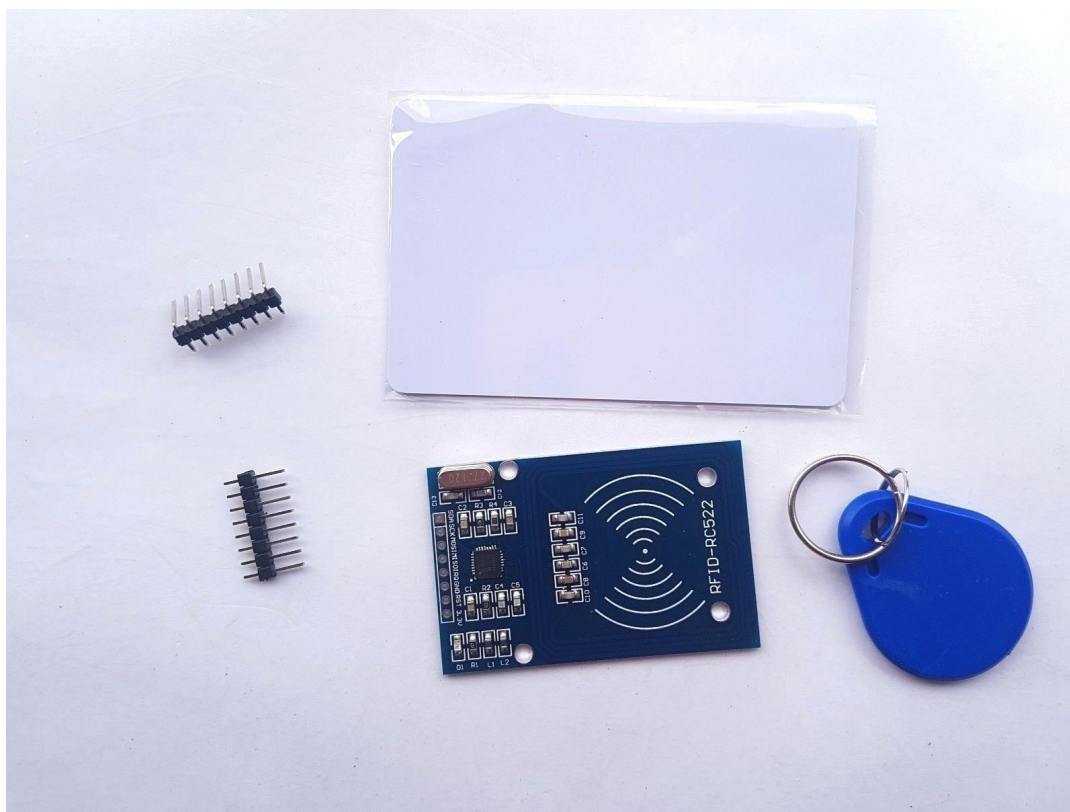
# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. Codificación en canales con ruido
2. Módulos RF2262 / 2272
- » 3. Módulo MFRC522



- El módulo **MF RC522** es un lector/escritor de tarjetas y dispositivos RFID.



- **RFID** (Radio Frequency Identification) es un sistema para transferir datos en distancias cortas (unos pocos centímetros).
- Se usan **dos dispositivos**: Lector/escritor, y dispositivo pasivo (tarjeta)

- **Ejemplos:**

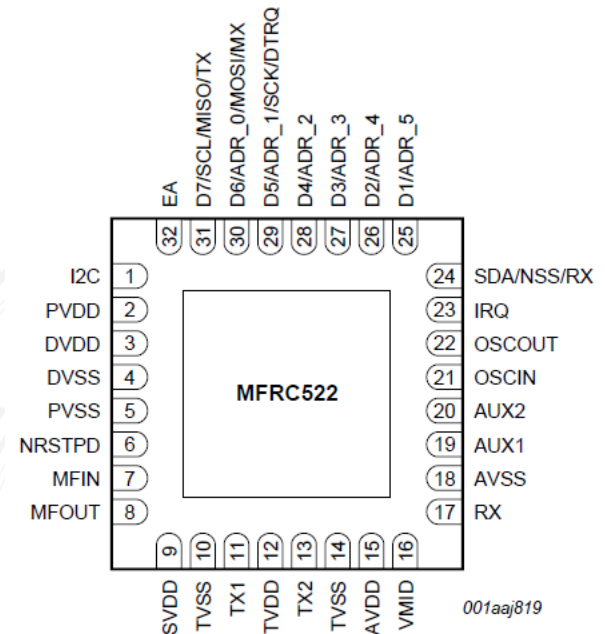
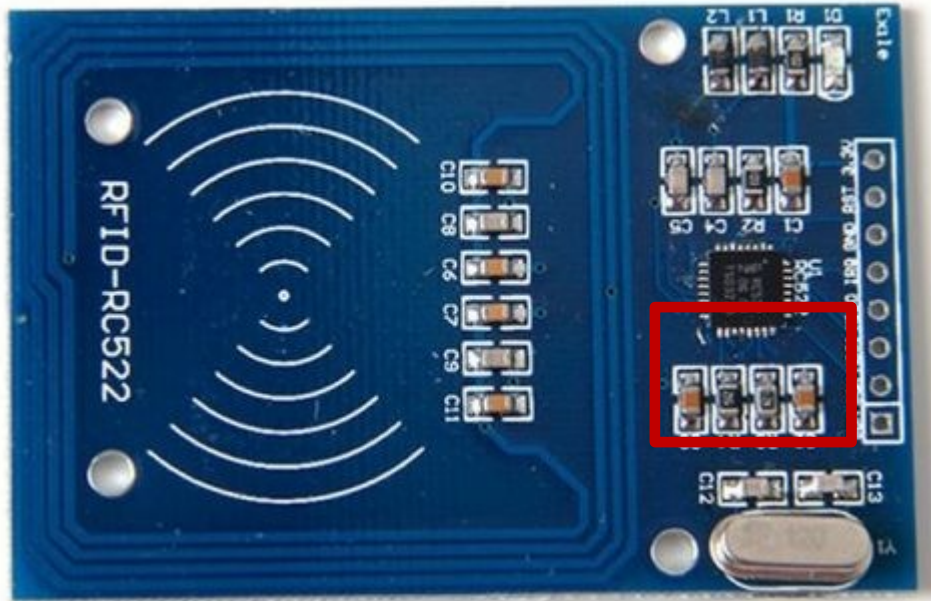
- Collares de mascotas
- Tarjetas de identificación
- Tarjetas de crédito
- Móviles (NFC)
- Smart watches (NFC)



- **Sólo el lector/escritor debe estar alimentado con corriente.**
- El dispositivo pasivo se excita eléctricamente por inducción. El lector/escritor transmite ondas que son percibidas por el dispositivo pasivo y utilizadas como fuente de alimentación.



- Al haber 2 dispositivos, también existen en nuestro sistema 2 microchips:
- **PCD**: El lector/escritor de tarjetas propiamente dicho.



- **PICC**: la tarjeta o dispositivo pasivo.



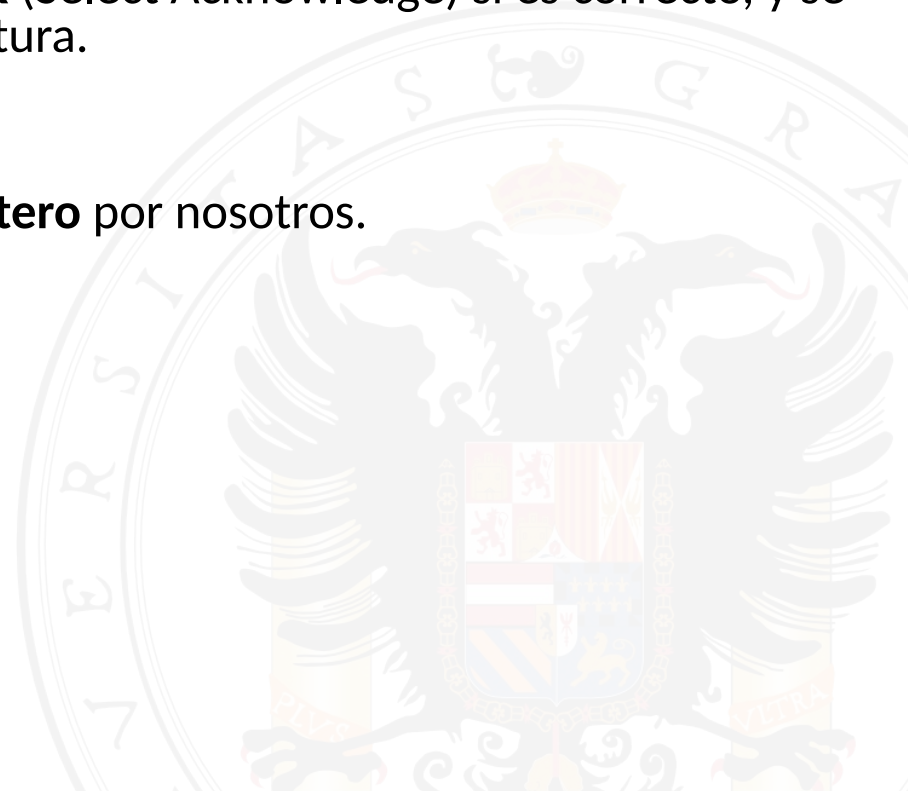
- Al haber 2 dispositivos, también existen en nuestro sistema 2 microchips:
  - **PICC:** la tarjeta o dispositivo pasivo.



- Todo PICC debe tener una *dirección* o identificador UID, para enviar o recibir datos del dispositivo concreto en caso de haber varios
- **UID: Unique Identifier. Secuencia de bits que identifican al dispositivo (algo parecido a la MAC de las tarjetas de red)**

- ¿Cómo se lee el UID de una tarjeta?
  - EL PCD realiza la petición de UID al PICC
  - El PICC devuelve su tamaño y los bits del UID
  - El PCD vuelve a preguntar al PICC si el UID recibido es correcto.
  - El PICC devuelve un byte **SAK** (Select Acknowledge) si es correcto, y se activa para su lectura o escritura.

Todo esto lo hace la biblioteca **Tarjetero** por nosotros.



## ¿Cómo se almacena y organiza la información en un PICC?

- Las tarjetas, por norma general pueden almacenar hasta 1KB de datos.
- La memoria se divide en sectores, que a su vez se dividen en bloques de 16 bytes.
- Por tanto, una tarjeta de 1KB tiene 16 sectores. Cada sector tiene 4 bloques. Cada bloque 16 bytes.

Sector	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Block (1 block = 16bytes)	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

CSN

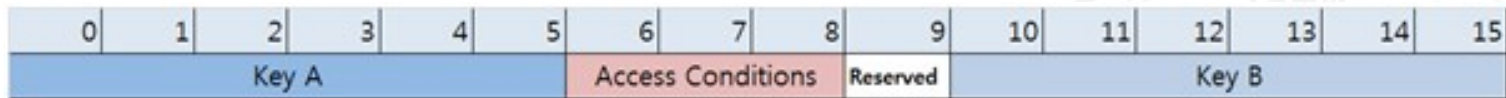
CIS

Data

Site Key

## ¿Cómo se almacena y organiza la información en un PICC?

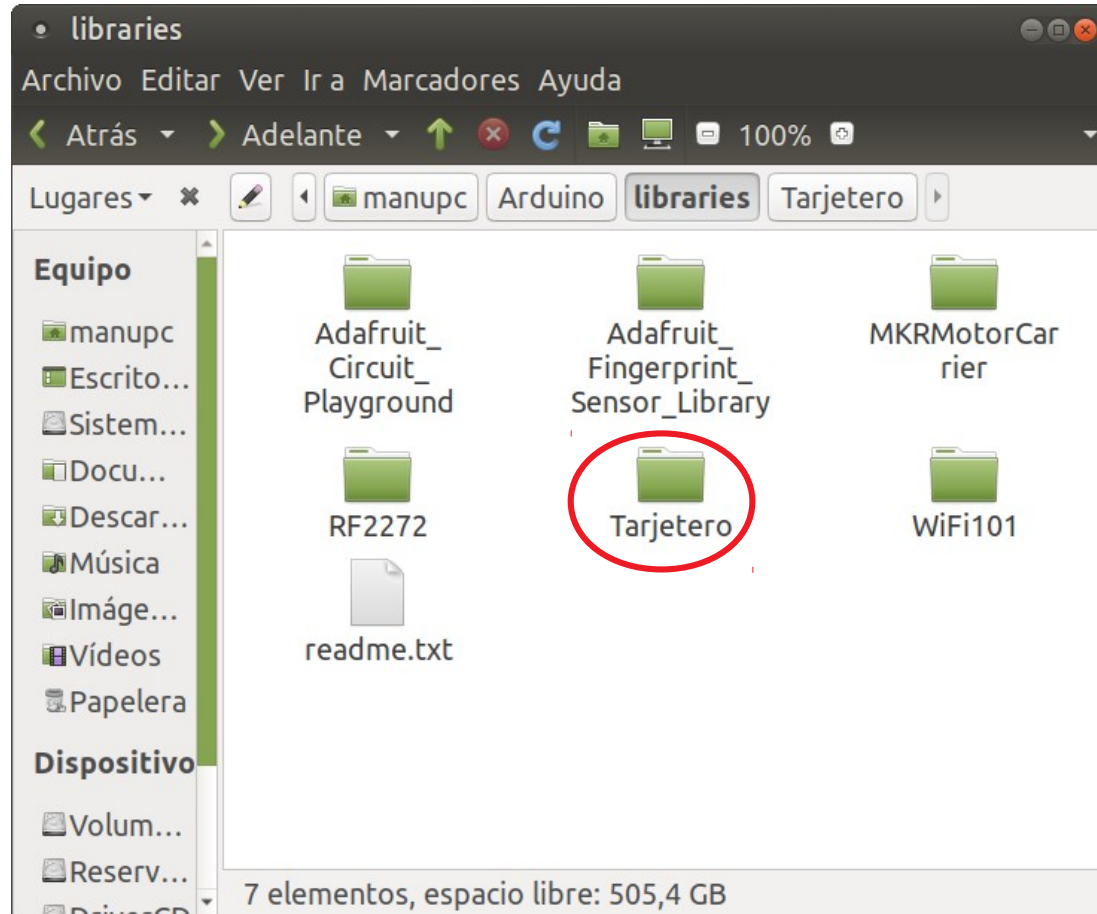
- La información y el acceso a los datos está encriptado y protegido por clave (**site key**), **que se almacena en el último bloque de cada sector.**
- También se tiene información sobre qué tipo de acciones se pueden hacer en cada bloque (lectura, escritura, o lectura/escritura).



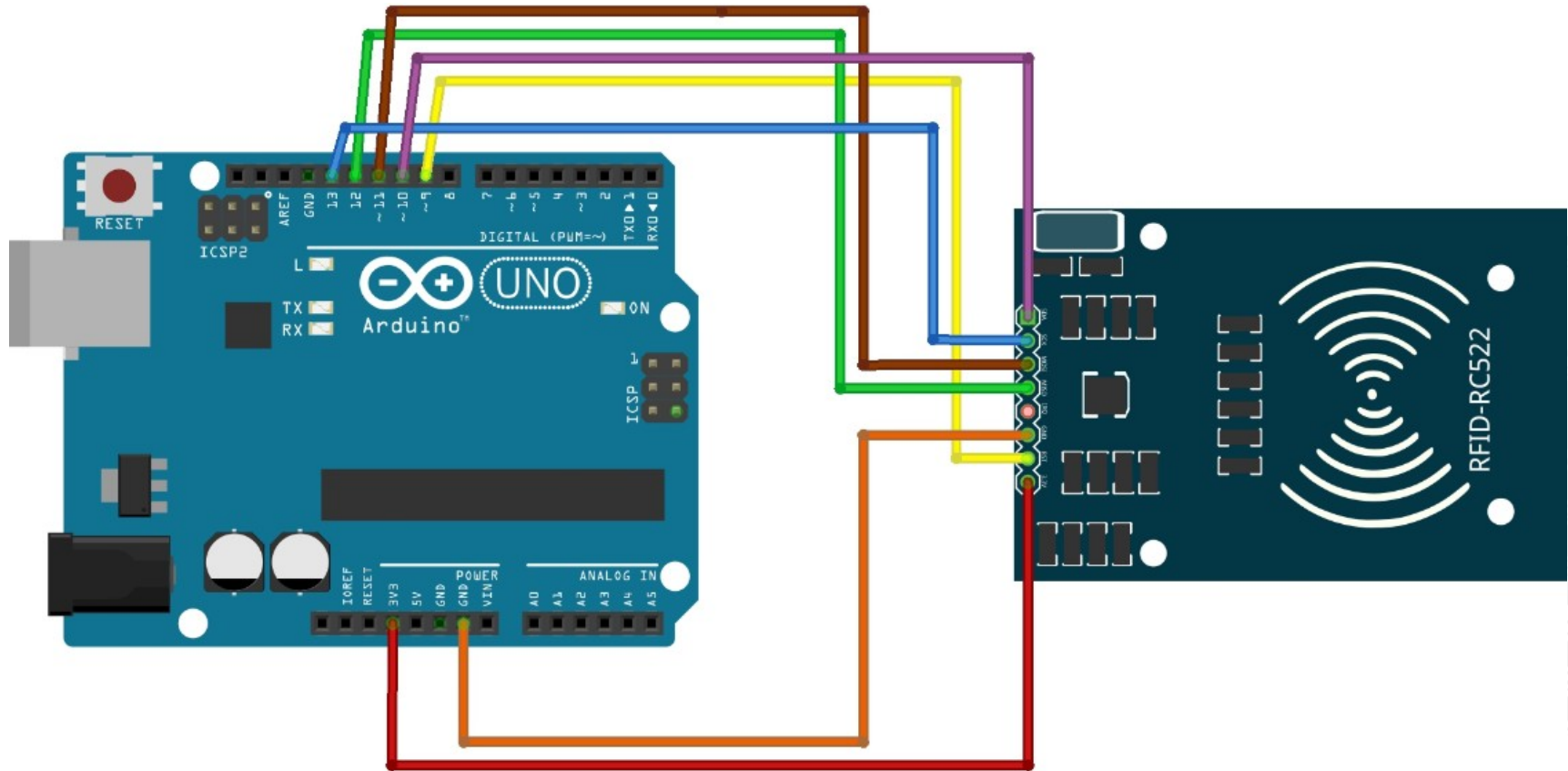
- A la hora de escribir, **hay que tener especial cuidado de no escribir en los bytes reservados a Site Key**, porque así **estaríamos cambiando la clave de acceso a los bloques**, y no podríamos acceder en un futuro.
- La biblioteca **Tarjetero.zip** desarrollada en la asignatura para las prácticas previene todas estas situaciones
- Se ha limitado a escribir 4 bloques (1, 2, 4, 5): 64 bytes.
- Sólo utiliza la clave Key A.

- La biblioteca **Tarjetero.zip** proporcionada en prácticas ha sido adaptada para poder trabajar con códigos detectores y correctores de errores.
- Además, abstrae los protocolos para detección de tarjetas, escritura y lectura.
- La probabilidad de error al leer/escribir es muy baja, por lo que la biblioteca introduce artificialmente errores en bits para poder verificar el funcionamiento de los algoritmos desarrollados.
- 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 **Tarjetero.zip** proporcionado en la práctica.
  - 4) Pulsar sobre **Aceptar**

- Podemos ver las bibliotecas que tenemos instaladas en nuestro sistema en la carpeta dedicada de **Arduino**, subcarpeta **libraries**.



- Usa transmisión por protocolo **SPI** (Serial Port Interface)





- La biblioteca **Tarjetero.zip** proporcionada facilita la siguiente interfaz:

// Inicializa el módulo MF RC522 con probabilidad de errores para 2 bits  
***void initMFRC522(float pError1=0.1, float pError2=0.2);***

- Se debe usar al comienzo del programa Arduino, para configurar e iniciar las comunicaciones con el tarjetero.



- La biblioteca **Tarjetero.zip** proporcionada facilita la siguiente interfaz:

// Devuelve true si detecta una nueva tarjeta cerca. False en otro caso  
***bool isCardDetected();***

- 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.**

- La biblioteca **Tarjetero.zip** proporcionada facilita la siguiente interfaz:

```
// Lee el UID de la tarjeta cercana al lector y lo devuelve como cadena
// (max 32 bytes)
// Devuelve false y cadena vacía "" si no se leyó el UID
bool readUIDCard(char *uid);
```

- Se debe usar cuando se quiere acceder a alguna tarjeta. Esta función sólo puede ser operativa después de iniciar las comunicaciones con **isCardDetected()**.
- La función **readUIDCard** siempre debe ser utilizada antes de iniciar operaciones de lectura/escritura, pues guarda internamente el UID de la tarjeta para posteriores comunicaciones.

- La biblioteca **Tarjetero.zip** proporcionada facilita la siguiente interfaz:

// Lee 64 bytes de la tarjeta

***bool readCard(uint16\_t data[32]);***

- Se debe usar cuando se quiere acceder a alguna tarjeta para leer sus datos. Esta función sólo puede ser operativa después de iniciar las comunicaciones con ***isCardDetected()*** y de haber accedido a su UID mediante ***readUIDCard***.
- 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***.
- Posteriormente, el vector podrá ser interpretado como un vector de caracteres o de bytes con el correspondiente casting (***char \****) o (***byte \****)

- La biblioteca **Tarjetero.zip** proporcionada facilita la siguiente interfaz:

*// Escribe 64 bytes en la tarjeta*

***bool writeCard(uint16\_t data[32]);***

- Se debe usar cuando se quiere acceder a alguna tarjeta para escribir datos en ella. Esta función sólo puede ser operativa después de iniciar las comunicaciones con ***isCardDetected()*** y de haber accedido a su UID mediante ***readUIDCard***.
- **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*.**
- **El vector de entrada puede ser un vector de tipos *char* o *byte*, al que se le ha hecho el correspondiente casting a *(uint16\_t \*)***

- La biblioteca **Tarjetero.zip** proporcionada facilita la siguiente interfaz:

// Cierra las comunicaciones con la tarjeta actual activa

***void closeCard();***

- Se debe usar cuando se quiere cerrar las comunicaciones con una tarjeta activa con ***isCardDetected()*** y cuyo UID ha sido obtenido con ***readUIDCard***.

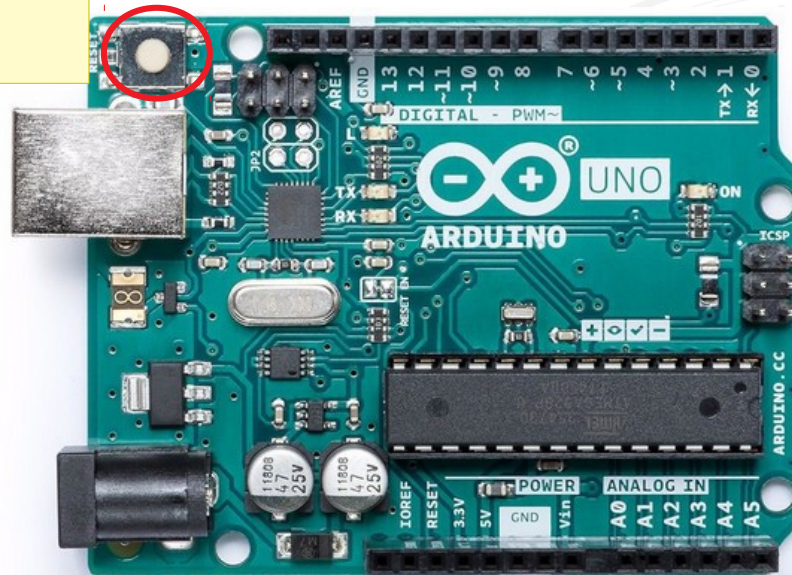
– **Un ejemplo de su uso:**

- Programa que inicia comunicaciones serie y espera a que se inicie el programa de **Monitor Serie**.
  - El programa pide al usuario introducir “L” (lectura) o “E” (escritura) en la tarjeta.
  - La opción “L” esperará a que se acerquen tarjetas 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.

## – Un ejemplo de su uso:

- Se puede reiniciar el programa utilizando el botón **reset** de la placa de Arduino, y así cambiar la funcionalidad de uso del programa.

Botón **reset**





- Código fuente del programa *LectorTarjetas (I): Variables globales*

## LectorTarjetas

```
#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)
```



- Código fuente del programa *LectorTarjetas (II): Función setup()*

## LectorTarjetas

```
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

    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');
```

- Código fuente del programa *LectorTarjetas (III): Función setup()* (continuación)

## LectorTarjetas

```

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...");
}
}

```

- Código fuente del programa *LectorTarjetas (IV): Función loop()*

## LectorTarjetas

```
void loop() {

    if (opcion != 'L' && opcion != 'E') return;

    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);
    }
}
```



- Código fuente del programa *LectorTarjetas (V): Función loop()* (continuación)

## LectorTarjetas

```

} 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();
}

```



Universidad de Granada

[decsai.ugr.es](http://decsai.ugr.es)

# Teoría de la Información y la Codificación

## Grado en Ingeniería Informática

Seminario 3.- Plataformas reales y Arduino.



**DECSAI**

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