[Trabajo Práctico N° 9](https://github.com/ISPC-TST-SyA-2024/Grupo-N-1/tree/main/TP%20N%C2%BA9)

**Módulo 3**

Módulo III: Visualizadores – Protocolos – Interfaces de E/S

**Profesor**

[Jorge Elias Morales](https://github.com/JorEl057)

**Integrantes**

Marcos Bordón Rios - [Marcos-BR-03](https://github.com/Marcos-BR-03)

Fernando Gimenez Coria - [FerCbr](https://github.com/FerCbr)

Karina Jazmin Barbero - [karina-lolis](https://github.com/karina-lolis)

Nicolás Barrionuevo - [NicolasBa27](https://github.com/NicolasBa27)

Macarena Aylen Carballo - [MacarenaAC](https://github.com/MacarenaAC)

Raul Jara - [r-j28](https://github.com/r-j28)

Diego Ezequiel Ares - [diegote7](https://github.com/diegote7)

Juan Diego Gonzaléz Antoniazzi - [JDGA1997](https://github.com/JDGA1997)

**Fecha de entrega**

Miércoles 23 de Octubre 2024

**Índice**

[**Ejercicio Nº1 3**](#_evl9oaej4k5w)

[1. Implementen una simulación de una Conexión en RF(radio frecuencia) en Wokwi o Proteus, utilizando ESP32 ó ARDUINO con las siguientes especificaciones: 3](#_rz9tnpuocrtz)

[A) Un ESP32 ó ARDUINO en modo Transmisor (TX) 3](#_i7ts1bd7ezaj)

[B) Un ESP32 ó ARDUINO en modo Receptor (RX) 3](#_brhdixgyw7d5)

[2. Salida a una pantalla LCD de 16x2 ó Monitor Serial ó Terminal Virtual, para que solo aparezca una línea de transmisión, por ejemplo: Hello World. 3](#_q1b9qdpad7xo)

[3. Realizar una presentación en \*.pptx, Canvas o software de su elección, con los pasos que siguieron para llegar al resultado final. La presentación no debe tener más de 10 diapositivas. 4](#_1bpu1k48u9xj)

# 

# **Ejercicio Nº1**

## **1. Implementen una simulación de una Conexión en RF(radio frecuencia) en Wokwi o Proteus, utilizando ESP32 ó ARDUINO con las siguientes especificaciones:**

### Comunicación entre ESP32 usando LoRa SX1278

#### 1. Introducción

En este proyecto se busca implementar una comunicación punto a punto entre dos ESP32 NodeMCU-32S utilizando módulos LoRa RA-01 (SX1278). El sistema consistirá en un emisor que enviará datos y un receptor que los mostrará en una pantalla LCD 2x20, con la confirmación de recepción mediante un ACK. A continuación se detalla el diseño completo, asignación de pines y la estructura del código para facilitar su implementación.

#### 2. Materiales Necesarios

* 2x ESP32 NodeMCU-32S
* 2x Módulo LoRa RA-01 (SX1278)
* 1x Pantalla LCD 2x20 con adaptador I2C
* 1x Botón pulsador
* 1x LED
* Cables jumper macho-macho y macho-hembra
* Protoboard

#### 3. Diagrama de conexión

##### 3.1. Conexión del Módulo LoRa RA-01 al ESP32 NodeMCU-32S

| **Módulo LoRa RA-01 (SX1278)** | **ESP32 NodeMCU-32S** |
| --- | --- |
| VCC | 3V3 |
| GND | GND |
| MISO | GPIO 19 |
| MOSI | GPIO 23 |
| SCK | GPIO 18 |
| NSS (CS) | GPIO 5 |
| RESET | GPIO 14 |
| DIO0 | GPIO 2 |

### 

##### 3.2. Conexión del botón y el LED (Emisor)

* Botón: Conectar un extremo al GPIO 12 y el otro a GND (con resistencia pull-up interna activada).
* LED: Conectar el ánodo al GPIO 13 y el cátodo a GND.

##### 3.3. Conexión del LCD 2x20 (Receptor)

* SDA del LCD al GPIO 21 (I2C SDA)
* SCL del LCD al GPIO 22 (I2C SCL)

#### 4. Instalación del Entorno

##### PlatformIO en Vscode

* Generar un nuevo proyecto de PlatformIO para la placa nodeMCU-32S
* Elegir el directorio adecuado
* Instalar las siguientes librerías desde el gestor de librerías:
  + LoRa de Sandeep Mistry.
  + LiquidCrystal\_I2C de marcoschwartz para el LCD con I2C.

#### 5. Flujo del Código

##### Emisor

* Cuando el botón se presiona, se genera un mensaje aleatorio y se envía a través del módulo LoRa.
* Si el emisor recibe un mensaje ACK del receptor, enciende un LED durante un segundo.

##### Receptor

* Escucha constantemente los mensajes entrantes mediante el módulo LoRa.
* Al recibir un mensaje, lo muestra en la pantalla LCD.
* Envía un mensaje ACK de vuelta al emisor.

### ¿Es posible mejorar el algoritmo para que los controladores no estén leyendo el botón o escuchando los mensajes constantemente?

Es posible mediante el uso de las interrupciones. Una interrupción, como su nombre lo indica, corta el flujo normal del programa para atender una actividad determinada y luego devuelve el control del flujo a donde había quedado.

#### ISR en Microcontroladores: Reglas y Buenas Prácticas

* ISR deben ser cortas y rápidas:  
  + Las interrupciones bloquean otras operaciones críticas del sistema. Ejecutar procesos largos (como comunicación SPI con LoRa) puede llevar a problemas de latencia o pérdida de interrupciones.
* No se recomienda llamar a funciones complejas en la ISR:  
  + Comunicaciones por SPI (como con el módulo LoRa) pueden ser problemáticas desde una ISR, ya que SPI usa interrupciones y podría generar un conflicto.
  + Manipular objetos de clase o funciones complejas podría llevar a comportamientos inesperados.
* Alternativa recomendada:  
  + Usar una bandera en la ISR y delegar el procesamiento al loop() es más seguro.

Así, podemos mejorar la eficiencia del código y optimizar el manejo de eventos sin bloquear la ISR.

#### Implementación con Debounce Directo en la ISR

Podemos hacer debounce en la ISR del botón y disparar directamente el método de envío o recepción en loop(), siguiendo las buenas prácticas.

##### Código Mejorado para el Emisor con Debounce en ISR

Se implementa un debounce en la ISR para evitar múltiples interrupciones y llamamos el envío desde el loop() con mínima latencia.

#include <Arduino.h>

#include "com\_LoRa.h"

#include "lcdDisplay.h"

com\_LoRa lora(433E6);

lcdDisplay display;

volatile bool mensajeDisponible = false;

void IRAM\_ATTR isrLoRa() {

mensajeDisponible = true; // Bandera para recibir mensaje

}

void setup() {

Serial.begin(9600);

display.init();

lora.init();

// Configura interrupción en DIO0

attachInterrupt(digitalPinToInterrupt(2), isrLoRa, RISING);

Serial.println("Receptor listo.");

}

void loop() {

if (mensajeDisponible) {

mensajeDisponible = false; // Reinicia bandera

// Procesa el mensaje recibido

String mensaje = lora.receiveMessage();

if (mensaje != "") {

Serial.println("Mensaje recibido: " + mensaje);

display.showMessage(mensaje); // Mostrar en LCD

lora.sendMessage("ACK");

Serial.println("ACK enviado.");

}

}

}

#### Llamadas a funciones dentro de la ISR vs Uso de Banderas en loop()

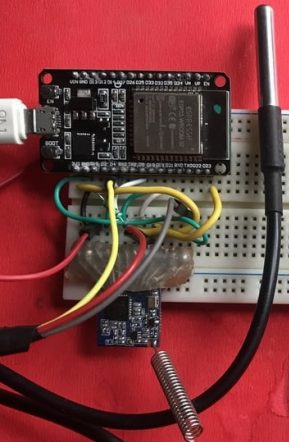
**1- Interrupciones son para eventos rápidos:** Las ISR deben ser cortas, manejando solo lógica simple como debounce o actualización de banderas.

**2- Uso seguro de SPI y LoRa:** Como SPI depende de interrupciones internas, el procesamiento complejo debe hacerse en el loop() para evitar conflictos.

**3- Mínima latencia:** Esta implementación ofrece una respuesta casi inmediata sin bloquear el sistema ni arriesgar conflictos.

### **A) Un ESP32 ó ARDUINO en modo Transmisor (TX)**

### Estructura del Emisor LoRa



#### 1. Archivo de cabecera "com\_Lora.h"

Este archivo define la clase com\_LoRa, que encapsula la funcionalidad del módulo LoRa.

##### Métodos públicos:

* com\_LoRa(long freq): Constructor que inicializa un objeto com\_LoRa con una frecuencia específica.
* void init(): Inicializa el módulo LoRa configurando los pines y la frecuencia.
* void sendMessage(String msg): Envía un mensaje a través de LoRa.
* String receiveMessage(): Recibe un mensaje a través de LoRa.

#### 2. Archivo fuente "com\_Lora.cpp"

Este archivo contiene la implementación de los métodos de la clase com\_LoRa.

##### Implementación de los métodos:

* com\_LoRa::com\_LoRa(long freq): Asigna la frecuencia al atributo frequency.
* com\_LoRa::init(): Configura los pines del módulo LoRa (NSS, Reset, DIO0) e inicia el módulo con la frecuencia especificada. Si la inicialización falla, muestra un mensaje de error y detiene el programa.
* com\_LoRa::sendMessage(String msg): Inicia un nuevo paquete LoRa, escribe el mensaje en el paquete y lo envía.
* com\_LoRa::receiveMessage(): Verifica si hay un paquete recibido. Si lo hay, lee el mensaje byte por byte y lo almacena en un String. Finalmente, devuelve el mensaje recibido.

#### 3. Archivo principal "main.cpp"

Este archivo contiene la lógica principal del programa.

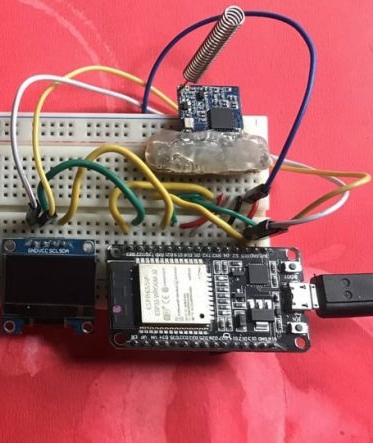
##### Estructura del programa:

El programa principal está estructurado en torno a un bucle principal (loop()) que se ejecuta continuamente.

* Dentro del loop(), se verifica si se ha pulsado un botón.
* Si se ha pulsado el botón, se selecciona un mensaje aleatorio de una lista predefinida.
* El mensaje se envía a través de LoRa utilizando la clase com\_LoRa.
* Luego, se espera la recepción de un ACK (acuse de recibo) con un tiempo de espera de 2 segundos.
* Si se recibe el ACK, se enciende un LED durante 1 segundo.
* Si no se recibe el ACK, se muestra un mensaje en el monitor serie.

### **B) Un ESP32 ó ARDUINO en modo Receptor (RX)**

### Estructura del Receptor LoRa



#### 1. Archivo de cabecera "com\_Lora.h"

#ifndef COM\_LORA\_H

#define COM\_LORA\_H

#include <SPI.h>

#include <LoRa.h>

class com\_LoRa {

private:

long frequency; // Frecuencia del módulo LoRa

public:

com\_LoRa(long freq);

void init(); // Inicializar LoRa

void sendMessage(String msg); // Enviar mensaje

String receiveMessage(); // Recibir mensaje

};

#endif

##### Métodos públicos:

* **com\_LoRa(long freq):** Constructor de la clase que recibe la frecuencia del módulo LoRa.
* **init():** Inicializa el módulo LoRa configurando los pines y la frecuencia.
* **sendMessage(String msg):** Envía un mensaje a través del módulo LoRa.
* **receiveMessage():** Recibe un mensaje a través del módulo LoRa.

#### 2. Archivo fuente "com\_Lora.cpp"

#include "com\_LoRa.h"

com\_LoRa::com\_LoRa(long freq) : frequency(freq) {}

void com\_LoRa::init() {

LoRa.setPins(10, 9, 2); // NSS, Reset, DIO0

if (!LoRa.begin(frequency)) {

Serial.println("Error al inicializar LoRa.");

while (1);

}

Serial.println("LoRa inicializado.");

}

void com\_LoRa::sendMessage(String msg) {

LoRa.beginPacket();

LoRa.print(msg);

LoRa.endPacket();

Serial.println("Mensaje enviado: " + msg);

}

String com\_LoRa::receiveMessage() {

String message = "";

int packetSize = LoRa.parsePacket();

if (packetSize) {

while (LoRa.available()) {

message += (char)LoRa.read();

}

Serial.println("Mensaje recibido: " + message);

}

return message;

}

##### Implementación de los métodos:

* **com\_LoRa(long freq):** Inicializa la variable frequency con la frecuencia recibida como parámetro.
* **init():**
  + Configura los pines del módulo LoRa: **NSS en el pin 10, Reset en el pin 9 y DIO0 en el pin 2**.
  + Inicia el módulo LoRa en la frecuencia especificada. Si falla la inicialización, se imprime un mensaje de error y el programa se detiene.
  + Si la inicialización es exitosa, se imprime un mensaje de éxito.
* **sendMessage(String msg):**
  + Inicia un nuevo paquete LoRa.
  + Escribe el mensaje en el paquete.
  + Finaliza y envía el paquete.
  + Imprime en el monitor serial el mensaje enviado.
* **receiveMessage():**
  + Declara una variable message de tipo String para almacenar el mensaje recibido.
  + Obtiene el tamaño del paquete recibido.
  + Si hay un paquete disponible (packetSize es mayor que 0), lee los bytes del paquete y los concatena en la variable message.
  + Imprime en el monitor serial el mensaje recibido.
  + Retorna el mensaje recibido.

#### 3. Archivo principal "main.cpp"

#include <Arduino.h>

#include "com\_LoRa.h"

#include "lcdDisplay.h"

com\_LoRa lora(433E6);

lcdDisplay display;

volatile bool mensajeDisponible = false;

void IRAM\_ATTR isrLoRa() {

mensajeDisponible = true; // Bandera para recibir mensaje

}

void setup() {

Serial.begin(9600);

display.init();

lora.init();

// Configura interrupción en DIO0

attachInterrupt(digitalPinToInterrupt(2), isrLoRa, RISING);

Serial.println("Receptor listo.");

}

void loop() {

if (mensajeDisponible) {

mensajeDisponible = false; // Reinicia bandera

// Procesa el mensaje recibido

String mensaje = lora.receiveMessage();

if (mensaje != "") {

Serial.println("Mensaje recibido: " + mensaje);

display.showMessage(mensaje); // Mostrar en LCD

lora.sendMessage("ACK");

Serial.println("ACK enviado.");

}

}

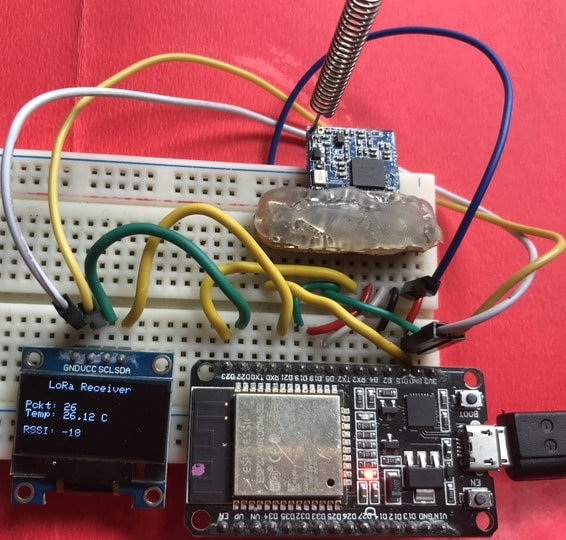
}

##### El programa está estructurado en:

* **Inicialización:** Se configura la comunicación serial, la pantalla LCD y el módulo LoRa. También se configura una interrupción en el pin DIO0 del módulo LoRa para detectar la llegada de un mensaje.
* **Bucle principal:** El programa principal espera la llegada de un mensaje a través de la interrupción. Cuando se recibe un mensaje, se procesa, se muestra en la pantalla LCD y se envía una confirmación ("ACK") al emisor.

## **2. Salida a una pantalla LCD de 16x2 ó Monitor Serial ó Terminal Virtual, para que solo aparezca una línea de transmisión, por ejemplo: Hello World.**

### Estructura para pantalla\_de\_16x2



### 

#### 1. Archivo de cabecera "DisplayOutput.h"

#ifndef DISPLAY\_OUTPUT\_H

#define DISPLAY\_OUTPUT\_H

#include <Arduino.h>

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

class DisplayOutput {

private:

LiquidCrystal\_I2C \*lcd; // Puntero a la pantalla LCD

int lcdColumns;

int lcdRows;

public:

// Constructor

DisplayOutput(int address, int lcdCols, int lcdRows);

// Inicializar la pantalla

void init();

// Mostrar mensaje en el LCD

void printMessage(const String &message);

};

#endif

##### Métodos públicos:

* DisplayOutput(int address, int lcdCols, int lcdRows): Constructor de la clase. Recibe la dirección I2C de la pantalla, el número de columnas y el número de filas.
* init(): Inicializa la pantalla LCD.
* printMessage(const String &message): Muestra un mensaje en la pantalla LCD.

#### 2. Archivo fuente "DisplayOutput.cpp"

#include "DisplayOutput.h"

// Constructor

DisplayOutput::DisplayOutput(int address, int lcdCols, int lcdRows)

: lcdColumns(lcdCols), lcdRows(lcdRows) {

lcd = new LiquidCrystal\_I2C(address, lcdCols, lcdRows); // Crear el objeto LCD

}

// Inicializar el LCD

void DisplayOutput::init() {

lcd->begin(lcdColumns, lcdRows);

lcd->backlight();

}

// Mostrar mensaje en el LCD

void DisplayOutput::printMessage(const String &message) {

lcd->clear();

lcd->setCursor(0, 0); // Iniciar desde la primera línea

lcd->print(message);

}

##### Implementación de los métodos:

* El constructor crea un objeto LiquidCrystal\_I2C con la dirección, columnas y filas especificadas.
* init() inicializa la comunicación con la pantalla LCD y enciende la retroiluminación.
* printMessage() limpia la pantalla, posiciona el cursor en la primera línea y muestra el mensaje.

#### 3. Archivo principal "main.cpp"

#include <Arduino.h>

#include "DisplayOutput.h"

// Dirección I2C de la pantalla LCD

#define LCD\_ADDRESS 0x27

#define LCD\_COLUMNS 16

#define LCD\_ROWS 2

// Crear un objeto de la clase DisplayOutput

DisplayOutput display(LCD\_ADDRESS, LCD\_COLUMNS, LCD\_ROWS);

void setup() {

// Inicializar la comunicación serial

Serial.begin(115200);

// Inicializar el display

display.init();

// Mostrar mensaje en la LCD

display.printMessage("Grupo Nro 1");

}

void loop() {

// Enviar el mismo mensaje por el Monitor Serial

Serial.println("Grupo Nro 1");

delay(2000); // Esperar 2 segundos

}

##### El programa está estructurado en:

* Se definen las constantes para la dirección I2C, columnas y filas de la pantalla LCD.
* Se crea un objeto DisplayOutput llamado display.
* En setup(), se inicializa la comunicación serial y la pantalla LCD, y se muestra un mensaje inicial.
* En loop(), se envía el mismo mensaje por el monitor serial y se espera 2 segundos.

#### 4. Archivo de cabecera "lcdDisplay.h"

#ifndef LCDDISPLAY\_H

#define LCDDISPLAY\_H

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

class lcdDisplay {

private:

LiquidCrystal\_I2C lcd;

public:

lcdDisplay();

void init();

void showMessage(String msg);

};

#endif

##### Métodos públicos:

* lcdDisplay(): Constructor de la clase.
* init(): Inicializa la pantalla LCD.
* showMessage(String msg): Muestra un mensaje en la pantalla LCD.

#### 2. Archivo fuente "lcdDisplay.cpp"

#include "lcdDisplay.h"

lcdDisplay::lcdDisplay() : lcd(0x27, 20, 2) {}

void lcdDisplay::init() {

lcd.begin(20, 2, 0);

lcd.backlight();

lcd.clear();

lcd.print("Esperando...");

}

void lcdDisplay::showMessage(String msg) {

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Recibido:");

lcd.setCursor(0, 1);

lcd.print(msg);

}

##### 

##### Implementación de los métodos:

* El constructor crea un objeto LiquidCrystal\_I2C con la dirección, columnas y filas predefinidas.
* init() inicializa la comunicación con la pantalla LCD, enciende la retroiluminación, limpia la pantalla y muestra un mensaje de espera.
* showMessage() limpia la pantalla, muestra "Recibido:" en la primera línea y el mensaje recibido en la segunda línea.

## **3. Realizar una presentación en \*.pptx, Canvas o software de su elección, con los pasos que siguieron para llegar al resultado final. La presentación no debe tener más de 10 diapositivas.**