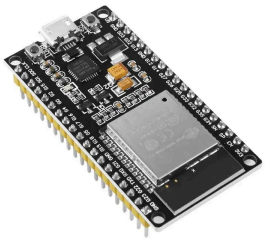
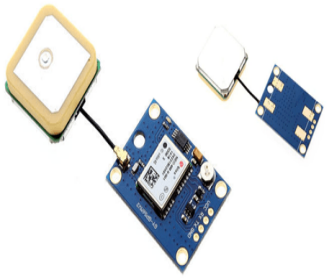
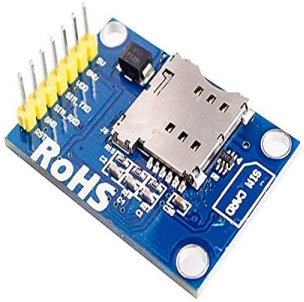
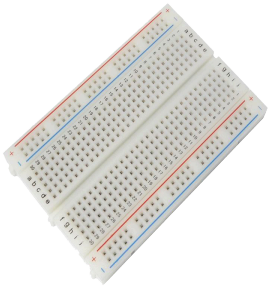
PRACTICA 8 : Buses de comunicación IV (uart)

MATERIAL

Para esta practica necesitaremos el microcontrolador ESP32, Tarjeta microSD y módulo lector, altavoz, cables y una protoboard.




OBJETIVO Y FUNCIONALIDAD DE LA PRACTICA

El objetivo de la practica es comprender el funcionamiento de la comunicacion serie asincrona usart. Esta comunicacion es muy utilizada de hecho en todas las practicas las estamos usando cuando nos referimos a la impresora serie Serial.println("").

Este tipo de protocolo dispone de muchos perifericos y por tanto la comprension de su funcionamiento es basica para cualquier ingeniero electronico.

En la practica de hoy veremos dos ejemplos de uso . La conexion a un modulo GPS de donde visualizaremos la posicion , velocidad y hora actual y una conexion a internet mediante un modem GSM/ GPRS

Software de arduino

El control de los perifericos viene soportado en arduino por una API (objeto) Serial, derivada de la clase stream con la que se puede controlar con facilidad cualquier protocolo de comunicación .

https://www.arduino.cc/reference/en/language/functions/communication/serial/

Las funciones principales són:

- if(Serial): Verifica si el puerto serial está listo

```
void setup() {
  Serial.begin(9600);
  while (!Serial); // Espera a que el puerto esté listo
  Serial.println("Serial listo");
}
```

- available(): Devuelve el número de bytes disponibles para leer.

```
void loop() {
  if (Serial.available() > 0) {
    char c = Serial.read();
    Serial.print("Recibido: ");
    Serial.println(c);
  }
}
```

- availableForWrite(): Devuelve cuántos bytes se pueden escribir sin bloquear.

```
void loop() {
  int espacio = Serial.availableForWrite();
  Serial.println(espacio);
  delay(1000);
}
```

- begin(): Inicia la comunicación serial.

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

- end(): Detiene la comunicación serial.

```
void loop() {
  Serial.end();
  delay(1000);
}
```

- find(): Busca una secuencia específica de caracteres.

```
void loop() {
  if (Serial.find("OK")) {
    Serial.println("Comando OK recibido");
  }
}
```

- findUntil(): Busca hasta encontrar un texto o hasta que otro lo cancele.

```
void loop() {
  if (Serial.findUntil("OK", "CANCEL")) {
    Serial.println("Encontrado OK (no cancelado)");
  }
}
```

- flush(): limpia el buffer de salida

```
void loop() {
  Serial.print("Enviando datos...");
  Serial.flush(); // Espera a que termine de enviarse todo
}
```

- parseFloat(): Lee el siguiente número en punto flotante del buffer.

```
void loop() {
  if (Serial.available()) {
    float valor = Serial.parseFloat();
    Serial.println(valor);
  }
}
```

- parseInt(): Lee el siguiente número entero del buffer.

```
void loop() {
  if (Serial.available()) {
    int num = Serial.parseInt();
    Serial.println(num);
  }
}
```

- peek(): Mira el siguiente byte sin eliminarlo del buffer.

```
void loop() {
  if (Serial.available()) {
    char c = Serial.peek();
    Serial.print("Próximo carácter: ");
    Serial.println(c);
  }
}
```

- print(): Envía datos al puerto serial (sin salto de línea).

```
void loop() {
  Serial.print("Hola ");
  Serial.print("Mundo");
}
```

- println(): Envía datos al puerto serial (con salto de línea).

```
void loop() {
  Serial.println("Hola con salto de línea");
}
```

- read(): Lee 1 byte del buffer de entrada.

```
void loop() {
  if (Serial.available()) {
    char c = Serial.read();
    Serial.print("Leído: ");
    Serial.println(c);
  }
}
```

- readBytes(): Lee varios bytes en un buffer.

```
char datos[10];

void loop() {
  if (Serial.available() >= 10) {
    Serial.readBytes(datos, 10);
    Serial.println("Leídos 10 bytes");
  }
}
```

- readBytesUntil(): Lee bytes hasta cierto carácter o máximo longitud.

```
char datos[20];

void loop() {
  if (Serial.available()) {
    Serial.readBytesUntil('\n', datos, 20);
    Serial.println(datos);
  }
}
```

- readString(): Lee todo lo disponible como cadena.

```
void loop() {
  if (Serial.available()) {
    String texto = Serial.readString();
    Serial.println("Texto recibido:");
    Serial.println(texto);
  }
}
```

- readStringUntil(): Lee hasta encontrar el delimitador.

```
void loop() {
  if (Serial.available()) {
    String dato = Serial.readStringUntil(',');
    Serial.println("Parte recibida:");
    Serial.println(dato);
  }
}
```

- setTimeout(): Define cuánto tiempo espera parseInt, readBytes, etc.

```
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(5000); // Espera hasta 5 segundos
}
```

- write(): Envía datos como bytes.

```
void loop() {
  Serial.write('A'); // Envía byte ASCII de 'A'
  delay(1000);
}
```

- serialEvent(): Función especial que se ejecuta cuando hay datos disponibles (solo si usas SerialEvent() y no bloqueas loop()).

```
void serialEvent() {
  while (Serial.available()) {
    char c = Serial.read();
    Serial.print("Evento: ");
    Serial.println(c);
  }
}
```

Ejercicio practico 1: Bucle de comunicacion uart2

Enunciado : Realizar un bucle de comunicacion de forma que los datos que se manden por el terminal rxd0 se redirijan a la uart 2 txd2 (que debe estar conectado a rxd2) y la recepcion de los datos de la uart2 se reenvien de nuevo a la salida txd0 para que aparezcan en la pantalla del terminal

Código comentado

```
// Se incluye la librería principal de Arduino para poder usar sus funciones
#include <Arduino.h>

// Se define el pin 16 como RX (recepción) de la UART1
#define RXD1 16

// Se define el pin 17 como TX (transmisión) de la UART1
#define TXD1 17

void setup() {
  // Se inicia la comunicación serie USB a 115200 baudios
  Serial.begin(115200);

  // Espera a que el puerto serie esté listo (útil en placas que lo requieren)
  while (!Serial);

  // Se inicia UART1 con 9600 baudios, 8 bits de datos, sin paridad, 1 bit de parada usand
  Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);

  // Mensaje inicial de arranque que se muestra por el monitor serie
  Serial.println("Bucle UART con ESP32-S3 iniciado");
}

void loop() {
  // Variable para almacenar el texto ingresado por el usuario desde el monitor serie
  static String inputString = "";

  // Si hay datos disponibles en el monitor serie (USB), se leen
  if (Serial.available()) {

    // Se lee un carácter
    char incomingByte = Serial.read();

    // Si se detecta el fin de línea (enter), se considera que la cadena está completa
    if (incomingByte == '\n' || incomingByte == '\r') {
      if (inputString.length() > 0) {

        // Se envía la cadena completa por UART1
        Serial1.write(inputString.c_str());
```

```
        // Se muestra en el monitor serie lo que se envió
        Serial.println("Enviado a UART1: " + inputString);

        // Se limpia la cadena para recibir una nueva
        inputString = "";
      }
    } else {
      // Si no es salto de línea, se agrega el carácter a la cadena
      inputString += incomingByte;
    }
  }

  // Si hay datos recibidos por UART1, se leen y se muestran en el monitor serie
  if (Serial1.available()) {
    // Se lee un carácter recibido por UART1
    char receivedByte = Serial1.read();

    // Se envía directamente al monitor serie (USB)
    Serial.write(receivedByte);
}
```

Funcionamiento

1. Lee lo que escribes en el monitor serie del ordenador.
2. Cuando presionas ENTER, envía ese texto por los pines UART1 (TX: 17, RX: 16).
3. Si el dispositivo conectado a esos pines responde, muestra esa respuesta en el monitor serie del ordenador.

En resumen:
• USB ↔ ESP32 ↔ UART1 (otro dispositivo)
• Sirve como puente de comunicación o consola de prueba entre el PC y un módulo externo (como un GPS, módulo Bluetooth, otro micro, etc.).

Para que poder visualizar el mensaje que escribes por pantalla es necesario ejecutar este codigo en la terminal:

```
pio device monitor --echo --baud 115200
```

También puedes ejecutar el código como lo hacemos normalmente, pero no verás tu mensaje hasta que cliques el enter.

Salida Monitor Serial

Este mensaje aparece al encender el ESP32, indicando que el programa ha comenzado y está listo.

```
Bucle UART con ESP32-S3 iniciado
```

Cuando escribes un texto en el monitor serial y pulsas Enter:
• El ESP32 lo captura y lo envía por UART1 al dispositivo conectado.
• Además, te muestra esto:

```
Enviado a UART1: tu_mensaje
```

Si el dispositivo conectado por UART1 (TXD1 y RXD1) responde, cada carácter recibido aparecerá tal cual en el monitor serial, sin prefijo.

Video y foto demostración

https://drive.google.com/file/d/1cSw_DoYaftVTcsIxOxUjfWsFZbA9muss/view?usp=share_link
https://drive.google.com/file/d/1A0dS4xWcSx6TZK-NsvqWNa3F2iCknK7V/view?usp=share_link

Ejercicio practico 2: Modulo GPS (optativo)

He tenido que realizar las siguientes modificaciones al código para que funcionará:

- Eliminar las siguientes librerias:

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>
```

- Incluir la libreria:

```
#include <TinyGPS++.h>
```

- Importante recordar que el puerto RX y el TX deben conectarse cruzados, sino no funcionará.

Código comentado

```
// Incluye la librería TinyGPS++ para procesar datos de GPS (NMEA)
#include <TinyGPS++.h>

// Crea un objeto llamado 'gps' para manejar y decodificar los datos recibidos del módulo G
TinyGPSPlus gps;

// Crea un objeto 'mySerial' usando el puerto UART1 del ESP32
HardwareSerial mySerial(1);

// Define el pin 16 como entrada RX (donde el ESP32 recibirá datos del GPS)
#define RX_PIN 16

// Define el pin 17 como salida TX (no suele usarse para GPS, pero se define por claridad)
#define TX_PIN 17

void setup()
{
  // Inicia la comunicación con el PC a 115200 baudios para mostrar datos en el monitor se
  Serial.begin(115200);

  // Espera hasta que la conexión con el monitor serie esté lista
  while (!Serial);

  // Inicia UART1 a 9600 baudios con configuración 8N1, usando los pines definidos
  mySerial.begin(9600, SERIAL_8N1, RX_PIN, TX_PIN);

  // Imprime un mensaje inicial informando que se está esperando datos del GPS
  Serial.println("Esperando coordenadas del GPS...");
}

void loop()
{
  // Mientras haya datos disponibles desde el GPS en UART1
  while (mySerial.available() > 0)
  {
    // Pasa cada byte recibido al objeto gps para que lo procese (decodifica sentencias NME
    gps.encode(mySerial.read());

    // Si la ubicación fue actualizada con nuevos datos
```

```
    if (gps.location.isUpdated())
    {
      // Obtiene la latitud actual en grados decimales
      float latitude = gps.location.lat();

      // Obtiene la longitud actual en grados decimales
      float longitude = gps.location.lng();

      Serial.print("Latitud: ");
      // Muestra la latitud con 6 cifras decimales
      Serial.print(latitude, 6);
      Serial.print("  Longitud: ");
      // Muestra la longitud con 6 cifras decimales
      Serial.print(longitude, 6);
      Serial.println();

      Serial.print("Satélites: ");
      // Muestra el número de satélites detectados por el GPS
      Serial.println(gps.satellites.value());
    }
  }
}
```

Funcionamiento

1. Se comunica con un módulo GPS a través del puerto UART1 del ESP32 (usando los pines 16 y 17).
2. Lee continuamente los datos GPS enviados en formato NMEA.
3. Decodifica esos datos usando la librería TinyGPS++.
4. Cuando hay una nueva ubicación disponible, muestra:
   • Latitud y longitud en el monitor serie.
   • Cantidad de satélites conectados, lo que indica la calidad de la señal GPS.

Salida Monitor Serial

La salida debe ser algo parecido a esta (con tus propias coordenadas):

```
Esperando coordenadas del GPS...
Latitud: 41.564594  Longitud: 2.023319
Satélites: 11
Latitud: 41.564594  Longitud: 2.023319
Satélites: 11
...
```

Como podemos comprobar, en nuestro caso, marca las coordenadas muy correctas de terrassa:
Coordenadas terrassa: 41.5667, 2.0167

Video demostración

https://drive.google.com/file/d/1yJWaiWN7-h2pAZzLmAoFQTiHtOZTwXUB/view?usp=share_link