PRACTICA 8 : Buses de comunicación IV (uart)

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.

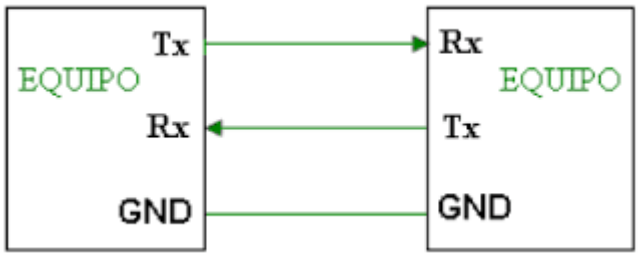
En la practica de hoy veremos dos ejemplos de uso . La conexion a un modulo GPS de donde visualizaremos la posicion , velocidad y hora actual y una conexion a internet mediante un modem GSM/ GPRS

Pero antes daremos una breve introduccion teorica

Introducción teórica

descripcion

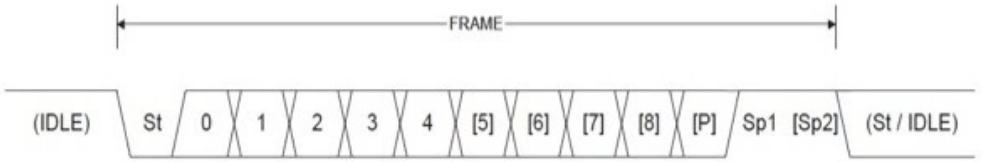
La nomenclatura **uart** (Universal Asyncronous Receiver Transmitter) se refiere a modulos dentro de micro procesador que tienen la capacidad de realizar comunicacion asincrona full duplex **emision y recepcion simultanea** . Habremos observado que hay modulos que indican tambien **usart** en este caso el modulo tambien tiene la posibilidad de realizar comunicacion sincrona (Universal Syncronous Asyncronous Receiver Transmitter). Que son las comunicaciones que hemos visto en las practicas anteriores donde se utiliza una señal de clock para sincronizar.



timing

En la comunicacion asincrona no hay señal de clock y solo se utilizan en la expresion minima de la comunicacion las señales TxD, RxD y GND .

Como se realiza este proceso



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

La comunicación comienza con una señal de Start, seguida de los bits a enviar, y se pueden seleccionar entre 5 y 9 bits a mandar, después tenemos que seleccionar si va a haber un bit de paridad para comprobar errores y por último si tenemos uno o dos bits de Stop. Estos parámetros han de estar configurados de igual manera en los dos dispositivos que se van a comunicar.

Es decir ; que si la velocidad de envio (Baudios) o la paridad o los bit de stop. No estan adecuadamente seleccionados la comunicacion no puede producirse .

Todos ustedes han tenido experiencia de esto cuando no han seleccionado un Baud distinto entre el monitor y el Serial.print .

La razon de esta falta de sincronizacion es que la falta de una señal de clock se soluciona en la comunicacion asincrona mediante la generacion de un clock interno

propio ; que debe tener la misma frecuencia que el emisor . Esta señal de clock interna se sincroniza con el pulso inicial de start , a partir de ese punto se calculan internamente los pediodos de clock y a cada uno se mide la señal de entrada ; y por tanto si ambar son la misma frecuencis se reproducira el dato original ; en otro caso tendremos ruido .

niveles de comunicacion

En funcion de como se identifique el **nivel logico** tendremos diferentes metodos de comunicacion . Los principales son los indicados a continuación.

- RS232 – http://en.wikipedia.org/wiki/RS-232
- RS485 – https://en.wikipedia.org/wiki/RS-485
- RS422 – https://en.wikipedia.org/wiki/RS-422

En el RS232 se utilizan noveles de tension ; mientras que en el RS485 o en RS422 se utilizan corrientes .

Sesugiere al lector que entre en las referencias para conprender la interface logica de los mismos .

Deben describirlas en el informe de la practica. Dando un ejemplo de uso

control de flujo

Adicionalmente a los niveles logicos existen lines an RS232 que se utilizan para informar al receptor que debe escuchar (protocolo hardware) . O protocolos de comunicacion que utilizan bytes de control para indicar que un paquete se ha recibido correctaente (protocolo software)

Observamos que hay 2 señales que adicionales no hemos comentado hasta ahora

- CTS (clear to send) (despejado para enviar)
- RTS (request to send)(solicitud de envío)

En RS-232 común hay pares de líneas de control que generalmente se denominan control de flujo de hardware:

RTS (solicitud de envío) y CTS (despejado para enviar), utilizados en el control de flujo

RTS
DTR (terminal de datos listo) y DSR (conjunto de datos listo), control de flujo DTR
El control de flujo de hardware generalmente lo maneja el DTE o el “extremo maestro”, ya que primero está levantando o afirmando su línea para ordenar al otro lado:

En el caso del flujo de control RTS, el DTE establece su RTS, que indica al extremo opuesto (el extremo esclavo, como un DCE) que comience a monitorear su línea de entrada de datos. Cuando esté listo para los datos, el extremo esclavo elevará su línea complementaria, CTS en este ejemplo, que indica al maestro que comience a enviar datos y que el maestro comience a monitorear la línea de salida de datos del esclavo. Si cualquiera de los extremos necesita detener los datos, baja su línea respectiva de “disponibilidad de datos”.
Para enlaces de PC a módem y enlaces similares, en el caso del control de flujo DTR, DTR / DSR se activan para toda la sesión del módem (por ejemplo, una llamada de acceso telefónico a Internet donde se activa DTR para indicar al módem que marque, y DSR se eleva por el módem cuando se completa la conexión), y se generan RTS / CTS para cada bloque de datos.

3.2 Legacy Hardware Flow Control

A point of confusion when talking about hardware flow control is that the same names are used for different protocols. Hardware Flow Control sometimes refer to another method for flow control. In this document we shall refer to this second method as legacy Hardware Flow Control to differentiate it from the type discussed in 3.1 Hardware Flow Control. The name legacy is used because this method was actually the early method of implementing flow control.

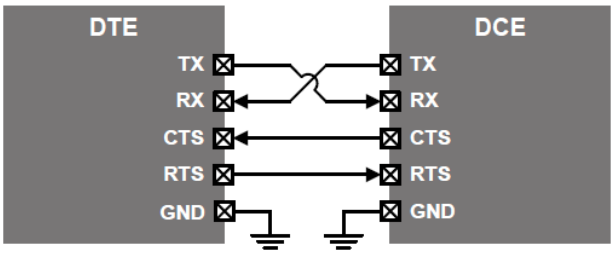


Figure 3.2. Legacy Hardware Flow Control

Legacy Hardware Flow Control still uses two extra wires named RTS and CTS, but the functionality is slightly different. In this scheme the flow control is unidirectional and there is a master/slave relationship (traditionally the master and slave are referred to as DTE (Data Terminal Equipment) and DCE (Data Communications Equipment)). When the master wants to transmit data to the slave it asserts the RTS line. The slave responds by asserting CTS. Transmission can then occur until the slave deasserts CTS, indicating that it needs a temporary halt in transmission. When the master has finished transmitting the entire message it will deassert RTS.

https://en.wikipedia.org/wiki/Flow_control_(data)

se recomienda leer la referencia para entender no solo el control de flujo hardware sino tambien el control de flujo software que utiliza bytes de comunicacion **ACK** **NACK** para determinar si un paquete ha llegado correctamente o no y en cuyo caso debe repetirse

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 .

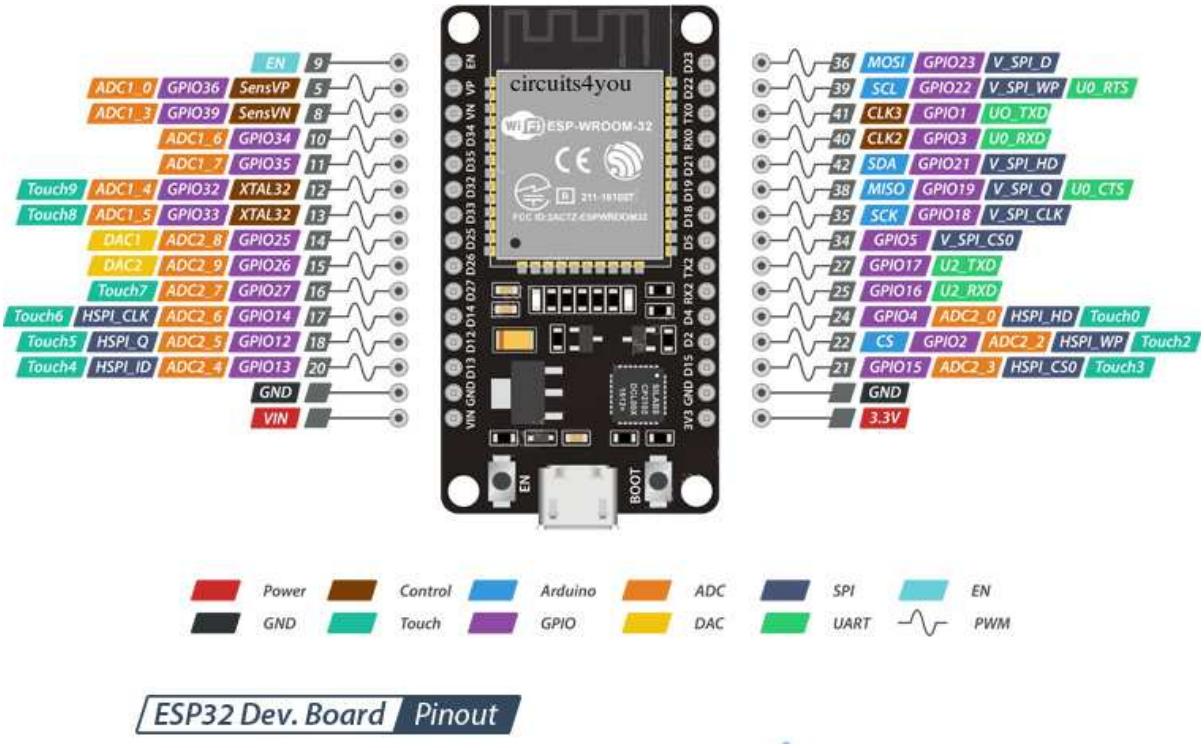
https://www.arduino.cc/reference/en/language/functions/communication/serial/

Las funciones principales

- if(Serial)
- available()
- availableForWrite()
- begin()
- end()
- find()
- findUntil()
- flush()
- parseFloat()
- parseInt()
- peek()
- print()
- println()
- read()
- readBytes()
- readBytesUntil()
- readString()
- readStringUntil()
- setTimeout()
- write()
- serialEvent()

Deben describirlas en el informe de la practica. Dando un ejemplo de uso

uart en ESP32



ESP32 Dev. Board Pinout

Hay tres puertos serie del ESP32 conocidos como U0UXD, U1UXD y U2UXD todo el trabajo en 3.3V Nivel TTL . Hay tres interfaces seriales compatibles con hardware en el ESP32 conocidas como UART0, UART1 y UART2. Como todos los periféricos, los pines de los UART se pueden asignar lógicamente a cualquiera de los pines disponibles en el ESP32. Sin embargo, los UART también pueden tener acceso directo, lo que mejora marginalmente el rendimiento. La tabla de asignación de pines para esta asistencia de hardware es la siguiente.

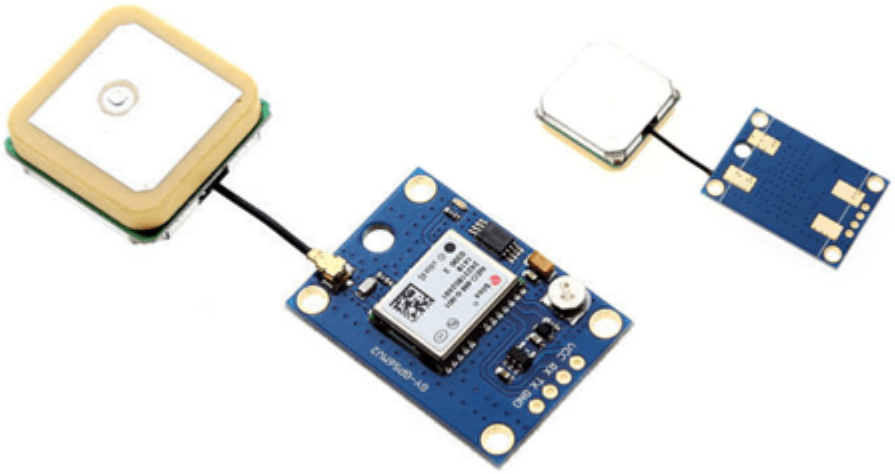
UART	RX IO	TX IO	CTS	RTS
UART0	GPIO3	GPIO1	N / A	N / A
UART1	GPIO9	GPIO10	GPIO6	GPIO11
UART2	GPIO16	GPIO17	GPIO8	GPIO7

ejemplos de aplicacion

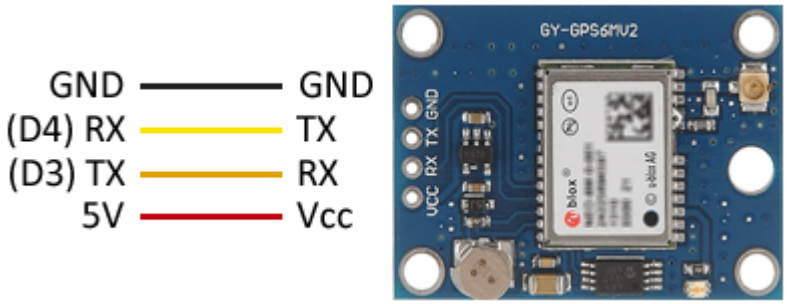
Introduciremos ahora algunos elementos que pueden darnos idea de la importancia de este metodo de comunicacion

modulo GPS

Los receptores de GPS Neo-6 están diseñados para ser dispositivos de bajo precio. Podemos encontrar módulos como el NEO6MV2 por unos 3 a 5 eur , en vendedores internacionales de AliExpress o eBay. amazon

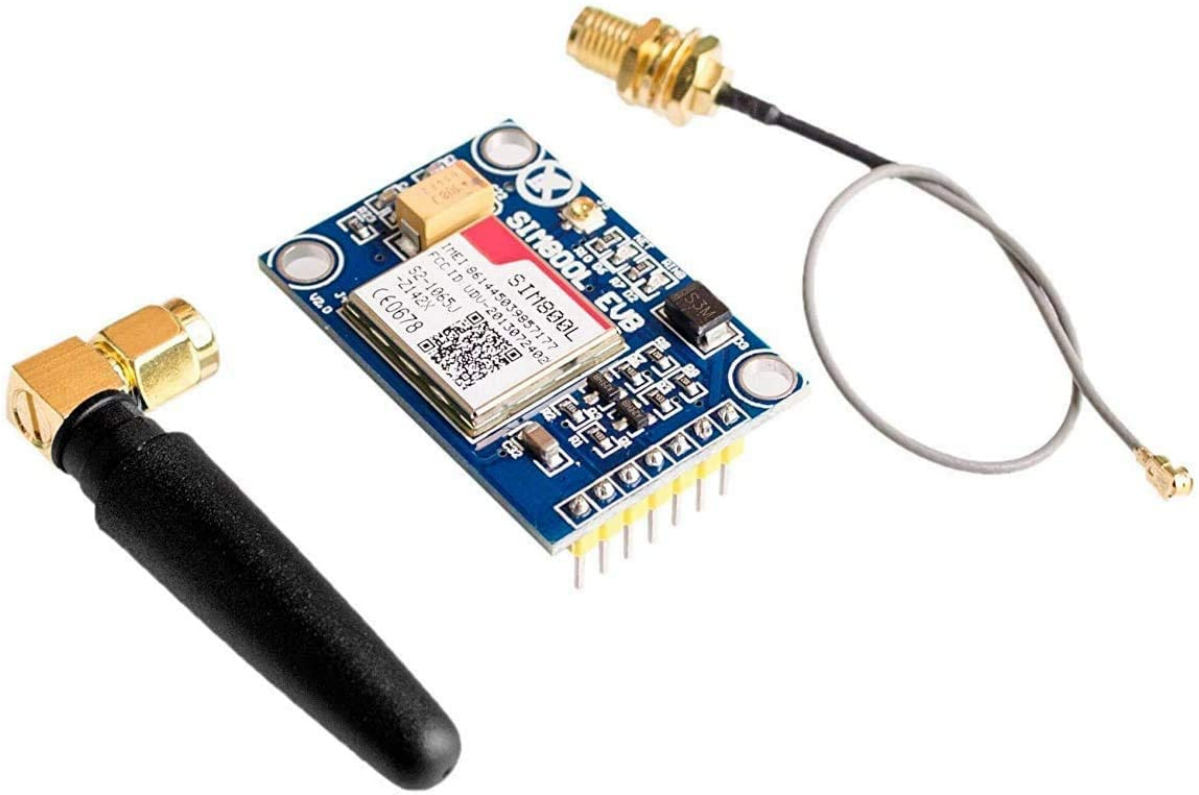


El montaje sera como el de la figura si bien RX y TX los adaptaremso a los pines uart 1 o uart 2 del esp 32

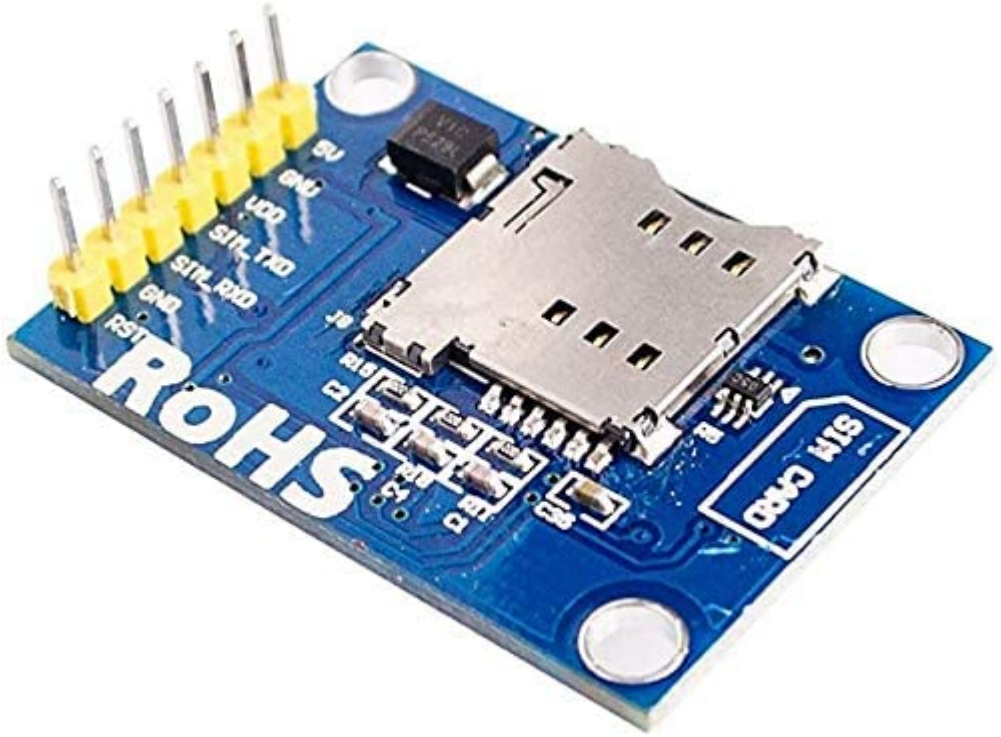


modulo GPRS // GSM

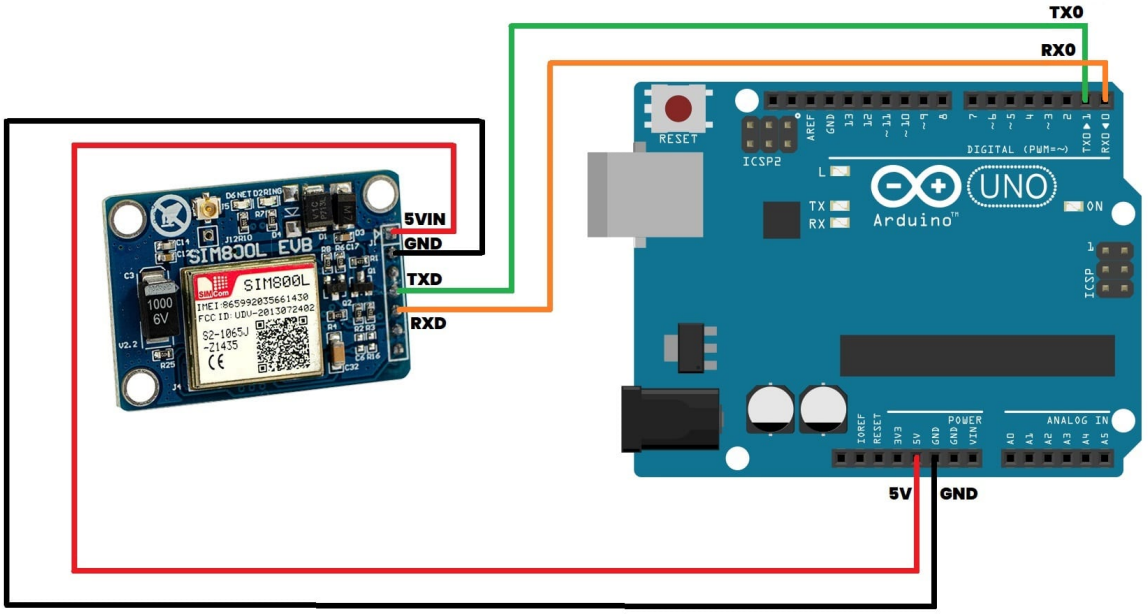
Modulo de comunicaciones telefono movil se puede conseguir por 4-5 euros sin embargo la conexion hardware de los modelos mas esonomicos es un poco mas complicada por lo que recomiendo esta amazon



Estos elements requieren una tarjeta sim ; por lo que para la realizacion de la practica les recomiendo que usen la de su movil .



La conexion la realizaremos a traves del puerto serie 2 del esp32 utilizando una conexion similar a la de la figura



Ejercicio practico 1 bucle de comunicacion uart2

Esta primera practica es la unica obligatoria y no requiere de ningun hardware adicional ; solo la utilizacion del ESP32 .

Sin embargo debido asu simplicidad no se indicara ningun codigo y se pide a cada estudiante que genere su codigo propio.

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

En el informe de la practica debe incluir

- Codigo fuente operativo
- descricion del mismo
- salidas y entradas de consola

Ejercicio practico 2 (optativo) modulo GPS

Realizar un proyecto con el siguiente codigo

Indicar las modificaciones necesarias para que el codigo compile

y probar su funcionamiento

EL MODULO GPS SE DEBE PROBAR EN EXTERIOR ; en caso contrario no cogera satelites

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

TinyGPS gps;
SoftwareSerial softSerial(4, 3);

void setup()
{
   Serial.begin(115200);
   softSerial.begin(9600);
}

void loop()
{
   bool newData = false;
   unsigned long chars;
   unsigned short sentences, failed;

   // Intentar recibir secuencia durante un segundo
   for (unsigned long start = millis(); millis() - start < 1000;)
   {
      while (softSerial.available())
      {
         char c = softSerial.read();
         if (gps.encode(c)) // Nueva secuencia recibida
            newData = true;
      }
   }

   if (newData)
   {
      float flat, flon;
      unsigned long age;
      gps.f_get_position(&flat, &flon, &age);
      Serial.print("LAT=");
      Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
      Serial.print(" LON=");
      Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
      Serial.print(" SAT=");
```

```
      Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ?
      Serial.print(" PREC=");
      Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop
   }

   gps.stats(&chars, &sentences, &failed);
   Serial.print(" CHARS=");
   Serial.print(chars);
   Serial.print(" SENTENCES=");
   Serial.print(sentences);
   Serial.print(" CSUM ERR=");
   Serial.println(failed);
}
```

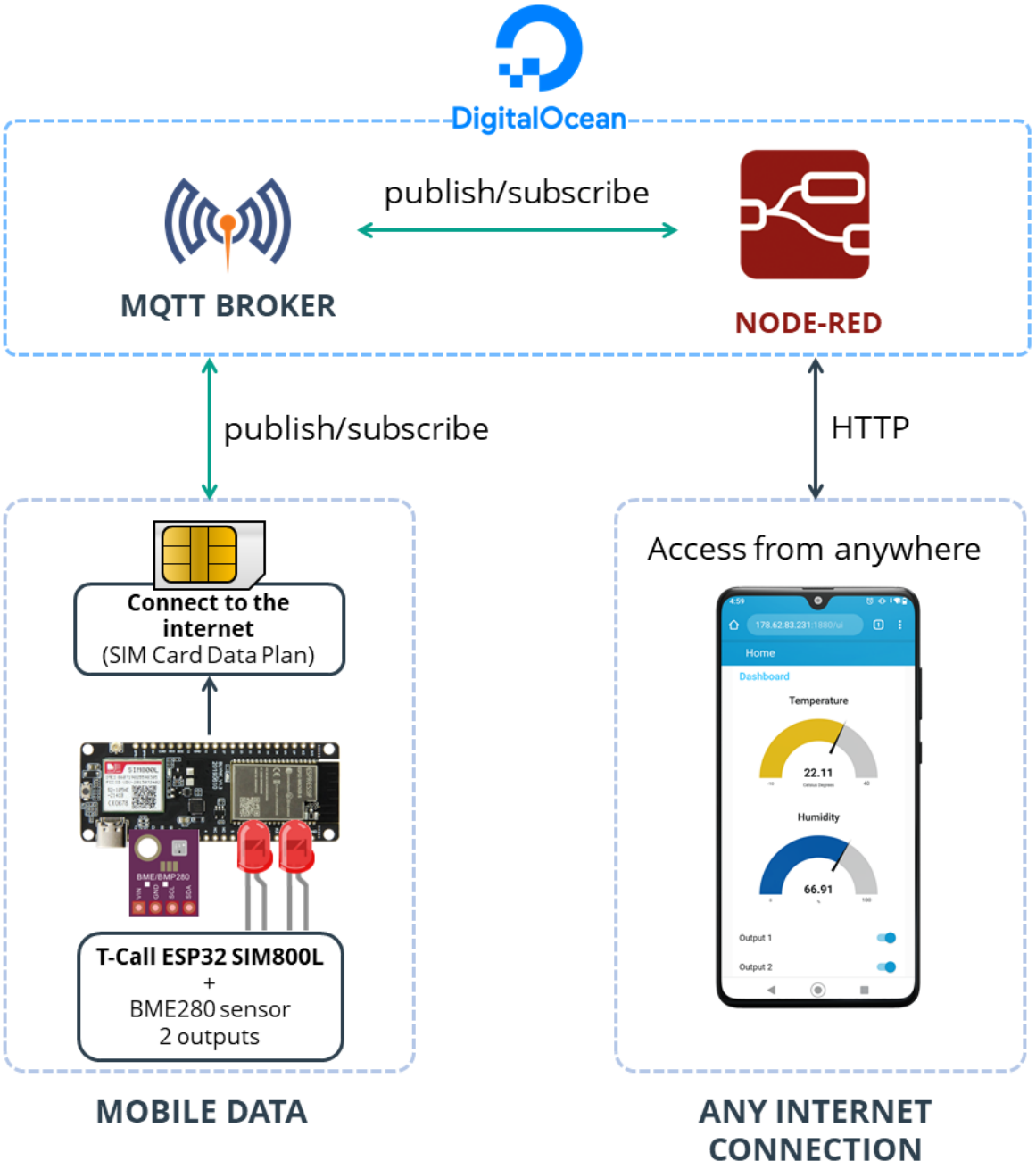
- describir el funcionamiento
- indicar la salida de consola

Ejercicio practico 3 (optativo) modulo GPRS // GSM

Utilizacion de un modulo GPRS/GSM para tener acceso a internet

referencia :https://forum.arduino.cc/t/sim800l-sim800l-evb-v2-0-variations-and-wiring/505185/2

ejemplo de uso



referencia : https://randomnerdtutorials.com/esp32-cloud-mqtt-broker-sim800l/

- buscar librerias arduino sim800L (recomendadas)TinyGSM

```
/*
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com/esp32-cloud

  Permission is hereby granted, free of charge, to any person obtaining a
  of this software and associated documentation files.

  The above copyright notice and this permission notice shall be included
  copies or substantial portions of the Software.
```

```
*/

// Select your modem:
#define TINY_GSM_MODEM_SIM800 // Modem is SIM800L

// Set serial for debug console (to the Serial Monitor, default speed 11
#define SerialMon Serial
// Set serial for AT commands
#define SerialAT Serial1

// Define the serial console for debug prints, if needed
#define TINY_GSM_DEBUG SerialMon

// set GSM PIN, if any
#define GSM_PIN ""

// Your GPRS credentials, if any
const char apn[] = ""; // APN (example: internet.vodafone.pt) use https:/
const char gprsUser[] = "";
const char gprsPass[] = "";

// SIM card PIN (leave empty, if not defined)
const char simPIN[]   = "";

// MQTT details
const char* broker = "XXX.XXX.XXX.XXX";                    // Public IP a
const char* mqttUsername = "REPLACE_WITH_YOUR_MQTT_USER";  // MQTT userna
const char* mqttPassword = "REPLACE_WITH_YOUR_MQTT_PASS";  // MQTT passwo

const char* topicOutput1 = "esp/output1";
const char* topicOutput2 = "esp/output2";
const char* topicTemperature = "esp/temperature";
const char* topicHumidity = "esp/humidity";

// Define the serial console for debug prints, if needed
//#define DUMP_AT_COMMANDS

#include <Wire.h>
#include <TinyGsmClient.h>

#ifdef DUMP_AT_COMMANDS
  #include <StreamDebugger.h>
  StreamDebugger debugger(SerialAT, SerialMon);
  TinyGsm modem(debugger);
#else
```

```
  TinyGsm modem(SerialAT);
#endif

#include <PubSubClient.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

TinyGsmClient client(modem);
PubSubClient mqtt(client);

// TTGO T-Call pins
#define MODEM_RST            5
#define MODEM_PWKEY          4
#define MODEM_POWER_ON       23
#define MODEM_TX             27
#define MODEM_RX             26
#define I2C_SDA              21
#define I2C_SCL              22

// BME280 pins
#define I2C_SDA_2            18
#define I2C_SCL_2            19

#define OUTPUT_1             2
#define OUTPUT_2             15

uint32_t lastReconnectAttempt = 0;

// I2C for SIM800 (to keep it running when powered from battery)
TwoWire I2CPower = TwoWire(0);

TwoWire I2CBME = TwoWire(1);
Adafruit_BME280 bme;

#define IP5306_ADDR          0x75
#define IP5306_REG_SYS_CTL0  0x00

float temperature = 0;
float humidity = 0;
long lastMsg = 0;

bool setPowerBoostKeepOn(int en){
  I2CPower.beginTransmission(IP5306_ADDR);
  I2CPower.write(IP5306_REG_SYS_CTL0);
  if (en) {
```

```
    I2CPower.write(0x37); // Set bit1: 1 enable 0 disable boost keep on
  } else {
    I2CPower.write(0x35); // 0x37 is default reg value
  }
  return I2CPower.endTransmission() == 0;
}

void mqttCallback(char* topic, byte* message, unsigned int len) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < len; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

  // Feel free to add more if statements to control more GPIOs with MQTT

  // If a message is received on the topic esp/output1, you check if the
  // Changes the output state according to the message
  if (String(topic) == "esp/output1") {
    Serial.print("Changing output to ");
    if(messageTemp == "true"){
      Serial.println("true");
      digitalWrite(OUTPUT_1, HIGH);
    }
    else if(messageTemp == "false"){
      Serial.println("false");
      digitalWrite(OUTPUT_1, LOW);
    }
  }
  else if (String(topic) == "esp/output2") {
    Serial.print("Changing output to ");
    if(messageTemp == "true"){
      Serial.println("true");
      digitalWrite(OUTPUT_2, HIGH);
    }
    else if(messageTemp == "false"){
      Serial.println("false");
      digitalWrite(OUTPUT_2, LOW);
    }
  }
```

```
}

boolean mqttConnect() {
  SerialMon.print("Connecting to ");
  SerialMon.print(broker);

  // Connect to MQTT Broker without username and password
  //boolean status = mqtt.connect("GsmClientN");

  // Or, if you want to authenticate MQTT:
  boolean status = mqtt.connect("GsmClientN", mqttUsername, mqttPassword)

  if (status == false) {
    SerialMon.println(" fail");
    ESP.restart();
    return false;
  }
  SerialMon.println(" success");
  mqtt.subscribe(topicOutput1);
  mqtt.subscribe(topicOutput2);

  return mqtt.connected();
}



void setup() {
  // Set console baud rate
  SerialMon.begin(115200);
  delay(10);

  // Start I2C communication
  I2CPower.begin(I2C_SDA, I2C_SCL, 400000);
  I2CBME.begin(I2C_SDA_2, I2C_SCL_2, 400000);

  // Keep power when running from battery
  bool isOk = setPowerBoostKeepOn(1);
  SerialMon.println(String("IP5306 KeepOn ") + (isOk ? "OK" : "FAIL"));

  // Set modem reset, enable, power pins
  pinMode(MODEM_PWKEY, OUTPUT);
  pinMode(MODEM_RST, OUTPUT);
  pinMode(MODEM_POWER_ON, OUTPUT);
  digitalWrite(MODEM_PWKEY, LOW);
  digitalWrite(MODEM_RST, HIGH);
  digitalWrite(MODEM_POWER_ON, HIGH);
```

```
pinMode(OUTPUT_1, OUTPUT);
pinMode(OUTPUT_2, OUTPUT);

SerialMon.println("Wait...");

// Set GSM module baud rate and UART pins
SerialAT.begin(115200, SERIAL_8N1, MODEM_RX, MODEM_TX);
delay(6000);

// Restart takes quite some time
// To skip it, call init() instead of restart()
SerialMon.println("Initializing modem...");
modem.restart();
// modem.init();

String modemInfo = modem.getModemInfo();
SerialMon.print("Modem Info: ");
SerialMon.println(modemInfo);

// Unlock your SIM card with a PIN if needed
if ( GSM_PIN && modem.getSimStatus() != 3 ) {
  modem.simUnlock(GSM_PIN);
}

// You might need to change the BME280 I2C address, in our case it's 0x
if (!bme.begin(0x76, &I2CBME)) {
  Serial.println("Could not find a valid BME280 sensor, check wiring!");
  while (1);
}

SerialMon.print("Connecting to APN: ");
SerialMon.print(apn);
if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
  SerialMon.println(" fail");
  ESP.restart();
}
else {
  SerialMon.println(" OK");
}

if (modem.isGprsConnected()) {
  SerialMon.println("GPRS connected");
}
```

```
  // MQTT Broker setup
  mqtt.setServer(broker, 1883);
  mqtt.setCallback(mqttCallback);
}

void loop() {
  if (!mqtt.connected()) {
    SerialMon.println("=== MQTT NOT CONNECTED ===");
    // Reconnect every 10 seconds
    uint32_t t = millis();
    if (t - lastReconnectAttempt > 10000L) {
      lastReconnectAttempt = t;
      if (mqttConnect()) {
        lastReconnectAttempt = 0;
      }
    }
    delay(100);
    return;
  }

  long now = millis();
  if (now - lastMsg > 30000) {
    lastMsg = now;

    // Temperature in Celsius
    temperature = bme.readTemperature();
    // Uncomment the next line to set temperature in Fahrenheit
    // (and comment the previous temperature line)
    //temperature = 1.8 * bme.readTemperature() + 32; // Temperature in F

    // Convert the value to a char array
    char tempString[8];
    dtostrf(temperature, 1, 2, tempString);
    Serial.print("Temperature: ");
    Serial.println(tempString);
    mqtt.publish(topicTemperature, tempString);

    humidity = bme.readHumidity();

    // Convert the value to a char array
    char humString[8];
    dtostrf(humidity, 1, 2, humString);
    Serial.print("Humidity: ");
    Serial.println(humString);
    mqtt.publish(topicHumidity, humString);
```

```
  }

  mqtt.loop();
}
```

Ejercicios de subida de nota

- La realizacion de las practicas 2 o 3 se considerara para subir nota
- La practica 3 con la inclusion de sensores adecuados puede utilizarse como proyecto final