

## Ejercicio #2 Consignas

1. Explicar detalladamente el funcionamiento del terminal virtual en proteus, del monitor serie en VsCode@platformIO y del monitor serie en el ide de Arduino.
2. Que función tienen los terminales RTS y CTS en el terminal virtual?
3. Que es una transmisión serie o UART? Y que significan las siguientes propiedades: Baud Rate, Data Bits, Parity, Stop Bits, Send XON/XOFF, terminal Type.
4. Que es el eco, en relación al tipeo y una pantalla. ¿Y porque no tengo eco en el terminal virtual de proteus?
5. Explique las propiedades avanzadas del terminal virtual.
6. Como funciona COMPIM y para qué sirve? Que es un virtualizador de puertos, de ejemplos de los más utilizados.

---

## ¿Qué es virtual terminal en Proteus?

---

El terminal teletipo virtual es un modelo que simula un terminal de teletipo TTY de comunicaciones serie convencional.

Sus principales características son: Completo soporte bi-direccional. Los datos recibidos se visualizan como caracteres ASCII y las teclas pulsadas se transmiten como datos serie ASCII.

---

## ¿Cómo usar el monitor serial en Proteus?

---

Para simular el Monitor Serial en Proteus debemos hacer uso del instrumento Virtual Terminal. Podemos tipear las ordenes si nos posicionamos dentro de la ventana del Virtual Terminal. Si observamos en Proteus: Otro ejemplo: En este caso enviaremos números de 3 dígitos para las órdenes.

---

## ¿Cómo ver el monitor serial en Proteus?

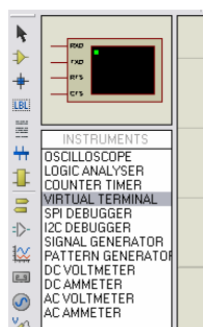
---

Para simular el Monitor Serial en Proteus debemos hacer uso del instrumento Virtual Terminal. Podemos tipear las ordenes si nos posicionamos dentro de la ventana del Virtual Terminal. Si observamos en Proteus: Otro ejemplo: En este caso enviaremos números de 3 dígitos para las órdenes

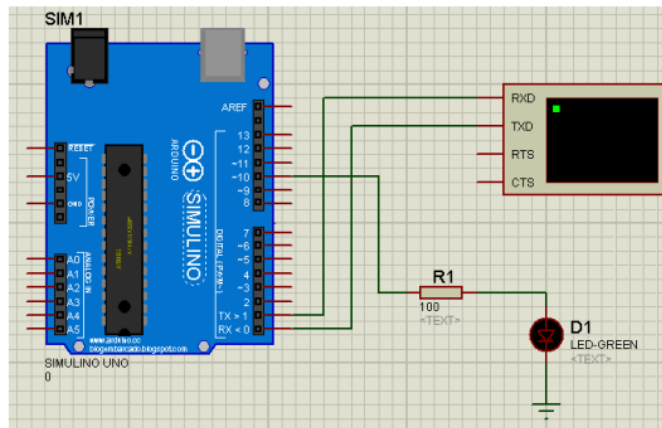
### SIMULAR MONITOR SERIAL EN PROTEUS

(versión 29-11-17)

Para simular el Monitor Serial en Proteus debemos hacer uso del instrumento Virtual Terminal.



Lo incluimos en nuestro circuito Arduino como muestra la figura:



Notar que los TX y RX van cruzados entre el Virtual Terminal y la placa Arduino.

Notar que los TX y RX van cruzados entre el Virtual Terminal y la placa Arduino.

Cargaremos el siguiente programa:

```
//Programa LED_CONTROL

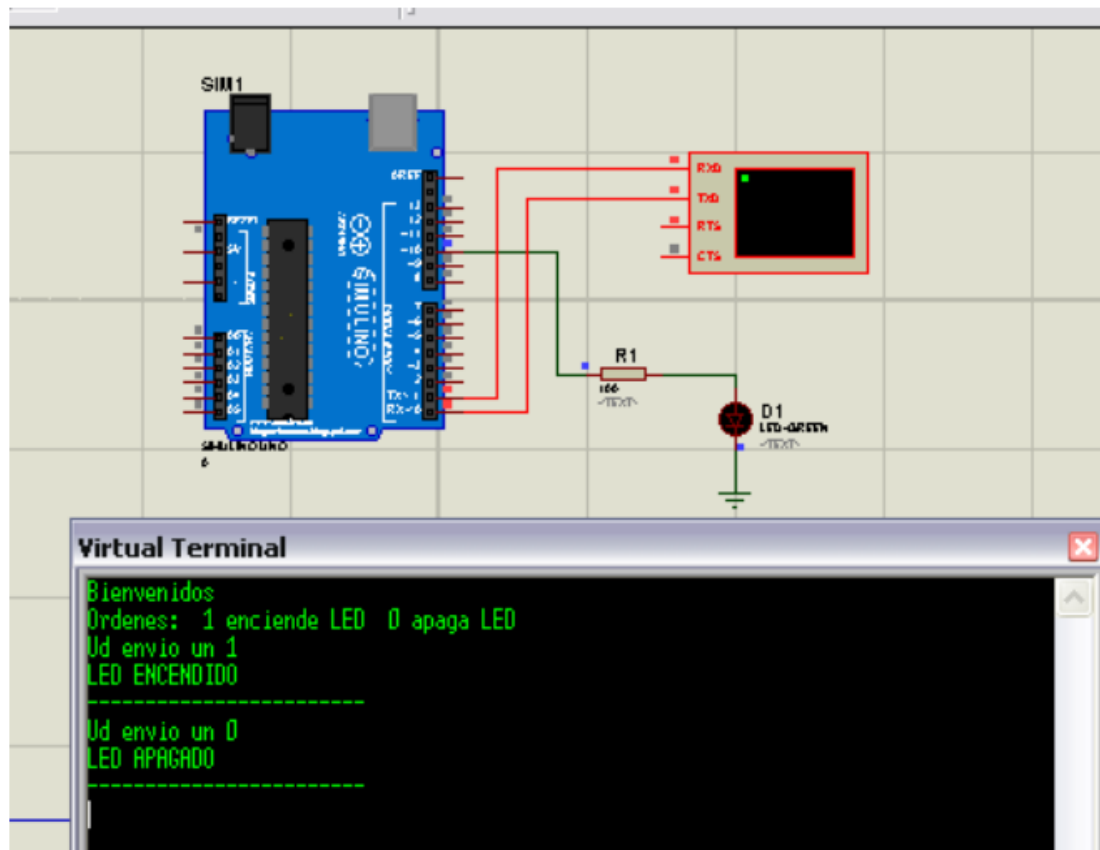
void setup()
{
  Serial.begin(9600); // inicializamos la comunicaci3n serial
  pinMode(10,OUTPUT); //definimos el PIN 10 como salida

  Serial.println("Bienvenidos "); //Mensaje a Monitor Serial
  Serial.println("Ordenes: 1 enciende LED 0 apaga LED"); //Mensaje por Monitor Serial
  delay(100); //Los retardos son necesarios en la practica para mejorar desempe1o
}

void loop()
{
  if(Serial.available()) //Si la comunicacion serial es utilizable, pregunta aqui.
  {
    char c=Serial.read(); // Se lee el monitor serial y esperando solo un caracter
    // se almacena en una variable tipo char que llamamos c
  }
}
```

```
if(c=='1') //pregunta por el contenido de la variable c
{
  digitalWrite(10,HIGH); // coloca en ALTO la salida digital PIN 10
  Serial.println("Ud envio un 1"); //Mensaje a Monitor Serial
  Serial.println("LED ENCENDIDO"); //Mensaje a Monitor Serial
  Serial.println("-----");
  delay(100); //Los retardos son necesarios en la practica para mejorar desempe1o
}
if(c=='0') //pregunta por el contenido de la variable c
{
  digitalWrite(10,LOW);
  Serial.println("Ud envio un 0"); //Mensaje a Monitor Serial
  Serial.println("LED APAGADO"); //Mensaje a Monitor Serial
  Serial.println("-----"); //Mensaje a Monitor Serial
  delay(100); //Los retardos son necesarios en la practica para mejorar desempe1o
}
}
}
```

Podemos tipear las ordenes si nos posicionamos dentro de la ventana del Virtual Terminal.



Otro ejemplo:

#### //PROGRAMA LED\_CONTROL1

```
void setup()
{
  Serial.begin(9600); // inicializamos la comunicaci3n serial
  pinMode(10, OUTPUT); // definimos el PIN 10 como salida
  Serial.println("Bienvenidos "); // Mensaje a Monitor Serial
  Serial.println("Ordenes: 1 enciende LED 0 apaga LED"); // Mensaje por Monitor Serial delay(100);
  // Los retardos son necesarios en la practica para mejorar desempe1o
}
void loop()
{
  if(Serial.available())
  {
    // Si la comunicaci3n serial es utilizable, pregunta aqu.
    {
      char c = Serial.read(); // Se lee el monitor serial y esperando solo un caracter
      // se almacena en una variable tipo char que llamamos c

      Serial.println("-----"); // Mensaje a Monitor Serial
      Serial.print("Ud digito: "); // Mensaje a Monitor Serial

      Serial.println(c); // Mensaje a Monitor Serial
    }
  }
}
```

```

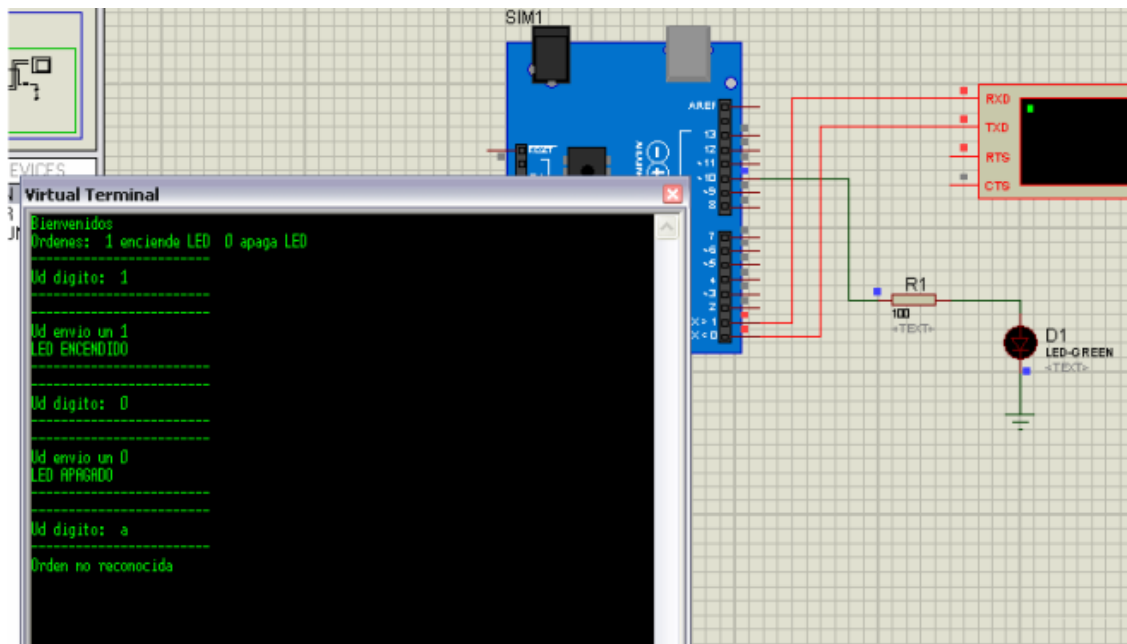
Serial.println("-----"); //Mensaje a Monitor Serial if(c=='1') //pregunta
por el contenido de la variable c { digitalWrite(10,HIGH); // coloca en ALTO la salida digital
PIN 10 Serial.println("-----"); //Mensaje a Monitor Serial
Serial.println("Ud envio un 1"); //Mensaje a Monitor Serial
Serial.println("LED ENCENDIDO"); //Mensaje a Monitor Serial
Serial.println("-----");
delay(100); //Los retardos son necesarios en la práctica para mejorar desempeño
}
if(c=='0') //pregunta por el contenido de la variable c
{
digitalWrite (10, LOW); Serial.println("-----"); //Mensaje a Monitor Serial

    Serial.println("Ud envio un 0"); //Mensaje a Monitor Serial
    Serial.println("LED APAGADO"); //Mensaje a Monitor Serial
    Serial.println("-----"); //Mensaje a Monitor Serial
    delay (100); //Los retardos son necesarios en la práctica para mejorar desempeño
}

if ((c! ='0') && (c! ='1')) Serial.println("Orden no reconocida"); //Mensaje a Monitor Serial}

```

Si observamos en Proteus



Otro ejemplo: En este caso enviaremos números de 3 dígitos para las órdenes.

//PROGRAMA LED\_CONTROL2 //

```

Lee algo en el puerto serial y lo almacena en num
int num; // Definida como variable global void setup()
{
Serial.begin(9600); // inicializamos la comunicación serial
pinMode(10,OUTPUT); //definimos el PIN 10 como salida

Serial.println("Bienvenidos "); //Mensaje a Monitor Serial
Serial.println("Ordenes: 345 enciende LED 678 apaga LED");//Mensaje por Monitor Serial
delay(100); //Los retardos son necesarios en la practica para mejorar desempeño
}

void loop()

```

```

{
/*
 * Evaluamos el momento en el cual recibimos un caracter
 * a través del puerto serie
 */

if(Serial.available()) //Si la comunicacion serial es utilizable, pregunta aqui.
{
//Delay para favorecer la lectura de caracteres

    delay(300); //Este tiempo es grande para PODER HACER SIMULACION PROTEUS
    //originalmente decia 22 para Arduino real

    //Se crea una variable que servirá como buffer
    String bufferString = ""; 5
/*
 * Se le indica a Arduino que mientras haya datos
 * disponibles para ser leídos en el puerto serie
 * se mantenga concatenando los caracteres en la
 * variable bufferString
 */ while (Serial.available()>0) {
    bufferString += (char)Serial.read();
}
num = bufferString.toInt(); //Se transforma el buffer a un número entero //Se carga lo leído
en la variable num
//Luego podemos preguntar sobre el valor
// de dicha variable - Por ejemplo
// en Tachos LED su valor selecciona color

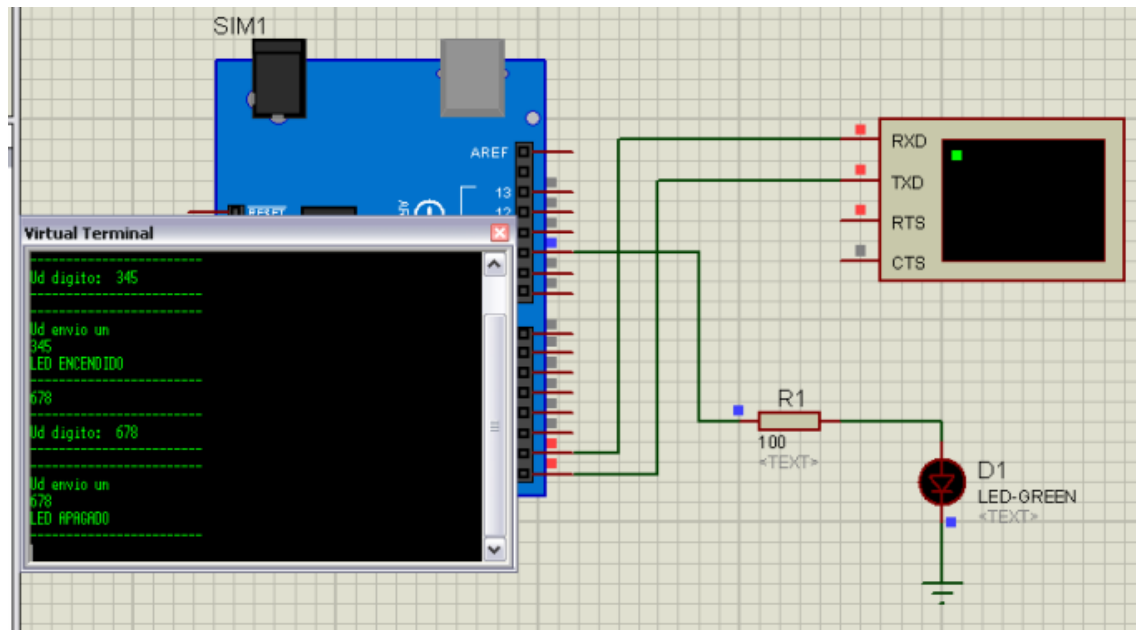
Serial.println("-----"); //Mensaje a Monitor Serial Serial.print("Ud digito:
"); //Mensaje a Monitor Serial
Serial.println(num); //Mensaje a Monitor Serial
Serial.println("-----"); //Mensaje a Monitor Serial

    if(num==345) //pregunta por el contenido de la variable num
    {
        digitalWrite(10,HIGH);
// coloca en ALTO la salida digital PIN 10
Serial.println("-----"); //Mensaje a Monitor Serial
Serial.println("Ud envio un "); //Mensaje a Monitor Serial
Serial.println(num);
Serial.println("LED ENCENDIDO"); //Mensaje a Monitor Serial
Serial.println("-----");
delay(2000); //Los retardos son necesarios en la practica para mejorar desempeño
    }
    if(num==678) //pregunta por el contenido de la variable num
    {
        digitalWrite(10,LOW);
Serial.println("-----"); //Mensaje a Monitor Serial
Serial.println("Ud envio un "); //Mensaje a Monitor Serial
Serial.println(num); Serial.println("LED APAGADO"); //Mensaje a Monitor Serial
Serial.println("-----"); //Mensaje a Monitor Serial
delay(200); //Los retardos son necesarios en la practica para mejorar desempeño }

if((num!=345)&& (num!=678)) Serial.println("Orden no reconocida"); //Mensaje a Monitor Serial }
}

```

## Si Observamos Proteus



**IMPORTANTE:** A la hora de hacer las simulaciones en Proteus, siempre se debe partir de tener un conocimiento de lo que se espera como resultado, si es necesario de debe jugar con los delay del programa para que se compense el tiempo de procesamiento que necesita su equipo informático.

---

# PlatformIO

---

## 1 ¿Qué es PlatformIO?

## 2 Instalación

## 3 Crear un nuevo proyecto

## 4 Compilando y ejecutando un proyecto.

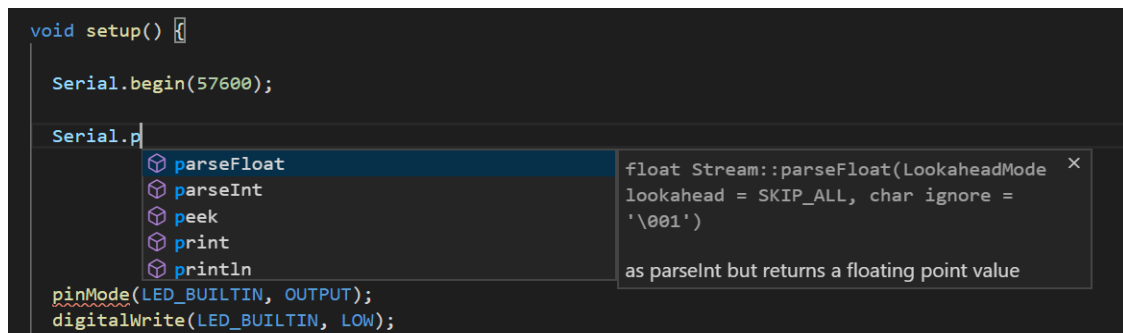
## 5 Configuración platformio.ini

## 6 Conclusión

Cuando acumulas una variedad de tarjetas de desarrollo y/ microcontroladores, al final terminas teniendo el ordenador lleno de mil programas, que si el IDE para ST, que si el de Kinetis, el de NXP, Arduino, ... Son miles de programas, que además de ocupar espacio en el disco duro, ocupan mucho tiempo si tienes que volver a reinstalar todo.

Si estáis acostumbrados a Arduino, igual no habéis tenido este problema, pero aún así os interesa lo que os voy a contar. Así que lo que os vengo a enseñar es el PlatformIO, un sistema o entorno preparado para poder trabajar sobre muchos microcontroladores diferentes, plataformas y frameworks, de la manera más sencilla posible.

Para los usuarios de Arduino, deciros que esta plataforma, tiene bastantes mejoras con respecto al IDE de Arduino original, entre ellas está el autocompletado, poder moverte por el código de manera automática o el mejor manejo para proyectos un poco más complejos.



```
void setup() {  
  
    Serial.begin(57600);  
  
    Serial.p  
        parseFloat  
        parseInt  
        peek  
        print  
        println  
    pinMode(LED_BUILTIN, OUTPUT);  
    digitalWrite(LED_BUILTIN, LOW);  
}
```

float Stream::parseFloat(LookaheadMode lookahead = SKIP\_ALL, char ignore = '\001')  
as parseInt but returns a floating point value

Ejemplo de autocompletado en Arduino

---

## ¿Qué es PlatformIO?

---

PlatformIO es un ecosistema para el desarrollo de sistemas embarcados e IoT, el cual facilita y mucho la gestión de proyectos de software embarcado, gestión de dependencias y librerías, test,



... Es un poco, todo lo que se le puede pedir a un ecosistema así. De momento, no le he encontrado ningún problema en su planteamiento (aunque que sí algún fallo), y es que te permite de manera muy sencilla generar entornos para programar casi cualquier microcontrolador desde el mismo entorno.

Además, te abstraes de toda la configuración que hay detrás de los *frameworks*. En algunos casos esto es malo, ya que pierdes el control de lo que se está haciendo por debajo. Aunque puedes prácticamente configurar todos los parámetros de compilación, flasheo y 'monitoreo', el funcionamiento básico es muy asequible. Por otra parte, te permite depurar el software mientras se ejecuta en la placa, siempre que esta lo permita.

En el momento de escribir esto soporta:

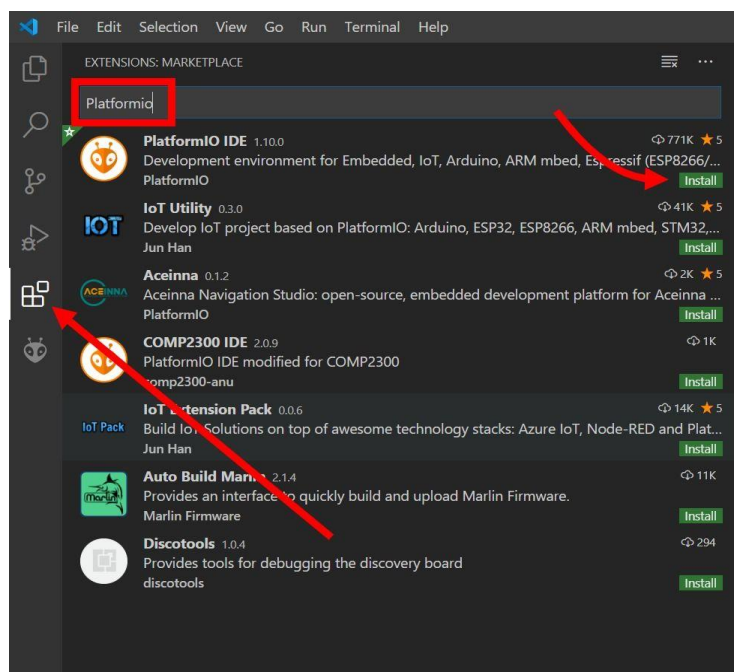
- 20 frameworks: Arduino, mbed, CMSIS, ESP-IDF, STM32Cube,....
- 785 placas de desarrollo: Casi todos los Arduinos, STM32F1, ESP32, ...
- 7163 librerías
- 179 ejemplos de código

---

## Instalación

---

La instalación de PlatformIO es muy sencilla, simplemente tenéis que descargar el **Visual Studio Code**, qué es el IDE de Microsoft, la versión gratuita. Una vez que la abres tienes que ir a la pestaña de complementos, y buscar el "PlatformIO", darle a Instalar y esperar. Os dejo una imagen con el proceso.



## Instalación PlatformIO

### Crear un nuevo proyecto

Una vez que lo tengas instalado, lo único que tienes que hacer es crear un nuevo proyecto, para ello vas a lo que se llama el **PIO Home**, que es como la pantalla de inicio del PlatformIO. Desde esta parte podés hacer todo lo referente a la gestión de proyectos del PlatformIO. Si no lo ves, en la parte izquierda nos ha tenido que salir una nueva pestaña con la cabeza de una abeja. Si abrís esa pestaña, nos saldrá a la izquierda abajo un menú **Quick Access** con un elemento: **PIO Home**, le haces click y luego a **Projects & Configuración** (debajo del PIO Home). Nos saldrá una lista con los proyectos que tienes, y un botón para añadir nuevo proyecto. En la ventana que nos tiene que salir, solo se piden 3 campos:

**Nombre del proyecto** (creo que es bastante autodescriptivo)

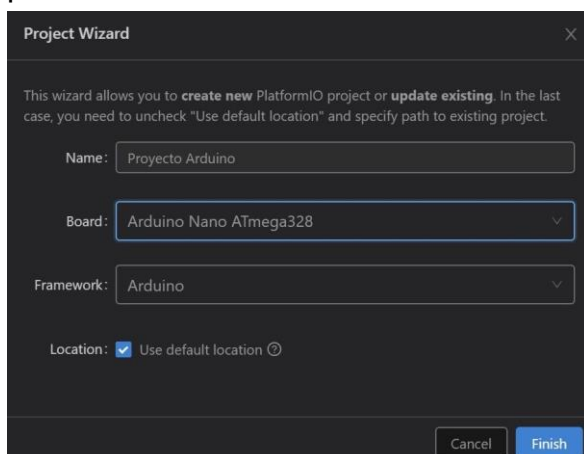
**Board** o placa de desarrollo. Aquí tienes que buscar vuestra placa, según vas escribiendo nos irá mostrando los resultados más parecidos. Algunos ejemplos (hay hasta casi 900 tarjetas):

Arduino Nano ATmega168

DOIT ESP32 DEVKIT V1

BluePill F103C8

**Framework:** Qué es el conjunto de librerías con el que quieres programar. Aquí podés elegir **Arduino** (para programar como en Arduino normal), o mejor, usar librerías más avanzadas si te lo permite como **mbed** o **ESP-IDF...**

The image shows a 'Project Wizard' dialog box with a dark theme. It contains the following fields and options: 'Name' with the text 'Proyecto Arduino'; 'Board' with a dropdown menu showing 'Arduino Nano ATmega328'; 'Framework' with a dropdown menu showing 'Arduino'; and 'Location' with a checked checkbox 'Use default location' and a help icon. At the bottom right are 'Cancel' and 'Finish' buttons.

Project Wizard

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name:

Board:

Framework:

Location: ☒ Use default location ⓘ

### Ventana de nuevo proyecto

Una vez que aceptado todo tardará un rato en generar el proyecto ya que la primera vez que generas un proyecto se descarga todo lo necesario, es decir, la información de la placa que has elegido, y el frameworks.

---

## Compilando y ejecutando un proyecto.

---

Cuando se genera un proyecto, se crean las carpetas y ficheros básicos necesarios, en algunos casos no usaremos todos.

En este caso para el PlatformIO se generarán:

**.pio/**

carpeta donde se generarán los ficheros intermedios de la compilación y temporales.

**include/**

Aquí se deberían guardar las cabeceras de tus ficheros, es decir los ficheros .h

**lib/**

carpeta para generar librerías propias de tu proyecto, es decir librerías privadas

**source/**

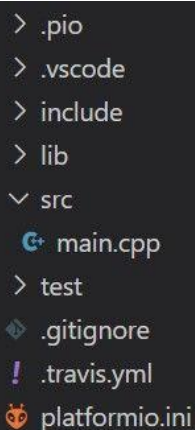
para guardar el código fuente (.c y .cpp)

**test/**

aquí puedes definir tus test para que se ejecuten y comprueben que el código funciona bien

**platformio.ini**

fichero de configuración del PlatformIO, aquí se definen las librerías a usar por el programa (por ejemplo, la librería SD de Arduino...), y muchas otras cosas. Generalmente con el que viene por defecto



```
> .pio
> .vscode
> include
> lib
▼ src
  C++ main.cpp
> test
.gitignore
! .travis.yml
platformio.ini
```

---

## Proyecto base PlatformIO

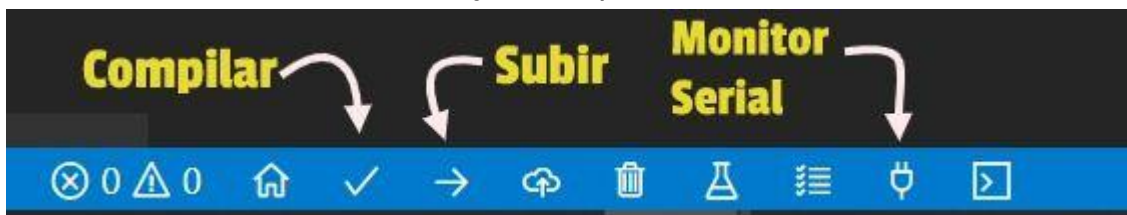
---

De hecho, el fichero *main.cpp* es el fichero principal de tu programa y si lo abres verás que ya está preparado para ser compilado con los *includes* necesarios. Si estas acostumbrado a los ficheros *\*.ino* verás que es igual solo que se le ha añadido una cabecera

**#include <Arduino.h>**

el resto es exactamente igual.

Para compilar y subir el programa nos habrán aparecido unos iconos en la parte inferior izquierda para poder trabajar de manera más cómoda. Son como los del IDE de Arduino, además de funciones más avanzadas. Dejamos aquí las más comunes.



---

## Nuevos botones PlatformIO

---

El único cambio que yo noto con respecto al Arduino es el puerto serie, hay que cerrarlo con **CTRL + T**, sino igual nos da problemas.

Configuración *platformio.ini*

Algunos valores de configuración que suelo cambiar yo son:

**Velocidad del monitor serial:** Para adecuarlo a la velocidad que se configura en nuestro micro.

`monitor_speed = 57600`

**Puerto monitor:** Podés cambiar el puerto que se configura por defecto gracias a `monitor_port = ...`. Aquí podés poner un puerto COM, o un socket, o lo que quieras.

### Conclusión

Dejando un primer vistazo al PlatformIO, al probar, y si se tiene mil placas de diferentes fabricantes y mil IDEs, con esto se solucionan los problemas.

Además, al ser Opensource, se puede ver y modificar el código, por ejemplo, si se necesita la placa FRDM-KL82Z y al ver que está fallando, al arreglarla no se tiene que esperar a una nueva versión.

---

## 2. ¿Qué función tienen los terminales RTS y CTS en el terminal virtual?

---

El protocolo RTS / CTS es un método de intercambio de información que utiliza un cable en cada dirección para permitir que cada dispositivo indique al otro si está o no listo para recibir datos en un momento dado. Un dispositivo envía en RTS y escucha en CTS; el otro hace lo contrario. Un dispositivo debe bajar su cable de salida de protocolo de enlace cuando está listo para recibir datos, y alto cuando no lo está. Un dispositivo que desea enviar datos no debe comenzar a enviar bytes mientras el cable de entrada de intercambio sea bajo; Si ve que el cable del protocolo de enlace pasa a nivel alto, debería terminar de transmitir el byte actual y luego esperar a que el cable del protocolo de enlace se agote antes de seguir transmitiendo.

Tenga en cuenta que, aunque lo ideal es que los dispositivos nunca envíen más de un byte después de que su entrada de intercambio sea alta (si la línea se pone alta al comenzar a transmitir un carácter, deben permitir que ese carácter se transmita por completo), muchos puertos serie de PC No cumpla con esto, incluso cuando está habilitada la comunicación. Los puertos en serie permiten que el software detecte el estado del cable entrante de intercambio y espera que el software decida cuándo se deben poner en cola los datos para la transmisión. Desafortunadamente, la única forma de lograr un buen rendimiento con un puerto en serie es poner en cola los datos para su transmisión un poco antes de que realmente se envíen, y muchos puertos en serie para PC siempre transmitirán los datos en cola tan rápido como puedan sin importarlos. Para los cables de apretón de manos. En consecuencia, no es infrecuente que los puertos serie de PC envíen una docena de caracteres, incluso después de que se les haya pedido que esperen.

RTS = Solicitud de envío. El dispositivo de envío le indica al otro extremo que se prepare para recibir y que configure su línea CTS cuando esté listo.

CTS = Borrado para enviar. El extremo receptor está listo ("todo despejado") y le dice al otro extremo que comience a enviar los caracteres.



### **CTS - Listo para enviar**

La señal CTS se recibe desde el otro extremo del cable serie. Un voltaje de espacio indica que está bien enviar más datos en serie desde su estación de trabajo.

CTS generalmente se usa para regular el flujo de datos en serie desde su estación de trabajo hasta el otro extremo.

---

### RTS - Solicitud de envío

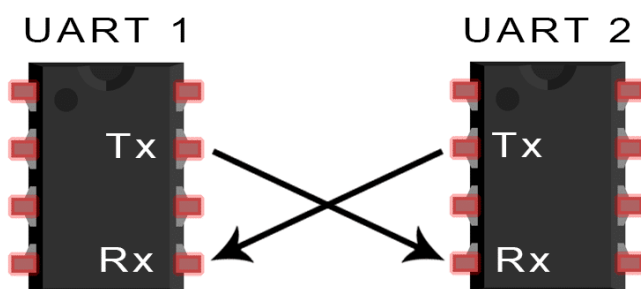
---

Su estación de trabajo establece la señal RTS en el voltaje espacial para indicar que hay más datos listos para enviarse.

Al igual que CTS, RTS ayuda a regular el flujo de datos entre su estación de trabajo y la computadora o dispositivo en el otro extremo del cable serie. La mayoría de las estaciones de trabajo dejan esta señal establecida en el voltaje del espacio todo el tiempo.

**3. Que es una transmisión serie o UART? Y que significan las siguientes propiedades: BaudRate, Data Bits, Parity, Stop Bits, Send XON/XOFF, terminal Type.**

En la comunicación UART, dos UART se comunican directamente entre sí. El UART transmisor convierte los datos paralelos de un dispositivo de control como una CPU en forma serial, los transmite en serie al UART receptor, que luego convierte los datos seriales nuevamente en datos paralelos para el dispositivo receptor. Solo se necesitan dos cables para transmitir datos entre dos UART. Los datos fluyen desde el pin Tx del UART transmisor al pin Rx del UART receptor:



Los UART transmiten datos de *forma asincrónica*, lo que significa que no hay una señal de reloj para sincronizar la salida de bits del UART transmisor con el muestreo de bits del UART receptor. En lugar de una señal de reloj, el UART transmisor agrega bits de inicio y parada al paquete de datos que se transfiere. Estos bits definen el comienzo y el final del paquete de datos para que el UART receptor sepa cuándo comenzar a leer los bits.

Cuando el UART receptor detecta un bit de inicio, comienza a leer los bits entrantes a una frecuencia específica conocida como tasa de *baudios*. La tasa de baudios es una medida de la velocidad de transferencia de datos, expresada en bits por segundo (bps). Ambos UART deben operar aproximadamente a la misma velocidad en baudios. La tasa de baudios entre los UART de transmisión y recepción solo puede diferir en aproximadamente un 10% antes de que la sincronización de los bits se aleje demasiado.

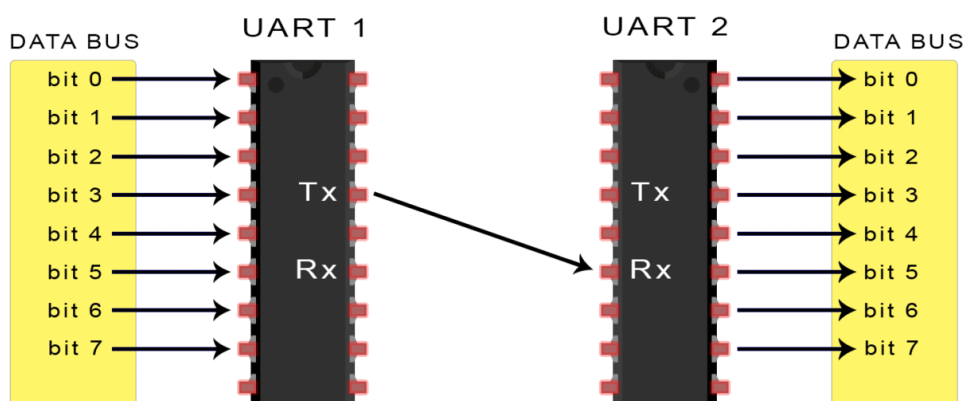
Ambos UART también deben estar configurados para transmitir y recibir la misma estructura de paquetes de datos.

---

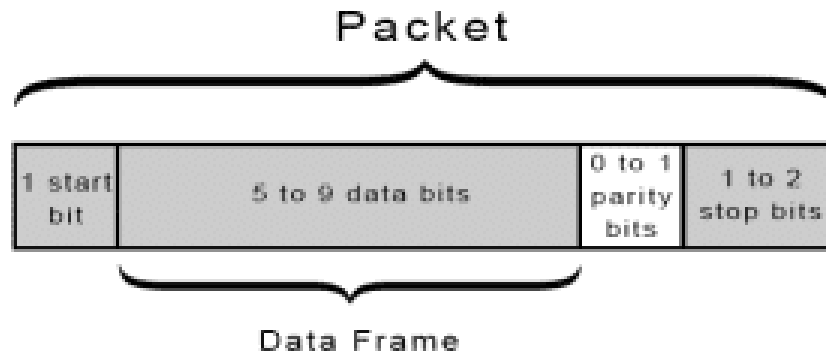
## ¿CÓMO FUNCIONA UART?

---

El UART que va a transmitir datos recibe los datos de un bus de datos. El bus de datos se utiliza para enviar datos al UART por otro dispositivo como una CPU, memoria o microcontrolador. Los datos se transfieren desde el bus de datos al UART transmisor en forma paralela. Después de que el UART transmisor obtenga los datos paralelos del bus de datos, agrega un bit de inicio, un bit de paridad y un bit de parada, creando el paquete de datos. A continuación, el paquete de datos se emite en serie, bit a bit en el pin Tx. El UART receptor lee el paquete de datos bit a bit en su pin Rx. Luego, el UART receptor vuelve a convertir los datos en formato paralelo y elimina el bit de inicio, el bit de paridad y los bits de parada. Finalmente, el UART receptor transfiere el paquete de datos en paralelo al bus de datos en el extremo receptor:



Los datos transmitidos por UART se organizan en *paquetes*. Cada paquete contiene 1 bit de inicio, de 5 a 9 bits de datos (según el UART), un bit de *paridad* opcional y 1 o 2 bits de parada:



## VENTAJAS Y DESVENTAJAS DE LOS UART

Ningún protocolo de comunicación es perfecto, pero los UART son bastante buenos en lo que hacen. Aquí hay algunos pros y contras para ayudarlo a decidir si se ajustan o no a las necesidades de su proyecto:

### VENTAJAS

- Solo usa dos cables
- No se necesita señal de reloj
- Tiene un bit de paridad para permitir la comprobación de errores.
- La estructura del paquete de datos se puede cambiar siempre que ambos lados estén configurados para ello.
- Método bien documentado y ampliamente utilizado.

### DESVENTAJAS

- El tamaño de la trama de datos está limitado a un máximo de 9 bits
- No es compatible con múltiples esclavos o múltiples sistemas maestros
- Las tasas de baudios de cada UART deben estar dentro del 10% de cada uno

---

## Baud Rate

---

La tasa de baudios es la velocidad a la que se transfiere la información en un canal de comunicación. La tasa de baudios se usa comúnmente cuando se habla de electrónica que usa



comunicación en serie. En el contexto del puerto serie, "9600 baudios" significa que el puerto serie es capaz de transferir un máximo de 9600 bits por segundo.

A velocidades de transmisión superiores a 76.800, será necesario reducir la longitud del cable. Cuanto mayor sea la velocidad en baudios, más sensible se vuelve el cable a la calidad de la instalación, debido a la cantidad de cable que se desenrosca alrededor de cada dispositivo.

---

### Data Bits

---

La cantidad de bits de datos en cada carácter puede ser 5 (para el código Baudot ), 6 (rara vez se usa), 7 (para ASCII verdadero ), 8 (para la mayoría de los tipos de datos, ya que este tamaño coincide con el tamaño de un byte ), o 9 (poco usado). 8 bits de datos se utilizan casi universalmente en las aplicaciones más nuevas. 5 o 7 bits generalmente solo tienen sentido con equipos más antiguos, como teleimpresoras.

La mayoría de los diseños de comunicaciones en serie envían primero los bits de datos dentro de cada byte , el bit menos significativo . También es posible, pero rara vez se usa, es el bit más significativo primero; esto fue utilizado, por ejemplo, por el terminal de impresión IBM 2741 . El orden de los bits generalmente no se puede configurar dentro de la interfaz del puerto serie, pero lo define el sistema host. Para comunicarse con sistemas que requieren un orden de bits diferente al predeterminado local, el software local puede reordenar los bits dentro de cada byte justo antes de enviar y justo después de recibir.

La línea de transmisión de datos UART normalmente se mantiene en un nivel de alto voltaje cuando no está transmitiendo datos. Para iniciar la transferencia de datos, el UART transmisor tira de la línea de transmisión de mayor a menor durante un ciclo de reloj. Cuando el UART receptor detecta la transición de alto a bajo voltaje, comienza a leer los bits en el marco de datos a la frecuencia de la velocidad en baudios.

---

### Parity

---

La paridad es un método para detectar errores en la transmisión. Cuando se utiliza la paridad con un puerto serie, se envía un bit de datos adicional con cada carácter de datos, dispuesto de modo que el número de bits 1 en cada carácter, incluido el bit de paridad, sea siempre par o impar. Si se recibe un byte con el número incorrecto de 1, entonces debe estar dañado. La paridad correcta

no indica necesariamente la ausencia de corrupción, ya que una transmisión corrupta con un número par de errores pasará la verificación de paridad. Un solo bit de paridad no permite la implementación de corrección de errores en cada carácter y protocolos de comunicación. Trabajar sobre enlaces de datos en serie generalmente tendrá mecanismos de nivel superior para garantizar la validez de los datos y solicitar la retransmisión de datos que se han recibido incorrectamente.

El bit de paridad en cada carácter se puede establecer en uno de los siguientes:

- Ninguno (N) significa que no se envía ningún bit de paridad y se acorta la transmisión.
- Impar (O) significa que el bit de paridad se establece de modo que el número de bits 1 sea impar.
- Par (E) significa que el bit de paridad está configurado para que el número de bits 1 sea par.
- La paridad de marca (M) significa que el bit de paridad siempre se establece en la condición de señal de marca (valor de 1 bit).
- La paridad de espacio (S) siempre envía el bit de paridad en la condición de señal de espacio (valor de bit 0).

Aparte de las aplicaciones poco comunes que usan el último bit (generalmente el noveno) para alguna forma de direccionamiento o señalización especial, la paridad de marca o espacio es poco común, ya que no agrega información de detección de errores.

La paridad impar es más útil que la paridad par, ya que garantiza que se produzca al menos una transición de estado en cada carácter, lo que la hace más confiable para detectar errores como los que podrían ser causados por discrepancias en la velocidad del puerto serie. Sin embargo, la configuración de paridad más común es none, con la detección de errores manejada por un protocolo de comunicación.

Para permitir la detección de mensajes dañados por el ruido de la línea, se dispusieron teleimpresores electromecánicos para imprimir un carácter especial cuando los datos recibidos contenían un error de paridad.

---

## Stop Bits

---

Los bits de parada enviados al final de cada carácter permiten que el hardware de la señal receptora detecte el final de un carácter y lo vuelva a sincronizar con el flujo de caracteres. Los dispositivos electrónicos suelen utilizar un bit de parada. Si se utilizan

teleimpresores electromecánicos lentos , es posible que se requieran uno y medio o dos bits de parada.

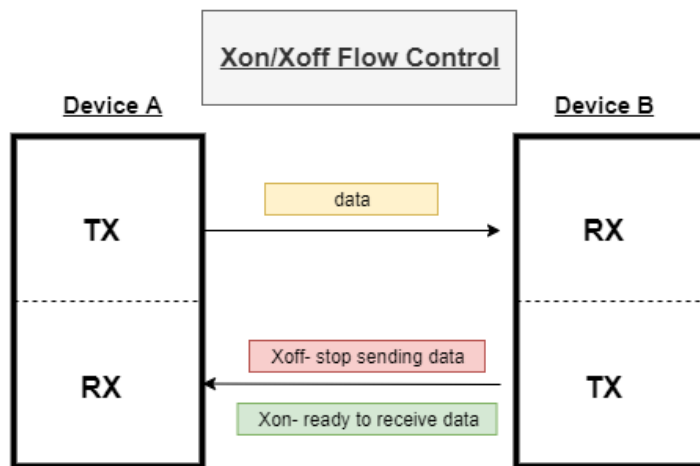
---

### Send XON/XOFF

---

Xon-Xoff es un método de control de flujo utilizado en la comunicación entre dispositivos. Se distingue del control de flujo de hardware, que se realiza mediante señales específicas fuera de banda, como DTR/DSR o RTS/CTS en el protocolo estándar RS232.

Por ejemplo, tiene dos dispositivos, *A* y *B*, y *A* es más rápido para enviar datos que el dispositivo *B* para recibirlos y procesarlos. El dispositivo *B* alcanzará muy rápidamente un punto en el que ya no podrá procesar más datos y se verá abrumado por la cantidad de datos que envía el dispositivo *A*. En este punto, el dispositivo *B* necesitaría enviar un carácter *Xoff* al dispositivo *A* para que deje de enviar datos. No enviará más datos al dispositivo *B* hasta que el dispositivo *B* haya enviado un carácter *Xon* al dispositivo *A*, lo que permite que el dispositivo *A* sepa que el dispositivo *B* está listo para recibir más datos.



Xon-Xoff utiliza códigos especiales acordados tanto por el transmisor como por el receptor.

La siguiente tabla es un ejemplo de algunas de las formas de expresar los códigos Xon-Xoff enviados en el flujo de datos, aunque se pueden establecer otros valores.

Código	Sentido	ASCII	DIC	MALEFICIO	Teclado
XAPAGADO	Pausar	DC3	19	13	Control
XON	Reanudar	DC1	17	11	Control

## VENTAJAS DE XON/XOFF

- La principal ventaja del control de flujo Xon-Xoff es que solo requiere la cantidad mínima de cableado de señal, *tres*. Esto se debe a que el control de flujo Xon-Xoff solo requiere dos señales (enviar, recibir) y, por supuesto, un solo cable de tierra común. El control de flujo de hardware requiere al menos dos cables adicionales entre los dos dispositivos, como en RTS/CTS y DTR/DSR . El costo de esto fue significativo en los primeros días de la informática.

## DESVENTAJAS DE XON/XOFF

- El envío de caracteres Xoff/Xon ocupa ancho de banda porque son señales dentro de la banda. Debido a esto, no pueden confundirse con datos, sino que deben reconocerse como comandos de control de flujo. Esto significa que se necesita codificación para la transmisión de los caracteres *Xon* y *Xoff* para que el receptor no confunda los comandos con comandos de control del dispositivo. Esto generalmente se hace mediante algún tipo de caracteres de escape, pero esto significa que estos comandos ocupan más ancho de banda. Por otro lado, las señales de hardware como RTS/CTS pueden afirmarse muy rápidamente como señales fuera de banda, ocupando menos ancho de banda.

---

### Terminal Type

---

Recuerde que Unix evolucionó en una época en la que muchos fabricantes fabricaban muchos modelos diferentes de terminales de computadora, cada uno con su propio conjunto de códigos de comando para borrar la pantalla, mover el cursor a diferentes posiciones de la pantalla, configurar negrita, subrayado y otras características de texto, y pronto. Una instalación típica de Unix estará equipada para comunicarse con cualquiera de los cientos de tipos diferentes de terminales.

Ahora, no estás usando una terminal, pero estás usando un programa que simula una. telnet originalmente estaba destinado a permitir que los terminales emitan comandos a las CPU remotas, y ssh pretende ser un reemplazo del telnet protocolo anterior, por lo que su programa de cliente ssh en realidad funciona simulando un terminal de computadora "anticuado". Los autores de su programa cliente eligieron uno o más tipos de terminales que simularían. Para que Unix manipule su pantalla adecuadamente, debe saber qué tipo de códigos de comando de terminal está preparado para aceptar su programa de cliente ssh.

Averigüe el tipo de terminal que está emulando su programa telnet. Es posible que deba consultar la documentación del programa o los archivos de ayuda para esto. También puede deducir esta información de los menús "Opciones" o "Preferencias" del programa.

Aquí hay algunas opciones comunes:

Programa cliente SSH	TÉRMINO tipo	Comentarios
Masilla	término x	
OpenSSH	xtérmino?	(Necesito comprobar esto)
SSH de línea de comandos, formato básico, Linux u OS/X	xterm o Linux	
SSH de línea de comandos, formato básico, CygWin	xterm o rxvt	Depende del tipo de ventana en la que escriba el comando.
SSH de línea de comandos, xtermvariante	término x	

Mire los mensajes que recibió después de iniciar sesión. Si incluye una línea que dice algo así como " El tipo de terminal es... " y el tipo de terminal que nombra tiene sentido para su programa de cliente ssh, está listo y puede omitir el próximo paso. Pero si el tipo de terminal parece incorrecto, o ve un mensaje que indica que está en una " terminal tonta " [ 6 ] , debe decirle al sistema Unix qué tipo de terminal está emulando realmente . El comando para hacerlo es

**setenv TERM xxxx** = (donde xxxx es el tipo de terminal (p. ej., **setenv TERM vt100**).)

La mayoría de los terminales " tontos " proporcionan 24 líneas de texto. Es posible que su conexión ssh tenga esto predeterminado, pero la mayoría de los sistemas Unix permitirán que sus terminales proporcionen un área de texto más grande. Si elige usar un área de texto más grande, debe decirle a Unix cuántas líneas está usando dando el comando

**Sttyrowsnn** = **Donde** nn es el número de filas/líneas.

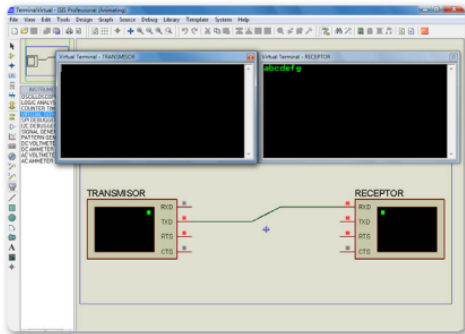
Una terminal tonta es aquella que muestra líneas de texto pero no tiene códigos de comando para mover cosas a ubicaciones específicas en la pantalla o realizar otras operaciones básicas. Muchas aplicaciones de Unix, como los editores de texto, no funcionarán con terminales tontas.

---

#### 4. ¿Qué es el eco, en relación al tipeo y una pantalla? ¿Y por qué no tengo eco en el terminal virtual de proteus?

---

El eco en relación al tipeo y una pantalla es hacer que se reflejen en la pantalla los caracteres que escribimos, para eso debemos activar la función (Echo Typed Characters).



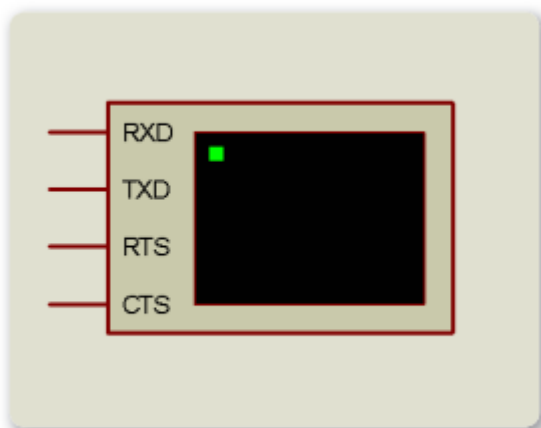
Una comunicación entre dos terminales virtuales realizada con éxito en la simulación.

---

#### 5. Explique las propiedades avanzadas del terminal virtual.

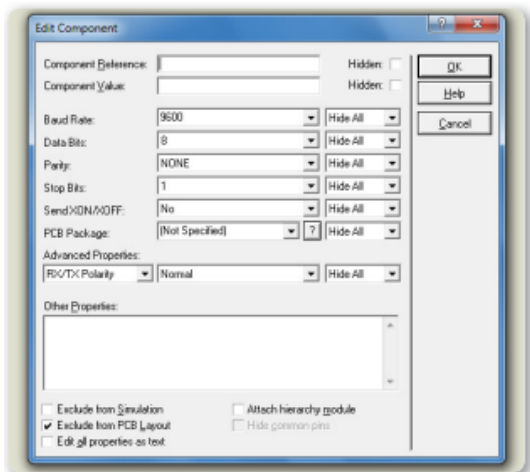
---

Una terminal virtual sirve para transmitir o recibir datos de forma serial y puede usarse para verificar las transmisiones seriales en nuestros circuitos, ya sea recibiendo datos de la terminal virtual o enviando datos hacia ella para verificar que los reciba. En la lista de instrumentos virtuales aparece como VIRTUAL TERMINAL y usa el protocolo RS232 para enviar o recibir datos.



La terminal virtual se utiliza en transmisiones de datos en serie y es completamente bidireccional.

Las terminales de este instrumento son: RXD para recibir datos; TXD para enviar datos en formato ASCII desde el teclado de la PC, es decir, la terminal virtual enviará los datos ASCII que ingresemos hacia donde conectemos esta terminal; RTS (Ready to send) y CTS (Clear to send). En las propiedades de la terminal virtual podemos configurar los parámetros de la transmisión, incluyendo su velocidad.



Las propiedades de la terminal virtual permiten definir la configuración y la velocidad de la transmisión, entre otras características.

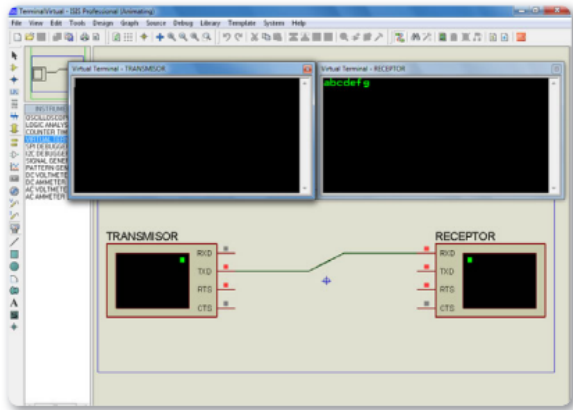
Entre las configuraciones podemos definir:

- **Baud Rate:** este campo contiene la velocidad en baudios de la transmisión serial, puede ir de 110 a 57600 baudios.
- **Data Bits (bits de datos):** para indicar cuántos bits por dato se enviarán, las opciones son 7 u 8.
- **Parity (paridad):** aquí se define el bit de paridad; las opciones son NONE (ninguno), EVEN (par) u ODD (impar).
- **Stop Bits (bits de detención):** permite elegir los bits para la detención.
- **Send XON/XOFF:** en este campo debemos definir si se enviarán los comandos XON y XOFF o no. Figura 15. Las propiedades de la terminal virtual permiten definir la configuración y la velocidad de la transmisión, entre otras características.
- **Advanced Properties:** en este campo podemos establecer la polaridad de las señales en TXD/RXD y en RTS/CTS.

La forma más fácil de ejemplificar el uso de las terminales virtuales es generando una comunicación entre dos de ellas, para lo cual, simplemente, conectamos la terminal TXD de una a la terminal RXD de la otra. Al iniciar la simulación, se mostrarán dos ventanas llamadas Virtual Terminal – TRANSMISOR y Virtual Terminal – RECEPTOR. Si hacemos un clic en la ventana del transmisor para resaltarla, podremos escribir un mensaje mediante el teclado de nuestra computadora. Los datos se transmitirán hacia el receptor y se mostrarán en su ventana. De esta manera hemos realizado una comunicación serial entre las dos terminales virtuales. Las dos terminales deben estar configuradas de forma idéntica para que la transmisión se lleve a cabo sin fallas; el nombre que dimos a las terminales nos permite identificarlas en la simulación. Si hacemos un clic con el botón derecho sobre la ventana de una terminal, se abrirá un menú contextual que contiene algunas opciones de configuración. Encontramos: borrar la pantalla (Clear Screen), pausar la transmisión (Pause), copiar o pegar (Copy/Paste), hacer que se reflejen en la pantalla los caracteres que escribimos (Echo Typed Characters), cambiar a modo

hexadecimal (Hex Display Mode) o modificar la fuente que se mostrará en la ventana de la terminal (Set Font).

**Advanced Properties:** en este campo podemos establecer la polaridad de las señales en TXD/RXD y en RTS/CTS.



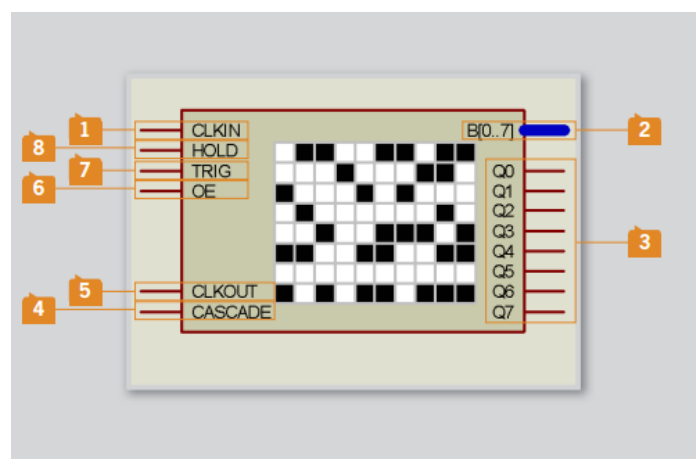
Una comunicación entre dos terminales virtuales realizada con éxito en la simulación.

### Textos automáticos al inicio de la simulación

Podemos especificar una cadena de texto que se enviará de forma automática al iniciar la simulación en una terminal virtual. Para hacerlo, en las propiedades de la terminal virtual debemos escribir, por ejemplo, en el campo Other Properties: TEXT=Hola o {TEXT=Hola} (las llaves se usan para que este atributo esté oculto). Esto enviará automáticamente el texto Hola al iniciar la simulación a través de la terminal TXD de esa terminal virtual.

### Generador de patrones digitales

Otro instrumento de gran utilidad es el generador de patrones digitales, que sirve para generar series de datos binarios en paralelo de forma personalizada. Se pueden generar patrones simples o complejos con una capacidad de hasta 1k x 8, es decir, hasta 1024 datos de 8 bits cada uno.





1. CLKIN (ENTRADA DE RELOJ): esta terminal sirve para conectar una señal de reloj en caso de usarse en modo de reloj externo.
2. B[0..7]: bus de salida de datos.
3. Q0 A Q7: terminales de salida de datos.
4. CASCADE (CASCADA): esta terminal es usada para conectar otros generadores de patrones en cascada. Se pone en estado alto durante la salida del primer dato de la secuencia y permanece en estado bajo durante el resto del tiempo.
5. CLKOUT (SALIDA DE RELOJ): es la salida de la señal de reloj interno.
6. OE (HABILITAR SALIDA): se usa para habilitar la salida de datos o inhabilitarla, poniendo las salidas en estado de alta impedancia. La salida se habilita con un estado alto en ella y se deshabilita con un estado bajo; el generador continuará funcionando aun si las salidas están deshabilitadas.
7. TRIG (DISPARO): se utiliza para dar un disparo externo al generador y que este inicie su funcionamiento.
8. HOLD (PAUSA): permite detener o pausar el funcionamiento del generador. Se detendrá en el punto donde se encuentre al habilitar este pin con un estado alto, y continuará al regresar al estado bajo.

Es posible configurar el generador de patrones para que funcione con reloj interno o externo, además de poder manejar su funcionamiento mediante los pines de control (HOLD, TRIG, OE). La salida de datos del generador será en las terminales Q0 a Q7, en el bus de datos B[0..7], o en ambos. El generador de patrones puede ser configurado ya sea mediante su ventana de propiedades o también de manera interactiva, es decir, durante la simulación.

---

## 6. Como funciona COMPIM y para qué sirve? Que es un virtualizador de puertos, de ejemplos de los más utilizados.

---

COMPIM modela un puerto serie físico. Almacena la comunicación serie recibida y la presenta como señales digitales al circuito. Cualquier dato serie transmitido desde el modelo UART o la CPU también viaja a través del puerto serie del ordenador. Existen soluciones alternativas que se pueden usar para crear un puerto serie virtual usando conectividad Bluetooth o USB. Otra característica del modelo COMPIM es que proporciona conversión de velocidad de transmisión. También hay una verificación opcional de software y hardware, que se puede implementar para abordar los aspectos físicos y virtuales del dispositivo.

Los puertos virtuales ofrecen una opción para conectar el hardware externo al software de simulación. El diseñador puede interactuar con cualquiera de los **módulos basados en UART** a través de los puertos virtuales en el software Proteus.

Hoy en día, varios **módulos de sensores como GSM, GPS, RTC (reloj en tiempo real)** se comunican a través del Protocolo de comunicación UART. Dichos módulos se pueden interconectar con los puertos virtuales en Proteus. Además, estos módulos pueden comunicarse con los microcontroladores dentro del software de simulación. Esto elimina el requisito de componentes de hardware para probar su idea de circuito. Entonces, el diseñador primero puede probar el circuito e implementar el programa integrado con la ayuda del software Proteus y, si los resultados son satisfactorios, puede implementarse utilizando los componentes de hardware adecuados.

### Módulos basados en UART



Este es básicamente un puerto serie, con protocolo **RS-232**. Se debe usar un IC de conversión de nivel como **MAX 232** para conectar con los microcontroladores. Pero, si el puerto serial no está disponible en su computadora, **los convertidores USB a serial** están disponibles. La salida de este

convertidor es de 5 voltios o 3,3 voltios **TTL Logic**. Esta salida se puede conectar a microcontroladores directamente. Se debe instalar un controlador de dispositivo adecuado.

## Convertidor USB a serie

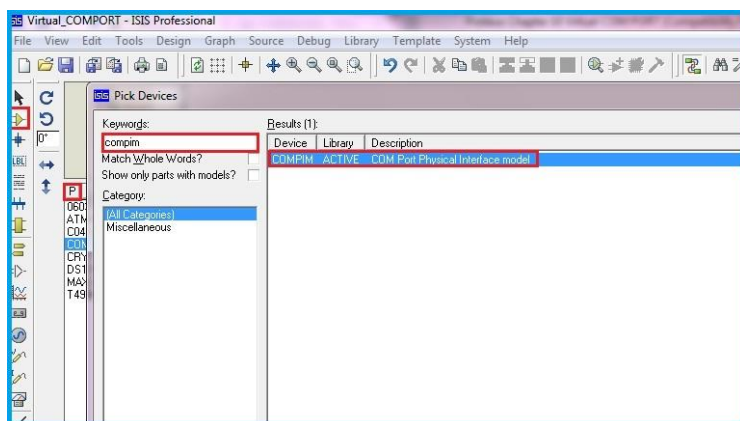


---

## Uso del PUERTO COM virtual en Proteus

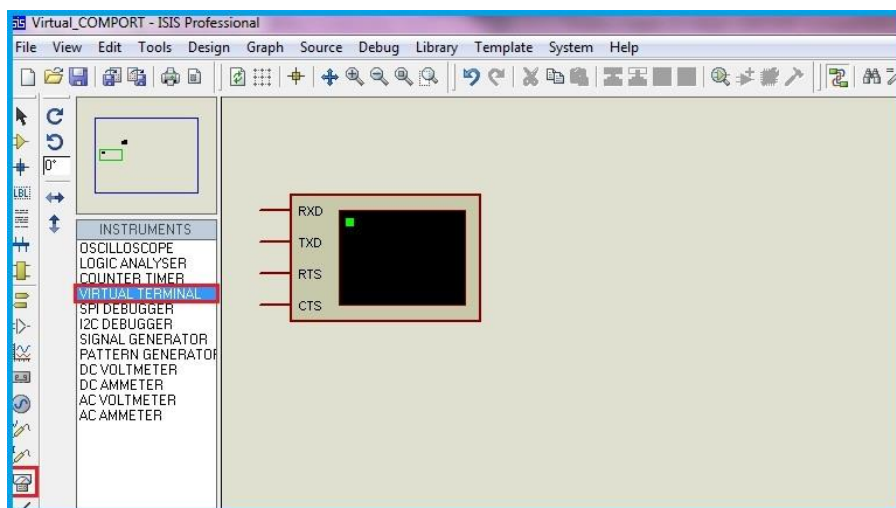
---

**Com Port** se encuentra en la biblioteca activa de componentes. Se describe como **Modelo de interfaz física - PIM**. Para la comunicación en serie, los pines RX y TX son suficientes. Pero para la comunicación real del protocolo RS 232 también se utilizan los terminales restantes.



## COM PORT in Proteus

Se utiliza un terminal virtual para enviar y recibir los datos a través del puerto. Los datos se pueden enviar a través del teclado y los datos recibidos se muestran en el terminal virtual.

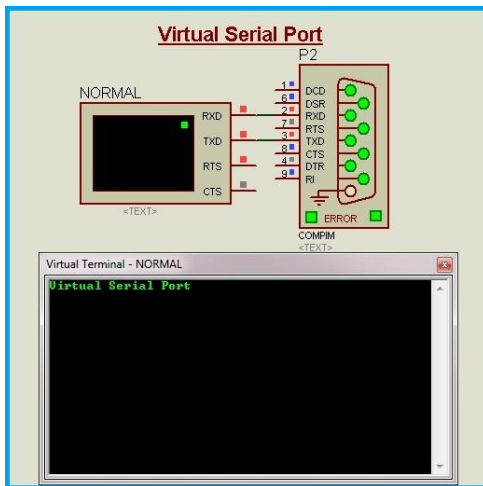


---

## Terminal virtual en Proteus

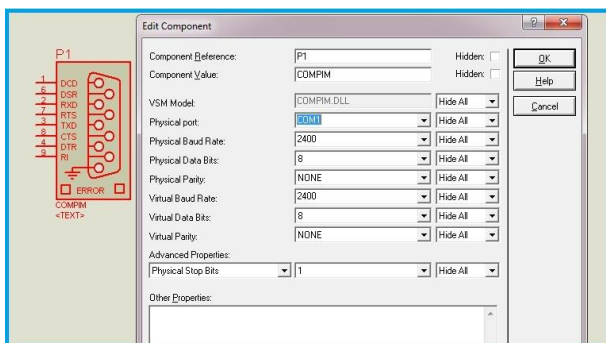
---

Este terminal virtual es como el circuito detrás del conector. Entonces, los pines RX y TX de ambos están conectados uno a uno directamente.



### Conexión del circuito virtual COM PORT

Ahora, tenemos que establecer las propiedades para la operación deseada. El número de puerto físico y la velocidad de comunicación son los parámetros esenciales que se deben configurar. Los parámetros restantes son generalmente comunes para la mayoría de las aplicaciones. Las propiedades del **terminal virtual** también deben configurarse de manera similar al **puerto COM**.



---

## Propiedades del PUERTO COM

---

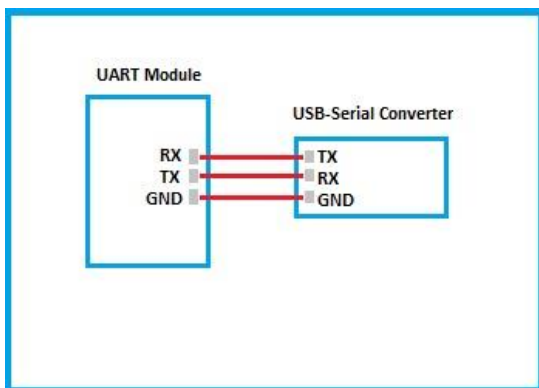
Estos parámetros deben coincidir con el módulo externo que estamos interconectando. Por lo tanto, es mejor estudiar la hoja de datos del módulo de interfaz, conocer su tasa de baudios predeterminada y otros parámetros y luego configurarlos en el software. Del mismo modo, estas propiedades deben coincidir con el Terminal virtual.

---

## Módulo UART de interfaz usando el puerto COM

---

En este ejemplo, se utiliza un **módulo GSM**. Se puede utilizar cualquier módulo con pines UART. El comando básico de los módulos UART es el **comando de atención**, es decir, los comandos AT. Tenemos que escribir AT y presionar la tecla enter en la terminal virtual. Estos datos se transfieren a través del puerto COM seleccionado y los datos enviados por el módulo se muestran en el terminal virtual.



Conexión del módulo GSM con convertidor de USB a serie

La imagen de arriba muestra la **conexión de un módulo GSM** y el convertidor de USB a serie. El convertidor USB-Serie se mostrará en los dispositivos e impresoras como *puerto de comunicación USB-Serie*. También se mostrará el número de puerto real.