¿Qué es virtual terminal en **Proteus**?

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

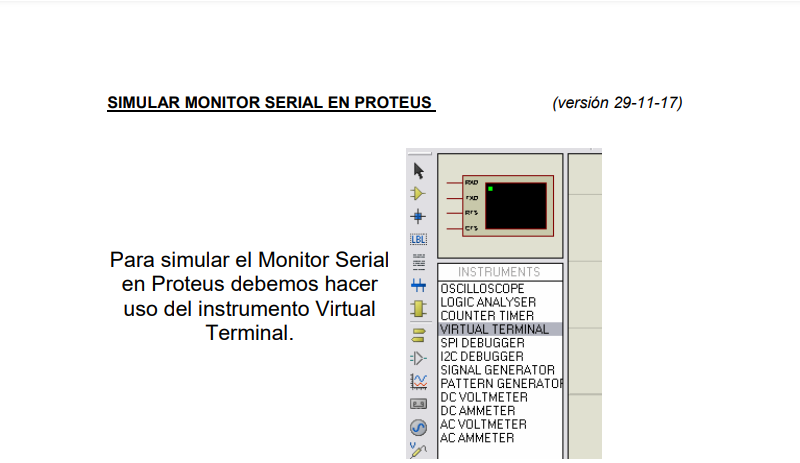
Sus principales caracteristicas son: Completo soporte bi-direccional. Los datos recibidos se visualizan como caracteres ASCII y las teclas pulsadas se transmiten como datos serie ASCII.

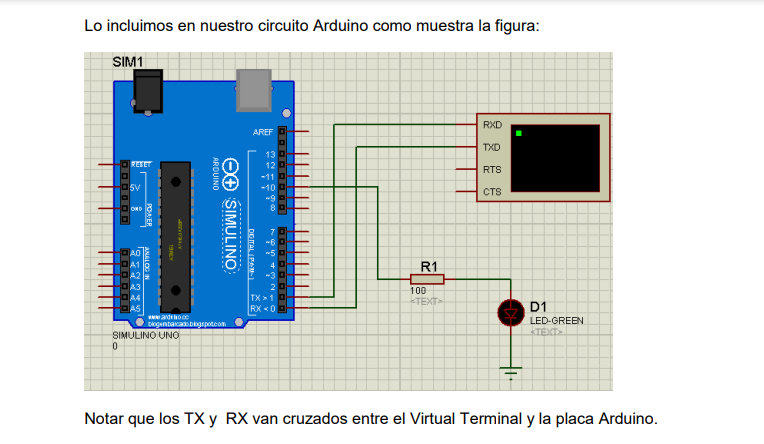
¿Cómo usar el monitor serial en Proteus?

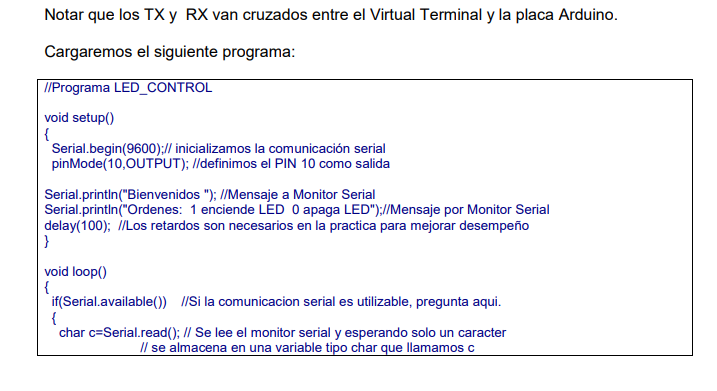
Para **simular el Monitor Serial en Proteus** debemos hacer **uso** del instrumento Virtual Terminal. Podemos tipear las ordenes si nos posicionamos dentro de la ventana del Virtual Terminal. Si observamos en **Proteus**: Otro ejemplo: En este caso enviaremos números de 3 dígitos para las órdenes.

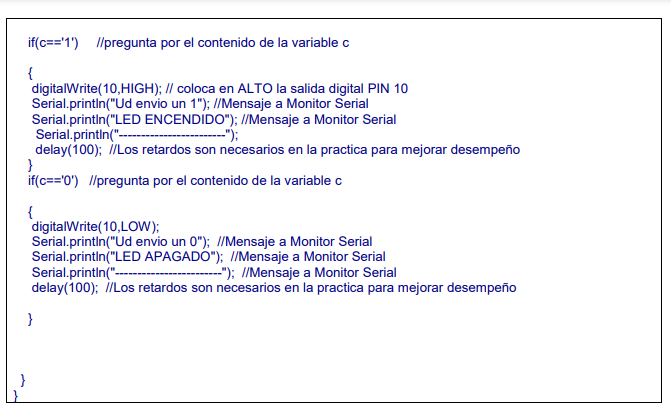
¿Cómo ver el monitor serial en Proteus?

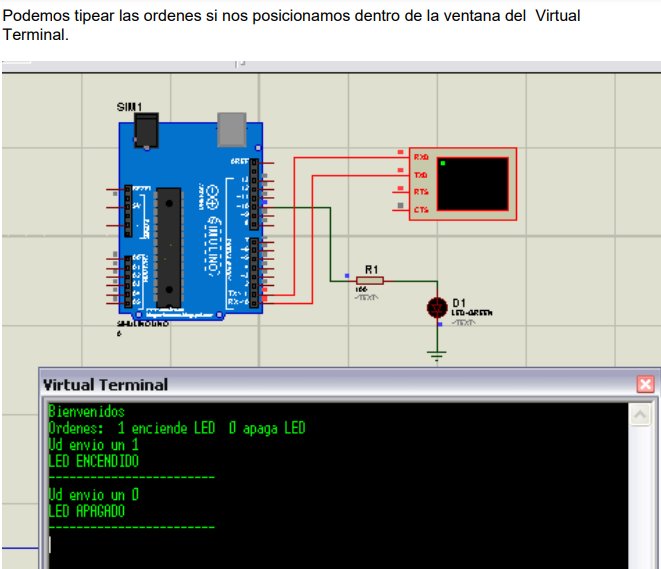
Para simular el **Monitor Serial en Proteus** debemos hacer uso del instrumento Virtual Terminal. Podemos tipear las ordenes si nos posicionamos dentro de la ventana del Virtual Terminal. Si observamos en **Proteus**: Otro ejemplo: En este caso enviaremos números de 3 dígitos para las órdenes











**Otro ejemplo:**

**//PROGRAMA LED\_CONTROL1**

**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: 1 enciende LED 0 apaga LED");//Mensaje por Monitor Serial delay(100); //Los retardos son necesarios en la practica para mejorar desempeño**

**}**

**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**

**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 practica 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**

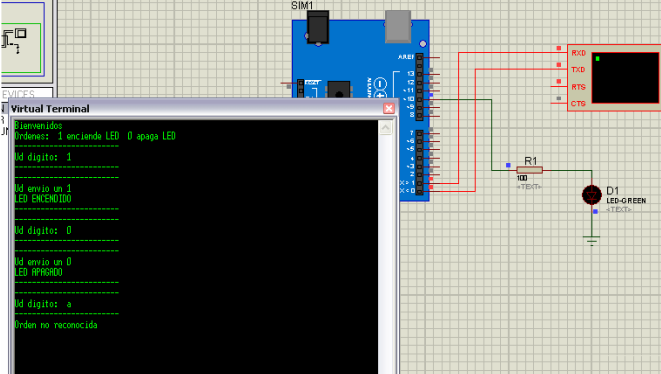
**Serial.println("------------------------"); //Mensaje a Monitor Serial**

**delay(100); //Los retardos son necesarios en la practica 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 leido 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**

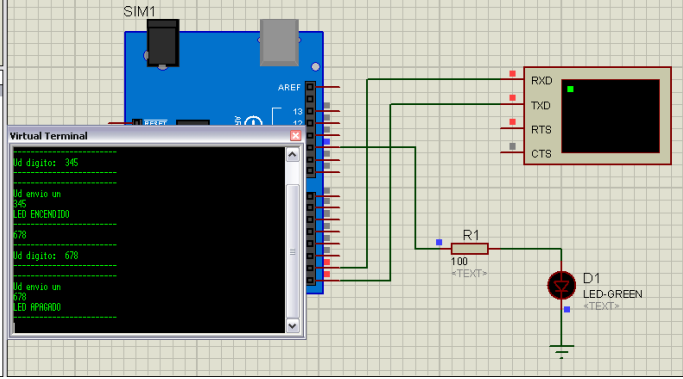
**Serial.println("Ud envio un "); //Mensaje a Monitor Serial**

**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.**

**1** **¿Qué es PlatformIO?**

**2** **Instalación**

**3** **Crear un nuevo proyecto**

**4** **Compilando y ejecutando un proyecto.**

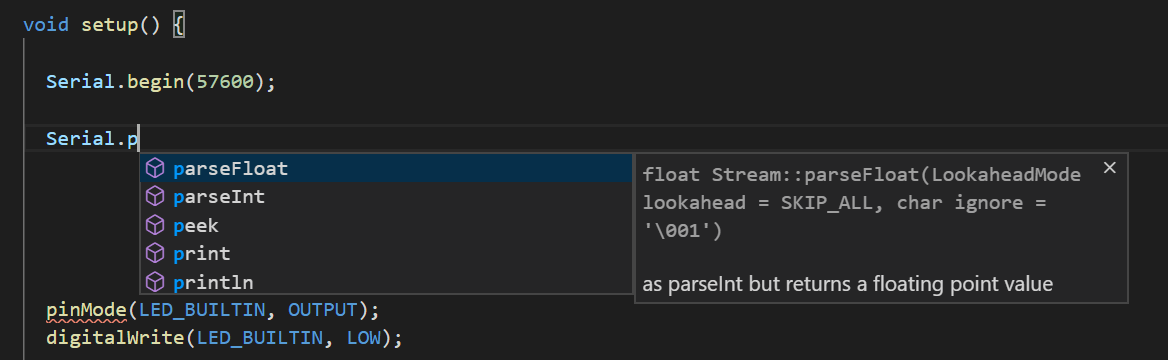
**5** **Configuración platformio.ini**

**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](https://platformio.org/)**, 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**.



**Ejemplo de autocompletado en Arduino**

## ¿Qué es PlatformIO?

PlatformIO es un ecosistema para el desarrollo de**sistemas embarcados** e IoT, el cuál facilita y mucho la gestión de proyectos de software embarcado, gestión de dependencias y librerías, tests,… Es un poco, todo lo que se le puede pedir a un ecosistema así. De momento, **no le he encontrado ningún problema** en su plantemiento (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.

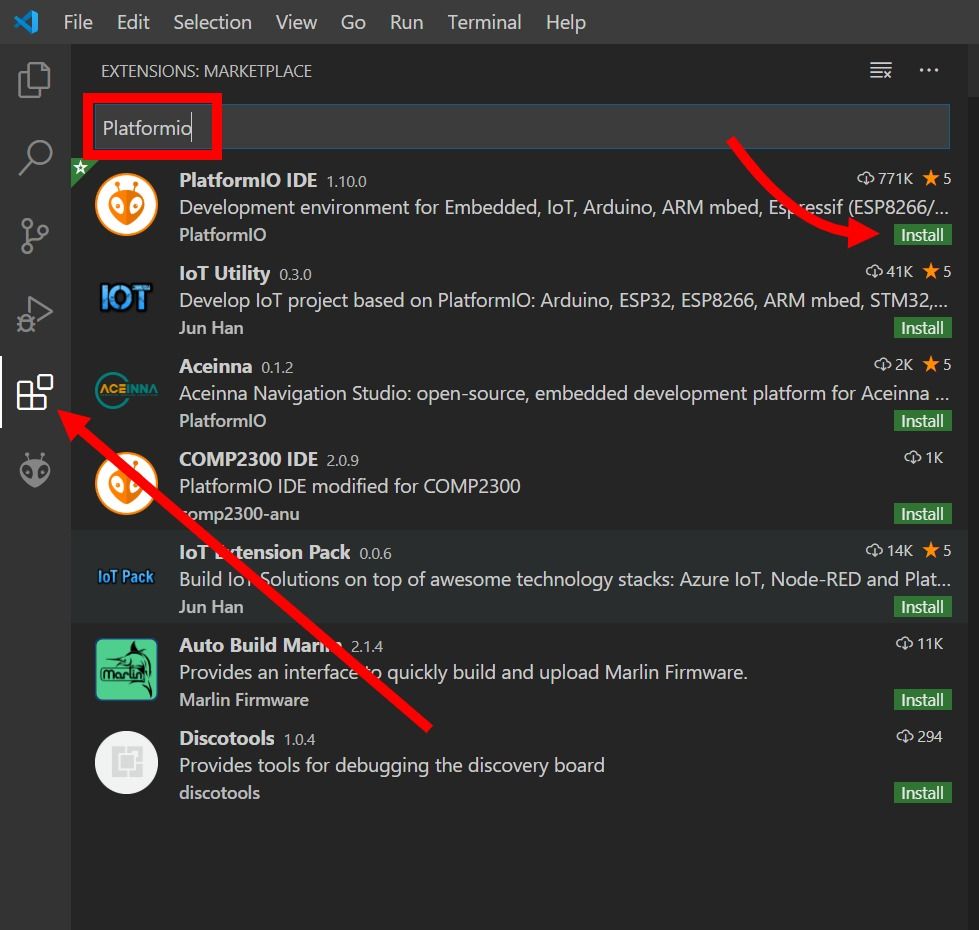
Además, te abstrae de toda la configuración que hay detrás de los frameworks. En algunos casos esto es malo, ya que pierdes el control de lo que se está haciendo por debajo. Aunque puedes **prácticamente configurar todos** los parámetros de compilación, flasheo y ‘monitoreo’, el funcionamiento básico **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**](https://code.visualstudio.com/), qué es el IDE de Microsoft, la versión gratuita. Una vez que la abráis teneis 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 tengáis instalado, lo único que tenéis que hacer es crear un nuevo proyecto, para ello vais a lo que se llama el **PIO Home** , que es como la pantalla de inicio del PlatformIO. Desde esta parte podéis hacer todo lo referente a la gestión de proyectos del PlatformIO. Si no lo veis, en la parte izquierda os ha tenido que salir una **nueva pestaña** con la cabeza de una aveja. Si abrís esa pestaña, os saldra a la izquierda abajo un menu **Quick Access** con un elemento: **PIO Home**, le hacéis click y luego a **Projects & Configuracion** (debajo del PIO Home). Os saldrá una lista con los proyectos que tenéis, y un botón para añadir nuevo proyecto.En la ventana que os tiene que salir, solo se piden 3 campos:

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

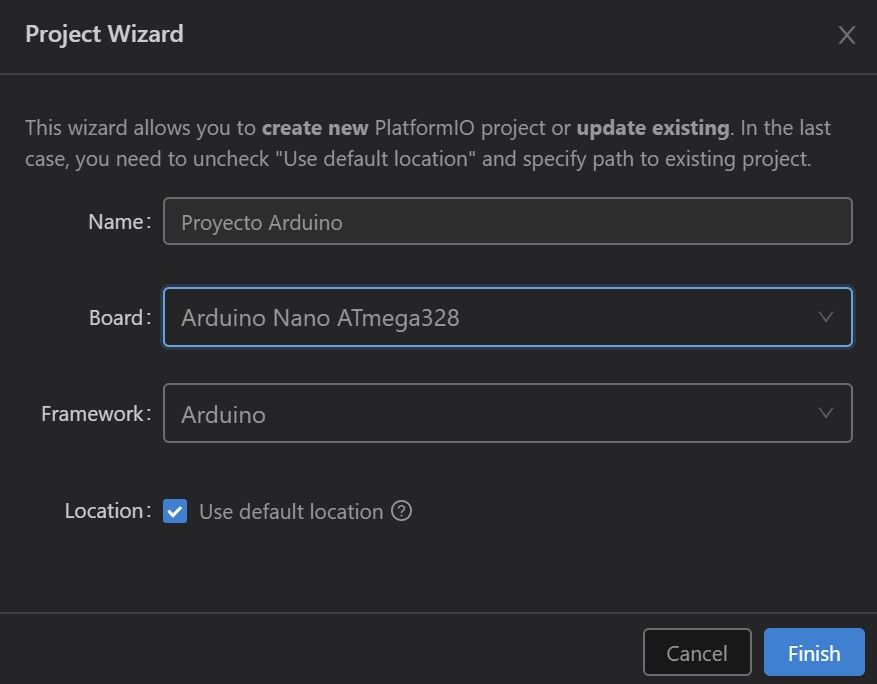
**Board** o placa de desarrollo. Aquí tenéis que buscar vuestra placa, según vais escribiendo os irá mostrando los resultados más parecidos. Algunos ejemplos (hay hasta casi 900 tarjetas):

**Arduino Nano ATmega168**

**DOIT ESP32 DEVKIT V1**

**BluePill F103C8**

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



**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 framework.

## Compilando y ejecutando un proyecto.

**es suficiente**. 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 tu ficheros, es decir los fichero .h

**lib/**

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

**source/**

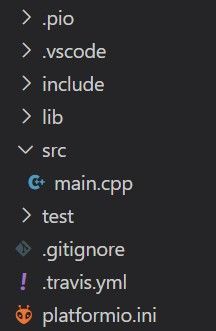
para guardar el código fuente (.c y .cpp)

**test/**

aquí puedes definir tus tests 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**



**Proyecto base platformIO**

De echo, 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 os habrán aparecido unos iconos en la **parte inferior izquierda** para poder trabajar de manera más cómoda. Son como los de el IDE de Arduino,  además de funciones**más avanzadas**. Os dejo 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 os 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 vuestro micro.

monitor\_speed = 57600

**Puerto monitor:** Podéis cambiar el puerto que se configura por defecto gracias a

monitor\_port = ...

. Aquí podes poner un puerto COM, o un socket, o lo que querais.

**Conclusión**

Dejando un primer vistazo al **platformIO**, al **probar, y si se tiene mil placas de diferente 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.

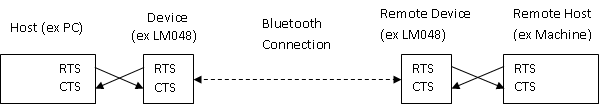
*2.    Que función tienen los terminales RTS y CTS en el terminal virtual?*

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 = Borrar 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:

Diagrama

Descripción generada automáticamente

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.

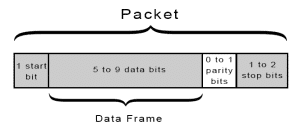
## CÓMO FUNCIONA UART

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:

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

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:

[](https://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Packet-Frame-and-Bits-2.png)

## 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](https://en.wikipedia.org/wiki/Baudot_code)[ASCII](https://en.wikipedia.org/wiki/ASCII) verdadero ), 8 (para la mayoría de los tipos de datos, ya que este tamaño coincide con el tamaño de un [byte](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Least_significant_bit) . También es posible, pero rara vez se usa, es [el bit más significativo](https://en.wikipedia.org/wiki/Most_significant_bit) primero; esto fue utilizado, por ejemplo, por el terminal de impresión [IBM 2741 .](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Error_correction) de errores en cada carácter y [protocolos de comunicación.](https://en.wikipedia.org/wiki/Communication_protocol) 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](https://en.wikipedia.org/wiki/Line_noise) , se dispusieron teleimpresores electromecánicos para imprimir un carácter especial cuando los datos recibidos contenían un error de paridad.

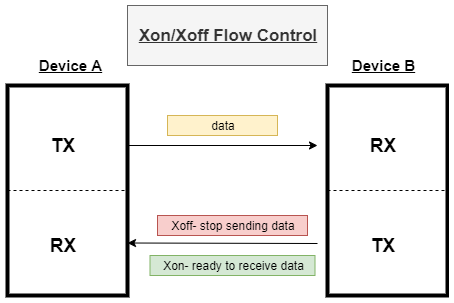
### Stop Bits

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](https://en.wikipedia.org/wiki/Teleprinter) 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](https://www.brainboxes.com/faq/Public-FAQs/FAQs.-Serial/%5bNew%5d-What-is-DTR-DSR-Hardware-Flow-Control) o [RTS/CTS](https://www.brainboxes.com/faq/Public-FAQs/FAQs.-Serial/%5bNew%5d-What-is-RTS-CTS-Hardware-Flow-Control) 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 transmisión | DC3 | 19 | 13 | Control + S |
| XON | Reanudar transmisión | DC1 | 17 | 11 | Control + Q |
|  |  |  |  |  |  |

### ***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](https://www.brainboxes.com/faq/Public-FAQs/FAQs.-Serial/%5bNew%5d-What-is-RTS-CTS-Hardware-Flow-Control) y [DTR/DSR](https://www.brainboxes.com/faq/Public-FAQs/FAQs.-Serial/%5bNew%5d-What-is-DTR-DSR-Hardware-Flow-Control) . 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. telnetoriginalmente estaba destinado a permitir que los terminales emitan comandos a las CPU remotas, y sshpretende ser un reemplazo del telnetprotocolo 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](https://www.cs.odu.edu/~cs252/Book/termtypes.html" \l "ftn.d0e902) ] , 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.

[ [6](https://www.cs.odu.edu/~cs252/Book/termtypes.html" \l "d0e902) ] 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.