

¿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.

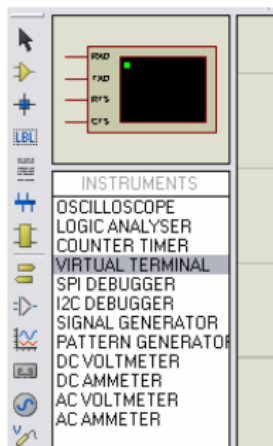
¿Cómo ver el monitor serial en Proteus?

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

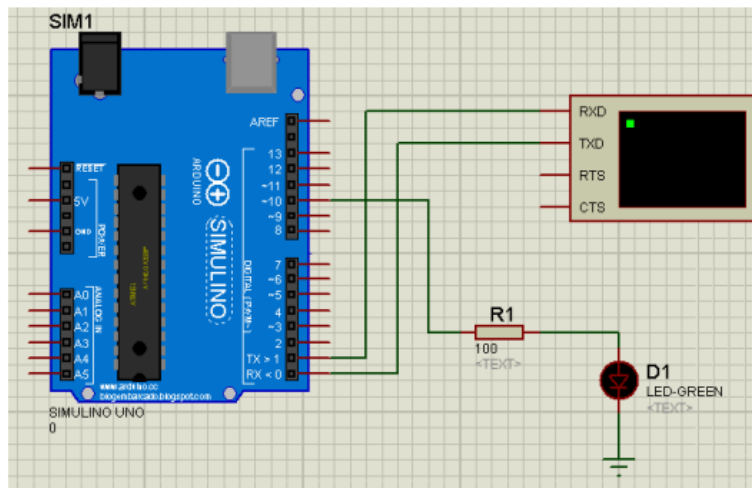
SIMULAR MONITOR SERIAL EN PROTEUS

(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 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 práctica para mejorar desempeño
}

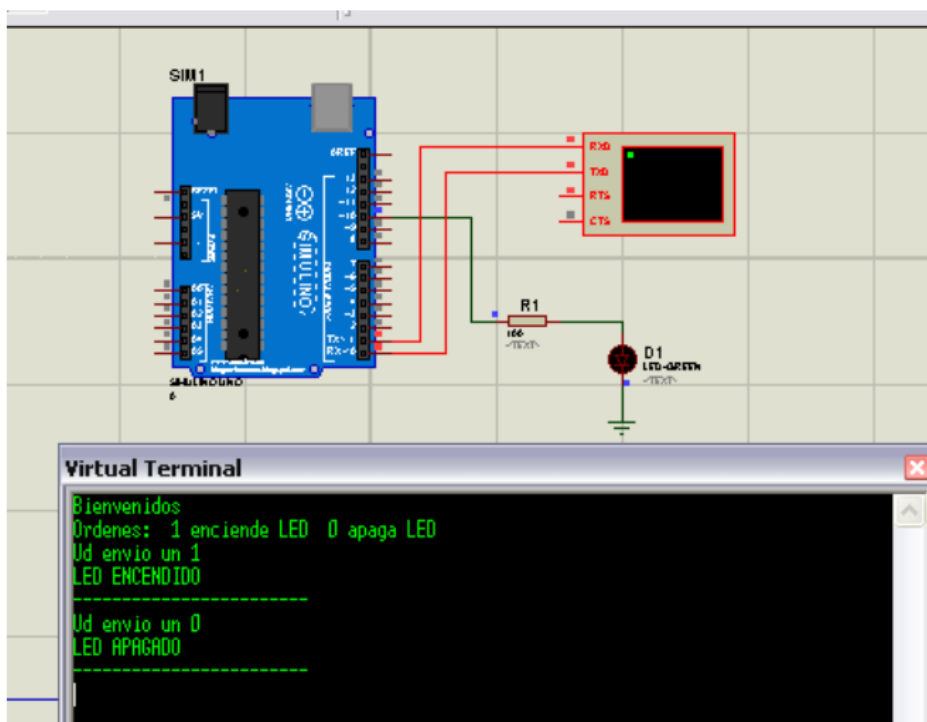
void loop()
{
  if(Serial.available()) // Si la comunicación serial es utilizable, pregunta aquí.
  {
    char c = Serial.read(); // Se lee el monitor serial y esperando solo un carácter
    // se almacena en una variable tipo char que llamamos c
  }
}
```

```

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 desempeño
}
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 desempeño
}
}
}

```

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
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

{

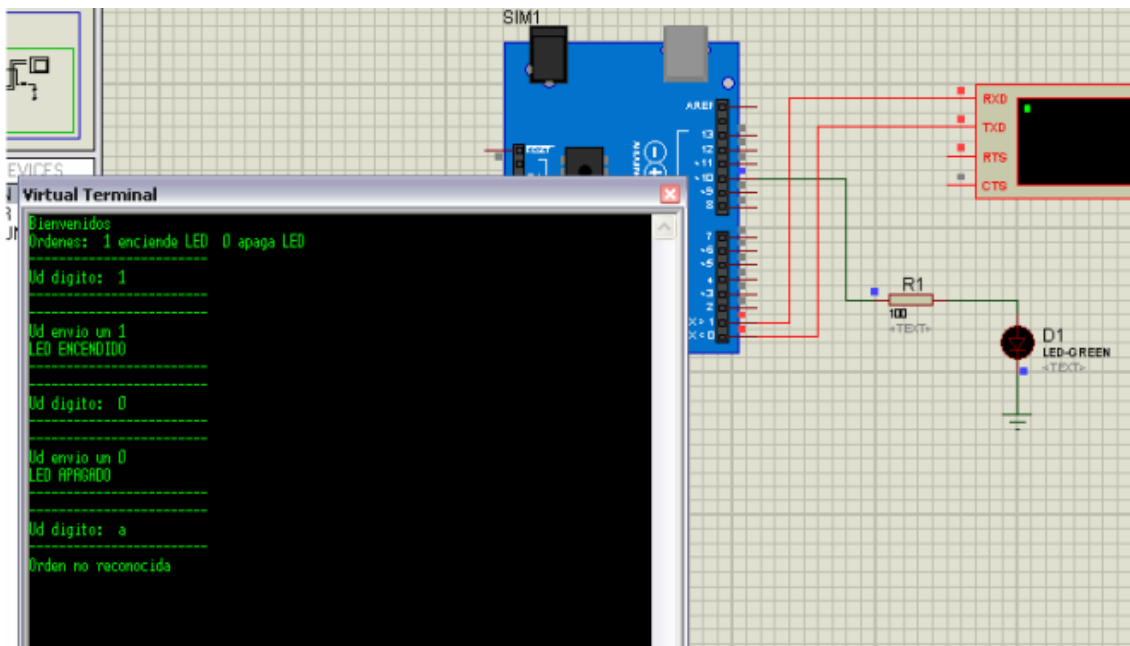
digitalWrite(10,LOW); Serial.println("-----"); //Mensaje a
Monitor Serial

    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 leído en la variable num
//Luego podemos preguntar sobre el valor
// de dicha variable - Por ejemplo
// en Tachos LED su valor selecciona color

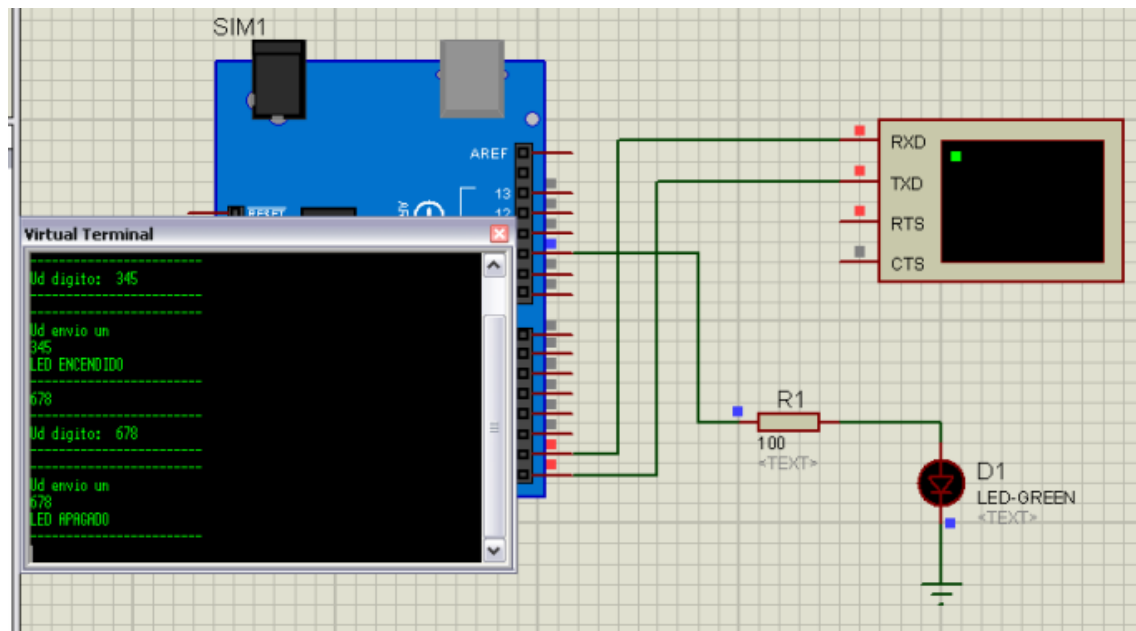
Serial.println("-----"); //Mensaje a Monitor Serial
Serial.print("Ud digito: "); //Mensaje a Monitor Serial
Serial.println(num); //Mensaje a Monitor Serial
Serial.println("-----"); //Mensaje a Monitor Serial

    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.

[1 ¿Qué es PlatformIO?](#)

[2 Instalación](#)

[3 Crear un nuevo proyecto](#)

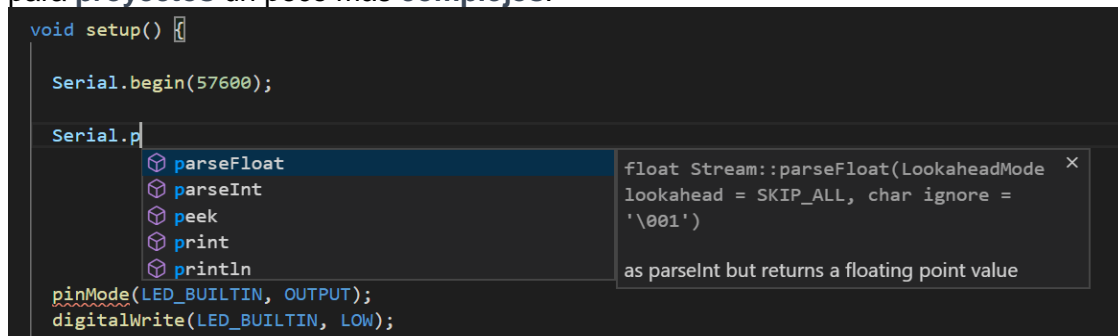
[4 Compilando y ejecutando un proyecto.](#)

[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 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.

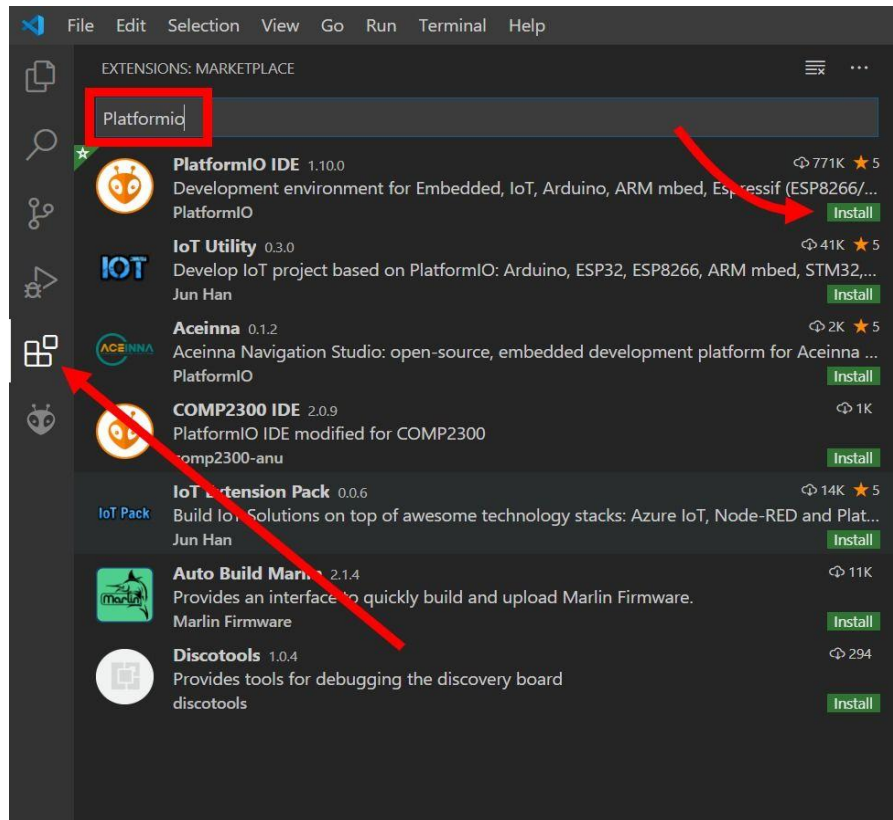
Además, te abstrae de toda la configuración que hay detrás de los *frameworks*. En algunos casos esto es malo, ya que pierdes el control de lo que se está haciendo por debajo. Aunque puedes **prácticamente configurar todos** los parámetros de compilación, flasheo y 'monitoreo', el funcionamiento básico **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**, 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 abríis 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)

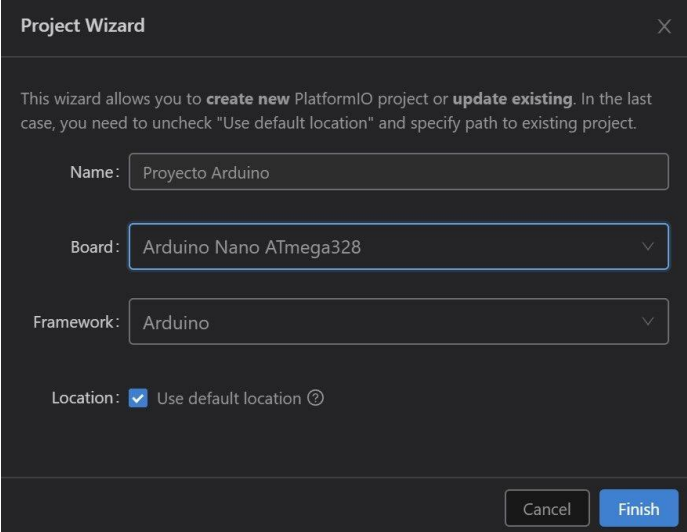
Board o placa de desarrollo. Aquí tenéis que buscar vuestra placa, según vais escribiendo os irá mostrando los resultados más parecidos. Algunos ejemplos (hay hasta casi 900 tarjetas):

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/

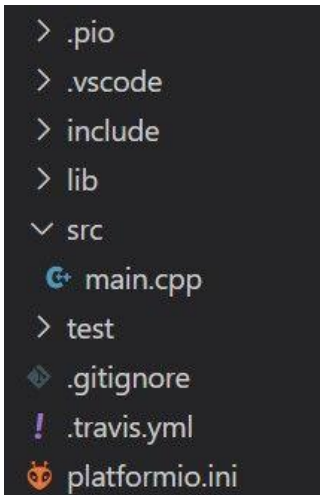
para guardar el código fuente (.c y .cpp)

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**



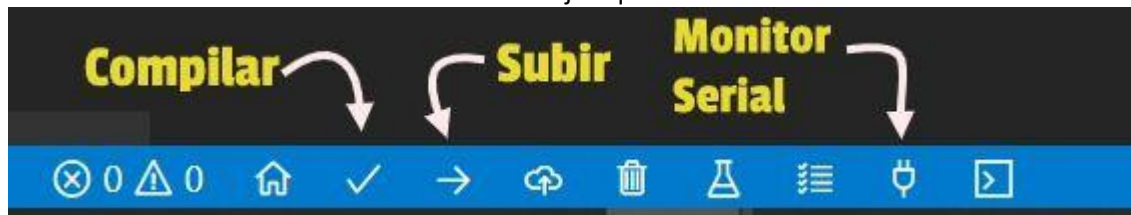
Proyecto base platformIO

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.