

Ejercicio # 2 - Punto 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

TERMINAL VIRTUAL EN PROTEUS

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.

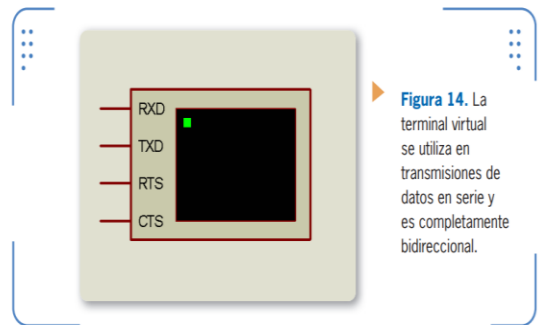


Figura 14. La terminal virtual se utiliza en transmisiones de datos en serie y es completamente bidireccional.

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

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

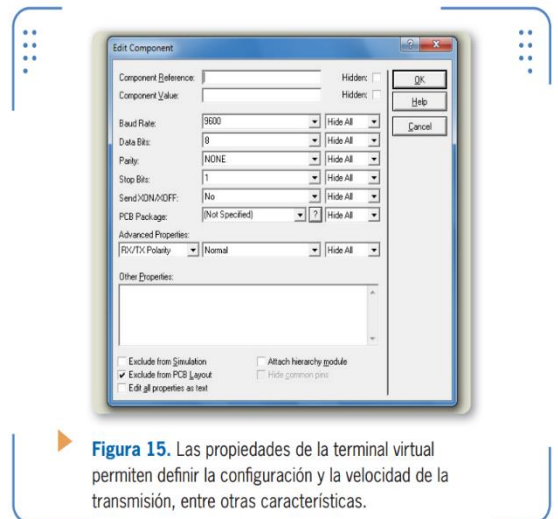
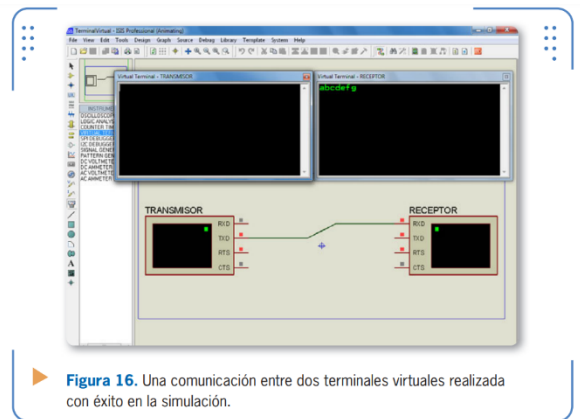


Figura 15. Las propiedades de la terminal virtual permiten definir la configuración y la velocidad de la transmisión, entre otras características.

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. Esto podemos verlo en el archivo TerminalVirtual.dsn. 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).



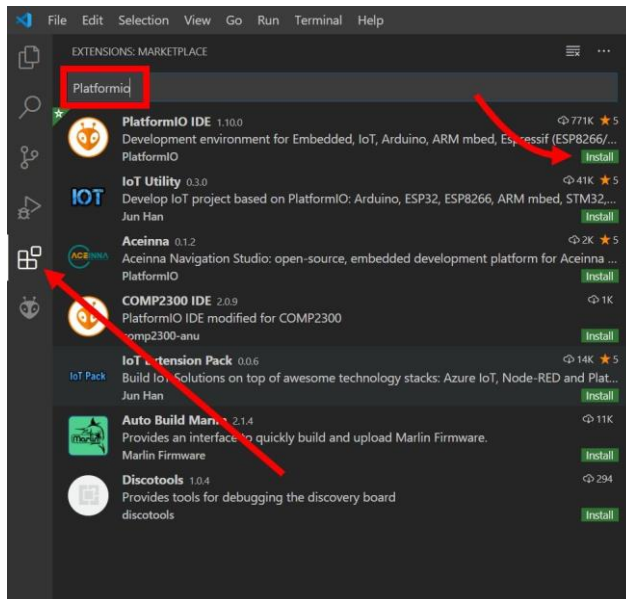
Podemos especificar una cadena de texto que se enviará de forma automática al iniciar la simulación en una terminal virtual. Para hacerlo, en las propiedades de la terminal virtual debemos escribir, por ejemplo, en el campo Other Properties: TEXT=Hola o {TEXT=Hola} (las llaves se usan para que este atributo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

A pesar de que Proteus cuenta con un modelo del circuito integrado MAX232, debemos darle prioridad al uso de terminales virtuales, ya que el modelo de estas es puramente digital, lo cual genera simulaciones mucho más ligeras. El modelo del MAX232 es analógico y produce una importante carga para la CPU.

MONITOR SERIE EN VSCODE @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. Permite de manera muy sencilla generar entornos para programar casi cualquier microcontrolador desde el mismo entorno.

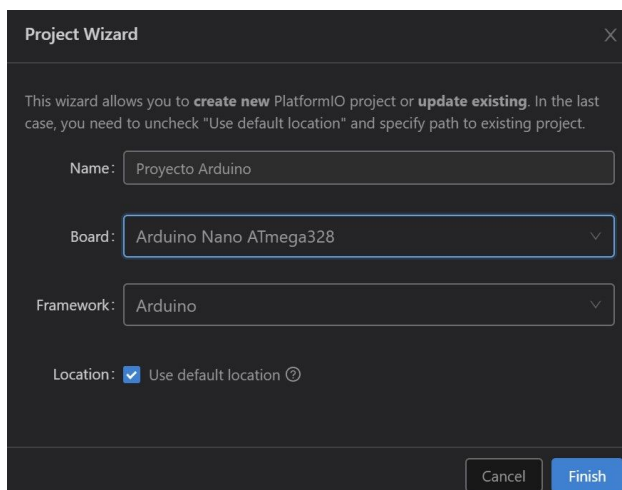
Además, te abstrae de toda la configuración que hay detrás de los *frameworks*. Se puede configurar todos los parámetros de compilación, flasheo y ‘monitoreo’, el funcionamiento básico es muy asequible. Por otra parte, permite depurar el software mientras se ejecuta en la placa, siempre que esta lo permita.



Una vez instalado, se debe crear un nuevo proyecto, para ello se accede a lo que se llama el *PIO Home*, que es como la pantalla de inicio del PlatformIO. Desde esta parte se puede hacer todo lo referente a la gestión de proyectos del PlatformIO.

En la ventana que tiene que salir, solo se piden 3 campos:

1. Nombre del proyecto (creo que es bastante autodescriptivo)
2. 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
3. Framework: Qué es el conjunto de librerías con el que se quiere programar. Aquí se puede elegir Arduino (para programar como en arduino normal), o mejor, usar librerías más avanzadas si te lo permite como *mbed* o *ESP-IDF*...



Una vez que se acepta todo, tarda un rato en generar el proyecto ya que la primera vez que se genera un proyecto se descarga todo lo necesario es decir, la información de la placa que has elegido, y el framework.

Cuando se genera un proyecto, se crean las carpetas y ficheros básicos necesarios, en algunos casos no se usarán 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 los ficheros, es decir los ficheros .h

- lib/

carpeta para generar librerías propias del proyecto, es decir librerías privadas

- source/

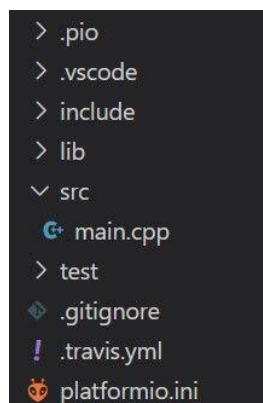
para guardar el código fuente (.c y .cpp)

- test/

aquí se pueden definir los 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 es suficiente.



El fichero *main.cpp* es el fichero principal del programa y si se abre se verá que ya está preparado para ser compilado con los *includes* necesarios. (#include <Arduino.h>)

Para compilar y subir el programa aparecen unos iconos en la parte inferior izquierda para poder trabajar de manera más cómoda.




Al puerto serie hay que cerrarlo con CTRL + T.

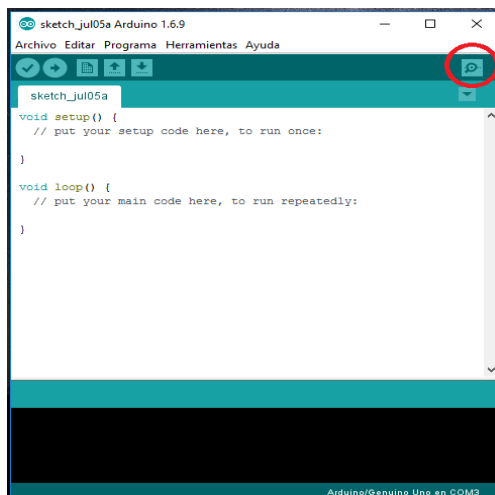
Algunos valores de configuración que suelen cambiarse son:

- Velocidad del monitor serial: Para adecuarlo a la velocidad que se configure en el micro. (**monitor_speed = ...**)
- Puerto monitor: Se puede cambiar el puerto que se configura por defecto con "**monitor_port = ...**". Acá se puede poner un puerto COM, o un socket, etc.

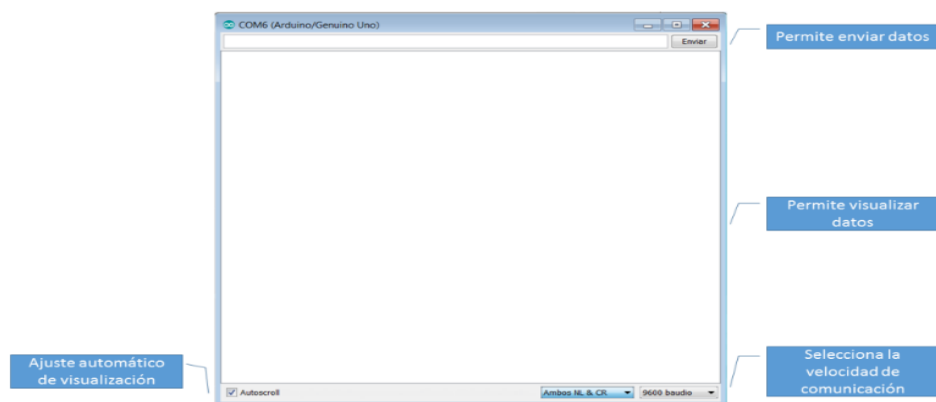
MONITOR EN SERIE EN EL IDE DE ARDUINO

El monitor serie consiste en una consola de entrada y salida, esto quiere decir que se pueden mostrar datos enviados por la placa arduino y también enviar datos a dicha placa.

Se puede acceder a él a través del menú Herramientas/Monitor serie o haciendo clic en el botón *Monitor Serie*, que está representado por el siguiente  icono: que se puede encontrar en la esquina superior derecha:



Si se hace clic en él, se abre una ventana en blanco como la de la siguiente imagen:



La comunicación serie se lleva a cabo por los pines TX / RX y usa niveles lógicos TTL (5V o 3.3V dependiendo de la placa). Esta se utiliza para la comunicación entre la placa Arduino y una computadora u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie

(también conocido como UART o USART). Por lo tanto, si se utiliza esta función, no se pueden utilizar los pines 0 y 1 para la entrada o salida digital.

Funciones básicas del Monitor serie

A continuación, se muestran algunas de las funciones básicas que permiten a comunicarse usando el monitor serie:

begin()

Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para comunicarse con el ordenador, utilice una de estas velocidades: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Sin embargo, se pueden especificar otras velocidades. Por tanto, su ejecución es imprescindible antes de realizar cualquier transmisión por dicho canal. Se debe indicar la velocidad del canal como argumento.

Sintaxis

`Serial.begin(velocidad)`

`Serial.begin(velocidad, configuracion)`

Parametro

velocidad: es la velocidad de comunicación del puerto serie en bits por segundo (baudios)

Configuracion: selecciona el numero de datos, paridad y bits de parada. Los valores validos son: SERIAL_5N1, SERIAL_6N1, SERIAL_7N1, SERIAL_8N1 (por defecto), SERIAL_5N2, SERIAL_6N2, SERIAL_7N2, SERIAL_8N2, SERIAL_5E1, SERIAL_6E1, SERIAL_7E1, SERIAL_8E1, SERIAL_5E2, SERIAL_6E2, SERIAL_7E2, SERIAL_8E2, SERIAL_5O1, SERIAL_6O1, SERIAL_7O1, SERIAL_8O1, SERIAL_5O2, SERIAL_6O2, SERIAL_7O2 y SERIAL_8O2.

end()

No tiene ningún argumento ni devuelve nada, y se encarga de cerrar el canal serie. Permitiendo que los pines RX y TX sean usados para entradas y salidas generales.

Sintaxis

`Serial.end()`

print()

Envía a través del canal serie un dato (especificado como parámetro) desde la placa arduino hacia el exterior. Ese dato puede ser de cualquier tipo: carácter, cadena, número entero, número decimal (por defecto de dos decimales), etc. Si el dato se especifica explícitamente (en vez de a través de una variable), hay que recordar que los caracteres se han de escribir entre comillas simples y las cadenas entre comillas dobles.

Sintaxis

`Serial.print(valor, formato)`

Parametros

valor: el valor a imprimir (cualquier tipo de dato)

formato: Especifica la base numérica (BIN (binario o base 2), OCT (octal o base 8), DEC (decimal o base 10), HEX (hexadecimal o base 16)) o el número de decimales (para los datos float).

println()

Hace exactamente lo mismo que `Serial.print()`, pero además, añade automáticamente al final de los datos enviados dos caracteres extra: el de retorno de carro (código ASCII nº 13)(\r) y el de nueva línea (código ASCII nº 10) (\n). La consecuencia es que al final de la ejecución de `Serial.println()` se efectúa un salto de línea.

Sintaxis

`Serial.println(valor, formato)`

Parametros

valor: el valor a imprimir (cualquier tipo de dato)

formato: Especifica la base numérica (BIN (binario o base 2), OCT (octal o base 8), DEC (decimal o base 10), HEX (hexadecimal o base 16)) o el número de decimales (para los datos float).