

Prácticas de I/O en sistemas embebidos v1.0

Bienvenido a las practicas con sistemas embebidos, en la misma desarrollaremos la ejercitación del lenguaje CPP para sistemas embebidos con [VsCode](#) @ [PlatformIO](#), con el framework de [Arduino](#).

La modalidad será la siguiente:

Cada practica se desarrollará en forma grupal, debiendo subir el desarrollo de la misma al repositorio (respetando la estructura de monorepositorio) establecido por grupo. Los ejercicios serán implementados de forma que a cada integrante le corresponda 1 o más tareas (issues); por lo que deberán crear el proyecto correspondiente, con la documentación asociada si hiciera falta, y asignar los issues por integrante. De esta forma quedara documentada la colaboración de cada alumno.

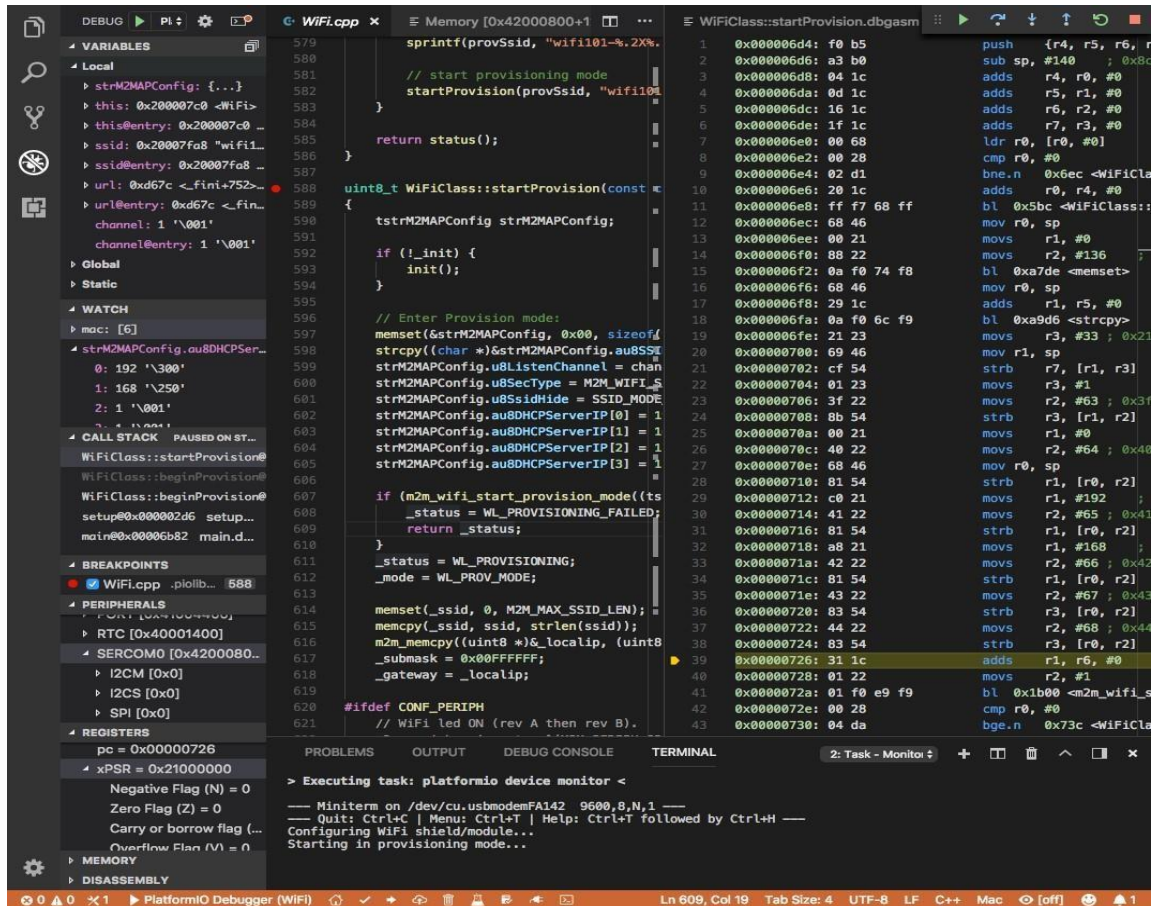
Ejercicio #2

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.

PlatformIO IDE para VSCode:

Visual Studio Code es un editor de código fuente ligero pero potente que se ejecuta en su escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C++, C#, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity)

Contenido

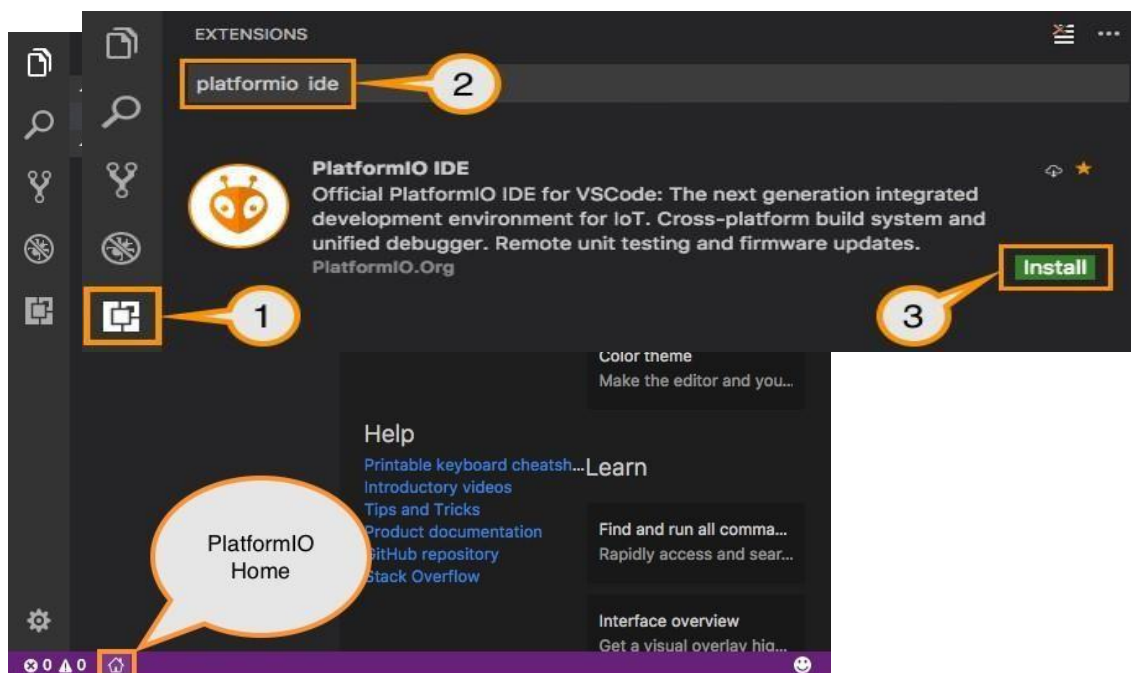


Instalación:

Tenga en cuenta que no necesita instalar PlatformIO Core por separado si va a utilizar PlatformIO IDE para VSCode. PlatformIO Core (CLI) está integrado en PlatformIO IDE y podrá usarlo dentro de PlatformIO IDE Terminal.

Si va a utilizar Git para instalar plataformas de desarrollo ascendentes, clonar proyectos externos, instalar dependencias de biblioteca desde un repositorio, asegúrese de que el comando funcione desde una terminal del sistema. De lo contrario, instale un Cliente de git.

1. Descargue e instale el código oficial de Microsoft Visual Studio.
PlatformIO IDE está construido sobre él.
2. **Abra** el administrador de paquetes VSCode
3. **Buscar** la extensión oficial platformIO ide
4. **Instale** PlatformIO IDE.

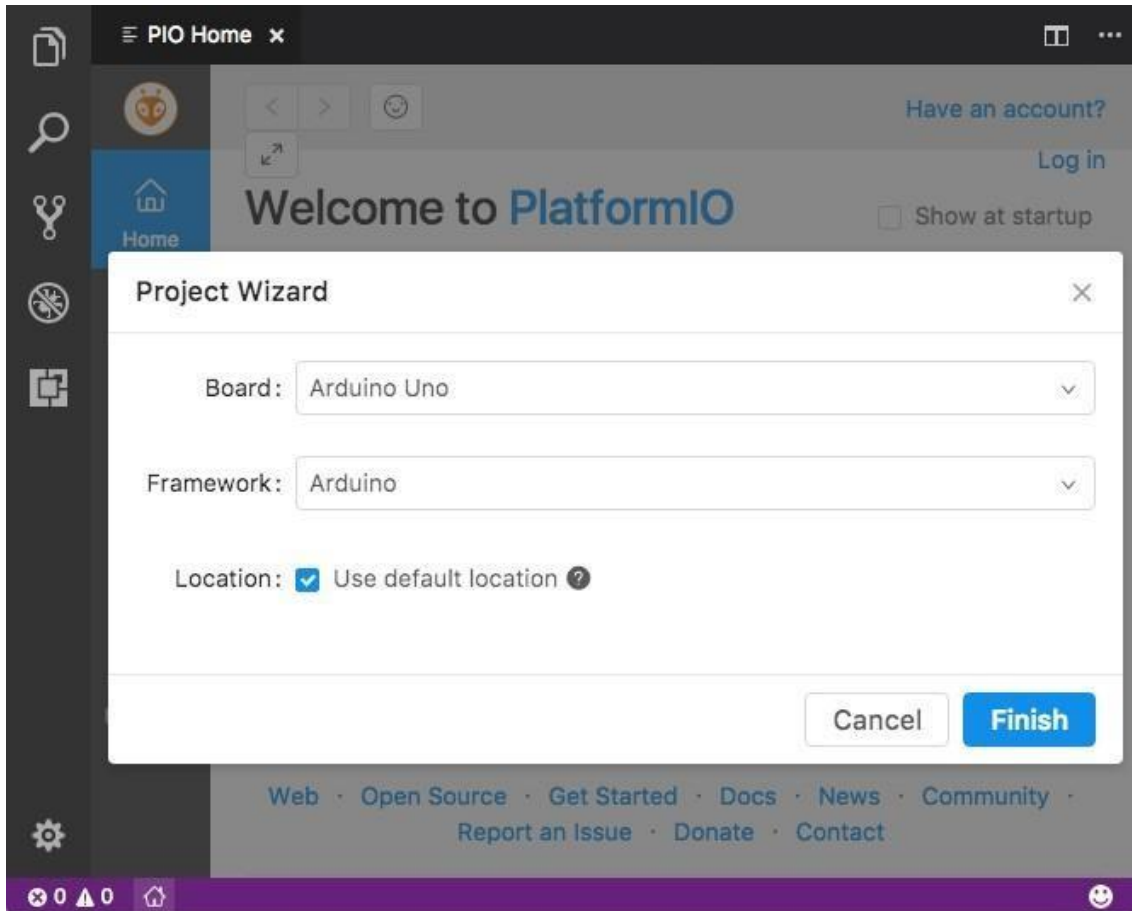


Inicio rápido:

Este informe presenta los conceptos básicos del flujo de trabajo de PlatformIO IDE y le muestra un proceso de creación de un ejemplo simple de "Blink". Después de terminar, tendrá una comprensión general de cómo trabajar con proyectos en el IDE.

Configuración del proyecto.

- 1)- Haga clic en el botón "Inicio de PlatformIO" en la barra de herramientas de PlatformIO inferior
- 2)- Haga clic en "Nuevo proyecto", seleccione una placa y cree un nuevo proyecto PlatformIO



3)- Abra la carpeta main.cpp del formulario de src. archivo y reemplace su contenido con el siguiente:

Nota: El siguiente código funciona solo en combinación con

```
/**
 * Blink
 *
 * Turns on an LED on for one second,
 * then off for one second, repeatedly.
 */
#include "Arduino.h"

// Set LED_BUILTIN if it is not defined by Arduino framework
// #define LED_BUILTIN 13

void setup()
{
  // initialize LED digital pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

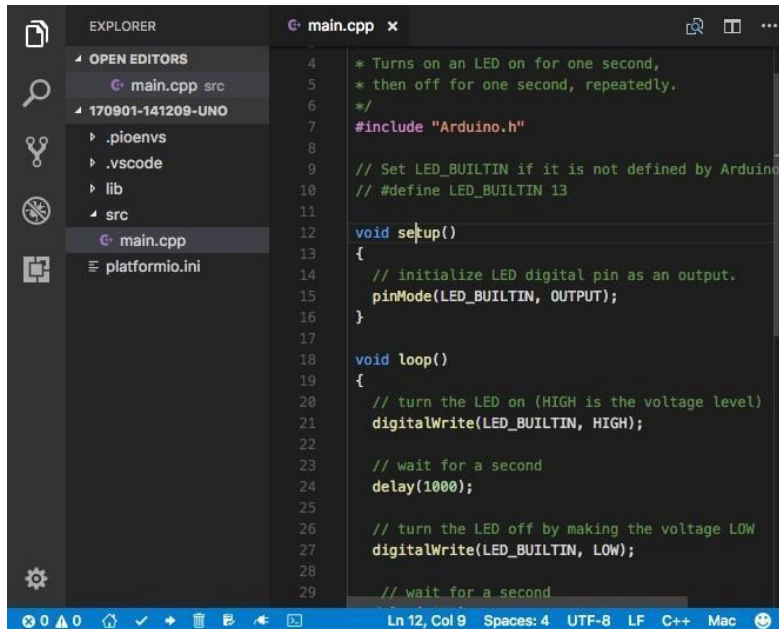
void loop()
{
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);

  // wait for a second
  delay(1000);

  // turn the LED off by making the voltage LOW
  digitalWrite(LED_BUILTIN, LOW);

  // wait for a second
  delay(1000);
}
```


Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montañó – Rodolfo Paz - Carolina Nis - Fernando Vexenat
placas basadas en Arduino. Siga el repositorio de ejemplos de proyectos de PlatformIO para ver otros proyectos preconfigurados.

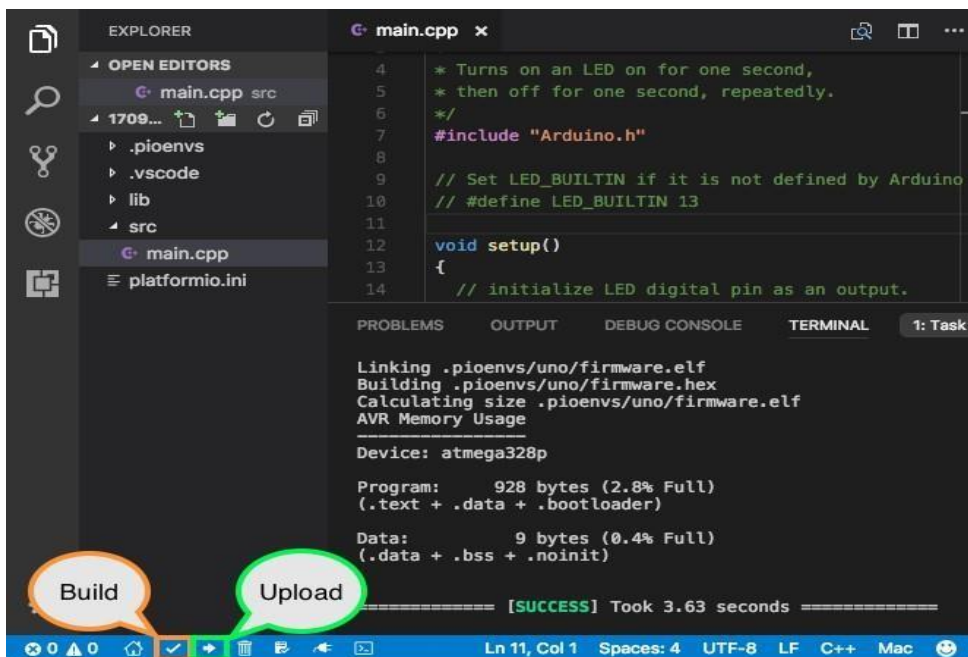


```

4  * Turns on an LED on for one second,
5  * then off for one second, repeatedly.
6  */
7  #include "Arduino.h"
8
9  // Set LED_BUILTIN if it is not defined by Arduino
10 // #define LED_BUILTIN 13
11
12 void setup()
13 {
14     // initialize LED digital pin as an output.
15     pinMode(LED_BUILTIN, OUTPUT);
16 }
17
18 void loop()
19 {
20     // turn the LED on (HIGH is the voltage level)
21     digitalWrite(LED_BUILTIN, HIGH);
22
23     // wait for a second
24     delay(1000);
25
26     // turn the LED off by making the voltage LOW
27     digitalWrite(LED_BUILTIN, LOW);
28
29     // wait for a second

```

4)- Cree su proyecto con una ctrl+alt+btecla de acceso rápido (vea todas las combinaciones de teclas en la sección "Guía del usuario" a continuación) o usando el botón "Crear" en la barra de herramientas de PlatformIO



```

4  * Turns on an LED on for one second,
5  * then off for one second, repeatedly.
6  */
7  #include "Arduino.h"
8
9  // Set LED_BUILTIN if it is not defined by Arduino
10 // #define LED_BUILTIN 13
11
12 void setup()
13 {
14     // initialize LED digital pin as an output.

```

Linking .pioenvs/uno/firmware.elf
Building .pioenvs/uno/firmware.hex
Calculating size .pioenvs/uno/firmware.elf
AVR Memory Usage

Device: atmega328p

Program: 928 bytes (2.8% Full)
(.text + .data + .bootloader)

Data: 9 bytes (0.4% Full)
(.data + .bss + .noinit)

[SUCCESS] Took 3.63 seconds

Barra de herramientas de PlatformIO

Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montaña – Rodolfo Paz - Carolina Nis - Fernando Vexenat

La barra de herramientas IDE de PlatformIO se encuentra en la barra de estado de VSCode (esquina izquierda) y contiene botones de acceso rápido para los comandos populares. Cada



botón contiene una pista (retraso del mouse sobre él).

1. PlataformalO Inicio
2. PlataformalO: Construir
3. PlataformalO: Subir
4. PlataformalO: Limpio
5. Monitoreo de puerto serie
6. PlatformIO Core (CLI)
7. Conmutador de entorno de proyecto (si hay más de un entorno disponible). Consulte

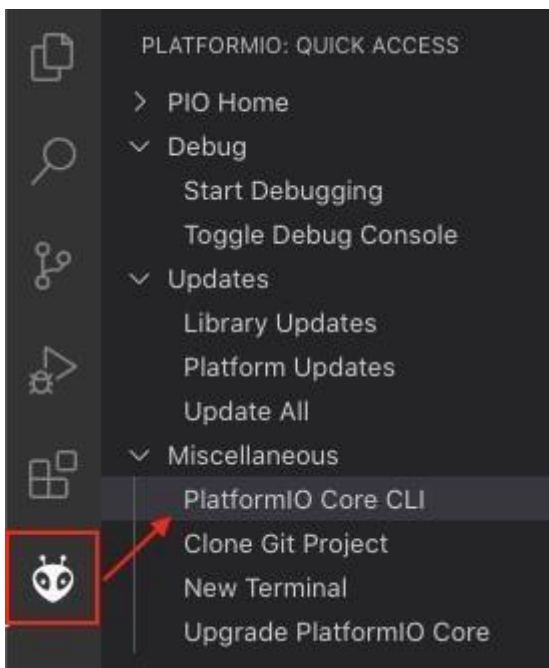
la Sección [env] de “platformio.ini” (Archivo de configuración del proyecto).

PlatformIO Core (CLI).

Hay 2 formas de acceder a PlatformIO Core (CLI) :

1. Icono de “Terminal” en la barra de herramientas de PlatformIO
2. Barra de actividad izquierda > PlatformIO (icono de hormiga) > Acceso rápido > Varios

> PlatformIO Core CLI



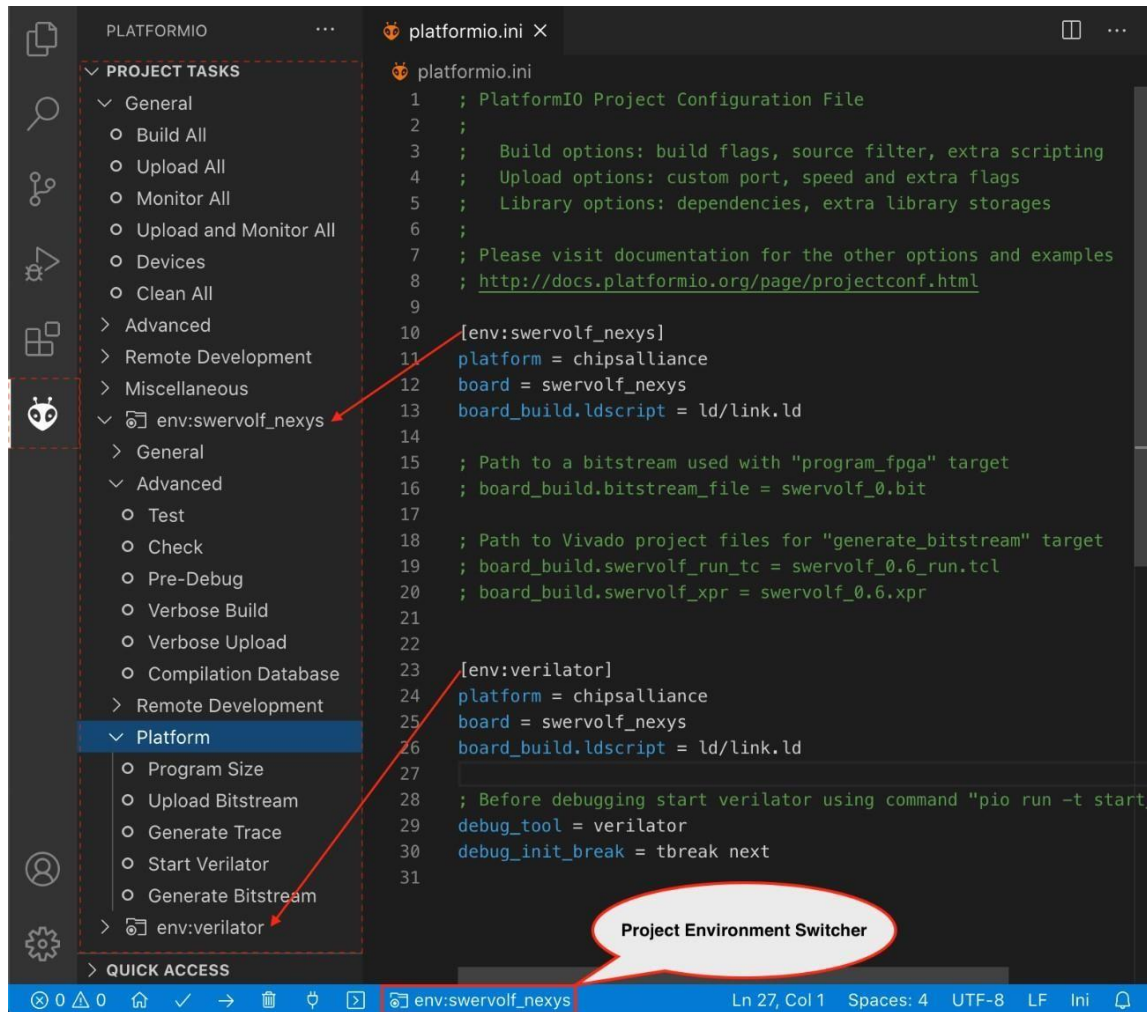
Tareas del proyecto:

Explorador de tareas.

PlatformIO proporciona acceso a la "Tarea del proyecto" donde puede controlar el proceso de construcción de los entornos declarados en "platformio.ini" (Archivo de configuración del proyecto) . El Explorador de tareas del proyecto se encuentra en la barra de actividad de VSCode, debajo del ícono de PlatformIO. También puede acceder a él a través de "VSCode Menu > Open View... > PlatformIO".

Insinuación

Tenga en cuenta que puede arrastrar/mover "Tarea del proyecto"



a otra vista dentro de VSCode, como "Explorador".

Ejecutor de tareas:

PlatformIO IDE proporciona tareas integradas a través del menú (Crear, Cargar, Limpiar, Monitorear, etc.) y tareas personalizadas por entorno "platformio.ini" (Archivo de configuración del proyecto) (.). El comportamiento predeterminado es usar paneles de terminales para la presentación, un panel dedicado a cada tarea única. Terminal > Run Task...[env:***]

El IDE de PlatformIO proporciona su propio Problems Matcher llamado \$platformio. Puede usarlo más tarde si decide cambiar la

Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montaña – Rodolfo Paz - Carolina Nis - Fernando Vexenat
configuración de la tarea base.

Puede anular las tareas existentes con sus propias opciones de presentación. Por ejemplo, configuremos PlatformIO Task Runner para usar un panel de terminal NUEVO para cada comando "Construir":

1. El elemento del menú abre una lista de tareas de VSCode para PlatformIO. En la línea, presione el icono de engranaje en el extremo derecho de la lista. Esto crea o abre el archivo con algún código de

plantilla.Terminal > Run Task...PlatformIO: Build.vscode/tasks.json

2. Reemplace la plantilla tasks.json con este código

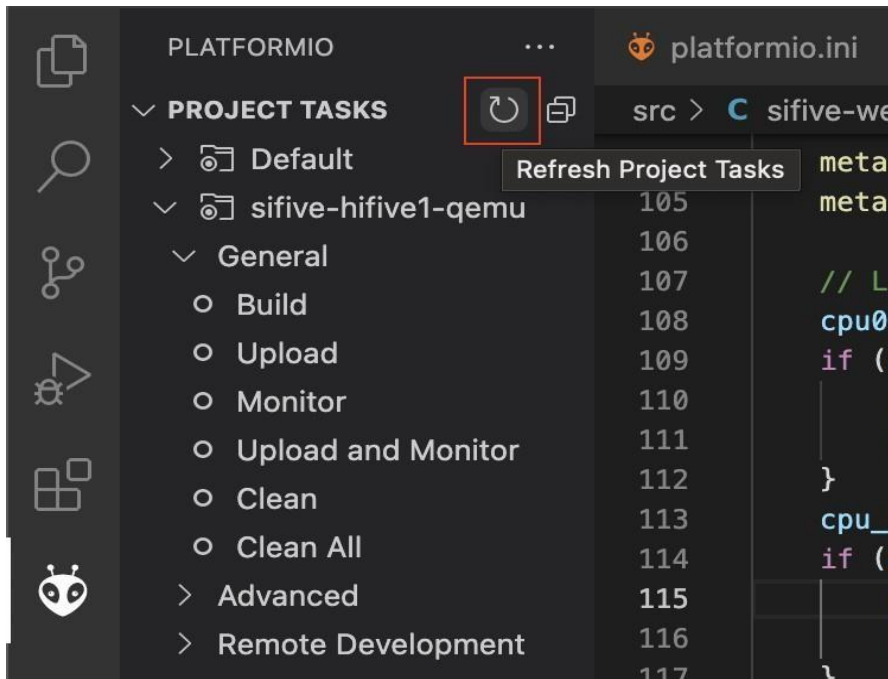
```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "PlatformIO",
      "task": "Monitor",
      "problemMatcher": [
        "$platformio"
      ],
      "presentation": {
        "panel": "new"
      }
    }
  ]
}
```

- 3.

Ver más opciones en la documentación oficial de VSCode.

Tareas personalizadas:

Declare objetivos personalizados y actualice el Explorador de



tareas:

Espacios de trabajo multiproyecto:

Puede trabajar con varias carpetas de proyectos en Visual Studio Code con espacios de trabajo multirraíz. Esto puede ser muy útil cuando está trabajando en varios proyectos relacionados al mismo tiempo. Obtenga más información en la documentación Espacios de trabajo multirraíz.

Monitoreo de puerto serie:

Puede personalizar Serial Port Monitor usando las opciones de Monitor en “platformio.ini”

(Archivo de configuración del proyecto):

- monitor_puerto.
- monitor_velocidad.
- monitor_paridad.
- monitor_filtros.

- monitor_rts.
- monitor_dtr.
- monitor_eol.
- monitor_sin procesar.
- monitor_echo.

Ejemplo:

[env:esp32dev] platform

= espressif32 framework

= arduino board = esp32dev

; Custom Serial Monitor port monitor_port

= /dev/ttyUSB1;

Custom Serial Monitor speed (baud rate) monitor_speed

= 115200

Depuración:

La depuración en VSCode funciona en combinación con la depuración. Debe tener una cuenta de PlatformIO para trabajar con ella. VSCode tiene una vista de actividad separada llamada "Depurar" (a la que se accede mediante el icono de error en la barra de herramientas de la izquierda). La depuración lo amplía con funciones e instrumentos de depuración más avanzados:

- Explorador de variables locales, globales y estáticas
- Puntos de ruptura condicionales
- Expresiones y puntos de observación
- Registros Genéricos
- Registros Periféricos
- Visor de memoria

Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montaña – Rodolfo Paz - Carolina Nis - Fernando Vexenat

- Desmontaje
- Soporte multihilo
- Un reinicio en caliente de una sesión de depuración activa. Hay dos configuraciones de depuración preconfiguradas: **Depuración de PIO:**

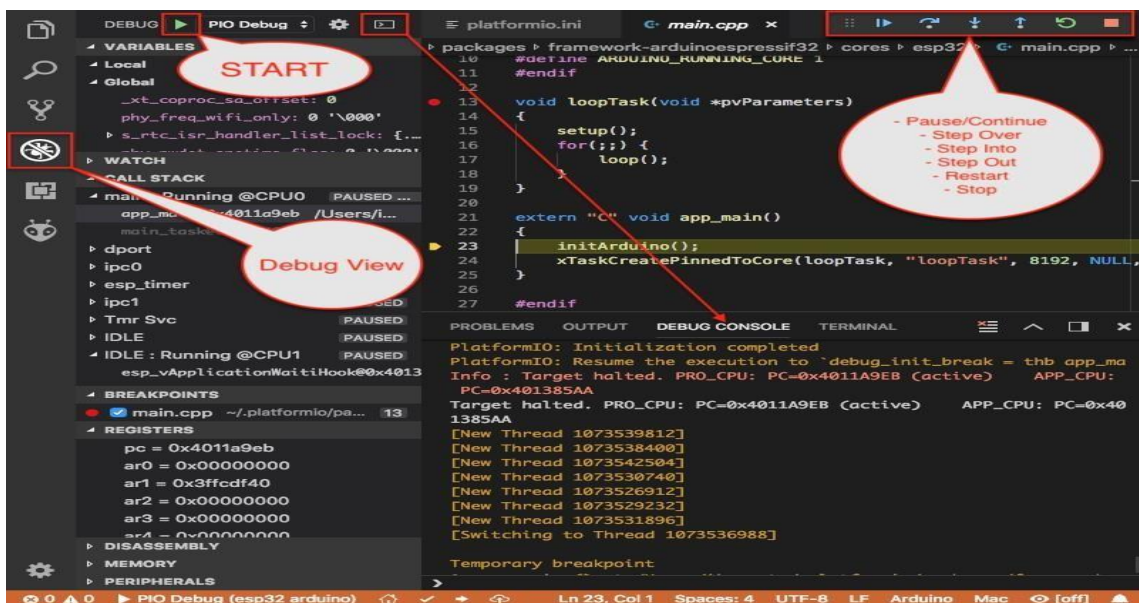
Configuración por defecto. PlatformIO ejecuta la tarea **previa a la depuración** y compila el proyecto mediante la configuración de depuración. Además, comprueba si hay cambios en el proyecto.

Depuración PIO (omitir Pre-Depuración):

PlatformIO se salta la etapa **previa a la depuración** y NO compila ni comprueba si hay cambios en el proyecto. Si realiza cambios en los archivos de origen del proyecto, no se reflejarán en las sesiones de depuración hasta que vuelva a cambiar a la configuración "PIO Debug" o ejecute manualmente la tarea "Pre-Debug".

Esta configuración es muy útil para una sesión de depuración rápida. Es súper rápido al omitir varios controles, lo que le permite controlar los cambios del proyecto manualmente.

Nota: Tenga en cuenta que la depuración utilizará el primer entorno de compilación declarado en "platformio.ini" (archivo de



configuración del proyecto) si no se especifica la opción `default_envs`.

Formato variable:

Actualmente, VSCode no proporciona una interfaz de usuario ni una API para cambiar el formato de la variable. Consulte el problema de VSCode n.º 28025 relacionado.

Una solución temporal es establecer la base numérica predeterminada en la que el depurador muestra la salida numérica en la consola de depuración. (La consola de depuración está visible durante las sesiones de depuración activas). Por ejemplo, para mostrar variables en formato hexadecimal, copie el siguiente código y péguelo en la "Consola de depuración": set output-radix 16.

Los valores posibles, enumerados en base decimal, son: 8, 10, 16.

Los puntos de vigilancia:

Lea primero GDB: Establecimiento de puntos de observación.

Actualmente, VSCode no proporciona una API para cambiar el formato de valor de los puntos de observación. Puede emitir manualmente expresiones de punto de observación para mostrar el valor como tipos de puntero específicos:

- \$pc, formato de entero decimal predeterminado
- *0x10012000, una dirección, formato de entero decimal predeterminado
- (void*)\$pc, registro \$pc, formato hexadecimal
- *(void**)0x10012000, una dirección, formato hexadecimal

Instalar comandos de shell.

Consulte Comandos de shell de instalación de PlatformIO Core.

Compatibilidad con servidor proxy:

Hay dos opciones para configurar un servidor proxy:

1. Declare las variables de entorno del sistema
HTTP_PROXYy (por ejemplo, etc.)
HTTPS_PROXYHTTP_PROXY=http://user:pass@10.10.1.1
0:3128/
2. Abra la configuración de VSCode y busque "Proxy".
Configure "Http: Proxy" y

deshabilite "Http: Proxy Strict SSL".

Atajos de teclado:

- ctrl+alt+b// cmd-shift-bConstruir ctrl-shift-bProyecto
- cmd-shift-d/ ctrl-shift-dProyecto de depuración
- ctrl+alt+uSubir firmware
- ctrl+alt+sMonitor de puerto serie abierto

Puede anular los enlaces de teclas existentes o agregar uno nuevo en VSCode. Consulte la documentación oficial de Key Bindings para Visual Studio Code .

Ajustes:

¿Cómo configurar los ajustes de VSCode? **platformio-
ide.activateOnlyOnPlatformIOProject:**

Si es verdadero, active la extensión solo cuando un proyecto basado en PlatformIO (que tiene un "platformio.ini" (archivo de configuración del proyecto)) está abierto en el espacio de trabajo. El valor predeterminado es. platformIO ide false **platformio-
ide.activateProjectOnTextEditorChange:**

Activar automáticamente el proyecto dependiendo de un editor de texto abierto activo. El valor predeterminado es false. **platformio-
ide.autoOpenPlatformIOIniFile:**

Abra automáticamente el archivo "platformio.ini" (Archivo de configuración del proyecto) de un proyecto cuando no haya otros editores abiertos. El valor predeterminado es true. **platformio-
ide.autoCloseSerialMonitor:**

Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montaña – Rodolfo Paz - Carolina Nis - Fernando Vexenat

Si es verdadero, cierre automáticamente el monitor del dispositivo pio antes de cargar/probar. El valor predeterminado es true.

platformio-ide.autoRebuildAutocompleteIndex:

Si es verdadero, reconstruirá automáticamente el índice de proyectos de C/C++ cuando se

cambie "platformio.ini" (archivo de configuración del proyecto) o cuando se instalen nuevas

bibliotecas. El valor predeterminado es true. **platformio-ide.buildTask:**

La tarea de compilación (etiqueta) que se inicia con el botón "Generar" en la barra de herramientas y enlaces de teclas de PlatformIO. El valor predeterminado es. PlatformIO: Build

Puede crear tareas personalizadas personalizadas y asignar una de ellas a platformioide.buildTask.

platformio-ide.autoPreloadEnvTasks:

Precargue automáticamente TODAS las tareas del entorno del proyecto. El valor predeterminado es false. **platformio-ide.customPATH:**

RUTA personalizada para el platformIOcomando. Pegue aquí el resultado del comando (Unix)

/ (Windows) escribiendo en la terminal de su sistema si prefiere usar una versión personalizada de PlatformIO Core (CLI). El valor predeterminado es, lo que significa que PlatformIO busca el comando en la ruta del sistema.echo \$PATHecho

%PATH%nullplatformio **platformio- ide.disableToolbar:**

Deshabilite la barra de herramientas de PlatformIO. El valor predeterminado es false.

platformio-ide.forceUploadAndMonitor:

Si es verdadero, el platformio-ide.uploadcomando Cargar () se cambia para usar la tarea "Cargar y monitorear". El valor predeterminado es false. **platformIO-ide.reopenSerialMonitorDelay:**

Configure el tiempo en milisegundos antes de volver a abrir Serial Port Monitor. El valor predeterminado es 0, lo que significa volver a abrir al instante. **platformIO- ide.useBuiltinPython:**

Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montaña – Rodolfo Paz - Carolina Nis - Fernando Vexenat

Utilice un intérprete de Python 3 portátil si está disponible. El valor predeterminado es true.

platformio-ide.useBuiltinPIOCore:

Si es verdadero, utilice PlatformIO Core (CLI) integrado. El valor predeterminado es true.

platformio-ide.useDevelopmentPIOCore:

Si es verdadero, use la versión de desarrollo de PlatformIO Core (CLI) . El valor predeterminado es false.

platformio-ide.disablePIOHomeStartup:

Deshabilite mostrar PlatformIO Home al inicio. El valor predeterminado es false. **platformio-**

ide.pioHomeServerHttpHost:

Host HTTP del servidor de PlatformIO Home. El valor predeterminado es 127.0.0.1, pero en el caso de entornos dockerizados 0.0.0.0. **platformio-ide.pioHomeServerHttpPort:**

Puerto HTTP del servidor principal de PlatformIO. El valor predeterminado 0 asigna automáticamente un puerto libre en el rango [8010..8100]). **platformio- ide.customPyPiIndexUrl:**

URL base personalizada del índice de paquetes de Python (predeterminado <https://pypi.org/simple>).

Problemas conocidos:

PackageManager no puede instalar la herramienta:

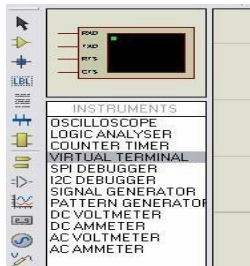
Este es un error conocido en el problema #61 de VSCode Terminal.

Una solución temporal es instalar paquetes usando una terminal del sistema (no la Terminal VSCode). Utilice la "Solución 3: Ejecutar desde la terminal" en Preguntas frecuentes > Administrador de paquetes > [Error 5] Acceso denegado. Luego, vuelva a usar la Terminal VSCode.

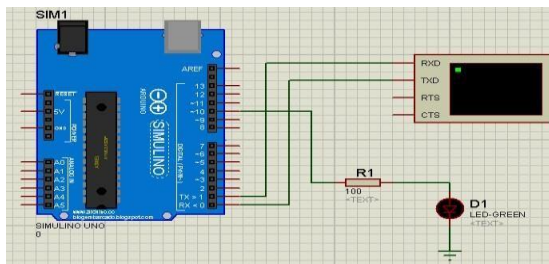
SIMULAR MONITOR SERIAL EN PROTEUS EN ARDUINO

Alumnos: Leonardo González - Juan Diego González Antoniazzi – Andres Montaña – Rodolfo Paz - Carolina Nis - Fernando Vexenat

Para simular el monitor serial en proteus debemos hacer uso del instrumento virtual terminal.



Lo incluimos en nuestro circuito Arduino muestra figura.



Notar que los TX y RX van a cruzados entre el Virtual Terminal y

```
//Programa LED_CONTROL

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

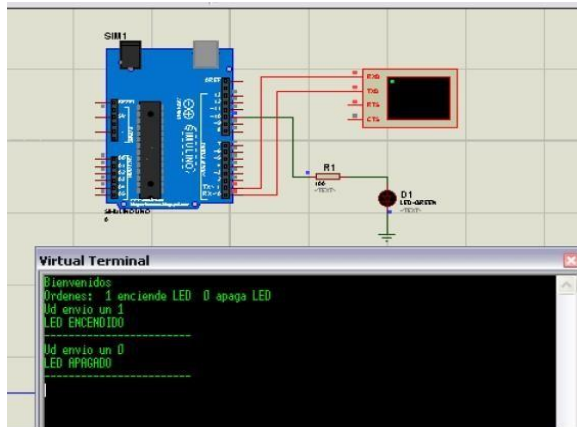
  Serial.println("Bienvenidos "); // Mensaje a Monitor Serial
  Serial.println("Ordenes: 1 enciende LED 0 apaga LED"); // Mensaje por Monitor Serial
  delay(100); // Los retardos son necesarios en la practica para mejorar desempe1o
}

void loop()
{
  if(Serial.available()) // Si la comunicaci3n 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
    }
  }
}
```

la placa Arduino.

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 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
}

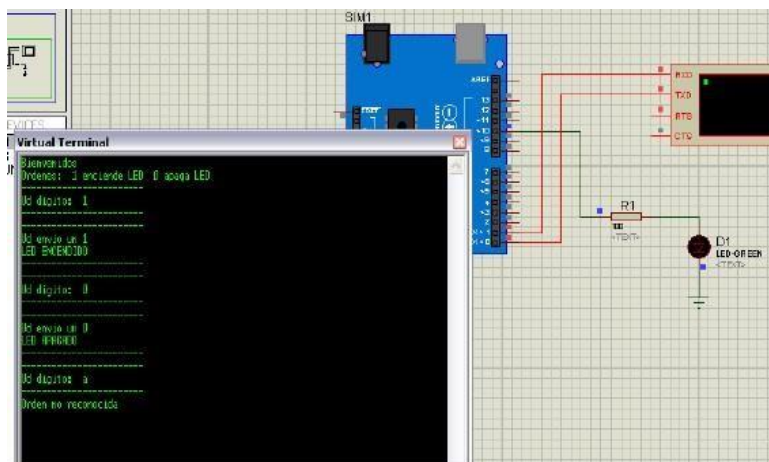
void loop()
{
  if(Serial.available()) // Si la comunicacion serial es utilizable, pregunta aqui.
  {
    char c=Serial.read(); // Se lee el monitor serial y esperando solo un caracter
    // se almacena en una variable tipo char que llamamos c

    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 sucede lo siguiente:



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 = "";

    /*
    * 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

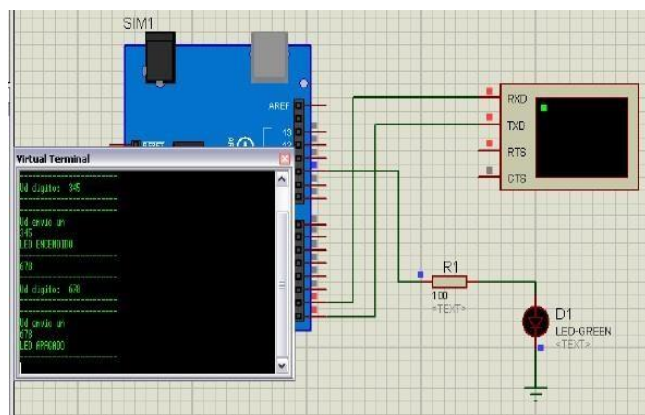
    if(num==345) //pregunta por el contenido de la variable num
    {
      digitalWrite(10,HIGH); // coloca en ALTO la salida digital PIN 10
      Serial.println("-----"); //Mensaje a Monitor Serial
      Serial.print("Ud envio un "); //Mensaje a Monitor Serial
      Serial.println(num);

      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.print("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 en proteus



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

RTS: Ready To Send (Listo para enviar)

Usada por el DTE (equipo de terminal de datos) para determinar la transmisión de datos del DCE (equipo de comunicación de datos). La transición a ON pone al DCE en modo de transmisión. La transición a OFF indica a CDE que complete la transmisión.

CTS: Clear To Send (Libre para envío)

Usada por el DCE para indicar si está listo para recibir datos del DTE. Cuando CTS, DSR, RTS y DTR están activadas (ON), el DCE está listo para recibir datos del DTE a través del canal de comunicaciones. Cuando sólo CTS está activada, el DCE sólo está listo para aceptar señales de marcación o de control. Cuando CTS está desactivada, el DTE no debería transferir datos a través de TXD.

DSR: Data Set Ready (Datos preparados)

Usada por el DCE para indicar si está listo para funcionar. Cuando DSR está activada, el DCE está conectado a la línea y listo para intercambiar más señales de control para iniciar la transferencia de datos.

DTR: Data Terminal Ready (Terminal de datos listo)

Usada para controlar la conmutación del DCE al canal de comunicaciones.

3. Que es una transmisión serie o UART? Y que significan las siguientes propiedades: Baud Rate, Data Bits, Parity, Stop Bits, Send XON/XOFF, terminalType.

¿Qué es UART?

UART (**universal asynchronous receiver / transmitter**, por sus siglas en inglés), define un protocolo o un conjunto de normas para el intercambio de datos en serie entre dos dispositivos. UART es sumamente simple y utiliza solo dos hilos entre el transmisor y el receptor para transmitir y recibir en ambas direcciones. Ambos extremos tienen una conexión a masa. La comunicación en UART puede ser **simplex** (los datos se envían en una sola dirección), **semidúplex** (cada extremo se comunica, pero solo uno al mismo tiempo), o **dúplex completo** (ambos extremos pueden transmitir simultáneamente). En UART, los datos se transmiten en forma de tramas.



¿Dónde se utiliza UART?

UART fue uno de los primeros protocolos en serie. Los puertos en serie, que en su día proliferaron a gran escala, se basan casi siempre en el protocolo UART, y los dispositivos que utilizan interfaces RS-232, módems externos, etc. son ejemplos típicos de la aplicación de UART. En los últimos años la popularidad de UART ha disminuido, y se ha ido sustituyendo por protocolos como SPI e I2C para la comunicación entre chips y componentes. En lugar de comunicarse por un puerto en serie, la mayoría de los ordenadores y periféricos modernos utilizan hoy tecnologías como Ethernet y USB. Sin embargo, UART se sigue utilizando para aplicaciones de baja velocidad y bajo rendimiento, puesto que es muy simple, económico y fácil de integrar.

Temporización y sincronización de protocolos UART

Una de las mayores ventajas de UART es que es asíncrono: el transmisor y el receptor no comparten la misma señal de reloj. Si bien esto simplifica en gran medida el protocolo, plantea determinados requisitos en el transmisor y el receptor. Puesto que no comparten un reloj, ambos extremos deben transmitir a la misma velocidad, previamente concertada, con el fin de mantener la misma temporización de los bits. Las velocidades en baudios más habituales en UART que se utilizan actualmente son 4800, 9600, 19,2 K, 57,6 K, y 115,2 K. Además de tener la misma velocidad en baudios, ambos extremos de una conexión UART deben utilizar también la misma estructura y parámetros de trama. La forma más sencilla de entender esto es observando una trama UART.

Formato de trama en UART

Como en la mayoría de los sistemas digitales, un nivel de tensión «alto» se utiliza

para indicar un «1» lógico, y un nivel de tensión «bajo» se emplea para indicar un «0» lógico. Dado que el protocolo UART no define tensiones específicas o rangos de tensión para estos niveles, a veces se denomina al nivel alto «marca» y al bajo «espacio». Obsérvese que en el estado de reposo (cuando no se transmiten datos) la línea se mantiene en el estado alto. Esto permite detectar con facilidad una línea o un transmisor averiado.

Bits de inicio y de parada

Puesto que UART es asíncrono, el transmisor necesita señalar que los bits de datos están llegando. Este se realiza con el bit de inicio. El bit de inicio es una transición del estado de reposo alto a un estado bajo, seguido inmediatamente de bits de carga útil (de datos). Una vez que finalizan los bits de datos, el bit de parada indica el fin de la carga útil. El bit de parada es o bien una transición de retorno al estado alto o de reposo, o bien la permanencia en el estado alto por un tiempo de bit adicional. Se puede configurar un segundo bit de parada (opcional), normalmente para dar al receptor tiempo para prepararse para la siguiente trama, si bien no es una práctica muy común.

Bits de datos (Data Bits)

Los bits de datos son la carga útil o los bits «útiles», y llegan inmediatamente después del bit de inicio. Puede haber de 5 a 9 bits de carga útil, si bien lo más habitual son 7 u 8 bits. Estos bits de datos se transmiten generalmente con el bit menos significativo primero.

Por ejemplo:

Si deseamos enviar la letra mayúscula «S» en ASCII de 7 bits, la secuencia de bits es 1 0 1 0 0 1 1. Primero invertimos el orden de los bits para colocarlos en el orden del bit menos significativo, es decir, 1 1 0 0 1 0 1, antes de enviarlos. Una vez que se han enviado los últimos bits de datos, el bit de parada se utiliza para finalizar la trama y la línea retrocede al estado de reposo.

- «S» en ASCII con 7 bits (0x52) = 1 0 1 0 0 1 1
- Orden de bit menos significativo = 1 1 0 0 1 0 1



Bits de datos



Bits de inicio y de parada

Bit de paridad (Parity)

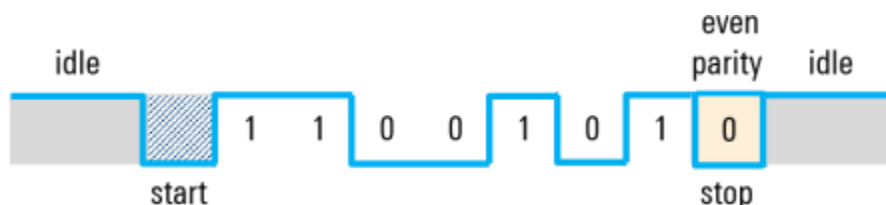
Una trama UART también puede contener un bit de paridad opcional que se utiliza para la detección de errores. Este bit se inserta entre los bits de datos y el bit de parada. El valor del bit de paridad depende del tipo de paridad utilizado (par o impar):

- En la **paridad par**, este bit se ajusta de tal modo que el número total de unos en la trama será par.
- En la **paridad impar**, este bit se ajusta de tal modo que el número total de unos en la trama será impar.

Ejemplo:

La letra «S» mayúscula (1 0 1 0 0 1 1) contiene en total tres ceros y cuatro unos. Si se utiliza la paridad par, el bit de paridad es cero, puesto que ya existe un número par de unos. Si se utiliza la paridad impar, entonces el bit de paridad deberá ser uno para que la trama tenga un número impar de unos.

El bit de paridad solo puede detectar un único bit invertido. Si hay más de un bit invertido, no es posible detectarlo con fiabilidad utilizando un único bit de paridad.



Ejemplo de bit de paridad

TASA DE BAUDIOS (BAUD RATE)

El baudio es una unidad de medida utilizada en telecomunicaciones, que representa el número de símbolos por segundo en un medio de transmisión digital. Cada símbolo puede codificar 1 o más bits dependiendo del esquema de modulación, un bit siempre representa dos estados, por lo tanto baudios por segundo no siempre es equivalente a bits por segundo, los símbolos son las

unidades de información estas se representan en bits, de manera que la tasa de bits será igual a la tasa de baudios solo cuando halla 1 bit por símbolo.

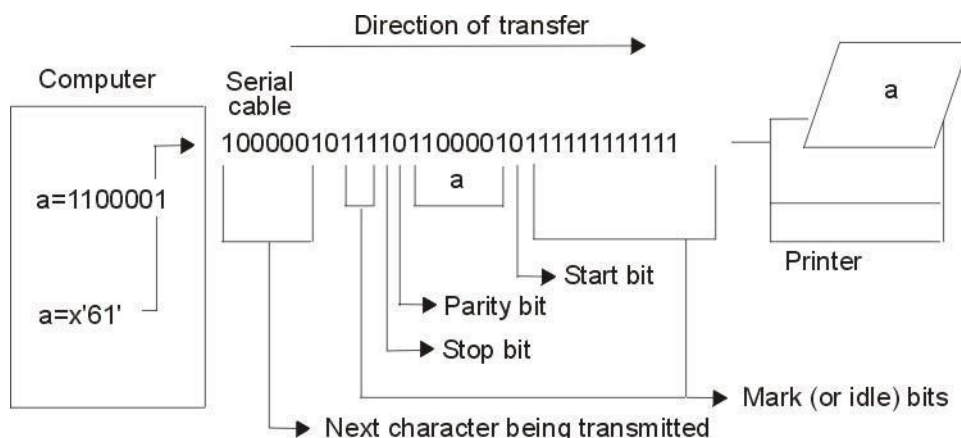
STOP BITS

Los bits de inicio y parada se utilizan en la comunicación asíncrona como un medio de temporización o sincronización de los caracteres de datos que se transmiten.

Sin el uso de estos bits, los sistemas de envío y recepción no sabrán dónde termina un carácter y comienza otro.

Otro bit utilizado para separar los caracteres de datos durante la transmisión es el bit RS de marca (o inactivo). Este bit, un 1 binario, se transmite cuando la línea de comunicación está inactiva y no se envían ni reciben caracteres.

Cuando el sistema recibe un bit de inicio (0 binario), se entiende que un bit de carácter de número fijo (determinado por el parámetro de **bits por carácter**), e incluso un bit de paridad (determinado por el parámetro de **paridad**), sigue al bit de inicio. Luego, el sistema recibe un bit de parada (1 binario). En el siguiente ejemplo, el bit de **paridad** está presente y los **bits por carácter**



son 7.

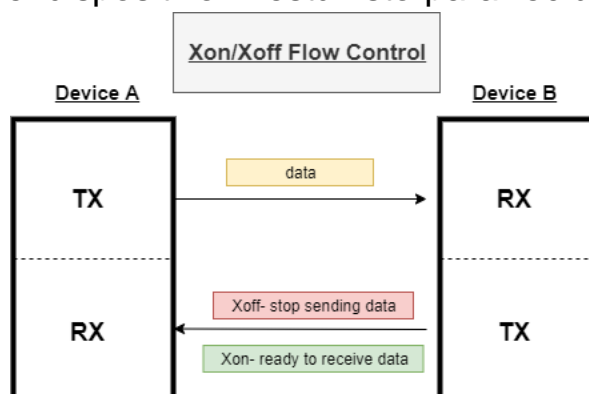
Bits de inicio, parada y marca

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* llegará

muy rápidamente a 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](#) 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.

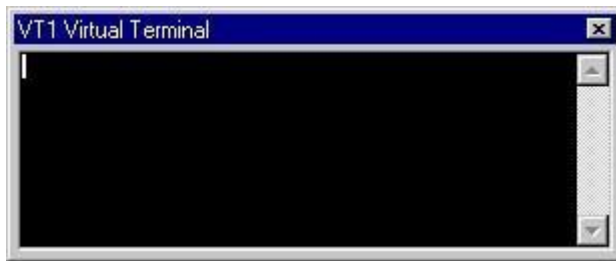
TERMINAL TYPE

USB se ha convertido en el protocolo de comunicación en serie preferido entre productos digitales como computadoras y dispositivos periféricos, ya que puede transmitir datos a través de cables más largos y ofrece velocidades de transmisión más altas. Sin embargo, UART todavía se usa hoy en día para ciertas aplicaciones y, a menudo, se ve en dispositivos más antiguos.

Para cerrar la brecha entre los dispositivos más antiguos y los más nuevos, los desarrolladores han mantenido viables las rutas de transmisión heredadas entre los sistemas USB y UART. Un convertidor de USB a UART es un circuito integrado que se utiliza para enviar o recibir datos en serie desde un puerto USB a datos en serie que pueden recibirse o enviarse mediante una interfaz UART. Este es un dispositivo pequeño que se conecta a su puerto USB y tiene al menos salidas de tierra, Rx y Tx. Actúa como un puerto serie para su computadora y la computadora envía datos a este puerto donde el módulo los convierte en señales UART.

4. Que es el eco, en relación al tipeo y una pantalla. Y porque no tengo eco en el terminal virtual de proteus?

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

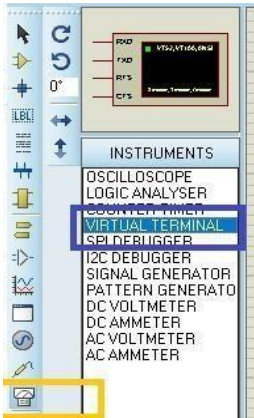


Sus principales características son:

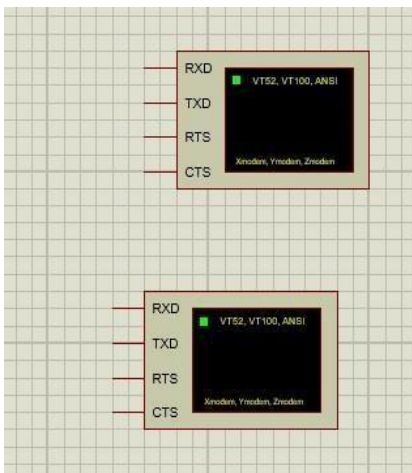
- Completo soporte bi-direccional. Los datos recibidos se visualizan como caracteres ASCII y las teclas pulsadas se transmiten como datos serie ASCII.
- Interface simple de transmisión de datos serie mediante utilización de dos cables: RXD para recibir datos y TXD para transmitirlos.
- Protocolo de sincronización hardware simple mediante dos hilos: RTS para "listo para enviar" y CTS para "limpio par enviar".
- Protocolo de sincronización software XON/XOFF adicional al protocolo de sincronización hardware.
- Velocidades de 300 a 57,6 kbaudios.
- Datos de 7 y 8 bits.
- Paridad par, impar y nula.
- 0,1 y 2 bits de parada.

Terminal virtual en Proteus Simulation

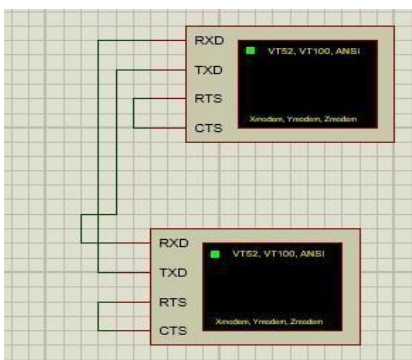
- En primer lugar, abra su software Proteus en su computadora, luego muévase a la pestaña que se muestra en la figura a continuación y presione para ver una opción diferente donde existe la terminal virtual.



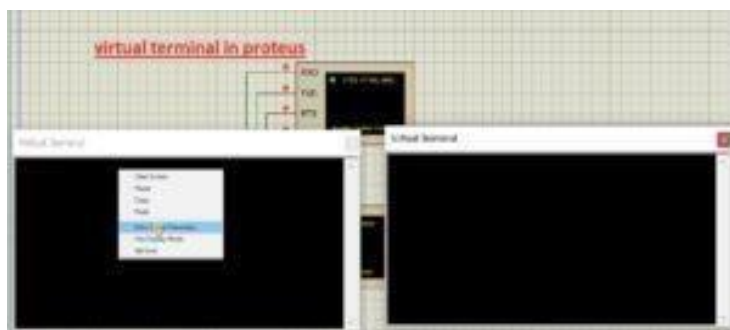
- Coloque la caja de terminal virtual en la pantalla de Proteus como se muestra a continuación



- Ahora haga la conexión entre dos cajas de terminales virtuales como se muestra aquí



- Ahora presione el botón ejecutar, se abrirán dos ventanas, presione a la derecha en una ventana para seleccionar los caracteres de tipo eco. Lo que escribirá se verá en otras pantallas.

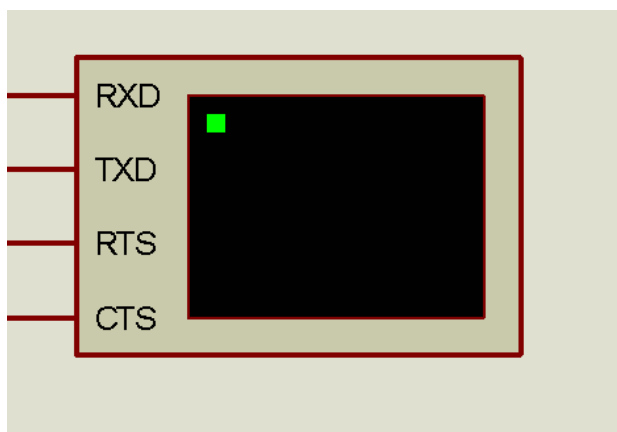


Eso es todo sobre el terminal virtual en Proteus Simulation. He explicado todos los pasos que se utilizan para que el terminal virtual transmita datos a través del puerto serie y reciba datos a través del puerto serie.

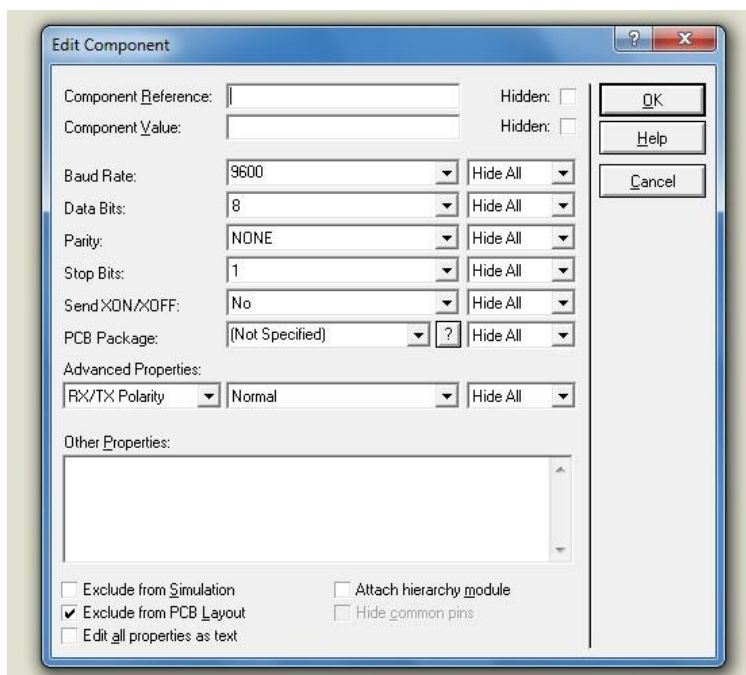
5. Explique las propiedades avanzadas del terminal virtual.

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.

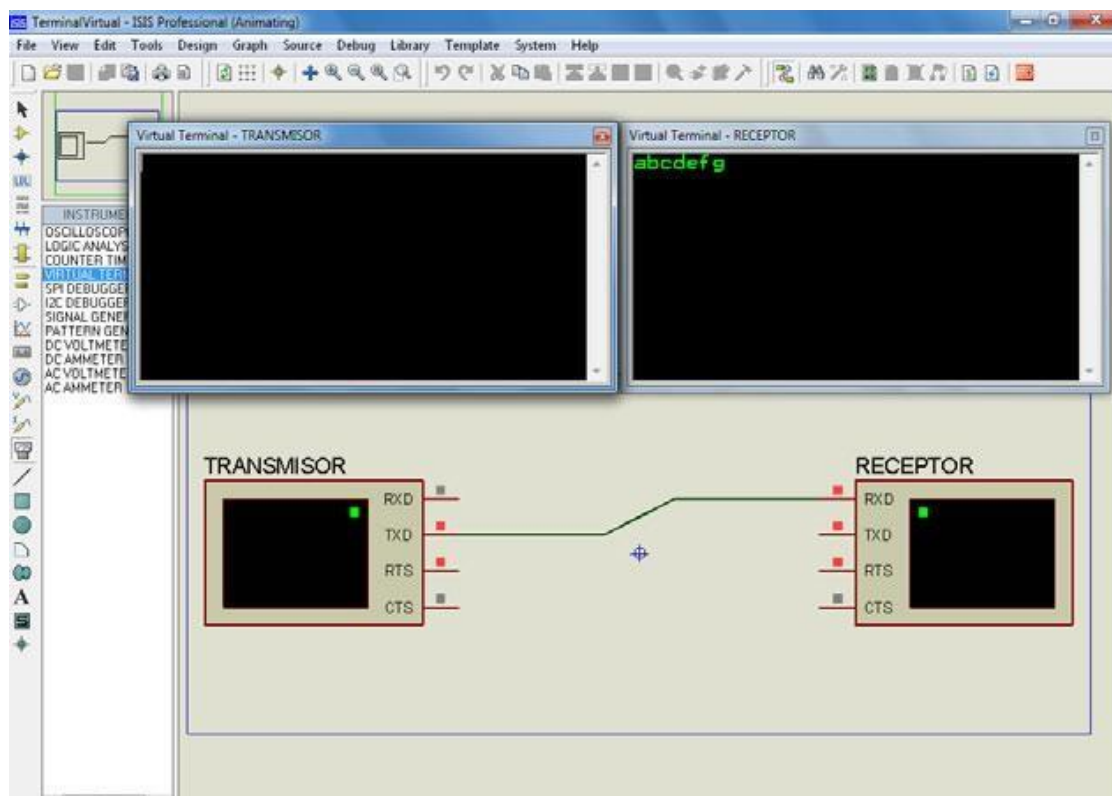


Las terminales de este instrumento son: RXD para recibir datos; TXD para enviar datos en formato ASCII desde el teclado de la PC, es decir, la terminal virtual enviará los datos ASCII que ingresemos hacia donde conectemos esta terminal; RTS (Ready to send) y CTS (Clear to send). En las propiedades de 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 57600baudios.
- **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. 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 enRTS/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.

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 CLICK 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).

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.

6. Como funciona COMPIM y para que sirve? Que es un virtualizador de puertos, de ejemplos de los mas utilizados.

El programa Proteus es una aplicación de simulación muy popular empleada por ingenieros y desarrolladores. El software Proteus simula circuitos eléctricos y se utiliza para diseño asistido por ordenador y modelado de microprocesadores, microcontroladores y otros dispositivos programables.

Para verificar esta funcionalidad, utilizaremos Proteus. Si agregamos un modelo de puerto serie y conectamos el RXD y el TXD, los datos transmitidos deben devolverse inmediatamente al ordenador. Esto verifica la capacidad de activar la comunicación serie en Proteus.

Idealmente, se crea un puerto serie virtual en Proteus que simula la interacción con una interfaz física. Luego, puede ejecutar la simulación del dispositivo y usar Serial Port Terminal como programa host para probar la conexión. Desafortunadamente, el puerto virtual no se crea en el simulador Proteus sin la ayuda de un software adicional.

La funcionalidad que permite que el modelo de puerto COM funcione en Proteus se conoce como COMPIM.

¿Qué es COMPIM en Proteus?

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.

Dos formas de trabajar con Proteus

Existen dos métodos que pueden usarse para verificar la funcionalidad del “programa host” <-> “puerto COM” <-> “modelo de dispositivo en el sistema Proteus..

- Configure el puerto virtual de Proteus en un puerto físico y el programa host en el otro. Conéctelos usando un cable serial.
- También puede usar dos ordenadores, uno de las cuales ejecuta la simulación del dispositivo mientras que el otro ejecuta el programa host y conectarlos a través de sus puertos COM.

Proteus tiene ventajas sobre otras herramientas como **VMLab** y **Atmel Studio** porque proporciona una simulación más rápida de puertos serie externos. Utilizando Proteus también puede trabajar con controladores comerciales.

Sin embargo, hay un problema cuando usamos un ordenador portátil moderno u otro ordenador que no tiene un puerto serie.



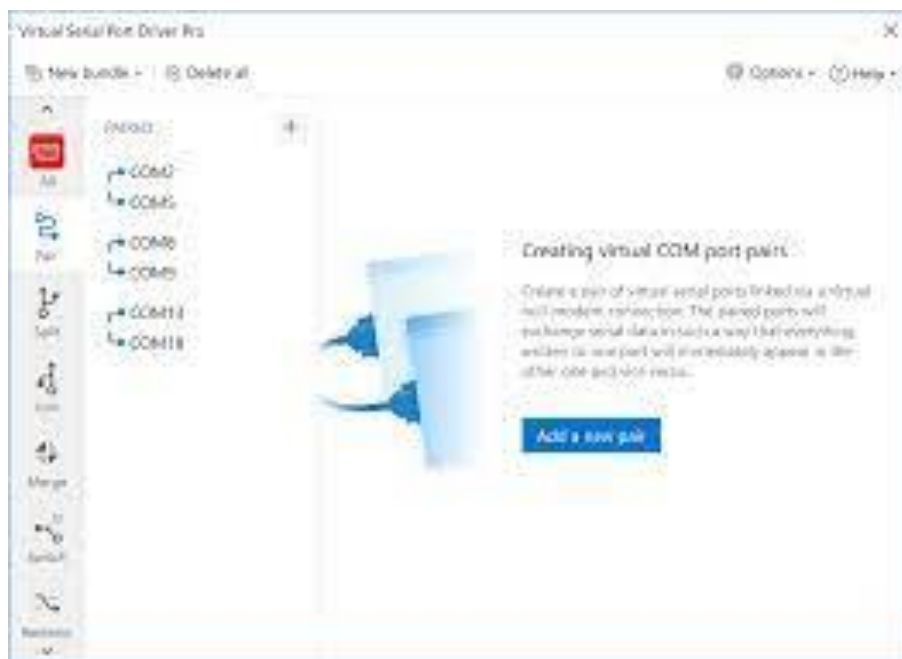
Instrucciones paso a paso para crear puertos virtuales para Proteus

Solucionar este problema implica aprovechar el poder de Virtual Serial Port Driver. Este software profesional de Electronic Team le permite crear fácilmente pares de puertos serie virtuales conectados.

Solo tiene que seguir estos sencillos pasos:

1. Descargue Virtual Serial Port Driver.

2. Inicie la aplicación y seleccione los números de puerto que utilizará. Haga clic en el botón "Agregar par" y su sistema verá inmediatamente dos puertos serie conectados.
3. Cree un par de puertos llamados COM1 y COM3.



4. Enlace el modelo COMPIM de Proteus a COM1 y use Serial Port Terminal para conectarse a COM3.
5. Transmitir datos en la línea. Si se devuelven como esperado, ha resuelto el problema de la falta de un puerto serie.

Con estos pasos, los puertos serie virtuales se pueden usar con el simulador Proteus incluso en ordenadores que no tienen puertos COM físicos.

Las 7 mejores aplicaciones de Puerto Serial Virtual:

- **Virtual Serial Port Driver [Electronic Team]:** es un paquete de software desarrollado por Electronic Team que funciona como emulador de puerto COM virtual en máquinas Windows. El software le permite crear puertos COM virtuales, que usted puede conectar en pares mediante un cable de módem nulo virtual.

Cada puerto serie virtual que cree se comunicará con sus aplicaciones en serie como si fueran puertos físicos reales.

- **Virtual COM Port Driver PRO - funcionalidad avanzada:** es un eficiente software todo-en-uno basado en la funcionalidad de Virtual Serial Port Driver. El programa ayuda a crear paquetes de puertos serie virtuales y personalizar los parámetros del puerto, que lo convierten en la solución ideal para una gran variedad de escenarios de uso.
- **Virtual Serial Port Kit [FabulaTech]:** es una herramienta de software que le permite emular puertos serie. Estos puertos virtuales RS232 creados se pueden conectar con un cable de módem nulo virtual.
- **Virtual Serial Ports [HHD Software]:** La solución ofrecida por HDD - Free Virtual Serial Ports se centra en la emulación de puertos personalizados Plug and Play y I/O interconectados por 16550 UART.

Virtual Serial Port Emulator [Eterlogic]: Está diseñado para ingenieros de software que necesitan desarrollar aplicaciones serie y les permite crear, probar y depurar aplicaciones y dispositivos que emplean interfaces serial. Algunas de las ventajas de este programa y de la virtualización de puertos es que el mismo dispositivo puede ser abierto por múltiples aplicaciones y usted puede crear pares de puertos serie virtuales.

- **COM Port Data Emulator [AGG Software]:** es un programa que permite a los usuarios emular una entidad, como un dispositivo Ethernet o un puerto COM, que transmite los datos como flujo serie. El programa toma el flujo de datos y lo convierte en paquetes de datos utilizando protocolos serial como RS232, TCP/IP o UDP. Luego reenvía los paquetes a través de la red.

- **Emulador del módem nulo [com0com]:** es un driver de puerto serial virtual de código abierto modo kernel que permite emular las interfaces de comunicación serie.