Librería para Arduino Segainvex_SCPI V1.0

La librería para Arduino, Segainvex_SCPI está escrita en C. Implementa un sistema de comunicación entre un PC y Arduino mediante comandos. De modo que el PC puede enviar comandos a Arduino y en respuesta a estos Arduino ejecutará una función por cada comando recibido. Los comandos pueden ir seguidos de parámetros. Así, además de conseguir que Arduino ejecute una función al recibir un comando, la propia función que ejecuta, puede leer dichos parámetros.

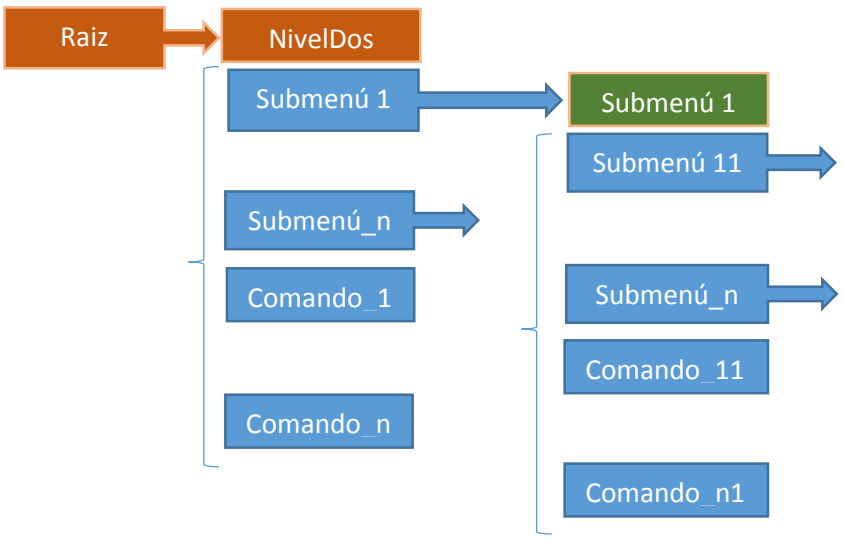
Para que Arduino pueda utilizar las funcionalidades de la librería, además de la declaración de la librería con el *include* correspondiente, en el código de Arduino debe definirse un menú que contenga los comandos más un array de cadenas con la definición de errores numerados, que se podrían cometer en el proceso de comunicación y ejecución de las funciones.

El menú se articula como un conjunto de arrays de estructuras enlazadas y jerarquizadas en niveles. El nivel más alto de la lista es el nivel "Raiz" que apunta a "NivelDos" que es el nivel principal. Es obligatorios definir ambos niveles. "Raiz" lo único que hace es apuntar a " NivelDos", que contiene estructuras, que o bien; son comandos, o apuntan a otra estructura semejante a NivelDos, que sería un submenú (menú de nivel inferior), que como "NivelDos", contiene estructuras comando o estructuras apuntando a submenús de nivel inferior.

Las funciones que se ejecutan al recibir comandos, están declaradas en el fichero de cabecera de la librería, pero no definidas, ya que se definen en el código de Arduino, para que hagan lo que el usuario desee. El prototipo de estas funciones es:

void fsNumeroFuncion(void); siendo "NumeroFuncion" un entero. Por ejemplo:

void fs1(void);



La definición del menú y lista de errores, se escribe antes de la función "setup()". Para hacerlo, la librería exporta dos tipos; *tipoNivel* y *tipoCodigoError*. Además exporta 4 macros para facilitar la tarea de crear el menú. Veamos un ejemplo, pero antes, un detalle importante: primero se definen los submenús de nivel inferior, luego "NivelDos" y por último el "Raiz".

```
#include <Arduino.h>
#include <segainvex_scpi.h> // funciones y variables de Segainvex_SCPI
/*
Submenú al que llamamos por ejemplo "SUBMENU1". Contiene dos comandos, a los que
decidimos llamar,COMANDO11 con nombre abreviado C11 y COMANDO12 con nombre
abreviado C12. El primero hará que se ejecute la función "void fs3(void)" y el
segundo la función "void fs4(void)" que definiremos como cualquier otra función.
*/
 tipoNivel SUBMENU1[] = //Array de estructuras tipo Nivel

{
       SCPI_COMANDO(COMANDO11,C11,fs3)//Comando que ejecuta la función fs3()
       SCPI_COMANDO(COMANDO12,C12,fs4)//Comando que ejecuta la función fs4()
       // TO DO Añadir aquí comandos o submenús
};

//Ahora podemos definir “NivelDos”
/*
 "NivelDos" Nivel obligatorio en todas las aplicaciones
*/
 tipoNivel NivelDos[] = //Array de estructuras tipo Nivel
{
       //Submenú con comandos declarado más arriba
       SCPI_SUBMENU(SUBMENU1,SM1 )
       //Comandos definidos por el usuario
       SCPI_COMANDO(COMANDO1,C1,fs1) //Comando que ejecuta la función fs1()
       SCPI_COMANDO(COMANDO2,C2,fs2) //Comando que ejecuta la función fs2()
       //Comandos que ejecutan funciones definidas en la librería Segainvex_SCPI
       SCPI_COMANDO(ERROR,ERR,fs243)// Envía el ultimo error
       SCPI_COMANDO(*IDN,*IDN,fs240)// Identifica el instrumento
       SCPI_COMANDO(*OPC,*OPC,fs248)// Devuelve un 1 al PC
       SCPI_COMANDO(*CLS,*CLS,fs255)// Borra la pila de errores
       // TO DO Añadir aquí comandos o submenús
};
```

/*
Vemos que “NivelDos” contiene la declaración del submenú SUBMENU1 definido anteriormente. Además contiene dos comandos a los que hemos decidido llamar COMANDO1 y COMANDO2 con nombre abreviado C1 y C2 que ejecutan las funciones que definiremos más adelante “void fs1(void)” y “void fs2(void)”; También hay cuatro comandos más ERROR, *IDN, OPC, y *CLS que ejecutan las funciones ·”fs240”,”fs243”,”fs248” y ”fs255” que ya están definidas en la librería y que se explicarán más adelante.
*/
//Por último declaramos el nivel raíz:

```
SCPI_NIVEL_RAIZ// Macro que hace la declaración obligatoria del nivel Raiz

//Solo nos falta definir la lista de errores:
/*
CodigoError está declarado en scpi.h, por lo que es global. Es obligatorio
definirlo aquí. También es obligatorio incluir los primeros 6 errores y
denominarlos como se ve a continuación.
Cada error que quiera registrar el usuario ha de incluirlo desde el 7 en
adelante.
*/
tipoCodigoError CodigoError=
{
  // Errores del sistema SCPI 0...6
  " ",                                    // ERROR N. 0
  "1 Caracter no valido",                 // ERROR N. 1
  "2 Comando desconocido",                // ERROR N. 2
```

```
  "3 Cadena demasiado larga",            // ERROR N. 3
  "4 Parametro inexistente",             // ERROR N. 4
  "5 Formato de parametro no valido",    // ERROR N. 5
  "6 Parametro fuera de rango",          // ERROR N. 6
  // Errores personalizados por el usuario
  "7 El dato no esta listo",             // ERROR N. 7
};

//Ahora el código habitual de Arduino:

void setup()
{

//Macro de Segainvex_SCPI que rellena una cadena con el nombre del sistema
       NOMBRE_DEL_SISTEMA_64B(Prueba de Segainvex_SCPI para Arduino V1.0)
       // Abre el puerto serie
       Serial.begin(57600);
}
void loop()
{
  // Si recibe algo por el puerto serie lo procesa con SEGAINVEX_SCPI
  if (Serial.available()){scpi();}
/*
TO DO Poner aquí el código de usuario
*/
}// loop()

//Ahora solo nos queda definir las funciones de nuestro sistema:

void fs1(void)
{
 Serial.println
 ("Se ha recibido el COMANDO1 y se ha ejecutado la funcion fs1");
}
void fs2(void)
{
 Serial.println
 ("Se ha recibido el COMANDO2 y se ha ejecutado la funcion fs2");
}
void fs3(void)
{
Serial.println
 ("Se ha recibido el COMANDO11 y se ha ejecutado la funcion fs3");
}
void fs4(void)
{
Serial.println
 ("Se ha recibido el COMANDO12 y se ha ejecutado la funcion fs4");
}
```

Prueba de lo programado hasta ahora

Para probar el código escrito hasta ahora podemos compilarlo y flashear nuestra placa Arduino favorita. Para enviar comandos podemos utilizar la aplicación para puerto serie “Terminal –by Br@y++”.

https://sites.google.com/site/terminalbpp/Terminal20141030.zip?attredirects=0&d=1

Ejecutamos Terminal y como vemos en la figura1, Arduino se ha instalado en el COM25. Hemos abierto el puerto COM25 con la configuración que se muestra. También

hay que activar el checkbox “+CR” (delante del botón “->Send”) para que al enviar el comando, se incluya al final de la cadena un carácter CR ’\r’.

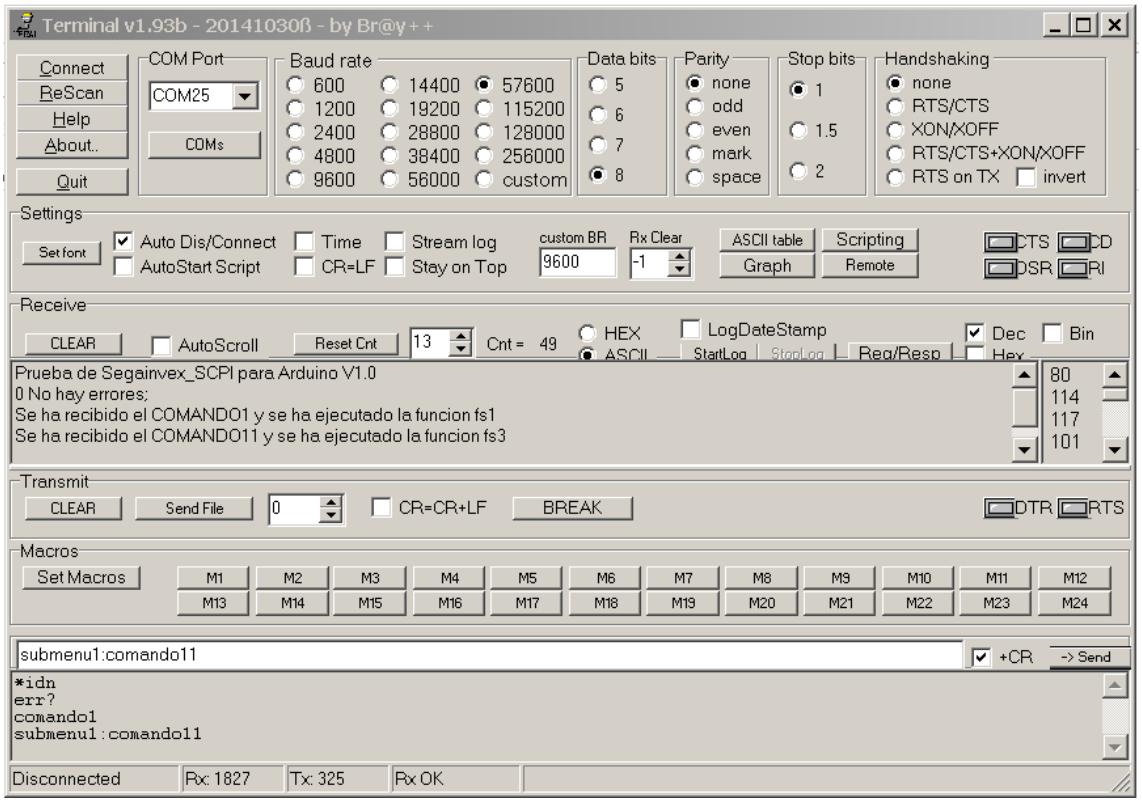


Figura 1

Enviar comandos a Arduino

Fijémonos en la figura1. En el *textbox* de cadenas enviadas por el puerto serie (abajo) vemos que se han enviado algunos comandos, en el *textbox* de arriba vemos las respuestas de Arduino a cada comando. Los comandos se pueden enviar con mayúsculas o minúsculas indistintamente. El último comando “COMANDO11” está en el submenú SUBMENU1 dentro de la estructura de comandos que hemos creado, por eso hay que escribirlo precedido por el nombre del submenú en el que se encuentra separado por ‘:’. “SUBMENU1:COMANDO11”. También podríamos haber escrito los nombres abreviados “SM1:C1”. El tercer comando de la lista “COMANDO1” se escribe directamente, porque está en el nivel principal, “NivelDos”. También podría haberse escrito abreviadamente “C1”.

Comandos definidos en librería Segainvex_SCPI

Hay 4 comandos definidos en la librería. Estos son:

“ERROR” abreviadamente “ERR”. Ejecuta la función fs243(); que envía al PC el último error almacenado en la pila de errores.

“*CLS”. Ejecuta la función fs255(); que limpia la pila de errores.

“OPC”. Ejecuta la función fs248(); que envía al PC la cadena “1”.

“*IDN”. Ejecuta la función fs240();que envía al PC una cadena con el nombre del sistema.

Cambiar el valor de variables del programa de Arduino con Segainvex_SCPI

Después de un comando y separado por espacios, el PC puede enviar parámetros a Arduino. Hemos de ser capaces de recuperar estos parámetros y asignarlos a variables del tipo apropiado. Para facilitar esta tarea, la librería Segainvex_SCPI implementa 4 funciones. Si el comando se envía seguido de un espacio y a continuación un parámetro numérico, estas funciones leen el parámetro, testean su rango y si pasa el test, asignan el parámetro a la variable correspondiente (si no pasa el test, se anota un error 6). Pero si el comando se envía seguido inmediatamente de '?', las funciones envían al PC el valor actual de la variable. Las funciones devuelven un entero: 1 si el valor de la variable se cambió, 0 si no se cambió y 2 si se devolvió al PC el valor actual de la variable.Los protipos de las funciones son:

```
int cambia_variable_int_del_sistema(int *,int,int);
```
Para cambiar el valor de una variable entera. Los argumentos de entrada son; la dirección del entero que queremos cambiar, su valor máximo y su valor mínimo.

```
int cambia_variable_int_discreta_del_sistema(int *,int*,int);
```
Para cambiar el valor de una variable entera que solo puede tener un conjunto discreto de valores. Los argumentos de entrada son; la dirección del entero que queremos cambiar, la dirección de un array de enteros con el conjunto de valores que puede tener la variable y por fin otro entero con el tamaño del array.

```
int cambia_variable_bool_del_sistema(bool *);
```
Para cambiar el valor de una variable booleana. El argumentos de entrada es la dirección del booleano que queremos cambiar.

```
int cambia_variable_double_del_sistema(double *,double,double);
```
Para cambiar el valor de una variable real doble. Los argumentos de entrada son; la dirección de la variable, su valor máximo y su valor mínimo.

Para ver como se utilizan, modificaremos las dos últimas funciones de nuestro código como sigue:

```
/*************************************************************
      Comando SUBMENU1:COMANDO11 ó SM1:C11
      Ejecuta la función void fs3(void);
      Cambia el valor de la variable tipo double "Variable1"
*************************************************************/
void fs3(void)
{
      static double Variable1=1.1;
      int Resultado;
      Resultado=cambia_variable_double_del_sistema(&Variable1,10.0,0.0);
      switch (Resultado)
      {
            case 0:
                  Serial.println("No Se cambio el valor de la Variable1");
                  errorscpi(7);
            break;
            case 1:
                  Serial.println("Se cambio el valor de la Variable1");
            break;
            case 2:
                  Serial.println("Se envio al PC el valor de la Variable1");
            break;
      }
}
/*************************************************************
      Comando SUBMENU1:COMANDO11 ó SM1:C12
      Ejecuta la función void fs4(void);
```

```
       Cambia el valor de la variable tipo int "Variable2"
*************************************************************/
void fs4(void)
{
       int Resultado;
       static int Variable2=100;
       int ValoresVariable2[]={1,10,100};
       Resultado=cambia_variable_int_discreta_del_sistema(&Variable2,
       ValoresVariable2,sizeof(ValoresVariable2));
       switch (Resultado)
       {
              case 0:
                     Serial.println("No Se cambio el valor de la Variable2");
              errorscpi(7);//Aquí vemos como se anota un error
              break;
              case 1:
                     Serial.println("Se cambio el valor de la Variable2");
              break;
              case 2:
                     Serial.println("Se envio al PC el valor de la Variable2");
              break;
       }
}
```

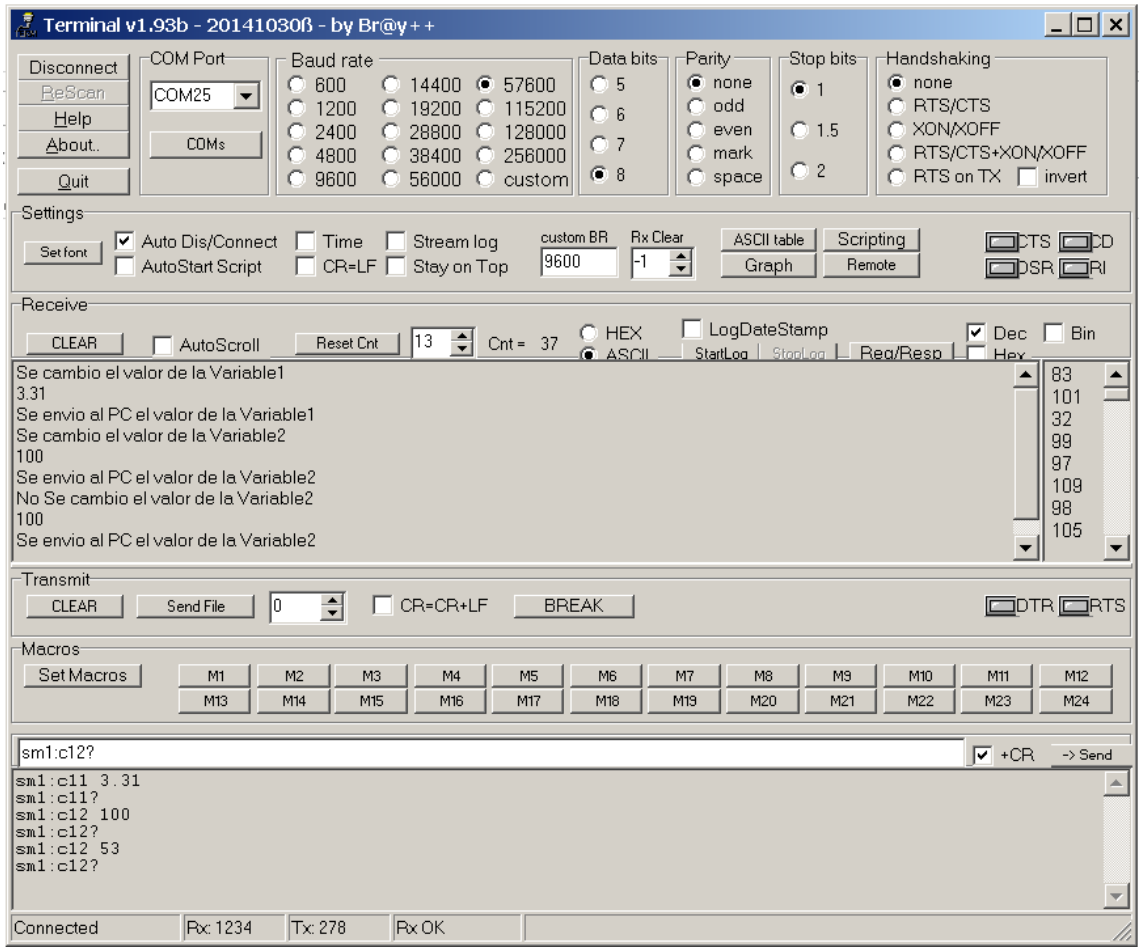


Figura2

Compilamos, flasheamos Arduino y ejecutamos “Terminal”. En la figura 2 podemos ver que se han enviado algunos comandos y las respuestas de Arduino.

Si queremos escribir nosotros mismos el código para leer los parámetros del comando, no tenemos más que editar el fichero Segainvex_SCPI.cpp de la librería y copiar el código de las funciones. Podemos ampliarlo para leer en una sola función más de un parámetro.

Terminadores utilizados en la comunicación con Segainvex_SCPI

Los comandos Segainvex_SCPI que se envían a Arduino desde el PC son cadenas de caracteres cuyo terminador ha de ser el carácter retorno de carro (ASCII 13,'\r', CR), ya que Segainvex_SCPI espera este carácter para decidir que ha recibido una cadena completa.

Si hacemos nuestra propia aplicación de PC para Comunicar con Arduino, hay que saber que cuando Arduino envía una cadena al PC con la función "Serial.println()" termina la cadena con los caracteres retorno de carro y nueva línea (ASCII 13 10, '\r\n', CR LF.

Buffer de recepción de Arduino

Segainvex_SCPI lee cadenas de caracteres del puerto serie en un array tipo char, que se dimensiona con la constante BUFFCOM_SIZE. Como el buffer de recepción del puerto serie de Arduino es de 64 bytes, BUFFCOM_SIZE, como mucho puede valer 64 bytes. Por defecto vale 32.

Gestión de errores

Si en el proceso de comunicación a cualquier nivel, se detecta un error, por ejemplo de sintaxis o se envía una cadena demasiado larga etc. Segainvex_SCPI anota el error en una pila LIFO. Cuando recibe el comando "ERROR" devuelve al PC la cadena correspondiente del array CodigoError, del último error anotado y lo saca de la pila LIFO.

El usuario puede describir los errores que se pueden cometer al ejecutarse su aplicación de Arduino en CodigoError. Por ejemplo; deseamos que si el PC pida a Arduino que le envíe una variable que no está lista, Segainvex_SCPI anote un error. Para eso describimos el error en CodigoError con el siguiente número disponible; en nuestro caso, el número 7.

```
tipoCodigoError CodigoError=
{
                    .
                    .
      "6 Parametro fuera de rango",     // ERROR N. 6
      // Errores personalizados por el usuario
      "7 El dato no esta listo",        // ERROR N. 7
};
```

Cuando se produzca el error, para que se anote, utilizamos la función Segainvex_SCPI errorscpi() así:

```
errorscpi(7);
```

Si ahora el PC envía el comando "ERR?" a Arduino, SegainvexSCPI enviará al PC la cadena: "7 El dato no está listo".

El tamaño de la pila de errores por defecto es 3, pero se puede redefinir con la constante MAXERR.

Constantes exportadas por la librería Segainvex_SCPI (redefinibles)

BUFFCOM_SIZE = 32 Longitud del buffer de lectura de Segainvex_SCPI.
LONG_SCPI = 32 Longitud máxima del comando (sin contar parámetros).
MAXERR = 3   Profundidad de la pila de errores.

Funciones exportadas por Segainvex_SCPI

void errorscpi(int)
void scpi(void)
int cambia_variable_int_del_sistema(int *,int,int)
int cambia_variable_int_discreta_del_sistema(int *,int*,int)
int cambia_variable_bool_del_sistema(bool *)
int cambia_variable_double_del_sistema(double *,double,double)
De “void fs0(void)” a “void fs100(void)” y de “void fs240()” a “void fs(255)”.

Variables exportadas por Segainvex_SCPI

tipoNivel Raiz[], array de la estructura raiz de comandos.
tipoCodigoError CodigoError, puntero al array de cadenas de errores.
char *FinComando, puntero al final del comando para leer parámetros.
char IdentificacionDelSistema[] cadena donde el usuario debe escribir el nombre.
de su sistema. La cadena se envía al PC cuando este envíe a Arduino el comando.
*IDN

Macros exportadas por Segainvex_SCPI

SCPI_SUBMENU(X,Y) Para definir submenús.
SCPI_COMANDO(X,Y,Z) Para definir comandos.
SCPI_NIVEL_RAIZ Para definir el nivel “Raiz”.
NOMBRE_DEL_SISTEMA_64B(X) Para poner en el array "IdentificacionDelSistema" el
nombre del sistema que implementa Arduino.