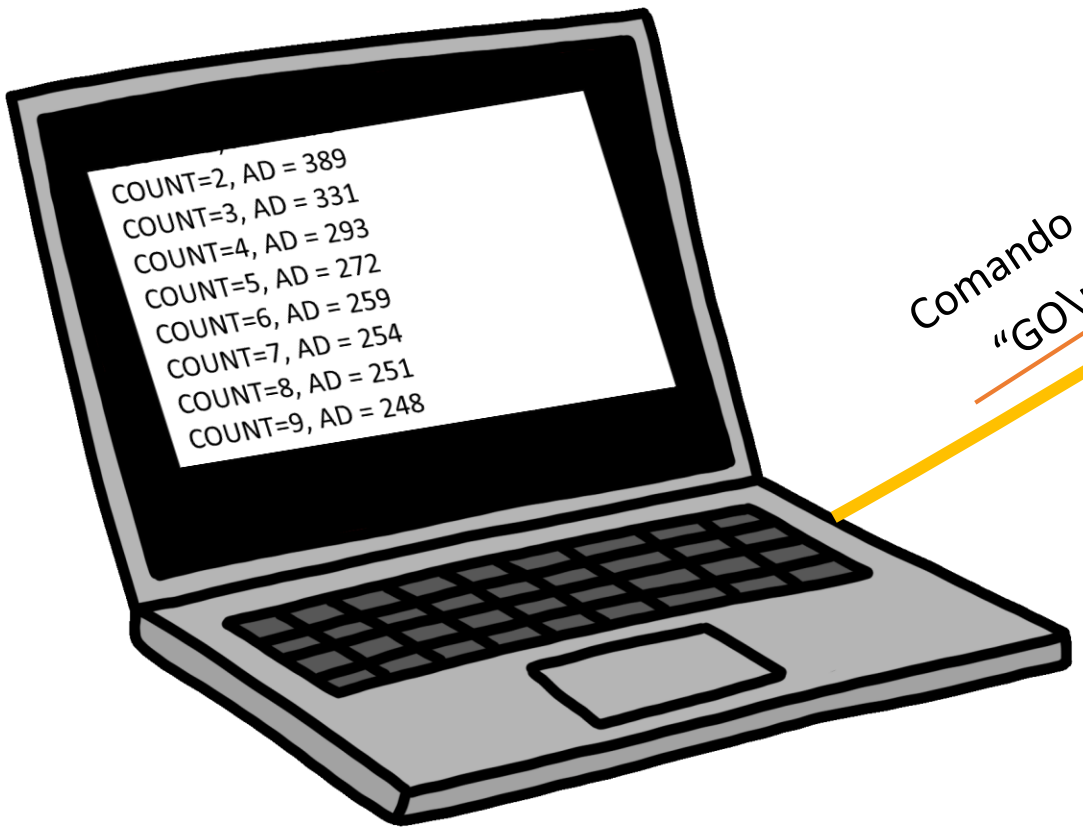
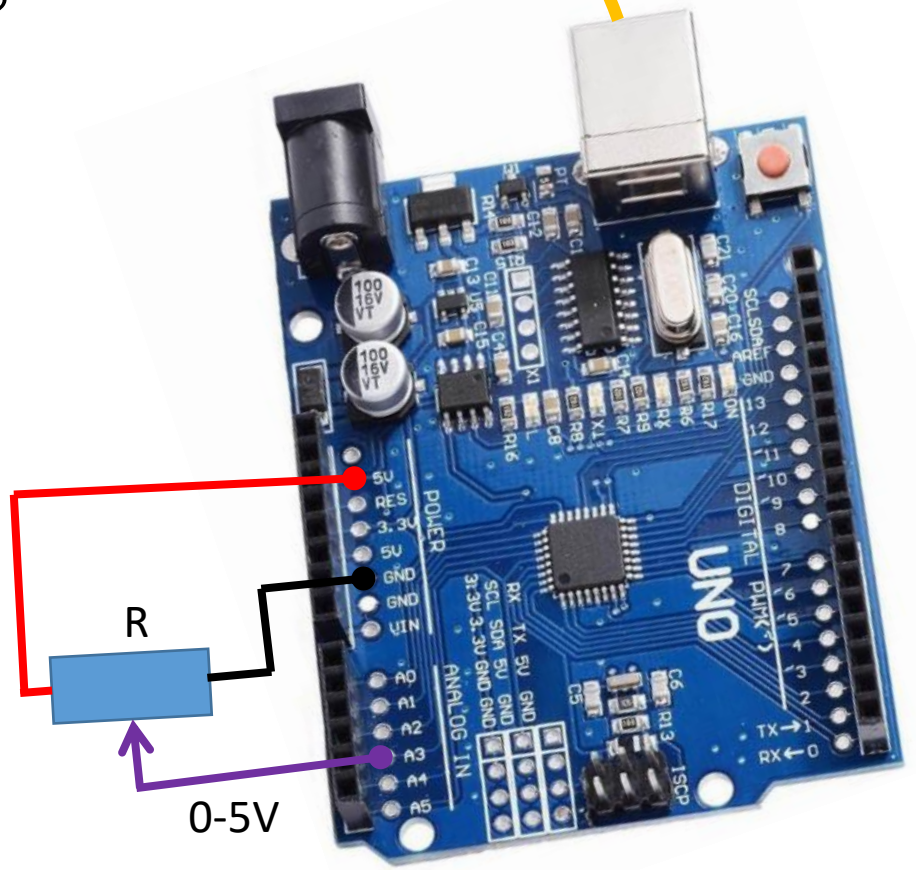
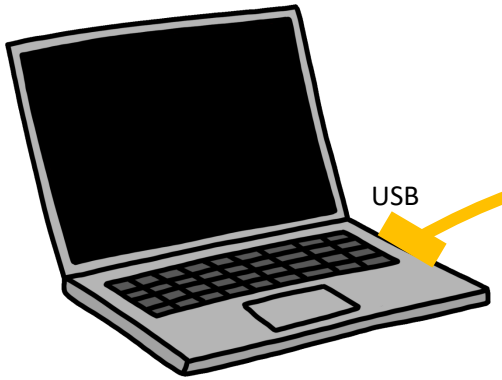
EJEMPLO COMUNICACIÓN SERIE ENTRE PC y ARDUINO Uno



Comando
"GO\n"

Respuestas: "AD=valor\n"
"END\n"

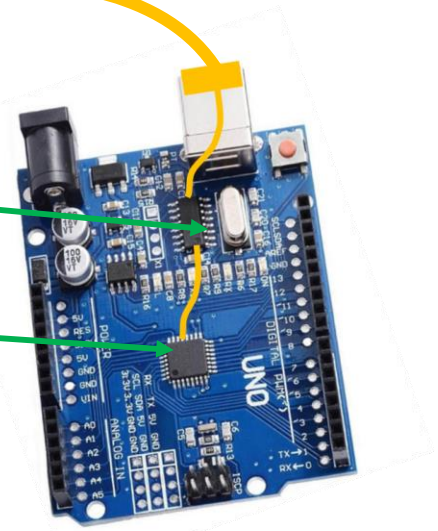




USB (comm + Power 5V/500mA)

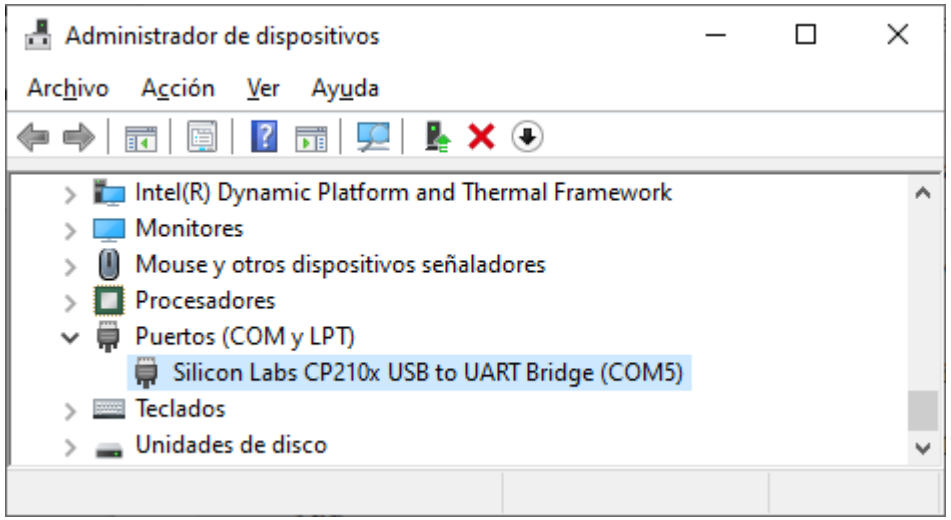
USB←→ Serial
(CH340)

Micro-controlador
(Atmega328p)



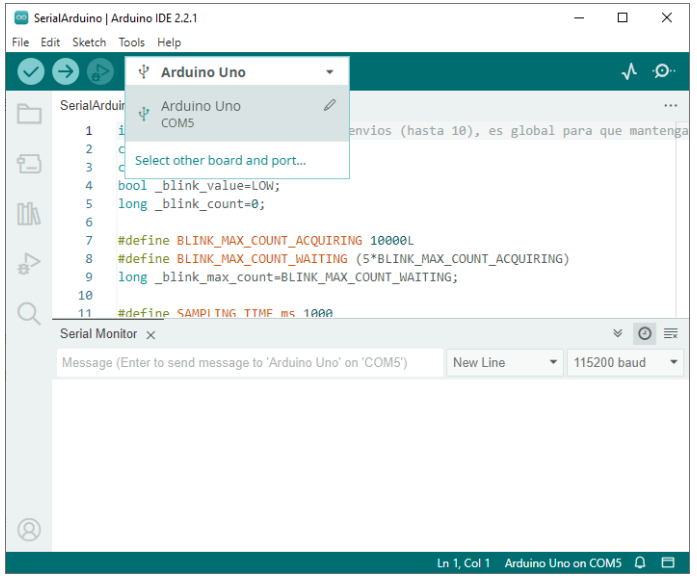
Lado PC:

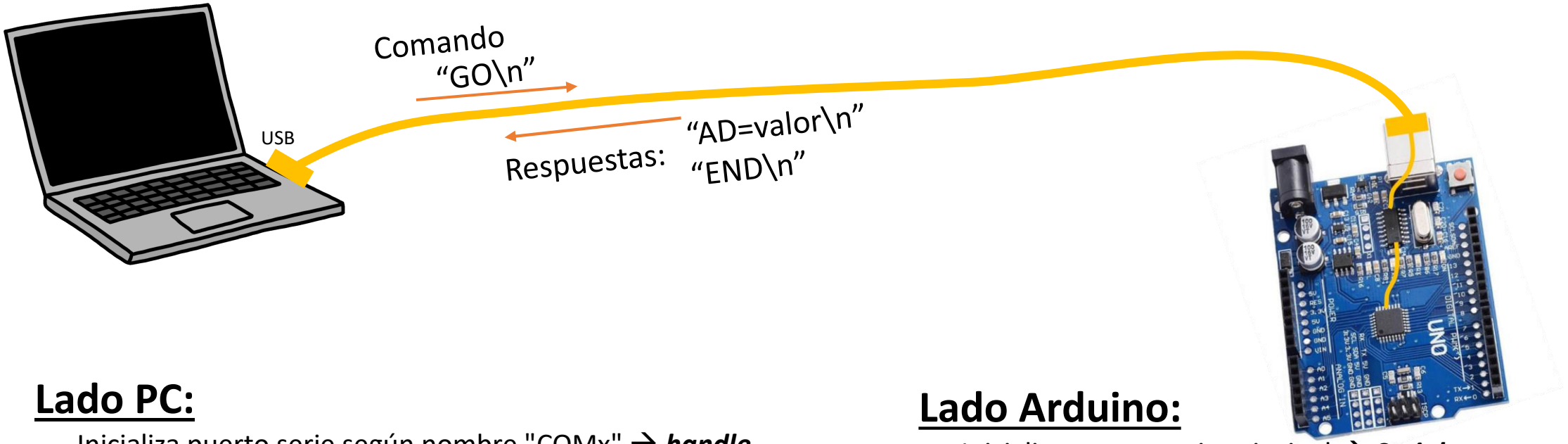
- Programa en C con Qt-Creator
- Puerto serie "COMx", según asigne el S.O. (Plug&Play)



Lado Arduino:

- Programa en C/C++ con Arduino-IDE
- Puerto serie sirve para carga del programa en Flash y para comunicación con Serial Monitor o programa aparte





Lado PC:

- Inicializa puerto serie según nombre "COMx" → ***handle***
  - Modo r/w
  - 115200 baud, 8 bit, n parity, 1 bit stop
- Escribe "GO\n" en dispositivo ***handle***
- Realiza bucle 10 veces:
  - Repite mientras carácter recibido no es '\n':
    - Espera a recibir carácter de ***handle***
    - Añade carácter a string
  - String completo: procesa y obtiene valor
  - Escribe valor y contador en consola
- Cierra ***handle***

Lado Arduino:

- Inicializa puerto serie principal → ***Serial***
  - Modo r/w
  - 115200 baud, 8 bit, n parity, 1 bit stop
- Inicializa entrada analógica AD3 y salida digital LED
- Realiza bucle indefinido:
  - Espera a recibir carácter
  - Añade carácter a string
  - Si carácter es '\n' → si string es "GO" → inicia cuenta
  - Si cuenta iniciada
    - Lee canal AD3 → valor
    - Compone texto para enviar
    - Envía "AD=valor\n"
  - Si cuenta >= 10 → envía "END\n" → cuenta = -1
  - Parpadea LED

PROGRAMA PRINCIPAL PC (Windows o Linux)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BAUD_RATE       115200      // Velocidad de comunicacion

#ifdef __WIN32__
#include "SerialPortFunctionsWin.h"
#define COMM_PORT   "COM5"      // Cambiar por el puerto serie adecuado
                                //(ver en administrador de dispositivos)
#elif __linux__
#include "SerialPortFunctionsLinux.h"
#define COMM_PORT   "/dev/ttyS0"      // Cambiar por el puerto serie adecuado
                                      // (/dev/ttyUSB0, /dev/ttyACM0)
#endif

int main()
{
    char snd[40],recv[40];      // Buffers de envio y recepcion
    DeviceHandle handle;        // Identificador de dispositivo
    int n_bytes;                // Num bytes enviados o recibidos

    // Inicializa dispositivo
#ifdef __WIN32__
    handle=InitSerialPortWin(COMM_PORT,BAUD_RATE);
#elif __linux__
    handle=InitSerialPortLinux(COMM_PORT,BAUD_RATE);
#endif
    // TODO: chequear handle!= DEVICE_HANDLE_IS_INVALID (falta de espacio)
    // Enviar orden GO (4 bytes)
    strcpy(snd,"\nGO\n");
#ifdef __WIN32__
    n_bytes=SendSerialPortWin(handle,snd,4);
#elif __linux__
    n_bytes=SendSerialPortLinux(handle,snd,4);
#endif

    if (n_bytes!=4)
    {
        printf("Error en envio\n");
#ifdef __WIN32__
        CloseSerialPortWin(handle);
#elif __linux__
        CloseSerialPortLinux(handle);
#endif
    }
    int terminado=0;        // Indica finalizar
    int cuenta=0;           // Num de conversiones AD recibidas
    while (! terminado)
    {
        // Recibir datos (uno a uno, almacenar en recv hasta leer el '\n')
        char* pt=recv;      // Puntero para recibir caracter a caracter
#ifdef __WIN32__
        while (n_bytes=RecvSerialPortWin(handle,pt,1) , n_bytes==1) // Leer 1 byte
#elif __linux__
        while (n_bytes=RecvSerialPortLinux(handle,pt,1)==1 , n_bytes==1) // Leer byte
#endif
        {
            if (*pt=='\n' || *pt=='\r')  // Se ha recibido fin de linea
            {
                *pt=0;      // Se cambia por el fin de cadena ...
                break;      // ... y se termina bucle while
            }
            else            // Caracter distinto de '\n':
                pt++;        // Incrementar puntero
        }

        if (n_bytes==1)
        {
            if (strncmp(recv,"AD=",3)==0)  // La cadena comienza por "AD="
            {
                int ad_value=atoi(recv+3);      // Obtener valor
                cuenta++;
                printf("COUNT=%d, AD = %d\n",cuenta,ad_value);
            }
            if (strncmp(recv,"END",3)==0)    // La cadena comienza por "END"
            {
                printf("END received. Closing\n");
                terminado=1;                    // Indicar final
            }
        }
    }  // Fin de bucle while (cuenta ha llegado a 10)

    // Finalizar conexion con el dispositivo (puerto serie)
#ifdef __WIN32__
    CloseSerialPortWin(handle);
#elif __linux__
    CloseSerialPortLinux(handle);
#endif

    return 0;
}
```

FUNCIONES PC (Windows)

SerialPortFunctionsWin.h

```
#ifndef SERIALPORTFUNCTIONSWIN_H
#define SERIALPORTFUNCTIONSWIN_H

#ifdef __WIN32__
// Include y define especificos para manejo de puerto serie en Windows
#include <windows.h>
#include <fcntl.h>
typedef HANDLE DeviceHandle;
#define DEEVICE_HANDLE_IS_INVALID INVALID_HANDLE_VALUE
DeviceHandle InitSerialPortWin(const char* commPort,int baud_rate);
int SendSerialPortWin(DeviceHandle h,const void* ptr,int snd_bytes);
int RecvSerialPortWin(DeviceHandle h,void* ptr,int rcv_bytes);
void CloseSerialPortWin(DeviceHandle h);
#endif // __WIN32__

#endif // SERIALPORTFUNCTIONSWIN_H
```

SerialPortFunctionsWin.c

```
#include "SerialPortFunctionsWin.h"

#ifdef __WIN32__

#include <stdio.h>

DeviceHandle InitSerialPortWin(const char *commPort,int baud_rate)
{
    DeviceHandle winHandle;  // Identificador de dispositivo Windows

    DCB dcb;                 // Estructura para configuracion de puerto serie
    int ok;

    // Crear conexion con el dispositivo (puerto serie)
    winHandle=CreateFileA(commPort,GENERIC_READ | GENERIC_WRITE, 0, NULL,
                          OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (winHandle == INVALID_HANDLE_VALUE)
    {
        printf ("CreateFile failed with error %d.\n", (int) GetLastError());
        return INVALID_HANDLE_VALUE;
    }
 // Poner parametros de comunicacion serie: 115200,8,n,1
    memset(&dcb,0,sizeof(DCB));     // Todos los valores a 0
    ok = GetCommState(winHandle, &dcb);     // Obtener parametros actuales
    if (!ok)
    {
         printf ("GetCommState failed with error %d.\n", (int) GetLastError());
         return INVALID_HANDLE_VALUE;
    }
    dcb.DCBlength = sizeof(DCB);
    dcb.BaudRate = baud_rate;
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = ONESTOPBIT;

    ok = SetCommState(winHandle, &dcb);  // Establecer nuevos parámetros comu
    if (!ok)
    {
         printf ("SetCommState failed with error %d.\n", (int) GetLastError());
         return INVALID_HANDLE_VALUE;
    }
    return winHandle;
}

int SendSerialPortWin(DeviceHandle h,const void *ptr, int snd_bytes)
{
    int  ok,n_bytes;
    ok=WriteFile(h,ptr,snd_bytes,(DWORD*) &n_bytes,NULL);
    if (ok)
        return n_bytes;
    return -1;
}

int RecvSerialPortWin(DeviceHandle h, void *ptr, int rcv_bytes)
{
    int  ok,n_bytes;
    ok=ReadFile(h,ptr,rcv_bytes,(DWORD*) &n_bytes,NULL);
    if (ok)
        return n_bytes;
    return -1;
}

void CloseSerialPortWin(DeviceHandle h)
{
    CloseHandle(h);
}

#endif
```

FUNCIONES PC (Linux)

SerialPortFunctionsLinux.h

```
#ifndef SERIALPORTFUNCTIONSLINUX_H
#define SERIALPORTFUNCTIONSLINUX_H

#ifdef __linux__
// Include y define especificos para manejo de puerto serie en Linux
#include <fcntl.h>
typedef int DeviceHandle;
#define DEEVICE_HANDLE_IS_INVALID       -1
DeviceHandle InitSerialPortLinux(const char* commPort,int baud_rate);
int SendSerialPortLinux(DeviceHandle h,const void* ptr,int snd_bytes);
int RecvSerialPortLinux(DeviceHandle h,void* ptr,int rcv_bytes);
void CloseSerialPortLinux(DeviceHandle h);
#endif // __linux__


#endif // SERIALPORTFUNCTIONSLINUX_H
```

SerialPortFunctionsLinux.c

```
#include "SerialPortFunctionsLinux.h"

#ifdef __linux__
#include <stdio.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <unistd.h>

DeviceHandle InitSerialPortLinux(const char *commPort,int baud_rate)
{
    DeviceHandle linuxHandle;           // Identificador de dispositivo Linux
    struct termios  newtio;

    //Comprobar baud_rates y asignar codigo
    int baud_rates[]={ 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200,
230400, 460800, 500000, 576000, 921600, 1000000, 1152000, 1500000, 2000000,
2500000, 3000000, 3500000, 4000000,-1};
    int
baud_codes[]={B1200,B2400,B4800,B9600,B19200,B38400,B57600,B115200,B230400,B46
0800,B500000,B576000,B921600,B1000000,B1152000,B1500000,B2000000,B2500000,B300
0000,B3500000,B4000000,B0};
    int i;
    for (i=0;baud_rates[i]>0;i++)
    {
        if (baud_rates[i]==baud_rate)
            break;
    }
    if (baud_rates[i]<0)
    {
        printf ("Baud rate %d not available\n",baud_rate);
        return -1;
    }

    // Abrir el puerto serie y ajustar parametros
    linuxHandle = open(commPort, O_RDWR | O_NOCTTY);
    if (linuxHandle < 0)
    {
        printf ("open failed with error %d.\n", (int) errno);
        return (-1);
    }

    fcntl(linuxHandle, F_SETOWN, getpid());     // Obtener posesión del puerto
    tcgetattr(linuxHandle, &newtio);            // Obtener parámetros actuales

    newtio.c_cflag=baud_codes[i];
    newtio.c_cflag |= CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_cflag &= ~CSTOPB;

    tcflush(linuxHandle, TCIFLUSH);
    tcsetattr(linuxHandle, TCSANOW, &newtio);   // Modificar parametros
    return linuxHandle;
}

int SendSerialPortLinux(DeviceHandle h,const void *ptr, int snd_bytes)
{
    return write(h,ptr,snd_bytes);
}

int RecvSerialPortLinux(DeviceHandle h, void *ptr, int rcv_bytes)
{
    return read(h,ptr,rcv_bytes);
}

void CloseSerialPortLinux(DeviceHandle h)
{
    close(h);
}

#endif
```

CODIGO ARDUINO UNO

```
// Variables globales para que mantengan su valor entre llamadas a loop()
int _count=-1;    // Cuenta de envios de valores AD(hasta 10)
char _recv[40];   // Buffer de recepción serie
char* _ptr_recv;  // Puntero al carácter del buffer de recepción

// Valores para hacer parpadeo del LED status del Arduino
#define BLINK_MAX_COUNT_ACQUIRING 10000L
#define BLINK_MAX_COUNT_WAITING (5*BLINK_MAX_COUNT_ACQUIRING)
bool _blink_value=LOW;
long _blink_count=0;
long _blink_max_count=BLINK_MAX_COUNT_WAITING;

// Valores para hacer temporización conersiones AD
#define SAMPLING_TIME_ms 1000
unsigned long _last_ad_read_time=0;


// Inicializacion
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  // LED status en modo OUTPUT
  pinMode(A3,INPUT);  // A3 en modo input
  Serial.begin(115200);   // Puerto serie=115200,8,n,1
  ptr_recv= recv;            // Inicializa puntero recepción
}

void loop() {
  CheckBlink();   // Parpadeo LED status

  if (_count<0)   // Esperar orden GO
  {
     if (! Serial.available())
        return;
     *_ptr_recv=Serial.read();   // Lee 1 byte
     if (*_ptr_recv!='\r' && *_ptr_recv!='\n')  // Si no es fin de linea
     {
         _ptr_recv++;    // Incrementa puntero recepcion
         if (_ptr_recv>=_recv+40)  // Comprueba dentro de rango buffer
            _ptr_recv=_recv+39;
         return;   // Nada mas que hacer
     }

// Solo pasa por aquí si se ha recibido fin de línea
     *_ptr_recv=0;  // Pone carácter nulo al final del buffer
     _ptr_recv=_recv;  // Reinicia puntero recepción

if (strcmp(_recv,"GO")==0)      // Se ha recibido "GO"
     {
       _count=0;          // Empieza cuenta adquisiciones AD
       _blink_max_count=BLINK_MAX_COUNT_ACQUIRING; // Parpadeo rápido
       _blink_count=0;
       _last_ad_read_time=millis(); // Ultimo instante de adquisición
     }
     return;
  }

  if (_count>=0 && count<10)          // Contando entre 0 y 9 adquisiciones AD
  {
     unsigned long cur_time=millis();
     if (cur_time-_last_ad_read_time >= SAMPLING_TIME_ms)
     // Si ha pasado más del tiempo de muestreo
     {
        int ad=analogRead(A3);   // Leer canal analogico
        char snd[40];

        sprintf(snd,"AD=%d\n",ad); // Escribir valor AD en cadena
        Serial.write(snd);         // Enviar cadena por puerto serie
        _count++;                  // Incrementa cuenta conversiones AD
        _last_ad_read_time=cur_time; // Ultimo instante de adquisición
     }
     return;
  }

  if (_count==10)       // Fin de cuenta: terminar
  {
     Serial.write("END\n");   // Enviar cadena final
     _count=-1;         // Reinicia cuenta adquisiciones AD
     _blink_max_count=BLINK_MAX_COUNT_WAITING;
     _blink_count=0;
  }
}

void CheckBlink()
{
    _blink_count++;
    if (_blink_count>=_blink_max_count)
    {
        _blink_count=0;
        _blink_value=(_blink_value==LOW) ? HIGH : LOW;
        digitalWrite(LED_BUILTIN,_blink_value);
    }
}
```