
Dispositivo hardware de bajo coste para bancos de pruebas de desarrollo de drivers USB en Linux



TRABAJO FIN DE GRADO

Guillermo Gascón Celdrán y Javier Rodríguez-Avello Tapias

Dirigido por: Juan Carlos Sáez Alcaide

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid
Curso 2022-2023

Dispositivo hardware de bajo coste para bancos de pruebas de desarrollo de drivers USB en Linux

Memoria de Trabajo Fin de Grado

Guillermo Gascón Celdrán y Javier Rodríguez-Avello Tapias

Dirigido por: Juan Carlos Sáez Alcaide

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid
Curso 2022-2023

Resumen

El protocolo de comunicación USB, ampliamente usado en numerosos dispositivos en todo el mundo, resulta complejo a la hora de realizar modificaciones para determinados periféricos, sobre todo a la hora de realizar cambios en la programación del dispositivo en cuestión. En este proyecto se ha trabajado en la realización de un firmware para un chip determinado (ATTiny85) que sea capaz de realizar distintas funciones sobre periféricos, que trabajen con un driver cargado en Linux, todo ello programado en lenguaje C.

La complejidad de la tecnología/especificación USB unido al tiempo limitado que puede dedicarse a describir aspectos de bajo nivel de entrada-salida en las titulaciones de Informática, dificulta la adquisición de conocimientos sobre desarrollo de drivers USB, muy valorados en sectores estratégicos. Asimismo, para iniciarse en el desarrollo de este tipo de drivers es preciso disponer de hardware suficientemente sencillo y de bajo coste. Un ejemplo de dispositivo para iniciación es el dispositivo Blinkstick Strip, que cuenta con 8 LEDs de colores cuyo estado puede alterarse individualmente.

En este proyecto se diseña y programa un dispositivo USB de bajo coste que pueda usarse masivamente como banco de pruebas para el desarrollo de drivers USB. Dicho dispositivo estará formado por varios componentes (LEDs, pantallas LCD, sensores, etc.), y permitirá al desarrollador familiarizarse con distintos aspectos y modos de interacción con el hardware USB, para poder afrontar la complejidad de USB de forma gradual. Además del diseño del hardware, y la construcción de un prototipo del mismo empleando un microcontrolador, como el ATTiny85, será necesario el desarrollo del firmware del dispositivo así como de un conjunto de drivers de ejemplo (preferentemente en Linux), para exponer al usuario los distintos componentes del dispositivo USB.

palabras clave: Firmware, USB, endpoint, kernel Linux, report-ID, AVR, V-USB, Display LCD, Digispark.

Abstract

The USB communication protocol, widely used in numerous devices around the world, is complex when making changes to certain peripherals, especially when making changes to the programming of the device in question. In this project we work on the realization of a firmware for a certain chip (ATTiny85) that is capable of performing different functions on peripherals, that work with a driver loaded in Linux, all programmed in C language.

The complexity of the USB technology/specification together with the limited time that can be devoted to describing low-level input-output aspects in Computer Science degrees, makes it difficult to acquire knowledge on USB driver development, highly valued in strategic sectors. Likewise, to start developing this type of drivers it is necessary to have sufficiently simple and low-cost hardware. An example of a starter device is the Blinkstick Strip device, which has 8 colored LEDs whose state can be individually altered.

In this project, a low-cost USB device is designed and programmed that can be used massively as a test bench for the development of USB drivers. Said device will be made up of several components (LEDs, LCD screens, sensors, etc.), and will allow the developer to become familiar with different aspects and modes of interaction with USB hardware, in order to gradually face the complexity of USB. In addition to the hardware design, and the construction of a prototype of it using a microcontroller, such as the ATTiny85, it will be necessary to develop the firmware of the device as well as a set of example drivers (preferably in Linux), to expose the user to the various components of the USB device.

Autorización de difusión y utilización

Los abajo firmantes, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado “Dispositivo hardware de bajo coste para bancos de pruebas de desarrollo de drivers USB en Linux”, realizado durante el curso académico 2022-2023 bajo la dirección de Juan Carlos Sáez Alcaide en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Guillermo Gascón Celdrán y Javier Rodríguez-Avello Tapias

Juan Carlos Sáez Alcaide

Índice general

Resumen	iii
Abstract	v
Autorización de difusión y utilización	i
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	1
1.3 Plan de trabajo	2
1.4 Organización de la memoria	2
2 Introducción al hardware y firmware utilizado con USB	3
2.1 Microcontroladores AVR	3
2.2 Microcontrolador ATTiny85	3
2.3 Librería V-USB	5
2.4 Placa Digispark con interfaz USB	7
3 Periféricos utilizados para el desarrollo del firmware	9
3.1 Anillo de LEDs de colores	9
3.1.1 Smart Sift Register en NeoPixel LED	10
3.1.2 Protocolo One-Wire	11
3.2 Pantalla LCD OLED	12
3.3 Sensor de temperatura	12
4 Test chapter	13
How-To Flash a Custom Firmware	15
4.1 Hardware	15
4.2 Software	15
4.3 Wiring	15
4.4 Flashing an Arduino Sketch	16
4.5 Flashing a custom hex file	17

How-To Monitor USB Traffic via Wireshark	19
4.6 Software	19
4.7 Opening Wireshark	19
4.8 Selecting the right usbmonX interface	19
4.9 Starting the capture	20
4.10 Filtering the traffic	20
Bibliografia	23

Índice de cuadros

Índice de figuras

2.1	Chip ATTiny85	4
2.2	Integración del entorno Arduino con ATTiny85	4
2.3	Detalle de los pines del ATTiny85	5
2.4	Placa de desarrollo Digispark	8
3.1	Tira de LEDs circular de NeoPixel sobre una placa Arduino	9
3.2	Disposición de los pines y del controlador en cada LED	10
3.3	Animación sobre la disposición de los bits dentro del controlador de cada LED	11
4.1	Esto es el título	13
4.2	Arduino Tools configuration	16
4.3	Firmware upload output	17
4.4	Wireshark capturing USB traffic	20
4.5	Filter example	21

Capítulo 1

Introducción

En los siguientes apartados se explican los motivos de la realización del proyecto, los objetivos y la planificación de las tareas para llevarlo a cabo.

1.1 Motivación

El protocolo USB es ampliamente usado en todo el mundo para la comunicación con casi cualquier periférico. De forma sencilla, cualquier usuario con un ordenador puede conectar un dispositivo, da igual la funcionalidad que tenga, al puerto USB de su ordenador y éste reconocerlo para que pueda comunicarse con la CPU y realizar la función para la que ha sido desarrollado.

Para que todo esto se pueda dar, el protocolo USB tiene una complejidad enorme en cuanto a paquetes de comunicación entre la CPU y el dispositivo o la controladora USB, a parte de necesitar un driver instalado en el sistema operativo para que pueda interpretarlo. A pesar de lo sencillo que pueda parecer el hardware (2 cables de datos), hay muchos elementos en la comunicación USB que tienen lugar para que el dispositivo pueda ser reconocido.

Para poder estudiar todo el protocolo USB y la interacción entre el dispositivo y la CPU, en este proyecto hemos desarrollado un firmware en C que realiza una serie de funciones sobre una pantalla LCD y un sensor de temperatura, utilizando un microchip ATTiny85 sobre una placa Digispark, que contiene el propio controlador USB.

1.2 Objetivos

Uso del firmware como banco masivo de pruebas para otros proyectos que puedan utilizar la librería V-USB.

1.3 Plan de trabajo

Para el desarrollo de este proyecto, se han mantenido distintas reuniones con los directores del proyecto y entre los desarrolladores. En la parte inicial del proyecto, se ha estudiado el funcionamiento de la librería V-USB con distintos proyectos que utilizan otros microchips, para ver qué uso se hacen de los report-IDs y las distintas funciones empleadas en el código. Se ha utilizado el software Wireshark para estudiar los endpoints usados y los paquetes URBs en la comunicación, y así estudiar la configuración del software y la posibilidad de utilizar, por ejemplo, interrupciones.

Para el desarrollo de los periféricos que se van a usar conjuntamente en la placa Digispark, Guillermo se ha encargado de estudiar y desarrollar el código correspondiente al sensor de temperatura y Javier al correspondiente con la pantalla LCD 2x16.

Para la integración del código, se ha utilizado GitHub.

1.4 Organización de la memoria

Para la realización de esta memoria, se ha dividido la misma en varios capítulos. En el primero, se explica a modo de introducción el hardware utilizado y algunos aspectos de bajo nivel sobre la librería utilizada. En el siguiente capítulo, se explican los elementos hardware utilizados, como el anillo circular de leds o la pantalla LCD OLED.

Capítulo 2

Introducción al hardware y firmware utilizado con USB

En este capítulo se detalla el uso de la librería utilizada para el desarrollo del firmware, que está basado en V-USB. También se explican los detalles hardware del microchip utilizado para ejecutar el firmware, y la placa embebida que utiliza la interfaz USB para la comunicación con el driver de nuestro sistema operativo.

2.1 Microcontroladores AVR

La familia de microcontroladores AVR provienen del fabricante Atmel, diseñados para el desarrollo de sistemas empujados y la ejecución de aplicaciones relacionados con robótica y una gran variedad de usos dentro del sector industrial.

Esta familia de microcontroladores AVR están caracterizados por su bajo consumo energético, además de tener una alta velocidad de procesamiento y una interfaz de programación relativamente fácil de usar. Cuentan con una arquitectura RISC, permitiendo un procesamiento más rápido y un uso eficiente de los recursos.

Los microcontroladores AVR suelen tener una memoria flash para el almacenamiento de programas, SRAM para el almacenamiento de datos y una gran variedad de periféricos como temporizadores, UART y ADC para interactuar con dispositivos externos. Muchos de estos microcontroladores son usados en Arduino para el desarrollo de proyectos basados en la interfaz USB.

2.2 Microcontrolador ATTiny85

Este es el microcontrolador usado en el desarrollo del firmware USB para el control de un anillo circular de leds junto a una pantalla LCD OLED y un sensor de temperatura.

Como se ha descrito previamente, este microcontrolador forma parte de la familia de AVR. Es un microchip de 8 bits de bajo consumo y alto rendimiento, cuenta con 8 kilobytes de memoria flash, 512 bytes de EEPROM y 512 bytes de SRAM. Tiene 6 pines de entrada/salida de propósito general (GPIOs), que se pueden usar para distintos propósitos, como entrada o salida digital, PWM o entrada analógica. También tiene un oscilador interno con frecuencias ajustables, eliminando la necesidad de un cristal o resonador externo.

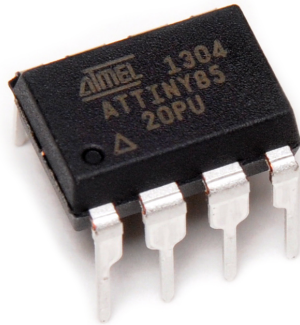


Figura 2.1: Chip ATTiny85

El ATTiny85 es un microcontrolador versátil y se puede usar en una gran variedad de aplicaciones, como sistemas de control, monitoreo de sensores y dispositivos que funcionan con baterías. Uno de los usos más populares de ATTiny85 (y es en el que se ha trabajado en este proyecto) es el desarrollo de proyectos pequeños y de bajo costo.

Para facilitar el desarrollo de estos proyectos, se ha añadido compatibilidad con este microcontrolador al entorno Arduino, lo que facilita en gran manera el programar e interactuar con dispositivos externos.

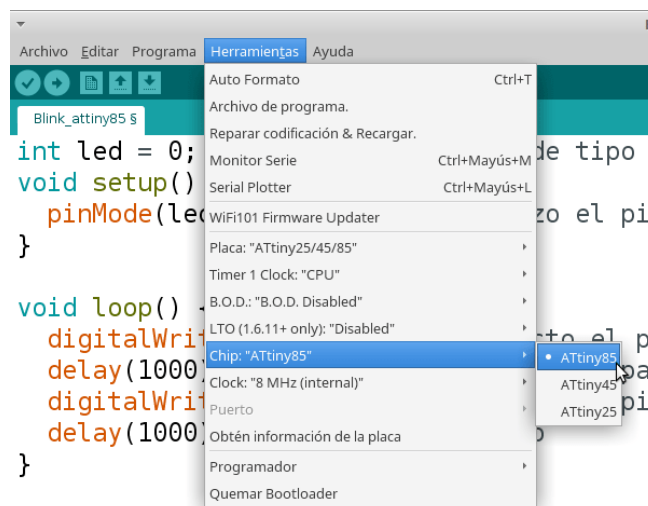


Figura 2.2: Integración del entorno Arduino con ATTiny85

En cuanto a los componentes hardware que integra, contamos con dos temporizadores 8 bits con PWM, un convertidor analógico-digital (ADC) de 8 canales, y una interfaz serie que se puede configurar para usarse como SPI, I2C o UART. Para la pantalla OLED, se ha utilizado el protocolo I2C con sus correspondientes librerías en C, que posteriormente se detallará.

Un factor a destacar de este microchip es el bajo consumo que requiere para funcionar, por lo que para futuras versiones del proyecto se puede integrar una batería para demostrar que no necesita de una cantidad muy grande de energía y puede aprovechar al máximo la duración de la batería.

A continuación, se detalla el esquema de la configuración de cada uno de los pines para el chip.

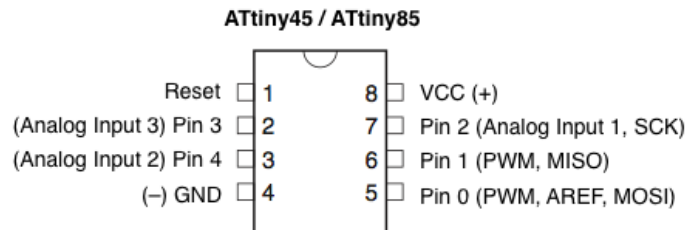


Figura 2.3: Detalle de los pines del ATTiny85

2.3 Librería V-USB

La librería V-USB [1] se usa para la implementación de software para dispositivos USB de baja velocidad que utilizan microcontroladores AVR (como los ATTiny o Atmel). Permite que estos microcontroladores actúen como un dispositivo USB, permitiendo la comunicación con el driver del host u otros dispositivos utilizando este protocolo. Más adelante se detallan algunos aspectos de esta comunicación, como el uso que hace esta librería de los report-id o los endpoints.

La gran ventaja de V-USB con respecto a otras librerías es el pequeño tamaño y bajo consumo que hace de los recursos disponibles. Para su funcionamiento, se requiere unos pocos kilobytes de código firmware y una pequeña cantidad de memoria RAM para funcionar. Esto es especialmente ventajoso para dispositivos como el ATTiny85. Además, esta librería no necesita de ningún componente hardware adicional, solamente un chip con interfaz USB, reduciendo notablemente su costo y diseño del circuito.

Para la comunicación USB, esta librería hace uso de dos endpoints en el chip ATTiny85: el endpoint 0 y endpoint 1. El endpoint 0 es usado para las transferencias de control, es decir, inicializar y configurar la interfaz USB. El endpoint 1, se usa para la transferencia de datos, es decir, para enviar y recibir los informes USB HID.

Para la configuración de los endpoints, están definidos en el archivo cabecera `usbconfig.h`. En el caso del ATTiny85, dicha configuración se puede encontrar en el directorio `usbdrv`. A continuación, se destacan los puntos más importantes de la configuración software utilizada en la librería.

En el siguiente fragmento de código, se encuentra la configuración definida para el puerto de entrada/salida usado en la configuración USB (para el ATTiny85, se utiliza el puerto B), junto con los bits usados para las señales D+ y D-. También se establece en la última línea la frecuencia a la que trabajará el dispositivo USB.

```
#define USB_CFG_IOPORTNAME      B
#define USB_CFG_DMINUS_BIT      3
#define USB_CFG_DPLUS_BIT       4
#define USB_CFG_CLOCK_KHZ       (F_CPU/1000)
```

A continuación se muestra la configuración para las interrupciones en los endpoints, en este caso al estar la variable a 1 se hace uso de interrupciones. En la macro `USB_CFG_INTR_POLL_INTERVAL` se indica el intervalo en milisegundos para las interrupciones por polling.

```
#define USB_CFG_HAVE_INTRIN_ENDPOINT  1
#define USB_CFG_HAVE_INTRIN_ENDPOINT3 1
#define USB_CFG_EP3_NUMBER             3
#define USB_CFG_SUPPRESS_INTR_CODE     0
#define USB_CFG_INTR_POLL_INTERVAL     100
```

```
#define USB_CFG_IS_SELF_POWERED        0
#define USB_CFG_MAX_BUS_POWER          40
#define USB_CFG_IMPLEMENT_FN_WRITE     1
#define USB_CFG_IMPLEMENT_FN_READ      1
#define USB_CFG_IMPLEMENT_FN_WRITEOUT  0
#define USB_CFG_HAVE_FLOWCONTROL        0
#define USB_CFG_LONG_TRANSFERS         0
#define USB_COUNT_SOF                   1
```

En el siguiente párrafo de código, se muestra la configuración definida para establecer diversos parámetros como el Vendor-ID o el Device-ID, a parte de los distintos nombres para el dispositivo, o el número de serie.

```
#define USB_CFG_CHECK_DATA_TOGGING     0
#define USB_CFG_HAVE_MEASURE_FRAME_LENGTH 1
#define USB_CFG_VENDOR_ID              0xA0, 0x20
#define USB_CFG_DEVICE_ID              0xE5, 0x41
#define USB_CFG_DEVICE_VERSION         0x00, 0x02

#define USB_CFG_VENDOR_NAME            'x', 'G', 'G', 'C'
```

```

#define USB_CFG_VENDOR_NAME_LEN 4

#define USB_CFG_DEVICE_NAME
'p', 'w', 'n', 'e', 'd', 'D', 'e', 'v', 'i', 'c', 'e'
#define USB_CFG_DEVICE_NAME_LEN 11

#define USB_CFG_SERIAL_NUMBER
'N', 'I', 'T', 'R', 'A', 'M', '0', '1', '-', '1', '.', '4'
#define USB_CFG_SERIAL_NUMBER_LEN 12
#define USB_CFG_DEVICE_CLASS 0
#define USB_CFG_DEVICE_SUBCLASS 0
#define USB_CFG_INTERFACE_CLASS 3
#define USB_CFG_INTERFACE_SUBCLASS 0
#define USB_CFG_INTERFACE_PROTOCOL 0
#define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH 58

```

Las siguientes líneas muestran la configuración relacionada con el manejo de interrupciones para el dispositivo USB. La macro `USB_INTR_CFG` nos indica la máscara de bits para el cambio de puerto de E/S para permitir interrupciones en el pin D+. Así, la macro `USB_INTR_CFG_SET` especifica el registro de desplazamiento para dicho cambio.

```

#ifndef SIG_INTERRUPT0
#define SIG_INTERRUPT0 _VECTOR(1)
#endif
#define USB_INTR_CFG PCMSK
#define USB_INTR_CFG_SET (1<<USB_CFG_DPLUS_BIT)
#define USB_INTR_ENABLE_BIT PCIE
#define USB_INTR_PENDING_BIT PCIF
#define USB_INTR_VECTOR SIG_PIN_CHANGE

```

La macro `USB_INTR_ENABLE_BIT` especifica el bit en el registro de control de interrupción de cambio de pin para habilitar las interrupciones de cambio de pin. Por otra parte, `USB_INTR_PENDING_BIT` especifica el bit que indica la presencia de una interrupción pendiente.

2.4 Placa Digispark con interfaz USB

La placa Digispark cuenta con el chip ATTiny85, es la placa elegida para desarrollar el firmware de este proyecto.

Los detalles técnicos con los que cuenta esta placa son los siguientes:

- Microcontroller: ATTiny85
- Operating Voltage: 5V

-
- Input Voltage (recommended): 7-12V
 - Input Voltage (limits): 5-16V
 - Digital I/O Pins: 6
 - PWM Channels: 3
 - Analog Input Channels: 4
 - Flash Memory: 8 KB (of which 2.5 KB is used by the bootloader)
 - SRAM: 512 bytes
 - EEPROM: 512 bytes

Es una placa realmente compacta, con un tamaño de 25 mm x 18 mm. Incluye una interfaz USB para programación y comunicación con el host, así como un regulador de voltaje y LED de encendido.

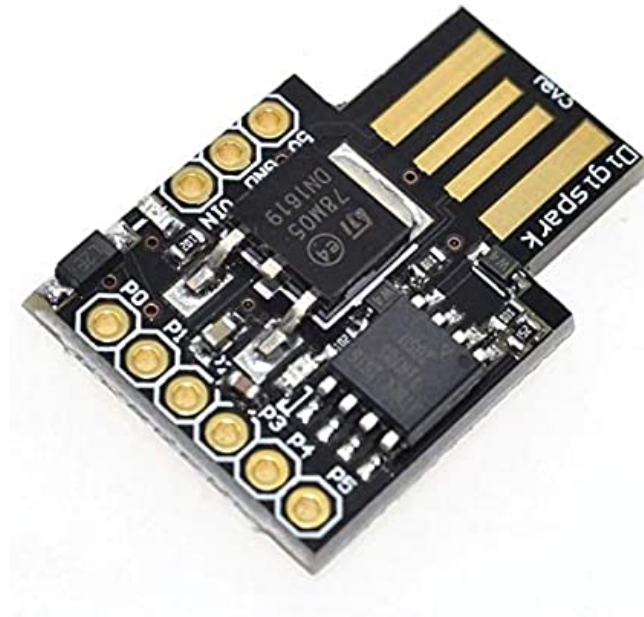


Figura 2.4: Placa de desarrollo Digispark

Una característica única de la placa Digispark es que viene preprogramada con un cargador de arranque que permite programarla a través de USB usando el IDE de Arduino (llamado Micronucleous Bootloader). Con ello se puede programar código desde Arduino fácilmente a la placa sin necesidad de un programador independiente. Aunque para el desarrollo de nuestro proyecto, se ha optado por utilizar la librería V-USB sin el entorno de Arduino, por lo que se requiere de un compilador de AVR y un programador externo.

Capítulo 3

Periféricos utilizados para el desarrollo del firmware

En este capítulo se describe con detalles de bajo nivel los periféricos utilizados en el firmware desarrollado.

3.1 Anillo de LEDs de colores

Para probar el firmware, en la primera versión se ha hecho uso de la tira de leds circular fabricado por la empresa NeoPixel. Dichos leds se encuentran en disposición circular con colores RGB direccionables individualmente.

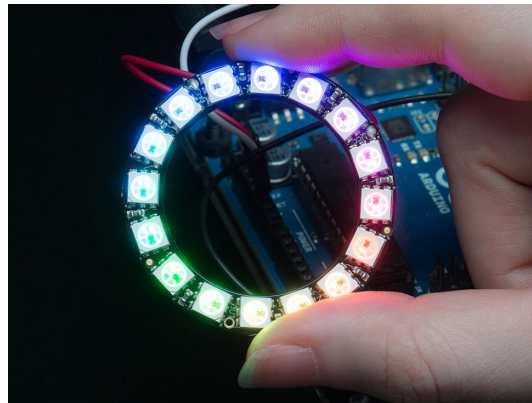


Figura 3.1: Tira de LEDs circular de NeoPixel sobre una placa Arduino

Estos LEDs están conectados en cadena y cada LED está controlado por una sola línea de datos. La línea de datos usa un protocolo de comunicación específico llamado *One-Wire Protocol*, que se implementa a través de software y se usa para controlar el color y el brillo de cada LED individual.

La tira de LED circular se alimenta con una fuente de alimentación de 5 V que proporciona la placa Digispark. Cada LED consume una cierta cantidad de energía, y el consumo total de energía de la tira depende de la cantidad total de LEDs y la configuración de brillo.

La tira de LEDs circular generalmente está controlada por un microcontrolador, en nuestro caso el ATTiny85 se encarga de ello, aunque también existen interfaces de comunicación para estas tiras de LEDs especializado, que envía los datos de color y brillo a los LED mediante el protocolo *One-Wire*. El microcontrolador se puede programar para crear una amplia variedad de efectos de iluminación, incluidos ciclos de color, animaciones y patrones.

3.1.1 Smart Shift Register en NeoPixel LED

Cada LED individual en la tira de LEDs circular de NeoPixel contiene un registro de cambio inteligente, con un pequeño microcontrolador y un LED RGB (Red, Green, Blue). El microcontrolador es responsable de controlar el color y el brillo de los LED y comunicarse con los otros LED de la cadena.

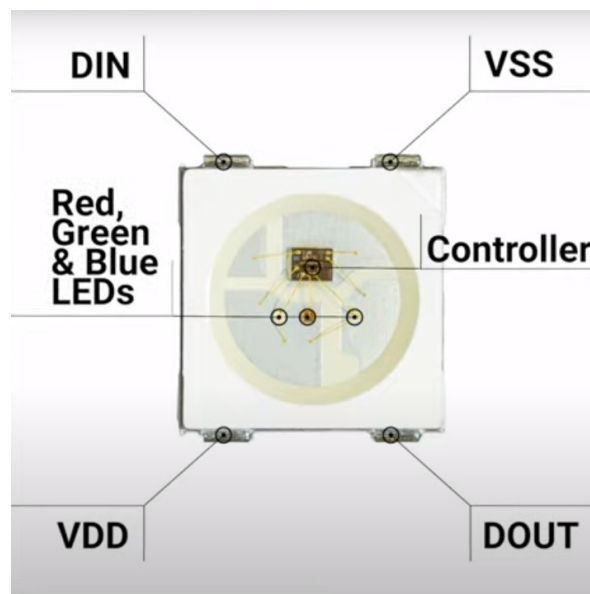


Figura 3.2: Disposición de los pines y del controlador en cada LED

El Smart Shift Register LED utiliza un protocolo de comunicación específico llamado *One-Wire Protocol* para recibir datos del microcontrolador. Dicho protocolo se basa en otro protocolo serie de bajo nivel que utiliza una sola línea de datos para transmitirlos entre el microcontrolador y los LED.

Para enviar datos al LED Smart Shift Register, el microcontrolador envía una serie de pulsos en la línea de datos, que el LED interpreta como una secuencia de comandos.

Los comandos incluyen instrucciones para configurar el color y el brillo de cada LED individual, así como para configurar el tiempo y la sincronización de la cadena de LEDs.

El LED de registro de desplazamiento inteligente también incluye un conjunto de registros de desplazamiento, que se utilizan para almacenar los datos de color y brillo de cada LED de la cadena. Los registros de desplazamiento permiten que el LED de registro de desplazamiento inteligente reciba y almacene datos del microcontrolador, y luego pase esos datos al siguiente LED de la cadena.

Además de los registros de desplazamiento, el Smart Shift Register LED también incluye un conjunto de controladores de corriente constante, que se utilizan para controlar el brillo de cada LED individual. Los controladores de corriente aseguran que cada LED reciba una cantidad constante de energía, independientemente del número de LEDs en la cadena o la configuración de brillo.

! Referencia al vídeo https://www.youtube.com/watch?v=PPVi3bI7_Z4

3.1.2 Protocolo One-Wire

El protocolo *One-Wire* es un protocolo de comunicación serie de bajo nivel que se utiliza para comunicarse con dispositivos que tienen una sola línea de datos. Fue desarrollado por *Dallas Semiconductor* en la década de los 90 y ahora se usa en una amplia gama de aplicaciones, incluidos sensores de temperatura, tiras de LED y EEPROM.

Dicho protocolo está diseñado para ser lo más simple, confiable y con bajo costo. Utiliza una sola línea de datos, que es bidireccional y normalmente está conectada al microcontrolador. La línea de datos también está conectada a uno o más dispositivos *One-Wire*, que suelen ser sensores u otros dispositivos de bajo consumo.

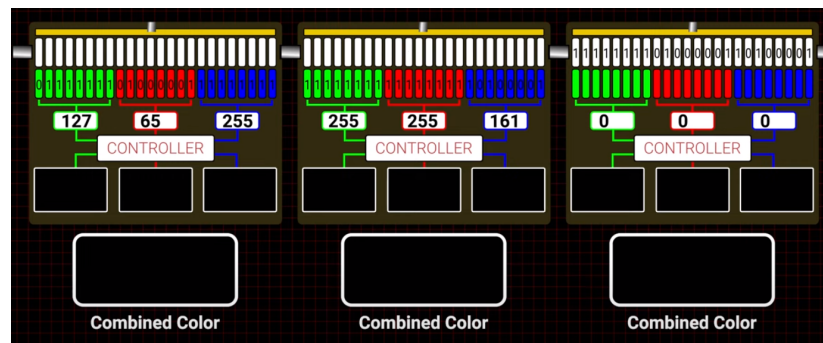


Figura 3.3: Animación sobre la disposición de los bits dentro del controlador de cada LED

Para comunicarse con un dispositivo *One-Wire*, el microcontrolador envía una serie de pulsos en la línea de datos, que el dispositivo interpreta como comandos. Los pulsos se generan utilizando una secuencia de tiempo específica, que incluye una combinación de niveles de voltaje alto y bajo en la línea de datos.

El protocolo *One Wire* utiliza una variedad de comandos diferentes para comunicarse con el dispositivo, incluidos comandos para leer y escribir datos, establecer opciones de configuración y enviar señales de control. Estos comandos se envían en una secuencia específica y se cronometran de acuerdo con las especificaciones del protocolo.

Una de las características más importantes es que incluye un mecanismo CRC (comprobación de redundancia cíclica) incorporado, que se utiliza para garantizar la integridad de los datos que se transmiten. El mecanismo CRC utiliza un algoritmo matemático para generar una suma de verificación para los datos, que luego se transmite junto a ellos. El dispositivo receptor puede usar el algoritmo para generar la suma de verificación para los datos recibidos, y compararla con la suma de verificación transmitida para verificar que los datos se hayan transmitido correctamente.

3.2 Pantalla LCD OLED

3.3 Sensor de temperatura

Capítulo 4

Test chapter

Esto es solo un ejemplo de capítulo. Vamos a citar algo ??? [2]. Esto es otra cita [1]



Figura 4.1: Esto es el título

- UNO
- DOS
- TRES

OK	A	B
AHSAJHD	ASHJDHAJSH ^o	SAHJDHJASH
SADHJHJ	ASHJDFHJASH	ASDASD

```
#ifdef CONFIG_PMC_PERF
    int safety_control;
    unsigned char prof_enabled;
#endif
```

How-To Flash a Custom Firmware

The *Digispark ATTiny85* comes by default with a bootloader called *Micronucleus* that can talk to the host computer via serial communication, giving the user the option to upload new code through the USB port and avoiding the use of a *USBASP-like* programmer.

That feature may be useful sometimes, but there are certain cases where we may need to flash a custom bootloader, *C/C++* firmware or any sort of program that need to completely erase the ROM, and this can't be done using Micronucleus.

So in this quick guide, we're going to solve that.

4.1 Hardware

- **ICSP Programmer:** we'll use that to talk to the chip's ROM. *USBASP* is the one I'm going to be using, but any programmer with *MISO/MOSI/SCK/RST* pins will work, even an *Arduino* flashed with the *ArduinoISP* sketch.
- **Digispark ATTiny85**

4.2 Software

- **Arduino IDE:** we'll use the *AVR* tools that come with it.

4.3 Wiring

ISCP Programmer	Digispark ATTiny85
MOSI	PB0
MISO	PB1
SCK	PB2
RESET	PB5
5V	5V

ISCP Programmer	Digispark ATTiny85
GND	GND

4.4 Flashing an Arduino Sketch

The first thing to do is click on the *Tools* tab and select the correct *Board* and CPU. Then, we select the *Programmer* we're using.

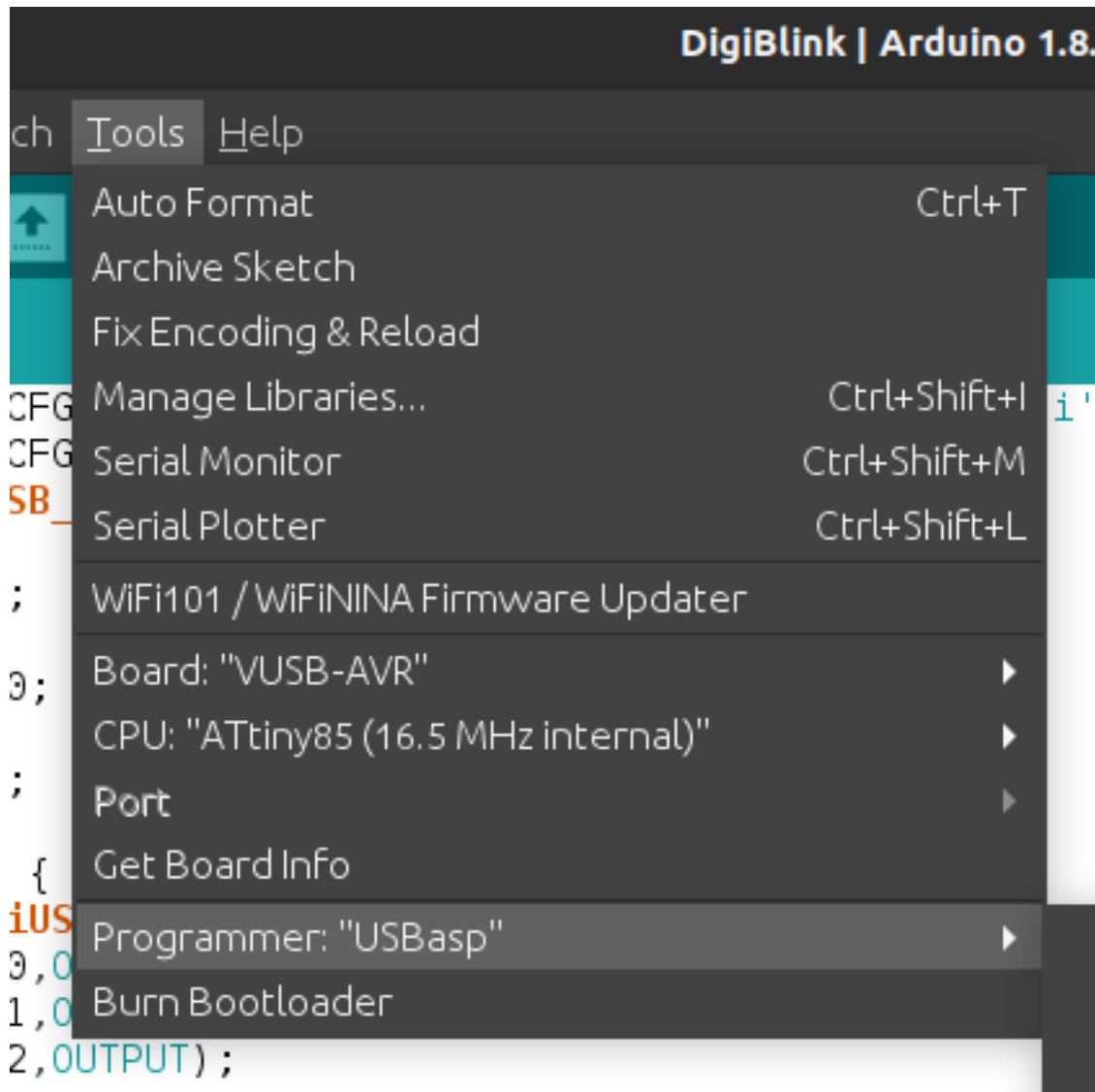


Figura 4.2: Arduino Tools configuration

Once all the parameters are set, we just need to hit the flash button and cross our fingers. If the flashing process has finished successfully, we'll get a message like the following:


```
Done uploading.
avrdude: 3340 bytes of flash written
avrdude: verifying flash memory against /tmp/arduino_build_779856/DigiBlink.ino.hex:
avrdude: load data flash data from input file /tmp/arduino_build_779856/DigiBlink.ino.hex:
avrdude: input file /tmp/arduino_build_779856/DigiBlink.ino.hex contains 3340 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 0.90s

avrdude: verifying ...
avrdude: 3340 bytes of flash verified

avrdude done. Thank you.
```

30 VUSB-AVR, ATtiny85 (16.5 MHz internal)

Figura 4.3: Firmware upload output

4.5 Flashing a custom hex file

Flashing an existent hex file may be useful if we want to install a pre-compiled *firmware/bootloader* or upload a *C/C++* program.

Arduino comes with some *AVR* utilities we can use from our terminal to compile and flash code to *AVR microcontrollers*.

First thing is to locate our *Arduino IDE* directory, below is one way to do it:

```
$ ls -l `which arduino`
lrwxrwxrwx root root 73 B Tue Jul 12 16:44:01 2022 /usr/local/bin/arduino /home/ggc/
```

Once we know where it's located, let's check for the following utilities and files:

- `arduino-1.8.19/hardware/tools/avr/bin/avrdude`
- `arduino-1.8.19/hardware/tools/avr/etc/avrdude.conf`

Now all that is left to do is flash the hex file using the command line using the following arguments:

- **-C**, default Arduino IDE configuration file
- **-v**, verbose output
- **-p**, microcontroller
- **-c**, programmer
- **-P**, port
- **-U**, instruction. Syntax: *memtype:op:filename[:format]*

```
arduino-1.8.19/hardware/tools/avr/bin/avrdude -Carduino-1.8.19/hardware/tools/avr/etc/
```

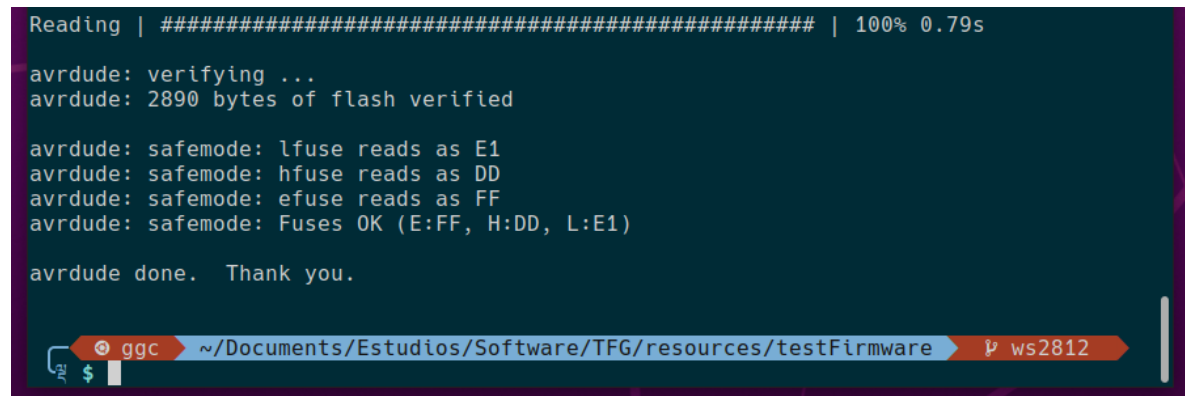
More `avrdude` arguments and in-depth explanations can be found here: https://www.non-gnu.org/avrdude/user-manual/avrdude_3.html

If the flashing process has finished successfully, we'll get a message like the following:

```
Reading | ##### | 100% 0.79s
avrdude: verifying ...
avrdude: 2890 bytes of flash verified

avrdude: safemode: lfuse reads as E1
avrdude: safemode: hfuse reads as DD
avrdude: safemode: efuse reads as FF
avrdude: safemode: Fuses OK (E:FF, H:DD, L:E1)

avrdude done. Thank you.
```

A terminal window with a dark teal background. The output of the avrdude command is displayed in white text. At the bottom, there is a status bar with a light blue background. It contains a gcc icon, the file path ~/Documents/Estudios/Software/TFG/resources/testFirmware, and the text ws2812.

How-To Monitor USB Traffic via Wireshark

This is a quick guide on how to capture and display USB traffic with Wireshark GUI interface which makes it very easy to interact with.

All the software and testing has been done under Ubuntu 20.04.5 LTS with the kernel 5.15.0-48-generic.

4.6 Software

All the packages needed to monitor the traffic can be installed from the Ubuntu repositories using `apt`.

```
sudo apt install wireshark libpcap0.8
```

Also, the `usbmon` module has to be enabled in order to access the usb traffic.

```
sudo modprobe usbmon
```

4.7 Opening Wireshark

As we're going to be accessing physical hardware, we need to open Wireshark as a superuser. To accomplish that, we can launch it from the terminal as follows:

```
sudo wireshark
```

4.8 Selecting the right `usbmonX` interface

Once we have opened Wireshark, we'll see a few `usbmon` interfaces we can monitor from. To select the correct one, we need to check which bus is our usb device connected to. We can use `lsusb` for that task.

```
$ lsusb
```

```
Bus 001 Device 067: ID 16c0:05df Van Ooijen Technische Informatica HID device
```

In this case, we'll choose *usbmon1* since our device is connected to the bus 001 as we see on the *lsusb* output. Also, we can see that the *DeviceID* is the number 67, this will be useful later for the display filter.

4.9 Starting the capture

To start capturing traffic, we just need to double click on the correct *usbmon* interface and it'll automatically start showing some traffic. Most of the traffic displayed won't be useful for us, so the best thing to do now is apply some filters to only show the traffic regarding our usb device.

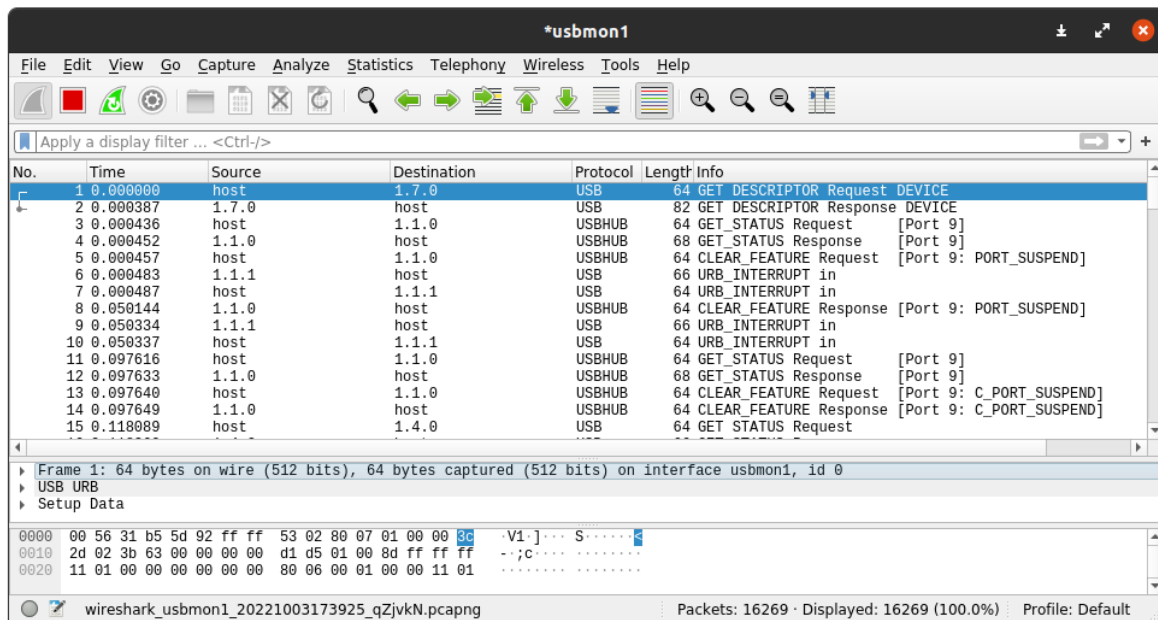


Figura 4.4: Wireshark capturing USB traffic

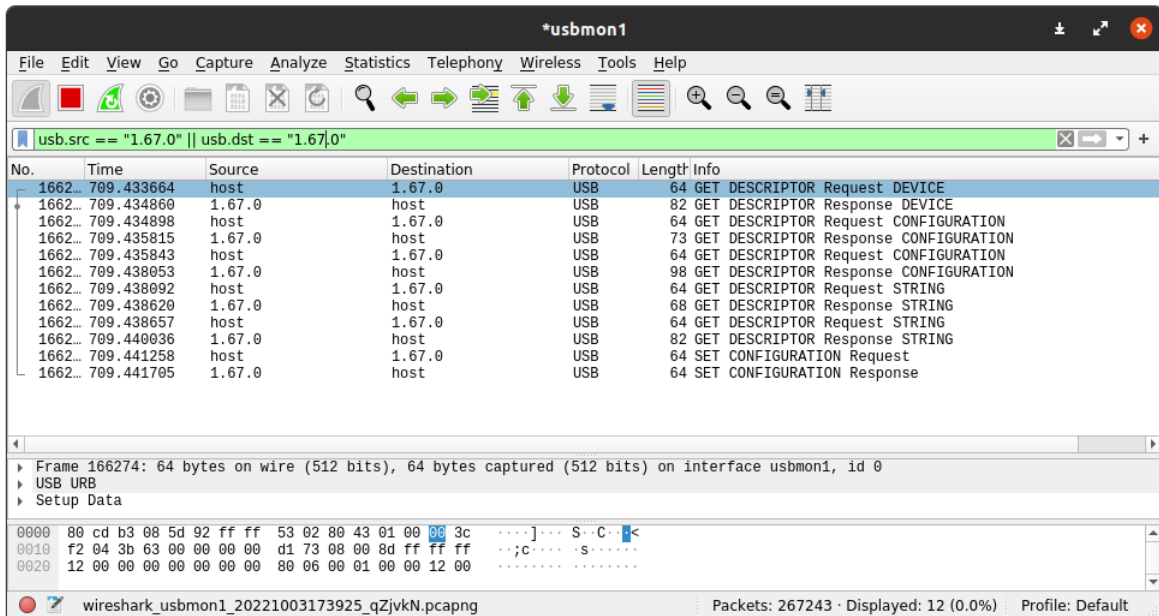
4.10 Filtering the traffic

As I've said before, the best way to display useful traffic is using display filters, so let's do it.

The syntax is very simple, so we can write it our selfs or right click a package from our device and prepare it as a filter, but basically all we need to introduce is:

```
usb.src == "1.67.0" || usb.dst == "1.67.0"
```

Where the first number corresponds to the bus, the second to the device id and the third to the endpoint id.



Bibliografía

[1] OBDEV, «V-USB open-source Project example projects for LCD Display». <https://www.ob-dev.at/products/vusb/projects.html>, 2011.

[2] W. Maurerer, *Professional Linux Kernel Architecture*. Birmingham, UK, UK: Wrox Press Ltd., 2008.