Universidad de Valladolid

TRABAJO DE FIN DE GRADO

Pantalla Multi-LED RGB conectada al IoT *(Internet of Things)* controlada via aplicación Android

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECIFICAS DE TELECOMUNICACIÓN: MENCIÓN EN SISTEMAS ELECTRÓNICOS

Autor
Michael Giovani Rojas Vargas

Tutor
Jesús M. Hernández Mangas

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN
—
Valladolid, Octubre de 2017

Dedicado a esas personas tan fugaces e increíbles como estrellas,
cuyo brillo permanece en nuestras vidas para siempre...
$\oint \pi$

Agradecimientos

Este trabajo de fin de grado no es más que el broche de oro a la primera etapa de mi vida profesional que ahora culmina y que desde luego, ha sido una de las más importantes en mi vida, y que no habría sido posible sin el apoyo de todas las personas que siempre estuvieron ahí. Debo agradecer especialmente:

A mis padres, por que son la inspiración, la razón y la causa. Por que me enseñaron a ser luchador, a seguir adelante por difícil que parezca y que con el sacrificio adecuado, las metas se pueden conseguir.

A mis hermanos, por que al igual que yo, cada enseñanza de mis padres fue aprovechada al máximo, aplicada a nuestra vida y cuyo apoyo nos ha permitido llegar hasta aquí unidos como familia.

A mi sobrino, por que a pesar de llevar tan poco tiempo entre nosotros, la felicidad que trajo a nuestra familia, nos ha vuelto a llenar de alegría y esperanza.

A mi tutor Jesús M. Hernández Mangas, por que sus tácticas de enseñanza me hicieron sentir la pasión por los *sistemas electrónicos*, lo que me motivó a elegirlo como tutor de este proyecto. Gracias por guiarme y apoyarme durante el desarrollo de este TFG.

A mis compañeros de clase, por haber compartido este camino conmigo, por los buenos y malos momentos, por los conocimientos compartidos, por que sabemos que en esta carrera cada uno va a un ritmo distinto y no por eso somos mejores o peores. ¡animo!, no os rindáis, ya falta poco.

A todos los profesores de la carrera, por que evidentemente aportaron sus conocimientos para permitirme conseguir una formación adecuada al perfil de un ingeniero de telecomunicaciones.

Y *a todos los que estuvieron* y que por alguna razón, tuvieron que irse. Por si nunca os agradecí lo mucho que me disteis. Espero veros pronto.

A todos vosotros, ***Gracias***.

Resumen

Este trabajo de fin de grado pretende mostrar el desarrollo de un proyecto de ingeniería de sistemas de telecomunicaciones. Se busca describir las diferentes etapas que se abarcan para conseguir un prototipo final: las especificaciones del dispositivo a desarrollar, las herramientas utilizadas para conseguir dichas funcionalidades, los cambios y contratiempos que puede sufrir el proyecto a medida que éste avanza en su desarrollo y desde luego, como se llega a conseguir un prototipo o producto final que cumpla las especificaciones iniciales.

Este proyecto ha sido enfocado teniendo en cuenta el panorama tecnológico actual, donde es cada vez más común tener todos nuestros dispositivos conectados a Internet, gracias al denominado *Internet of Things (IoT)* y donde es igualmente importante permitir el control desde nuestro *Smartphone* a través de una aplicación intuitiva y simple.

El objetivo principal a grandes rasgos, es diseñar y construir un sistema electrónico que permita el control de una pantalla de LEDs inteligentes para poder visualizar información (en nuestro caso únicamente texto) y cuyo control de contenido se realizará a través de una aplicación para móvil con sistema operativo Android.

Índice general

Índice de figuras

Capítulo 1

Introducción

El *Internet of Things* (IoT) (adaptado al castellano como *Internet de las Cosas*) es un concepto relativamente nuevo de comunicación, que hace referencia a la extensión de las conexiones a la red en el mundo de los objetos de uso doméstico e industrial. Esto es posible gracias al desarrollo de diferentes tecnologías que implementan sensores y/o actuadores, que nos permiten mantener nuestros dispositivos monitorizados y nos brindan la capacidad de poder actuar sobre ellos, permitiéndonos así, denominarlos como dispositivos inteligentes.

El proyecto desarrollado a través de este trabajo de fin de grado, está basado en el uso de algunas de las tecnologías disponibles actualmente, que usan cada vez más en las actividades de nuestra vida cotidiana: LEDS inteligentes, cuyo uso está cada vez más extendido para la iluminación ambiente. Conexiones de redes inalámbricas, indispensables para la comunicación de la gran cantidad de dispositivos inteligentes se utilizan actualmente y dispositivos baratos de tipo microcontrolador, capaces de proporcionar un control sobre diferentes componentes periféricos y de comunicar cualquier tipo de sensor o dispositivo a distancia haciendo posible el *Internet of Things*.

1.1. Propuesta del Problema

Se busca diseñar e implementar un sistema electrónico para el control de visualización de texto en una pantalla multi-LED RGB con conexión inalámbrica. Esta Pantalla estará compuesta por una matriz de LEDs RGB, que será ampliable y podrá mostrar caracteres en cualquier color. Debe poder conectarse a internet y permitirá la modificación del texto visualizado a través de una aplicación sencilla para Android.

Este proyecto fue planteado pensando en todos los procesos de ingeniería detrás del desarrollo de un producto (en este caso un sistema electrónico) que se deben seguir para poder desarrollar un prototipo final que satisfaga los requerimientos de un cliente, que buscan un dispositivo cuyo uso ayude a la modernización de pequeños aspectos del pequeño comercio.

En este documento se da constancia en profundidad del diseño y construcción de un sistema ideado con el fin de satisfacer las necesidades y especificaciones de la pantalla Multi-LED, de las comunicaciones existentes entre los nodos y el servidor encargado de gestionar las comunicaciones, permitiendo en conjunto, el funcionamiento deseado.

1.2. Fases del proyecto

Para llevar a cabo el desarrollo del proyecto, es importante dividirlo en diferentes fases que nos permitan desarrollarlo en pasos escalonados, que sean relativamente simples y que en conjunto nos lleven a la solución o producto final elaborado. Estos pasos deben estar, en la medida de lo posible, temporalmente coordinados de tal forma que nos permita para sacar el mayor provecho de los recursos de los que disponemos y del tiempo dedicado al desarrollo del producto final. A continuación, se listan en la siguiente tabla cada una de las fases que debemos seguir a la hora de poner el proyecto en marcha:

Proyecto: TFG pantalla muti-LED RGB conectada a IoT controlada vía aplicación Android	
Fase 1: Especificación y Planeación	1.1 Estimación Temporal Inicial
	1.2 Estudio y selección de Componentes
Fase 2: Captura Esquemática	2.1 Creación de componentes nuevos
	2.2 Posicionado de componentes (en plano eléctrico)
	2.3 Rutado de conexiones
	2.4 Análisis de consumo energético
	2.5 Análisis Térmico
	2.6 Generación del B.O.M. (bill of materials)
	2.7 Esquemático
Fase 3: Diseño Placa de circuito impreso [PCB]	3.1 Creación de nuevos componentes
	3.2 Definición de Borde de placa y zona de conectores
	3.2 Posicionado de componentes (en software de diseño PCB)
	3.3 Rutado de conexiones
	3.4 Definición de planos de Masa y otros planos.
	3.5 Generación de ficheros
Fase 4: Fabricación PCB	4.1 Envío al fabricante PCB
	4.2 Recepción del pedido
	4.3 Montaje de componentes
Fase 5: Desarrollo firmware y Software	5.1 Especificaciones del software necesario
	5.2 Control de los elementos Hardware: Drivers
	5.3 Búsqueda de los entornos de desarrollo (IDEs) necesarios
	5.4 Programación del microprocesador del módulo WiFi
	5.5 Programación de la aplicación de control en Android
	5.6 Programación del funcionamiento en Servidores externos
	5.7 Documentación y control de cambios de firmware
Fase 6: Verificación del Hardware	6.1 Verificación inicial de la placa
	6.2 Verificación del frimware *on-board*
	6.3 Resolución de problemas
	6.4 Analisis de Prestaciones
	6.5 Montajes y pruebas finales
Fase 7: Documentación del Proyecto	7.1 Realización de la documentación
	7.2 Memoria Final y presentación

Tabla 1.1: Fases del desarrollo del proyecto

Capítulo 2

Especificaciones y Planificación

2.1. Estimación inicial de plazos del proyecto

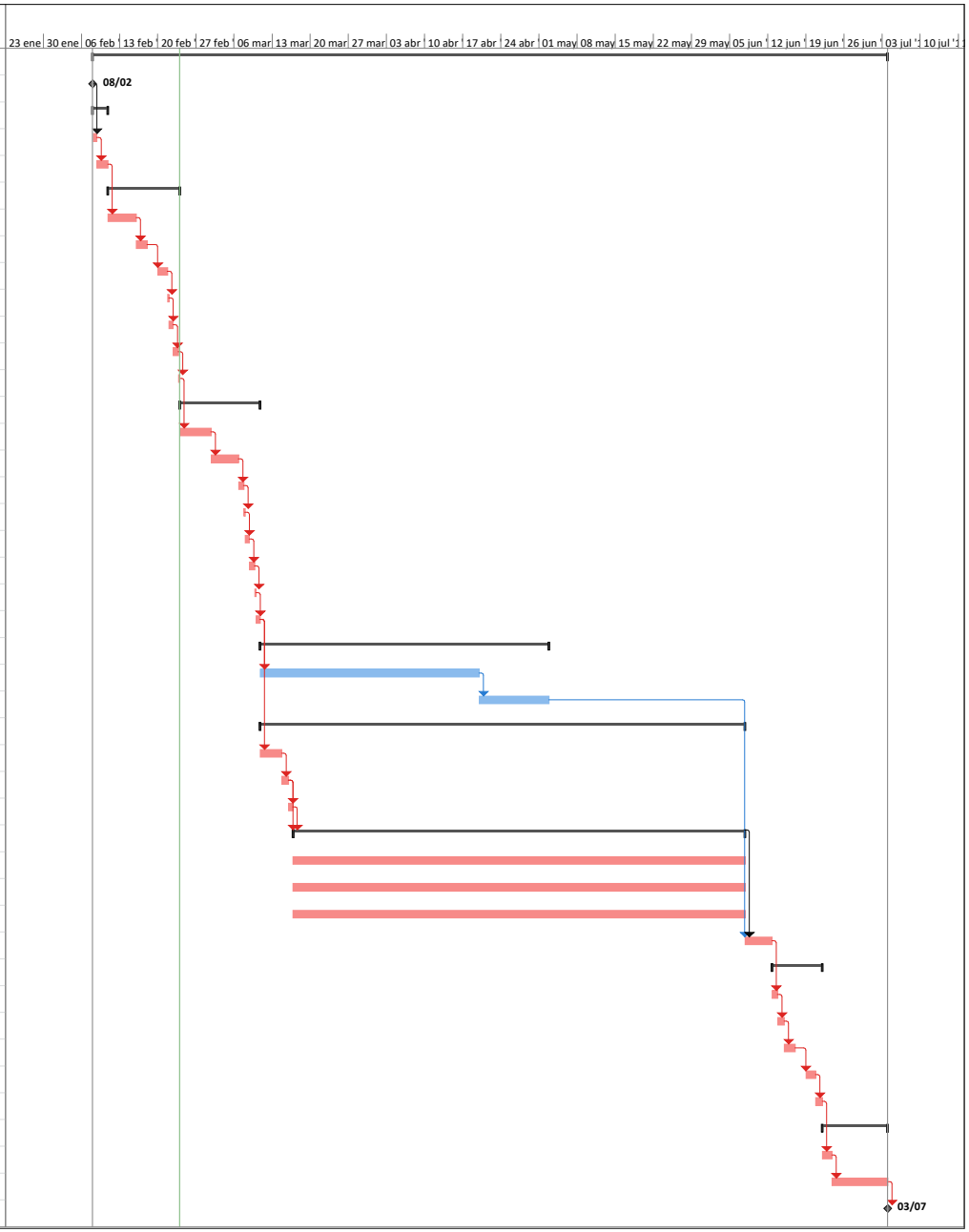
Para cada una de las fases se define un tiempo de duración, determinado principalmente por la complejidad de la tarea. Esta complejidad hará que una tarea nos lleve más o menos tiempo en ser completada. Al realizar la estimación temporal del proyecto, estamos estudiando el orden adecuado para cada una de las actividades a desarrollar, permitiéndonos prever el tiempo que nos llevará el desarrollo completo de nuestro proyecto.

Para realizar esta estimación,utilizamos un Diagrama de Gantt, herramienta muy gráfica para la gestión de proyectos cuyo objetivo principal es exponer el tiempo de dedicación previsto para las diferentes actividades que conforman un proyecto.

Al empezar este TFG, se ha realizado un Diagrama de Gantt con la estimación de tiempos inicial (figura 2.1), la cual no tiene en cuenta cuenta los imprevistos que pueden hacer que nuestro proyecto se retrase. Se puede comparar con un segundo diagrama de Gantt (al final del documento), el cual fue elaborado al finalizar el proyecto y que muestra el tiempo real dedicado a la elaboración este proyecto. Los principales desfases en los plazos que se estimaron en un principio están asociados al no cumplimiento con excesivo rigor de los plazos por diferentes motivos personales, entre los que se incluyen el cumplimiento del horario laboral en las prácticas curriculares de la universidad y la posterior contratación. Trabajar en un proyecto paralelo a la vez que se desarrollaba este TFG, es una fuente evidente de retrasos en los plazos.

Figura 2.1: Diagrama de Gannt inicial

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**Proyecto: TFG pantalla muti-LED RGB conectada a IoT (Internet of Things) controlada via a**	**300 horas**	**mié 08/02/17**	**lun 03/07/17**
2		Inicio	0 horas	mié 08/02/17	mié 08/02/17
3		**Fase 1: Especificación y Planeación**	**8 horas**	**mié 08/02/17**	**vie 10/02/17**
4		Estimación Temporal Inicial	2 horas	mié 08/02/17	mié 08/02/17
5		Estudio y selección de Componentes	6 horas	mié 08/02/17	vie 10/02/17
6		**Fase 2: Captura Esquemática**	**28 horas**	**vie 10/02/17**	**jue 23/02/17**
7		Creación de componentes nuevos	10 horas	vie 10/02/17	mié 15/02/17
8		Posicionado de componentes (en plano eléctrico)	6 horas	jue 16/02/17	vie 17/02/17
9		Rutado de conexiones	4 horas	lun 20/02/17	mar 21/02/17
10		Análisis de consumo energético	2 horas	mar 21/02/17	mar 21/02/17
11		Análisis Térmico	2 horas	mié 22/02/17	mié 22/02/17
12		Documentación del esquemático	2 horas	mié 22/02/17	jue 23/02/17
13		Generación del B.O.M. (bill of materials)	2 horas	jue 23/02/17	jue 23/02/17
14		**Fase 3: Diseño Placa de circuito impreso [PCB]**	**31 horas**	**vie 24/02/17**	**vie 10/03/17**
15		Creación de nuevos componentes	10 horas	vie 24/02/17	mié 01/03/17
16		Posicionado de componentes (en software de diseño PCB)	10 horas	mié 01/03/17	lun 06/03/17
17		Definición de Borde de placa y zona de conectores	2 horas	lun 06/03/17	mar 07/03/17
18		Rutado de conexiones	2 horas	mar 07/03/17	mar 07/03/17
19		Definición de planos de disipación térmica	2 horas	mié 08/03/17	mié 08/03/17
20		Planos de masa y otros	2 horas	mié 08/03/17	jue 09/03/17
21		Documentación PCB	2 horas	jue 09/03/17	jue 09/03/17
22		Generación de ficheros	1 hora	vie 10/03/17	vie 10/03/17
23		**Fase 4: Fabricación PCB**	**100 horas**	**vie 10/03/17**	**mar 02/05/17**
24		Envío al fabricante PCB	26,67 días	vie 10/03/17	mié 19/04/17
25		Montaje de componentes	20 horas	jue 20/04/17	mar 02/05/17
26		**Fase 5: Desarrollo firmware y Software**	**178 horas**	**vie 10/03/17**	**mié 07/06/17**
27		Control de los elementos hardware: Drivers	6 horas	vie 10/03/17	mar 14/03/17
28		Aplicación de control. Especificaciones	5 horas	mar 14/03/17	mié 15/03/17
29		Verificación del software	2 horas	jue 16/03/17	jue 16/03/17
30		**Desarrollo de las Aplicaciónes para el funcionamiento**	**165 horas**	**jue 16/03/17**	**mié 07/06/17**
31		Programación del microprocesador del modulo wifi	165 horas	jue 16/03/17	mié 07/06/17
32		Programación de la aplicación de control en Android	165 horas	jue 16/03/17	mié 07/06/17
33		Programacion del funcionamiento en Servidores externos	165 horas	jue 16/03/17	mié 07/06/17
34		Documentación y control de cambios de firmware	8 horas	mié 07/06/17	lun 12/06/17
35		**Fase 6: Verificación del Hardware**	**23 horas**	**lun 12/06/17**	**mié 21/06/17**
36		Verificación inicial de la placa	4 horas	lun 12/06/17	mar 13/06/17
37		Verificación del software "on-board"	4 horas	mar 13/06/17	mié 14/06/17
38		Resolución de problemas	6 horas	jue 15/06/17	vie 16/06/17
39		Análisis de prestaciones	4 horas	lun 19/06/17	mar 20/06/17
40		Montaje final	5 horas	mar 20/06/17	mié 21/06/17
41		**Fase 7: Documentación del Proyecto**	**24 horas**	**jue 22/06/17**	**lun 03/07/17**
42		Realizacion de la documentación	4 horas	jue 22/06/17	vie 23/06/17
43		Memoria Final y Presentación.	20 horas	vie 23/06/17	lun 03/07/17
44		fin	0 horas	lun 03/07/17	lun 03/07/17



2.2. Estudio de las especificaciones y selección de componentes

Es igualmente importante que, si el proyecto se desarrolla para un cliente, éste nos dé la mayor cantidad de especificaciones que desea para su producto final. En este caso las especificaciones se han descrito en el problema planteado, pero a continuación las listaremos de una manera más detallada:

- **Pantalla Multi-LED RGB. Debe ser escalable:** la pantalla debe ser capaz de representar contenido, en principio texto o imágenes. En este TFG nos centraremos por simplicidad en la representación de texto únicamente. Para esto se debe diseñar una matriz de LEDs, los cuales deben estar espaciados de forma regular, de tal forma que nos permita tener una densidad constante de LEDs a medida que se cumple la escalabilidad de la pantalla.

  Para que sea posible la representación de texto en una matriz de LEDs, es una condición necesaria que los LEDs tengan la posibilidad de ser encendidos individualmente, además de que se pide texto en cualquier color. Estas condiciones hacen que para el proyecto trabajemos con el LED **WS2812B**.

  Este *Smart-LED* vendido comercialmente, posee unas prestaciones que lo hacen ideal para la matriz del proyecto, pues, son direccionables individualmente siempre y cuando se coloquen en una configuración de transmisión de datos en serie. Dentro de este pequeño encapsulado de 5mm x 5mm se incluyen 3 Diodos emisores de luz, uno para cada color RGB y un pequeño controlador, el cual le permite procesar la información que se le envía. Posee 4 pines: dos de los cuales están dedicados para la alimentación (V_{DD} y V_{SS}), mientras que los otros dos son los pines para la recepción de los datos de entrada (D_{in}) y para la transmisión de los datos al resto de los LEDs a través de su pin de salida (D_{out}). Los datos se trasmiten en serie. Se deben enviar ristras de 24 bits de datos por LED, donde cada uno de los bits debe ser codificado con el timing que se puede observar en la figura 2.2.

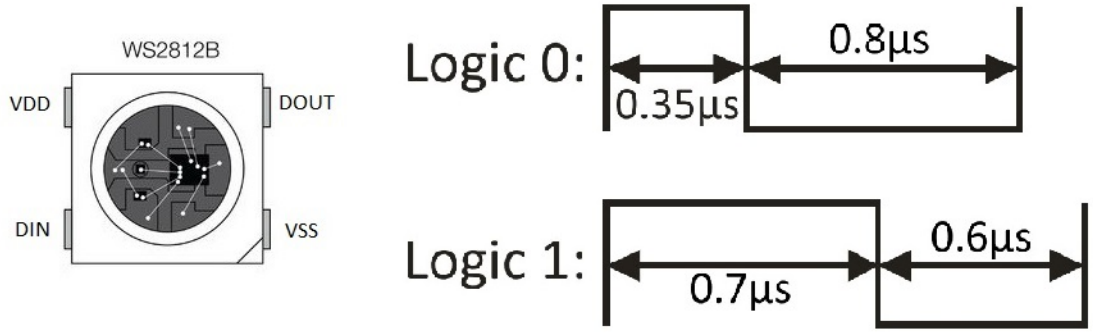


Figura 2.2: Diagrama del WS2812B y *timing* del protocolo

Como podemos ver los '1' lógicos se codificarán como un pulso en alta con una duración mayor que la de el pulso en baja dentro de un periodo de la señal, mientras que en el '0' lógico, la duración del tiempo en alta debe ser inferior a la de la duración del tiempo que la señal es en baja.

De los 24 bits que usa cada LED, se dividen en grupos de 8 bits (1 byte) que se dedican para la codificación de color: los primeros 8 bits son para el Rojo, los siguientes 8 bits para el Verde, y los últimos 8 para el Azul, permitiéndonos la posibilidad de obtener hasta 16 Millones de colores por LED, donde cada color resulta de una mezcla de las diferentes intensidades de luz que puede emitir cada uno de los LEDs. En la tabla podemos observar las ristras de bits que deben enviarse para obtener algunos de los colores básicos. Una vez un LED ha tomado la información necesaria para iluminarse, quita estos bits de la ristra de bits serie y emite el resto de los datos para los LEDs posteriores.

R	G	B	Color
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	
0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1	
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	

Figura 2.3: Ristras de datos para los LEDs RGB y color obtenido

- **Sistema Electrónico que controle la visualización en la pantalla. Conectividad inalámbrica y conexión a internet** Para el control de los LEDs, será necesario un dispositivo que sea capaz de generar el protocolo que controla el funcionamiento de los LEDs inteligentes vistos anteriormente. Para conseguirlo es evidente que necesitamos disponer de un microcontrolador, con el que seamos capaces de generar el protocolo descrito a través de alguna de sus interfaces GPIO mediante la técnica conocida como *bit-banging* y a su vez, debemos ser capaces de conectarnos de forma inalámbrica para el envío de contenido.

  Por estas razones, hemos elegido un módulo WiFi comercial denominado **ESP-12F**. Se trata de un módulo WiFi desarrollado por *Espressif Systems*,que incorpora un microcontrolador, pensado para usarse en aparatos de bajo consumo.

  En este módulo se incluye un microcontrolador que nos permite usar su interfaces típicas de comunicación: I2C, SPI, UART y GPIO, entre otras. Ademas incluye una memoria *EEPROM* para almacenar información no-volátil durante tiempo de ejecución y un módulo WiFi incorporado en su diseño, que nos permite conexiones a redes inalámbricas WiFi, implementando de una pila de protocolos TCP/IP para la comunicación. A todo esto debemos añadirle una de sus características más atractivas: su bajo precio, pues en el mercado puede conseguirse por menos de 3 euros.

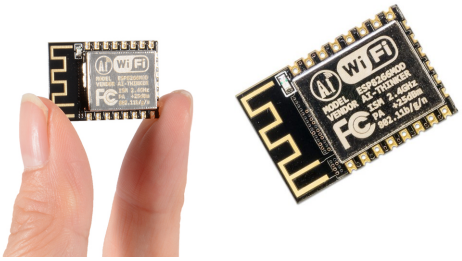


Figura 2.4: Módulo WiFi ESP-12F

- **Aplicación Android para controlar el contenido visualizado:** Para el control del contenido que se visualizará en la pantalla multi-LED, se pide desarrollar una aplicación en Android., la cual debe ser capaz de conectarse a una red WiFi que nos suministre la conexión a internet y el acceso al módulo WiFi ESP12-F. Este apartado se desarrollará más adelante cuando hablemos del desarrollo del software necesario para el producto.

Una vez que hemos identificados los principales elementos que formarán parte de nuestro proyecto, lo siguiente que nos debemos plantear, sera la infraestructura que deben seguir las comunicaciones. Para analizar el funcionamiento de la infraestructura de comunicaciones, analizaremos el siguiente diagrama:

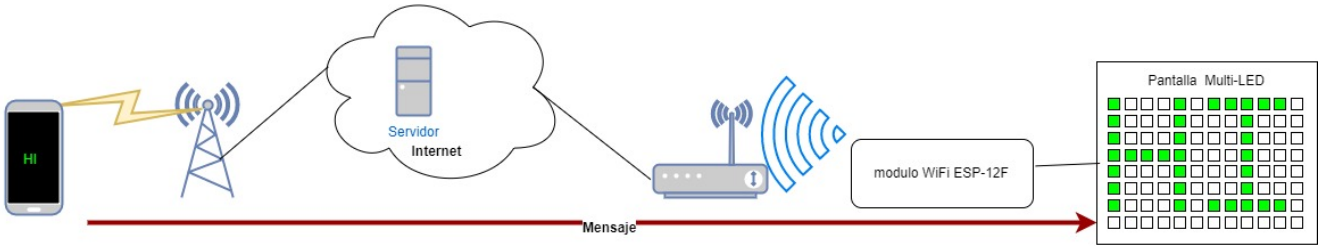


Figura 2.5: Infraestructura de la comunicación

El módulo WiFi se encargará de controlar el contenido que se envía a la pantalla multi-LED. Debe estar conectado a un punto de acceso que le suministre una conexión a Internet, con el fin de ser accesible desde cualquier parte del mundo, pues recordemos que se trata un proyecto basado en el IoT.

Con el módulo es accesible desde internet, la otra parte de la comunicación es llevada a cabo por la aplicación de control para Android que se ejecuta en nuestro smartphone. Esta aplicación de control se encargará de enviar el contenido vía internet hasta el módulo para que al final se muestre dicho contenido en la pantalla.

Una vez detalladas las necesidades, principales especificaciones del proyecto e identificado los principales objetivos y componentes del prototipo,debemos proceder primero por el desarrollo del hardware. Es necesario que el hardware se encuentre diseñado y finalizado si se desean implementar las funcionalidades software, las cuales se desarrollarán en etapas posteriores.

Capítulo 3

Desarrollo del Hardware

3.1. Captura esquemática

Durante esta fase, nos centraremos en la parte del desarrollo hardware del proyecto. Para empezar a realizar la captura esquemática, es de gran ayuda primero realizar un diagrama de bloques en el que se identifiquen los bloques funcionales que nos harán falta para el prototipo.

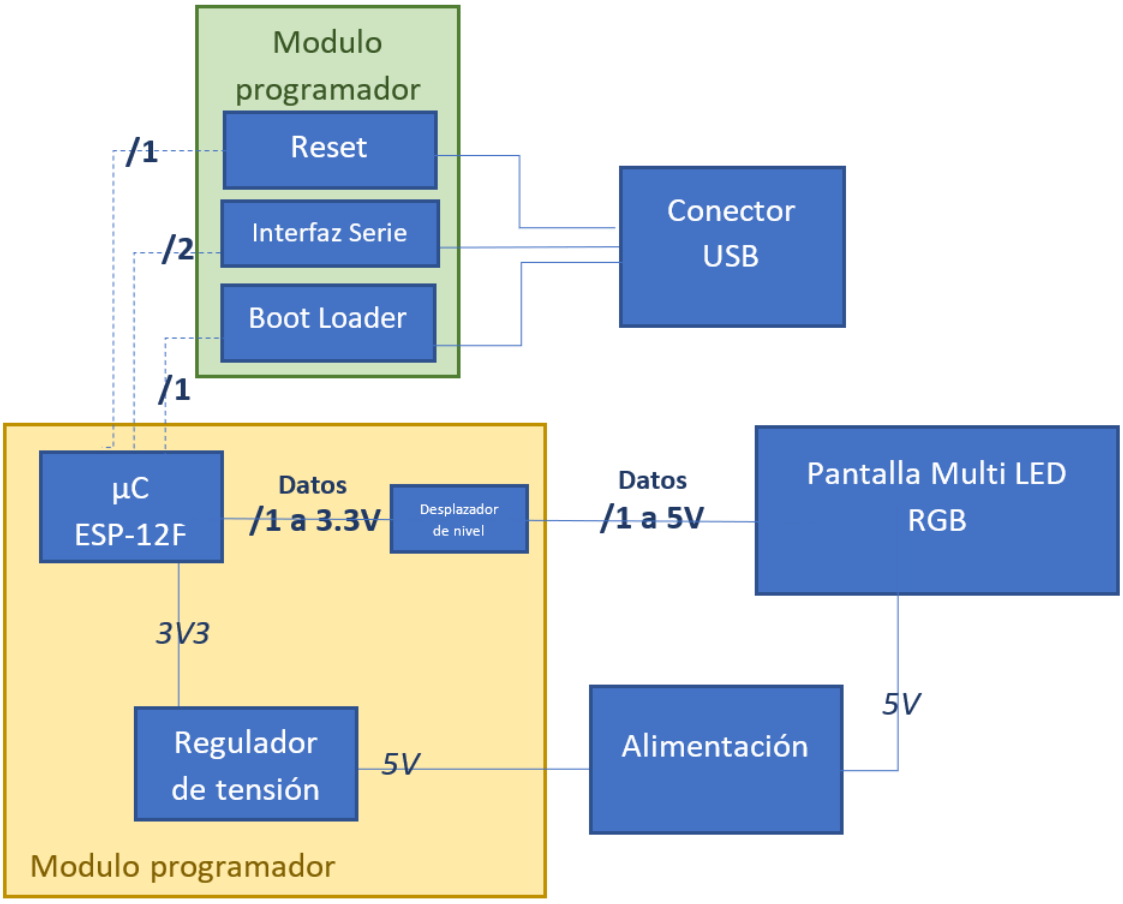


Figura 3.1: Infraestructura de la comunicación

La elaboración de este diagrama implica un pequeño análisis previo de los componentes importantes que se van a incluir y sus requerimientos funcionales para estar operativos:

- **Es evidente que nuestro sistema debe debe estar alimentado:** Analizando las hojas de datos de los dos componentes identificados hasta ahora, el **ESP-12F** y los LEDs **WS2812B**, observamos que se alimentan a 3.3V y 5V respectivamente. Esto nos permite identificar que en el hardware a desarrollar, debemos incluir una fuente de alimentación regida por los 5V que necesitan los LEDs y que para conseguir los 3.3V necesarios para el funcionamiento del módulo WiFi se debe incluir un regulador de tension.

- **Nivel de tensión de los datos del ESP-12F**: Como puede verse en los *datasheets* del módulo WiFi, los datos de salida que se van a enviar a los LEDs estarán limitados a la tension de funcionamiento del propio módulo, es decir, 3.3V. Las hojas de especificaciones de los LEDs, nos dicen que para el correcto funcionamiento de estos, las señales en alta de los datos deben ser por lo menos un 70 % de la tension de alimentación. En este caso, al estar alimentados a 5V, la señal de alta debe ser de por lo menos 3.5V, tensión que el módulo WiFi es incapaz de generar. Para solventar este problema se debe incluir en el circuito un desplazador de nivel, que convierta nuestros '1' lógicos de 3.3V a '1' lógicos de 5V.

- **Sección del circuito que sea capaz de interconectar el módulo WiFi con el PC:** se debe implementar una parte del circuito que cumpla esta condición con el fin de cargar el software a ejecutar en el microcontrolador. Esta sección, debe incluir partes que se encarguen de establecer una comunicación a través de un interfaz serie y una conexión USB con el ordenador, cargar el software de arranque (*Boot Loader*) y gestionar el reset por medio de software.

En este diagrama se propone una primera idea para el diseño del hardware, en donde se incluyen los componentes en bloque necesarios para el funcionamiento y la interconexión entre los distintos módulos que componen el sistema. Partiendo de este diagrama base, nos disponemos al desarrollo esquemático del hardware del proyecto, ayudándonos del programa *Proteus ISIS*. Basado en este análisis funcional anterior, haremos una subdivisión en pequeños circuitos modulares diferentes que, interconectados entre sí, nos brinden las funcionalidades deseadas. Serán tres sub-módulos:

- Módulo WiFI Controlador: contendrá el módulo ESP-12F, los conectores de alimentación, el regulador de tensión y el desplazador de nivel, y se encargará de las conexiones inalámbricas y el control de la pantalla multi-LED.

- Módulo Programador: Permitirá la conexión entre el PC y el módulo WiFi controlador, implementando dicha comunicación a través de un puerto serie. Será un módulo extraíble y que se conectará únicamente cuando se desee reprogramar el módulo controlador.

- Pantalla Multi-LED RGB: Mostrará el contenido a través de los LEDs WS2812. Además, incluirá conectores de alimentación y de transferencia de datos.

El esquemático del circuito incluye todos los componentes necesarios para el correcto funcionamiento de cada módulo por separado. Se incluyen las resistencias, condensadores, transistores, ferritas, etc. La mayoría estos componentes comerciales están incluidos en las librerías de componentes que trae Proteus de fábrica. Sin embargo, aquellos componentes especiales que no se encuentran en el programa, deben ser diseñados e incluidos en nuestro esquemático. Esto supone un diseño de la forma esquemática del componente y la elaboración de una huella. Ambos diseños deben guardar una correcta relación entre pines del componente y pads de la huella con el fin de evitar problemas de routing y conexiones erróneas.

Todos los componentes deben estar debidamente etiquetados e identificados. Además, debe completarse otro tipo de datos como el *precio* o la *referencia del fabricante*, todo esto, con la finalidad de generar un documento con el listado de materiales (denominado B.O.M. por sus siglas en inglés, *Bill Of Materials*). Este documento generado debe contener todos los datos necesarios para que, en un supuesto caso en el que sea una empresa la que se encarga del desarrollo del proyecto, el departamento de compras se encargue de realizar el pedido de los componentes sin mayores complicaciones.

Figura 3.2: Esquemático del Módulo WiFi Controlador

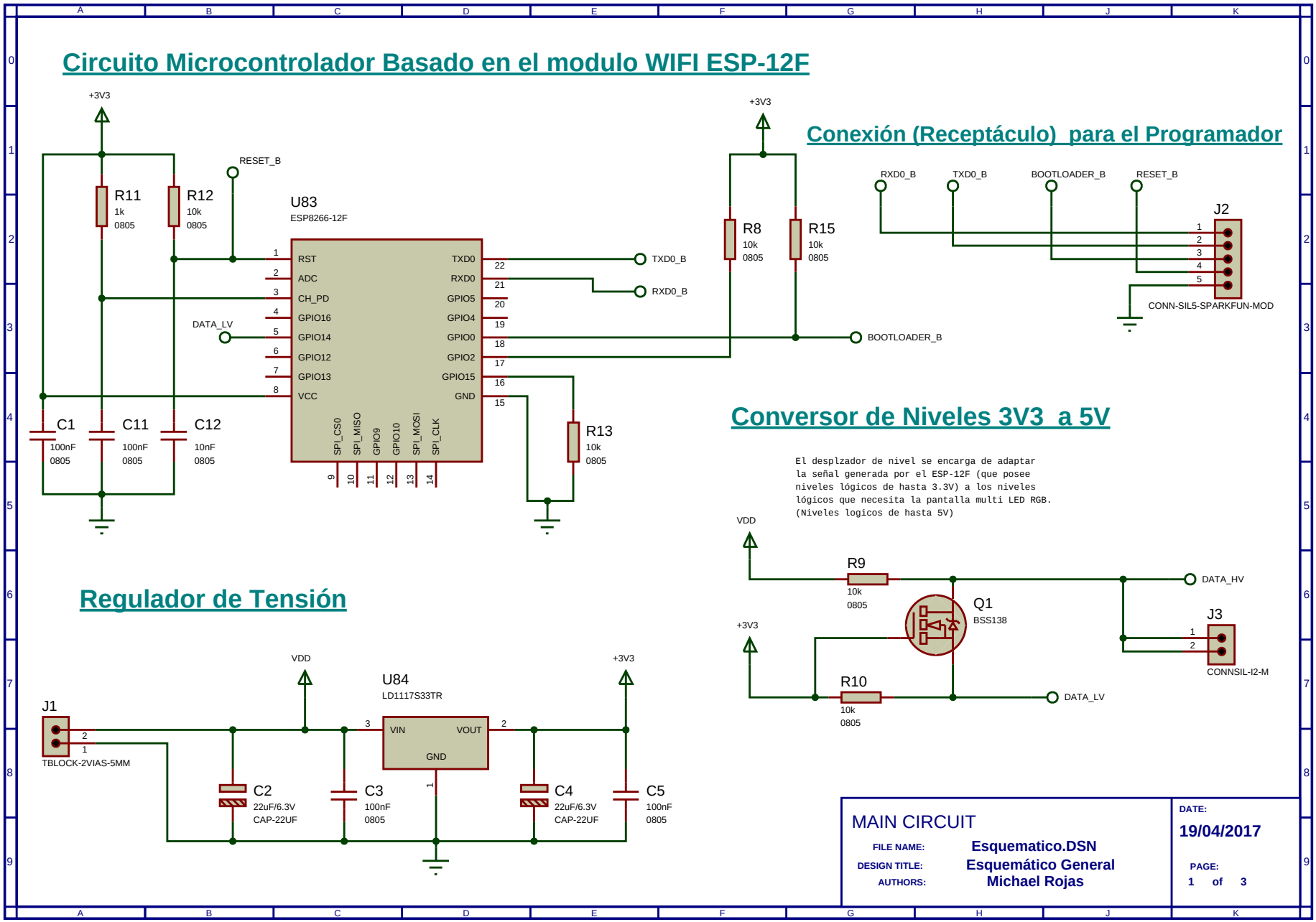


Figura 3.3: Esquemático de la Pantalla Multi-LED RGB, basada en WS1228B

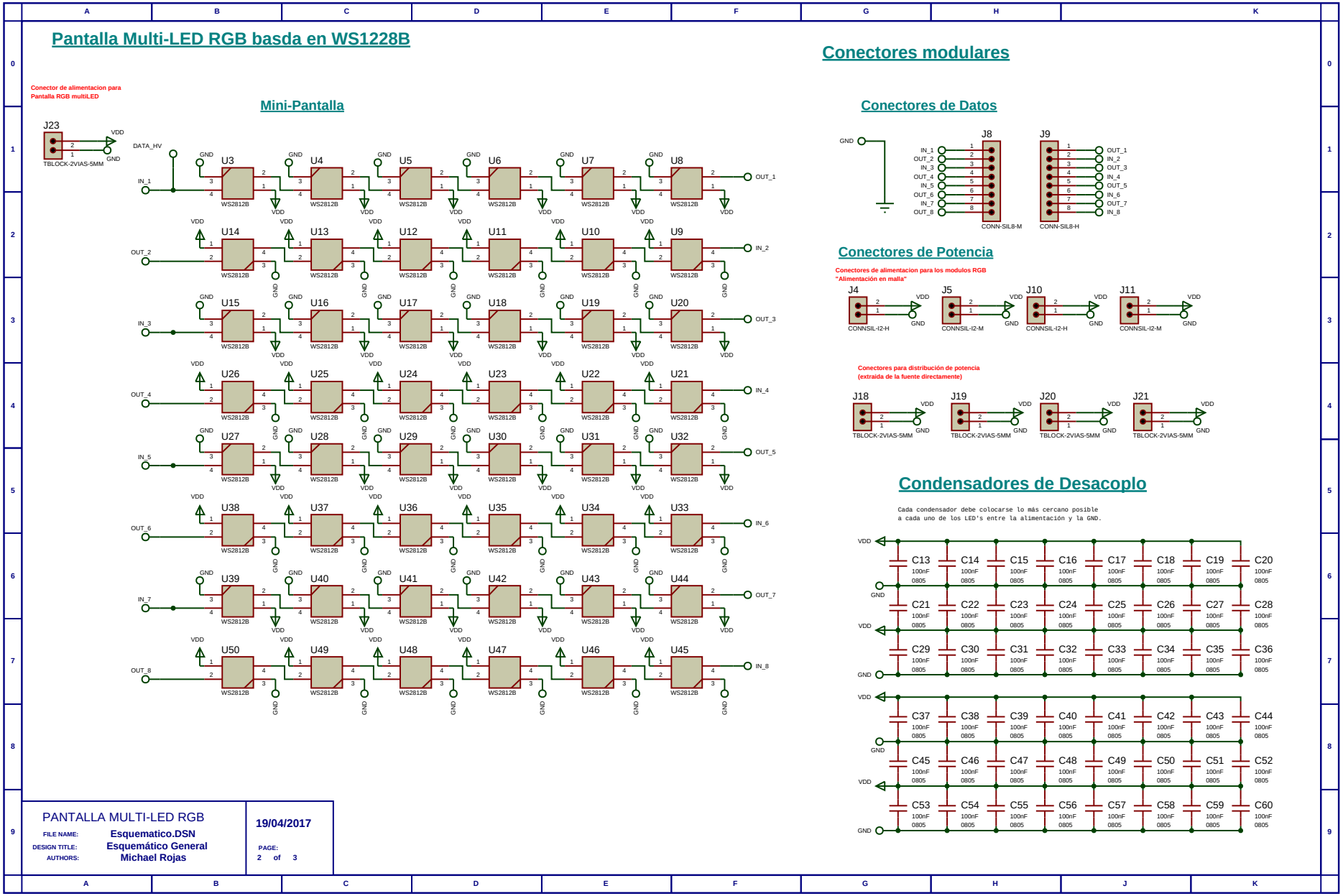
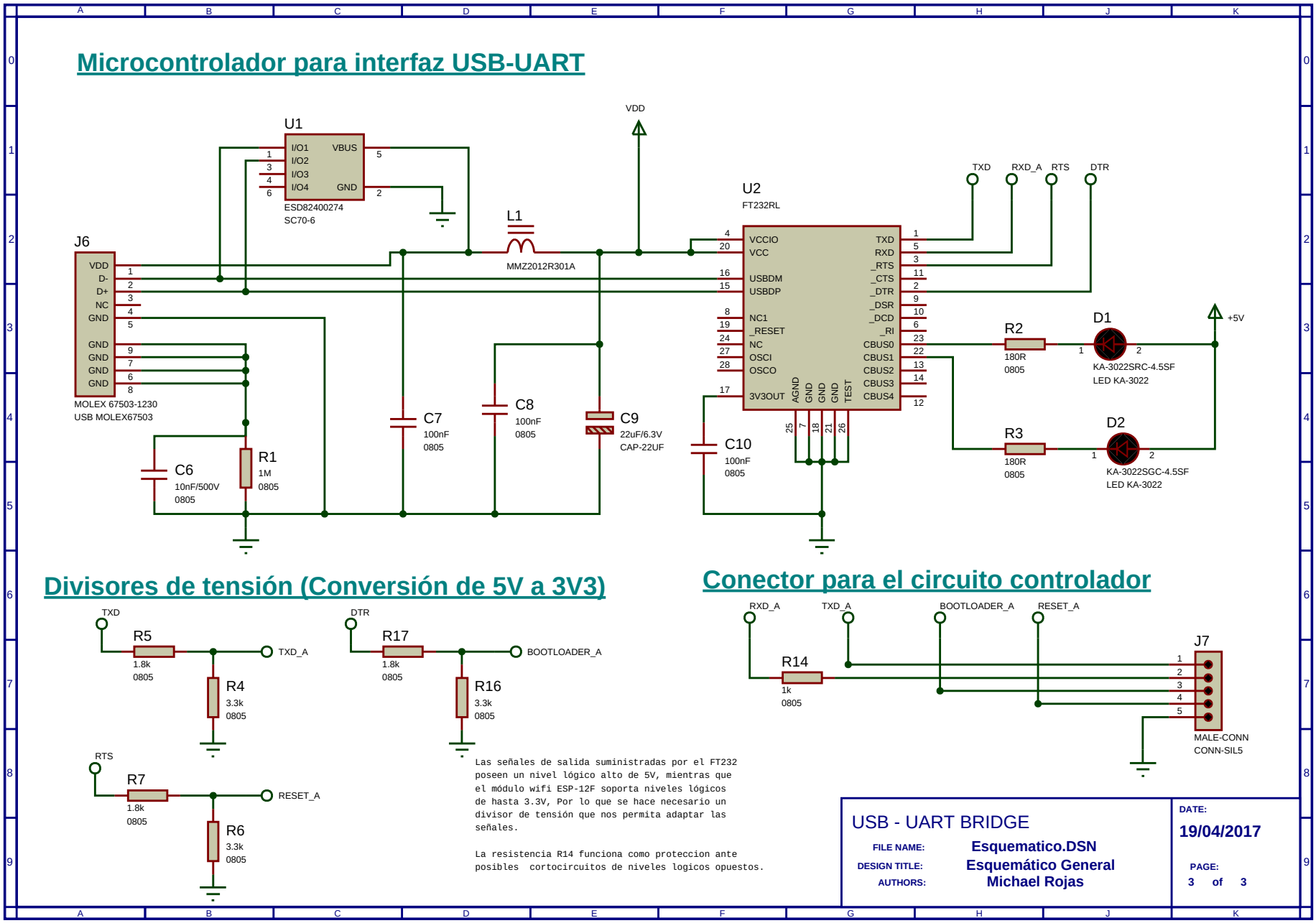


Figura 3.4: Esquemático del Circuito programador USB-Serie



Bill Of Materials For Esquemático General

Design Title : Esquemático General
Author : Michael Rojas
Revision :
Design Created : miércoles, 15 de febrero de 2017
Design Last Modified : miércoles, 19 de abril de 2017
Total Parts In Design : 149

17 Resistors

Quantity:	References	Value	Price	PF	Package
1	R1	1M	0,0276		0805
2	R2, R3	180R	0,0276		0805
3	R4, R6, R16	3.3k	0,0276		0805
3	R5, R7, R17	1.8k	0,0276		0805
6	R8-R10, R12, R13, R15	10k	0,0276		0805
2	R11, R14	1k	0,0276		0805

60 Capacitors

Quantity:	References	Value	Price	PF	Package
55	C1, C3, C5, C7, C8, C10, C11, C13-C60	100nF	0,053	1650864	0805
3	C2, C4, C9	22uF/6.3V	0,414	2346354	CAP-22UF
1	C6	10nF/500V	0,222	1284130	0805
1	C12	10nF	0,544		0805

52 Integrated Circuits

Quantity:	References	Value	Price	PF	Package
1	U1	ESD82400274	0,85	1825876	SC70-6
1	U2	FT232RL	4,27	1146032	SSOP28
48	U3-U50	WS2812B	0.38	Itead	WS2812B
1	U83	ESP8266-12F	2.60	IM151228001	ESP-12F
1	U84	LD1117S33TR	0,218	1202826	SOT223-4

1 Transistors

Quantity:	References	Value	Price	PF	Package
1	Q1	BSS138	0.34	1843693	SOT23

2 Diodes

Quantity:	References	Value	Price	PF	Package
1	D1	KA-3022SRC-4.5SF	0,552	1142617	LED KA-3022
1	D2	KA-3022SGC-4.5SF	0,309	1142615	LED KA-3022

Figura 3.5: Billl of materials (1)

Figura 3.6: Billl of materials (2)

17 Miscellaneous

Quantity:	References	Value	Price	PF	Package
6	J1, J18-J21, J23	TBLOCK-2VIAS-5MM	0,394	9632972	TBLOCK-2VIAS-5MM
1	J2	CONN-SIL5-SPARKFUN-M	0	Modified Holes	CONN-SIL5-SPARKFUN-MOD
3	J3, J5, J11	CONNSIL-I2-M	0.025	1593411	CONN-SIL2
2	J4, J10	CONNSIL-I2-H	0,0971	1593458	CONN-SIL2
1	J6	MOLEX 67503-1230	0,862	2313554	USB MOLEX67503
1	J7	MALE-CONN	0.0939	1593414	CONN-SIL5
1	J8	CONN-SIL8-M	0,33	1593416	CONN-SIL8
1	J9	CONN-SIL8-H	0,888	1593463	CONN-SIL8
1	L1	FERRITE LEAD	0,0819	1669724	MMZ2012R301A

martes, 23 de mayo de 2017 21:31:19

Seleccionados los componentes que formaran parte de nuestro esquemático, procedemos a explicar brevemente cada uno de los circuitos:

3.1.1. Circuito del Microcontrolador ESP12-F.

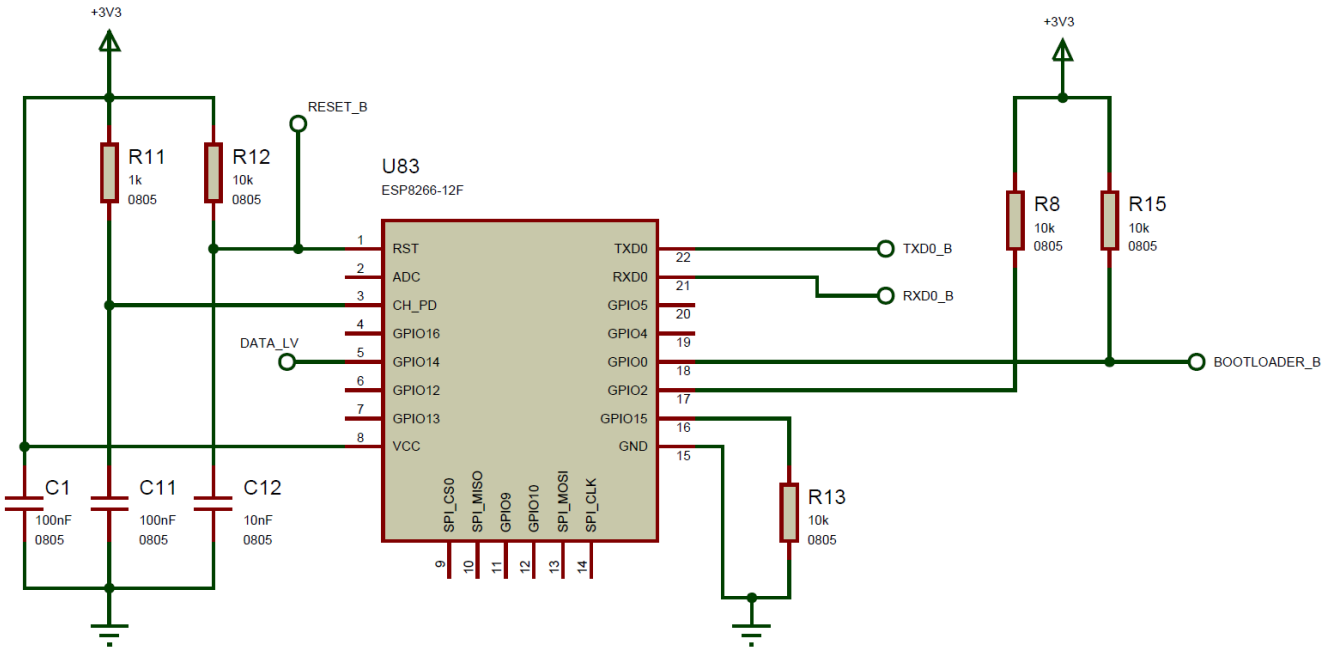


Figura 3.7: Circuito del Controlador WiFi

Se trata del circuito que habilita el microcontrolador para su funcionamiento. El condensador C1, hace la función de un condensador de desacoplo para el pin por el que se suministra la alimentación. El conjunto de la resistencia R12 y C12 suministra una constante de tiempo de carga rápida para que el reset del microcontrolador se active cuando se conecte la entrada a tierra. En este caso, se ha sustituido el interruptor que viene siempre asociado a los circuitos de RESET por un RESET controlado mediante software, conectado a través del conversor USB-serie FT232RL.

El conjunto de la resistencia R11 y condensador C11, da una entrada en alta para el pin de ENABLE del microcontrolador, habilitando su funcionamiento. Según la documentacion del ESP-12F, para habilitar el inicio del módulo, se debe sumnisitrar la siguiente combinación de estados en los pines que se muestran:

- GPIO 0: debe estar en ALTA durante el arranque.
- GPIO 2: debe estar en ALTA durante el arranque.
- GPIO 15: debe estar en LOW durante el arranque.

Por estas tres condiciones se necesitan las respectivas resistencias de pull-up y pull-down R8, R15 y R13. El resto de conexiones son necesarias para la comunicación serie (como es el caso de los pines RX- TXD) y para la transmisión de datos, la cual se realizará a través de la conexión GPIO 14.

3.1.2. Circuito regulador de tensión

Dado que los LEDs necesitan una alimentación de 5V y que ésta será la mayor tensión que se necesite en todo el montaje, será la que nos condicione la elección de una fuente de alimentación adecuada y de la que se extraerán las tensiones inferiores que se necesiten. Es por esto que se hace necesario un regulador de tensión para obtener una tensión de 3.3V, pues es la tensión a la que funciona el microcontrolador ESP-12F. El regulador elegido, es capaz de darnos una tensión estable de 3.3V, posee un bajo *drop out* (1V típicamente) por lo que funcionaría con hasta 4V de entrada mínima y es capaz de suministrar hasta 900 mA, corriente más que suficiente para la alimentación del módulo WiFi.

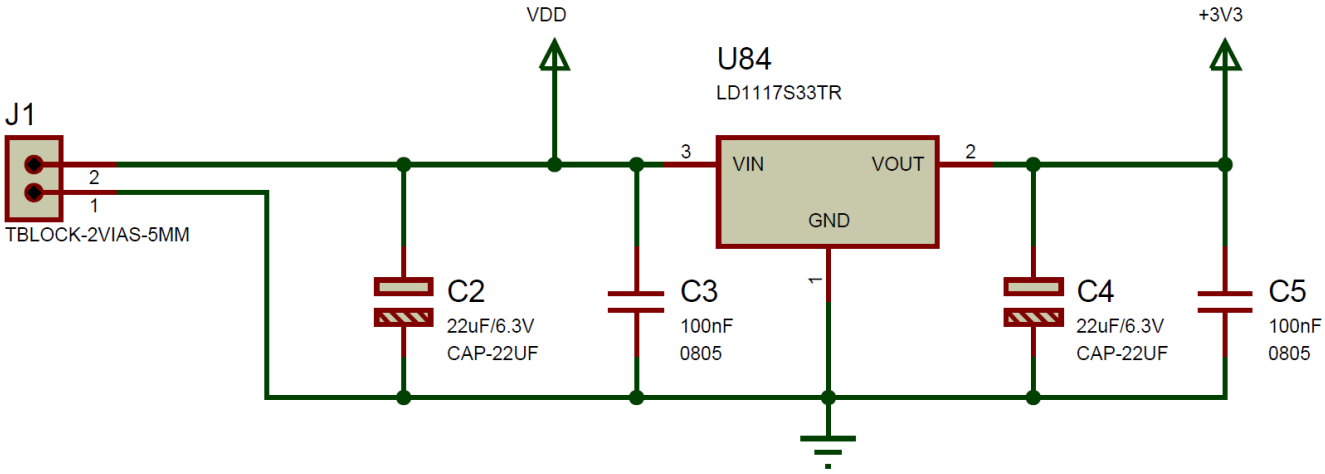


Figura 3.8: Circuito regulador de tensión

3.1.3. Circuito desplazador de nivel

La salida de los datos desde el módulo tiene una tensión de salida en alta de 3.3V como máximo (pues es la tensión a la que se alimenta). Esto supone un problema a la hora de generar el protocolo para los LEDs, pues, al estar alimentados a 5V, el nivel de tensión de para detectar un nivel alto es superior. Necesitaremos entonces desplazar nuestros “1” de 3.3V a “1” lógicos de 5V. Esta configuración con un diodo, nos permite “reconstruir” la señal con estos requerimientos.

Su funcionamiento es simple: La puerta del transistor esta conectada a 3.3V (el 1 lógico de *"baja tensión"*). Cuando se introduce un 1 lógico en la fuente (entrada etiquetada como DATA_LV y de donde los datos vienen desde el módulo ESP-12F) , la fuente y la puerta tendrán la misma tensión, por lo

que el transistor no conducirá y se obtiene a la salida, etiquetada como DATA_HV la tension de salida VDD=5V, es decir, un '1' lógico adecuado para los LEDs.

Cuando lo que se tiene en DATA_LV, es un 0 lógico, la tensión entre puerta y fuente permite al transistor ponerse en conducción, conectando directamente DATA_LV y DATA_HV, transmitiendo el '0' lógico a la salida.

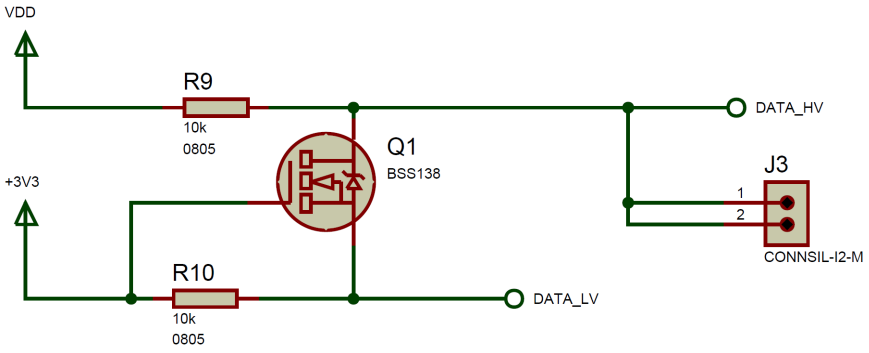


Figura 3.9: Circuito desplazador de nivel

3.1.4. Circuito microcontrolador para interfaz USB-Serie

Este circuito es el encargado de convertir las comunicaciones desde la interfaz USB que viene desde el ordenador a través del conector mini-USB, en una comunicación serie capaz de conectarnos con la UART del ESP-12F, todo gracias a ese microcontrolador intermedio denominado FT232RL. Este circuito es uno de los circuitos de aplicación disponibles en los datasheets del FT232RL. Incluye además todos los elementos necesarios para la protección frente descargas electroestáticas (componente U1, filtro C6-R1), acoplamientos electromagnéticos no deseados (ferrita L1) y condensadores de desacoplo. También incluye dos LEDs que nos permiten monitorizar visualmente que se está produciendo una comunicación serie.

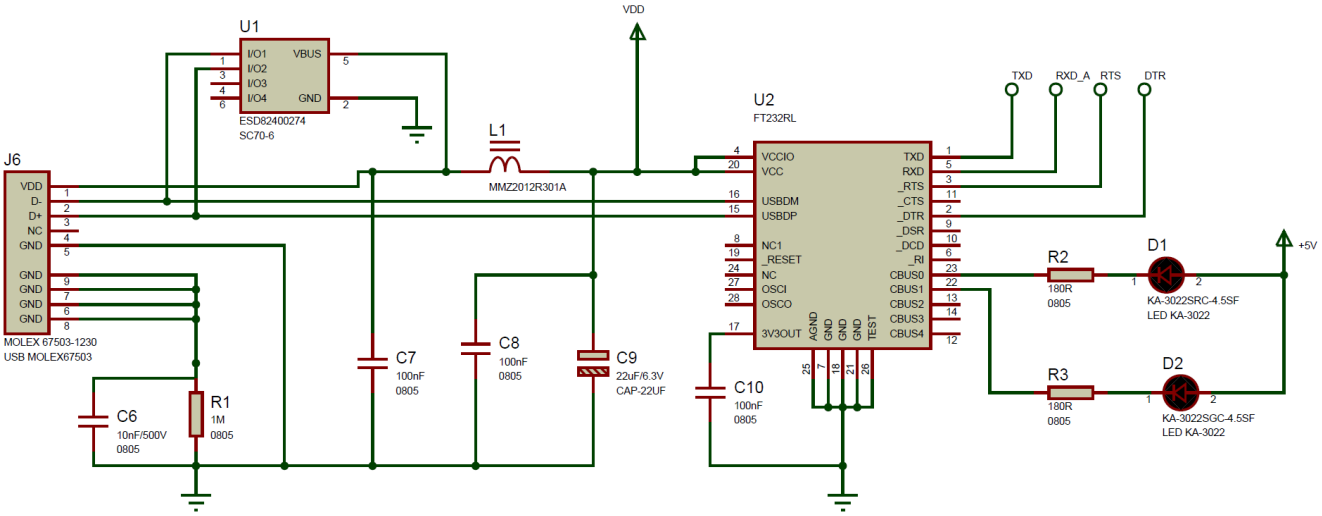


Figura 3.10: Circuito para comunicación USB-serie

Como el FT232RL debe comunicarse directamente con el ESP-12F,y dado que los voltajes de entrada que soporta este último son inferiores que los valores de salida de conversor USB-Serie, se necesitan divisores de tensión con el fin de realizar la conversión de los datos desde niveles de alta de 5V hacia los 3.3V de nivel alto que soporta el ESP-12F.

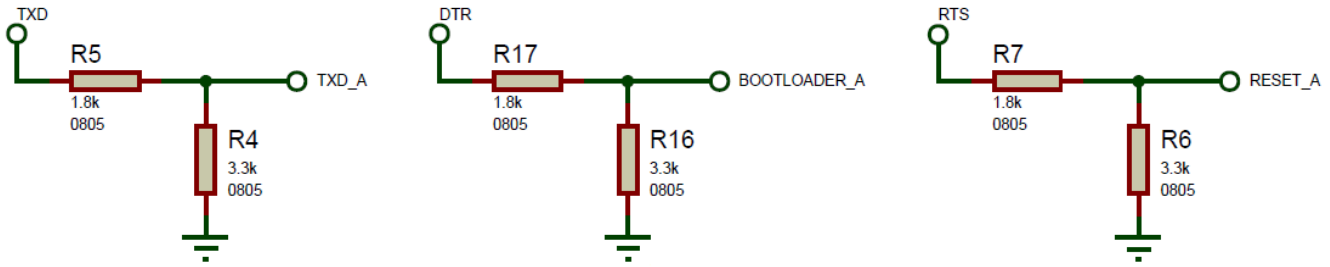


Figura 3.11: Divisores de tension: Conversión de 5V a 3.3V

3.1.5. Circuito de la pantalla Multi-LED RGB

Cada uno de los LEDs, posee 4 pines: Uno para la alimentación, otro para la conexión a GND, otro para los datos de entrada y uno para los datos de salida. Se deben colocar de tal forma que la salida de datos del LED anterior vaya a la entrada de datos del LED siguiente. Además, se necesita un condensador de desacoplo para cada uno. Abordaremos la distribución elegida para los LEDs en relación con su escalabilidad en el capítulo siguiente.

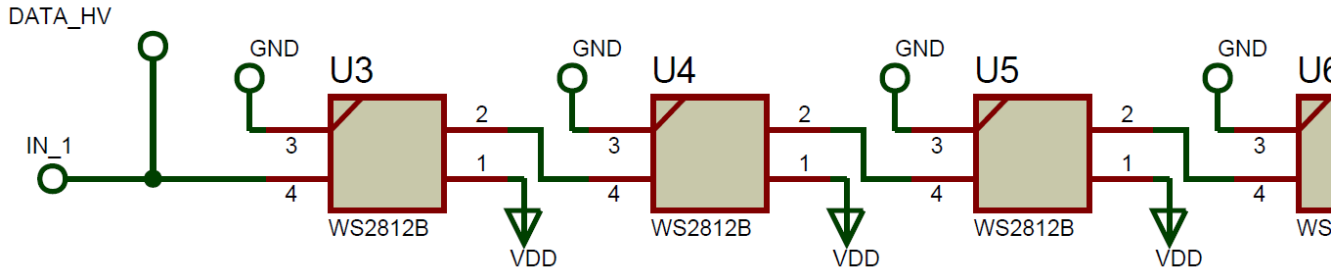


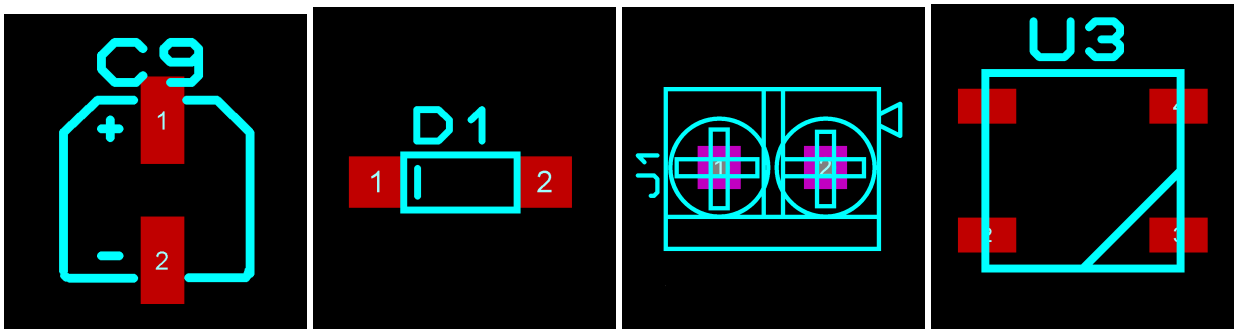
Figura 3.12: Circuito de la pantalla Multi-LED: Interconexión de los LEDs

3.2. Diseño de la placa de circuito impreso (PCB)

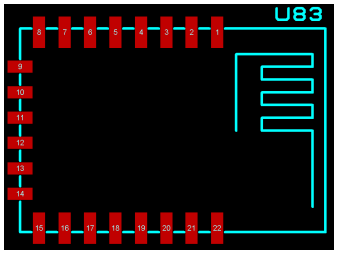
Una vez el diseño esquemático está hecho y revisado, pasamos a el diseño de la placa de circuito impreso, tarea para la cual nos ayudaremos del módulo *ARES* del programa *Proteus*, software dedicado al diseño de PCBs.

Para empezar, se debe verificar que todos los componentes tienen la huella correcta. Se le denomina huella en este ámbito, al conjunto de marcas y pads (que son superficies metálicas que interconectan pines de componentes y pistas), que se colocan sobre una placa de circuito para que un componente encaje perfectamente sus pines con el trazado del circuito, conectándolo con el resto de componentes del diseño. Esta huella debe tener los pads exactos para sus pines, que varían en función del componente, y se deben ajustar a las medidas de los componentes de la forma más exacta posible, pues en caso contrario, no resultaría posible colocar los componentes sobre la placa de circuito impreso.

Los componentes con fabricación comercial típica, como son las resistencias, condensadores, reguladores de tensión,etc. tienen huellas con medidas estandarizadas (estas medida son denominadas de *métrica 0805*) y que por tanto están incluidos dentro de las librerías que trae el software ARES por defecto. Para el resto de componente no habituales, debemos encargarnos de diseñar las huellas. Este es un proceso que debe llevarse a cabo con la ayuda de las herramientas de dibujo del propio ARES, basándonos en los datos mecánicos de los componentes, los cuales vienen típicamente descritos en las hojas de datos de cada uno de ellos. Algunas de las huellas que se han diseñado para el proyecto son:



(a) Condensador (b) Diodo (c) Alimentación (d) LED WS2812B



(e) ESP-12F

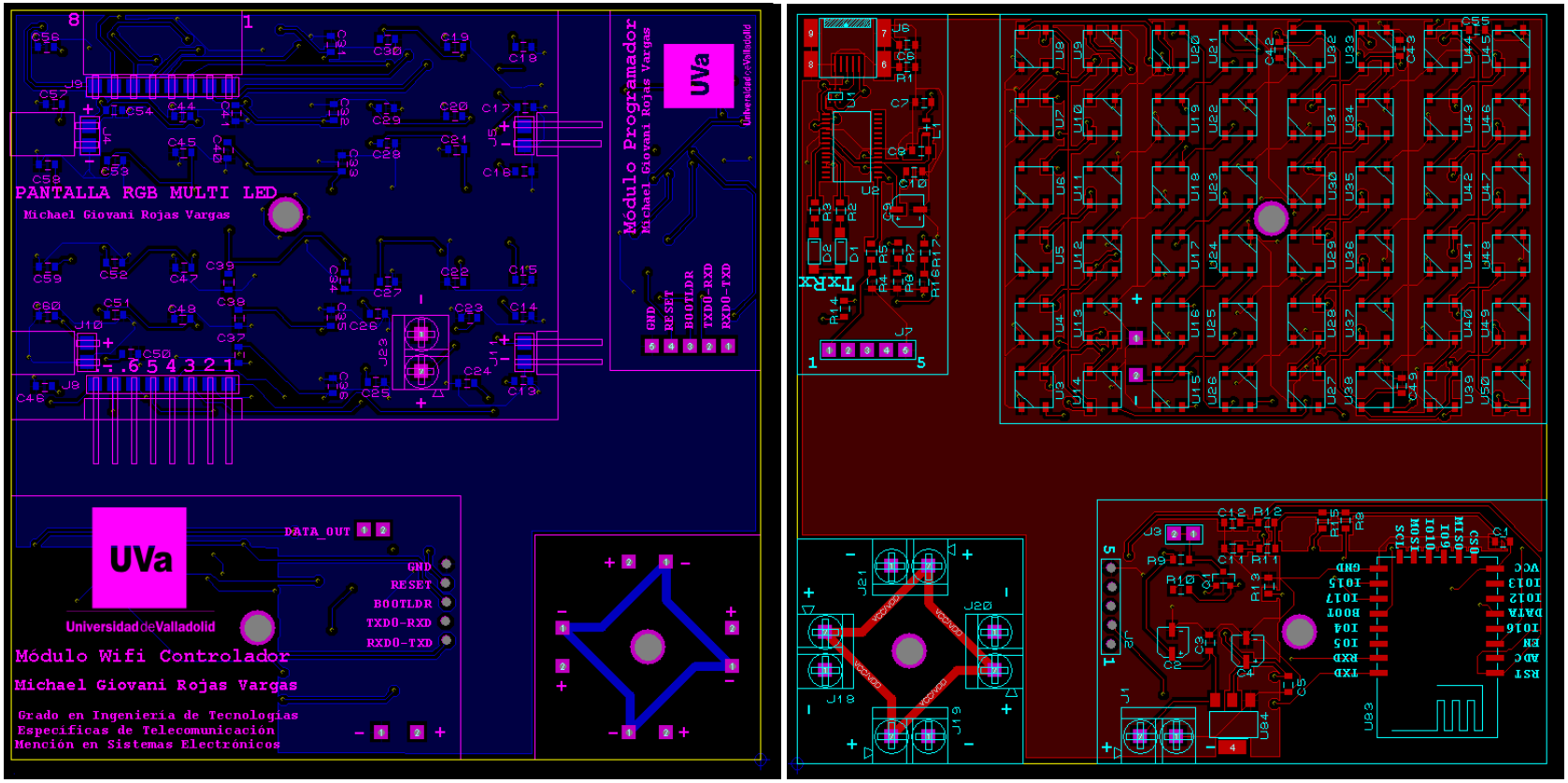
Figura 3.13: Huellas diseñadas para el layout

Con todas las huellas diseñadas y asignadas correctamente, procedemos con la distribución de los componentes en el espacio de la placa a fabricar. Dado que recurriremos a una empresa de fabricación externa, debemos pensar en el diseño ajustándonos a los tamaños de placa que se ofrecen. Se ha elegido un conjunto de 10 placas de un área de 10cm x 10cm.

Durante el diseño de la PCB siempre ha de tenerse en cuenta y cumplir en la medida de lo posible las reglas para un buen diseño de circuitos PCB:

- Los conectores deben estar siempre lo más cerca posible del borde, asegurando que son fácilmente accesibles. En este caso para que las pantallas encajaran a la hora de aplicar la escalabilidad del diseño, era fundamental que los conectores estuvieran correctamente alineados.
- Los condensadores de desacoplo deben estar lo más próximo a el elemento que deben proteger.
- Todas las pistas deben ser lo más cortas posibles. Deben evitarse los ángulos rectos durante el trazado de las mismas, pues esto ocasiona perdida en la trasmisión de datos.
- Debe incluirse uno o varios planos de masa que asegure el mejor camino de retorno para las corrientes.
- Un buen serigrafiado de los componentes es también una buena práctica para nuestro circuito, sobre todo para hacernos una idea de donde esta ubicado cada componente dentro del circuito e identificarlos fácilmente, ya sea para el montaje o para el mantenimiento de componentes que pudieran resultar estropeados.

El diseño final de la placa de circuito impreso cuando se han colocado todos los componentes se muestra a continuación:



(a) cara trasera

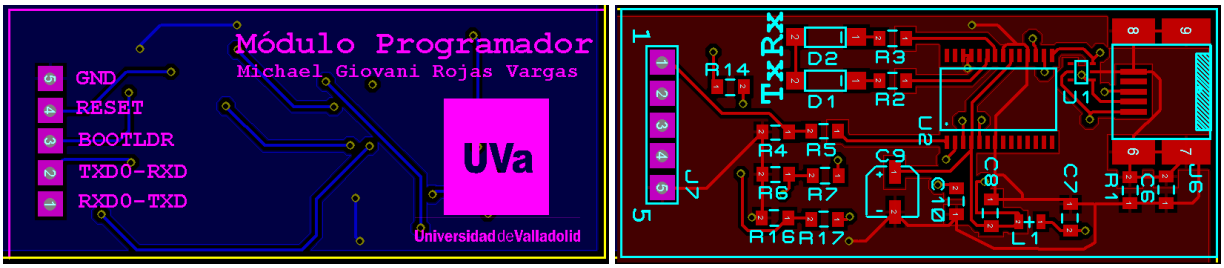
(b) cara frontal

Figura 3.14: Layout Completo de la PCB

Para comprender la distribución de los componentes, debemos tener en cuenta que el módulo programador (el cual incluye el conversor USB-serie), no estará incorporado en el sistema, sino que se conectará al módulo WiFi controlador únicamente cuando se quiera reprogramar el microcontrolador ESP-12F desde el ordenador. A su vez, el microcontrolador se conectará a las pantallas multi-LED únicamente a través de la línea de datos y dicha conexión debe hacerse con un cable, debido al diseño modular de las pantallas. A continuación, se explica detalladamente cada uno de los sub-módulos:

3.2.1. Módulo programador

Esta pequeña tarjeta, incluye todos los componentes necesarios para realizar la conversión desde la interfaz USB hacia la interfaz serie que nos permite comunicarnos con el módulo ESP-12F, Los pines de conexión se soldarán a un conector macho para encajar con la otra parte en el módulo del microcontrolador durante la programación.

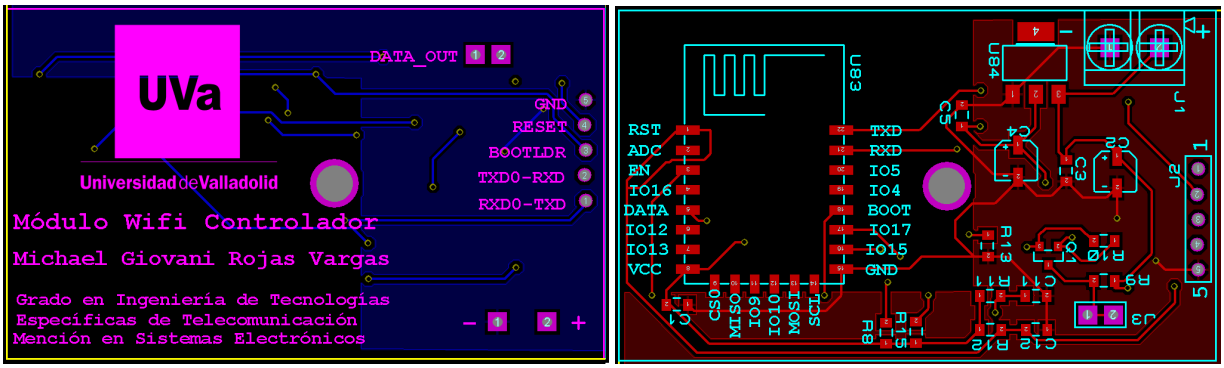


(a) cara trasera (b) cara frontal

Figura 3.15: Layout del módulo programador

3.2.2. Módulo WiFi Controlador

Este módulo incluye el microcontrolador ESP-12F y los circuitos que se encargan de habilitar su funcionamiento: en él se implementan los pull-ups y los pull-downs necesarios para su arranque. Incluye también el regulador de tensión (U84) que obtiene los 3.3V que necesita el ESP-12F, a partir de los 5V que nos suministra el conector de alimentación (J1). También se incluye el conector J2, el cual será un conector tipo hembra para recibir el Módulo programador visto anteriormente y el conector de salida de los datos que se enviaran hacia la pantalla multi-LED.

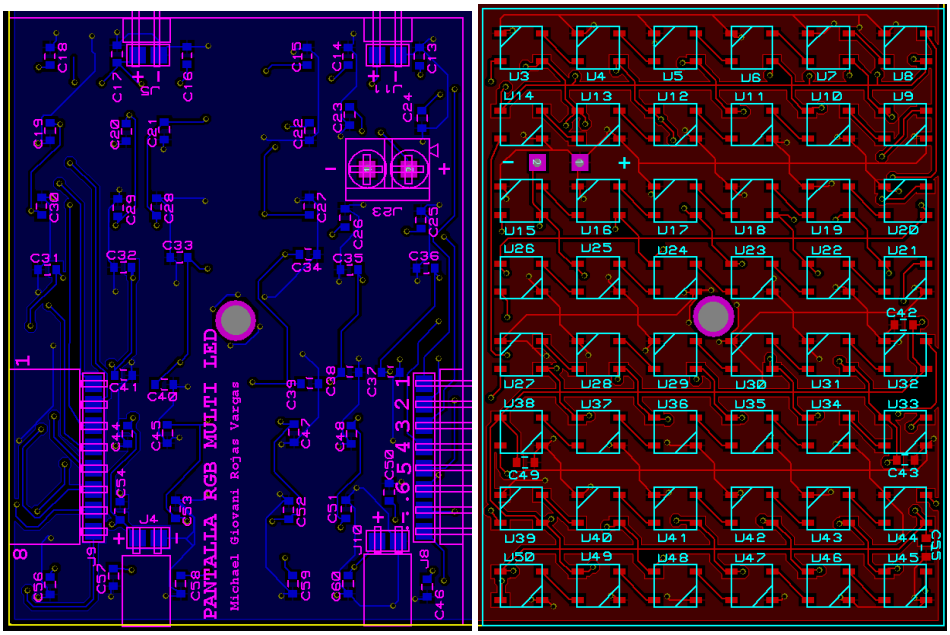


(a) cara trasera (b) cara frontal

Figura 3.16: Layout del módulo WiFi Controlador

3.2.3. Pantalla Multi-LED RGB

Uno de los requisitos para la pantalla multi-LED, es que su diseño sea escalable. Al tratarse de una pantalla que va a mostrar contenido, los LEDs deben encontrarse siempre distribuidos de la misma forma regular, sea cual sea la dimension de la pantalla. Se implementó en el diseño una matriz de 6x8 LEDs, entre cada uno de los cuales habría una separación de 4mm a excepción de los que están más próximos al borde de la pantalla, los cuales contarían con una separación de 2mm hasta el borde. En caso de que se coloque una pantalla adyacente, ambas separaciones sumarian la separación regular de 4 mm y la distribución de la matriz conformada seguiría siendo regular, aunque ahora con una dimensión de 12x8 LEDs.



(a) cara trasera (b) cara frontal

Figura 3.17: Layout de la pantalla Multi-LED RGB

Cada una de estas 'mini-pantallas' incorporará las conexiones necesarias de alimentación y datos que permitirán unirlas entre ellas como si se tratará de un puzle. Esto afectaría entonces a la manera en que conectan los LEDS y sus lineas de datos internamente dentro del diseño de la PCB. El flujo de datos seria como se ve en la imagen 3.18 donde podemos ver que los datos recorren cada fila de LEDs en forma de zig-zag. En caso de una única pantalla, el flujo de datos que se seguiría es el marcado por las flechas sólidas. Al agregar una segunda pantalla, el flujo de datos se hace en toda la fila concatenada, siguiendo el camino marcado por el conjunto de flechas del mismo color. En todos los casos sería necesario un conector terminal (jumper entre las entradas y salidas de datos) para permitir que los datos viajen de vuelta desde el final de las pantallas.

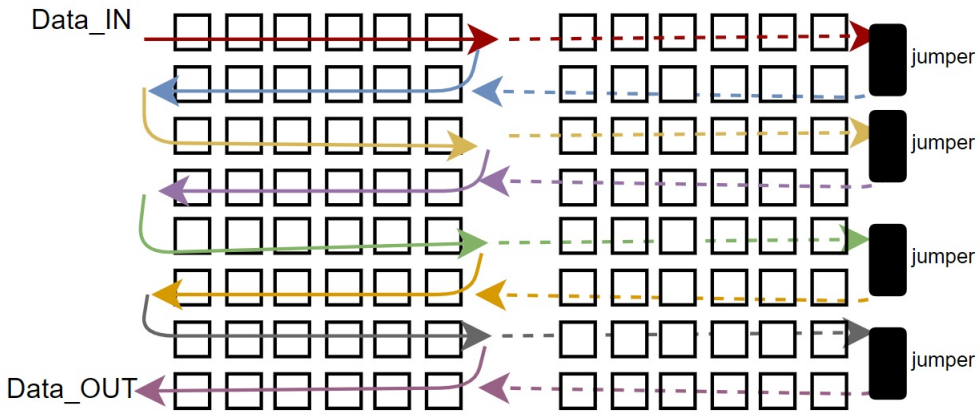


Figura 3.18: Flujo de los datos

La alimentación se suministra individualmente a cada pantalla, pero también se puede distribuir a través de los conectores J4, J5,J10 y J11, de una manera similar a como se suministran los datos. Esto permitiría que la alimentación estuviera distribuida y mallada, evitando así posibles caídas de tensión por perdidas en el suministro a través de las pistas.

Es importante detenernos en este punto para hacer una estimación del consumo que pueden tener estas pantallas. Según las hojas de especificaciones cada uno de los LEDs puede tener un consumo de aproximadamente 60 mA cuando se ilumina con la máxima intensidad de blanco (los tres LEDs RGB encendidos.

Estando alimentados a 5V nos supone un consumo de 0,3W por LED, un total de 14.4W para una matriz de 48 LEDs. Este es el consumo estimado en un caso extremo, pues nuestra aplicación no va a estar encendida todo el tiempo en blanco y tampoco a la máxima intensidad, pero será un detalle a tener en cuenta al momento de elegir una fuente de alimentación para el prototipo. El tamaño de la pantalla que se quiera montar también será un parámetro determinante, pues será necesario alimentar en más de un punto del circuito en función de la cantidad de LEDs que se tenga, por lo que la escalabilidad también afecta a las fuentes de alimentación que se necesitan.

Cada uno de los LEDs lleva su correspondiente condensador de desacoplo. Estos condensadores deben colocarse lo más cerca posible del pin de alimentación y de GND del mismo, así que, en su mayoría, se han colocado en parte trasera de la pantalla justo entre estos dos pines. Por motivos de diseño, para dejar espacio para los soportes donde se colocarían las pantallas o para los conectores, en determinados casos el condensador está más alejado de su LED.

En cada mini-pantalla se incluyen nada menos que 48 LEDs y 48 condensadores, además de los conectores de alimentación y datos.

3.2.4. Estimación del consumo eléctrico

Para escoger el tipo de fuente de alimentación que se debería incluir en nuestro prototipo, debemos tener en cuenta todo el consumo que pueden llegar a efectuar los componentes en conjunto. En este TFG, por cuestiones económicas solo se han adquirido 100 LEDs WS2812B, por lo que solo podremos montar dos pantallas de 6x8 LEDs, aunque suficiente para mostrar su escalabilidad.

El consumo sobre estimado de los 96 LEDs, será de aproximadamente 6A alimentados a 5V, lo cual nos da un total de 30W. De la misma forma, el consumo del ESP-12F es de aproximadamente 170mA cuando está funcionando a máxima capacidad de transmisión. Podemos ver que se trata de un consumo despreciable frente al consumo del conjunto de los LEDs, así que, con suministrar una alimentación adecuada para estos, podremos alimentar todo el sistema.

Comercialmente se distribuyen fuentes de alimentación que nos proporcionan la potencia suficiente, como es el caso de la familia de fuentes 'TXL series' de Farnell. Podemos encontrar fuentes que nos suministran desde 15W hasta los 100W.

Una vez tenemos el Layout con todos los componentes situados, debidamente enrutado y serigrafiado, procedemos a la generación de los archivos Gerber/Excellon. Estos archivos son los que se enviarán al fabricante de PCB, ya que contienen la información necesaria para la fabricación de la placa de circuito impreso.

3.2.5. Fabricación PCB

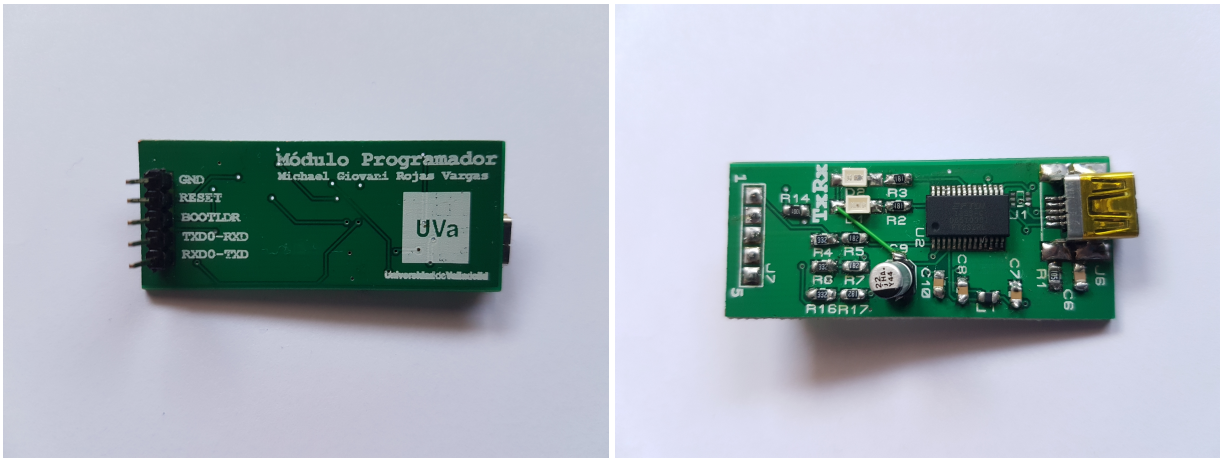
Para la fabricación de nuestro diseño PCB, hemos recurrido a la empresa 'ITEAD Studio', cuyas instalaciones se encuentran en Shenzhen, China. El envío de los archivos al fabricante se ha realizado el día 26 de abril y se recibió el pedido el 23 de mayo. Durante este mes que ha tardado la fabricación, se ha trabajado de manera paralela en el desarrollo del software para nuestro prototipo.

3.3. Montaje de Componentes

Recibidas las PCBs, procedemos al montaje de los componentes. Para el proceso de montaje del prototipo hacemos uso del laboratorio de proyectos que disponemos por parte de la Universidad de Valladolid. En este laboratorio disponemos de soldadores de precisión, microscopios digitales y pantallas para ayudarnos en el montaje de los componentes más pequeños. A la hora de posicionar cada componente, debemos asegurarnos que las patillas de los componentes quedan bien sujetas a los pads y que realmente hacen contacto. Es de gran ayuda tener a mano los esquemáticos y el layout generados para su fabricación, con el fin de comprobar la ubicación correcta de los componentes.

3.3.1. Montaje del módulo programador

Primero realizamos el montaje de los componentes del módulo programador. Es importante colocar primero los componentes más pequeños y con menos pines. Al evitar colocar los componentes grandes primero, podemos hacer la pieza más manejable a la hora de soldar. Empezamos con componentes pequeños como pueden ser las resistencias y condensadores de métrica 0805. Luego soldamos el FT232RL, que, aunque su tamaño no sea voluminoso, la gran cantidad de pines que tiene hace que su montaje no sea tan simple. Dejamos para el final el conector mini USB y los pines para la comunicación serie con el módulo WiFi. Su gran tamaño hace que sea necesario una soldadura con más cantidad de estaño, para que se sujeten muy bien a la placa de circuito. Esto es necesario, ya que deben soportar distintas fuerzas mecánicas a la hora de interconectarse con los otros módulos y/o cables.

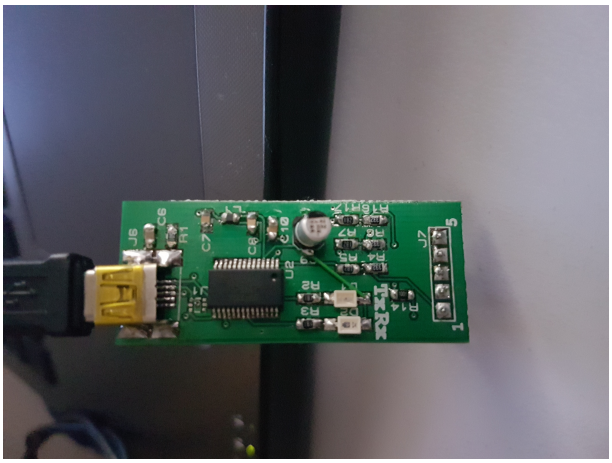


(a) cara trasera (b) cara frontal

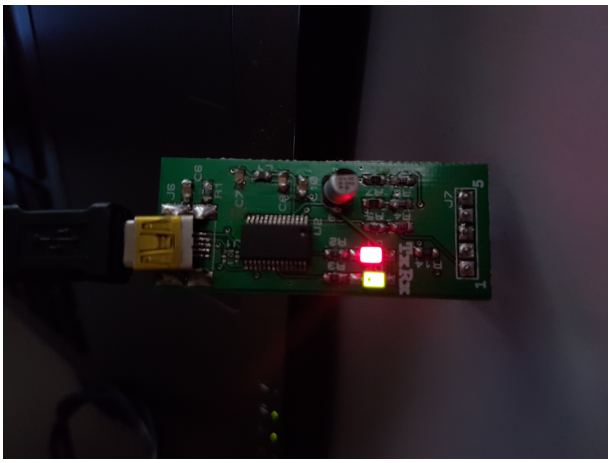
Figura 3.19: Módulo Programador

Podemos comprobar que el diseño y montaje esté hecho de manera correcta conectando la interfaz USB al módulo programador USB-Serie. El FT232RL será reconocido por el ordenador y este

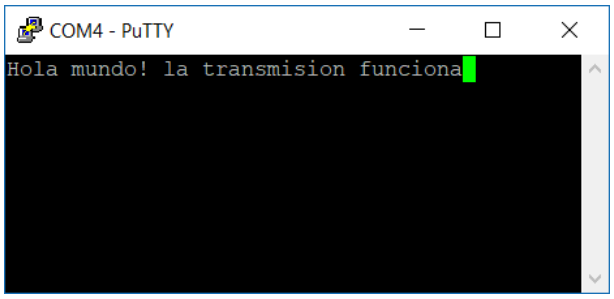
le asignará un puerto serie, que podemos visualizar con la ayuda de programas como PuTTY(es un cliente SSH, Telnet, etc. Y entre los que se incluye soporte para conexiones de puerto serie local). Para visualizar la trasferencia de datos, debemos interconectar las patillas RxD y TxD de nuestro módulo. Si al introducir texto obtenemos un "eco" del mismo, significa que la comunicación USB-Serie se está realizando de manera correcta. Además, podemos observar cómo se iluminan los leds de transmisión y recepción a la vez, como testigos de la comunicación.



(a) en standby



(b) envío/recepción de datos



(c) eco de los datos recbidios

Figura 3.20: pruebas de funcionamiento del módulo programador

3.3.2. Montaje del módulo WiFi

Nos disponemos a soldar los componentes del módulo WiFi controlador, que incluye el microcontrolador ESP-12F. Seguimos el mismo criterio para montar los componentes. En este caso, nuestro componente estrella es el propio módulo WiFi, cuyos pines resultan un poco complicados para soldar debido a que se encuentran solo en la parte inferior del cuerpo del dispositivo. También se realiza el montaje del regulador de tensión y del desplazador de nivel, además de todos los demás componentes vistos en el esquemático del módulo.

Para comprobar que el montaje está bien, basta con alimentar la placa a 5V. El módulo ESP-12F mostrará un parpadeo del led Azul que lleva incorporado, y haciendo uso del API de ARDUINO (El uso de

este API se describirá posteriormente) podemos cargar programa de ejemplo que nos permitirán ver el módulo en funcionamiento.

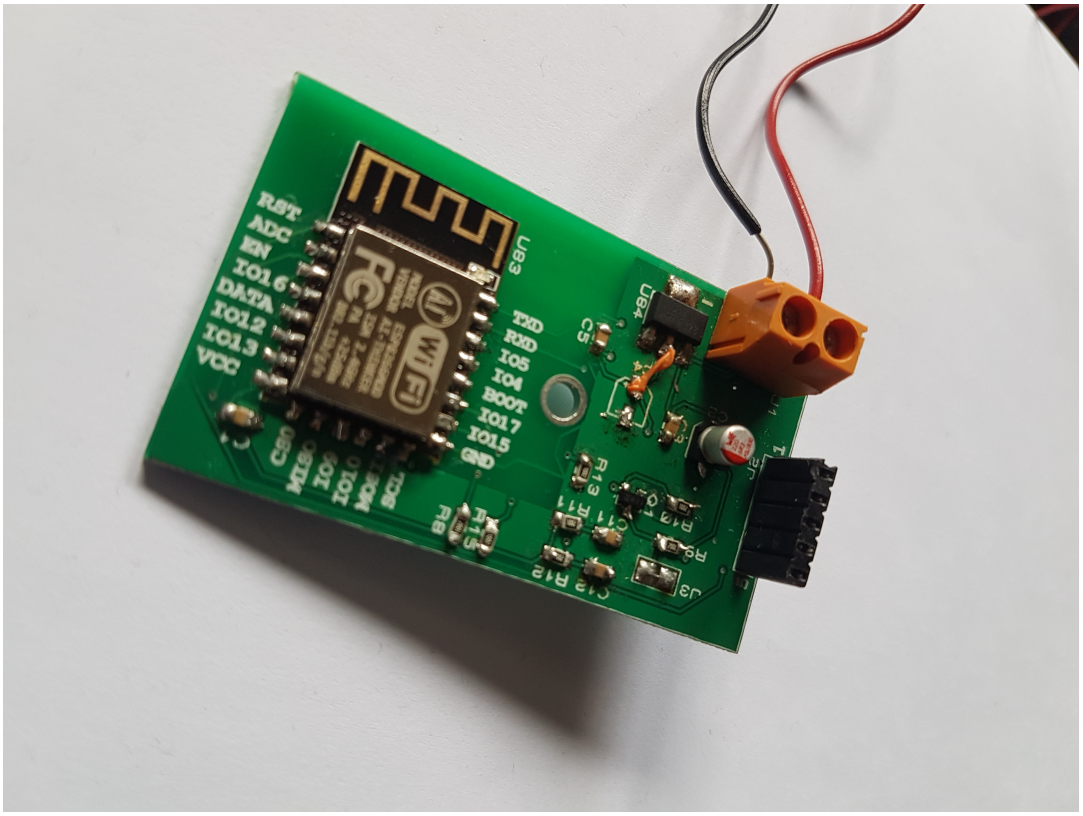
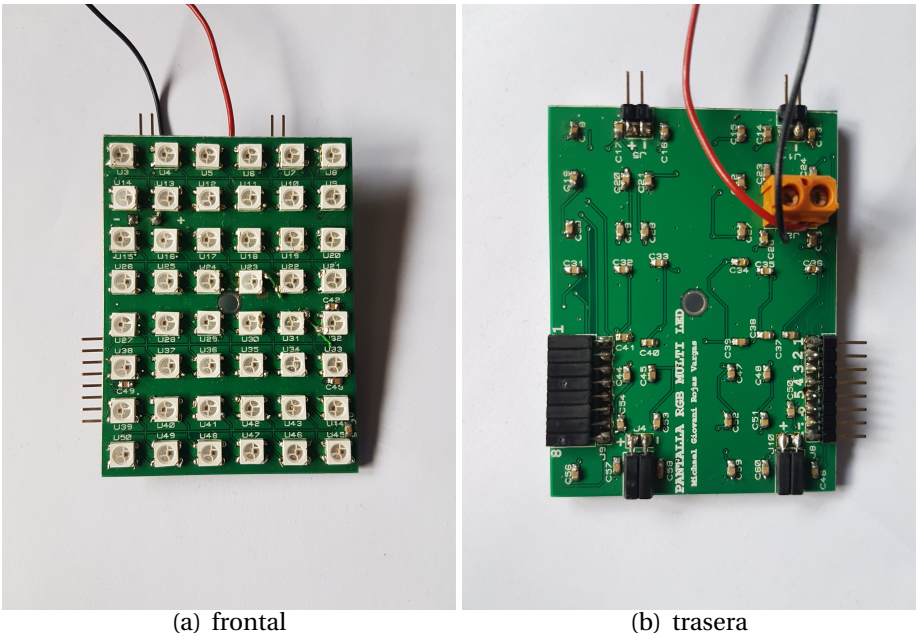


Figura 3.21: Módulo WiFi Controlador

3.3.3. Montaje de las pantallas Multi-LED RGB

Por último, procedemos al montaje de las pantallas multi-LED que utilizaremos en el prototipo. La mayor complejidad de este montaje viene dada por la numerosa cantidad de LEDs por pantalla. Cada una de las pantallas incluye 48 LEDs con 4 pines de conexión, además de un condensador de desacoplo por cada uno de los LEDs con 2 pines de conexión. En total, 288 puntos de soldadura en cada pantalla, sin contar los conectores para datos y alimentación.

La cantidad de puntos de soldadura es, sin duda, uno de los factores que más puede influir a la hora de generar problemas de funcionamiento debidos al montaje, pues, un montaje manual de tal cantidad de componentes no nos asegura que todos se monten de una manera correcta. Para asegurar una mínima calidad de montaje, con cada punto de soldadura que se realizaba, se realizaba una verificación de los pines, con el fin de comprobar que estuvieran realmente conectados con el pad y que existiera conectividad con otros puntos comunes dentro de la placa. Una tarea de verificación de la calidad que resultaba bastante extensa y agotadora.



(a) frontal (b) trasera

Figura 3.22: pantallas Multi-LED RGB

Para realizar las verificaciones de los LEDs, el proceso no es tan simple comparado con los módulos anteriores, pues, se necesita comprender primero el funcionamiento del módulo WiFi, su programación y posteriormente generar el protocolo que nos permita encender y apagar los LEDs.

Capítulo 4

Desarrollo del Firmware y del Software

4.1. Introducción

Hasta ahora hemos comentado todo lo relacionado a la parte hardware de nuestro proyecto, todo lo relacionado con la estructura de la instalación y el diseño de los circuitos. Una vez hemos conseguido que la parte hardware este desarrollada, es hora de afrontar el diseño y desarrollo del software necesario para el funcionamiento de la comunicación entre el módulo controlador y los LEDs, así como el software necesario para el control del contenido a través de la aplicación Android.

El desarrollo software es una de las actividades que requiere más tiempo dentro del proyecto, así que debe realizarse en paralelo en la medida de lo posible, mientras el desarrollo hardware se lleva a cabo. Por eso, en la estimación inicial de tiempos, mientras que la placa de circuito impreso se encuentra en proceso de fabricación y envío por parte de la empresa china, en el desarrollo software se llevaba a cabo sus primeros pasos y pruebas.

Para conseguir el software final es necesario seguir una serie de fases en el proceso de desarrollo:

I) Estudio de las herramientas y entornos de desarrollo a fin de familiarizarse con estas. Esto supones aprender el lenguaje de programación, aprender a controlar las librerías que sea necesarias, etc. En esta primera fase, nos dedicamos sobre todo al IDE de Arduino.

II) Diseño de los distintos programas que habiliten las funciones necesarias para nuestro proyecto: Comunicaciones WiFi, generación de Puntos de acceso, generación del protocolo de comunicación con internet y generación del protocolo de control para los LEDs s RGB WS2812B.

III) Primeras pruebas sobre las pantallas Multi-LED RGB. Durante esta fase se comprueba el funcionamiento correcto de la pantalla, dando paso a la corrección de los posibles errores de montaje. Posteriormente nos ocuparemos de mostrar contenido adecuado dentro de la pantalla.

IV) Desarrollo de la aplicación Android: Paralelamente se debe desarrollar la aplicación en Android que permite comunicarnos con el módulo programador. Así que debe desarrollarse una inter-

faz gráfica, implementar la conectividad interna a través de los protocolos y las funciones que veamos necesarias para un funcionamiento adecuado.

v) Cuando tengamos ambas partes del software medianamente desarrollado de manera aislada, podemos dar paso a la integración de ambas partes y así poder perfeccionar del funcionamiento de la comunicación y del software en conjunto.

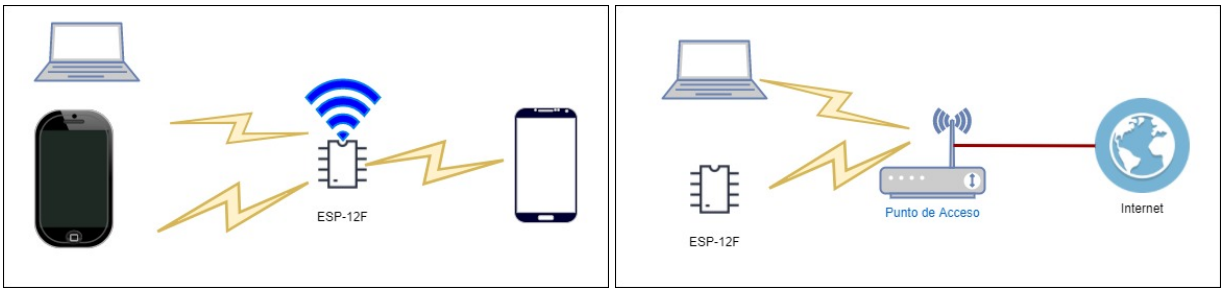
Abordemos entonces ambas partes del software, empezando primero por la parte que se encarga del funcionamiento del microprocesador y modulo wifi ESP-12F, a la que llamaremos desarrollo firmware.

4.2. Estructura de la comunicación

Modos de conexión del ESP-12F

El módulo ESP-12F que hemos usado para la implementación del Módulo WiFi controlador de nuestra aplicación, posee dos modos básicos de funcionamiento respecto a su conectividad WiFi.

- **Modo STATION:** *Modo Estación o modo cliente*. En este modo, el microcontrolador es capaz de conectarse a cualquier punto de acceso de las a redes WiFi que tenga a su alcance con el fin de conectarse a internet, integrarse en la red o de desplegar un servicio tipo web server o servidor DNS sobre esta red.
- **Modo AP:** *Modo punto de acceso*. Este modo permite al módulo generar una red WiFi de area local propia y admitir las conexiones de otros dispositivos que tengan sus credenciales de red.



(a) Modo AP (b) Modo STATION

Figura 4.1: Modos de conexión WiFi del ESP-12F

Utilizaremos ambos modos para diferentes funcionalidades:

- Utilizaremos el modo AP para cuando nuestro módulo este en un "punto de fábrica", cuando no sabe las credenciales de ninguna red WiFi y no puede conectarse a internet para establecer una comunicación de transferencia de contenido. Solo podremos conectarnos a su propia red WiFi generada, para configurar el modulo ESP-12F haciendo uso de la aplicación de control para Android.
- El modo STATION lo usaremos para realizar la comunicación a traves de internet con la aplicación que envía el contenido. Evidentemente para funcionar en este modo es necesario tener unas credenciales WiFi válidas, un punto importante a la hora de desarrollar el software de nuestro dispositivo.

El módulo al conectarse a la WiFi, tiene acceso a internet de la misma forma que la aplicación en nuestro dispositivo Android debe ser capaz de conectarse, pero, ¿Cómo realizamos el intercambio de datos?, es aquí cuando entra en juego el **protocolo MQTT**.

4.2.1. Protocolo MQTT (Message Queue Telemetry Transport)

MQTT es un protocolo de comunicación que nos ofrece una plataforma de intercambio de datos extremadamente ligera, que se basa en la publicación y suscripción de topics que además, ocupa muy poco ancho de banda. Es ideal para aplicaciones móviles gracias a sus bajos requerimientos de recursos de procesamiento, poco tamaño de los paquetes datos, baja potencia de consumo y a su distribución eficiente de la información a uno o más receptores.

Este protocolo se sitúa en las capas superiores del modelo OSI de comunicaciones, lo cual hace que sea necesario que los participantes de la comunicación contengan una pila de protocolos TCP/IP, como es el caso del módulo ESP-12F y de nuestro dispositivo Android.

Figura 4.2: MQTT sobre capas OSI

Características del protocolo MQTT

- Como se basa en la publicación/suscripción de mensajes, no es necesario saber el destino de los datos que se quieren enviar, lo cual reduce la carga efectiva en los mensajes enviados a través de la red. Esto es lo que lo hace un protocolo ligero, al no necesitar demasiada carga de datos en las cabeceras o al no implementar comunicaciones bajo demanda.
- Es open source, lo que lo hace fácilmente integrable a cualquier tipo de tecnología, protocolo y aplicaciones dedicadas al internet de las cosas.
- Posee comandos de control sencillos y permite varios modos de gestión de los mensajes, además, el contenido de los mensajes no es necesario transmitirse en un formato especifico.
- Posee mecanismos que pueden garantizar una comunicación fiable, implementando hasta tres niveles de calidad de servicio (QoS), lo cual permite garantizar la entrega de los mensajes que sean realmente importantes.

¿Como funciona el protocolo MQTT?

MQTT basa su funcionamiento en una topología estrella, donde existe un nodo central denominado broker, que tiene la capacidad para trabajar con una gran cantidad de clientes y que sigue la estrategia del modelo de comunicaciones consumidor-productor. Esta estrategia consiste en la publicación de mensajes de determinados tipos o temas (denominados *topics*) y en la suscripción a estos mensajes. Los mensajes se publican en la red asociados siempre a alguno de los diferentes topics existentes y los clientes MQTT reciben solo aquellos a los que están suscritos, ignorando todos aquellos mensajes en los que no están interesados. De la misma forma, los nodos pueden publicar mensajes asociados a los topics y comunicarse con otro nodos o aplicaciones que también estén interesados en el mismo topic.

Este modelo de comunicación aporta un desacoplo entre el transmisor y el receptor del mensaje gracias a la intermediación del broker. El desacoplo se realiza en varios niveles:

- Es una comunicación no orientada a conexión, donde no hace falta que ambos este conectados a la vez. El broker puede almacenar mensajes para clientes que no se encuentren en ese momento si se desea. Esto es más utilizado cuando se especifica algún tipo de QoS, pues generalmente, las comunicaciones se llevan a cabo en tiempo real.
- El nodo que publica el mensaje no necesita saber nada acerca de la aplicación que va a consumir el mensaje, a diferencia de las comunicaciones punto a punto donde se necesita saber cómo mínimo las direcciones IP de origen y destino. En MQTT solo es necesario conocer la dirección IP y el puerto del broker que gestiona los mensajes.

4.2.2. Arquitectura MQTT

Broker MQTT

El broker MQTT es un servicio software que se implementa para el funcionamiento del protocolo MQTT y que se encarga de establecer la comunicación a nivel de aplicación entre los diferentes nodos implicados. Funciona como servidor intermediario entre los nodos que producen el mensaje y los nodos que lo consumen. Se encarga de recibir los mensajes, filtrarlos y encaminarlos a todos aquellos clientes que se encuentren suscritos al topic asociado del mensaje. También puede realizar las tareas de identificación y autorización de los clientes si así estuviera implementado.

Clientes MQTT

Los clientes MQTT pueden ser de diferentes tipos. Se pueden tratar de dispositivos que se encargan de enviar información recolectada(como puede ser un sensor o un sistema embebido), o aplicaciones software que ejecutan librerías MQTT y que interactúan con los datos. Todos los nodos pueden publicar mensajes y suscribirse a estos, así como controlar los sensores o dispositivos que se encuentren dentro de su estructura. Para su comunicación, es imprescindible la existencia del nodo centralizado o broker.

Topics

Todos los mensajes deben llevar asociado un topic concreto. Estos topics, no son más que el tema o asunto concreto al que está referido el contenido de este mensaje. Son entonces, un identificador del contenido que lleva cada uno de los mensajes y cuyo fin es clasificar y distinguir unos mensajes de otros y, por ende, unos nodos de otros que no estén suscritos a determinados temas.

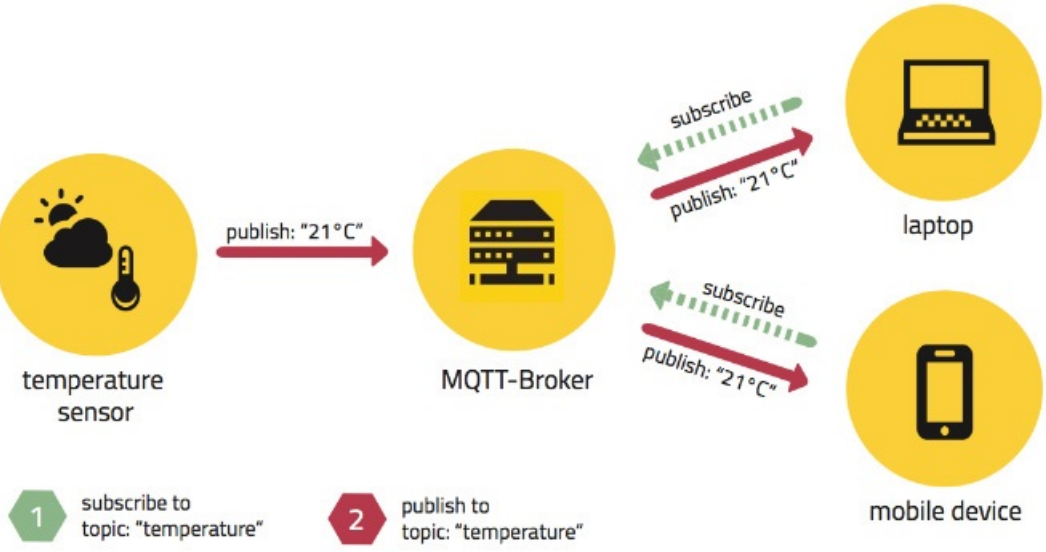


Figura 4.3: Arquitectura de comunicación MQTT

Ahora que sabemos el protocolo que usaremos en nuestro proyecto, vamos a pasar a describir como se ha desarrollado cada una de las partes del software para nuestro proyecto.

4.3. Desarrollo del Firmware del ESP-12F

4.3.1. Introducción

El módulo ESP-12F es un chip de alta integración, que incluye grandes prestaciones en un solo dispositivo, interiorizando el funcionamiento de un microcontrolador con bastantes prestaciones (que se puede controlar a través de su UART) e incluyendo una solución completa TCP/IP para la conexión con redes WiFi, lo cual lo hace ideal para interconectar on el mundo exterior las distintas aplicaciones en las que es usado controlador, a través de Internet.

Por defecto, este tipo de módulos viene con un firmware de fábrica que implica que su programación se realice a través de comandos AT. Estos comandos fueron desarrollados por Dennis Hayes para permitir configurar modems de una manera simple. Estos comandos son, básicamente, instrucciones codificadas que conforman un lenguaje que permite comunicar al programador con el dispositivo, e indicar su modo de funcionamiento. En cualquier caso, el uso de estos comandos en la práctica, no acaba de ser completamente sencillo.

Debido a la gran acogida que ha tenido no solo este módulo, sino la familia entera de módulos ESP8266, las comunidades de programadores del mundo han desarrollado diferentes IDEs (*Integrated Development Enviroment*, en castellano, Entorno integrado de desarrollo), que hacen la programación del ESP-12F más fácil, añadiendo una capa de abstracción entre los comandos AT y el IDE a utilizar.

En este caso, gracias al amplio uso por parte de la comunidad, se ha elegido el IDE de ARDUINO para desarrollar el firmware del controlador ESP-12F de nuestro prototipo.

4.3.2. IDE Arduino

Arduino es una plataforma de prototipos de electrónica de código abierto (open-source), que se basa en un hardware flexible (típicamente compuesto por una placa y un micrcontrolador) y un entorno de desarrollo que nos permite crear programas para ser cargados y ejecutados en dicho hardware.

El entorno de desarrollo es un software libre, a través del cual podemos crear los programas (denominados *Sketches*) a ejecutar. Para ello el entorno incluye, básicamente, un editor de texto para escribir el código y un compilador provisto con un corrector de sintaxis para corregir los errores más simples de la programación. Todo esto se compila y se vuelca en la memoria del microcontrolador haciendo uso de un puerto Serie, habilitando una comunicación serie entre PC y microprocesador que nos permite depurar nuestro código.

Gracias a su uso extendido en la comunidad de programadores, el IDE de Arduino tiene una gran variedad de librerías, la cuales nos permite usar funcionalidades complejas de forma simplificada. Esta es una de las grandes ventajas de Arduino, la reutilización de códigos desarrollados previamente por la comunidad esta a la orden del día, permitiendo el desarrollo colaborativo de librerías estables para multitud de periféricos.

Estas librerías, pueden ser añadidas por defecto dentro del propio IDE. Este es el caso de la familia de microcontroladores ESP8266, ya que nos encontramos con muchas librerías existentes. La más genérica de ellas, nos permite el control de las principales características del microcontrolador, a través de la programación en lenguaje Arduino (el cual está basado en C/C++), evitándonos así usar el control por comandos AT.

Figura 4.4: IDE de Arduino

4.3.3. Descripción del programa de control para el ESP 12-F

Descripción de las librerías usadas

Una parte importante para el desarrollo del software es identificar las librerías necesarias para poder implementar el control completo de nuestro modulo:

La librería ***ESP8266WiFi.h***, nos permite controlar desde el microprocesador incorporado dentro del propio ESP-12F las diferentes funciones relacionadas con el módulo WiFi. En esta librería se incluyen clases muy importantes como *WiFiClient*, la cual nos permite realizar las conexiones TCP necesarias para el cliente MQTT. También nos permite ejecutar métodos como *WiFi.mode()*, el cual nos da la posibilidad de elegir el modo de funcionamiento del WiFi. Según el parámetro que acompañe a este método, nuestro módulo pondrá en marcha el funcionamiento WiFi adecuado.

En la tabla 4.1 se resumen los parámetros y el modo de funcionamiento que se obtiene:

Parámetro del método *WiFi.mode()*	**Modo de funcionamiento**
WIFI_AP	El módulo ESP-12F genera su propia red WiFi permitiendo conectarse a otros dispositivos.
WIFI_STA	El ESP-12F es capaz de conectarse a redes WiFi existentes.
WIFI_AP_STA	Modo dual: funcionan a la vez el modo STATION y el modo AP
WIFI_OFF	No se genera señal WiFi

Tabla 4.1: Parámetros para el modo de conexión WiFi

ESP8266WebServer.h, es una librería diseñada para poner en marcha un servidor HTTP que escucha las posibles peticiones que se le realicen, permitiéndonos pasar parámetros a través de estas peticiones. Para iniciarlo, se utiliza la clase con el mismo nombre, *ESP8266WebServer*, y se implementan métodos como *arg()*, usado para la identificación de los parámetros de las peticiones o el método *handleClient()* para activar el modo escucha del servidor.

También se utiliza la librería ***EEPROM.h***, la cual nos permite trabajar con la memoria flash no volátil que viene incorporada dentro del módulo ESP-12F. Es importante a la hora de hacer que nuestro prototipo recuerde las credenciales de la Red WiFi a la que se desea conectar para recibir la información.

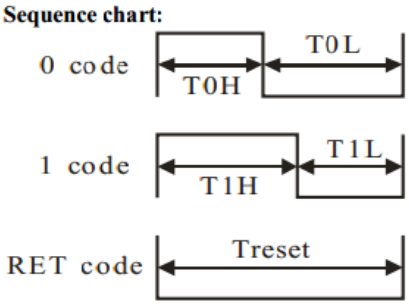
Se ha localizado también, una librería desarrollada por *Adafruit Industries*, que implementa una solución para el establecimiento de un cliente MQTT en la familia de módulos ESP8266, llamada ***Adafruit_ MQTT_ Client.h***. Es distribuida de forma libre para su uso en el IDE de Arduino, razón por la cual hemos elegido este IDE. Esta librería implementa el método *Adafruit_ MQTT_ Client()* , que consta de 5 parámetros:

- *Cliente WiFi*, para la trasmisión de conexiones UDP a través WiFi
- *IP del servidor Broker* desde donde recibiremos los datos.
- *Puerto* a través del cual se cursarán las conexiones.
- *Usuario* y *contraseña*, para dotar de seguridad nuestro broker.

Finalmente se incluyen dos librearías muy importantes para el proyecto, ***Adafruit_ NeoMatrix.h*** y ***Adafruit_ NeoPixel.h***. Estas librerías, que también han sido desarrolladas por *Adafruit Industries*, son capaces de generar el protocolo que se necesita para el funcionamiento de los LEDs WS2812B y de habilitar el funcionamiento de la matriz de LEDs de nuestro proyecto.

Los métodos de funcionamiento básicos se generan en la librería ***Adafruit_ NeoPixel.h***. Es esta la librería encargada de generar el timing correcto de las señales para los '1' y '0' lógicos del protocolo interno de los LEDs, generar la ristra de datos y enviarlos en forma de serie, para el control de uno o varios LEDs colocados de manera secuencial.

Recordemos que el protocolo es relativamente simple: el valor de '1' o '0' viene determinado por el tiempo durante el que la señal se mantiene en alta durante un periodo completo de 1.25μs, con variaciones de ±150ns como máximo (ver Figura 4.5).



WS2812B

Data transfer time(TH+TL=1.25μs±150ns)

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.9us	±150ns
T0L	0 code , low voltage time	0.9us	±150ns
T1L	1 code ,low voltage time	0.35us	±150ns
RES	low voltage time	Above 50μs	

Figura 4.5: timing de las señales para los LEDs WS2812B

En resumen, el '1' es codificado como un pulso largo de alta, mientras que el '0' se codifica como un pulso corto en alta, simplemente debe procurar cumplir los tiempos estimados.

Los pulsos deben enviarse en serie, recordemos que cada LED necesitaría 24 pulsos consecutivos para poder iluminar en un determinado color. Cada LED se encargará de tomar la información que necesita para su funcionamiento y transmitir el resto de los datos a los LEDs que le suceden.

La librería ***Adafruit_ NeoMatrix.h***, se construye basándose en la librería descrita anteriormente, haciendo uso de los métodos para la generación de la señal, con el fin de extender estos métodos para el uso de una matriz de LEDs bidimensional. Dentro de esta librería se gestiona el envío de datos a través de las tiras de LEDs, donde se tiene en cuenta la forma en que están cableados los LEDs para decidir el orden en que se deben enviar los datos.

El método ***Adafruit_ NeoMatrix()***, nos permite definir cuál es el LED en el que se inicia la trasmisión, las dimensiones de la propia matriz e indicar el orden en el que están cableados los LEDs. Esto es importante, pues, en función de cómo estén colocados, el envío de los datos debe realizarse en una sucesión distinta, como se puede apreciar en el ejemplo:

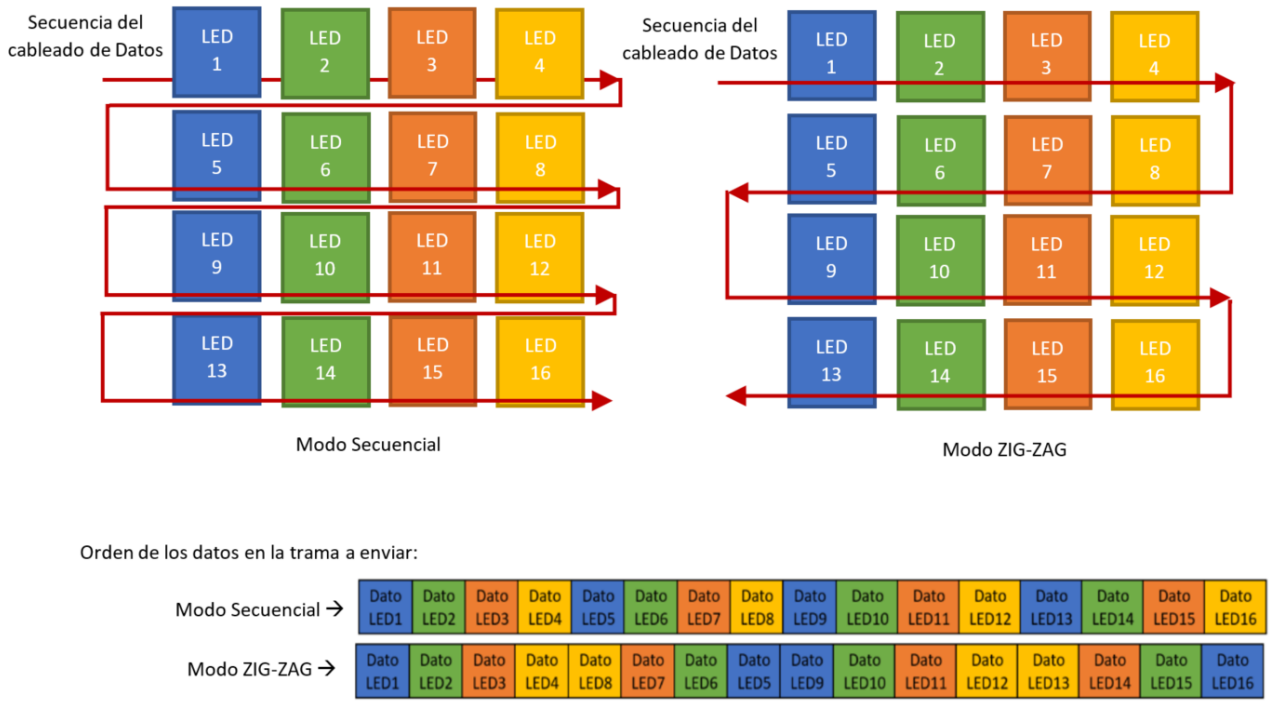


Figura 4.6: Secuencia de los datos en función del cableado de los LEDs

La ristra de datos generada en ambos casos no se produce en la misma secuencia. En nuestro proyecto, se ha seguido la configuración 'ZIG-ZAG', pues se ha considerado más adecuada al momento de seguir un orden en la transmisión de los datos, a la vez que se pensaba en la escalabilidad de la pantalla; Sea cual sea la dimension de ésta, siempre se seguirá una distribución de cableado de datos en forma de *ZIG-ZAG*,

La declaración que hacemos en nuestro código para la matriz es la siguiente:

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(12, 8, PIN,
   NEO_MATRIX_TOP + NEO_MATRIX_LEFT + NEO_MATRIX_ROWS +
   NEO_MATRIX_ZIGZAG,NEO_GRB + NEO_KHZ800);
```

Donde cada uno de los parámetros indica las características de la matriz:

- Los primeros parámetros, **12** y **8**, indican las dimensiones de la matriz de LEDs en su totalidad, una matriz de 12x8 una vez se han unido nuestras dos mini-pantallas multi-LED de 6x8.
- La constante **PIN** definida en el *sketch*, indica cuál de las salidas GPIO se debe usar para emitir la ristra de datos.
- Los parámetros **NEO_ MATRIX_ TOP** y **NEO_ MATRIX_ LEFT**, indican cuál de los LEDs que componen nuestra matriz será el primero en recibir los datos. En nuestro diseño, el primer LED será el superior izquierdo. Siguiendo la secuencia de los datos por las filas (**NEO_ MATRIX_ ROWS**) y estando interconectado entre filas en modo zig-zag (**NEO_ MATRIX_ ZIGZAG**)

- Los últimos parámetros, hacen referencia al tipo de LEDs que se están usando en la matriz, siendo **NEO_ GRB** y **NEO_ KHZ800**, los adecuados para las características de los LEDS WS2812B.

Diagrama de Flujo del programa para el módulo WiFi Controlador

Analizadas las principales librerías y siendo conscientes de las diferentes funciones, variables y métodos que disponemos, vamos a plantear ahora la solución software que se ha de implementar para poder satisfacer las necesidades de comunicación y funcionamiento de esta parte del proyecto.

Al arrancar por primera vez el módulo controlador WiFi, este verificará que existan en la memoria EEPROM credenciales para conectarse a una red WiFi. De no detectar ningún tipo de dato para hacer *log-in* a ninguna red WiFi, el módulo empezará a funcionar en **MODO CONFIGURACIÓN**. En este modo, se genera una red WiFi propia a la que debe conectarse el usuario con su dispositivo móvil y donde debe utilizar la aplicación para configurar el módulo. La aplicación permitirá introducir las credenciales de una WiFi a la que el módulo pueda conectarse para acceder a internet, y dichas credenciales serán enviadas a través de una petición HTTP hacia el servidor web que el ESP-12F se encuentra desplegando sobre su propia red WiFi.

Una vez que las credenciales se han enviado, el microcontrolador recibirá estos parámetros, los almacenará en su memoria EEPROM y emitirá una confirmación de que éstas se han recibido correctamente. Después de recibir la confirmación de ello en la aplicación, el módulo debe reiniciarse para poder arrancar en el modo de funcionamiento normal.

Al arrancar de nuevo, el módulo ve que dispone de un SSID y de una contraseña para la conexión a una red WiFi, por lo que intentará conectarse a ésta WiFi. Si por alguna razón no consigue conectarse después de varios intentos, volverá a arrancar en **MODO CONFIGURACIÓN**.

Si consigue conectarse a la red WiFi, arrancará entonces a funcionar en el que se ha denominado **MODO NORMAL**. Dentro de este modo, se inicializa el cliente MQTT y se realizan las suscripciones a los topics que nos transportarán la información que se mostrará en la pantalla:

- El topic ***Mensaje***, transportará el mensaje de texto propiamente dicho que se mostrará en la pantalla multi-LED.
- El topic ***ColorLEDS***, transportará el color en que se debe visualizar el texto.

Adicionalmente, se hace uso de un tercer topic llamado ***Modo***, el cual se utiliza para gestionar el modo de arranque una vez que se han introducido unas credenciales válidas. Si por alguna razón se desean modificar las credenciales de la conexión WiFi que se está usando, se envía un mensaje asociado a este topic, modificando la variable almacenada en la memoria que determina en cada arranque, en cuál de los modos debe arrancar el módulo. Al modificarse su valor, el siguiente inicio del módulo será en **MODO CONFIGURACIÓN** solicitando de nuevo credenciales SSID y la contraseña para la nueva conexión. El funcionamiento detallado se representa en el siguiente diagrama de flujo:

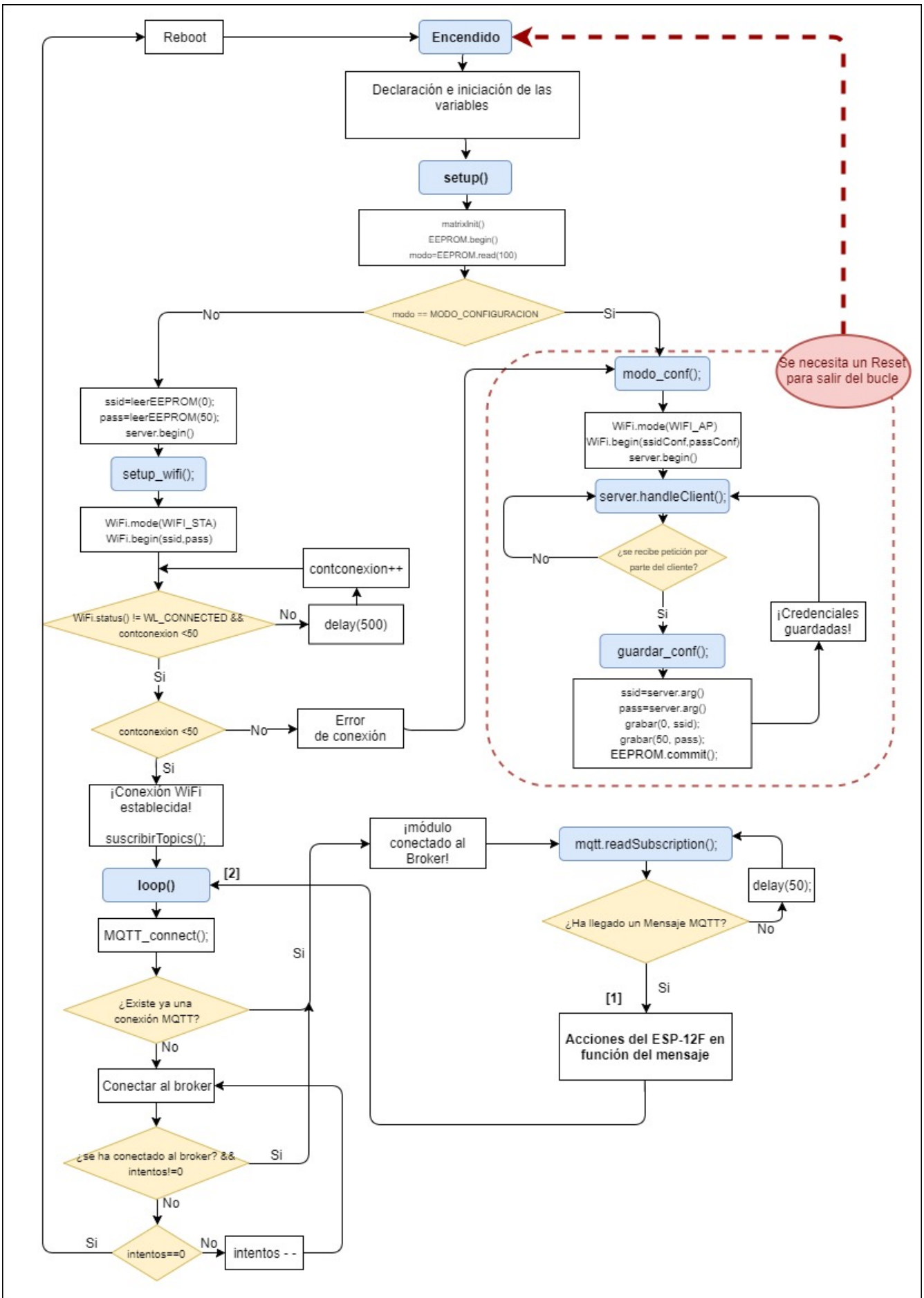


Figura 4.7: diagrama de flujo general

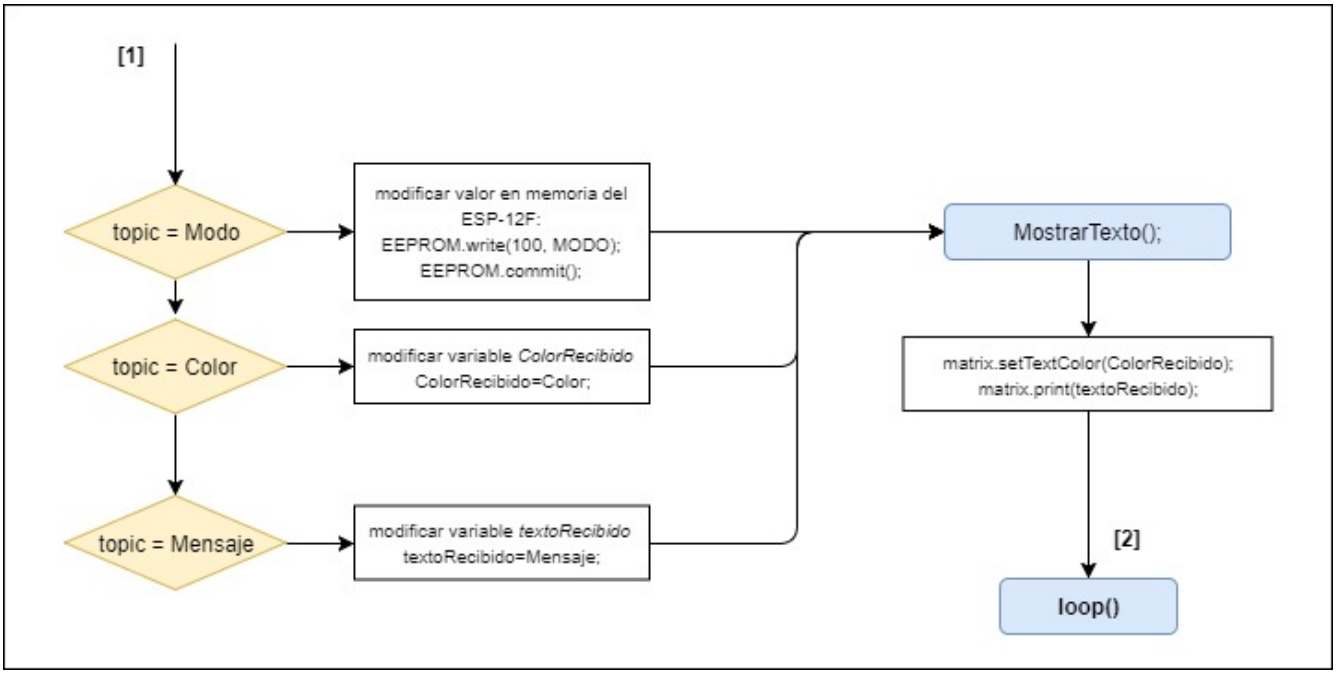


Figura 4.8: diagrama de flujo: Acciones del ESP-12F en función del mensaje topic recibido

Constantes, variables y funciones importantes del programa

En nuestro *sketch* se definen como constantes aquellos datos que son imprescindibles para que el cliente MQTT pueda realizar su conexión hacia el broker, como es el caso de la dirección del propio broker, el puerto por el que se debe cursar la conexión y el usuario y contraseña que hemos definido para el establecimiento de la comunicación:

```
#define BROKER_SERVER       "wledscontrolbroker.ddns.net"
#define BROKER_SERVERPORT   1883
#define BROKER_USERNAME     "mike921217"
#define BROKER_PASS          "921217"
```

Como variables importantes tenemos declaradas las credenciales para la red WiFi que generará el módulo, con el fin de permitir su configuración cuando arranca en **MODO CONFIGIRUACION**. Se trata de una SSID y una contraseña por defecto, que no se pueden modificar en tiempo de ejecución:

```
const char *ssidConf = "ESP-LEDS-WiFi";
const char *passConf = "12345678";
```

Cuando se introduzcan las credenciales de la red WiFi que suministrará la conexión a internet al ESP12F, se almacenarán en las variables declaradas para tal fin:

```
char ssid[50];
char pass[50];
```

Otra variable importante que debemos definir, es la matriz de colores disponibles para el texto que se va mostrar en nuestra pantalla. Los LEDs utilizados permiten hasta 8 bits para cada uno de los colores. Pero por problemas de montaje que afectann al funcionamiento eléctrico, se ha limitado al uso de los colores básicos (rojo, azul y verde) por separado y en intensidad media.

```
const uint16_t colors[] = {
      matrix.Color(100, 0,0),    // Rojo, indice: 0.
      matrix.Color(0, 100, 0),   // Verde, indice: 1.
      matrix.Color(0, 0, 100) };  // Azul, indice: 2.
```

El color en el que se ha de mostrar el texto, se determina cuando llega un mensaje asociado al topic *ColorLEDs*. En función del color recibido, éste es seleccionado en la matriz de colores usando el índice adecuado para posteriormente establecerlo como color de la matriz a través del método *setTextColor()*:

```
ColorRecibido = (char *)ColorLEDS.lastread;
    if(ColorRecibido=="Rojo"){
    indiceColor=0;  //indice del Rojo en la matriz
    }else if(ColorRecibido=="Verde"){
    indiceColor=1;  //indice del Verde en la matriz
    }else if(ColorRecibido=="Azul"){
    indiceColor=2;  //indice del Azul en la matriz
    }

    (...)
    matrix.setTextColor(colors[indiceColor]); //asignacion del color
       a los LEDs
```

Algunas de las funciones importantes se resumen en la siguiente tabla:

Nombre de la función	**Acción**
matrixInit()	Inicializa nuestra matriz de LEDs.
setup_wifi()	Inicia la conexión WiFi del modo NORMAL
modoconf()	Inicia la conexión WiFi del modo CONFIGURACION y pone en funcionamiento el servidor de peticiones HTTP sobre esta conexión.
guardar_conf()	Cuando se recibe la orden a través de una petición HTTP, el ESP ejecuta esta función para almacenar el SSID y la PASS introducidos.
leerEEPROM(int addr)	Nos permite leer 50 posiciones de memoria, de la memoria EEPROM a partir de la dirección de memoria indicada.
grabar(int addr, String a)	Nos permite grabar la cadena a, a partir de la dirección indicada. Dicha cadena no deberá tener más de 50 Bytes de longitud
SuscribirTopics()	Inicia a través del cliente MQTT la suscripción de los temas necesarios para el funcionamiento: *Modo, Mensaje y ColorLEDS*.
MQTT_connect()	Se encarga de realizar la conexión con el broker MQTT
mostrarTexto()	Muestra el contenido de la variable *textoRecibido*, en la pantalla de LEDs RGB con el color determinado en la matriz de colores predeterminados por la variable *ColorRecibido*.

Tabla 4.2: Principales funciones implementadas

Completada la implementación de todo el firmware para el Módulo WiFi controlador, es hora de pasar al desarrollo de otra de las partes de la comunicación: *la aplicación de control en Android*.

Capítulo 5

Desarrollo del software para la aplicación de control en Android

5.1. Introducción: La aplicación de Control

Otra de las partes fundamentales para el funcionamiento del prototipo en desarrollo, es la aplicación de control. Esta debe permitir implementar las comunicaciones con el módulo con el fin de poder enviar el contenido a mostrar en la pantalla o poder configurar las credenciales de acceso.

Dado que se trata de un proyecto enfocado a la movilidad y al internet de las cosas, era evidente que se requiriera una aplicación de control capaz de funcionar desde nuestros smartphones. En este caso, se ha elegido el sistema operativo Android, por la facilidad para conseguir su entorno de desarrollo y porque su programación está basada en *java*, lenguaje con el que contamos con experiencia de programación gracias a alguna asignatura vista durante la carrera.

5.2. Entorno de desarrollo: *Android Studio*

Android Studio es el *entorno de desarrollo integrado* oficial para el desarrollo de aplicaciones en Android, el cual esta basado en *IntelliJ IDEA* (IDE para desarrollo de programas informáticos). Android Studio ofrece muchas funciones que nos facilitan las pruebas durante la compilación de apps para Android:

- Sistema de compilación basado en *Gradle* flexible
- Emulador virtual rápido y posibilidad de emular en dispositivos móviles reales.
- Entorno unificado en el que realizar desarrollos para todos los dispositivos Android
- Ejecución instantánea para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK

- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código.

La gran ventaja sin duda es que se trata del IDE oficial suministrado por Google, con lo cual la cantidad de información con la que se cuenta para los desarrolladores es de gran ayuda a la hora de desarrollar tu aplicación.

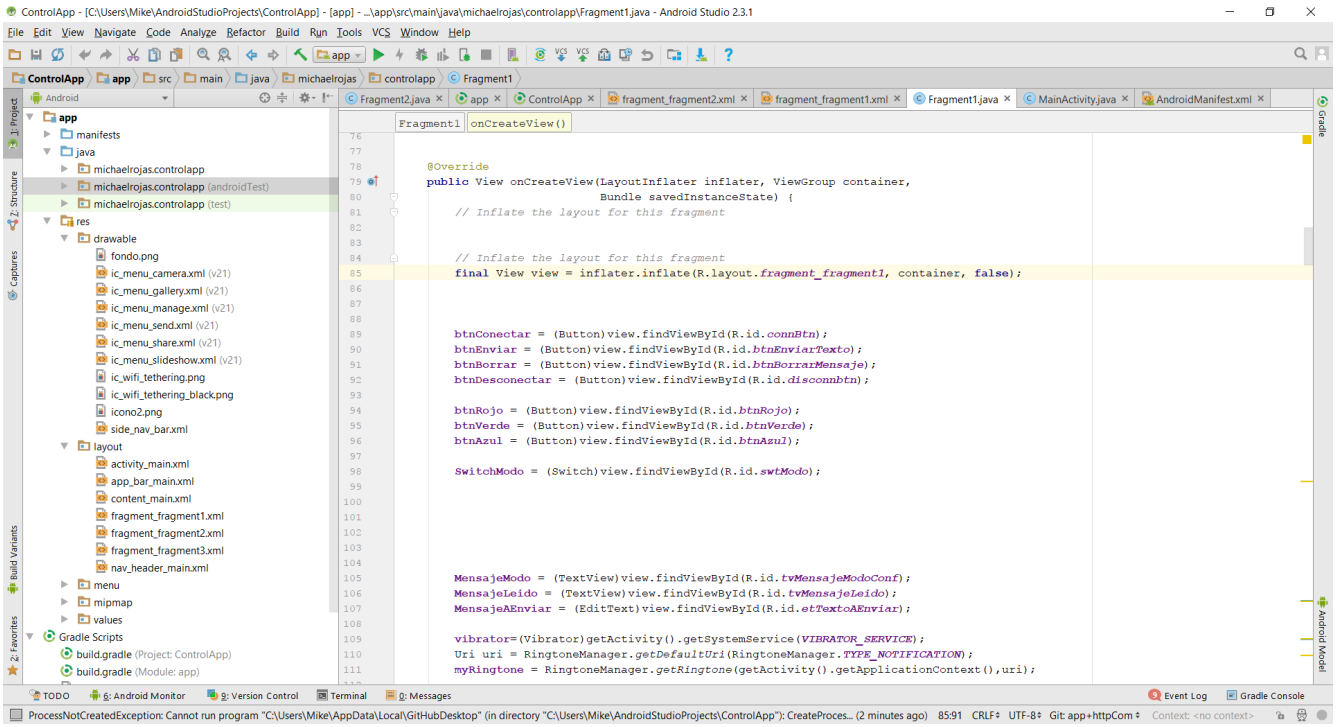


Figura 5.1: Entorno de *Android Studio*

El desarrollo de la aplicación en Android se divide en dos partes, La parte gráfica y la parte relacionada con las comunicaciones: EL cliente MQTT para comunicarse con el broker y la conexión para comunicarse directamente con el ESP-12F a través de las peticiones HTTP.

5.3. Interfaz gráfica de la Aplicación

Como toda aplicación Android, es necesario desarrollar una interfaz que permita interactuar con el usuario de una forma simple, amigable e intuitiva. Estas características se adaptarán a nuestra aplicación de control, a la que se le ha denominado *ControlApp*.

Para empezar, hemos diseñado un icono para nuestra aplicación. En este se intentan plasmar las ideas fundamentales del prototipo: Iluminación por parte de los LED a color y conexiones inalámbricas. Mezclando estos componentes, obtenemos el icono de la *ControlApp*: un LED donde el logotipo de una conexión WiFi se superpone a este haciendo de rayos de luz emitidos a color.

Figura 5.2: Icono de la *ControlApp*

Si iniciamos la aplicación, se nos recibe con una pantalla de bienvenida, en la que encontramos una pestaña lateral desplegable que nos permite seleccionar entre dos posibles opciones:

A. Enviar contenido:

Al seleccionar esta opción, se nos carga en el contenedor principal de la aplicación, una interfaz (figura XXXX*) que permite el control del contenido mostrado en la pantalla multi-LED, gestionar la conexión entre la aplicación y el broker, y controlar el modo en el que el módulo controlador debe arrancar en su próximo reinicio. Se describen a continuación cada uno de los elementos funcionales de esa interfaz.

Figura 5.3: Interfaz: *Enviar contenido*

1. **Interruptor de modo configuración:** Este interruptor envía una instrucción para el ESP-12F encapsulada en un mensaje MQTT, el cual está etiquetado bajo el topic *"Modo"*. La instrucción enviada, indica al ESP-12F el modo en el que debe arrancar en el próximo reinicio.

   Activar el interruptor, hará que el módulo olvide las credenciales WiFi actuales y se ponga en MODO CONFIGURACION para almacenar unas nuevas credenciales. Si el interruptor se desactiva, seguirá recordando las credenciales actuales, en caso de que existan.

2. **Campo Mensaje a enviar:** En este campo se introduce el texto que se va a mostrar en la pantalla Multi-LED RGB.

3. **Botón Enviar:** Este botón da la orden a los métodos que controlan la comunicación con el broker, de que se envíe el texto introducido en el campo [2], a través de un mensaje MQTT, etiquetado con el topic *"Mensaje"*. Si la conexión está establecida correctamente, el mensaje aparecerá tanto en la pantalla multi-LED RGB, como en la pantalla de retroalimentación de la aplicación [5].

4. **Borrar Mensaje:** Su función es borrar cualquier mensaje enviado previamente que se esté mostrando en la pantalla multi-LED, lo cual nos permite apagar la pantalla. Esta función se consigue de la misma forma que lo hace el botón Enviar, simplemente en este caso, se envía un mensaje vacío.

5. ***Pantalla* de retroalimentación:** Es un campo que está suscrito en el broker al topic "Mensaje", simplemente con el fin de verificar que los mensajes se reciben correctamente en cualquier cliente suscrito.

6. **Botones de selección de Color del texto:** Este campo se compone de un conjunto de botones, los cuales envían la información relativa a su respectivo color, encapsulado en un mensaje MQTT bajo el topic *"ColorLEDs"*. Al recibirlo, la pantalla Multi-LED cambiará la variable que controla el color del texto mostrado. El cambio en el color del texto se verá reflejado en la siguiente ejecución del bucle que controla el texto.

7. **Botones de Conexión con el broker:** Estos botones controlan la conexión con el servidor broker. Si realizamos una desconexión de la comunicación, todos los botones anteriores no podrán realizar ninguna función, así que la aplicación inhibe su funcionamiento y mostrará un mensaje indicando *"No connected"*, al menos hasta que se vuelva a lanzar una reconexión con el broker, en ese caso, todos los botones volverán a funcionar.

A. Configuración WiFi:

Esta segunda opción nos carga otra pantalla distinta en la aplicación, que es usada únicamente cuando el módulo arranca en modo configuración. El usuario debe conectarse con su móvil a la *WiFi de configuración* que éste genera, cuyo nombre está definido por defecto como *"ESP-LEDS-WiFi"* y cuya contraseña es *"12345678"*.

Esta pantalla nos ofrece menos opciones, pues su cometido es únicamente permitir comunicarnos con el módulo ESP-12F directamente y suministrarle las credenciales de una WiFi con internet. Los elementos que contiene son los siguientes:

Figura 5.4: Interfaz: *Configuración WiFi*

1. **Campo para el SSID:** aquí se debe introducir el nombre de la WiFi a la que queremos que el módulo controlador se conecte para acceder al broker.
2. **Campo PASSWORD:** En este campo se introduce la contraseña de la red WiFi. El campo oculta los caracteres para brindar privacidad a la hora de introducir este dato.
3. **Botón guardar configuración:** Este botón se encarga de establecer la comunicación con el ESP-12F con el fin de guardar las credenciales introducidas. La comunicación se hace a través de una petición HTTP enviada hacia el web Server desplegado en la WiFi de configuración. Se construye la petición utilizando los parámetros introducidos en los campos anteriores y se envían al servidor Web. Esta petición es recibida por el módulo ESP-12F, el cual extrae los datos que le hemos pasado como argumentos en la petición, y los almacena en la memoria EEPROM

de la que dispone, modificando así la variable que controla el modo de arranque y habilitando el arranque en MODO NORMAL.

Cuando los datos se han escrito de forma satisfactoria en la memoria, el módulo controlador envía un mensaje de confirmación hacia la aplicación (que se cursa también a través de HTTP). La aplicación recibe dicha confirmación y nos muestra un mensaje que nos informa de que los datos se han recibido y almacenado correctamente, y solicita el reinicio del módulo ESP-12F para poder arrancar en MODO NORMAL.

5.4. Software de comunicación de la aplicación

La segunda parte necesaria para completar la aplicación de control, es el código que se encarga de establecer el comportamiento de los elementos gráficos que se han listado en la sección anterior. La parte fundamental de este software para la comunicación, está en la implementación de los clientes para los protocolos de comunicación utilizados, es decir, tanto para el protocolo MQTT como para las peticiones HTTP. Para la implementación del cliente MQTT, se ha utilizado la librería *Paho Java MQTT Client*, la cual encapsula todas las conexiones MQTT a través de los servicios de Android que se ejecutan en el segundo plano de la aplicación. Esta librería incluye todos los métodos necesarios para realizar la conexión MQTT hacia el broker.

De la misma forma que en el módulo controlador, se suministran a la aplicación los datos básicos para realizar la conexión con el broker, como son la dirección del broker, el puerto, usuario y contraseña de nuestra conexión configurada y los topics a los que debemos suscribirnos.

```
static String MQTTHOST = "tcp://wledscontrolbroker.ddns.net:1883";
static String USERNAME = "mike921217";
static String PASSWORD = "921217";
static String topicColor = "Color";
static String topicMensaje = "Mensaje";
static String topicModo = "Modo";
```

Para cada botón se implementa un método *setOnclickListener*. Este tipo de métodos se activan cada vez que se pulsa un botón. El botón pulsado llama a su método asociado y ejecuta las acciones que se describen en su interior. Por ejemplo, observemos el método del botón que envía la información del color de los LEDs para que se iluminen de azul:

```
btnAzul.setOnClickListener(new View.OnClickListener() {
    @Override
   public void onClick(View v) {

        String topic = topicColor;
        String message = "Azul";

        if(Conectado==1) {

            try {
                client.publish(topic, message.getBytes(), 0, false);
            } catch (MqttException e) {
                e.printStackTrace();
            }

        }else{
          Toast.makeText(getActivity(), "No connected", Toast.LENGTH_LONG).show();
        }
    }
});
```

El método carga en la variable *topic* el parámetro que hace falta para hablar sobre el color del que se deben encender los LEDs (*topicColor*), y como mensaje a transmitir, pone en la variable *message* el dato del color *"Azul"*. Si se encuentra conectado al broker MQTT, enviará a través del cliente MQTT que se ha implementado, una publicación con el *topic* y el *message*, dichos. En caso de no estar conectado al broker, el botón no hará ninguna acción y simplemente muestra un mensaje flotante en la app indicando que no está conectado.

El código completo del software de la de control y del módulo ESP-12F puede consultarse en los anexos del documento.

5.5. Implementación del servidor Broker

Hasta ahora hemos hablado de cómo se han desarrollado e implementando los nodos extremos de una red que se comunica a través del protocolo MQTT. Hemos comprobado que todas las peticiones se cursan y gestionan a través de un servidor broker central, así que para nuestro proyecto nos hará falta poner en marcha un servidor que sea capaz de hacer de intermediario de los mensajes MQTT.

Como ya comentamos, el broker MQTT no es más que un servicio software que se ejecuta en una máquina anfitriona. Esta máquina como cualquier servidor que ofrece un servicio al exterior, debe ser accesible desde cualquier punto del internet, puesto que, estamos desarrollando un servicio para el *internet de las cosas*.

5.5.1. Mosquitto Broker

Mosquitto es una solución MQTT, de tipo *open source* y desarrollada por *ECLIPSE*, que nos permitirá poner en marcha el servidor Broker. Se encuentra disponible para su instalación en Windows y distribuciones Linux, permitiéndonos implementar el protocolo MQTT en sus versiones 3.1 y 3.1.1., haciéndolo adecuado para redes de IoT, permitiendo el intercambio de miles de mensajes entre sensores, teléfonos ordenadores o microcontroladores.

5.5.2. Instalación

Para instalar mosquitto en el anfitrión, hace falta acceder a la página oficial de *mosquito.org*, y descargar el instalador adecuado para nuestro sistema operativo, ejecutarlo y seguir las instrucciones del asistente de instalación.

Durante la instalación, se nos proporcionarán dos enlaces para la instalación de las dependencias de OpenSSL y pThreads, las cuales son necesarias para el funcionamiento del servicio broker. La instalación de estos dos paquetes nos permitirá obtener dos ficheros DLL, que debemos agregar en el mismo directorio que se ha instalado mosquitto. Esto habilitará el servicio de mosquitto en Windows.

Después será necesario repetir la ejecución del instalador de mosquitto para terminar la puesta en marcha. Podemos verificar que el servicio Mosquitto Broker se encuentra ahora ejecutandose en Windows y listo para el funcionamiento:

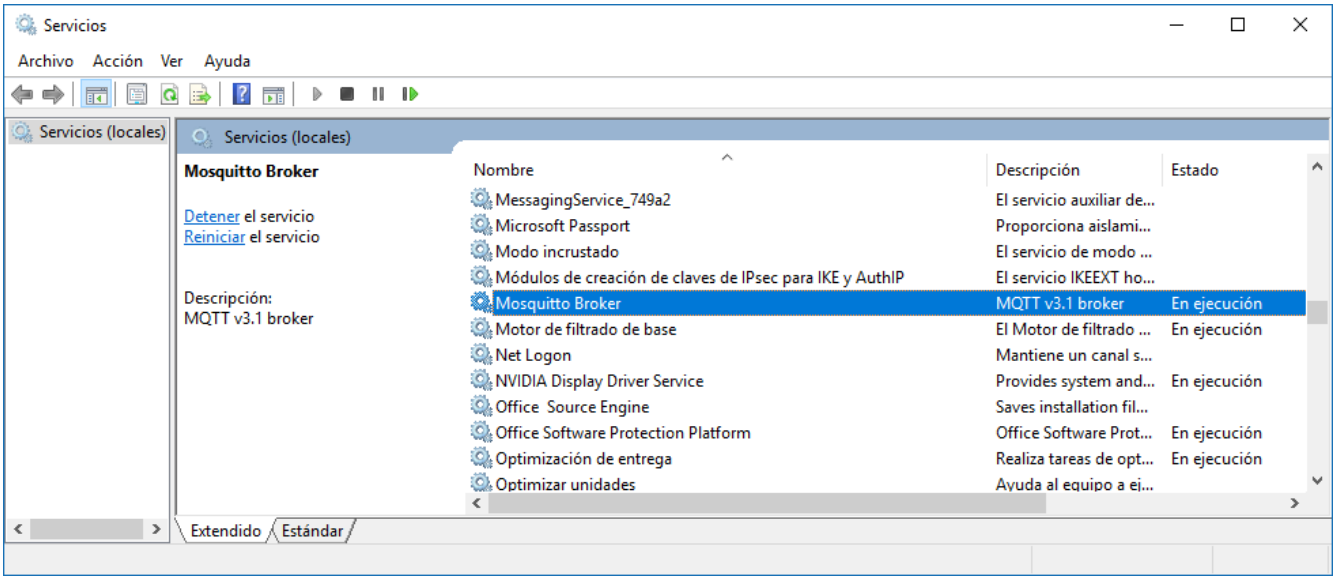


Figura 5.5: Servicio del Mosquitto broker en ejecución

Una vez instalado, para hacer que las conexiones que se establezcan con nuestro servidor broker se hagan de una manera más segura, mosquitto nos permite configurar credenciales de usuario haciendo que toda conexión que sea solicitada al broker solo sea posible si se proporcionan unas credenciales válidas que el broker reconozca. Para esto, es necesario modificar los ficheros de configuración que utiliza el servicio del sistema, que en este caso, se trata del fichero *mosquitto.conf*.

Dentro de este fichero, debemos modificar el parámetro *'allow_ anonymous'*, que por defecto viene activado en *'true'* y asignarle el atributo *'false'*. Con este cambio denegaremos toda conexión que se haga de manera anónima.

También se debe añadir el parámetro *'password_ file'*, seguido de la ruta donde se encuentre nuestro fichero con el listado de usuarios permitidos, *"C:\ProgramFiles\mosquitto\usuariosTFG´´*. Este fichero con el listado de usuarios, se genera con comandos del servicio Mosquitto a través consola.

5.5.3. Comandos básicos de gestión para el Mosquitto Broker

El servidor se encargará de iniciar los topics que vamos a usar en la transferencia de datos. Para la ejecución de estos comandos, es necesario abrir una consola de símbolo de sistema y ubicarnos en la carpeta donde se ha instalado mosquitto, una vez ahí, podremos ejecutarlos. Los comandos de mosquitto tienen una sintaxis establecida en la que se especifican los parámetros que se desean usar. A continuación, describimos los comandos utilizados:

- **Creación de usuarios:**

  En nuestro broker, no se permitirá ninguna conexión que se haga de manera anónima. Será necesario entonces, habilitar credenciales de usuario para permitir a los clientes conectarse al

broker mosquitto. Para ello utilizaremos el siguiente comando:

mosquitto_ passwd [-c] passwordfile username

El parámetro *-c*, nos permitirá crear el fichero de configuración que se está especificando, en caso de que no exista. De existir, el archivo de contraseñas se sobrescribirá, eliminando toda información que contuviera anteriormente. Al ejecutar este comando, se nos pedirá que se introduzca la contraseña para el usuario a crear y se almacenarán encriptadas en dicho fichero.

- **Publicación de un Mensaje:**

  Para la publicación de un mensaje se debe usar el ejecutable *mosquito_ pub.exe*, siguiendo la siguiente sintaxis:

  mosquitto_ pub [-h servidor] [-p puerto] [-u usuario] [-P password] -m "mensaje" -t topic

  - El primero de los flags, *-h*, nos permite indicar la dirección del servidor broker que se encargará de gestionar los mensajes. En nuestro caso, para nuestro servidor se ha habilitado un dominio dinámico gratuito de internet, ***wledscontrolbroker.ddns.net***, con el fin de que sea cual sea la dirección IP que tenga nuestro servidor, sea siempre localizable en internet a través de su dominio. Este dominio está asociado a la IP pública que la máquina anfitriona toma desde el router desmilitarizado del que disponemos en el laboratorio de proyectos.
  - *-p*, nos permite indicar el puerto a través del cual se cursará el tráfico de los datos. Aunque por defecto mosquitto y MQTT, usa el puerto 1883.
  - Seguido de los flags *-u* y *-P*, se introducen el usuario y la contraseña para conectarnos al broker. Como ya se ha comentado, a los nodos no solo les bastará el nombre del topic para suscribirse, sino que también deben contar con las credenciales adecuadas.
  - Los últimos flags hacen referencia al topic (*-t*) determinado y al contenido del mensaje (*-m*) que se desea transmitir.

- **Suscripción a un topic:**

  La suscripción no es un servicio que sea necesario ejecutar desde nuestro servidor para el funcionamiento del proyecto, pero es útil para la monitorización de las comunicaciones, sobre todo a la hora de poner en marcha el servidor y conectarlo con nuestros clientes implementados en el módulo ESP-12F y en la aplicación Android. Para suscribirse a un topic, se debe usar el ejecutable *mosquito_ sub.exe*, siguiendo la siguiente sintaxis:

  mosquitto_ sub -d [-h servidor] [-u usuario] [-P password] -t topic

Todos los flags tienen el mismo significado y uso que en el caso de la publicación de un mensaje, simplemente se añade el flag *-d*, que habilita los mensajes de monitorización de las comunicaciones.

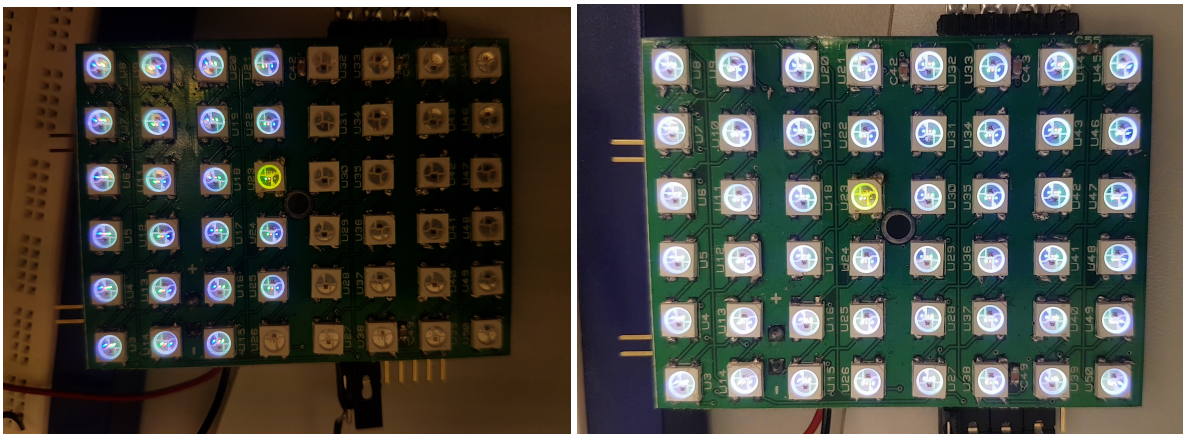
Con todo debidamente configurado y el servidor broker habilitado y disponible, la comunicación entre los nodos clientes, los cuales se han implementado tanto en el módulo controlador WiFi como en la aplicación de control en Android (ControlApp), debe realizarse a través del broker, sin problemas.

5.6. Pruebas de Montaje: Integración Hardware y Software

Después de haber desarrollado el hardware y software del sistema, debemos proseguir con la integración de ambas partes y comprobar que el funcionamiento es el esperado. Esta es una de las etapas más largas del proyecto, por que a partir de aquí, se puede empezar a comprobar que las funciones programadas se realizan o no de una manera correcta, gracias a que se tiene una realimentación del montaje completo.

5.7. Resolución de Problemas

Hasta este punto, no se había podido comprobar si el montaje de las pantallas multi-LED se ha realizado de una manera correcta, pues no disponíamos del Módulo controlador con un programa que nos permitiera encenderlos. Para comprobar el montaje, se ordena al modulo que encienda los LEDs de manera secuencial, con el fin de comprobar hasta que punto los datos se envían de forma correcta.



(a) Fallo en la transmisión

(b) Envío de datos correctos

Figura 5.6: Resolución de problemas en la pantalla multi-LED

Como se puede ver en la figura 5.6, en la imagen (a), a partir del LED U25, los siguientes LEDs no se encienden. Con ayuda del osciloscopio y otras herramientas, podemos verificar que el LED U26 no esta recibiendo los datos en la entrada, razón por la que él y los siguientes LEDs no se encienden. Podemos concluir que se trata de un mal contacto entre la salida de los datos del LED U25 y la pista que lo interconecta con el U26.

Siguiendo este proceso, se pudo habilitar el funcionamiento de todos los LEDs como se puede ver en la imagen (b) de la misma figura. A partir de aquí y siguiendo este tipo de pruebas, empiezan a solventarse muchos de los problemas de montaje que existían en las pantallas, para posteriormente probar el software final en pleno funcionamiento.

Capítulo 6

Resultados y Conclusiones

6.1. Montaje Final

Después de haber desarrollado todos los componentes tanto hardware como software que se describen en la presente memoria, al realizar el montaje final se puede comprobar que se ha conseguido un prototipo que cumple con los principales objetivos que se han planteado al principio del Trabajo de Fin de Grado.

Se ha considerado llamarle *'prototipo'* y no producto final, por que no se considera que este montaje final, esté listo para la comercialización, pues, aún hace falta desarrollar algunos aspectos, sobre todo los relacionados con componentes mecánicos del producto, como podría ser una carcasa en la que se incluya todo el prototipo con el fin de protegerlo y venderlo como un único producto.

También hace falta en este montaje la fuente de alimentación para los LEDs, la cual a pesar de haberse estimado el consumo eléctrico y pensado en su escalabilidad ofreciendo una familia de fuentes diferentes en función de la potencia que se requiera, no se ha incluido en el prototipo debido a los presupuestos que se han destinado para el proyecto.

6.2. Limitaciones encontradas

En líneas generales, todos los entornos seleccionados para la programación del software han sido adecuados para cumplir con las principales especificaciones, no siendo así para el montaje del Hardware, donde algunos aspectos han presentado carencias limitando el funcionamiento del prototipo final.

Estas carencias están asociadas a las herramientas que se disponen para realizar el montaje de los componentes. Al tratarse de piezas que requieren el montaje de muchos componentes, la introducción del factor humano en la mano de obra, hace que los errores de montaje puedan aparecer en cualquier momento, como ha sido nuestro caso.

La limitación que se ha encontrado en nuestro montaje esta asociada a la pantalla, más exactamente, asociada al color en el que se deben mostrar los contenidos. Durante las pruebas de montaje de las pantallas, se ha detectado un fallo que impide trabajar a los LEDs en colores que sean resultado de combinación de sus tres colores en alta intensidad.

Es decir, si se intentaba generar una luz magenta (resultado de encender los LEDs azul y rojo a la vez), la pantalla dejaba de responder y los LEDs empezaban a encenderse de forma aleatoria. Este mal funcionamiento se obtiene al realizar cualquier combinación de colores.

Después de realizar distintas pruebas, se detectó que esto no ocurre si se generaba el texto en los colores básicos, es decir, si se generaba el texto únicamente en azul, texto únicamente en rojo o texto únicamente en verde. Es por esta razón que nuestra aplicación solo permite elegir entre estos tres colores, dejando de lado una de las grandes características que incluía el WS2812B, como es la representación de su amplia gama de colores.

6.3. Líneas Futuras

Habiendo evaluado los resultados que se han conseguido al termino del desarrollo del proyecto, siendo conscientes de sus defectos y fortalezas, ahora se sugieren algunos puntos que convendrían mejorar para el prototipo:

I) **Terminación del producto final:** Como se ha comentado ya, es evidente que hacen falta algunos elementos a incluir, con el fin de tener un producto que sea comercializable. Hace falta una carcasa o armazón exterior, que contenga todos los componentes que conforman el prototipo. Esta carcasa debería proveer a los módulos de las sujeciones necesarias para hacerlos resistentes a los golpes o movimientos bruscos. También debería incluir un cableado con conexiones estables, y sujetas a la misma, sin olvidar que la fuente de alimentación también debería incluirse en el conjunto.

II) **Implementación de imágenes y vídeo:** El fallo identificado y que no se ha podido solventar, el cual nos limita el uso de colores debería corregirse. Cuando se disponga de todos los colores que el WS2812B es capaz de suministrar, se podría plantear el paso a la representación de imágenes y vídeo a todo color. Esto desde luego, implica la implementación de pantallas más grandes y un método para la trasferencia de los archivos a visualizar.

III) **Mejora del software de funcionamiento:** Se podría implementar otras funcionalidades dentro del bucle de funcionamiento que permitiera visualizar información mientras se espera la llegada de mensajes útiles. Podría visualizarse por ejemplo, información como la hora o la temperatura de la ubicación de la pantalla, mientras se espera el siguiente mensaje que nos interesa.

Apéndice A

Código del Módulo WiFi Controlador

A.1. Código en Arduino C/C++: *Programa-Final.ino*

```
#include <ESP8266WiFi.h> //Libreria para control del WIFI del ESP12F
#include <ESP8266WebServer.h> //Local WebServer
#include <EEPROM.h> //Libreria para Trabajar con la memoria EEPROM integrada en el ESP-12F

////Para la matrix de LEDS

#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>
#ifndef PSTR
 #define PSTR // Make Arduino Due happy
#endif

#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"

/************************* LEDS Matrix Control PIN *********************************/
#define PIN 14 //Placa
/************************* Configuracion del Broker MQTT *********************************/
//#define BROKER_SERVER "192.168.0.150"
//#define BROKER_SERVER "wifi-lights-control-broker.etowns.net"
```

```
22 #define BROKER_SERVER "wledscontrolbroker.ddns.net"
23 #define BROKER_SERVERPORT 1883
24 #define BROKER_USERNAME "mike921217"
25 #define BROKER_PASS "921217"
26
27 /************************* Otras constantes
      *********************************/
28 #define MODO_CONFIGURACION 255
29 #define MODO_NORMAL "00"
30
31 //--------------------VARIABLES GLOBALES--------------------------
32 int contconexion = 0;
33 unsigned long previousMillis = 0;
34
35 char ssid[50];
36 char pass[50];
37
38 const char *ssidConf = "ESP-LEDS-WiFi";
39 const char *passConf = "12345678";
40
41 String mensaje = "";
42
43 /************ Global State ******************/
44 // Create an ESP8266 WiFiClient class to connect to the MQTT server.
45 WiFiClient espClient;
46 // Create an ESP8266 WebServer
47 ESP8266WebServer server(80);
48
49
50 // Setup the MQTT client class by passing in the WiFi client and MQTT
       server and login details.
51 Adafruit_MQTT_Client mqtt(&espClient, BROKER_SERVER,
      BROKER_SERVERPORT, BROKER_USERNAME, BROKER_PASS);
52
53 //MATRIX DECLARATION:
54 Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(12, 8, PIN,
55   NEO_MATRIX_TOP + NEO_MATRIX_LEFT +
56   NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,
```

```
57  NEO_GRB + NEO_KHZ800);
58
59 /***************************** Feeds
     *************************************/
60
61 // subscribing topics
62
63 Adafruit_MQTT_Subscribe Modo = Adafruit_MQTT_Subscribe(&mqtt,"Modo");
64 Adafruit_MQTT_Subscribe Mensaje = Adafruit_MQTT_Subscribe(&mqtt,"
     Mensaje");
65 Adafruit_MQTT_Subscribe ColorLEDS = Adafruit_MQTT_Subscribe(&mqtt,"
     Color");
66
67 /************************* Sketch Code
     ************************************/
68
69 const uint16_t colors[] = {
70  matrix.Color(100, 0,0), matrix.Color(0, 100, 0), matrix.Color(0, 0,
       100) };
71
72 void MQTT_connect();
73
74 //-----------------------SETUP WIFI-----------------------------
75 void setup_wifi() {
76 // Conexion WIFI
77  pinMode(2, OUTPUT);
78  WiFi.mode(WIFI_STA); //para que no inicie el SoftAP en el modo
       normal
79  WiFi.begin(ssid, pass);
80  while (WiFi.status() != WL_CONNECTED and contconexion <50) { //
       Cuenta hasta 50 si no se puede conectar lo cancela
81    ++contconexion;
82    delay(500);
83
84  }
85  if (contconexion <50) {
86     Serial.println("");
87     Serial.println("WiFi conectado");
```

```
88      Serial.println(WiFi.localIP());
89      digitalWrite(2, HIGH);
90     }
91   else {
92      Serial.println("");
93      Serial.println("Error de conexion");
94      modoconf();
95   }
96 }
97 //--------------------MODO_CONFIGURACION-----------------------
98 void modoconf() {
99
100
101   pinMode(2, OUTPUT);
102
103
104   WiFi.mode(WIFI_AP);
105   WiFi.softAP(ssidConf,passConf);
106
107   IPAddress myIP = WiFi.softAPIP();
108   Serial.print("IP del acces point: ");
109   Serial.println(myIP);
110   Serial.println("WebServer iniciado...");
111
112   server.on("/guardar_conf", guardar_conf); //Graba en la eeprom la
        configuracion
113
114   server.begin();
115
116   while (true) {
117      server.handleClient();
118      digitalWrite(2, HIGH); // turn the LED on (HIGH is the voltage
           level)
119      delay(1000); // wait for a second
120      digitalWrite(2, LOW); // turn the LED off by making the voltage
           LOW
121      delay(1000);
122   }
```

```
123 }
124
125 //---------------------Guardar configuracion del WiFi
    -------------------------
126 void guardar_conf() {
127
128  Serial.println(server.arg("ssid"));//Recibimos los valores que se
       envian desde la App
129  grabar(0,server.arg("ssid"));
130  Serial.println(server.arg("pass"));
131  grabar(50,server.arg("pass"));
132
133  mensaje = "Configuracion␣Guardada..";
134  server.send(200, "text/html", mensaje);
135 }
136
137 //----------------Funcion para grabar en la EEPROM------------------
138 void grabar(int addr, String a) {
139  int tamano = a.length();
140  char inchar[50];
141  a.toCharArray(inchar, tamano+1);
142  for (int i = 0; i < tamano; i++) {
143    EEPROM.write(addr+i, inchar[i]);
144  }
145  for (int i = tamano; i < 50; i++) {
146      EEPROM.write(addr+i, 255);
147   }
148  //al grabar las credenciales activamos el inicio en Modo Normal
149   EEPROM.write(100, 0);
150  EEPROM.commit();
151 }
152
153 //-----------------Funcion para leer la EEPROM
    ------------------------
154 String leerEEPROM(int addr) {
155   byte lectura;
156   String strlectura;
157   for (int i = addr; i < addr+50; i++) {
```

```
158     lectura = EEPROM.read(i);
159     if (lectura != 255) {
160       strlectura += (char)lectura;
161     }
162   }
163   return strlectura;
164 }
165 byte modo;
166 
167 void setup() {
168 
169   ///Matrix initialization
170   matrixInit();
171 
172   //EEPROM init
173   EEPROM.begin(512);
174 
175   // MQTT and ESP initialization
176   Serial.begin(115200);
177   delay(10);
178 
179   modo = EEPROM.read(100);
180 
181 // if(modo==255){
182 if (modo == MODO_CONFIGURACION) {
183    modoconf();
184   }
185   pinMode(PIN, OUTPUT);
186 
187   leerEEPROM(0).toCharArray(ssid, 50);
188   Serial.print(ssid);
189   Serial.print("␣");
190 
191   leerEEPROM(50).toCharArray(pass, 50);
192   Serial.print(pass);
193 
194   //WiFi Initialization
195   setup_wifi();
```

```
196
197   // Setup MQTT subscriptions.
198   SuscribirTopics();
199
200 }
201
202 void matrixInit(){
203   ///Matrix initializarion
204   matrix.begin();
205   matrix.setTextWrap(false);
206   matrix.setBrightness(40);
207   matrix.setTextColor(colors[0]);
208 }
209
210 void SuscribirTopics(){
211   // Setup MQTT subscriptions
212   mqtt.subscribe(&Modo);
213   mqtt.subscribe(&Mensaje);
214   mqtt.subscribe(&ColorLEDS);
215 }
216
217 int x = matrix.width();
218 int indiceColor;
219 String textoRecibido;
220 String ColorRecibido;
221 String ModoRecibido;
222
223 void loop() {
224
225   MQTT_connect();
226
227   digitalWrite(2, HIGH);
228   Adafruit_MQTT_Subscribe *subscription;
229
230   while ((subscription = mqtt.readSubscription(50))) {
231     if (subscription == &Modo) {
232
233       ModoRecibido =(char *)Modo.lastread;
```

```
234     if(ModoRecibido=="FF"){ //MODO_CONFIGURACION
235
236       //si recibimos este mensaje, activamos la opcion de inicion en
              modo configuracion
237         EEPROM.write(100, 255);
238         EEPROM.commit();
239
240     }else if(ModoRecibido==MODO_NORMAL){
241       //si recibimos este mensaje, DESACTIVAMOS la opcion de inicion
              en modo configuracion
242         EEPROM.write(100, 0);
243         EEPROM.commit();
244     }
245
246   }else if(subscription == &Mensaje){
247
248     textoRecibido = (char *)Mensaje.lastread;
249
250     int longitud =LongitudCadena(textoRecibido);
251
252      Serial.print(F("Mensaje␣Recibido:␣"));
253      Serial.println(textoRecibido);
254      Serial.print(F("Longitud:␣"));
255      Serial.println(longitud);
256
257   }else if(subscription == &ColorLEDS){
258
259      ColorRecibido = (char *)ColorLEDS.lastread;
260      Serial.print(F("\nColor␣de␣LEDS:␣"));
261
262      if(ColorRecibido=="Rojo"){
263
264       Serial.print(F("texto␣en␣Rojo␣"));
265       indiceColor=0; //indice del rojo en la matriz
266
267      }else if(ColorRecibido=="Verde"){
268
269       Serial.print(F("texto␣en␣Verde␣"));
```

```
270       indiceColor=1; //indice del verde en la matriz
271
272      }else if(ColorRecibido=="Azul"){
273
274       Serial.print(F("texto␣en␣Azul␣"));
275       indiceColor=2; //indice del azul en la matriz
276
277      }else{
278
279       Serial.print(F("Color␣invalido..␣:/␣"));
280
281      }
282    }
283  }
284
285  mostrarTexto();
286
287  digitalWrite(2, HIGH);
288
289 }// Fin de Loop
290
291 void mostrarTexto(){
292
293  matrix.fillScreen(0);
294  matrix.setCursor(x, 0);
295
296  matrix.print(textoRecibido);
297  int le = (textoRecibido.length() * 7);
298
299  if(--x < -le) {
300    x = matrix.width();
301    matrix.setTextColor(colors[indiceColor]);
302  }
303  matrix.show();
304  delay(50);
305 }
306 //Funcion para contar longitud de cadenas
307 int LongitudCadena(String cadena){
```

```
308   int i=0, contador=0;
309
310   while(cadena[i]!='\0'){
311     i++;
312   }
313   return i;
314 }
315 // Function to connect and reconnect as necessary to the MQTT server.
316 // Should be called in the loop function and it will take care if
      connecting.
317 void MQTT_connect() {
318   int8_t ret;
319
320   // Stop if already connected.
321   if (mqtt.connected()) {
322     return;
323   }
324
325   Serial.print("Connecting to MQTT... ");
326
327   uint8_t retries = 3;
328   while ((ret = mqtt.connect()) != 0) { // connect will return 0 for
        connected
329       Serial.println(mqtt.connectErrorString(ret));
330       Serial.println("Retrying MQTT connection in 5 seconds...");
331       mqtt.disconnect();
332       delay(5000); // wait 5 seconds
333       retries--;
334       if (retries == 0) {
335         // basically die and wait for WDT to reset me
336       }
337   }
338   Serial.println("MQTT Connected!");
339 }
```

Apéndice B

Código de la aplicación de control *ControlApp* para Android: ControlApp

En este segundo apéndice, se incluyen los ficheros del código en *java* que implementan las comunicaciones de los diferentes protocolos y el funcionamiento interno de la aplicación.

B.1. Código del fichero *'MainActivity.java'*

```
1  package michaelrojas.controlapp;
2
3  import android.net.Uri;
4  import android.os.Bundle;
5  import android.support.design.widget.FloatingActionButton;
6  import android.support.design.widget.Snackbar;
7  import android.support.v4.app.Fragment;
8  import android.support.v4.app.FragmentManager;
9  import android.support.v4.app.FragmentTransaction;
10 import android.view.View;
11 import android.support.design.widget.NavigationView;
12 import android.support.v4.view.GravityCompat;
13 import android.support.v4.widget.DrawerLayout;
14 import android.support.v7.app.ActionBarDrawerToggle;
15 import android.support.v7.app.AppCompatActivity;
16 import android.support.v7.widget.Toolbar;
17 import android.view.Menu;
18 import android.view.MenuItem;
19 import android.graphics.Color;
20 import android.widget.Toast;
21
22 import org.eclipse.paho.client.mqttv3.IMqttActionListener;
23 import org.eclipse.paho.client.mqttv3.IMqttToken;
```

```
24  import org.eclipse.paho.client.mqttv3.MqttException;
25
26  public class MainActivity extends AppCompatActivity
27          implements NavigationView.OnNavigationItemSelectedListener, Fragment1.OnFragmentInteractionListener,
                Fragment2.OnFragmentInteractionListener,
28          Fragment3.OnFragmentInteractionListener {
29
30      @Override
31      protected void onCreate(Bundle savedInstanceState) {
32          super.onCreate(savedInstanceState);
33          setContentView(R.layout.activity_main);
34          Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
35          setSupportActionBar(toolbar);
36
37          getSupportActionBar().setTitle("Bienvenido!");
38
39          //// Esto permite cargar el fragment de Bienvenida
40          Fragment3 fragment = new Fragment3();
41          FragmentManager fragmentManager = getSupportFragmentManager();
42          FragmentTransaction ft = fragmentManager.beginTransaction();
43          ft.add(R.id.content_main, fragment);
44          ft.commit();
45
46          DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
47          ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
48                  this, drawer, toolbar, R.string.navigation_drawer_open, R.string.navigation_drawer_close);
49          drawer.setDrawerListener(toggle);
50          toggle.syncState();
51
52          NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
53          navigationView.setNavigationItemSelectedListener(this);
54      }
55
56      @Override
57      public void onBackPressed() {
58          DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
59          if (drawer.isDrawerOpen(GravityCompat.START)) {
60              drawer.closeDrawer(GravityCompat.START);
61          } else {
62              super.onBackPressed();
63          }
64      }
65
66      //
67      @SuppressWarnings("StatementWithEmptyBody")
```

```
68      @Override
69      public boolean onNavigationItemSelected(MenuItem item) {
70          // Handle navigation view item clicks here.
71          int id = item.getItemId();
72
73          Fragment fragment = null;
74          boolean FragmentTransaction = false;
75
76          if (id == R.id.enviarTexto) {
77              // Handle the camera action
78
79              fragment = new Fragment1();
80              FragmentTransaction = true;
81
82          } else if (id == R.id.configuracion) {
83
84              fragment = new Fragment2();
85              FragmentTransaction = true;
86
87          }
88
89          if(FragmentTransaction ){
90              getSupportFragmentManager().beginTransaction().replace(R.id.content_main, fragment).commit();
91              //Cambia el nombre de la parte superior en la App
92              item.setChecked(true);
93              getSupportActionBar().setTitle(item.getTitle());;
94          }
95
96          DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
97          drawer.closeDrawer(GravityCompat.START);
98          return true;
99      }
100
101     @Override
102     public void onFragmentInteraction(Uri uri) {
103     }
104 }
```

B.2. Código del fichero *'Fragment1.java'*

```
1  package michaelrojas.controlapp;
2
3  import android.content.Context;
4  import android.media.Ringtone;
5  import android.media.RingtoneManager;
6  import android.net.Uri;
7  import android.os.Bundle;
8  import android.os.Vibrator;
9  import android.support.v4.app.Fragment;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.view.ViewGroup;
13 import android.widget.Button;
14 import android.widget.CompoundButton;
15 import android.widget.EditText;
16 import android.widget.Switch;
17 import android.widget.TextView;
18 import android.widget.Toast;
19
20 import org.eclipse.paho.android.service.MqttAndroidClient;
21 import org.eclipse.paho.client.mqttv3.IMqttActionListener;
22 import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
23 import org.eclipse.paho.client.mqttv3.IMqttToken;
24 import org.eclipse.paho.client.mqttv3.MqttCallback;
25 import org.eclipse.paho.client.mqttv3.MqttClient;
26 import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
27 import org.eclipse.paho.client.mqttv3.MqttException;
28 import org.eclipse.paho.client.mqttv3.MqttMessage;
29
30 import static android.content.Context.VIBRATOR_SERVICE;
31
32 public class Fragment1 extends Fragment {
33
34     private OnFragmentInteractionListener mListener;
35
36     static String MQTTHOST = "tcp://wledscontrolbroker.ddns.net:1883"; //Cable
37     static String USERNAME = "mike921217";
38     static String PASSWORD = "921217";
39     static String topicColor = "Color";
40     static String topicMensaje = "Mensaje";
41     static String topicModo = "Modo";
42
43
```

```
44      MqttAndroidClient client;
45      TextView subText, MensajeLeido, MensajeModo;
46      Button btnRojo, btnVerde, btnAzul, btnConectar;
47      Button btnEnviar,btnBorrar,btnDesconectar;
48      EditText MensajeAEnviar;
49      Switch SwitchModo;
50
51      MqttConnectOptions options;
52      Vibrator vibrator;
53      Ringtone myRingtone;
54      int Conectado;
55
56      public Fragment1() {
57          // Required empty public constructor
58      }
59
60      @Override
61      public View onCreateView(LayoutInflater inflater, ViewGroup container,
62                               Bundle savedInstanceState) {
63
64          // Inflate the layout for this fragment
65          final View view = inflater.inflate(R.layout.fragment_fragment1, container, false);
66
67          btnConectar = (Button)view.findViewById(R.id.connBtn);
68          btnEnviar = (Button)view.findViewById(R.id.btnEnviarTexto);
69          btnBorrar = (Button)view.findViewById(R.id.btnBorrarMensaje);
70          btnDesconectar = (Button)view.findViewById(R.id.disconnbtn);
71
72          btnRojo = (Button)view.findViewById(R.id.btnRojo);
73          btnVerde = (Button)view.findViewById(R.id.btnVerde);
74          btnAzul = (Button)view.findViewById(R.id.btnAzul);
75
76          SwitchModo = (Switch)view.findViewById(R.id.swtModo);
77
78          MensajeModo = (TextView)view.findViewById(R.id.tvMensajeModoConf);
79          MensajeLeido = (TextView)view.findViewById(R.id.tvMensajeLeido);
80          MensajeAEnviar = (EditText)view.findViewById(R.id.etTextoAEnviar);
81
82          vibrator=(Vibrator)getActivity().getSystemService(VIBRATOR_SERVICE);
83          Uri uri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
84          myRingtone = RingtoneManager.getRingtone(getActivity().getApplicationContext(),uri);
85
86          String clientId = MqttClient.generateClientId();
87          client = new MqttAndroidClient(getActivity().getApplicationContext(), MQTTHOST, clientId);
88
```

```
89          options = new MqttConnectOptions();
90          options.setUserName(USERNAME);
91          options.setPassword(PASSWORD.toCharArray());
92
93          btnConectar.setOnClickListener(new View.OnClickListener() {
94              @Override
95              public void onClick(View v) {
96
97                  Conectado=1;
98
99                  try {
100                     IMqttToken token = client.connect(options);
101                     token.setActionCallback(new IMqttActionListener() {
102                         @Override
103                         public void onSuccess(IMqttToken asyncActionToken) {
104
105                             Toast.makeText(getActivity(), "Broker␣Connected!!", Toast.LENGTH_LONG).show();
106                             setSubscription();
107                         }
108
109                         @Override
110                         public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
111
112
113                             Toast.makeText(getActivity(), "Connection␣Failed", Toast.LENGTH_LONG).show();
114                         }
115                     });
116                 } catch (MqttException e) {
117                     e.printStackTrace();
118                 }
119             }
120         });
121
122         btnBorrar.setOnClickListener(new View.OnClickListener() {
123             @Override
124             public void onClick(View v) {
125
126                 String topic = "Mensaje";
127                 String message = "␣";
128
129                 if(Conectado==1) {
130
131                     try {
132
133                         client.publish(topic, message.getBytes(), 0, false);
```

```
134                     } catch (MqttException e) {
135                         e.printStackTrace();
136                     }
137
138                 }else{
139
140                     Toast.makeText(getActivity(), "No connected", Toast.LENGTH_LONG).show();
141
142                 }
143             }
144         });
145
146         btnDesconectar.setOnClickListener(new View.OnClickListener() {
147             @Override
148             public void onClick(View v) {
149
150                 Conectado=0;
151
152                 try {
153                     IMqttToken token = client.disconnect();
154                     token.setActionCallback(new IMqttActionListener() {
155                         @Override
156                         public void onSuccess(IMqttToken asyncActionToken) {
157                             // We are connected
158                             Toast.makeText(getActivity(), "Disconnected!!", Toast.LENGTH_LONG).show();
159                         }
160
161                         @Override
162                         public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
163                             // Something went wrong e.g. connection timeout or firewall problems
164
165                             Toast.makeText(getActivity(), "DCould not disconnect..", Toast.LENGTH_LONG).show();
166                         }
167                     });
168                 } catch (MqttException e) {
169                     e.printStackTrace();
170                 }
171             }
172         });
173
174         btnEnviar.setOnClickListener(new View.OnClickListener() {
175             @Override
176             public void onClick(View v) {
177
178                 String topic = "Mensaje";
```

```
179                 String message = MensajeAEnviar.getText().toString();
180
181                 if(Conectado==1) {
182
183                     try {
184
185                         client.publish(topic, message.getBytes(), 0, false);
186                     } catch (MqttException e) {
187                         e.printStackTrace();
188                     }
189                 }else{
190                     Toast.makeText(getActivity(), "No connected", Toast.LENGTH_LONG).show();
191                 }
192             }
193         });
194
195
196
197         btnRojo.setOnClickListener(new View.OnClickListener() {
198             @Override
199             public void onClick(View v) {
200
201                 String topic = topicColor;
202                 String message = "Rojo";
203
204                 if(Conectado==1) {
205
206                     try {
207
208                         client.publish(topic, message.getBytes(), 0, false);
209                     } catch (MqttException e) {
210                         e.printStackTrace();
211                     }
212                 }else{
213                     Toast.makeText(getActivity(), "No connected", Toast.LENGTH_LONG).show();
214                 }
215             }
216         });
217
218         btnVerde.setOnClickListener(new View.OnClickListener() {
219             @Override
220             public void onClick(View v) {
221
222
223                 String topic = topicColor;
```

```
224                 String message = "Verde";
225
226                 if(Conectado==1) {
227
228                     try {
229                         client.publish(topic, message.getBytes(), 0, false);
230                     } catch (MqttException e) {
231                         e.printStackTrace();
232                     }
233
234                 }else{
235                     Toast.makeText(getActivity(), "No connected", Toast.LENGTH_LONG).show();
236                 }
237             }
238         });
239
240         btnAzul.setOnClickListener(new View.OnClickListener() {
241             @Override
242             public void onClick(View v) {
243
244                 String topic = topicColor;
245                 String message = "Azul";
246
247                 if(Conectado==1) {
248
249                     try {
250
251                         client.publish(topic, message.getBytes(), 0, false);
252                     } catch (MqttException e) {
253                         e.printStackTrace();
254                     }
255                 }else{
256                     Toast.makeText(getActivity(), "No connected", Toast.LENGTH_LONG).show();
257                 }
258             }
259         });
260
261         SwitchModo.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
262             @Override
263             public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
264
265                 if(isChecked){
266                     String topic = topicModo;
267                     String message = "FF";
268                     if(Conectado==1) {
```

```
269
270                         try {
271                             client.publish(topic, message.getBytes(), 0, false);
272                         } catch (MqttException e) {
273                             e.printStackTrace();
274                         }
275                         Toast.makeText(getActivity(), "Se␣arrancara␣en␣modo␣confiuracion␣!!!", Toast.LENGTH_LONG
                                ).show();
276                         MensajeModo.setText("Se␣reiniciara␣en␣modo␣configuracion!!");
277                     }else{
278                         Toast.makeText(getActivity(), "No␣connected", Toast.LENGTH_LONG).show();
279                     }
280
281
282                 }else{ //Si el boton esta en off
283
284                     String topic = topicModo;
285                     String message = "00";
286
287                     if(Conectado==1) {
288
289                         try {
290
291                             client.publish(topic, message.getBytes(), 0, false);
292                         } catch (MqttException e) {
293                             e.printStackTrace();
294                         }
295
296                         Toast.makeText(getActivity(), "Se␣arrancara␣en␣modo␣confiuracion␣!!!", Toast.LENGTH_LONG
                                ).show();
297                         MensajeModo.setText("␣");
298
299
300                     }else{
301
302                         Toast.makeText(getActivity(), "No␣connected", Toast.LENGTH_LONG).show();
303                     }
304                 }
305             }
306         });
307
308
309         try {
310             IMqttToken token = client.connect(options);
311             token.setActionCallback(new IMqttActionListener() {
```

```
312                 @Override
313                 public void onSuccess(IMqttToken asyncActionToken) {
314                     // We are connected
315                     Toast.makeText(getActivity(), "Broker␣Connected!!", Toast.LENGTH_LONG).show();
316                     Conectado=1;
317                     setSubscription();
318                 }
319
320                 @Override
321                 public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
322                     // Something went wrong e.g. connection timeout or firewall problems
323
324                     Toast.makeText(getActivity(), "Connection␣Failed", Toast.LENGTH_LONG).show();
325                 }
326             });
327         } catch (MqttException e) {
328             e.printStackTrace();
329         }
330
331         client.setCallback(new MqttCallback() {
332             @Override
333             public void connectionLost(Throwable cause) {
334
335             }
336
337             @Override
338             public void messageArrived(String topic, MqttMessage message) throws Exception {
339
340
341                 if(topic.equals(topicColor)) {
342                     subText.setText(new String(message.getPayload()));
343                     myRingtone.play();
344                     vibrator.vibrate(500);
345                 }
346
347                 if(topic.equals(topicMensaje)) {
348                     MensajeLeido.setText(new String(message.getPayload()));
349                     vibrator.vibrate(1000);
350                 }
351             }
352             @Override
353             public void deliveryComplete(IMqttDeliveryToken token) {
354
355             }
356         });
```

```
357
358             return view;
359         }
360
361         public void onButtonPressed(Uri uri) {
362             if (mListener != null) {
363                 mListener.onFragmentInteraction(uri);
364             }
365         }
366
367         @Override
368         public void onAttach(Context context) {
369             super.onAttach(context);
370             if (context instanceof OnFragmentInteractionListener) {
371                 mListener = (OnFragmentInteractionListener) context;
372             } else {
373                 throw new RuntimeException(context.toString()
374                         + " must implement OnFragmentInteractionListener");
375             }
376         }
377
378         @Override
379         public void onDetach() {
380             super.onDetach();
381             mListener = null;
382         }
383
384         public interface OnFragmentInteractionListener {
385             // TODO: Update argument type and name
386             void onFragmentInteraction(Uri uri);
387         }
388
389         private void setSubscription(){
390
391             String topic=topicColor;
392
393             try{
394                 client.subscribe(topicColor, 0);
395                 client.subscribe(topicMensaje, 0);
396             }catch(MqttException e){
397                 e.printStackTrace();
398             }
399         }
400 }
```

B.3. Código del fichero *'Fragment2.java'*

```
1  package michaelrojas.controlapp;
2
3  import android.content.Context;
4  import android.graphics.Color;
5  import android.net.ConnectivityManager;
6  import android.net.NetworkInfo;
7  import android.net.Uri;
8  import android.os.AsyncTask;
9  import android.os.Bundle;
10 import android.support.v4.app.Fragment;
11 import android.view.LayoutInflater;
12 import android.view.View;
13 import android.view.ViewGroup;
14 import android.widget.Button;
15 import android.widget.EditText;
16 import android.widget.TextView;
17 import android.widget.Toast;
18
19 public class Fragment2 extends Fragment {
20
21     private OnFragmentInteractionListener mListener;
22
23     static String IP_SERVER = "http://192.168.4.1/";
24
25     Button btnEscanear, btnGuardar;
26     TextView tvReinicio;
27     EditText etSSID, etPass;
28
29     ////////////*********************************
30
31     String cadena;
32
33     public Fragment2() {
34         // Required empty public constructor
35     }
36
37     @Override
38     public View onCreateView(LayoutInflater inflater, ViewGroup container,
39                              Bundle savedInstanceState) {
40         // Inflate the layout for this fragment
41         final View view = inflater.inflate(R.layout.fragment_fragment2, container, false);
42
43         btnGuardar = (Button)view.findViewById(R.id.btnGuardar);
```

```
44          etPass = (EditText) view.findViewById(R.id.etPass);
45          etSSID = (EditText) view.findViewById(R.id.etSSID);
46          tvReinicio = (TextView) view.findViewById(R.id.tvReinicio);
47
48          btnGuardar.setOnClickListener(new View.OnClickListener() {
49              @Override
50              public void onClick(View v) {
51
52                  String parametroSSID = etSSID.getText().toString();
53                  String parametroPasswd = etPass.getText().toString();
54
55                  ConnectivityManager conMngr = (ConnectivityManager)
56                          getActivity().getSystemService(Context.CONNECTIVITY_SERVICE);
57
58                  NetworkInfo networkInfo = conMngr.getActiveNetworkInfo();
59
60                  if(networkInfo != null && networkInfo.isConnected()){
61
62                      String url = IP_SERVER+"guardar_conf?ssid="+ parametroSSID +"&pass="+parametroPasswd;
63
64                      new SolicitaDatos().execute(url);
65                  }else{
66                      Toast.makeText(getActivity(), "Ninguna␣conexion␣detectada", Toast.LENGTH_LONG).show();
67                  }
68              }
69          });
70          return view;
71      }
72
73
74      private class SolicitaDatos extends AsyncTask<String, Void, String > {
75
76          @Override
77          protected String doInBackground(String... url) {
78              return Conexion.getData(url[0]);
79          }
80
81          @Override
82          protected void onPostExecute(String resultado) {
83              if(resultado != null){
84
85                  if(resultado.contains("Configuracion␣Guardada")){
86                      Toast.makeText(getActivity(), "Configuracion␣Guardada...", Toast.LENGTH_LONG).show();
87                      tvReinicio.setText("Configuracion␣WiFi␣Guardada.␣Por␣favor,␣reinicie␣el␣controlador␣ESP...");
88                  }
```

```
89                  }else{
90                      Toast.makeText(getActivity(), "Error", Toast.LENGTH_LONG).show();
91                  }
92              }
93          }
94
95          public void onButtonPressed(Uri uri) {
96              if (mListener != null) {
97                  mListener.onFragmentInteraction(uri);
98              }
99          }
100
101         @Override
102         public void onAttach(Context context) {
103             super.onAttach(context);
104             if (context instanceof OnFragmentInteractionListener) {
105                 mListener = (OnFragmentInteractionListener) context;
106             } else {
107                 throw new RuntimeException(context.toString()
108                         + " must implement OnFragmentInteractionListener");
109             }
110         }
111
112         @Override
113         public void onDetach() {
114             super.onDetach();
115             mListener = null;
116         }
117
118         public interface OnFragmentInteractionListener {
119             // TODO: Update argument type and name
120             void onFragmentInteraction(Uri uri);
121         }
122 }
```

B.4. Código del fichero *'Fragment3.java'*

```
1  package michaelrojas.controlapp;
2
3  import android.content.Context;
4  import android.net.Uri;
5  import android.os.Bundle;
6  import android.support.v4.app.Fragment;
7  import android.view.LayoutInflater;
8  import android.view.View;
9  import android.view.ViewGroup;
10
11 public class Fragment3 extends Fragment {
12
13     private OnFragmentInteractionListener mListener;
14
15     public Fragment3() {
16         // Required empty public constructor
17     }
18
19     @Override
20     public View onCreateView(LayoutInflater inflater, ViewGroup container,
21                              Bundle savedInstanceState) {
22         // Inflate the layout for this fragment
23         return inflater.inflate(R.layout.fragment_fragment3, container, false);
24     }
25
26     public void onButtonPressed(Uri uri) {
27         if (mListener != null) {
28             mListener.onFragmentInteraction(uri);
29         }
30     }
31
32     @Override
33     public void onAttach(Context context) {
34         super.onAttach(context);
35         if (context instanceof OnFragmentInteractionListener) {
36             mListener = (OnFragmentInteractionListener) context;
37         } else {
38             throw new RuntimeException(context.toString()
39                     + "␣must␣implement␣OnFragmentInteractionListener");
40         }
41     }
42
43     @Override
```

```
44      public void onDetach() {
45          super.onDetach();
46          mListener = null;
47      }
48
49      public interface OnFragmentInteractionListener {
50          void onFragmentInteraction(Uri uri);
51      }
52  }
```

Bibliografía

[1] *Documentation for ESP8266 Arduino Core. Installation instructions, functions and classes reference.* http://esp8266.github.io/Arduino/versions/2.1.0-rc1/doc/installing.html.*Accessed: 2017.*

[2] *Light_ WS2812 library V2.0 – Part I: Understanding the WS281.* https://cpldcpu.com/2014/01/14/light_ws2812-library-v2-0-part-i-understanding-the-ws2812/. *Accessed: 2017.*

[3] *Adafruit_ NeoMatrix documentation repository.* https://github.com/adafruit/Adafruit_NeoMatrix.*Accessed: 2017.*

[4] *Adafruit_ NeoPixel documentation repository.* https://github.com/adafruit/Adafruit_NeoPixel.*Accessed: 2017.*

[5] *MQTT Client Library Encyclopedia – Paho Android Service.* https://www.hivemq.com/blog/mqtt-client-library-enyclopedia-paho-android-service.*Accessed: 2017.*

[6] *Introducción a MQTT (Sofia2 IoT Platform Blog).* https://about.sofia2.com/2015/09/05/introduccion-a-mqtt-y-uso-en-sofia2/.*Accessed: 2017.*

[7] *ESP-12F ESP8266 Wifi Board* http://www.electrodragon.com/w/ESP-12F_ESP8266_Wifi_Board.*Accessed: 2017.*

[8] *ESP8266 ¿Qué módulo elegir?* https://polaridad.es/esp8266-modulo-wifi-elegir-caracte *Accessed: 2017.*

[9] *WS2812 VS WS2812B.* https://acrobotic.com/datasheets/WS2812B_VS_WS2812.pdf.*Accessed: 2017.*

[10] *ESP8266 Community Forum.* http://www.esp8266.com/index.php?sid=304a2774506efaf7142ca3da7103388d.*Accessed: 2017.*

[11] *Datasheet: ESP-12F WiFi Module. Version 1.0* https://www.elecrow.com/download/ESP-12F.pdf.*Accessed: 2017.*

[12] *MQTT, Adafruit IO and You!.* `https://learn.adafruit.com/mqtt-adafruit-io-and-you/overview`.*Accessed: 2017.*

[13] *Mosquito Documentation* `https://mosquitto.org/documentation/`.*Accessed: 2017.*

[14] *Android Studio User guide. Conoce Android Studio.* `https://developer.android.com/studio/intro/index.html?hl=es-419`.*Accessed: 2017.*