UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIONES

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS
ESPECÍFICAS DE LAS TELECOMUNICACIONES
MENCIÓN EN SISTEMAS ELECTRÓNICOS

Sistema electrónico de apertura de puerta de garaje desde terminal smartphone

---

Autora:

Icíar Pascua Maestro

Tutor:

Jesús Manuel Hernández Mangas

Agradecimientos

En primer lugar agradecer a mi tutor Jesús M. Hernández Mangas. No sólo por guiarme y apoyarme en este proyecto, también por la pasión con la que transmite sus conocimientos en las clases de sistemas electrónicos, por su ayuda y sus consejos.

A todos los profesores de la escuela de Ingenieros de Telecomunicaciones por su dedicación a la hora de enseñar y por aportarme todos los conocimientos y aptitudes necesarias para poder ser un buen ingeniero.

A mi familia, por todo su apoyo durante estos años de carrera y por no dejar nunca de animarme.

A mis amigos, a todos, por haber compartido conmigo esta experiencia y haberme apoyado y ayudado en ella. Por haber compartido conmigo tantos momentos y siempre estar dispuestos a ayudar con cualquier problema.

Resumen

La rápida expansión de la tecnología lleva a las personas a incorporarla a ciertos aspectos de su vida cotidiana. Con el desarrollo de este proyecto se busca incorporar las nuevas tecnologías a la apertura de puertas de garaje.

Para ello se desarrolla un sistema que permite a través de un servidor Web recibir las peticiones que realiza el usuario desde su terminal Smartphone, y que con esta información se comunica a través de un enlace vía radio con el dispositivo de control situado en la puerta permitiendo que se ejecute la acción requerida por el usuario (apertura o cierre).

Palabras clave: ESP, transceptor, servidor web, decodificador, codificador.

Abstract

The rapid expansion of technology leads people to incorporate it into certain aspects of their daily lives. With the development of this project, the aim is to incorporate new technologies to the opening of garage doors.

For this, a system is developed that allows through a Web server to receive the requests made by the user from his Smartphone terminal, and with this information is communicated through a radio link with the control device located in the door allowing that the action required by the user (opening or closing) is executed.

Palabras clave: ESP, transceiver, web server, decoder, encoder

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción.

1.1. Objetivos

En la actualidad, las nuevas tecnologías se están incorporando a la vida cotidiana, a través de los terminales Smartphone este hecho se masifica aún más. Siguiendo esta corriente, con este proyecto se quiere dar la facilidad al usuario de poder controlar la puerta de su garaje desde su smartphone sin necesidad de ningún mando.

El proyecto consiste en el diseño de un dispositivo que permita realizar el manejo remoto de puertas de garaje sin necesidad de sustituir el mecanismo de apertura ya existente en la puerta. De tal manera que, una vez implantado, se pueda emplear para realizar la apertura de la puerta tanto el mando antiguo como el smartphone. Por lo tanto, el dispositivo ha de ser capaz de almacenar y descifrar el código maestro que emita la llave empleada normalmente por el usuario y a partir de este código, generar un mensaje compatible cada vez que se indique a través de una aplicación o página web, que se desea realizar una maniobra con la puerta.

1.2. Fases del proyecto

Con el fin de cumplir el objetivo del proyecto, este se divide en diferentes fases. Gracias a esta división conseguimos que el proyecto sea mas fácilmente abordable. A continuación, se listan las diferentes fases a realizar en el desarrollo del proyecto.

Realización de un sistema electrónico de apertura de garaje	
Fase 1: Especificación del proyecto	1.1 Realización de un estudio de mercado
	1.2 Definición del funcionamiento detallado del proyecto
	1.3 Búsqueda y selección de componentes
Fase 2: Diseño electrónico	2.2 Creación de componentes
	2.3 Posicionado de componentes
	2.4 Análisis de potencia consumida
	2.5 Rutado de las conexiones
Fase 3: Diseño de la placa de circuito impreso	3.1 Creación de huellas
	3.2 Posicionado de componentes
	3.3 Análisis de potencia consumida
	3.4 Rutado de las conexiones
	3.5 Verificación y generación de los ficheros Gerber
Fase 4: Fabricación de la placa de circuito impreso	4.1 Envío al fabricante
	4.2 Montaje
Fase 5: Realización del firmware	5.1 Programa principal
	5.2 Sincronización de mandos existentes
	5.3 Verificación y corrección de errores
Fase 6: Interconexión global del sistema	
Fase 7: Verificación del sistema	
Fase 8: Documentación final del proyecto	

Tabla 1.1: Fases requeridas para funcionamiento correcto del proyecto.

Para tener una visión general de como están organizadas estas fases, se realiza una estimación de la duración de las mismas. Y una vez que se tiene esta estimación se realiza un diagrama de Gannt, lo que nos permitirá ver de una forma muy sencilla y gráfica la evolución de las fases anteriormente enumeradas durante el tiempo. Se puede observar este diagrama en la figura 1.1, el diagrama consta de dos partes, a la izquierda se enumeran las fases a implementar en orden cronológico y a la derecha se introduce mediante barras temporales la duración de las mismas.

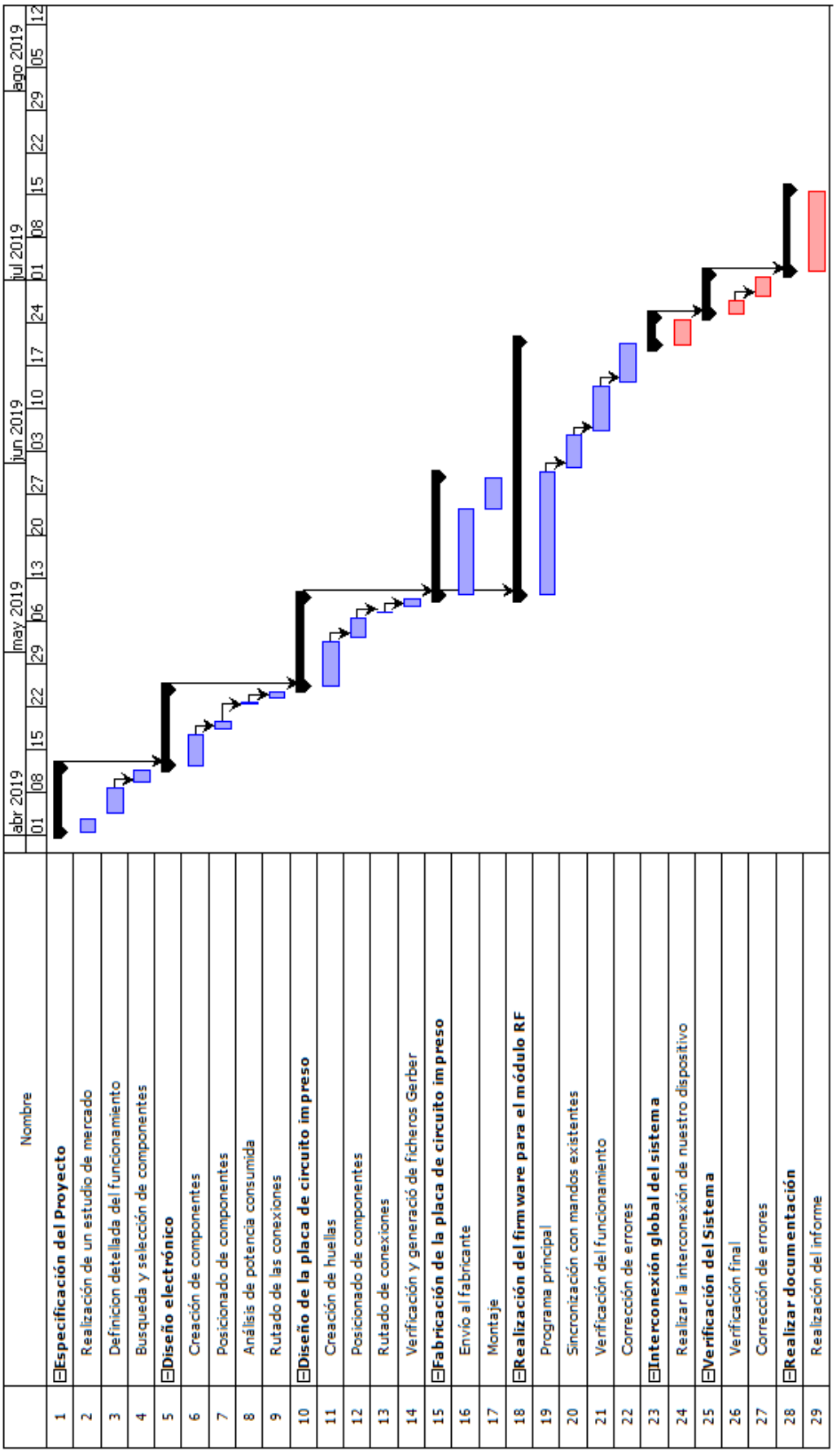

Figura 1.1: Diagrama de Gannt

1.3. Estructura del documento

En los capítulos del presente documento, se procederá a detallar las fases descritas anteriormente. Así como los conocimientos adquiridos y elementos necesarios, tanto software como hardware para realizar cada una de ellas.

El primer capítulo como ya hemos visto es meramente introductorio, en el segundo capítulo se desarrolla en profundidad las especificaciones técnicas del proyecto y como se ha llegado a ellas, así como el hardware necesario para desarrollar esas especificaciones.

El tercer capítulo muestra y describe los pasos seguidos a la hora de realizar el diseño electrónico y la captura esquemática de el sistema. El capítulo cuatro, tiene un esquema similar al tres solo que en esta ocasión se trata de los pasos seguidos a la hora de realizar la placa de circuito impreso .

En el quinto capítulo, se describen las características que deberá poseer el firmware del sistema para cumplir los objetivos, así como cual será el diagrama de actuación del dispositivo. En el siguiente capítulo se muestran las pruebas realizadas para verificar el correcto funcionamiento del dispositivo, así como una prueba de funcionamiento global, incorporando otro dispositivo (con el mismo diseño electrónico que la placa inicial pero con un código diferente) que haga de receptor.

Por último, se han introducido las conclusiones finales extraídas de la realización del proyecto, así como unas líneas futuras en el desarrollo del mismo. También, se introducen al final del documento unos apéndices en los que se puede observar el código del prototipo y del dispositivo receptor.

Capítulo 2

Especificación técnica del proyecto.

2.1. Estudio de mercado

En primer lugar, se necesita conocer cuáles son las frecuencias y tecnologías más empleadas por los fabricantes de llaves de garaje, para que de esta manera el dispositivo sea compatible con un mayor número de puertas y receptores. Para ello, se realiza un estudio de mercado y se examinan 40 mandos de garaje diferentes. Se realiza una comparativa en dos de los parámetros principales de los mandos, la banda de frecuencia empleada y el modo de sincronización con el receptor. Los mandos se agrupan en tres bandas en lo que respecta a frecuencia de emisión, estas son 40.685MHz, 433MHz y 866Mhz.

La otra clasificación realizada como ya se ha mencionado, se centra en la forma en la que los mandos consiguen sincronizarse para poder ponerse en funcionamiento, en esta categoría también encontramos tres grandes grupos: Switch, grabación en el receptor y autoaprendizaje.

La tecnología de Switch codifica la clave maestra del mando mediante un conjunto de microinterruptores situados dentro del mando (El número de microinterruptores empleados puede variar). Con este tipo de mandos el mensaje que se transmite en las ordenes de apertura y cierre es siempre el mismo por lo que con escuchar un mensaje ya se podría replicar la comunicación.

En cuanto a los mandos que emplean grabación en el receptor, el control se realiza grabando con códigos aleatorios predefinidos los dispositivos, en el momento de la fabricación. Si se desea generar nuevos códigos aleatorios se debe ejecutar un procedimiento de RESET que permita llevar al estado de fábrica el dispositivo y a continuación volver

a grabar el código. Si se quiere obtener una copia de los códigos de transmisión, es requisito indispensable que los mandos sean del mismo modelo. La función de copia se realiza conectando los dos mandos mediante un cable suministrado. Este tipo de mandos suelen disponer de la opción de conectarlos a un programador.

Los mandos con autoaprendizaje, permiten copiar el código que emite el mando maestro cuando se le selecciona la opción de programación. Para seleccionar esta opción es necesario realizar una serie de pasos, los cuáles son diferentes para cada marca de mandos y se encuentran en las instrucciones de uso de estos.

En la tabla 2.1, se puede observar una comparativa con los 40 mandos consultados. Observando los resultados obtenidos, se toma la decisión de que la emisión del sistema sea a 433MHz únicamente y así evitar tener que hacer compatibilidad para varias bandas, ya que la gran mayoría de los dispositivos actuales (el 82 %) emplean la frecuencia de 433MHz.

En cuanto a la forma de aprendizaje, se decide emplear el autoaprendizaje ya que aunque esta casi igualado al de grabación en el receptor resulta más sencillo para el usuario final.

Resumen Final:

Total de mandos que emplean como frecuencia:	433MHz	40.685MHz	868MHz
	41	4	9
Total de mandos que emplean como forma de aprendizaje:	Switch	Grabación en el receptor	Autoaprendizaje
	9	20	21

Figura 2.1: Resumen final de la comparativa de mercado para 50 mandos.

Marca	Modelo	Frecuencia de emisión			Forma de aprendizaje		
		433MHz	40.685 MHz	868MHz	Switch	Grabación en el receptor	Autoaprendizaje
Clemsa	Mastercode MV-12	✓			✓		
HR	RQ F2		✓				✓
HÖRMANN	HSE 4 BS			✓			✓
SEAV	BEFREE S3 NEW	✓				✓	
HY-DOM	NEO 2	✓				✓	
KING-GATES	STYLO 4K						
CELINSA	ME 3	✓			✓		
FAAC	XT4 868 SLH Black			✓			✓
ERREKA	RESON 2	✓			✓		
BFT	Mitto B RCB TX4	✓				✓	
PRASTEL	TC4E	✓					✓
CYACSA	NEO-20	✓				✓	
SICE	WHY2 EVO	✓	✓	✓			✓
DELMA	YOUNG 4ch	✓				✓	
CAME	T434	✓					✓
MOTORLINE	MX4SP RCA	✓				✓	
DOITRAND	TS4RE	✓				✓	
COMPATIBLE	CHAMBERLAIN	✓				✓	
TOKO	TO40TX-4MS		✓		✓		
BERNER	BHS221			✓		✓	
TEDSEN	SKX4WD	✓			✓		
LEB	SMARTY 433	✓					✓
DOITRAND	TS4RE	✓			✓		
MOTOSTAR	C4	✓					✓
PROGET	ETY-4N	✓			✓		
B&B	BUGGY-F	✓					✓
SOMMER	4022 TX02-434-2	✓				✓	
SENTINEL	SG LINE	✓					✓
NICE	FLO2R-S	✓				✓	
NICE	FLO1	✓			✓		
MHOUSE	GTX4	✓					✓
PRASTEL	TC4E	✓				✓	
JCM	GO PRO MINI IMEBA			✓			✓
NICE	ON2E	✓				✓	
MOTORLINE	FALC RCM	✓				✓	
FADINI	VIX 53			✓			✓
TELECO	TXP-433-A01	✓				✓	
EXTEL	ATEM 5	✓					✓
GIBIDI	Domino	✓					✓

Tabla 2.1: Comparativa de 40 mandos.

2.2. Estudio del sistema

Para poder realizar la apertura de puertas, el dispositivo debe de disponer de una interfaz RF funcionando en la banda de frecuencia de los 433MHz, mientras que para que el usuario desde su Smartphone pueda conectarse a este debe incluir también una interfaz Wi-Fi.

Además, el dispositivo necesita para su correcto funcionamiento disponer de un microcontrolador, codificadores y decodificadores para poder generar la codificación requerida en la trasmisión RF. Así como un transceptor que permita modular la señal para su posterior emisión y demodularla para su correcta recepción.

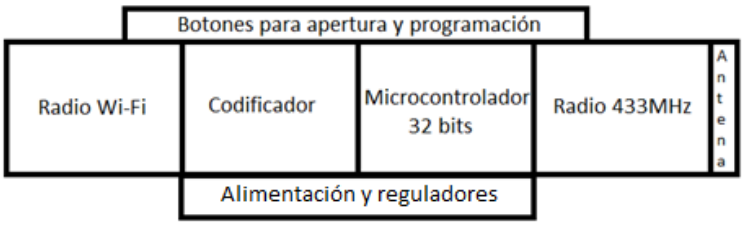

Figura 2.2: Esquema sobre el dispositivo a realizar

Para llevar a cabo la elección del microcontrolador se tiene en cuenta que se debe implementar una interfaz Wi-Fi, por lo que se decide escoger un micro que tenga ya integrado esta característica, este es el ESP-12F.

El codificador y decodificador son necesarios ya que los mandos actuales para evitar la grabación del código mediante una escucha, incluyen un sistema de código evolutivo o rolling code. El código evolutivo emplea un número de secuencia cambiante y una clave para encriptar los datos. Esta clave es conocida tanto como por el emisor como por el receptor.

El funcionamiento del emisor consiste por lo tanto en que, por cada pulsación genera un mensaje en el que trasmite la operación a realizar y un número de secuencia, este número se aumenta cada vez que se trasmite un mensaje diferente. El mensaje, antes de transmitirse se encripta con la clave, la cuál ha sido grabada en el emisor en el momento

de la fabricación.

En el receptor se produce el proceso contrario, se desencripta el mensaje con la clave y se comprueba si la secuencia se encuentra en un marco aceptable de sincronización con el contador propio, que va aumentando a cada mensaje que recibe. Si es así, se realiza la operación contenida en el mensaje.

Dado que el sistema debe ser capaz tanto de emitir como de recibir, se decide incorporar al circuito un transceptor, que nos permite disponer en un sólo circuito integrado de las funcionalidades de un emisor y un receptor RF. Se consulta a diversos fabricantes de mandos (Erreka y JMA entre ellos) y se obtiene la información de que en la actualidad la gran mayoría de los mandos emplean la modulación ASK (Amplitude-shift keying). Esta es una modulación en amplitud, de tal manera que en ella se representan los datos digitales como variaciones de amplitud de la señal portadora, más concretamente como presencia o ausencia de señal. Manteniendo la frecuencia y fase de la portadora constante. En la figura 4 podemos observar un ejemplo de modulación ASK.

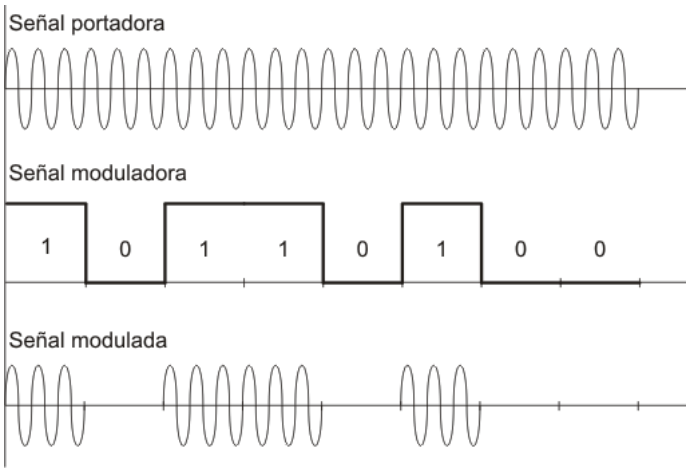

Figura 2.3: Ejemplo de modulación ASK.

Por lo tanto para cumplir con esta especificación, se requiere que el transceptor permita modular en ASK a una frecuencia de 433MHz, se escoge el transceptor **CC1101** , por su gran versatilidad y su bajo consumo.

2.3. Componentes empleados

2.3.1. ESP-12F

El ESP-12F es el cerebro del sistema, es un módulo Wi-Fi desarrollado por Ai-thinker Team, su procesador central es el ESP8266. Este procesador es de 32 bits, bajo consumo y permite velocidades de reloj de 80MHz y 160MHz, posee una antena on-board, es compatible con sistemas RTOS (Real Time Operating Sistem), dispone de Wi-Fi integrado y de un LNA (Low Noise Amplifier.)

En cuanto a la memoria, dispone de una RAM de 36kB y una memoria flash externa con comunicación SPI de 4MB. Este módulo es compatible con los estándares IEEE802.11 b/g/n, y tiene integrado por completo la pila de protocolo TCP/IP.

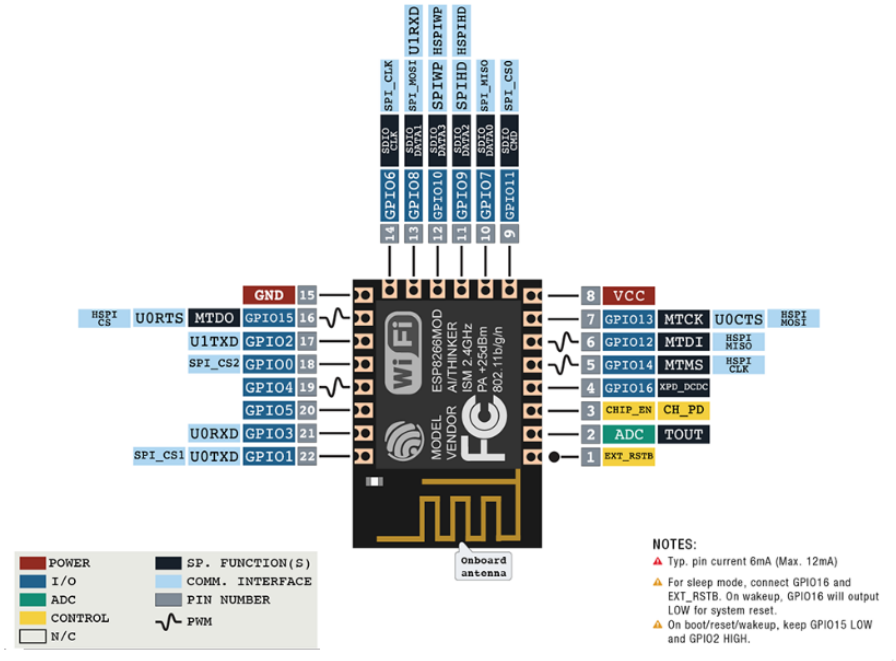

Figura 2.4: Pin Out del módulo ESP12-F.

Posee diversas interfaces de comunicación, entre ellas SPI, HSPI, I2C, comunicación serie a través de una UART, interfaz dedicada a la conversión analógica-digital. El ESP12-F puede ser programado desde el entorno de Arduino, el cuál dispone de una serie de librerías que facilitan el proceso, algunas de ellas como la de la comunicación SPI es compartida con la de Arduino, esta requiere una pequeña modificación, ya que hay que cambiar la definición de los pines que emplea como MOSI, MISO Y SCK, para que se pueda emplear

de forma satisfactoria.

Para programar el ESP12-F emplearemos la interfaz serie, conectada a un conversor que transforme la comunicación USB en comunicación serie asíncrona. En este caso se elige el FT23RL.

2.3.2. FT23RL

El FT23RL, es un circuito integrado que permite convertir una interfaz USB a una UART. Permite eliminar, en la mayoría de los casos la necesidad de emplear un driver USB. Su velocidad de transferencia de datos va desde 300 baudios hasta 3 Megabaudios en comunicaciones RS422/RS485 y de 300 baudios a 1 Megabaudio para RS232. Dispone de un buffer de recepción de 256 bytes, y un buffer de transmisión de 128 bytes.

Permite conectarse a un microprocesador para realizar la carga en el dispositivo del programa que deberá ejecutar y mantener una comunicación con un puerto serie para depurar el código.

Figura 2.5: Circuito integrado FT232RL.

En el proyecto, la comunicación FT232RL-ESP12-F se realiza mediante el protocolo de comunicación serie asíncrona, para realizar esta comunicación se dispone de dos líneas denominadas TXD y RXD. Este tipo de comunicación es bidireccional y consiste en la trasmisión serie de palabras de datos multi-bit en intervalos indeterminados donde cada palabra tiene un número de bits y un tiempo de bit fijo. Sin embargo, el intercambio de datos entre el USB y el FT232RL se hace mediante el envío diferencial de datos con las líneas D+ y D-.

2.3.3. HCS300

El HCS300 es un codificador Rolling Code (de código cambiante) que emplea la tecnología **Keeloq**, la cuál es propiedad de Microchip, esta es una de las más conocidas de las codificaciones Rolling Code. El HCS300 combina un código cambiante de 32 bits encriptado, con un número de serie de 28 bits y 6 bits de información, para dar lugar a una palabra de 66 bits. Para realizar la encriptación del código cambiante emplea una Crypt Key generada a partir de un número de serie.

Podemos observar el proceso de generación de una palabra en la figura 2.5.

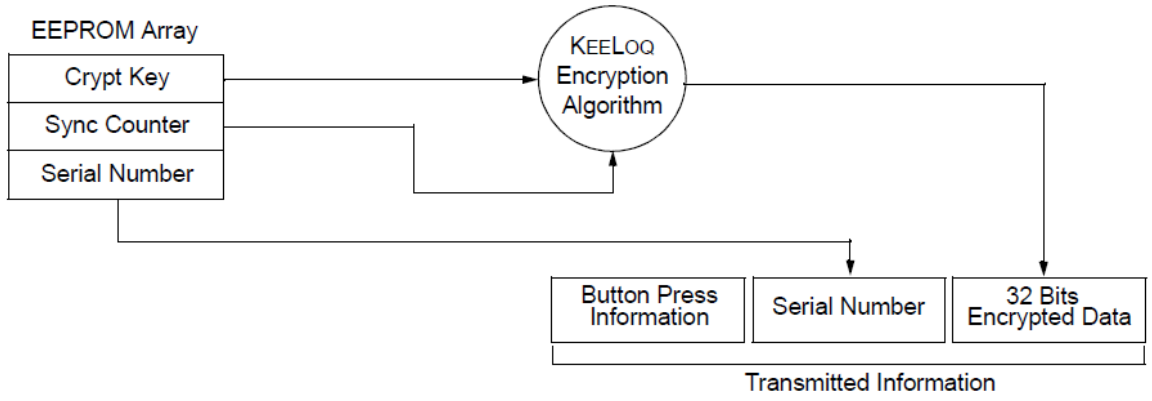

Figura 2.6: Generación de la palabra a transmitir por el HCS300.

El codificador genera como se ha mencionado, un mensaje compuesto por dos partes una encriptada y otra sin encriptar. En la parte encriptada, va situado un contador de sincronización de 16 bits, este es el motivo de que cada mensaje transmitido sea diferente, ya que el contador de sincronización se incrementa cada vez que un mensaje es generado.

El funcionamiento del códificador HCS300 sera el siguiente: se despertará al detectar una pulsación en alguna de las 4 entradas de datos que dispone, leerá todas las entradas, aumentará el contador de sincronización y generará el siguiente mensaje:

La parte fija del mensaje está formada por dos bits de status, el contenido de las entradas y el numero de serie del codificador que emite el mensaje. La parte encriptada del mensaje, se genera con el contenido de las entradas, 12 bits de dsicriminación que suelen ser los 12 bits menos significativos del número de serie y 16 bits del contador de sincronismo.

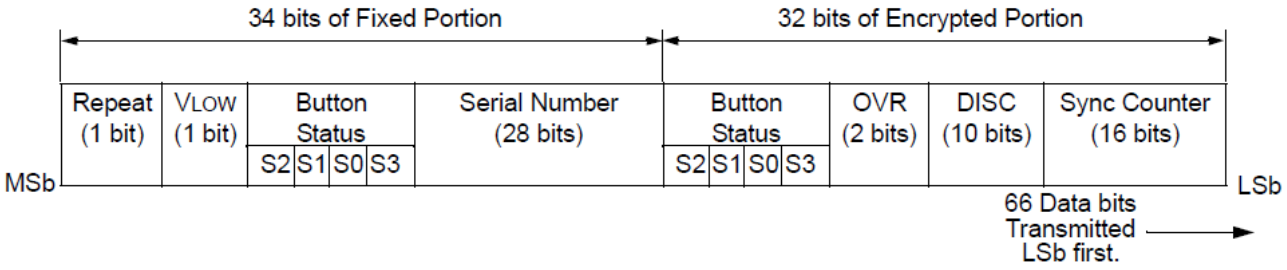

Figura 2.7: Palabra de datos transmitida por el HCS300.

Cada palabra transmitida contiene un preámbulo, una cabecera, los 34 bits de datos fijos, los 32 bits de parte codificada y un periodo de guarda en el cuál no se debe emitir nada. Los datos tanto fijos como encriptados, van modulados en PWM de tal manera que el resultado final de la palabra emitida por el HCS300 es el que nos muestra la figura 2.7.

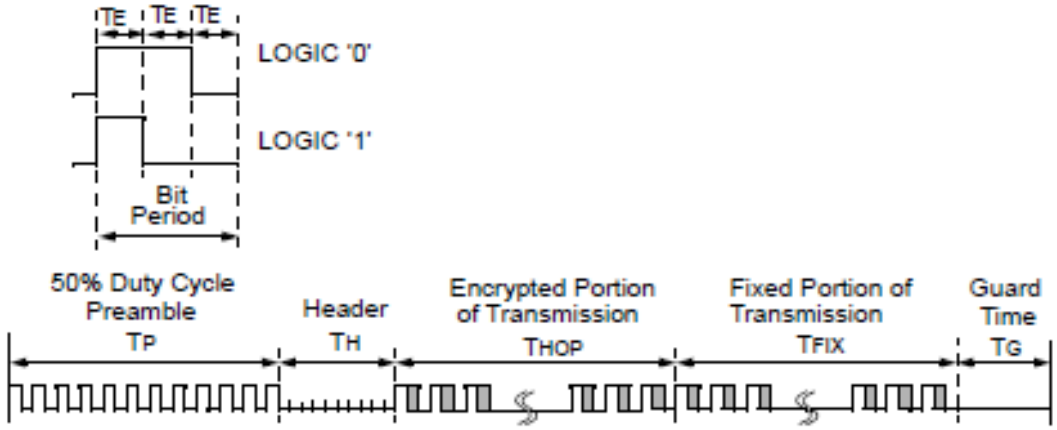

Figura 2.8: Formato de palabra transmitida por el HCS300.

Es necesario pasar la palabra emitida por el HCS300 al microcontrolador, para ello se debe realizar una trasmisión asíncrona de tal manera que, la línea S2 se mantenga en alta hasta que el codificador acepte subiendo la linea de PWM, a continuación se sitúa en el resto de lineas la combinación que se quiere transmitir y por ultimo empleando la línea de S2 como reloj se van extrayendo bits del codificador. La frecuencia del reloj empleado, no debe exceder los 20KHz.

Al realizar este proceso para obtener el código, el codificador añade a la palabra de datos 16 bits reservados. Estos bits pueden ser eliminados de la trama a la hora de realizar

la transmisión, ya que no aportan información útil para el decodificador.

2.3.4. HCS500

HCS500 es un decodificador compatible con el codificador escogido para la realización del proyecto. Para funcionar conjuntamente con un codificador, debe aprender antes de él la crypt key, el número de serie del transmisor y el contador de sincronización. Estos valores los almacena en una EEPROM externa, de tal manera que un solo decodificador puede actuar con varios codificadores.

El HCS500 puede conectarse a un microcontrolador mediante una comunicación serie síncrona, de tal manera que compartiendo dos lineas (reloj y datos), el microcontrolador puede efectuar diversas operaciones sobre el decodificador. Las operaciones que se pueden llevar a cabo son :

- Lectura de un byte de la EEPROM.
- Escritura de un byte en la EEPROM.
- Activar el procedimiento de aprendizaje.
- Borrar todos los códigos aprendidos por el decodificador.
- Programar el byte de configuración y el código de fabricación.

El procedimiento de aprendizaje por el cual el decodificador es capaz de trabajar conjuntamente con un codificador es vital para este proyecto, ya que se necesita almacenar la información que emite la lleve original y desencriptarla para emitir después sincronizados con el receptor ya existente.

El comando de activación del modo aprendizaje está constituido de varias partes, una secuencia de activación para trabajar en modo comando, el byte de ese comando (0xD2) y a continuación dos dummy bytes (es indiferente el valor que se les dé). Una vez seguido este proceso el integrado pondrá la línea de datos en alta, para aceptarlo.

Una vez que el decodificador se encuentra en modo aprendizaje, está esperando a que le llegue una trama válida. Al recibir una trama válida, enviará al microcontrolador a través de la linea de datos un mensaje de estado. En este momento se deberá repetir el proceso y volver a enviar una trama válida al decodificador. Con esta información será

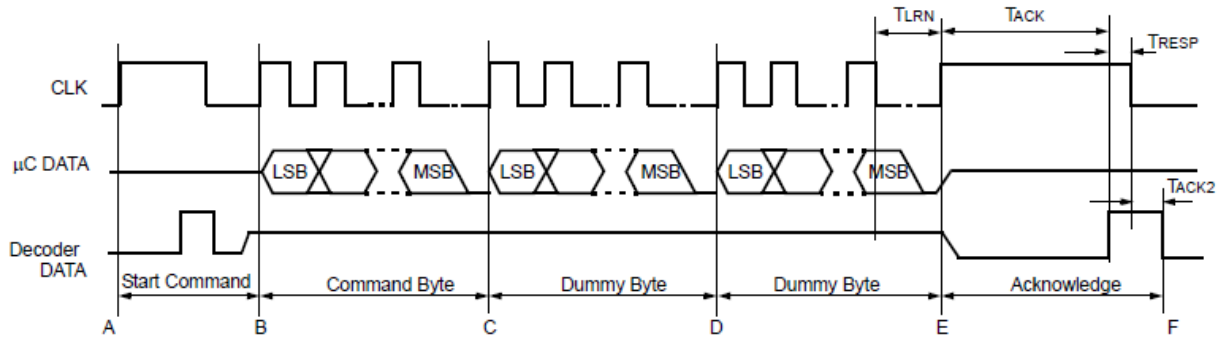

Figura 2.9: Activación del modo aprendizaje.

capaz de generar la llave, el número de serie y el valor del contador de sincronización. A continuación, guardará estos datos en la memoria externa y enviará al microcontrolador un mensaje de estado con la palabra que ha recibido decodificada.

2.3.5. CC1101

El CC1101 es un transceptor de bajo coste, que trabaja a frecuencias por debajo de 1GHz. Está diseñado para aplicaciones inalámbricas de muy bajo consumo, sus principales bandas de funcionamiento son las frecuencias de 315, 433, 868 y 915 MHz, pero puede fácilmente ser programado para su funcionamiento en otra, dispone de una velocidad de datos configurable de hasta 600kbps. El dispositivo permite ser controlado mediante una interfaz SPI y dispone de diversos tipos de modulaciones entre ellas 2-FSK, 4-FSK, GFSK y OOK.

La comunicación con el MCU será a través de la interfaz SPI, a través de ella podremos configurar numerosos registros de configuración y acceder a las FIFO de trasmisión y recepción. El transceptor, dispone de multitud registros de configuración con los que poder elegir, la frecuencia de emisión, la modulación empleada, el ancho de banda de cada canal, la tasa de datos, el ancho de banda del filtro de recepción. Como estos parámetros son muy números el propio fabricante (Texas Instrument), recomienda emplear el programa Smart RF Studio, para generar unos valores óptimos para los registros de configuración.

Se puede observar una captura de este programa en la figura 2.10, dispone de varios campos en los que introducir las características deseadas para el transceptor y a su derecha nos muestra una tabla con los registros de configuración y los valores adecuados para

cada uno de estos registros basándose en las características seleccionadas.

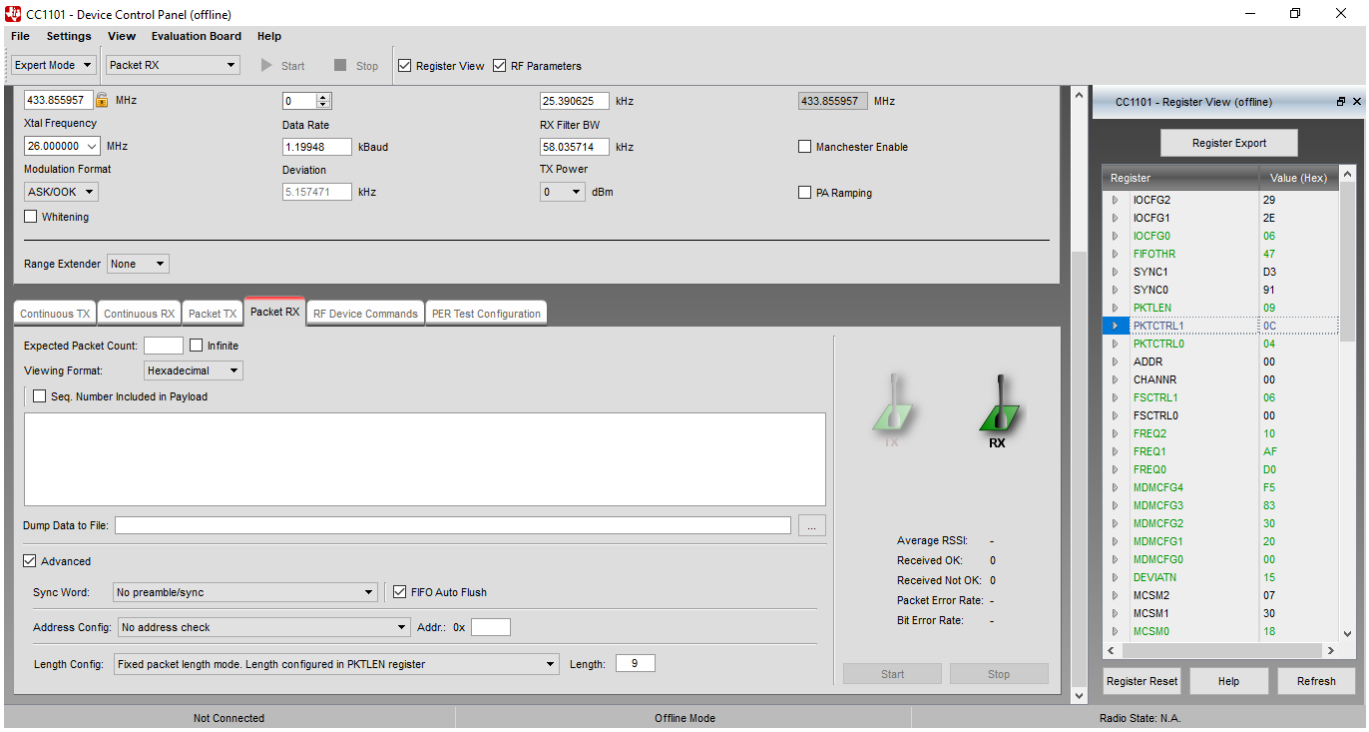

Figura 2.10: Captura del programa Smart RF Studio, resaltados en verde los registros a modificar.

Es importante destacar, que la configuración de registros generada en el programa Smart RF Studio, está optimizada para modulaciones 2-FSK, GFSK, MSK. Sin embargo para la modulaciones OOK y ASK estos registros pueden resultar en una recepción no óptima o incluso inestable. Es necesario, modificar algunos registros dependiendo de las características de la señal recibida, tal y como nos indica la nota de aplicación DN022.[4]

Los valores finales que se deben situar en los registros de configuración del transceptor se pueden observar en la tabla 2.3, en esta tabla solo se muestran los valores finales de aquellos registros cuyo valor difiere del valor por defecto.

El módulo CC1101, funciona como una máquina de estados. Posee un estado IDLE en el cuál el consumo es reducido. De este estado se pasa a los estados de trasmisión(TX) y recepción(RX) mediante las llamadas “Command Strobes”, estas se envían a través de la interfaz SPI. A diferencia de los mensajes para la modificación de los registros por SPI, en la que se tiene que enviar la dirección del registro y el valor que tomará, en este tipo de órdenes basta solo con enviar la dirección. En la Tabla 2.2 se pueden observar las

diferentes "Strobes" disponibles para el CC1101.

Address	Register	Description	Details on page number
0x30 (0xF0)	PARTNUM	Part number for CC1101	92
0x31 (0xF1)	VERSION	Current version number	92
0x32 (0xF2)	FREQEST	Frequency Offset Estimate	92
0x33 (0xF3)	LQI	Demodulator estimate for Link Quality	92
0x34 (0xF4)	RSSI	Received signal strength indication	92
0x35 (0xF5)	MARCSTATE	Control state machine state	93
0x36 (0xF6)	WORTIME1	High byte of WOR timer	93
0x37 (0xF7)	WORTIME0	Low byte of WOR timer	93
0x38 (0xF8)	PKTSTATUS	Current GDOx status and packet status	94
0x39 (0xF9)	VCO_VC_DAC	Current setting from PLL calibration module	94
0x3A (0xFA)	TXBYTES	Underflow and number of bytes in the TX FIFO	94
0x3B (0xFB)	RXBYTES	Overflow and number of bytes in the RX FIFO	94
0x3C (0xFC)	RCCTRL1_STATUS	Last RC oscillator calibration result	94
0x3D (0xFD)	RCCTRL0_STATUS	Last RC oscillator calibration result	95

Tabla 2.2: Strobes disponibles en el CC1101.

En lo que respecta a la transmisión y recepción de mensajes, para enviar paquetes a través del transceptor, estos deben escribirse en la FIFO de transmisión de la que dispone el transceptor y a continuación activar el modo transmisión. Para recibir paquetes, se debe activar el modo recepción y una vez hecho esto todos los bytes que detecte el transceptor serán almacenados en la FIFO de recepción. A ambas memorias, la de transmisión y la de recepción, se pude acceder mediante la comunicación SPI y el microcontrolador será el encargado de vigilar que no se desborden.

Estas memorias FIFO de transmisión y recepción no son de tamaño fijo, es decir, se disponen de 64 bytes de almacenamiento para ambas partes y se permite fijar mediante el registro FIFOTHR el número de bytes que se pueden almacenar en cada una ellas.

Name	Address	Value	Description
IOCFG0	0x0002	0x06	GDO0 Output Pin Configuration
FIFOTHR	0x0003	0x47	RX FIFO and TX FIFO Thresholds
PKTLEN	0x0006	0x1C	Packet Length
PKTCTRL0	0x0008	0x04	Packet Automation Control
FREQ2	0x000D	0x10	Frequency Control Word, High Byte
FREQ1	0x000E	0xEB	Frequency Control Word, Middle Byte
FREQ0	0x000F	0x85	Frequency Control Word, Low Byte
MDMCFG4	0x0010	0xC6	Modem Configuration
MDMCFG3	0x0011	0xAE	Modem Configuration
MDMCFG2	0x0012	0x32	Modem Configuration
MDMCFG1	0x0013	0x00	Modem Configuration
AGCCTRL2	0x001B	0x07	AGC Control
AGCCTRL1	0x001C	0x00	AGC Control
AGCCTRL0	0x001D	0x92	AGC Control
FREND0	0x0022	0x11	Front End TX Configuration
FSCAL3	0x0023	0xE9	Frequency Synthesizer Calibration
FSCAL2	0x0024	0x2A	Frequency Synthesizer Calibration
FSCAL1	0x0025	0x00	Frequency Synthesizer Calibration
FSCAL0	0x0026	0x1F	Frequency Synthesizer Calibration
TEST2	0x002C	0x81	Various Test Settings
TEST1	0x002D	0x35	Various Test Settings
TEST0	0x002E	0x09	Various Test Settings

Tabla 2.3: Valor de los registros de configuración del transceptor CC1101.

2.3.6. Otros

Además de los componentes principales ya descritos, se decide introducir un expansor, en concreto el MCP23S08-E/SS, que nos permitirá disponer de más entradas y salidas de propósito general de las que disponemos en el ESP12-F, concretamente 8 más. La comunicación con este integrado se realizará a través del bus SPI y podrá adquirir una velocidad de hasta 10MHz.

Se introducirá el regulador de voltaje TC1262 que permite disponer en la placa de 3.3V, lo cuál es de gran utilidad para alimentar tanto el ESP-12F, como el transceptor, el codificador y decodificador. Este regulador es capaz de suministrar una corriente de salida de 500mA.

Por último, se incorporará al diseño un botón que permita seleccionar de forma manual y sin recurrir a la aplicación abrir o cerrar la puerta desde el mismo dispositivo. Este botón se conectará a uno de las entradas de propósito general del ESP del tal manera que cuando se detecte un cambio en el estado de ese pin se proceda a enviar el mensaje de apertura o cierre.

Capítulo 3

Diseño electrónico y captura esquemática.

3.1. Asignación de pines requeridos

Un aspecto a tener en cuenta y aclarar desde un primer momento, son los pines necesarios del microcontrolador. Como ya se ha comentado, el microcontrolador que se va a emplear es el ESP-12F, este dispone de 13 pines de propósito general, un pin para el conversor AD, 4 dedicados a la realización de una comunicación SPI. Todo está controlado por un /RESET común.

PINES		
NOMBRE	TIPO/PERIFÉRICO	FUNCIÓN
P1	/RST	
P2		
P3	DE HABILITACIÓN	EN
P4	DECODE MODULE	DEC_CLK
P5	SPI	SCK
P6	SPI	MISO
P7	SPI	MOSI
P8	ALIMENTCIÓN	
P9		
P10		
P11		
P12		
P13		
P14		
P15	ALIMENTACIÓN	
P16		
P17		
P18	BOOTLOADER	
P19	DECODE MODULE	DEC_DATA
P20	SPI	CS
P21	UART	TXD
P22	UART	RXD

Tabla 3.1: Resumen final sobre la funcionalidad de los pines en el ESP12-F.

Debemos tener en cuenta, que los pines dedicados para la comunicación SPI, así como

dos pines de propósito general no se pueden emplear en esta versión del micro. Debido a esto, deberemos emplear los pines de HSPI para realizar la comunicación SPI, limitándonos el número de pines de propósito general que podemos emplear.

Como se quiere implementar una comunicación serie asíncrona a mayores de la comunicación SPI, debemos reservar dos pines (TX y RX) para realizarla. El desglose de los pines requeridos, y la función que a desempeñar cada uno de ellos podemos observarla en la tabla 3.1.

3.2. Diseño electrónico

Para la realización del proyecto, se utilizará el software Proteus versión 8.6. Con esta herramienta se realizará tanto el esquemático del dispositivo, como la creación de nuevos componentes que Proteus no contiene en sus librerías, permitirá también realizar el posicionado de componentes y el rutado de las pistas necesarias para nuestro dispositivo. Con este software podremos también realizar un BOM (Bill Of Materials), que nos será útil para hacer una estimación de costes del proyecto.

Para obtener la placa de circuito impreso, el primer paso a seguir es el diseño de un esquemático. Se comienza creando una página en el esquemático e introduciendo en ella el componente principal del proyecto, es decir, el módulo ESP-12F. En esta misma página se introducen tanto sus condensadores de desacoplo, como el sistema de RESET del dispositivo.

Se debe tener en cuenta, que alguno de los pines del dispositivo tienen que tener un valor especial en el arranque, estos son GPIO15, GPIO2 Y GPIO2. En la siguiente tabla se observan los valores necesarios.

PIN MODE FOR			
	GPIO15	GPIO0	GPIO2
UART BOOT	0	0	1
FLASH BOOT	0	1	1

Tabla 3.2: Valores predeterminados para el arranque.

Si se observa la figura 3.1, se puede comprobar el circuito necesario para el correcto funcionamiento del micro. Se puede observar, que el micro dispone de una pata para la habilitación, esta debe conectarse a la alimentación a través de una resistencia de 10k si se quiere que el dispositivo esté habilitado y funcionando de forma correcta.

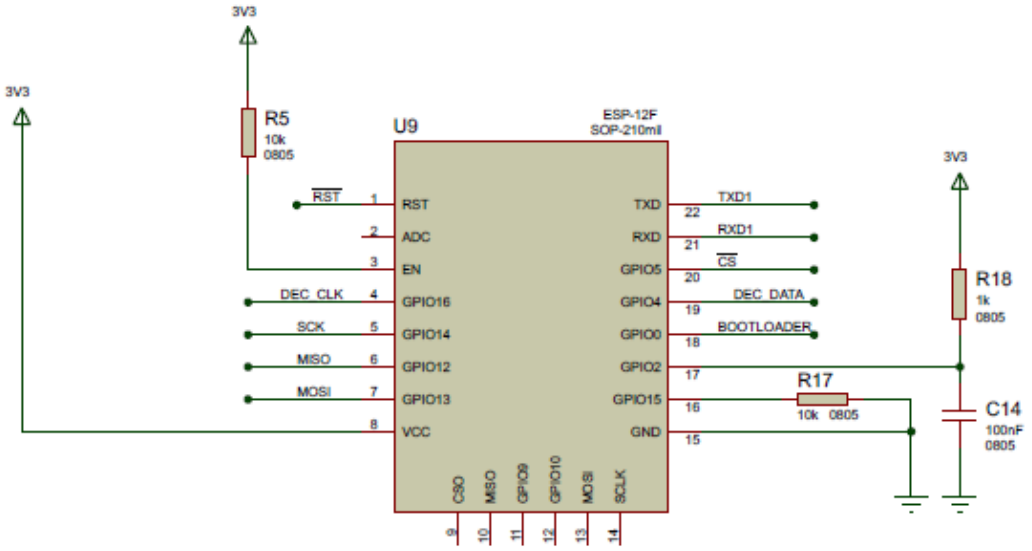

Figura 3.1: Circuito del microcontrolador WiFi.

En la primera página del esquemático (figura 3.3), se introducen también unos divisores de voltaje, esto se debe a que en el diseño hay líneas a diferentes voltajes ya que hay componentes alimentados a 5V y otros a 3.3V. El microcontrolador esta alimentado a 3.3V por lo que no se debe introducir en sus pines un valor superior de voltaje. Es por esto que las señales de Bootloader, Reset y RXD que introducirían niveles superiores (ya que proceden del FT232RL y este integrado está alimentado a 5V), deben pasar por estos divisores que nos permiten bajarlas el nivel.

Se continua el diseño, introduciendo una segunda página (figura 3.4), en ella se coloca el circuito utilizado para la conexión de datos (PC-microprocesador) mediante un mini USB. En primer lugar, se crean los componentes J1 y U6 que hacen referencia al MINI-USB-B5 (conector mini USB) y al circuito integrado USBLC6-4SC6 respectivamente. El USBLC6-4SC6 es un circuito dedicado a protección ESD, en interfaces de alta velocidad. Permite a la corriente ESD fluir a tierra cuando se produce un evento ESD en la línea de datos.

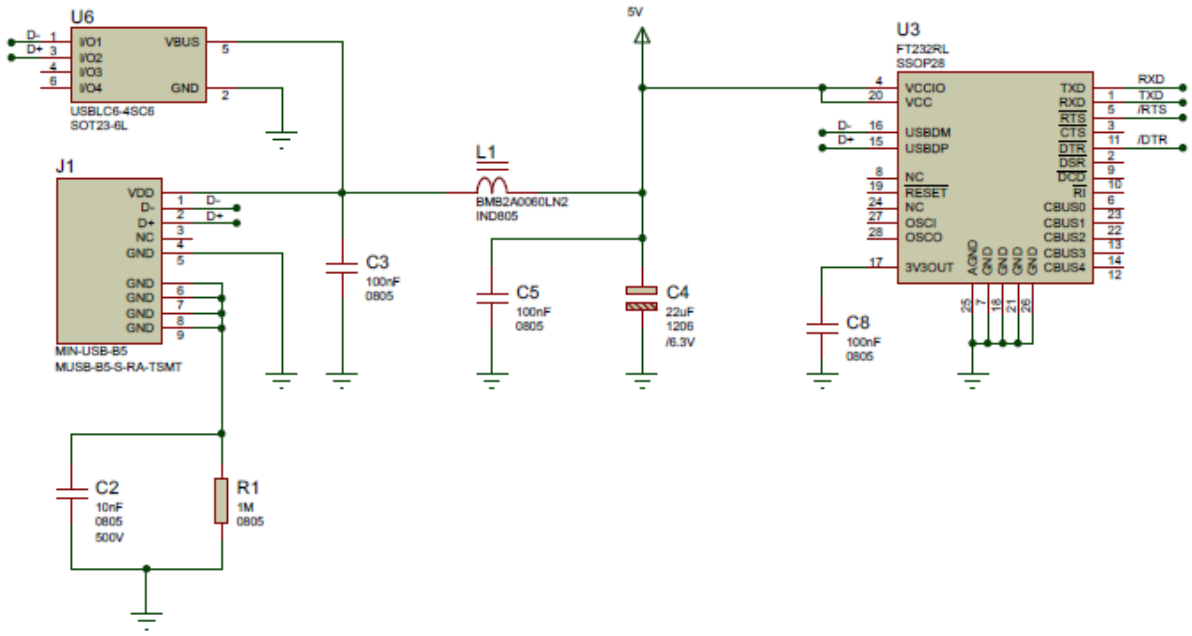

Figura 3.2: Circuito para la interfaz USB-serie.

Para integrar estos circuitos en el esquemático, se añaden condensadores de desacoplo, y se conectan estos dos componentes con una bobina al componte U3 un FT232RL, el cual será en encargado de enviar y recibir datos.

Además, en esta parte del esquemático se incluye el componente U4, un regulador de voltaje que como ya se mencionó permite disponer en nuestro sistema a partir de 5V un voltaje fijo de 3.3V. Este regulador es cortocircuitable y dispone de una protección contra temperaturas elevadas que apaga el regulador cuando la temperatura alcanza 160º, este no volverá a activarse hasta que la temperatura descienda al menos a 150º. Tiene una corriente de salida máxima de 500mA y su voltaje de entrada puede oscilar desde los 2.7V hasta los 6V. [15]

Se introduce en el diseño esquemático una tercera página (figura 3.5), en ella se sitúan los componentes que van a permitir, decodificar y codificar los datos necesarios, estos son los componentes U1 y U2 (HCS300 y HCS500 respectivamente). Ambos nos permiten trabajar con el protocolo Keeloq sin tener la necesidad de implementarlo por software.

También se introduce U8, una memoria EEPROM (24LC02B/SN) de 2K de capacidad que permite al decodificador almacenar los datos de los nuevos transmisores que aprende el circuito decodificador, en total puede almacenar datos de hasta 7 transmisores diferen-

tes.En la línea de alimentación de todos ellos se deben introducir unos condensadores de desacoplo.

Por último, se genera una nueva página en el esquemático (figura 3.6) en la que se crea el componente U5, que representa al transceptor RC-CC1101-SPI-SMT-434, que nos permitirá realizar transmisiones moduladas en ASK a una frecuencia de 433.92MHz. Este, es un módulo completo que tiene integrado la adaptación a la antena y oscilador para su correcto funcionamiento.

Se introduce el componente U7, expansor MCP23S08-E/SS que nos permite aumentar el número de entradas y salidas de propósito general, ya que las que nos ofrecía el ESP-12F no eran suficientes para nuestra aplicación. Gracias a el se consigue manejar las líneas del codificador manteniendo una comunicación SPI con el ESP y extraer la información en formato PM del decodificador.

En las figuras 3.3, 3.4, 3.5, y 3.6 se pueden observar todas las páginas del esquemático con todos los elementos detallados anteriormente y las conexiones que hay que realizar entre ellos para un correcto funcionamiento. Se puede destacar, que siempre los condensadores de desacoplo de cada uno de los integrados deben situarse cerca de estos.

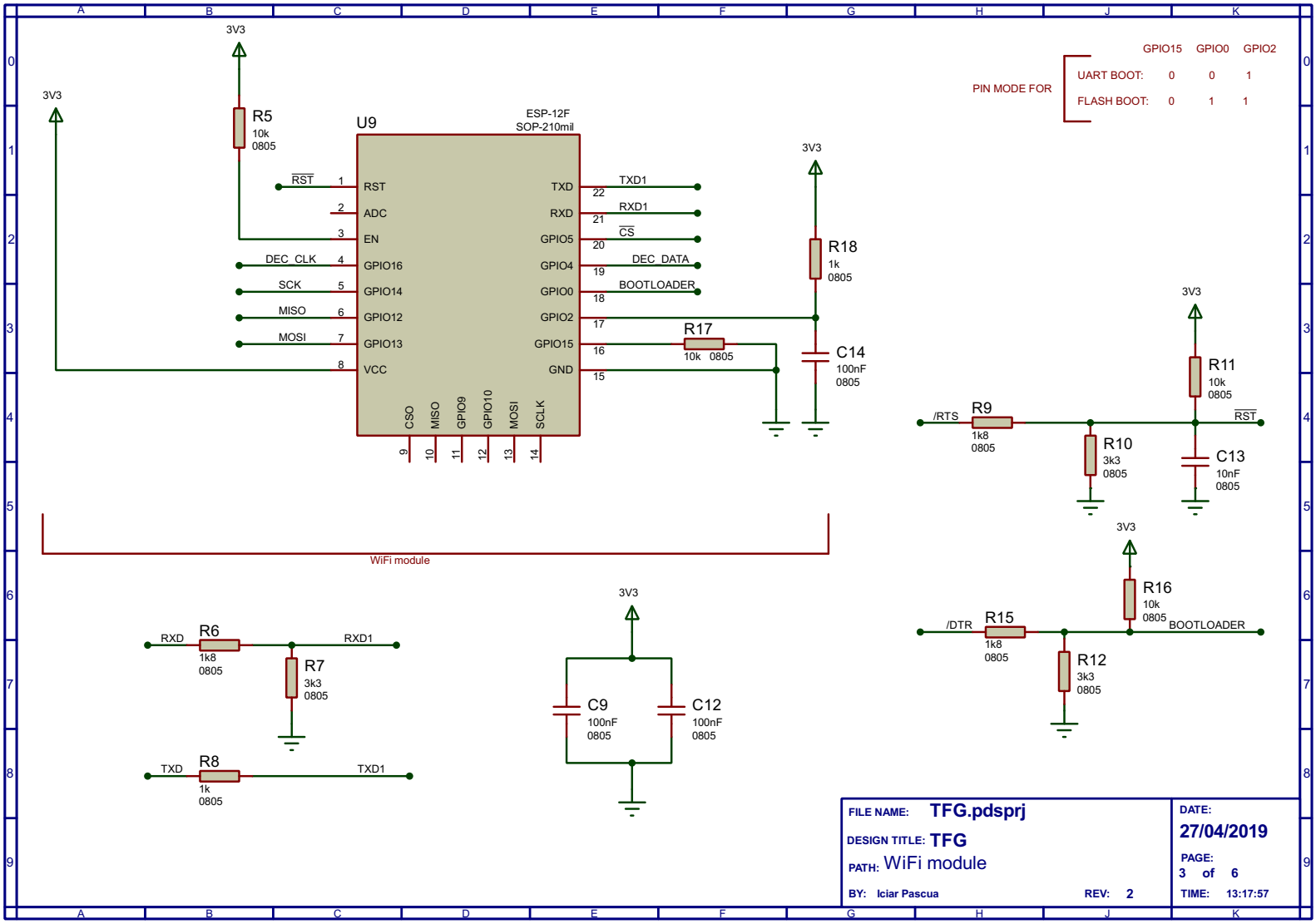

Figura 3.3: Página 1 del esquemático.

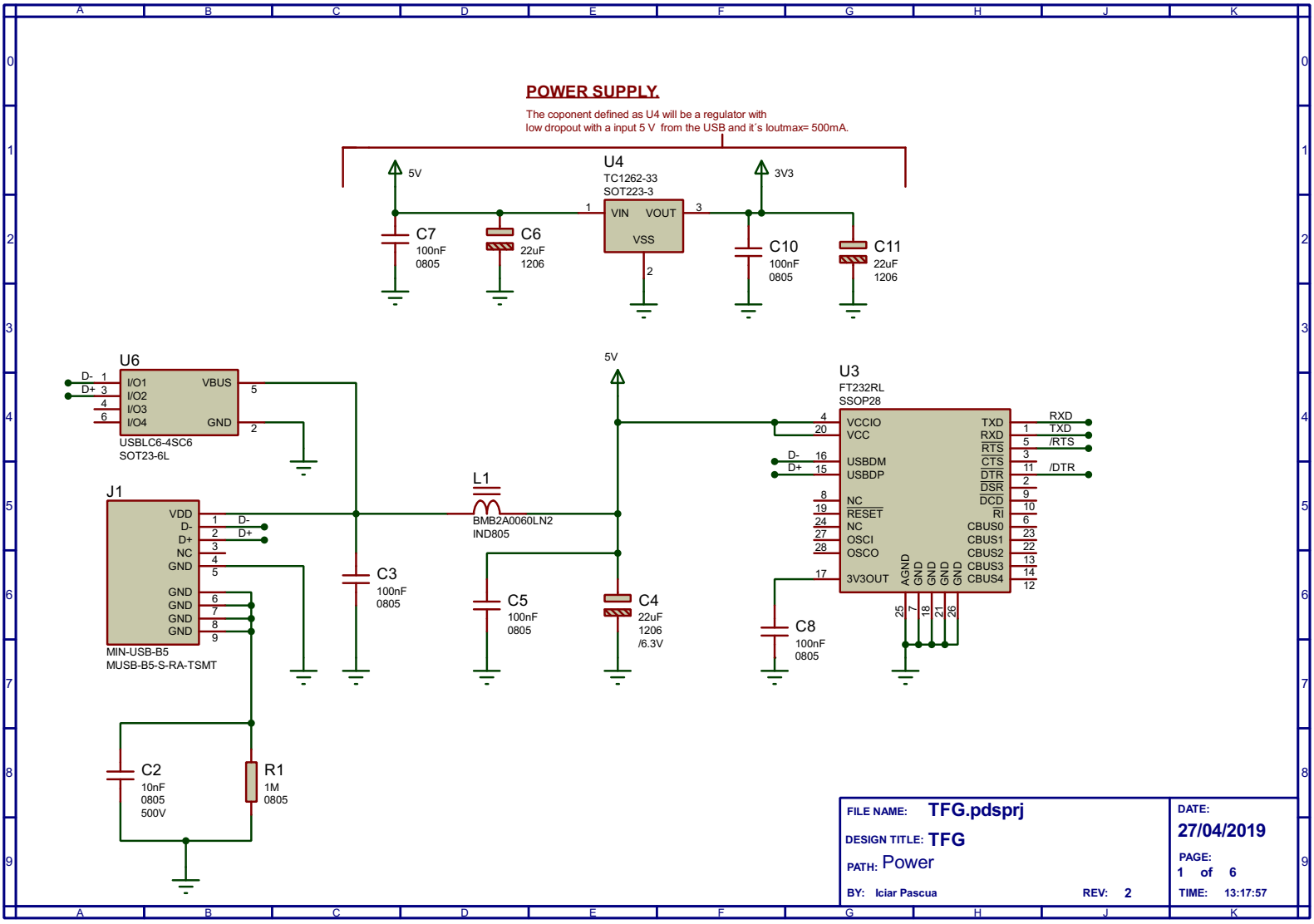

Figura 3.4: Página 2 del esquemático.

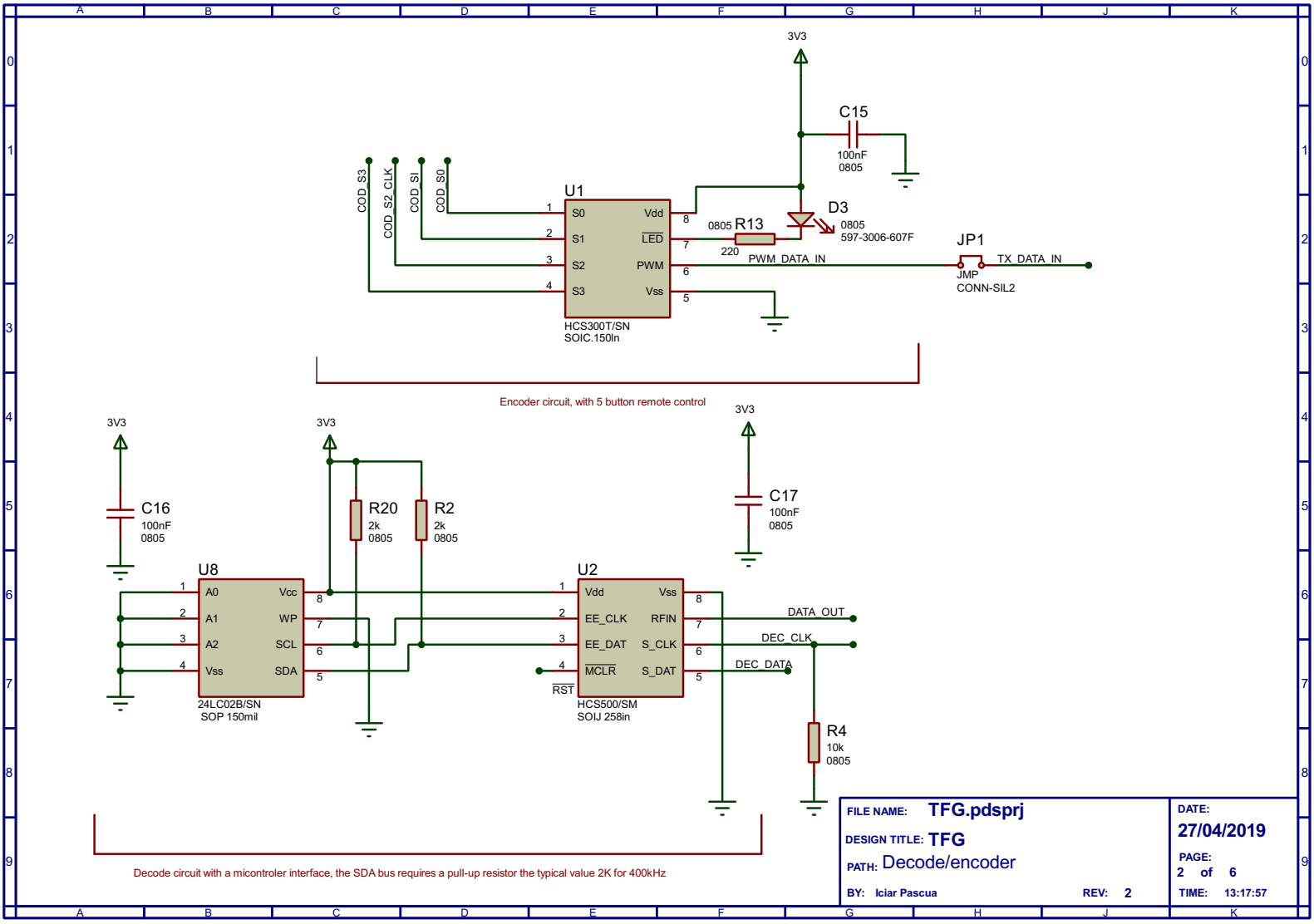

Figura 3.5: Página 3 del esquemático.

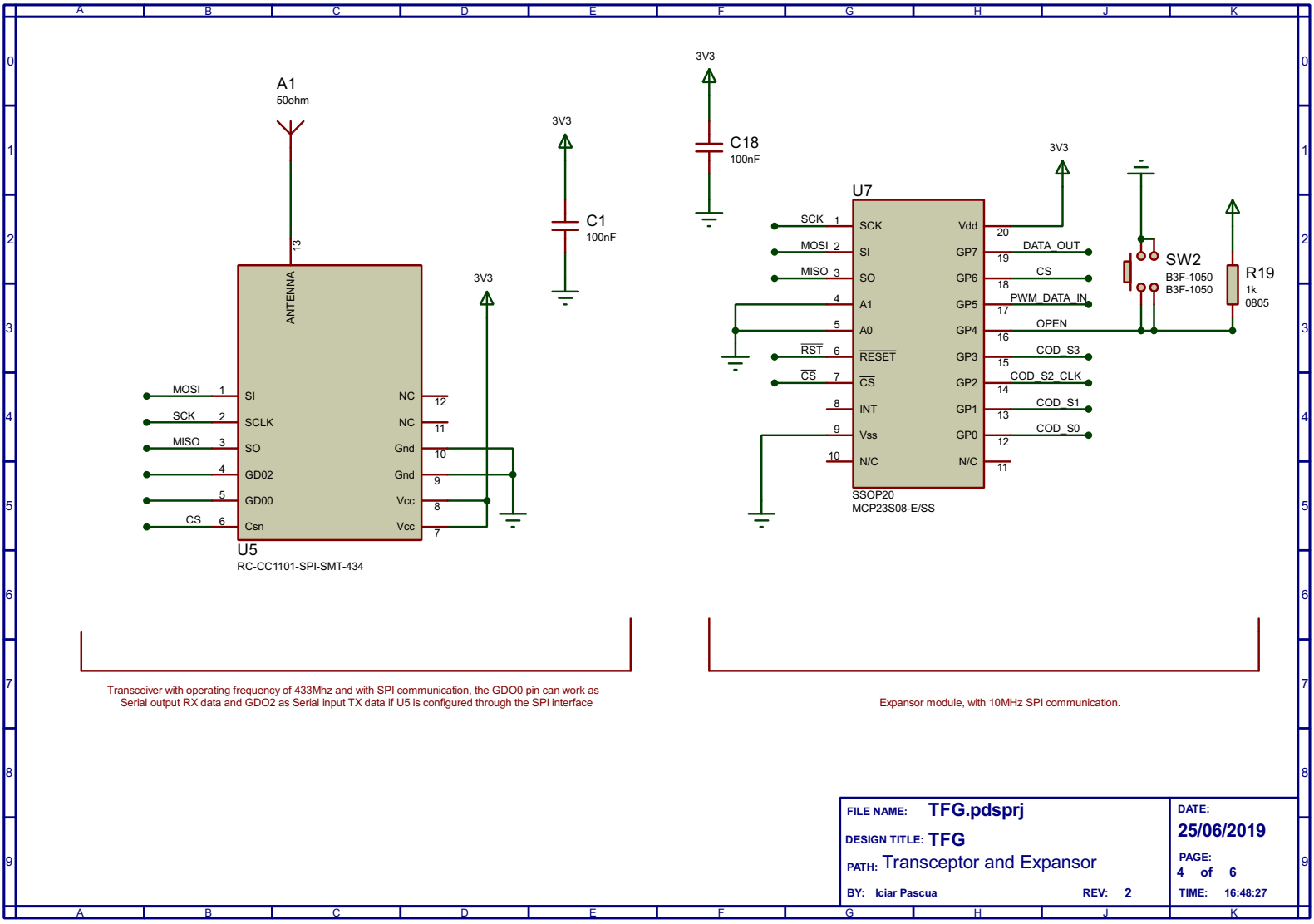

Figura 3.6: Página 4 del esquemático.

3.3. Análisis de consumo eléctrico

El análisis de consumo eléctrico se lleva a cabo una vez conocidos todos los componentes consultando en los datasheets de cada uno los datos necesarios. Así pues, se recoge en la figura siguiente tanto los voltajes, como las corrientes máximas que nos darán los componentes por separado. Pudiendo así, obtener en su totalidad el consumo que tendrá finalmente nuestro proyecto.

CONSUME POWER.

COMPONENT	VOLTAGE	CURRENT
ESP12f	3.3v	170mA
diode	3.3v	2.7mA
HCS300T/SN	3.3v	1mA
24LC02B/SN	3.3v	5mA
HCS500/SM	3.3v	1.8mA
RC-CC1101	3.3v	25.2mA
MCP23S08	3.3v	1mA
TOTAL	5V	206.7mA

TC1262-33	5v	0.130mA
FT232rl	5v	15mA
TOTAL	5V	15.130mA

USB 500mA máx

FINAL CALCULATIONS:

El regulador da 500mA y solo consumimos 206 luego sería suficiente

Figura 3.7: Consumo eléctrico.

Este análisis de consumo, marca la intensidad mínima de salida del regulador a emplear. Como los cálculos muestran que el consumo de los dispositivos conectados a la línea de 3.3V es de 203.7mA, se debe verificar que el regulador aporta al menos esta intensidad. El regulador escogido tiene una corriente de salida de 500mA, por lo que será válido para la aplicación.

Capítulo 4

Diseño y fabricación de la placa de circuito impreso

4.1. Creación de huellas de nuevos componentes

En esta tercera parte del proyecto, se pretende incluir todos los componentes con sus respectivas huellas para obtener el diseño de la placa PCB, con sus dimensiones correctas. Para realizar esto, se empleará el software Proteus al igual que para la realización del esquemático. En primer lugar, se deben crear las huellas de los componentes que no tiene definido Proteus en sus librerías.

Para la creación de las huellas de los componentes hay que basarse en los datos mecanicos del componente y las dimensiones recomendadas para los pads y la localización de estos presentes en el datasheet de cada uno de los componentes a emplear.

Tras crear la huella de los componentes, se deben introducir todos en el layaout y se procede a decidir la posición que va a tomar cada uno de ellos en nuestra placa PCB. Esta posición no debe ser aleatoria, si no que ha de seguir unas normas que se explican en el siguiente apartado. En la tabla 4.1 se muestran las huellas de cada uno de los componentes:

LISTADO	
COMPONENTE	HUELLA
ESP12-F	SOP-210MIL
USBLC6-4SC6	SOT23-6
MIN-USB-B5	MUSB-B5-S-RA-TSMT
FT232RL	SSOP28
HCS300T/SN	SOIC.150ln
HCS500/SM	SOIC 7.94MM*5.26
TC1262-33	SOT223-3
RC-CC1101-SPI-SMT-434	SMT
MCP23S08-E/SS	SSOP20
24LC02B/SN	SOP 150mil
597-3006-607F	0805
SWITCH B3F-1050	B3F-1050
MCAB 035075-33	IND805
RESISTENCIAS (220, 2K2, 10K, 1M ,2K ,1K8, 3K3)	0805_RES
CAPACIDADES (100nF, 10nF)	0805_CAP
CAPACIDAD 22nF	1206_CAP
XRISTAL	HC-49/SM

Tabla 4.1: Resumen de las huellas de los componentes.

4.2. Posicionado de componentes

Tras crear la huella de los componentes, se procede a decidir la posición que va a tomar cada uno de ellos en la PCB. En primer lugar, se introduce en el espacio de dibujo un componente que tenga las dimensiones de 10cmx10cm (ya que es una medida estándar de placa para el fabricante), a continuación se posicionan todos los componentes en el interior de esta zona de dibujo.

A la hora de realizar este posicionado, se debe tener en cuenta que las antenas de 433MHz y la antena Wi-Fi estén separadas lo más posible. Además, se debe buscar que las partes analógicas del circuito estén separadas de las digitales, para que no se produzcan interferencias de unas a otras.

Tras posicionar los componentes se debe pasar a realizar el rutado de las conexiones.

Un aspecto a tener en cuenta a la hora de realizar este rutado es que las zonas en las que irán situadas tanto la antena Wi-Fi como la antena de 433MHz no deben ser atravesadas por pistas. Además, las pistas deben ser lo más cortas posibles ya que unas pistas largan favorecerán las emisiones EMI al actuar estas como antenas.

Otra buena estrategia de diseño, es evitar en la medida de lo posible que las pistas posean ángulos de 90º, para evitar esto Proteus dispone de una hermanienta denominada "mitring"que elimina todos los ángulos rectos del diseño.

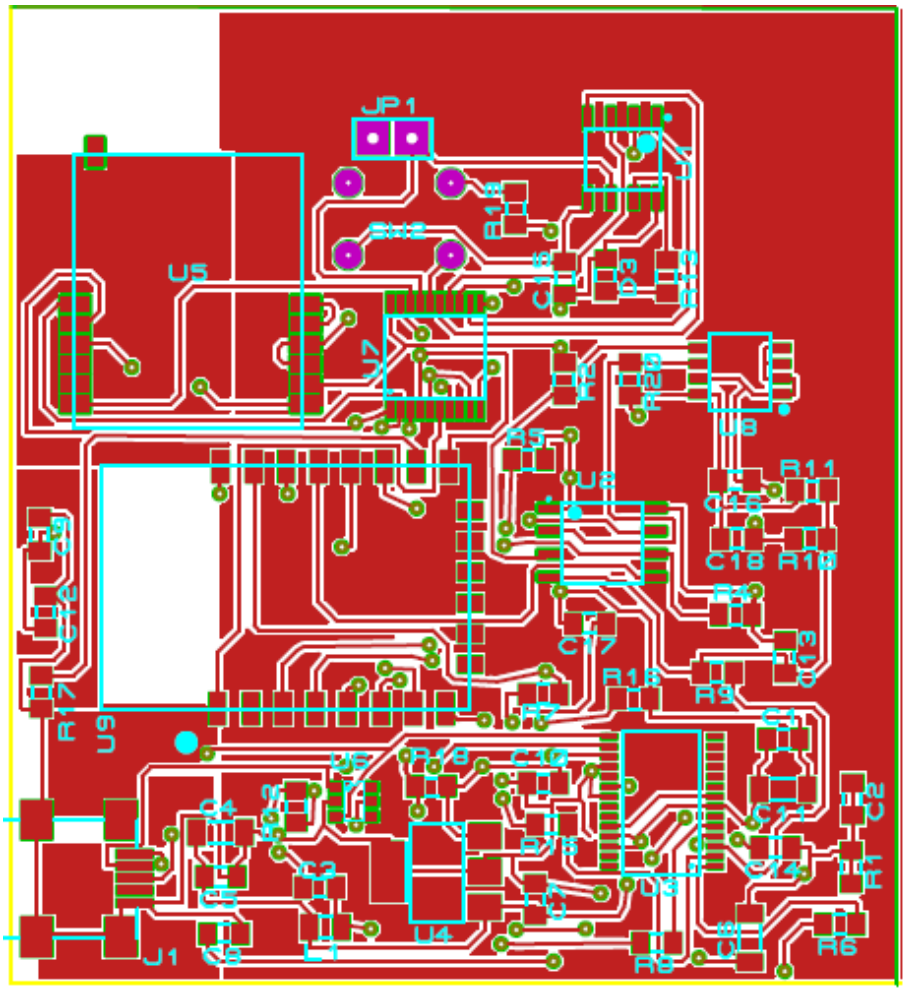

Figura 4.1: Cara superior de la PCB.

Una vez que todas las rutas estas trazadas es conveniente incluir planos de masa en ambas caras, esto se debe a que normalmente, se supone que una masa teórica es ideal. Es decir, tiene impedancia cero, potencial constante, y se considera sólo desde el punto de vista de su comportamiento en DC. Sin embargo, una masa real que actúa como camino

de retorno de las señales, como cualquier conductor, tiene una impedancia y por lo tanto las corrientes que circulan por la masa provocarán diferencias de potencial dentro de la misma.

Además, es importante tener en cuenta que las señales pueden regresar por un camino diferente al previsto, incluso pueden elegir caminos diferentes según la frecuencia. La impedancia de ese camino será responsable de acoplamientos por impedancia común.

Por lo tanto, al introducir un plano de masa en el diseño se tendrán infinitos caminos posibles de retorno para la señal, esto permite que la corriente regrese siempre por el camino de mínima impedancia. En las figuras 4.1 y 4.2 podemos observar el esquema con el rutado realizado y los planos de masa incluidos.

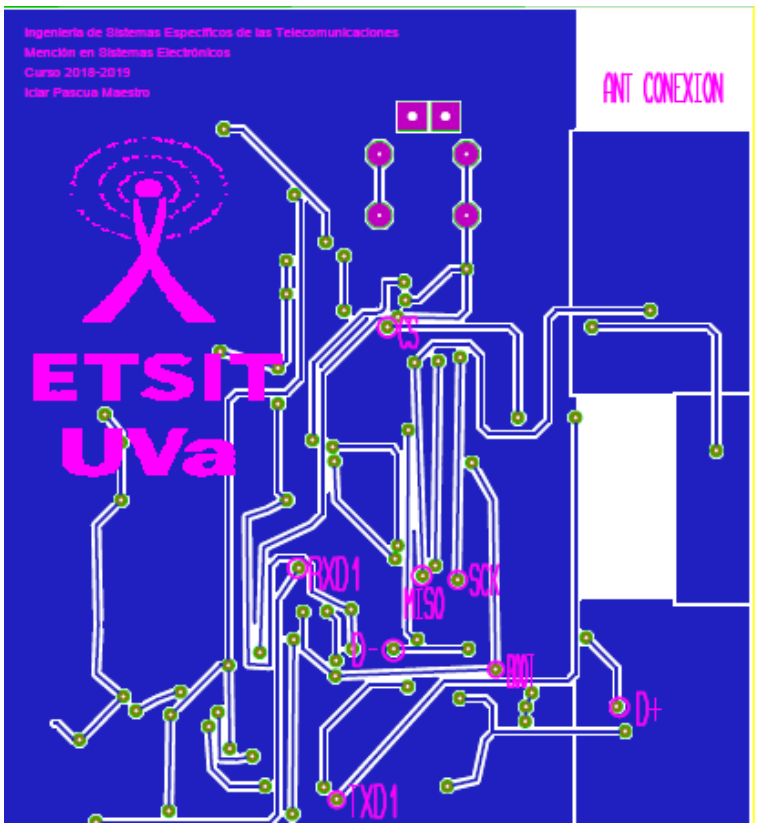

Figura 4.2: Cara inferior de la PCB.

4.3. Documentación: serigrafía

Proteus da la opción de serigrafiar la placa PCB, este proceso es meramente informativo. De esta manera una vez realizada la placa se podrá tener la información necesaria para el montaje sin necesidad de consultar el esquemático (por ejemplo, cual es el pin uno de cada uno de los componentes, o el ánodo y cátodo del diodo).

Además, también se puede incluir en la serigrafía información útil a la hora de realizar una depuración hardware, esto consiste en marcar y nombrar sobre dicha placa las líneas más importantes del proyecto como pueden ser la de RST, BOOTLOADER, RXD, TXD, etc. De esta manera cuando se quiera comprobar si la actividad en esas líneas es la correcta serán mucho más fácil de encontrar en la placa. Podemos ver estos datos en la figura 4.2 en la que se observa en las vías de algunas líneas que se han marcado con un círculo y al lado se ha situado el nombre de la línea.

4.4. Generación de ficheros Gerber

Una vez finalizado el diseño de la placa, se deben generar los ficheros necesarios para que el fabricante pueda fabricar la placa. Esta acción se realiza de forma automática a través del software de Proteus y estos ficheros se denominan GERBER.

Dentro de los ficheros Gerber se pueden encontrar diferentes archivos, en concreto se generará uno por cada capa del diseño, es decir se dispondrá de uno para la capa de cobre superior, otro para la inferior, uno por cada capa mecánica empleada y para la capa de SMT y Resist. Además, se generará un fichero Netlist que indicará las rutas y otro para el Drill de la placa.

4.5. Fabricación y montaje

Tras la generación de los ficheros GERBER, estos se deben enviar al fabricante para proceder a realizar la fabricación de la placa. En esta ocasión el fabricante elegido ha sido ITEAD, esta empresa está situada en China y se especializa en fabricación y desarrollo de productos hardware.

Una vez realizada la fabricación de la placa y antes de soldar los componentes en ella, se guillotinará de tal manera que se eliminen las partes sobrantes (ya que la placa se hizo

de una medida estándar, pero el diseño era más pequeño). En las figuras 4.3 y 4.4 se puede observar respectivamente la cara superior e inferior de la placa realizada una vez eliminadas las partes sobrantes.

Cuando se procede a soldar el diseño, se deben tener varios aspectos en cuenta, uno de ellos es soldar primero aquellos componentes más complejos. Por ejemplo, el conector mini USB deberá ser uno de los primeros en soldar.

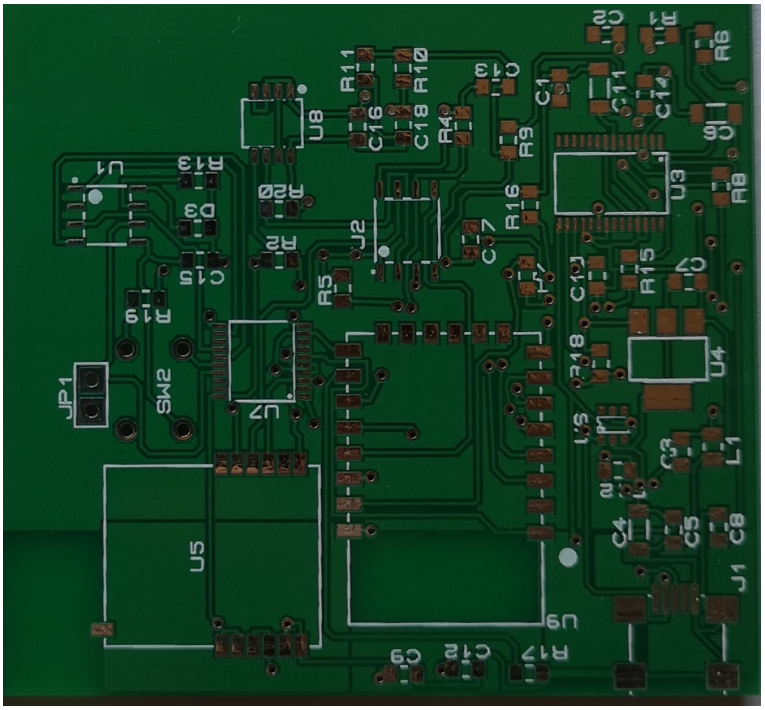

Figura 4.3: Placa PCB cara superior.

Otra recomendación a la hora de soldar es, aplicar flux allí donde se vaya a colocar un circuito integrado. El flux o pasta fundente, es un producto químico que permite eliminar el óxido de las superficies que se van a soldar, aumentando así la fiabilidad de la soldadura. No solo aumenta la calidad de la soldadura, si no que también simplifica el trabajo de soldar ya que concentra el calor permitiendo que el estaño fluya y se distribuya de forma uniforme por la unión.

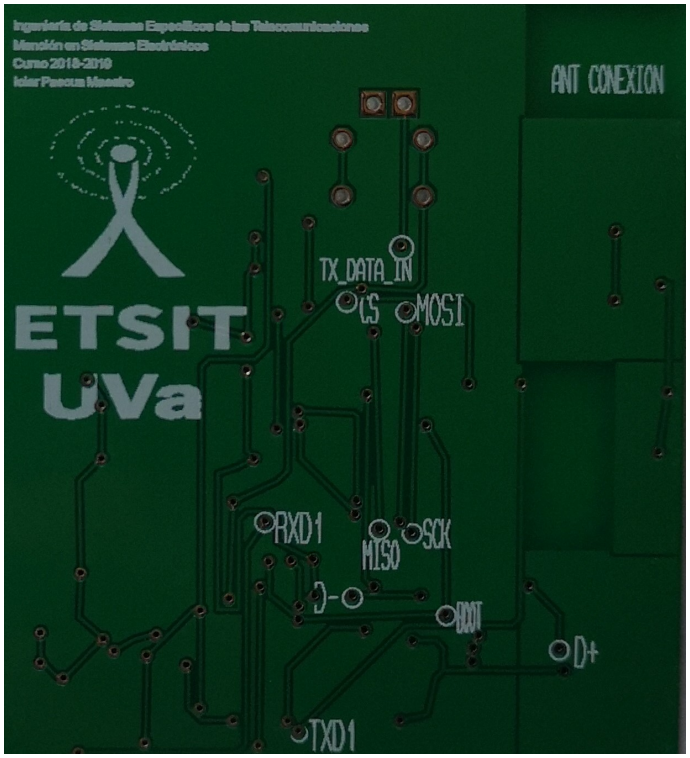

Figura 4.4: Placa PCB cara inferior.

4.6. Verificación del Hardware

Una vez realizado el montaje completo de la placa se debe proceder verificar su correcto funcionamiento. Lo primero que se debe verificar, alimentando la placa a 5V, es si el funcionamiento del regulador es el correcto. Es decir, si disponemos a su salida de un nivel fijo de 3.3V.

Una vez verificado esto, se puede conectar la placa mediante un cable con una conexión mini USB a un ordenador, si el ordenador reconoce el dispositivo como un FT232R USB UART (conversor USB bus serie) el conexionado del conversor y de los conectores será correcto.

AL realizar estas pruebas, se pudo comprobar que había algunos errores en el diseño inicial. El primer error detectado fue en la linea de reset, a esta línea se la había dado diferente nombre en sus dos extremos, por lo que no estaba bien rutada hasta el ESP12-F, este error se corrige soldando un cable desde el pin reset del ESP12-F (pin 1) hasta un punto de la placa donde este disponible, en este caso el condensador C13.

Otro error detectado en la placa a la hora de verificar el hardware fue que la línea de CS que selecciona al transceptor no estaba correctamente rutada por lo que de igual manera que con el error anterior, se procede a colocar un cable desde el pin correspondiente en el expansor (pin 18), hasta el CS del transceptor.

Se introduce una última modificación en el hardware en la cual se conecta la línea de salida de datos (DATA_OUT) directamente a la línea disponible en el expansor (GP7), con esto lo que se pretende es conseguir monitorizar los datos de salida del decodificador pasando a través del micro, en vez de enviarlos directamente al transceptor.

Para realizar más verificaciones sobre el hardware se deberá cargar en él un programa. Esto se desarrollará mas a fondo en el capítulo 6. En la figura 4.5 se puede observar el resultado final, la placa montada y verificada.

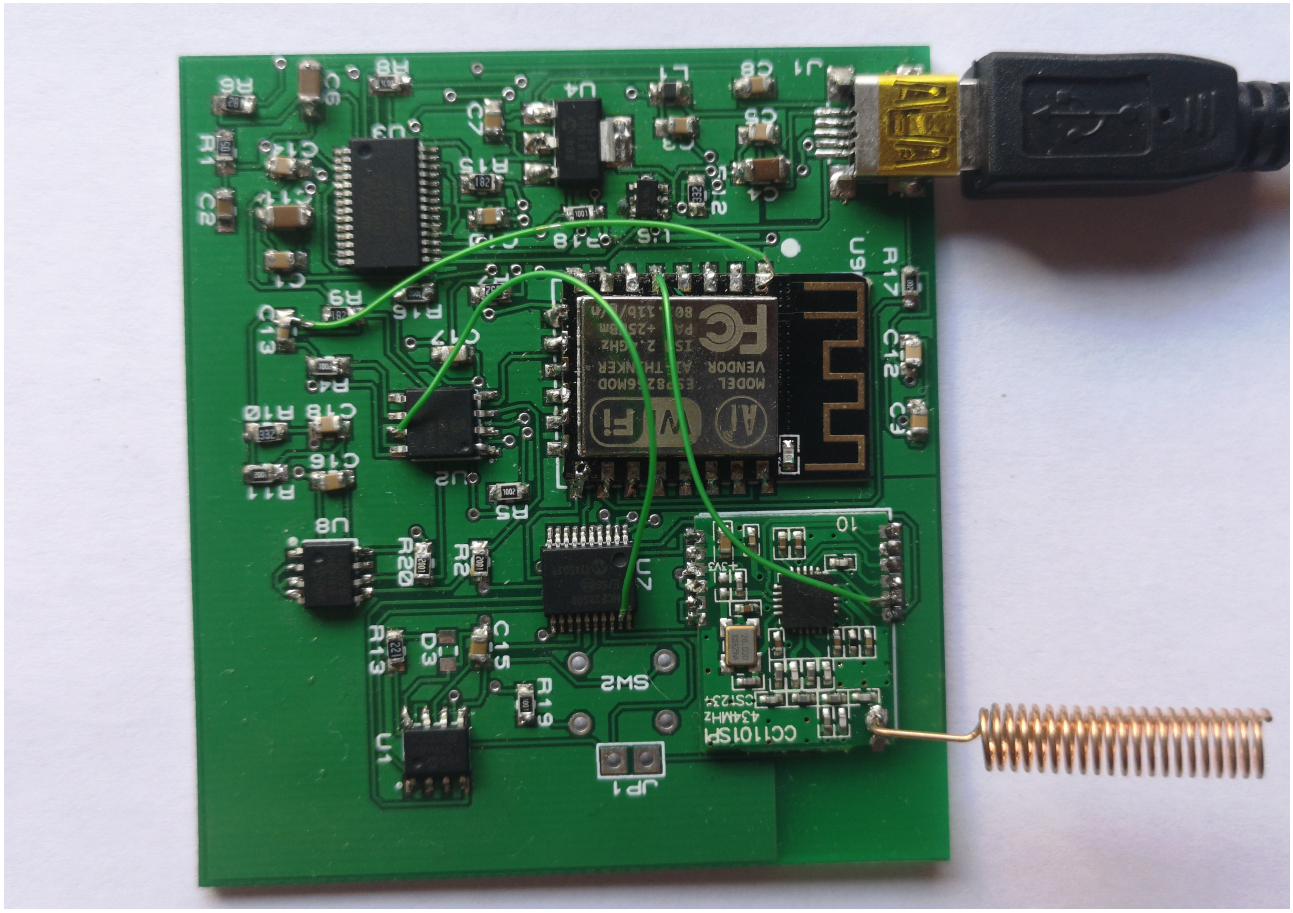

Figura 4.5: Montaje completo.

Capítulo 5

Realización del firmware. Simulación y depuración

5.1. Introducción

Hasta ahora, se ha comentado todo lo necesario para llevar a cabo la parte hardware del proyecto. Una vez hecho esto, se debe proceder a desarrollar y diseñar el firmware necesario para un funcionamiento correcto.

Para el diseño de este firmware se recurrirá al IDE de Arduino. El IDE de Arduino, es un entorno de desarrollo integrado que tiene como base el entorno de Processing. Este entorno de desarrollo es de software libre y permite al usuario crear sus propios programas y cargarlos en sus dispositivos. Para realizar esto, el entorno esta formado por un editor de texto, que permite escribir el código y un compilador. Una vez que el compilador ha compilado el programa, este se pasa a la memoria del microcontrolador empleando un puerto serie.

Arduino dispone de una gran variedad de librerías, que facilitan la creación de programas. Dentro de esa gran variedad de librerías encontramos las del ESP8266, microprocesador de nuestro módulo ESP-12F.

El primer paso a seguir, consiste en preparar el entorno de Arduino para que reconozca al ESP-12F. Para ello, se debe instalar el plugin "esp8266 by ESP8266 Community", una vez hecho esto al generar un nuevo proyecto ya se podrán seleccionar las características de la placa ESP que se va a programar.

5.2. Programa principal

5.2.1. Comunicaciones a implementar

Wi-Fi

Como ya se ha mencionado anteriormente, para realizar la comunicación del dispositivo con el smarthphone del usuario se va emplear una comunicación a través de Wi-Fi. El módulo ESP dispone de dos modos de funcionamiento en lo que a la comunicación Wi-Fi respecta:

- Modo Station: En este modo, el ESP es capaz de conectarse a un punto de acceso de una red existente y desplegar un servicio de tipo servidor web.
- Modo AP: En este modo, el ESP es capaz de generar una red Wi-Fi y admitir las conexiones de otros dispositivos.

En el proyecto actual, se empleará el modo Station. De tal manera que se debe proporcionar el SSID y contraseña de una red Wi-Fi al módulo ESP, para que este se conecte a ella y cree un servidor web. A este servidor web, será al que se conectará mediante un navegador el usuario desde su smartphone, permitiendole realizar las operaciones desde su móvil sin necesidad de tener que descargarse una aplicación.

Para implementar esta comunicación, el IDE de Arduino dispone dos librerías que nos servirán de gran utiladad, estas son: ESP8266WiFi.h y ESP8266WebServer.h. Estas librería se ha desarrollado basándose en el SDK del ESP8266. Gracias a la librería ESP8266WiFi.h, se tiene la posibilidad de controlar las diversas funciones relacionadas con el módulo WiFi. Y gracias a ESP8266WebServer disponemos de las funciones necesarias para realizar el tratamiento del servidor web al cual nos conectaremos desde el Smartphone. También será de mucha utilidad el método WiFi.mode(), el cual da la posibilidad de elegir el modo de funcionamiento entre los descritos anteriormente.

SPI

El bus SPI (Serial Peripheral Interface) tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro es el encargado de iniciar la comunicación con uno o varios dispositivos esclavos, y de enviar o recibir datos de ellos. Los dispositivos esclavos no pueden iniciar la comunicación directamente. En este bus, la comunicación es Full Duplex, es decir se pueden recibir y enviar datos de forma simultánea.

Este bus es fundamental para el proyecto, ya que será mediante el cual, el módulo ESP pueda comunicarse tanto con el expansor, como con el transceptor. El módulo ESP-12F, dispone de dos interfaces SPI, pero sólo una de ellas es funcional, la HSPI.

Para implementar esta comunicación, el IDE de Arduino dispone de la librería SPI.h, que es compatible con el módulo empleado. Las funciones básicas que nos incluye esta librería son :

- SPI.beginTransaction(), permite iniciar una comunicación SPI de las características que se le indiquen. Se puede seleccionar la velocidad, cuál es el primer bit transmitido, si el MSB o el LSB y el modo.
- SPI.transfer(), permite enviar un byte.
- SPI.endTransaction(), finaliza la comunicación SPI iniciada.

Serie Síncrona

Este tipo de comunicación se emplea para comunicar el módulo ESP con el codificador HCS300 y con el decodificador HCS500. Para realizar esta comunicación, se ha de emplear bit-banging, es decir, deberemos implementar la comunicación por software ya que no disponemos de recursos hardware en el módulo que nos la proporcionen.

Las líneas dedicadas a realizarla serán 2, una destinada a los datos y otra al reloj. Como solo tenemos una línea de datos, la comunicación será Semi Duplex. Es decir, no se puede escribir y leer simultáneamente. Las características de esta comunicación vienen impuestas por el HCS300 y el HCS500, ya que son los dispositivos con los que se va comunicar. En la figura 5.1 podemos observar las características de esta comunicación.

Como es un protocolo particular, no se dispone de librerías en Arduino para implementarlo, por lo que se deberá programar las funciones necesarias. En el proyecto se han creado dos funciones principales:

- escribirByteDecoder()
- leerByteDecoder()

Estas funciones suben y bajan la linea del reloj y la de datos, siguiendo el esquema mostrado en la figura 5.1, de tal manera que permiten tanto escribir como leer un byte

procedente del decodificador.

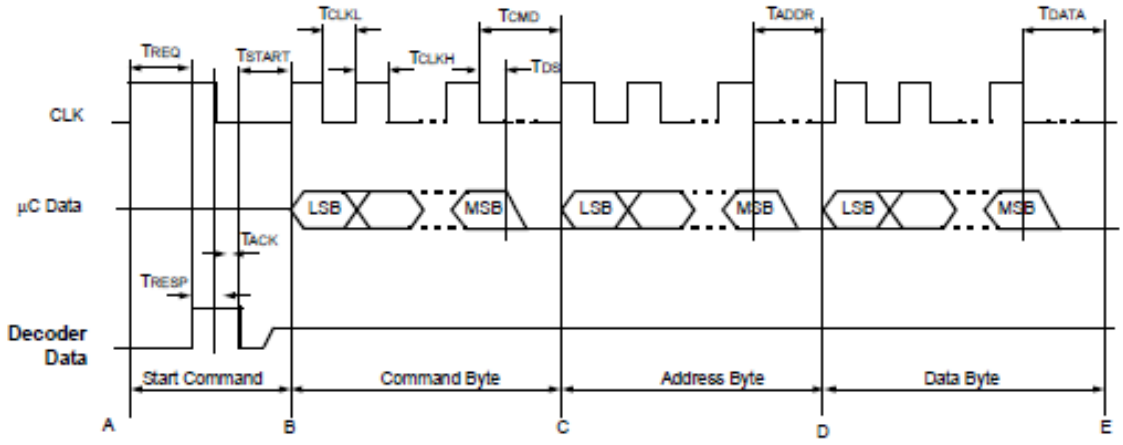

Figura 5.1: Comunicación serie síncrona empleada para el HCS500.

5.2.2. Desarrollo principal

A la hora de plantear la solución final del firmware, es de utilidad esbozar como va a ser el flujo principal del programa. En primer lugar, al arrancar el programa se debe declarar la funcionalidad de cada uno de los pines del módulo ESP, a continuación se deben realizar todas las inicializaciones necesarias:

- Inicialización de los parámetros de la comunicación SPI.
- Inicialización de los registros del expansor, para indicar cuáles de sus pines serán salidas y cuáles entradas.
- Inicialización del transceptor con los parámetros necesarios para la comunicación a 433Mhz, ASK, de una tasa de datos de 2.66Kbps y con una potencia de emisión de 10dBm. Para lograr esto se deben modificar numerosos registros, el valor de estos vendrá determinado por el programa Smart Rf Studio del que ya se ha hablado anteriormente.
- Inicialización de las variables globales del código
- Inicialización del módulo Wi-Fi, en el cuál se intentará establecer conexión con la red Wi-Fi especificada y a continuación se creará el servidor, definiendo la función que se debe realizar para cada una de las diversas peticiones que lleguen.

Después de estas inicializaciones se pasa al bucle principal del programa, en el será en el que se traten las peticiones al servidor web creado, y dependiendo de las peticiones recibidas se hará una u otra operación. Las operaciones disponibles pasa el usuario serán:

- Aprender el código de una nueva llave.
- Abrir puerta.
- Cerrar puerta.

Además de vigilar las peticiones al servidor, en el bucle principal también se debe vigilar la linea conectada al pulsador, de tal manera que cada pulsación detectada desencadenará una acción de cerrar o abrir la puerta según sea el estado anterior de esta. Si el estado de la puerta es abierto y se detecta una pulsación, se procederá a cambiar el estado a cerrado y a mandar un mensaje de cierre y viceversa.

5.2.3. Análisis del código

Dentro del proyecto hay unas variables globales, fundamentales para su correcto funcionamiento. Estas variables son :

- **ssid**: Dónde se almacenará el nombre de la red Wi-Fi a la que se debe conectar el módulo.
- **pasword**: Contendrá una cadena correspondiente a la contraseña de la red Wi-Fi a la que se va a conectar.
- **estadoPuerta**: Indicará en que estado se encuentra la puerta en cada momento puede ser abierta (1) o cerrada (0).
- **datosFifoRX[29]**: Buffer en el que se almacenará todo el contenido que llegue a la FIFO de recepción del transceptor para su posterior análisis.
- **datosFifoTX[40]**: Buffer en el que se introducirán todos los datos que se quieren enviar a través del transceptor para que se vayan situando en la FIFO de trasmisión de este sin producir un desbordamiento.

Algunas de las funciones principales junto con un una breve descripción de su funcionamiento son las siguientes:

- **handleClient()**: En esta función se detecta si hay un nuevo cliente, si es así se recoge la petición que envía al servidor y se analiza. Hay tres tipos de peticiones principales, la de abrir puerta, cerrar puerta y aprender llave. Dependiendo de que petición sea se llamará a la función asociada a esta petición.

- **handleRoot()**: Esta es la función asociada a la accion de entrar en el root de la página web. Y su respuesta es mostrar la estructura de esta.

- **handleAbrir()**:Se coloca en las entrada al codificador el código correspondiente a apertura y se recoge el mensaje PWM a transmitir. Este mensaje se guarda en el buffer de trasmisión y se envía en cuanto sea posible.

- **handleCerrar()**:Se coloca en las entrada al codificador el código correspondiente a cierre y se recoge el mensaje PWM a transmitir. Este mensaje se guarda en el buffer de trasmisión y se envía en cuanto sea posible.

- **handleAprender()**:Se procede a realizar el aprendizaje en el decodificador del mando y después se pasan los datos leídos y almacenados en la memoria EEPROM conectada al codificador para que pueda sincronizarse.

- **ttoBoton()** En esta función se escanea la línea de entrada del botón. Si se detecta una pulsación, se consulta el estado de la puerta y si el estado es abierto, se llama a la función handleCerrar(), para que se emita un mensaje de cierre de puerta. Si el estado de la puerta es cerrado y se detecta una pulsación se llama a la función handleAbrir() de tal manera que se emita un mensaje de apertura de puerta.

A continuación se han incluido los diagramas de flujo del programa, para poder tener una mejor perspectiva de él, si se quiere observar con detalle cada una de las funciones del código y como se implementan, en el Apéndice I esta incluido el código del programa completo.

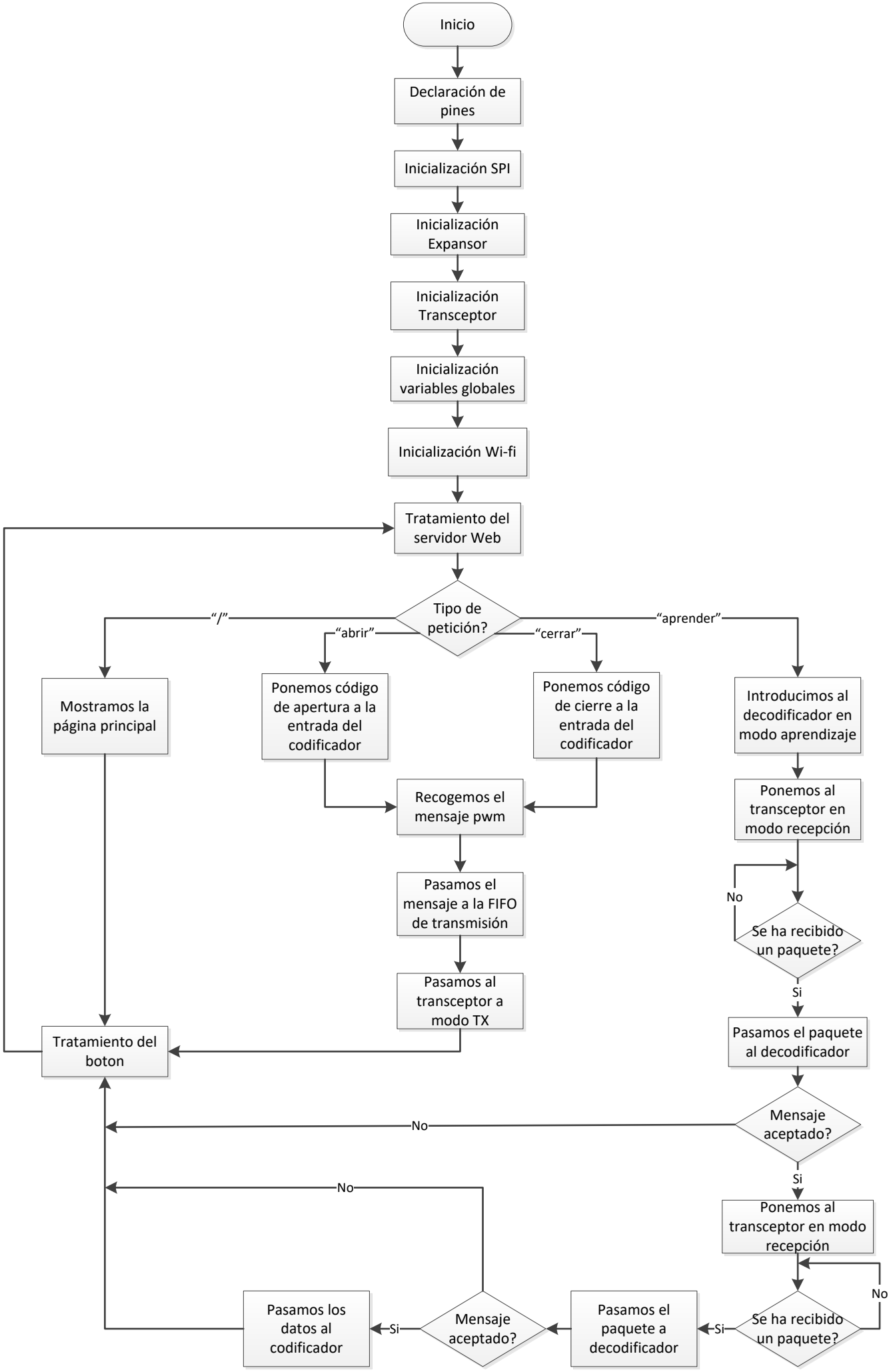

Figura 5.2: Diagrama principal del programa.

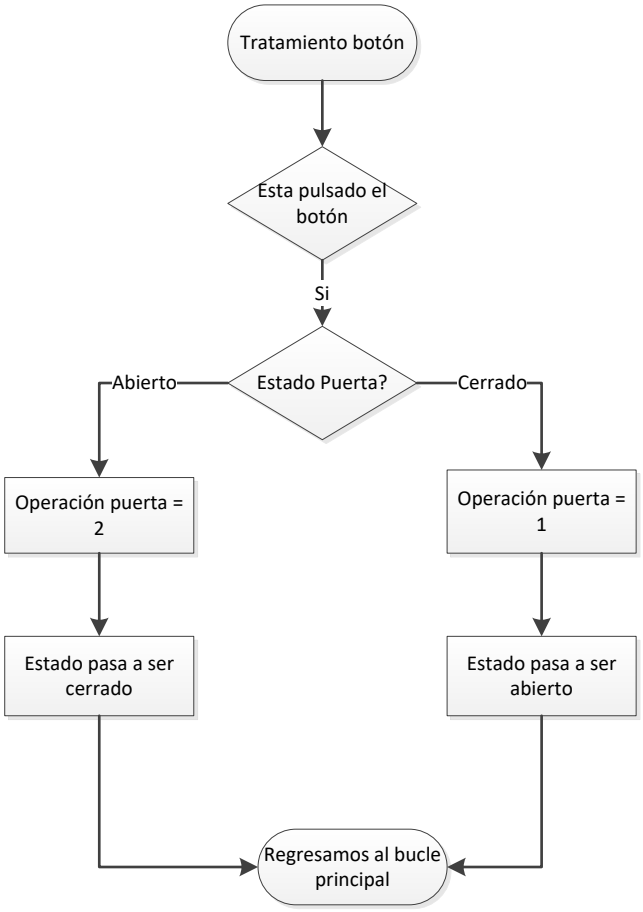

Figura 5.3: Diagrama acerca del tratamiento del botón.

Capítulo 6

Pruebas y verificación de resultados

6.1. Pruebas

Para la realización de las pruebas, va a ser fundamental conocer las características de la señal emitida por el dispositivo y por la llave. Para ello se van emplear dos programas diferentes. El primero de ellos es SDR (Software Device Radio) Sharp, este es un programa muy intuitivo que permitirá escanear el espectro de frecuencias y ver en que frecuencia se esta emitiendo, cuál es el ancho de banda de la señal emitida, la calidad señal a ruido, y la potencia de emisión. Otra característica del programa que puede resultar muy útil es que permite grabar la señal emitida para después poder analizarla. Para el uso de este programa es fundamental disponer de un SDR.

El segundo programa empleado para realizar las pruebas de funcionamiento del proyecto será Audacity. Es una plataforma open source que permite analizar señales. Este programa permitirá analizar las grabaciones realizadas con SDR Sharp y ver cual es la tasa de datos enviada, que datos se han enviado así como la modulación de estos.

6.1.1. Verificación Llave

Una vez definido el software, se puede comenzar con las pruebas. Para comprobar el correcto funcionamiento del sistema será conveniente ir verificando por partes que funciona y una vez hecho esto verificar el resultado completo. Lo primero será disponer de una llave compatible con nuestro sistema de la cuál escanearemos el código. Por lo tanto, la primera prueba a realizar será un escaneo de la señal emitida por una llave.

Para que la llave sea compatible en el sistema, deberá emitir en la banda de 433MHz y el formato de la trama debe ser como el de la figura 2.7. Se procede a analizar la señal emitida en SDR Sharp, y se observa (como muestra la figura 6.1) que en el primer canal del mando lo que se emite es una señal de frecuencia central 433.85Mhz, con una potencia de emisión de -1.7dBFS y un ancho de banda de 20KHz.

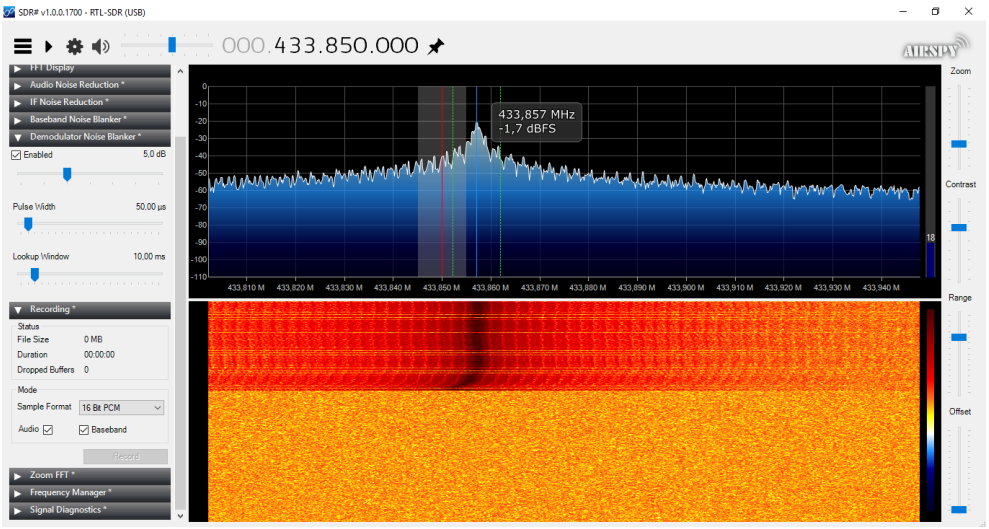

Figura 6.1: Captura del programa SDR Sharp analizando una pulsación del mando de garaje.

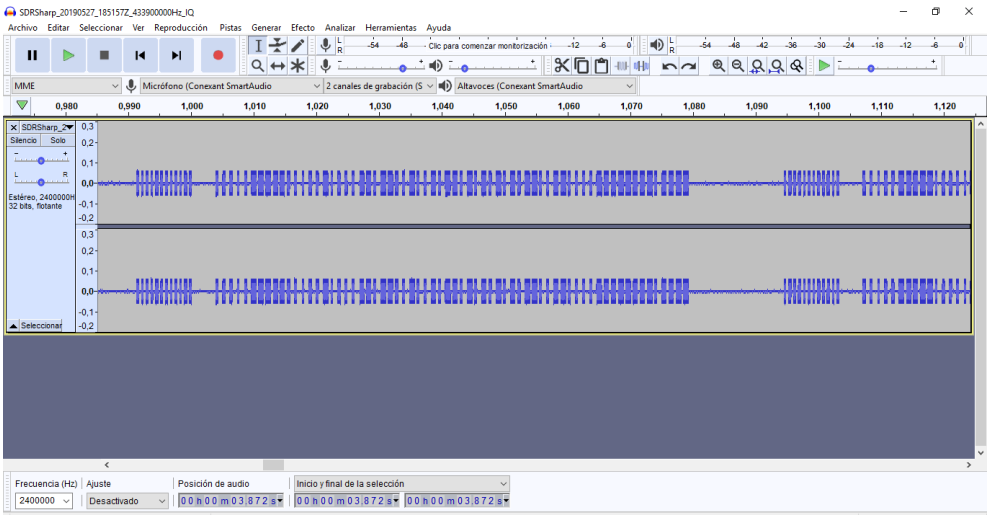

Figura 6.2: Captura del programa Audacity mostrando el contenido de una pulsación en llave.

Una vez que se ha comprobado que la frecuencia de emisión es compatible, desde SDR Sharp se realiza una grabación de la señal y se analiza con Audacity (Figura 6.2). Se

observa, que la señal posee un preámbulo al 50 % del ciclo de trabajo, a continuación la cabecera que mantiene a cero los datos, y luego comienza un mensaje de 66 bits modulado en PWM tal y como se mostraba en la figura 2.7. Por lo tanto, se puede concluir que el mando escogido es válido para la aplicación y comenzar a realizar pruebas con él.

6.1.2. Verificación Transceptor

Continuando con las pruebas, se debe corroborar la correcta emisión y recepción del transceptor. Una vez que mediante SPI se han modificado los registros de configuración, y se han fijado tal y como indica el software de Texas Instrument Smart RF para que la señal tenga las características más similares a la llave posibles. Se prueba a realizar una emisión con este y analizar lo emitido tal y como se procedió con la llave.

En este proceso se observó que aunque la frecuencia fijada era de 433.85Mhz, la emitida no era la misma si no que estaba desviada en frecuencia. Se prueba a fijar otro valor y también se observa que se desvía y que la desviación es igual para ambas frecuencias y de un valor de -0.92Mz. Visto este resultado se decide fijar una frecuencia central en los parámetros de configuración de:

$$f_c = 433{,}85Mhz + 0{,}92MHz = 434{,}77Mhz$$

Con esta frecuencia central fijada ya se puede ver una frecuencia de emisión correcta. En cuanto a los datos, como prueba se escribe en la FIFO de transmisión una trama de nueve bytes todos con valor 0xAB y se comprueba en Audacity que los datos son correctos y que la baud rate es correcta.

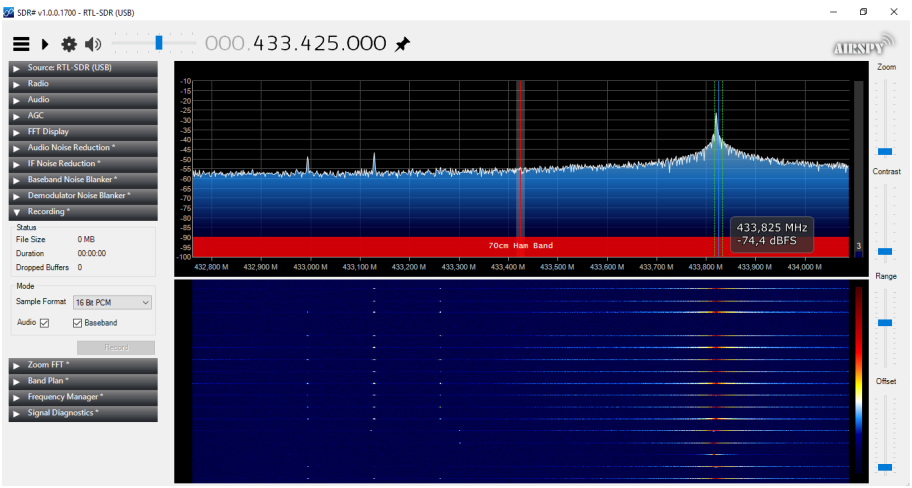

Figura 6.3: Captura del programa SDR Sharp analizando una emisión del transceptor.

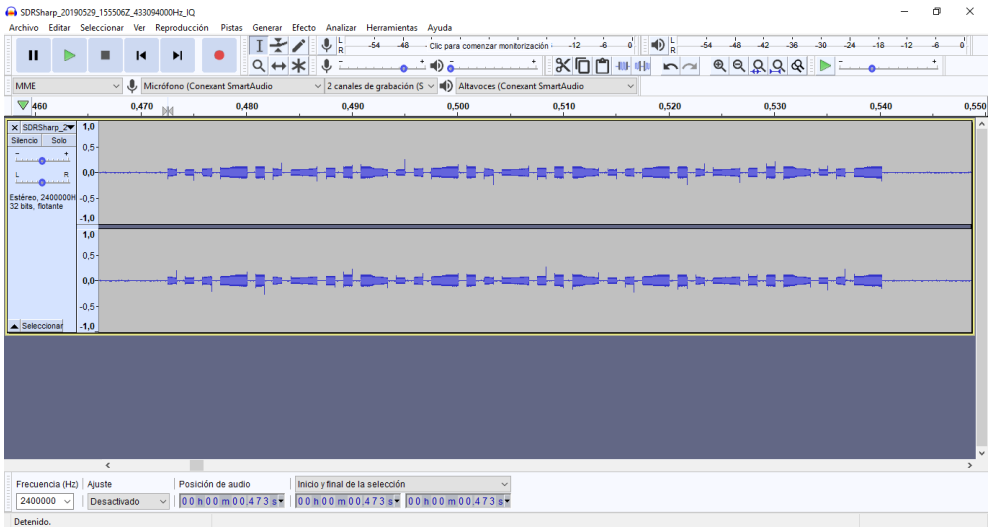

Figura 6.4: Captura del programa Audacity analizando una emisión del transceptor.

Una vez comprobado que la emisión se realiza de una forma satisfactoria se pasa a comprobar que generando una trama desde la llave, el transceptor es capaz de almacenarla en su FIFO de recepción para un posterior análisis.

Hay que tener en cuenta que para que el transceptor pueda decodificar correctamente la señal emitida por la llave se deben modificar los registros de configuración. Al ser la señal modulada en ASK, los parámetros prefijados para el bloque demodulador no son válidos ya que estos están optimizados para modulaciones FSK/MSK/GFSK y si se utilizase la configuración prefijada, cambiando el tipo de modulación, daría lugar a una recepción no óptima e incluso inestable.

Una vez establecidos los parámetros de recepción correctos, se deben desactivar los filtros de recepción, en transceptor ofrece las opciones de filtrar los paquetes que llegan por su dirección, su CRC o su longitud. Como la trama que emite la llave no dispone de ninguno de estos parámetros, si no se desactivaran no guardaría ninguno de los paquetes emitidos.

Con los parámetros establecidos se introduce el transceptor en modo recepción a través de la Strobe correspondiente (SRX), se pulsa el mando y se recogen los datos de esta pulsación. Si se analizan los datos presentes en la FIFO, se puede observar que se han almacenado 23 bits de preámbulo, 10 bits a cero correspondientes a la cabecera y a continuación 198 bits correspondientes a los 66 bits de información modulados en PWM.

En la figura 6.5, se puede observar la trama recibida. Se ha marcado en rojo el preámbulo, en verde la cabecera,y en azul y amarillo los bits de datos. Cada tres bits almacenados en la FIFO tendríamos un bit sin modular, la combinación de bits 110 representa un 0 binario y la combinación de bits 100 representa un 1 binario. Por lo tanto, al ser la estructura de la trama obtenida igual que la emitida por la llave podemos afirmar funcionamiento del transceptor es correcto.

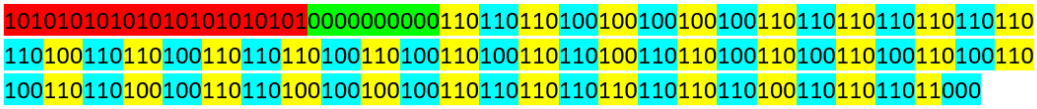

Figura 6.5: Trama almacenada en la FIFO del transceptor.

6.1.3. Verificación Decodificador

Para el tratamiento del decodificador, se han creado unas funciones especificas en el código que permiten enviar y recibir un byte de este. Para comprobar que funciona correctamente lo primero es verificar que entiende las tramas que se le envían.

Gracias a la función de enviarByteDecodificador(), ponemos en la línea de datos el comando necesario para escribir un registro de la EEPROM asociada a él, a continuación, implementamos el comando de lectura de la misma posición de memoria. El resultado obtenido es el mismo, por lo que podemos concluir que la memoria funciona correctamente y que el codificador comprende los mensajes que le enviamos.

Para concluir con la verificación de funcionamiento del HCS500, se le debe poner en modo aprendizaje y enviarle la información que llega del transceptor procedente de la llave. Si este responde con un mensaje de estado y el valor es el correcto, el funcionamiento completo del HCS500 habrá sido verificado.

6.1.4. Verificación codificador

El codificador, será el encargado de generar un código PWM (que sustituirá al generado por la pulsación de la llave) cuando alguna de sus entradas sea activada. El código generado por el codificador, pasará a almacenarse en el módulo ESP mediante una transmisión síncrona implementada a través de las lineas PWM y S2.

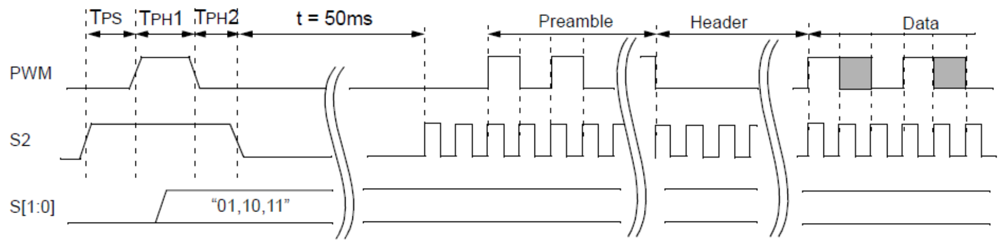

Figura 6.6: Implementación de la transmisión síncrona para obtener al trama PWM.

La trasmisión síncrona, permitirá que la trama sea obtenida, controlando la salida de los bits a través de la línea PWM mediante la línea de reloj, la frecuencia de este reloj no debe superar los 20KHz. En este proyecto, ya que se muestrean las lineas a través de la comunicación SPI mantenida con el expansor y la velocidad de la conexión SPI es de 1MHz, se decide que el período de la señal de reloj sea de 50us (ya que en subir o bajar una linea se emplean 25us).

Para verificar el correcto funcionamiento del HCS300, se sitúa la combinación binaria 01 en S1 y S0 respectivamente, a continuación se guarda en un buffer la trama generada por el HCS300 y se analiza. Hay que tener en cuenta que en está trama habrá 16 bits más. Estos bits introducidos por el codificador antes de la carga útil del mensaje cuando se realiza una transmisión síncrona y pueden ser eliminados a la hora de enviar la trama a través de el transceptor.

En la figura 6.7 se puede observar la trama obtenida y guardada en el buffer al fijar la combinación correspondiente en S1 y S2, en ella se distinguen en rojo los 23 bits correspondientes a preámbulo, en verde los 10 bits correspondientes a la cabecera, y en azul y amarillo los de la carga útil del mensaje. La trama obtenida cumple con las especificaciones descritas en el datasheet del HCS300, por lo que podemos verificar el correcto funcionamiento del decodificador.

```
101010101010101010101010000000000110100100100110110110110100100110100110110100
110100100110100100110110110110100110110100110100110110110110110110110110110110
110110110110110110110110110110110110110110110110110110110110100110110110110110
110110110110110110110110110110110110110110110
```

Figura 6.7: Trama generada por el codificador al fijar una combinación en sus entradas.

6.2. Verificación firmware final

La realización de la verificación final se hará en dos etapas. En la primera etapa se debe comprobar que con el programa corriendo en la placa diseñada, si se introduce la dirección IP dada por el ESP (192.168.43.174) en un navegador web, observamos la página creada por el servidor (figura 6.8). Podemos observar que en esta página se disponen de tres botones: abrir, añadir y cerrar puerta. Al pulsarlos el dispositivo debería realizar las operaciones correspondientes.

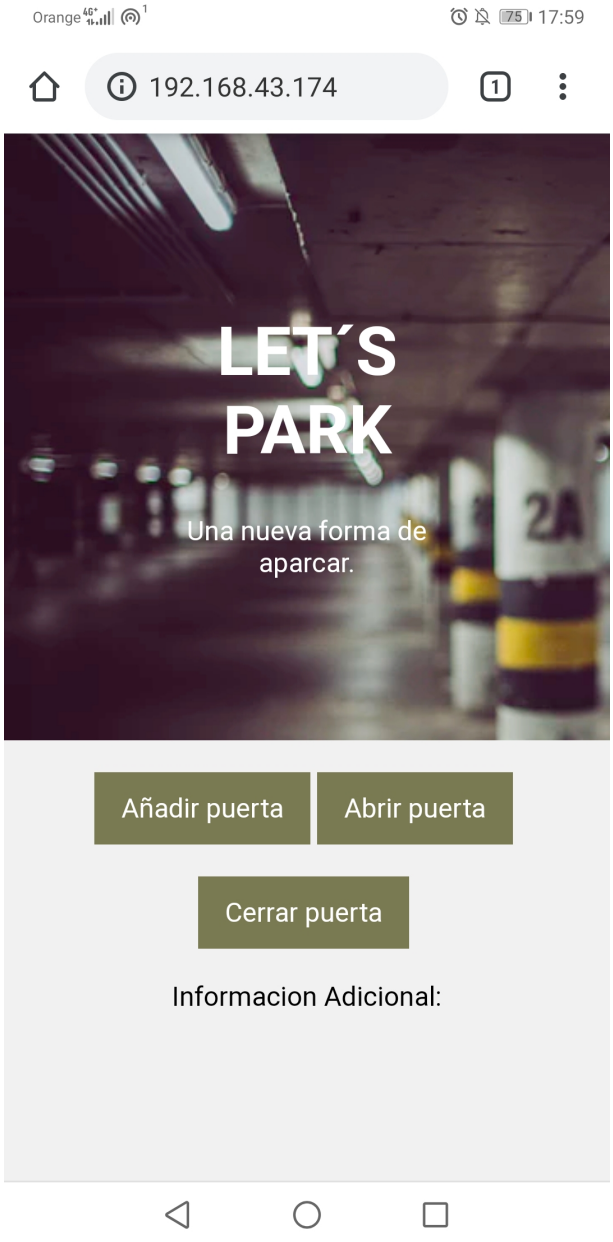

Figura 6.8: Captura de la página web creada por el ESP-12F.

A continuación, con el programa SDR Sharp, se observará que es lo que se transmite al pulsar una de las opciones disponibles y si el formato de la trama trasmitida es el correcto.

Realizando este experimento, se verificará todo el funcionamiento relacionado tanto como con la parte Wi-Fi del proyecto como con la parte de codificación y emisión RF. En la figura 6.9, se puede observar un mensaje de cierre de puerta emitido por el dispositivo cuando se ha pulsado la opción de cierre de puerta desde la página web.

Para poder analizar el funcionamiento completo del sistema, sería necesario disponer de una puerta de garaje en la que hacer las pruebas de recepción del código RF. Como esto no es posible, se decide montar una segunda placa igual que en la anterior pero en esta, se probará la función de aprendizaje y cada vez que se reciba un código correcto se procederá a iluminar el LED que esta disponible en el módulo y que hasta ahora no se había empleado.

Para este segundo prototipo, no será necesario la parte del código que crea el servidor Web, ya que su único funcionamiento sería activar un relé que permitiera el movimiento de la puerta, en este caso en vez del relé como ya se ha mencionado será un LED, de tal manera que el efecto de apertura pueda ser visual.

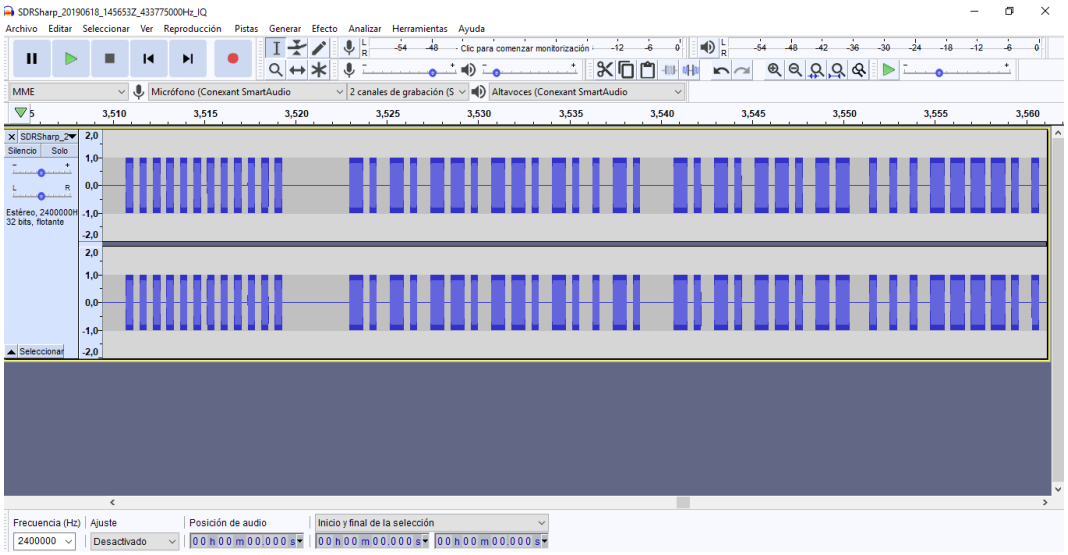

Figura 6.9: Señal de apertura de puerta emitida por el dispositivo final.

En el apéndice 2 se puede observar el código que ha de introducirse en la segunda placa, para que esta tenga el funcionamiento deseado. Su funcionamiento es básicamente

introducir al módulo transceptor en modo recepción y esperar hasta que se haya recibido una trama. Una vez que la trama a sido recibida esta se pasa al decodificador, si este sube la línea de datos quiere decir que la trama ha sido correcta, se deberá monitorizar la salida de la palabra que ha llegado y encender y apagar el LED para mostrarlo.

Si la recepción no ha sido correcta no se levantará la línea de datos y por lo tanto no se procederá a encender ningún LED. Hay que destacar que se debe realizar una pequeña modificación en la placa, el LED esta conectado a la alimentación y a la patilla LED del codificador. Por lo que para su uso se debe cortar la conexión con el decodificador y llevar una línea del expansor hasta este punto, de tal manera que la línea se tira a 0 lógico cuando se quiere encender el LED y se deja en 1 cuando se quiera mantener apagado.

Sin embargo, no se ha realizado esta modificación. Ya que el codificador no va a ser empleado para nada en esta placa. Por lo que se puede situar un valor en sus entradas que haga que se encienda el LED, aunque genere el código correspondiente este no va a ser empleado posteriormente. Se escoge que la línea del expansor la de COD_S0.

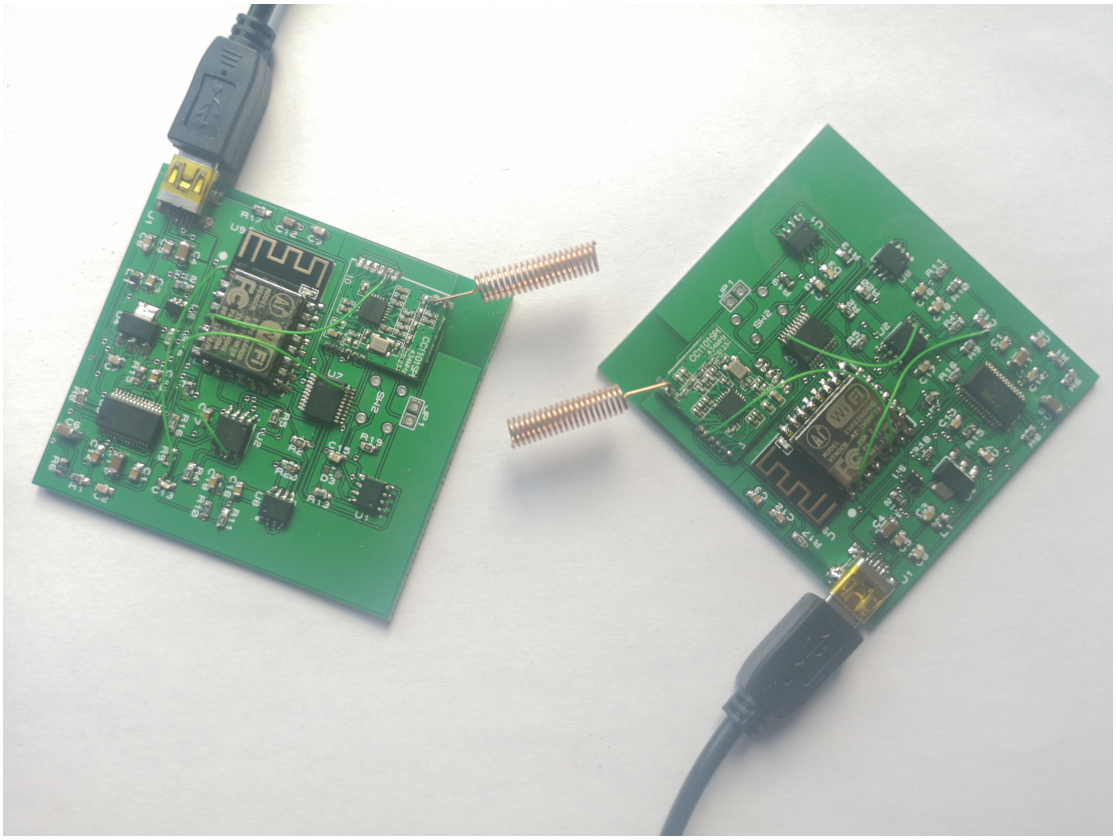

Figura 6.10: Montaje completo del sistema.

6.3. Líneas futuras

Habiendo evaluado los resultados conseguidos al termino del desarrollo del proyecto, siendo conscientes de sus defectos y fortalezas, se sugieren algunos cambios para el prototipo:

- **Mejora en el servidor Web:** Salta a la vista que el servidor Web es mejorable. Seria conveniente que primero se mostrase una página de registro en la que los usuarios pudieran registrarse (si es la primera vez que lo usan) o logearse. Y que una vez introducido su usuario y contraseña puedan disponer de un campo que les permita ver las puertas ya añadidas, modificarlas, añadir mas, realizar operaciones de apertura y cierre con estas...

- **Mejora en el prototipo:** En el prototipo , según se han detectado errores se han ido corrigiendo mediante un cableado añadido. Sería conveniente volver a realizar la fabricación de la placa con los errores ya corregidos.

- **Cambio de antena:** Durante la realización de las pruebas con el prototipo. Se observo que la emisión no era de mucha potencia y que estaba muy orientada en una dirección. Sería conveniente por lo tanto que la antena fuera integrada de tal manera que no se perdiera calidad debido a la unión en la soldadura de la antena y la placa.

- **Sustitución del transceptor:** Tras las pruebas realizadas y dado que el transceptor no tiene optimizados sus registros para una modulación ASK, para un nuevo diseño se escogería un transceptor que aunque no disponga de más tipos de modulación esté optimizado para ASK.

Bibliografía

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

[2] *ESP-12F WiFi Module. Version 1.0* Url: https://www.elecrow.com/download/ESP-12F.pdf

[3] *ESP8266 Community Forum.* Url: https://www.esp8266.com/index.php?sid=304a2774506efaf7142ca3da7103388d

[4] *Sverre Hellan. DN022 CC11xx OOK/ASK register settings.* Url: http://www.ti.com/lit/an/swra215e/swra215e.pdf

[5] *Siri Namtvedt. DN501 PATABLE Access.* Url: http://www.ti.com/lit/an/swra110b/swra110b.pdf

[6] *Mathias Jensen & Henrik Vatnar. DN010 Close-in Reception with CC1101.* Url: http://www.ti.com/lit/an/swra147b/swra147b.pdf

[7] *Texas Instrument. CC1101 Low-Power Sub-1 GHz RF Transceiver* Url: http://www.ti.com/lit/ds/symlink/cc1101.pdf

[8] *Microchip Technology Inc. HCS500 KEELOQ® Code Hopping Decoder* Url: https://www.mouser.es/datasheet/2/268/40153c-30068.pdf

[9] *Microchip Technology Inc. HCS300 KEELOQ® Code Hopping Encoder* Url: http://ww1.microchip.com/downloads/en/devicedoc/21137g.pdf

[10] *Microchip Technology Inc. An Introduction to KEELOQ® Code Hopping* Url: http://ww1.microchip.com/downloads/en/devicedoc/21143b.pdf

[11] *Espressif Systems. ESP8266EX Datasheet Versión 6.0* Url: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[12] *Microchip Technology Inc. MCP23008 8-Bit I/O Expander with Serial Interface* Url: http://ww1.microchip.com/downloads/en/DeviceDoc/MCP23008-MCP23S08-Data-Sheet-20001919F.pdf

[13] *Microchip Technology Inc. 24LC02B 2K I2C? Serial EEPROM* Url: http://ww1.microchip.com/downloads/en/DeviceDoc/24AA02-24LC02B-24FC02-Data-Sheet-20001709L.pdf

[14] *Future Technology Devices International Limited. FTDI 2005FT232R USB UART I.C.* Url: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf

[15] *Microchip Technology Inc. TC1262 500mA Fixed Output CMOS LDO* Url: http://ww1.microchip.com/downloads/en/DeviceDoc/21373C.pdf

Apéndice A

Código en C para el ESP-12F

```
/*----------------Includes---------------------------*/
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <WiFiClient.h>
#include <SPI.h>
/*----------------Pagina HTML----------------------*/
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>TFG Iciar Pascua</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
.boton_personalizado{
  text-decoration: none;
  padding: 10px;
  font-weight: 600;
  font-size: 20px;
  color: #ffffff;
  background-color: #1883ba;
  border-radius: 6px;
  border: 2px solid #0016b0;
}

.boton_personalizado:hover{
  color: #1883ba;
  background-color: #ffffff;
}

body {
  font-family: Arial, Helvetica, sans-serif;
  background-color: #f1f1f1;
  margin: 0;
}
.header {
  padding: 80px;
  text-align: center;
  background-image: url('https://images.unsplash.com/photo-
1532439771208-8fda693b96b0?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=750&q=80');
  color: white;
}
.header h1 {
  font-size: 40px;
}
h1   {color: white;}.btn {
  background-color: #7a7a52;
  border: none;
  color: white;
  padding: 12px 16px;
  font-size: 16px;
  cursor: pointer;
}
.btn:hover {
  background-color: #0f0f0a;
}
</style>
</head>
<body>
<center>
```

```
<div class="header">
  <h1>LET´S PARK</h1>
  <p>Una nueva forma de aparcar.</p>
</div>
<br/>
<a href="Aniadir" target="myIframe"><button class="btn">Añadir puerta
</button></a>
<a href="Abrir" target="myIframe"><button class="btn">Abrir puerta
</button></a>
<br/>
<br/>
<a href="Cerrar" target="myIframe"><button class="btn">Cerrar puerta
</button></a>
<br/>
<br/>
Informacion Adicional:<br>
<iframe name="myIframe" width="500" height="25" frameBorder="0"><br>
<hr>
</center>
</body>
</html>
)=====";

/*--------------Defines-----------------------------*/
#define CS 5          //Se trabaja con pines lógicos
#define DEC_DATA 4
#define DEC_CLK 16
#define EXPANSOR_CTRL 0x40
#define IODIR   0b00000000
#define INTCON  0b00000100
#define GPINTEN 0b00000010
#define IOCON   0b00000101
#define GPIO    0b00001001

/*------Declaracion de variables globales------*/
const char* ssid ="HUAWAEI P20 Pro";
const char* password = "123456789";
char estadoPuerta;
char operacionPuerta;
char CryptKey[8];
char datosFifoTX[40]; //Cada trasmision será de 28 bytes
char datosFifoRX[29];
ESP8266WebServer server(80);

/*-----------Declaración de funciones----------*/
void declaracionPines();
void escribirRegistroExpansor(char registro, char valor);
char leerRegistroExpansor(char registro);
void escribirRegistroTransceptor(char registro, char valor);
char leerRegistroTransceptor(char registro);
char comandStrobe(char registro);
void escribirRegistroPA();
void inicializacionExpansor();
void inicializacionTransceptor();
void inicializacionVariables();
void inicializacionWifi();
void handleRoot();
void handleAbrir();
void handleAprender();
void handleCerrar();
void escribirByteDecoder(char dato);
```

```
char leerEEPROM(char adress);
char escribirEEPROM(char adress, char dato);
void pasarByteDecodificador(char indice);
char leerBytePWM(char aux);
void leerPWMCodificador();
long long int modoAprendizajeDecodificador();
void escribirRegistroCodificador(char valor1, char valor2);
void realizarTransmionTX(char indice);
void realizarRecepcionRX();
void programarCodificador(long long int hoppingCode);
void ttoBoton();


/*------Funciones Principales Decdicadas a las inicializaciones  -----
------------*/

void declaracionPines()
{
  pinMode(CS,OUTPUT);       //Patilla de chip Select
  digitalWrite(CS, HIGH);   //La inicializo en alta, no esta
seleccionado aun
  pinMode(DEC_CLK,OUTPUT); //Reloj para la comunicacion con el
decodificador
  pinMode(DEC_DATA,INPUT); //Entrada de datos en la comunicacion con
el decodificador
}

void escribirRegistroExpansor(char registro, char valor)
{
   SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
   digitalWrite(CS,LOW);         //Seleccionamos el expansor
   SPI.transfer(EXPANSOR_CTRL); //Enviamos el opcode
   SPI.transfer(registro);       //Enviamos la direccion del registro
   SPI.transfer(valor);
   digitalWrite(CS,HIGH);        //Deseleccionamos el expansor
   SPI.endTransaction();
   wdt_reset();
}

char leerRegistroExpansor(char registro)
{
   char datos;
   SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
   digitalWrite(CS,LOW);       //Seleccionamos el expansor
   SPI.transfer(EXPANSOR_CTRL|0b00000001); //Enviamos el opcode para
lectura
   SPI.transfer(registro);     //Enviamos la direccion del registro
   datos=SPI.transfer(0);      //Enviamos un dummy byte
   digitalWrite(CS,HIGH);      //Deseleccionamos el expansor
   SPI.endTransaction();
   wdt_reset();
   return datos;
}

void escribirRegistroTransceptor (char registro, char valor)
{
  char aux,aux1;
  aux=leerRegistroExpansor(GPIO);
  escribirRegistroExpansor(GPIO,aux&0b10111111); //Seleccionamos el CS
del transceptor
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
```

```
  while(digitalRead(12)==HIGH);    //Debemos comprobar que SO esta en
LOW
  aux1=SPI.transfer(registro);     //Primero va un byte de header con
R/W, B, A5-A0
  aux1=SPI.transfer(valor);        //Situamos el valor que queremos
darle al registro
  Serial.print("");
  SPI.endTransaction();
  escribirRegistroExpansor(GPIO,aux|0b01000000); //Deseleccionamos el
transceptor
  wdt_reset();
}

char leerRegistroTransceptor(char registro)
{
  char aux,aux1,valor;
  aux=leerRegistroExpansor(GPIO);
  escribirRegistroExpansor(GPIO,aux&0b10111111); //Seleccionamos el
transceptor
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  while(digitalRead(12)==HIGH);     //Debemos comprobar que SO esta en
LOW
  aux1=SPI.transfer(registro);      //Primero va un byte de header con
R/W, B, A5-A0
  valor=SPI.transfer(0xff);
  //Serial.println(registro,BIN);
  SPI.endTransaction();
  escribirRegistroExpansor(GPIO,aux|0b01000000); //Deseleccionamos el
transceptor
  wdt_reset();
  Serial.println(valor,BIN);
  return valor;
}

char comandStrobe(char registro)
{
  char aux,aux1;
  aux=leerRegistroExpansor(GPIO);
  escribirRegistroExpansor(GPIO,aux&0b10111111); //Seleccionamos el
transceptor
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  while(digitalRead(12)==HIGH);       //Debemos comprobar que SO esta
en LOW
  aux1=SPI.transfer(registro);        //Byte de header con R/W, B, A5-
A0
  Serial.println(aux1,BIN);
  SPI.endTransaction();
  escribirRegistroExpansor(GPIO,aux|0b01000000); //Deselecciono el
transceptor
  wdt_reset();
  return aux1;
}

void escribirRegistroPA()
{
  char aux,aux1;
  aux=leerRegistroExpansor(GPIO);
  escribirRegistroExpansor(GPIO,aux&0b10111111); //Selecciono el
transceptor
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
```

```
  while(digitalRead(12)==HIGH);     //Debemos comprobar que SO esta en
LOW
  SPI.transfer(0b00111110);         //Byte de header con R/W, B, A5-A0
  SPI.transfer(0b00000000);         //Valor que queremos darle al
registro

  SPI.transfer(0b00111110);         //Byte de header con R/W, B, A5-A0
  SPI.transfer(0b11000000);         //Despues situamos el valor para
10dbm de emision, debemos escribir primero el 0 para tener acceso l 1

  Serial.println("");
  SPI.endTransaction();
  escribirRegistroExpansor(GPIO,aux|0b01000000); //Deseleccionamos el
transceptor
  wdt_reset();
}

void inicializacionExpansor()
{

  Serial.println("Iniciando Expansor");
  escribirRegistroExpansor(IODIR,0b00010000);//Valor a
1=entrada,0=salida
                                             //GP6 CS para el
transceptor
                                             //GP5 PWM
                                             //GP4 open
                                             //GP3 COD_S3
                                             //GP2 COD_S2
                                             //GP1 COD_S1
                                             //GP0 COD_S0

  escribirRegistroExpansor(INTCON,0b00000000);
  escribirRegistroExpansor(GPINTEN,0b00010000);
  escribirRegistroExpansor(IOCON,0b00111010);
  escribirRegistroExpansor(GPIO, 0b01000000);  //Damos un valor
inicial a los pines ddel expansor
  Serial.println("Expansor Iniciado");
}

void inicializacionTransceptor()
{
    //escribirRegistroTransceptor(0b00000000,0b00011101);//Habilitamos
el pin GD2 como pin de salida TX
    //escribirRegistroTransceptor(0b00000010,0b00001101);//Habilitamos
el pin GD0 como pin de salida TX
    escribirRegistroTransceptor(0b00000011,0b01000111); //Establecemos
que quepan 41bytes en TX y 24 en RX
    escribirRegistroTransceptor(0b00000100,0b10101010); //SYNC1
    escribirRegistroTransceptor(0b00000101,0b00000000); //SYNC0
    escribirRegistroTransceptor(0b00000110,0b00011001); //Escribimos
PKTLEN
    escribirRegistroTransceptor(0b00000111,0b00000000); //Escribimos
PKTCTRL1
    escribirRegistroTransceptor(0b00001000,0b00000000); //Escribimos
PKTCTRL0
    escribirRegistroTransceptor(0b00001011,0b00000110); //Escribimos
FSCTRL1
    escribirRegistroTransceptor(0b00001101,0b00010000); //Escribimos
FREC2
```

```
    escribirRegistroTransceptor(0b00001110,0b10110111); //Escribimos
FREC1
    escribirRegistroTransceptor(0b00001111,0b11111010); //Escribimos
FREC0
    escribirRegistroTransceptor(0b00010000,0b11000110); //Escribimos
MDMCFG4
    escribirRegistroTransceptor(0b00010001,0b10101010); //Escribimos
MDMCFG3
    escribirRegistroTransceptor(0b00010010,0b00110110); //Escribimos
MDMCFG2 Establecemos modulacion ASK/OOK y sin preambulo
    escribirRegistroTransceptor(0b00010011,0b00000000); //Escribimos
MDMCFG1
    escribirRegistroTransceptor(0b00010100,0b00000000); //Escribimos
MDMCFG0
    escribirRegistroTransceptor(0b00010101,0b00010101); //Escribimos
DEVIATN
    escribirRegistroTransceptor(0b00010110,0b00000111); //Escribimos
MCSM2
    escribirRegistroTransceptor(0b00011000,0b00011000); //Escribimos
MCSM0
    escribirRegistroTransceptor(0b00011001,0b00010100); //Escribimos
FOCCFG
    escribirRegistroTransceptor(0b00100000,0b11111011); //Escribimos
WORCTL
    escribirRegistroTransceptor(0b00100010,0b00010001); //Escribimos
FREND0
    escribirRegistroTransceptor(0b00100011,0b11101001); //Escribimos
FSCAL3
    escribirRegistroTransceptor(0b00100100,0b00101010); //Escribimos
FSCAL2
    escribirRegistroTransceptor(0b00100101,0b00000000); //Escribimos
FSCAL1
    escribirRegistroTransceptor(0b00100110,0b00011111); //Escribimos
FSCAL0
    escribirRegistroTransceptor(0b00101100,0b10000001); //Escribimos
TEST2
    escribirRegistroTransceptor(0b00101101,0b00110101); //Escribimos
TEST1
    escribirRegistroTransceptor(0b00101110,0b00001001); //Escribimos
TEST0

    escribirRegistroTransceptor(0b00011011,0b00000111); //AGCCTRL2 7
    escribirRegistroTransceptor(0b00011100,0b00110000); //AGCCTRL1 0
    escribirRegistroTransceptor(0b00011101,0b10010011); //AGCCTRL0 92
}

void inicializacionVariables()
{
  char i;
  estadoPuerta=0;
  for(i=0;i<40;i++)
  {
     datosFifoTX[i]=0;
  }
  datosFifoRX[0]=0xAA; //Preambulo H
  datosFifoRX[1]=0xAA; //Preambulo L
  datosFifoRX[2]=0xAA; // SYNC H
  datosFifoRX[3]=0x00; // SYNC L
}
```

```
void inicializacionWifi()
{
   Serial.println();
   Serial.println(),
   Serial.print("Conectandose a red: ");
   Serial.println(ssid);
   WiFi.mode(WIFI_STA);        //Se introduce al ESP en modo STATION
   WiFi.disconnect();
   WiFi.begin(ssid,password); //Conexion a la red

   while(WiFi.status()!=WL_CONNECTED)
   {
     delay(500);
     wdt_reset();
     Serial.print(".");
   }
   Serial.println("");
   Serial.println("WiFi conectado");
   wdt_reset();
   server.on("/",handleRoot);            //Para mostrar la pantalla
principal
   server.on("/Abrir",handleAbrir);      //Para manejar el boton abrir
   server.on("/Aniadir",handleAprender);//Para manejar el boton
aniadir
   server.on("/Cerrar",handleCerrar);    //Para manejar el boton cerrar
   server.begin();                       //Iniciamos el servidor
   Serial.println("Servidor Iniciado");
   Serial.println("Ingrese desde un navegador web usando la IP: ");
   Serial.println(WiFi.localIP());       //Obtenemos la IP

}

/*------Funciones Principales Decdicadas al tratamiento de datos  ----
-------------*/

void handleRoot() {
  String s=MAIN_page; //contenido HTML
  server.send(200,"text/html",s);
}

void handleAbrir(){
  unsigned char aux;
  inicializacionTransceptor();
  Serial.println("Se ha seleccionado abrir");
  escribirRegistroExpansor(IODIR,0b00010000);
  //Modificamos una de las lineas de control del transmisor y
recogemos el codigo emitido
  aux=leerRegistroExpansor(GPIO);
  aux=aux&0b11111100;
  escribirRegistroExpansor(GPIO, aux); //Empiezo con S0 y S1 a 0
  //Para recoger el código PWM entramos en modo trasmision sincrona
  aux=aux|0b00000100;
  escribirRegistroExpansor(GPIO,aux);  //Pongo la linea S2 en alta (va
a ser el reloj que controle la salida de pwm)
  delay(4);//Espero TPS
  aux=aux|0b00100000;
  escribirRegistroExpansor(GPIO,aux);  //Subo PWM
  delay(2);//Espero TPH1
  //Pongo el codigo deseado
  aux=leerRegistroExpansor(GPIO);
  aux=aux|0b00000001;
```

```
  escribirRegistroExpansor(GPIO,aux);  //Pongo un uno en COD_S0
  delay(2); //Espero el resto de TPH1
  aux=aux&0b11011111;
  escribirRegistroExpansor(GPIO, aux); //Bajo la linea de PWM
  delayMicroseconds(50);               //Espero TPH2
  aux=aux&0b11111011;
  escribirRegistroExpansor(GPIO, aux); //Pongo la linea S2 en baja (en
este moemnto se establce el codigo de operacion)
  delay(50);
  escribirRegistroExpansor(IODIR,0b00110000);
  leerPWMCodificador();
  realizarTransmionTX(29);             //Enviamos los bytes leidos
  aux=leerRegistroExpansor(GPIO);
  aux=aux&0b11111100;
  escribirRegistroExpansor(GPIO, aux); //Rrestauramos el valor de S0
  server.send(200,"text/html", "Mensaje de apertura enviado");

}

void handleAprender(){
  long long int hoppingCode;
  Serial.println("Se ha seleccionado añadir");
   hoppingCode=modoAprendizajeDecodificador(); //Activamos el modo
aprendizaje
   if(hoppingCode==0)
      {
          //Ha habido un error
          server.send(200,"text/html", "Error en el aprendizaje");
      }
   else
      {
           //Escribimos en el codificador los datos aprendidos
           Serial.println("Llave memorizada con exito");
           //programarCodificador(hoppingCode);
           server.send(200,"text/html", "Llave memorizada con
exito");
      }
}

void handleCerrar(){
  unsigned char aux;
  Serial.println("Se ha seleccionado cerrar");
  escribirRegistroExpansor(IODIR,0b00010000);
  //Modificamos una de las lineas de control del transmisor y
recogemos el codigo emitido
  aux=leerRegistroExpansor(GPIO);
  aux=aux&0b11111100;
  escribirRegistroExpansor(GPIO, aux);  //Empiezo con S0 y S1 a 0
  //Para recoger el código pwm entramos en modo trasmision sincrona
  aux=aux|0b00000100;
  escribirRegistroExpansor(GPIO,aux);   //Pongo la linea S2 en alta
(va a ser el reloj que controle la salida de pwm)
  delay(4);                             //Espero TPS
  aux=aux|0b00100000;
  escribirRegistroExpansor(GPIO,aux);   //Subo PWM
  delay(2);                             //Espero TPH1
  //Pongo el codigo deseado
  aux=leerRegistroExpansor(GPIO);
  aux=aux|0b00000001;
  escribirRegistroExpansor(GPIO,aux);   //Pongo un uno en COD_S0
  delay(2);                             //Espero el resto de TPH1
```

```
  aux=aux&0b11011111;
  escribirRegistroExpansor(GPIO, aux);  //Bajo la linea de PWM
  delayMicroseconds(50);//Espero TPH2
  aux=aux&0b11111011;
  escribirRegistroExpansor(GPIO, aux);  //Pongo la linea S2 en baja
(en este moemnto se establce el codigo de operacion)
  delay(50);
  escribirRegistroExpansor(IODIR,0b00110000);
  leerPWMCodificador();
  realizarTransmionTX(29);              //Enviamos los bytes leidos
  aux=leerRegistroExpansor(GPIO);
  aux=aux&0b11111100;
  escribirRegistroExpansor(GPIO, aux);  //Acabo con S0 y S1 a 0
  server.send(200, "text/html", "Mensaje de cierre enviado");
}


void escribirByteDecoder(char dato)
{
  char i;
  for(i=0;i<7;i++)
  {
    digitalWrite(DEC_CLK, HIGH);
    digitalWrite(DEC_DATA,(dato>>i)&0b1); //Aqui pongo el dato con el
LSB primero
    delayMicroseconds(500);
    digitalWrite(DEC_CLK, LOW);
    delayMicroseconds(500);
  }
  digitalWrite(DEC_CLK, HIGH);
  digitalWrite(DEC_DATA,(dato>>7)&0b1);  //Aqui pongo el MSB
  delayMicroseconds(500);
  digitalWrite(DEC_CLK, LOW);
  delayMicroseconds(500);
  wdt_reset();
}

char escribirEEPROM(char adress, char dato)
{
  pinMode(DEC_DATA,INPUT); //PASA A SER LA ENTRADA DE DATOS
  //primero va la secuencia de inicializacion
  digitalWrite(DEC_CLK, HIGH);
  while(digitalRead(DEC_DATA)==LOW);
  delay(1);
  digitalWrite(DEC_CLK, LOW);
  while(digitalRead(DEC_DATA)==HIGH);
  delay(1);
  //Fin de la secuencia de inializacion
  pinMode(DEC_DATA,OUTPUT);   //PASA A SER LA SALIDA DE DATOS
  escribirByteDecoder(0xE1);  //Situamos el byte de comando, para
escribir es 0xE1
  escribirByteDecoder(adress);//Adress Byte
  escribirByteDecoder(dato);  //Adress Byte
  //espero Twr
  digitalWrite(DEC_CLK, LOW);
  delayMicroseconds(500);
  pinMode(DEC_DATA,INPUT);    //PASA A SER LA ENTRADA DE DATOS
  digitalWrite(DEC_CLK, HIGH);
  while(digitalRead(DEC_DATA)==LOW);
  delay(1);                   //Espero Tresp
  digitalWrite(DEC_CLK, LOW);
```

```
  while(digitalRead(DEC_DATA)==HIGH);
}

char leerEEPROM(char adress)
{
  unsigned char i;
  char dato=0;

  //Secuencia de inicializacion
  pinMode(DEC_DATA,INPUT); //PASA A SER LA SALIDA DE DATOS
  digitalWrite(DEC_CLK, HIGH);
  while(digitalRead(DEC_DATA)==LOW);
  delay(1);
  digitalWrite(DEC_CLK, LOW);
  while(digitalRead(DEC_DATA)==HIGH);
  delay(1);
 //Fin de la secuencia de inializacion

 //Datos del comando  0XF0
  pinMode(DEC_DATA,OUTPUT); //PASA A SER LA SALIDA DE DATOS
  escribirByteDecoder(0xF0); //Comando
  escribirByteDecoder(adress);//Adress Byte
  escribirByteDecoder(0x00);//Dummy byte 0XFF
  //espero TRD
  digitalWrite(DEC_CLK, LOW);
  delayMicroseconds(500);
  pinMode(DEC_DATA,INPUT); //PASA A SER ENTRADA DE DATOS
  digitalWrite(DEC_CLK, HIGH);

  //Recibo el dato
  for(i=0;i<7;i++)
  {
      delayMicroseconds(250);
      dato=dato|(digitalRead(DEC_DATA)<<i);
      delayMicroseconds(250);
      digitalWrite(DEC_CLK, LOW);
      delayMicroseconds(500);
      digitalWrite(DEC_CLK, HIGH);
  }
  delayMicroseconds(250);
  dato=dato|(digitalRead(DEC_DATA)<<7);
  delayMicroseconds(250);
  digitalWrite(DEC_CLK, LOW);
  delayMicroseconds(500);
  return dato;
}

void pasarByteDecodificador(char indice)
{
  char aux,i;
  Serial.println(indice);
  aux=leerRegistroExpansor(GPIO);
  if(indice==28) //El ultimo bit del ultimo byte no es válido
  {
    for(i=0;i<7;i++)
     {
        if((datosFifoRX[indice]>>(7-i))&0b1)
        {
          //El bit es un 1
           aux=aux|0b10000000;
           escribirRegistroExpansor(GPIO,aux);
```

```
            delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
          else
          {
            //El bit es un 0
            aux=aux&0b01111111;
            escribirRegistroExpansor(GPIO,aux);
            delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
      }
    }
    else
    {
      for(i=0;i<8;i++)
       {
          if((datosFifoRX[indice]>>(7-i))&0b1)
          {
            //El bit es un 1
             aux=aux|0b10000000;
             escribirRegistroExpansor(GPIO,aux);
             delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
          else
          {
            //El bit es un 0
            aux=aux&0b01111111;
            escribirRegistroExpansor(GPIO,aux);
            delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
      }
    }
    wdt_reset();
}

char leerBytePWM(char aux)
{
  char i ,dato=0;
  char pwm;

  for(i=0;i<8;i++)
  {
    aux=aux|0b00000100;
    escribirRegistroExpansor(GPIO,aux);      //Subo el reloj (s2)
    delayMicroseconds(5);
    pwm=(leerRegistroExpansor(GPIO)>>5)&0b1; //Me quedo solo con el
valor de PWM
    dato=dato|(pwm<<(7-i));
    aux=aux&0b11111011;
    escribirRegistroExpansor(GPIO,aux);      //Bajo el reloj (s2)
    delayMicroseconds(30);
  }
  return dato;

}

void leerPWMCodificador()
{
```

```
  char aux,i;
  char j=0;
  aux=leerRegistroExpansor(GPIO);
  for(i=0;i<2;i++)
  {
     aux=aux|0b00000100;
     escribirRegistroExpansor(GPIO,aux); //Subo el reloj (s2)
     delayMicroseconds(30); //Tarda en subir y bajar 24us y
necesitamos que sean como minimo 25
     aux=aux&0b11111011;
     escribirRegistroExpansor(GPIO,aux); //Bajo el reloj (s2)
     delayMicroseconds(30);
  }
  wdt_reset();
  for(i=0;i<39;i++)
  {
    datosFifoTX[i]=leerBytePWM(aux);
  }
  for(i=0;i<3;i++) //Realizo tres periodos de reloj
  {
     aux=aux|0b00000100;
     escribirRegistroExpansor(GPIO,aux); //Subo el reloj (s2)
     delayMicroseconds(30); //Tarda en subir y bajar 24us y
necesitamos qe sean como minimo 25
     aux=aux&0b11111011;
     escribirRegistroExpansor(GPIO,aux); //bajo el reloj (s2)
     delayMicroseconds(30); //Tarda en subir y bajar 24us y
necesitamos qe sean como minimo 25
  }

  //Elimino los 16 bits añadidos al principio de la palabra
  datosFifoTX[4]=datosFifoTX[4]&0b10000000;
  aux=(datosFifoTX[10]&0b01111111);
  datosFifoTX[4]=datosFifoTX[4]|aux;
  j=0;
  for(i=5;i<29;i++)
  {
    datosFifoTX[i]=datosFifoTX[11+j];
    j++;
  }
  for(i=29;i<35;i++)
  {
    datosFifoTX[i]=0;
  }
  wdt_reset();
}

long long int modoAprendizajeDecodificador()
{
  char i=0;
  char statusByte=0;
  int secndStatusByte=0;
  long long int hoppingCode=0;

  //METO AL DECODIFICADOR EN MODO LEARN
  //primero va la secuencia de inicializacion
  pinMode(DEC_DATA,INPUT); //PASA A SER LA SALIDA DE DATOS
  digitalWrite(DEC_CLK, HIGH);
  while(digitalRead(DEC_DATA)==LOW);
  delay(1);
  digitalWrite(DEC_CLK, LOW);
```

```
  while(digitalRead(DEC_DATA)==HIGH);
  delay(1);
  //Fin de la secuencia de inializacion

  //Datos del comando  0XD2
  pinMode(DEC_DATA,OUTPUT); //PASA A SER LA SALIDA DE DATOS
  escribirByteDecoder(0XD2);//COMANDO
  escribirByteDecoder(0XFF);//Dummy byte
  escribirByteDecoder(0XFF);//Dummy byte

  //espero TLRN
  digitalWrite(DEC_CLK, LOW);
  delayMicroseconds(500);
  pinMode(DEC_DATA,INPUT); //PASA A SER ENTRADA DE DATOS
  digitalWrite(DEC_CLK, HIGH);
  //ACK del decodificador
  while(digitalRead(DEC_DATA)==LOW);
  //espero TRESP
  delay(1);
  digitalWrite(DEC_CLK, LOW);
  while(digitalRead(DEC_DATA)==HIGH);

  //MUESTRO POR PANTALLA QUE ESTOY LISTO PARA APRENDER
  Serial.print("Modo aprendizaje activo, pulse una tecla del mando
\n");
  server.send(200,"text/html", "Presione una tecla del mando");
  //recojo el mensaje de la primera pulsacion
  realizarRecepcionRX();
  wdt_reset();
  //Se la paso al decodifcador empleando el GPIO 7 del expansor
  for(i=0;i<29;i++) //Se transmiten 29Bytes
   {
      pasarByteDecodificador(i);
   }
  //Esperamos a que acepte el codigo transmitido
  while(digitalRead(DEC_DATA)==LOW)
   {
      Serial.print(".");
      delay(400);
      wdt_reset();
   }
   //Espero Tcla y si sigue en alta subo el reloj
   delay(60);
   if(digitalRead(DEC_DATA)==HIGH)
    {
      digitalWrite(DEC_CLK, HIGH);
      while(digitalRead(DEC_DATA)==HIGH)
      {
        wdt_reset();
      }
      delay(1);//tclh

      digitalWrite(DEC_CLK, LOW);
      delayMicroseconds(500); //tcll
      for(i=0;i<8;i++)
      {
        delayMicroseconds(500);
        digitalWrite(DEC_CLK, HIGH);
        delayMicroseconds(250);
        statusByte=statusByte|(digitalRead(DEC_DATA)<<(i));
        delayMicroseconds(250);
```

```
        digitalWrite(DEC_CLK, LOW);
      }
      delayMicroseconds(500);
      digitalWrite(DEC_CLK, LOW);
      wdt_reset();

      //Muestro que la primera lectura se ha realizado y tengo que
pulsar de nuevo
      Serial.println(statusByte,BIN);
      Serial.println("Repita la pulsación \n");
      server.send(200,"text/html", "Repita la pulsación");

      //recojo el mensaje de la primera pulsacion
      realizarRecepcionRX();
      wdt_reset();
      //Se la paso al decodifcador empleando el GPIO 7 del expansor
      for(i=0;i<29;i++) //Se transmiten 28Bytes
      {
        pasarByteDecodificador(i);
      }
      //Esperamos a que acepte el codigo transmitido
      while(digitalRead(DEC_DATA)==LOW)
      {
        Serial.print(".");
        delay(300);
        wdt_reset();
      }
      delay(2);
      digitalWrite(DEC_CLK, HIGH);
      while(digitalRead(DEC_DATA)==HIGH);
      delay(1);
      digitalWrite(DEC_CLK, LOW);
      delayMicroseconds(500); //tcll
      for(i=0;i<16;i++)
      {
        delayMicroseconds(500);
        digitalWrite(DEC_CLK, HIGH);
        delayMicroseconds(250);
        secndStatusByte=secndStatusByte| digitalRead(DEC_DATA)<<(i);
        delayMicroseconds(250);
        digitalWrite(DEC_CLK, LOW);
      }
      wdt_reset();
      for(i=0;i<64;i++)
      {
        delayMicroseconds(500);
        digitalWrite(DEC_CLK, HIGH);
        delayMicroseconds(250);
        hoppingCode=hoppingCode|(digitalRead(DEC_DATA)<<(i));
        delayMicroseconds(250);
        digitalWrite(DEC_CLK, LOW);
        wdt_reset();
      }
      delayMicroseconds(500);
      //Muestro que se ha finalizado el proceso
      Serial.print("Segunda pulsacion detectada , código almacenado
\n");
      return  hoppingCode; //Para poder almacerarlo
    }
     Serial.println("Error");
   return 0; //Error
```

```
}

void escribirRegistroCodificador(char valor1, char valor2)
{
  unsigned char i;
  char valor=0;
  for(i=0;i<8;i++) //Le paso los datos obtenidos del decodificador
  {
    if(((valor1>>i)&0b1))
    {
      escribirRegistroExpansor(GPIO,0b00100100); //Pongo pwm en alta
      valor=0b00100100;
    }
    else
    {
      escribirRegistroExpansor(GPIO,0b00000100); //Pongo pwm en baja
      valor=0b00000100;
    }
    delayMicroseconds(50);
    escribirRegistroExpansor(GPIO,valor&0b11111011); //Pongo S2 en
baja
    delayMicroseconds(50);
    escribirRegistroExpansor(GPIO,valor|0b00000100); //Pongo S2 en
alta
  }

  for(i=0;i<8;i++) //Le paso los datos obtenidos del decodificador
  {

    if(((valor2>>i)&0b1))
    {
      escribirRegistroExpansor(GPIO,0b00100100); //Pongo pwm en alta
      valor=0b00100100;
    }
    else
    {
      escribirRegistroExpansor(GPIO,0b00000100); //Pongo pwm en baja
      valor=0b00000100;
    }
    delayMicroseconds(50);
    escribirRegistroExpansor(GPIO,valor&0b11111011); //Pongo S2 en
baja
    delayMicroseconds(50);
    escribirRegistroExpansor(GPIO,valor|0b00000100); //Pongo S2 en
alta

  }
  escribirRegistroExpansor(GPIO,valor&0b11111011); //Pongo S2 en baja
  wdt_reset();
  //Espero TWC
  delay(50);
  wdt_reset();
}

void realizarTransmionTX(char indice)
{
  char i;
  escribirRegistroPA(); //Escribimos la potencia a la que queremos
transmitir
  comandStrobe(0b00110110); //Le meto en modo idle COMANDO 0x36
```

```
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x01) //Espero a
que el estado este bien
    {
      wdt_reset();
    }
  for(i=4;i<indice;i++)
    {
       escribirRegistroTransceptor(0b00111111,datosFifoTX[i]);
//Escribo en la FIFO
       delay(1);
    }
  delay(2);
  comandStrobe(0b00110110); //Le meto en modo idle COMANDO 0x36
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x01) //Espero a
que el estado este bien
   {
     wdt_reset();
   }
  comandStrobe(0b00110101); //Le meto en modo trasmision COMANDO 0x35
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x13) //Espero a
que el estado este bien
   {
     wdt_reset();
   }
  Serial.println("Enviado");
  delay(500);
  wdt_reset();
}

void realizarRecepcionRX()
{
  char i;
  Serial.println("Activando recepcion");
  delay(500);
  comandStrobe(0b00110110); //Le meto en modo idle COMANDO 0x36
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x01) //Espero a
que el estado este bien
   {
     wdt_reset();
   }
  comandStrobe(0b00111010); //Borro a FIFO de recepcion 0x3A
  delay(2);
  comandStrobe(0b00110100); //Le meto en modo recepcion COMANDO 0x34
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x0D) //Espero a
que el estado este bien
   {
     wdt_reset();
   }
  delayMicroseconds(100);
  Serial.println("Modo recepcion activo");

  //Leo el numero de Bytes disponible y mientras sea menor que 28
espero
  while((leerRegistroTransceptor(0b11111011))<25)
   {
     delay(15);
     wdt_reset();
   }
   if(leerRegistroTransceptor(0b11111011)&0x80)//Significa que esta
activo el bit de overflow
   {
```

```
      comandStrobe(0b00110110); //Le meto en modo idle COMANDO 0x36
      while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x01) //Espero
a que el estado este bien
      {
        wdt_reset();
      }
      comandStrobe(0b00111010); //Le mando el SFRX para restaurar el
buffer 0x3A
      Serial.println("Recepcion fallida");
   }
   else //Tenemos ya los veintinueve bytes listos
   {
      for(i=4;i<29;i++)
      {
        datosFifoRX[i]=leerRegistroTransceptor(0b10111111);
      }
      Serial.println("Recepcion exitosa");
   }
}

void programarCodificador(long long int hoppingCode)
{
  char i,valor;
  long int serialNumber;
  int  counter=0, aux1=0, discrimination;
  long int aux;

  //del hopping code obtengo el serial number y el counter
  serialNumber= ((hoppingCode>>32)&0x0FFFFFFF); //Lo unico que lo
tenemos los bits al reves LSB
  counter=(hoppingCode&0xFFFF000000000000)>>48; //Estan al reves de
menos a mas significativo
  discrimination=(serialNumber>>16); //Son los 12 bits menos
significativos del numero de serie
  //Obtengo la crypt key de la memoria eeprom
  for(i=0;i<8;i++)
  {
    CryptKey[i]=leerEEPROM(4+i); //Empiezo a leer en la direccion
cuatro por que antes se guarda el numero de serie
  }

  //Secuencia de inicializacion
  escribirRegistroExpansor(GPIO,0b00000100); //Pongo S2 en alta
  delay(4);
  escribirRegistroExpansor(GPIO,0b00100100); //Pongo PWM en alta
  delay(3);
  delayMicroseconds(500);
  escribirRegistroExpansor(GPIO,0b00000100); //Pongo pwm en baja
  escribirRegistroExpansor(GPIO,0b00000000); //Pongo S2 en baja
  delay(4);
  escribirRegistroExpansor(GPIO,0b00000100); //Pongo S2 en alta
  wdt_reset();
  //PRIMERA PALABRA (KEY0)
  escribirRegistroCodificador(CryptKey[0], CryptKey[1]);
  //SEGUNDA PALABRA (KEY1)
  escribirRegistroCodificador(CryptKey[2], CryptKey[3]);
  //TERCERA PALABRA (KEY2)
  escribirRegistroCodificador(CryptKey[4],CryptKey[5]);
  //CUARTA PALABRA (KEY3)
  escribirRegistroCodificador(CryptKey[6],CryptKey[7]);
  //ESCRIBIMOS EL VALOR DE SINCRONIZACION
```

```
  escribirRegistroCodificador((counter>>8),(counter&0xFF));
  //CAMPO RESERVADO TODO A 0
  escribirRegistroCodificador(0x00,0x00);
  //ESCRIBIMOS EL NUMERO DE SERIE
  escribirRegistroCodificador((serialNumber>>24),((serialNumber>>16)&0
xFF));
  escribirRegistroCodificador(((serialNumber>>8)&0xFF),((serialNumber&
0xFF)|0x80)); //Le hago la OR para poner habilitado el auto-shutoff
  //LA SEMILLA NO LA SABEMOS
  escribirRegistroCodificador(0x00,0x00);
  escribirRegistroCodificador(0x00,0x00);
  //CAMPO RESERVADO TODO A 0
  escribirRegistroCodificador(0x00,0x00);
  //PALABRA DE CONFIGURACION
  escribirRegistroCodificador(discrimination>>8,discrimination&0xC0);
}

void ttoBoton()
{
  char valorPuerto;
  //Como no he cableado el pin de INT del expansor no podre ver si se
produce una interrupcion
  //por lo que en vez de manejarlo por interrupciones estaré
escaneandolo de forma continua
  valorPuerto=leerRegistroExpansor(GPIO);
  Serial.println(valorPuerto,BIN);
  if(valorPuerto&0b00010000)
  {
      //Significa que tenemos el boton sin pulsar luego no haccemos
nada;
  }
  else
  {
    if(estadoPuerta==0)
    {
      //Debemos abrir la puerta y conmutar el estado
      Serial.println("Abriendo Puerta");
      estadoPuerta=1;
      handleAbrir();
    }
    else
    {
      //Debemos cerrar la puerta y conmutar el estado
      Serial.println("Abriendo Puerta");
      estadoPuerta=0;
      handleCerrar();
    }
  }
}

void setup() {
  char valorPuerto,aux,i;
  Serial.begin(9600);  //Abrimos la comunicacion con el puerto serie
para realizar depuración
  delay(10);
  declaracionPines();            //Declaramos los pines del ESP que
van a actuar como salidas o entradas
  SPI.pins(14,12,13,5);          //Para que el ESP emplee HSPI se
señaliza el GPIO12,13,14
  SPI.begin();                   //Inicializamos la comunicacion SPI a
10Mhz
```

```
  wdt_reset();                    //Se debe resetear el perro guardián
a menudo para que no nos produzca un reset
  inicializacionExpansor();       //Inicializamos el comportamiento del
expansor
  wdt_reset();
  inicializacionVariables();      //Establecemos los valores iniciales
de las variables principales
  wdt_reset();
  inicializacionWifi();           //Inicializamos la comunicacion WiFi
  wdt_reset();
}

void loop(void) {

  server.handleClient();  //Realizamos el tratamiento de las
peticiones web
  wdt_reset();
  ttoBoton();             //Tratamiento de la entrada aux (pulsador)
  wdt_reset();
}
```

Apéndice B

Código en C para el ESP-12F emulando una puerta de garaje

```
/*------Includes--------------------------------*/
#include <SPI.h>

/*------Defines---------------------------------*/
#define CS 5          //Se trabaja con pines lógicos
#define DEC_DATA 4
#define DEC_CLK 16
#define EXPANSOR_CTRL 0x40
#define IODIR   0b00000000
#define INTCON  0b00000100
#define GPINTEN 0b00000010
#define IOCON   0b00000101
#define GPIO    0b00001001

/*------Declaracion de variables globales------*/
char CryptKey[8];
char datosFifoRX[29];

/*-----------Declaración de funciones----------*/

void escribirRegistroExpansor(char registro, char valor); //Funcion
Verificada
char leerRegistroExpansor(char registro); //Funcion Verificada
void escribirRegistroTransceptor(char registro, char valor);//Funcion
Verificada
char leerRegistroTransceptor(char registro);//Funcion Verificada
char comandStrobe(char registro); //Funcion verificada
void escribirRegistroPA(); //Funcion Verificada
void escribirByteDecoder(char dato);//Funcion Verificada
void declaracionPines();//Funcion Verificada
void inicializacionExpansor();//Funcion Verificada
void inicializacionTransceptor();//Funcion Verificada   ----PREGUNTAR
POR LA POTENCIA Y LA FRECUENCIA-----
void inicializacionVariables();//Funcion Verificada
char realizarRecepcionRX(); //Funcion Verificada
void escribirRegistroDecodificador(char valor1, char valor2);
void pasarByteDecodificador(char indice);
void ttoPuerta();
/*------Funciones Principales Decdicadas a las inicializaciones  -----
------------*/

void declaracionPines()
{
  pinMode(CS,OUTPUT); //Patilla de chip Select
  digitalWrite(CS, HIGH); //La inicializo en alta, no esta
seleccionado aun
  pinMode(DEC_CLK,OUTPUT);  //Patillade salida del reloj para la
comunicacion con el decodificador
  pinMode(DEC_DATA,INPUT); //Patilla de entrada de datos para la
comunicacion con el decodificador
}

void escribirRegistroExpansor(char registro, char valor)
{
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  digitalWrite(CS,LOW); //Seleccionamos el expansor
  SPI.transfer(EXPANSOR_CTRL); //Enviamos el opcode
  SPI.transfer(registro); //Enviamos la direccion del registro
  SPI.transfer(valor);
  digitalWrite(CS,HIGH); //Deseleccionamos el expansor
  SPI.endTransaction();
```

```
  wdt_reset();
}

char leerRegistroExpansor(char registro)
{
  char datos;
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  digitalWrite(CS,LOW); //Seleccionamos el expansor
  SPI.transfer(EXPANSOR_CTRL|0b00000001); //Enviamos el opcode con el
ultimo bit a uno ya que es una lectura
  SPI.transfer(registro); //Enviamos la direccion del registro
  datos=SPI.transfer(0); //Enviamos un dummy byte
  digitalWrite(CS,HIGH); //Deseleccionamos el expansor
  SPI.endTransaction();
  wdt_reset();
  return datos;
}

void escribirRegistroTransceptor (char registro, char valor)
{
  char aux,aux1;
    aux=leerRegistroExpansor(GPIO);
    escribirRegistroExpansor(GPIO,aux&0b10111111); //Para poner un 0
en el chip select del transceptor
    SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    while(digitalRead(12)==HIGH); //Debemos comprobar que SO esta en
LOW
    aux1=SPI.transfer(registro);   //Primero va un byte de header con
R/W, B, A5-A0
    aux1=SPI.transfer(valor);   //Despues situamos el valor que
queremos darle al registro
    Serial.print("");
    SPI.endTransaction();
    escribirRegistroExpansor(GPIO,aux|0b01000000); //Para poner un 1
en el chip select del transceptor
    wdt_reset();
}
char leerRegistroTransceptor(char registro)
{
  char aux,aux1,valor;
    aux=leerRegistroExpansor(GPIO);
    escribirRegistroExpansor(GPIO,aux&0b10111111); //Para poner un 0
en el chip select del transceptor
    SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    while(digitalRead(12)==HIGH); //Debemos comprobar que SO esta en
LOW
    aux1=SPI.transfer(registro);   //Primero va un byte de header con
R/W, B, A5-A0
    valor=SPI.transfer(0xff);
    //Serial.println(registro,BIN);
    SPI.endTransaction();
    escribirRegistroExpansor(GPIO,aux|0b01000000); //Para poner un 1
en el chip select del transceptor
    wdt_reset();
    Serial.println(valor,BIN);
    return valor;
}
char comandStrobe(char registro)
{
  char aux,aux1;
    aux=leerRegistroExpansor(GPIO);
```

```
    escribirRegistroExpansor(GPIO,aux&0b10111111); //Para poner un 0
en el chip select del transceptor
    SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    while(digitalRead(12)==HIGH); //Debemos comprobar que SO esta en
LOW
    aux1=SPI.transfer(registro);   //Solo va un byte de header con
R/W, B, A5-A0
    Serial.println(aux1,BIN);
    SPI.endTransaction();
    escribirRegistroExpansor(GPIO,aux|0b01000000); //Para poner un 1
en el chip select del transceptor
    wdt_reset();
    //Serial.println(valor,BIN);
    return aux1;
}
void escribirRegistroPA()
{
  char aux,aux1;
    aux=leerRegistroExpansor(GPIO);
    escribirRegistroExpansor(GPIO,aux&0b10111111); //Para poner un 0
en el chip select del transceptor
    SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
    while(digitalRead(12)==HIGH); //Debemos comprobar que SO esta en
LOW
    SPI.transfer(0b00111110);   // byte de header con R/W, B, A5-A0
    SPI.transfer(0b00000000);   //Despues situamos el valor que
queremos darle al registro

    SPI.transfer(0b00111110);   // byte de header con R/W, B, A5-A0
    SPI.transfer(0b11000000);   //Despues situamos el valor que
queremos darle al registro (10dbm de emision), debemos escribir
primero el 0 para tener acceso l 1

    Serial.println("");
    SPI.endTransaction();
    escribirRegistroExpansor(GPIO,aux|0b01000000); //Para poner un 1
en el chip select del transceptor
    wdt_reset();
}
void inicializacionExpansor()
{

  Serial.println("Iniciando Expansor");
  escribirRegistroExpansor(IODIR,0b00010000);//Enviamos el 1 si el pin
es entrada 0 si es salida
                                             //GP6 CS para el
transceptor
                                             //GP5 PWM
                                             //GP4 open
                                             //GP3 COD_S3
                                             //GP2 COD_S2
                                             //GP1 COD_S1

  escribirRegistroExpansor(INTCON,0b00000000);//queremos tener una
interrupcion cuando se pulse el boton OPEN luego debemos establecer un
valor de comparacion
  escribirRegistroExpansor(GPINTEN,0b00010000);//habilitamos
interrupcion cuando se pulse el boton OPEN
  escribirRegistroExpansor(IOCON,0b00111010);//habilitamos
```

```
  escribirRegistroExpansor(GPIO, 0b01000001);//ponemos el chip select
a 1 para que no esté seleccionado y s0 a 1 para que el led este
apagado
  Serial.println("Expansor Iniciado");
}
void inicializacionTransceptor()
{
    escribirRegistroTransceptor(0b00000000,0b00011101);//Habilitamos
el pin GD2 como pin de salida TX
    escribirRegistroTransceptor(0b00000010,0b00001101);//Habilitamos
el pin GD0 como pin de salida TX
    escribirRegistroTransceptor(0b00000011,0b01000111);//Establecemos
que quepan 41bytes en TX y 24 en RX 01000111
    escribirRegistroTransceptor(0b00000100,0b10101010);//SYNC1
    escribirRegistroTransceptor(0b00000101,0b00000000);//SYNC0
    escribirRegistroTransceptor(0b00000110,0b00011001); //Escribimos
PKTLEN ¿? 25
    escribirRegistroTransceptor(0b00000111,0b00000000); //EScribimos
PKTCTRL1
    escribirRegistroTransceptor(0b00001000,0b00000000); //EScribimos
PKTCTRL0
    escribirRegistroTransceptor(0b00001011,0b00000110); //EScribimos
FSCTRL1 había un seis
    escribirRegistroTransceptor(0b00001101,0b00010000); //EScribimos
FREC2
    escribirRegistroTransceptor(0b00001110,0b10110111); //EScribimos
FREC1
    escribirRegistroTransceptor(0b00001111,0b11111010); //Escribimos
FREC0
    escribirRegistroTransceptor(0b00010000,0b11000110); //Escribimos
MDMCFG4 (habia f6)
    escribirRegistroTransceptor(0b00010001,0b10101010); //Escribimos
MDMCFG3
    escribirRegistroTransceptor(0b00010010,0b00110110); //Escribimos
MDMCFG2 Establecemos modulacion ASK/OOK y sin preambulo
    escribirRegistroTransceptor(0b00010011,0b00000000); //Escribimos
MDMCFG1
    escribirRegistroTransceptor(0b00010100,0b00000000); //Escribimos
MDMCFG0
    escribirRegistroTransceptor(0b00010101,0b00010101); //Escribimos
DEVIATN
    escribirRegistroTransceptor(0b00010110,0b00000111); //Escribimos
MCSM2¿?¿?¿?¿
    escribirRegistroTransceptor(0b00011000,0b00011000); //Escribimos
MCSM0
    escribirRegistroTransceptor(0b00011001,0b00010100); //Escribimos
FOCCFG
    escribirRegistroTransceptor(0b00100000,0b11111011); //Escribimos
WORCTL
    escribirRegistroTransceptor(0b00100010,0b00010001); //Escribimos
FREND0
    escribirRegistroTransceptor(0b00100011,0b11101001); //Escribimos
FSCAL3
    escribirRegistroTransceptor(0b00100100,0b00101010); //Escribimos
FSCAL2
    escribirRegistroTransceptor(0b00100101,0b00000000); //Escribimos
FSCAL1
    escribirRegistroTransceptor(0b00100110,0b00011111); //Escribimos
FSCAL0
    escribirRegistroTransceptor(0b00101100,0b10000001); //Escribimos
TEST2
```

```
    escribirRegistroTransceptor(0b00101101,0b00110101); //Escribimos
TEST1
    escribirRegistroTransceptor(0b00101110,0b00001001); //Escribimos
TEST0

    escribirRegistroTransceptor(0b00011011,0b00000111);//AGCCTRL2 7
    escribirRegistroTransceptor(0b00011100,0b00110000);//AGCCTRL1 0
    escribirRegistroTransceptor(0b00011101,0b10010011);//AGCCTRL0 92
    Serial.println("Transceptor Iniciado ");
}


void inicializacionVariables()
{
    datosFifoRX[0]=0xAA; //Preambulo H
    datosFifoRX[1]=0xAA; //Preambulo L
    datosFifoRX[2]=0xAA; // SYNC H
    datosFifoRX[3]=0x00; // SYNC L
}

/*------Funciones Principales Decdicadas al tratamiento de datos  ----
-------------*/


void escribirByteDecoder(char dato)
{
  char i;
  for(i=0;i<7;i++)
  {
    digitalWrite(DEC_CLK, HIGH);
    digitalWrite(DEC_DATA,(dato>>i)&0b1); //Aqui pongo el dato con el
LSB primero
    delayMicroseconds(500);
    digitalWrite(DEC_CLK, LOW);
    delayMicroseconds(500);
  }
  digitalWrite(DEC_CLK, HIGH);
  digitalWrite(DEC_DATA,(dato>>7)&0b1); //Aqui pongo el MSB
  delayMicroseconds(500);
  digitalWrite(DEC_CLK, LOW);
  delayMicroseconds(500);
  wdt_reset();
}


//Pasa el byte de datos de la fifo apuntado por indice a la entrada
pwm de datos del decodificador
void pasarByteDecodificador(char indice)
{
  char aux,i;
  aux=leerRegistroExpansor(GPIO);
  if(indice==28) //El ultimo bit del ultimo byte no es válido
  {
    for(i=0;i<7;i++)
     {
        if((datosFifoRX[indice]>>(7-i))&0b1)
        {
          //El bit es un 1
           aux=aux|0b10000000;
           escribirRegistroExpansor(GPIO,aux);
```

```
            delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
          else
          {
            //El bit es un 0
            aux=aux&0b01111111;
            escribirRegistroExpansor(GPIO,aux);
            delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
      }
    }
    else
    {
      for(i=0;i<8;i++)
       {
          if((datosFifoRX[indice]>>(7-i))&0b1)
          {
            //El bit es un 1
             aux=aux|0b10000000;
             escribirRegistroExpansor(GPIO,aux);
             delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
          else
          {
            //El bit es un 0
            aux=aux&0b01111111;
            escribirRegistroExpansor(GPIO,aux);
            delayMicroseconds(350); //Tarda en cambiarse 25us y el min
periodo es de 65
          }
      }
    }
    wdt_reset();

}

char realizarRecepcionRX()
{
  char i;
  Serial.println("Activando recepcion");
  delay(500);
  comandStrobe(0b00110110); //Le meto en modo idle COMANDO 0x36
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x01) //Espero a
que el estado este bien
     {
       wdt_reset();
     }
  comandStrobe(0b00111010); //Borro a FIFO de recepcion 0x3A
  delay(2);
  comandStrobe(0b00110100); //Le meto en modo recepcion COMANDO 0x34
  while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x0D) //Espero a
que el estado este bien
     {
       wdt_reset();
     }
  delayMicroseconds(100);
  Serial.println("Modo recepcion activo");
```

```
  //Leo el numero de Bytes disponible y mientras sea menor que 25
espero
  while((leerRegistroTransceptor(0b11111011))<25)
   {
     delay(15);
     wdt_reset();
   }
   if(leerRegistroTransceptor(0b11111011)&0x80)//Significa que esta
activo el bit de overflow
   {
      comandStrobe(0b00110110); //Le meto en modo idle COMANDO 0x36
      while((leerRegistroTransceptor(0b11110101)&0x1F)!=0x01) //Espero
a que el estado este bien
      {
        wdt_reset();
      }
      comandStrobe(0b00111010); //Le mando el SFRX para restaurar el
buffer 0x3A
      Serial.println("Recepcion fallida");
      return 1;
   }
   else //Tenemos ya los veintinueve bytes listos
   {
      for(i=4;i<29;i++)
      {
        datosFifoRX[i]=leerRegistroTransceptor(0b10111111);
      }
      return 0;
   }
}

void ttoPuerta()
{
  unsigned char i,aux;
  int secndStatusByte=0;
  long long int hoppingCode=0;
   if(realizarRecepcionRX()==0)
   {
      //recepcion realizada con exito hay que pasar los datos al
decodificador
      for(i=0;i<29;i++) //Se transmiten 28Bytes
      {
        pasarByteDecodificador(i);
      }
      //Compruebo si el código es correcto
      delay(10);
      pinMode(DEC_DATA,INPUT);
      if(digitalRead(DEC_DATA)==HIGH) //Ha sido correcto
      {
        delay(60); //espero tcla
        digitalWrite(DEC_CLK,HIGH);
        while(digitalRead(DEC_DATA)==HIGH)
        {
          wdt_reset();
        }
        delayMicroseconds(500); //espero tclkh
        digitalWrite(DEC_CLK,LOW);
        delayMicroseconds(200); //espero tpp3
        for(i=0;i<16;i++)
        {
          delayMicroseconds(500);
```

```
          digitalWrite(DEC_CLK, HIGH);
          delayMicroseconds(250);
          secndStatusByte=secndStatusByte| digitalRead(DEC_DATA)<<(i);
          delayMicroseconds(250);
          digitalWrite(DEC_CLK, LOW);
        }
        wdt_reset();
        for(i=0;i<64;i++)
        {
          delayMicroseconds(500);
          digitalWrite(DEC_CLK, HIGH);
          delayMicroseconds(250);
          hoppingCode=hoppingCode|(digitalRead(DEC_DATA)<<(i));
          delayMicroseconds(250);
          digitalWrite(DEC_CLK, LOW);
          wdt_reset();
        }

        //Como ha sido correcto enciendo un LED para mostrarlo empleo
la linea de cod_S0
        aux=leerRegistroExpansor(GPIO);
        escribirRegistroExpansor(GPIO, aux&0b11111110);
        Serial.println("Enciendo Led");
        delay(500); //Lo dejo encendido 500ms
        escribirRegistroExpansor(GPIO, aux|0b00000001); //Y lo apago
      }
   }

}

void setup() {
  char valorPuerto,aux,i;
  Serial.begin(9600);  //Abrimos la comunicacion con el puerto serie
  delay(10);
  declaracionPines();            //Declaramos los pines del ESP
  SPI.pins(14,12,13,5);          //Para que el ESP emplee HSPI se
señaliza el GPIO12,13,14
  SPI.begin();                   //Inicializamos la comunicacion SPI a
10Mhz
  wdt_reset();                   //Se debe resetear el perro guardián
  inicializacionExpansor();      //Inicializamos el Expansor
  wdt_reset();
  inicializacionVariables();     //Establecemos los valores iniciales
de las variables principales
  wdt_reset();
  inicializacionTransceptor();   //Inicializamos el transceptor
}

void loop() {
  wdt_reset();
  ttoPuerta();              //Realizamos la gestión de recepción de
mensajes
}
```