Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Propuesta de Actualización del Diseño Hardware y Software de un Juego de Espigas

Autor: Jaime Villarejo Crespo
Tutor/es: Bernardo Alarcos Alcázar

2022

Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo de Fin de Grado

Propuesta de Actualización del Diseño
Hardware y Software de Juego de Espigas

Autor: Jaime Villarejo Crespo

Tutor: Bernardo Alarcos Alcázar

TRIBUNAL:

Presidente: Juan Ramón Velasco Pérez

Vocal 1º: Antonio García Herraiz

Vocal 2º: Bernardo Alarcos Alcázar

FECHA: 2022

Índice

Figuras

Tablas

Anexos

Resumen

Hoy en día hay diversas formas de educar y entretener a los niños en colegios a la vez que se intentan detectar anomalías en su desarrollo. El juego de espigas es uno de estos modos de enseñanza donde se intenta medir las capacidades y aptitudes de los más pequeños, mediante una actividad interactiva, para poder detectar a tiempo estas irregularidades en sus capacidades motoras e intelectuales.

A lo largo de este trabajo se realizarán una serie de modificaciones en un juego de espigas el cual, a través de unas espigas de madera y unos sensores, recopilará información del niño que esté utilizándolo en ese momento, como puede ser su capacidad de detectar el lugar correcto en el que debe introducir la espiga o su velocidad motora para hacer ese movimiento. Estos datos permiten detectar a tiempo posibles problemas de vital importancia.

Palabras clave: ESP-WROOM-32, ATmega328P, BLE, Arduino

Abstract

Nowadays there are several ways to educate and entertain children in schools while trying to detect anomalies in their development. The game of spikes is one of these ways of teaching where we try to measure the capacities and aptitudes of the smallest, by means of an interactive activity, to be able to detect in time these irregularities in their motor and intellectual capacities.

Throughout this work, a series of modifications will be made to a dowel game which, through wooden dowels and sensors, will collect information from the child who is using it at that moment, such as his ability to detect the correct place where the dowel should be introduced or his motor speed to make that movement. This data allows early detection of potential problems of vital importance.

Key words: ESP-WROOM-32, ATmega328P, BLE, Arduino

Capítulo I: Introducción y objetivos

Introducción y objetivos

El juego de espigas es un dispositivo electrónico que está formado por dos filas de 10 fotosensores y unas espigas de madera, en la Figura 1 puede verse el juego de espigas actual. El juego consiste en introducir la espiga en un agujero con fotosensor y después mover esa espiga lo más rápido posible al agujero correspondiente. Cada agujero consta de un led con dos colores:

- Verde: para indicar que la espiga está introducida correctamente en el agujero.
- Naranja: para indicar que el agujero está libre de espiga.

El dispositivo se encargará de tomar el tiempo en el que el usuario, en este tipo de dispositivo suelen ser niños, tarda en cambiar e introducir la espiga de un agujero a otro que este libre. Este tipo de dispositivos sirven para poder detectar anomalías y problemas en el desarrollo intelectual y motriz de una persona, e intentar corregirlas a tiempo.

La idea de realizar este proyecto surge de la necesidad de mejorar el juego de espigas para que su uso y mantenimiento sea más sencillo para el usuario y por tanto ser capaces de detectar a mayor velocidad algún tipo de problema en el desarrollo de los niños.

Figura 1. Juego de espigas con sensores

Objetivos del proyecto

El principal objetivo del proyecto será la actualización del hardware y del software del dispositivo para que su uso sea mucho más intuitivo y sencillo para el usuario. Este rediseño de ambos sistemas permitirá la eliminación de componentes y que, por tanto, el peso del dispositivo también sea menor. Además, habrá una serie de subobjetivos que a la vez que se realizan los cambios de hardware y software se abordarán. Estas actualizaciones se dividen en diversas partes:

- Actualización del Hardware: En cuanto al hardware, el principal cambio que se va a llevar a cabo es la sustitución del chip principal ATmega328p por un chip más moderno y actualizado, el ESP-WROOM-32. Esta sustitución permitirá cambiar la forma de enviar y recibir los datos tomados del usuario además de, como ya se había mencionado anteriormente, eliminar componentes y hacer más liviano el dispositivo. Este cambio de chip conllevará el diseño de una nueva placa principal de circuito impreso (PCB).
- Actualización del software: En cuanto al software, la principal actualización es la creación de un nuevo software basado en la transmisión de datos empleando tecnología *Bluetooth Low Energy* (BLE), manteniendo el mismo entorno de programación (Arduino), el lugar de la transmisión por radio frecuencia utilizada actualmente.
- Subobjetivos: Al satisfacer estos rediseños de hardware y software será necesario llevar a cabo otras modificaciones necesarias:
    - Revisar conectores de los sensores.
    - Cambiar el circuito de voltaje de 5V a 3.3V en los componentes que sea posible.
    - Revisar las resistencias de los optoacopladores y probar otro tipo de configuraciones.
    - Realizar una conexión eterna de carga con USB-C y un sensor de estado de batería.
    - Diseño 3D de las tapas del juego de espigas para incluir el sensor de estado de batería y la conexión de carga.

A lo largo de la exposición de trabajo se desarrollarán paso por paso la metodología empleada para poder llevar a cabo la actualización del dispositivo.

En cuanto a la estructura que se va a seguir para el desarrollo de la memoria de la propuesta de actualización es la siguiente:

- En el capítulo II se desarrollará la metodología empleada para el desarrollo del proyecto.
- En el capítulo III se explicarán los pasos seguidos para llevar a cabo la actualización del hardware.
- En el capítulo IV se expondrán los distintos cambios que se van a llevar a cabo en el software del juego de espigas.
- En el capítulo V se actualizará el diseño del dispositivo sustituyendo algunos conectores e incorporando nuevos elementos a las tapas del mismo.
- En el capítulo VI se explicarán las conclusiones obtenidas de la propuesta de actualización y las líneas futuras que se abren para poder seguir desarrollando el dispositivo.
- Po último, en el capítulo VII y VIII se dedicarán a la bibliografía y a los anexos respectivamente.

Capítulo II: Metodología

Metodología

A continuación, se va a exponer cronológicamente la metodología empleada para la realización del proyecto:

1. Estudio del juego de espigas actual y los elementos que lo componen para poder cumplir los objetivos propuestos.
2. Estudio del chip ATmega328p para comprenderlo y poder sustituirlo.
3. Estudio del chip ESP-WROOM-32 para comprenderlo y poder acoplarlo como chip principal.
4. Cambio de ATmega328p por ESP-WROOM-32 y eliminación de elementos sobrantes.
5. Actualización de componentes a 5V y 3.3V.
6. Elección del sistema de carga USB-C y sensor de estado de batería.
7. Montaje del sensor y del puerto de carga en el juego de espigas.
8. Diseño de nuevo PCB principal.
9. Estudio de la tecnología BLE (*Bluetooth Low Energy*) con motivo de poder utilizarla como nuevo método para transmitir información.
10. Realización del nuevo software de comunicaciones con Arduino.
11. Revisión de conectores y resistencias del sistema.
12. Nuevo diseño 3D de las tapas del dispositivo para incluir elementos externos.

En este orden cronológico es en el que se van a desarrollar las diferentes etapas del trabajo realizado, así como las posibles complicaciones y conclusiones obtenidas del mismo.

Capítulo III: Actualización del hardware:

1. Estudio de los elementos actuales del juego de espigas.
2. ATmega328p.
3. ESP-WROOM-32.
4. Cambio de microcontrolador principal.
5. Actualización de voltaje a 5V y 3.3V.
6. Selección y montaje de USB-C y sensor de estado de batería.
7. Diseño de PCB principal

1. Estudio de los elementos actuales del juego de espigas.

En primer lugar, hay que conocer los elementos que actualmente forman parte del juego de espigas y cuál es la función que cumple cada uno de ellos en el dispositivo. Primero se elaborará una pequeña lista con los elementos principales de este para después explicar cuál es la función de cada uno, excepto del chip principal ATmega328P, del cual se hablará más adelante. Así, los elementos principales actuales son:

- ATmega328P: Microcontrolador principal del juego de espigas.
- NRF24L01: Módulo transceptor inalámbrico de bajo consumo que funciona a 2,4GHz.
- 4021N: Registro de desplazamiento de 8 bits.
- LM1085-3.3: Regulador de voltaje a 3.3V
- F/CM12P: Bocina para que el juego de espigas pueda emitir sonidos.
- Fotosensores: Sensor que detecta cuando se introduce una espiga.

A parte de estos elementos, el juego también consta de resistencias, condensadores, un potenciómetro, leds para la iluminación del juego, un Switch para poder encender y apagar el juego y conectores para poder ensamblar todas las piezas del juego entre sí. A continuación, se expone que es cada uno de los elementos anteriores y el uso principal que se le da en el dispositivo de espigas.

NRF24L01

El módulo NRF24L01 es un módulo transceptor que sirve para el envío de datos entre dos dispositivos. Existen varios modelos en la actualidad operando todos ellos en la banda de 2.4GHz, son muy usados debido a sus funcionalidades, su bajo costo y su bajo consumo. Además, hay varios modelos de ellos, diferenciándose principalmente por si llevan antena o no. Aquellos que lleven antena permiten una transmisión de datos de hasta 1km, una distancia mucho mayor que aquellos módulos sin antena. En el juego de espigas este dispositivo se usa para poder transmitir los datos generados por el usuario a través de un software configurado con Arduino. Es la antena principal del juego para la emisión y el recibimiento de datos. El chip se muestra en la Figura 2. [1]

Figura 2. Módulo NRF24L01 con antena

4021N

El dispositivo 4021N es un registro de desplazamiento de 8 bits. Estos registros de desplazamiento son circuitos digitales secuenciales, que suelen ser de tipo D, conectados en cascada, que, según como estén conectados, tienen un desplazamiento hacia derecha o izquierda de la información almacenada. El registro de desplazamiento puede ser MASTER o SLAVE. En el dispositivo de espigas se tiene una configuración de 4 módulos, dos MASTER y dos SLAVE. El principal uso que se le da a estos cuatro módulos es el de disponer de más inputs digitales de los que hay en el programa de Arduino. Así con solo utilizar 2 pines en el programa de Arduino el juego es capaz de poder utilizar y reconocer los 20 inputs de espigas totales que tiene el dispositivo completo. En la Figura 3 se muestra el módulo 4021N. [2]

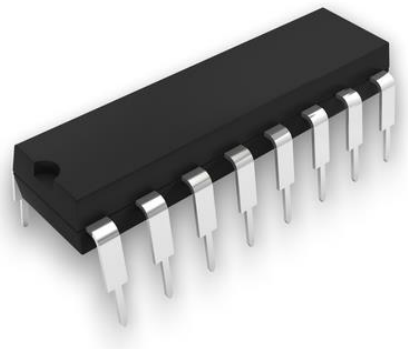


Figura 3. Módulo 4021N

LM1085-3.3

Es un dispositivo regulador de voltaje que permite disminuir el voltaje del circuito para no dañar los componentes de este, ya varios componentes de un mismo circuito pueden funcionar a un distinto nivel de voltaje. Está disponible en versiones de 3.3V, 5V y 12V. Tiene un dropout máximo de 1,5V a 3A de corriente. En el juego de espigas es utilizado para regular el voltaje de la antena del NRF24L01 de 5V a 3.3V y no dañar la misma. En la Figura 4 se puede observar el LM1085 de 3.3V. [3]

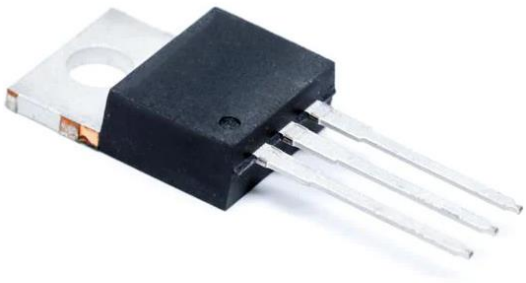


Figura 4. Módulo LM1085-3.3V

F/CM12P

Dispositivo de alarma o sonido que sirve para generar tonos de aviso ante un estímulo elegido por el diseñador del circuito. En el dispositivo de espigas su uso es el de generar un sonido cada vez que el usuario consiga un objetivo, como puede ser introducir la espiga en el agujero correcto. Este sonido se configurará a través del software y se explicará más adelante como su configuración y el sonido elegido. En la Figura 5 se muestra la bocina empleada.

Figura 5. Módulo F/CM12P (Buzzer)

FOTOSENSORES

Un fotosensor es un dispositivo sensible a la luz que convierte la energía de fotones en una señal eléctrica que es capaz de ser transmitida y generar una respuesta en un circuito eléctrico. Se utilizan para recibir datos de entrada a través de detectar un cambio o la intensidad de la energía electromagnética. En el dispositivo de espigas se utiliza para detectar cuando se introduce o cuando se extrae una espiga de un agujero y generar un cambio en el led de este, pasando de estado vacío (naranja) ha estado ocupado (verde) y viceversa, y poder tomar el tiempo que tarda en llevar la espiga de un agujero a otro detectando la salida y la entrada de esta. El fotosensor empelado en este juego es el LTH-301-32, mostrado en la Figura 6. [4]

Figura 6. Módulo Fotosensor el LTH-301-32.

Una vez conocidos los componentes principales y el uso que se le da en el juego de espigas se explicarán ambos microcontroladores y el motivo principal que ha llevado a la sustitución del chip actual (ATmega328P) por uno nuevo (ESP-WROOM-32).

2. ATmega328p.

En la versión anterior del juego de espigas este era el chip principal encargado de recibir la información y procesarla. Es un microcontrolador simple, pero efectivo para la tarea que se quiere llevar a cabo con el juego, que además permite en uso del IDE Arduino. Está basado en un microcontrolador RISC (*Reduced Instruction Set Computer*) y tiene 32Kb de memoria FLASH (capaz de leer y escribir al mismo tiempo), una memoria EEPROM de 1Kb, memoria SRAM de 2Kb, además de entradas digitales y analógicas, distintos tipos de salidas y varios periféricos de comunicación. [5] [6].

El Atmega328p está construido, como se puede observar en la Figura 7, de la siguiente manera [7]:

- Tiene 28 puertos de entrada, con distintas funciones, de los cuales el 7 y el 20 son exclusivos de VCC y el 8 y el 22 de GND o tierra.
- VCC y GND son los pines propios de alimentación y tierra. Este microcontrolador puede alimentarse a 5V o a 3.3V, dependiendo la configuración que se desee utilizar.
- El RESET está ubicado en el puerto 1 y cumple la función de poder reiniciar todo el proceso que se está llevando a cabo dentro del microcontrolador y ponerlo a 0.
- El resto de los puertos del microcontrolador están divididos en subgrupos (PuertoB, PuertoC, PuertoD). En algunos software de programación es necesario conocer al pie de la letra estos puertos ya que es donde se localiza la dirección física de los pines externos, no siendo necesario en el IDE Arduino utilizado actualmente ya que estos se denominan con números específicos.
- A parte de que todos los pines son pines digitales de entrada algunos de ellos tienen funciones distintas de las que cabe destacar:
    1. INT0, INT1: Son interrupciones como tal, cuya principal función es interrumpir el proceso que se está llevando a cabo ante una entrada digital externa programada. El microcontrolador deja de ejecutar el proceso y da prioridad a la interrupción. Estas interrupciones están colocadas en los puertos 4 y 5.

2. XTAL1, XTAL2: Estás funciones sirven para colocar osciladores crystals para poder definir el ciclo de trabajo, ubicadas en los puertos 9 y 10.

- El Atmega328p consta de entradas analógicas, con una resolución de 10 bits, situadas en los puertos del 23 al 28.
- Cuenta con 6 pines de salida PWM ubicados en los puertos 5, 21, 12, 15, 16 y 17, con una resolución de 8 bits. La función de estos es generar salidas analógicas a través de pines digitales.

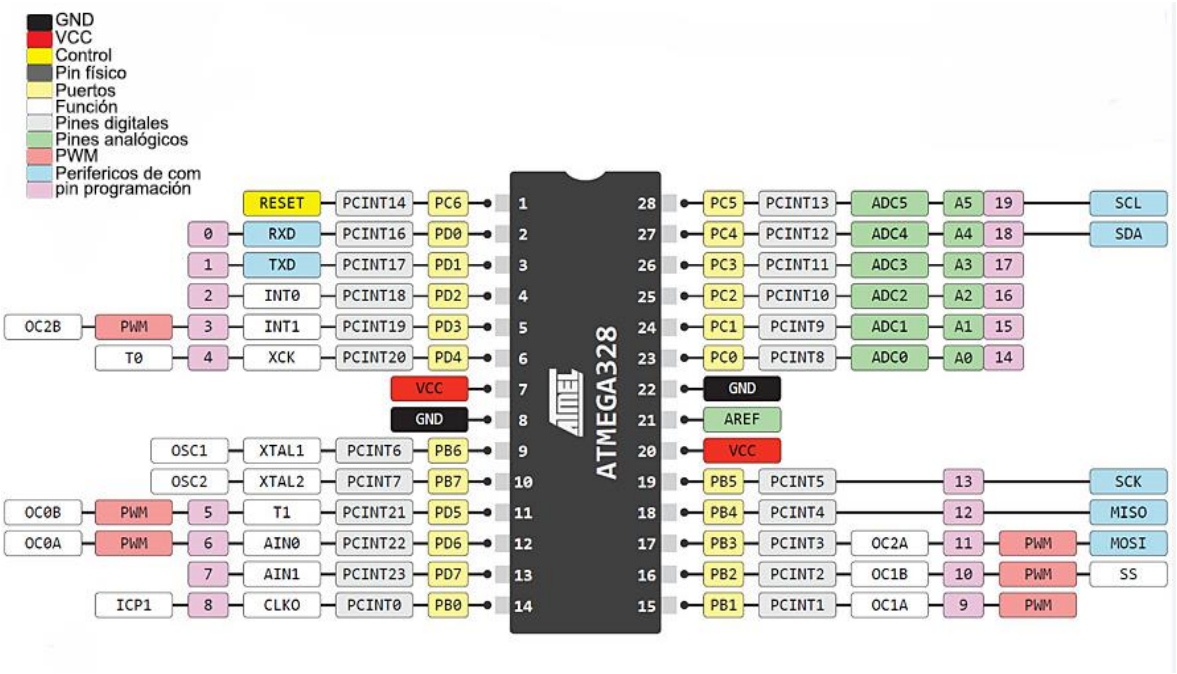


Figura 7. ATmega328p.

Hay diversos motivos por los cuales se escogió este microcontrolador por encima de otros desde un inicio, y estos tienen que ver con el IDE Arduino escogido para su programación [8]:

- Barato y sencillo: El IDE Arduino que se está utilizando hace que trabajar con microcontroladores compatibles sea mucho más sencillo e intuitivo para el usuario. Además, el ATmega328p es, dentro de los chips de su misma categoría, de los más baratos que se pueden encontrar en el mercado.

- Multiplataforma: Arduino es compatible con casi cualquier sistema operativo actual como Windows, MacOS o Linux, mientras que muchos otros IDE de programación solo son compatible con Windows limitando así la experiencia del usuario.

- Simplicidad: Arduino simplifica el proceso de trabajar con este tipo de microcontroladores, siendo un software de programación fácil e inmediato. Es flexible para todo tipo de usuarios independiente del nivel que tenga el mismo. Está basado en Procesing por lo que cualquiera que aprenda a desarrollar en ese método de programación se sentirá familiarizado con Arduino.

- Software libre: El código de programación de Arduino es de licencia libre y puede ser utilizado también por programadores expertos. El lenguaje se basa en C++ y puede ser ampliado en cualquier momento a través de librerías del mismo. En caso de querer profundizar se puede dar el salto al lenguaje AVR C, pudiendo también ampliar este a través de librerías y añadiendo código directamente en el programa del usuario.

- Hardware: El hardware disponible para Arduino es ampliable y se dispone de muchas opciones cuyos módulos están publicados bajo la licencia de Creative Commons. Cualquier diseñador que tenga experiencia puede hacer su propio diseño y su propia versión del módulo, haciéndolo óptimo para la aplicación que desee. Estos pueden incluso construir sus propias versiones de la placa abaratando los costes de la misma y probando distintos tipos de configuraciones para la optimización de esta.

Una vez explicado y conocido el microcontrolador anterior y el porqué de su elección se procede a explicar el nuevo módulo principal (ESP-WROOM-32).

3. ESP-WROOM-32.

Es necesario conocer este tipo de microcontrolador y como está construida su estructura para poder así hablar de los motivos que han llevado al cambio de microcontrolador.

El ESP-WROOM-32 es un módulo muy potente que integra tanto wifi como bluetooth para la transmisión de datos, ideal para llevar a cabo y desarrollar proyectos de IOT (*Internet Of Things*), aunque se puede aplicar en diversas áreas por su elevado rendimiento. El wifi de este microcontrolador permite una conexión estable que sea de mediano alcance y conectarse a una red LAN para, a través de un modem router, tener conexión a internet, mientras que el bluetooth que incorpora permite conectar dos dispositivos directamente entre sí e intercambiar datos e información entre ellos. El microcontrolador incorpora un procesador de 2 núcleos donde cada uno de ellos puede trabajar en frecuencias entre 80 y 240 MHz individualmente. El ESP-WROOM-32 cuenta con 48 pines con múltiples funciones, aunque algunos de ellos no pueden ser utilizados para ciertas tareas [9]. La Figura 8 hace referencia al ESP-WROOM-32 y muestra como está construido. Este microcontrolador incluye los siguientes periféricos [10]:

- 18 canales ADC (convertidor analógico digital).
- 3 interfaces SPI.
- 3 interfaces UART.
- 2 interfaces I2C.
- 16 PWM de salida.
- 2 canales DAC (convertidor analógico digital).
- 2 interfaces I2S.
- 10 GPIOs con sensores capacitivos tactiles.

Figura 8. ESP-WROOM-32

A continuación, se expone el principal funcionamiento de cada uno de estos periféricos y algunas características propias del chip [11] [12]:

- Las funciones del ADC y del DAC del dispositivo están asignadas a pines específicos del mismo. Sin embargo, el usuario puede decidir que pines utilizar como UART, I2C, SPI, PWM, etc. Todo esto es posible debido a la función multiplexor del ESP-WROOM-32.
- Además, estas funciones pueden ser definidas en el propio software de control del dispositivo. Los pines se encuentran predefinidos como se muestra en la Figura 8.
- Existen pines con características específicas que los hacen adecuados o no para diversos proyectos. En la tabla 1 se indica que pines pueden ser utilizados como Inputs, Outputs o si necesitas ser cauteloso a la hora de programarlos.

GPIO	INPUT	OUTPUT	*GPIO*	INPUT	OUTPUT
0	Pulled Up	OK	**20**	X	X
1	TX pin	OK	**21**	OK	OK
2	OK	OK	**22**	OK	OK
3	OK	RX pin	**23**	OK	OK
4	OK	OK	**24**	X	X
5	OK	OK	**25**	OK	OK
6	X	X	**26**	OK	OK
7	X	X	**27**	OK	OK
8	X	X	**28**	X	X
9	X	X	**29**	X	X
10	X	X	**30**	X	X
11	X	X	**31**	X	X
12	OK	OK	**32**	OK	OK
13	OK	OK	**33**	OK	OK
14	OK	OK	**34**	OK	X
15	OK	OK	**35**	OK	X
16	OK	OK	**36**	OK	X
17	OK	OK	**37**	X	X
18	OK	OK	**38**	X	X
19	OK	OK	**39**	OK	X

Tabla 1. Modos de uso de los pines del ESP-WROOM-32

- GPIOs 34, 35, 36 y 39 son solo inputs. Estos pines no tienen resistencias pull-up o pull-down internas, por lo que solo pueden ser inputs.
- Los GPIOs 6, 7, 8, 9, 10 y 11 están conectados directamente a la flash SPI integrada en el chip ESP-WROOM-32 y no se recomiendan para otros usos. No se recomienda su uso en proyectos.

- El ESP-WROOM-32, como hemos comentado anteriormente, tiene 10 sensores capacitivos en su interior. Estos sensores son capaces de detectar variaciones en cualquier cosa que tenga carga eléctrica, como puede ser la piel humana. Así, son capaces de detectar las variaciones que produce el tocar el sensor capacitivo con un dedo. Estos GPIOs pueden integrarse fácilmente en las almohadillas capacitivas y sustituir a los botones mecánicos. También pueden ser utilizados para despertar al ESP-WROOM-32 de un *Deep-Sleep* (modo de espera). Estos pines capacitivos están localizados en los GPIOs 0, 2, 4, 12, 13, 14, 15, 27, 32 y 33.

- En cuanto a los 18 canales de ADC tienen una resolución de 12 bits cada uno. Esto quiere decir que puedes obtener lecturas analógicas de un rango de 0 a 4095, donde 0 corresponde a 0V y 4095 a 3.3V. El ESP-WROOM-32 permite ajustar la resolución de los canales a través de código, así como el rango del ADC. Estos pines no tienen un comportamiento lineal. El comportamiento de los ADC del microcontrolador se muestra en la Figura 9.

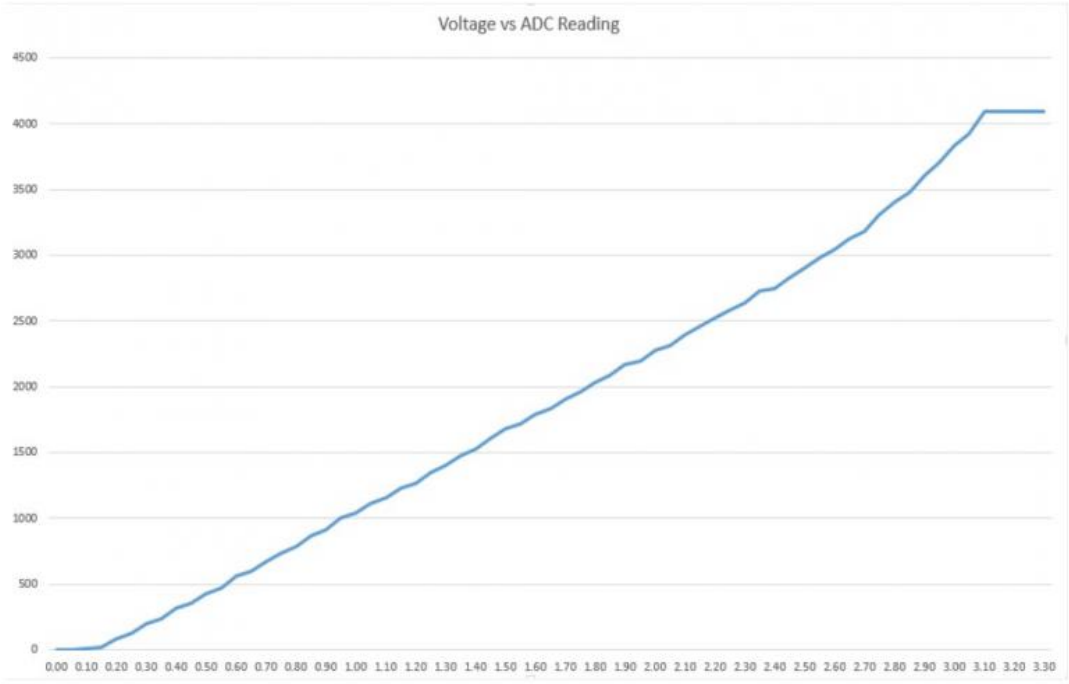


Figura 9. Voltaje VS lectura de los canales ADC

- Los canales DAC del microcontrolador tienen una resolución de 8 bits cada uno. Sirven para convertir la señal digital recibida en un voltaje de salida. Estos pines se corresponden con los GPIOs 25 y 26.
- El ESP-WROOM-32 tiene GPIO RTC (*Real Time Clock*). Los GPIOs enrutados al subsistema de bajo consumo pueden utilizarse cuando el ESP-32 se encuentra en *Deep-Sleep*. Estos se pueden utilizar para despertar al microcontrolador del *Deep-Sleep* cuando el coprocesador de ultra bajo consumo (ULP) se está ejecutando.
- Tiene 16 canales independientes de PWM que pueden ser configurados para generar señales PWM de distintas características. Todos los pines que pueden utilizarse como output se pueden utilizar como PWM. (Excepto el pin 34, 35, 36 y 39)
- Los canales I2C del dispositivo y cualquier otro pin pueden ser configurados como SDA o SCL.
- Todos los pines pueden ser configurados para generar interrupciones.

Por todos estos motivos expuestos y por la diversidad de funciones y características que posee el ESP-WROOM-32, así como su facilidad para transmitir datos tanto por WIFI como por Bluetooth sin necesidad de terceros elementos, llevaron a este microcontrolador a ser el elegido para sustituir al ATmega328p.

Una vez conocidos en profundidad ambos chips, se procede a explicar tanto el proceso de cambio de un microcontrolador por otro, como los motivos principales en profundidad que han llevado a ese cambio.

4. Cambio de microcontrolador principal.

Es necesario conocer los principales motivos que llevaron al cambio de un microcontrolador por otro. Así, estos son:

1. Transferencia de datos. Mientras que con el ATmega328p la transferencia de datos se realizaba mediante la antena NRF24L01 a través de por medio de tecnología de radio frecuencia (RF), con el ESP32 es posible enviar datos tanto por wifi como por bluetooth. En nuestro caso para el envió de datos nos centraremos en el envió por bluetooth, más específicamente en el BLE (*Bluetooth Low Energy)* una tecnología inalámbrica mucho más estable que la radio frecuencia y con unos consumos inferiores para alargar la vida del dispositivo.

2. Mayor número de periféricos y memoria flash. Con la versión anterior del juego de espigas la memoria del microcontrolador quedaba llena cuando se cargaba el software para hacerlo funcionar. Al cambiar el microcontrolador por uno más moderno y de mayor capacidad, además de eliminar ciertos elementos que ya no son necesarios en el dispositivo, este problema de memoria queda resuelto ya que el ESP32 posee una capacidad de memoria flash mayor y más avanzada tecnológicamente, por lo que la respuesta y el envío de datos del dispositivo será más rápida. Además, el mayor número de periféricos permitirá aumentar funciones del juego de espigas en un futuro si se desea, ya que este mayor número permite conectar nuevos elementos, siendo antes más difícil su implementación, para poder llevar a cabo estas mejoras futuras. En la Tabla 2 se hace una comparación general entre ambos microcontroladores, donde se puede observar que el ESP32 es mejor microcontrolador en todos los aspectos comparables de ambos procesadores [13].

	ESP-WROOM-32	ATmega328P
Tensión de trabajo	3.3V	5V
Consumo de corriente	80mA-90mA	45mA-80mA
Pines digitales	19	14
Pines analógicos	18	8
UARTs	3	1
SPI	4	1
I2C	2	1
Wifi	Sí	X
Bluetooth	4.2 y BLE	X
Frecuencia CPU	80-240MHz	0-20MHz
Memoria Flash	520Kb	32Kb
Memoria SRAM	448Kb	2Kb
Dimensiones	18x25.5x3.1 mm	7x7x1.05 mm

Tabla 2. ATmega328p VS ESP-WROOM-32

3. Software. Con el microcontrolador anterior para poder realizar la carga de software era necesario desmontar parte de las tapas del dispositivo para poder tener acceso al puerto de carga y así actualizar el software. Con el ESP32 esta carga de software se puede hacer a través de conexión inalámbrica u OTA *(Over The Air)* una característica propia del BLE que hace que el uso y programación de los microcontroladores que lo tienen sea mucho más sencillo. Más adelante, en la parte de software se explicará la configuración del software necesario para poder llevar a cabo la transferencia de datos a través del BLE, si finalmente se escoge esta manera de cargar el nuevo software.

A continuación, y una vez conocidos los motivos que llevan al cambio de procesador, se explicará paso por paso y de manera detallada los pasos que se han llevado a cabo para sustituir uno por el otro.

En primer lugar, era necesario conocer el esquema actual del PCB principal para ver como estaban conectados los diferentes elementos entre sí. En la Figura 10 se puede observar este primer esquema, esquema a partir del cual se va a trabajar para intentar alterar lo menos posible el juego de espigas y que su uso sea lo más parecido posible. Es necesario descargarse las librerías de Eagle para poder observar el esquemático correctamente [14].

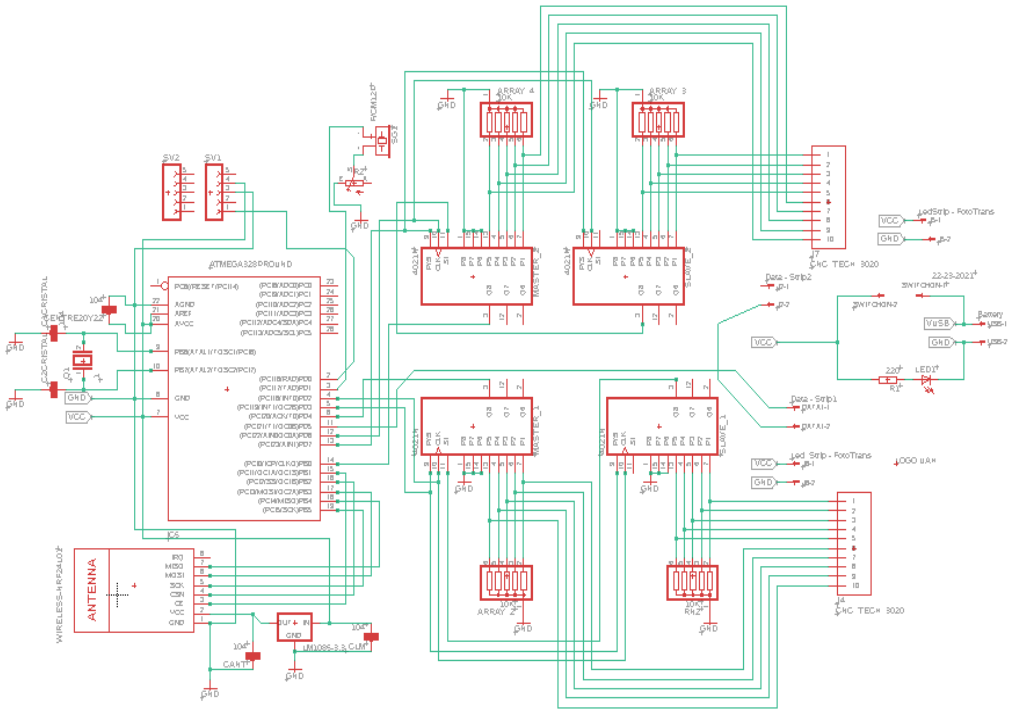


Figura 10. Esquemático general con el ATmega328P

Como se puede observar en la Figura 10 el ATmega328P está conectado directamente a los 4021N, a las tiras de led a través de *Data Strips*, al buzzer o bocina para poder emitir los sonidos, a la antena NRF24L01 y a un oscilador Crystal para poder definir el ciclo de trabajo.

Una vez conocido y estudiado el esquema inicial, se procede a realizar cambios para poder acoplar el nuevo microcontrolador. Primero de todo vamos a estudiar de cerca el Atmega328P, mostrado en la Figura 11, para ver a que pines del chip están conectados cada uno de los elementos.

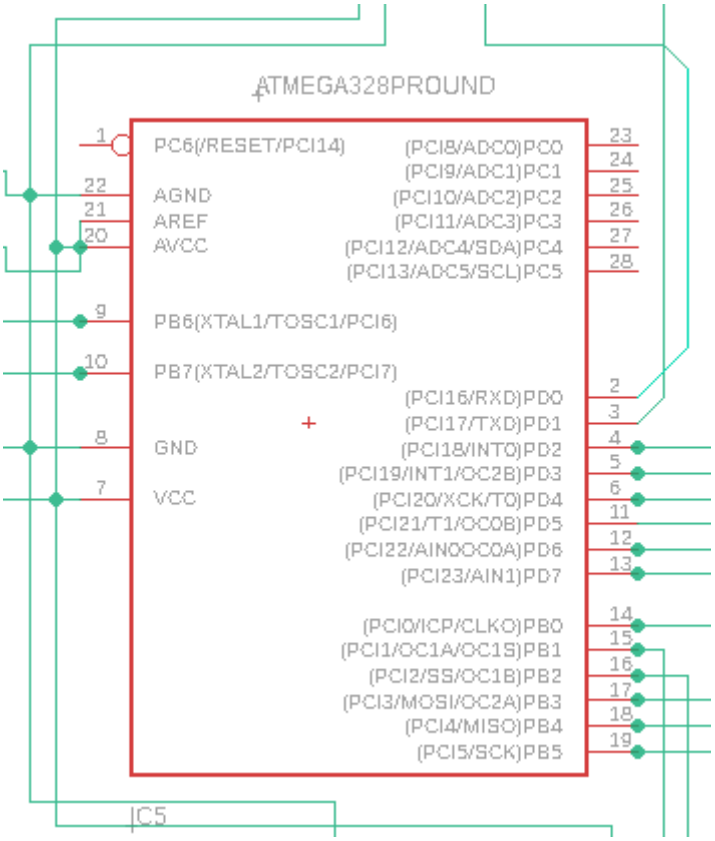


Figura 11. Esquema del ATmega328P

Se pueden observar claramente los pines que tiene conectados, y con ayuda del esquemático de la Figura 10 podemos ver a que elemento corresponde cada pin. Así:

- El pin 2 (PD0) está conectado al NRF24L01 a través de un cabezal hembra SV1 situado en la parte superior izquierda del esquema general.
- El pin 3 (PD1) está conectado al buzzer o bocina.
- El pin 4 (PD2) está conectado al 4021N master1.
- El pin 5 (PD3) está conectado al 4021N master1.
- El pin 6 (PD4) está conectado al 4021N master1.
- El pin 11 (PD5) está conectado al *Data Strip 1* y este a su vez conectado a los leds el dispositivo de juego.
- El pin 12 (PD6) está conectado al 4021N master2.
- El pin 13 (PD7) está conectado al 4021N master2.
- El pin 14 (PB0) está conectado al 4021N master2.
- El pin 15 (PB1) está conectado al pin 3 (CE) del NRF24L01.
- El pin 16 (PB2) está conectado al pin 4 (CSN) del NRF24L01.

- El pin 17 (PB3) está conectado al pin 6 (MOSI) del NRF24L01.
- El pin 18 (PB4) está conectado al pin 7 (MISO) del NRF24L01.
- El pin 19 (PB2) está conectado al pin 5 (SCK) del NRF24L01.

Conocidos los elementos y como están conectados entre sí se comienzan a eliminar los elementos que no van a hacer falta una vez instalado el nuevo procesador. Comenzamos eliminando la antena NRF24L01, cuyo esquemático se muestra en la Figura 12, y todas las conexiones que esta tiene.

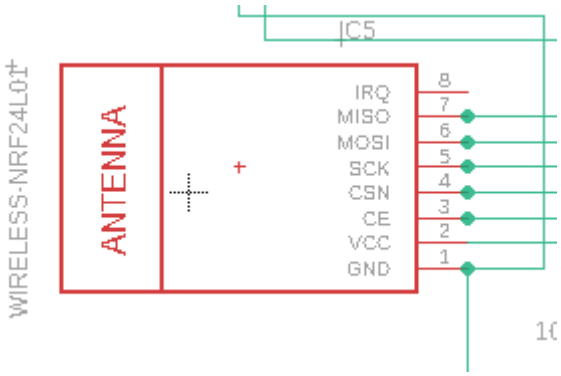


Figura 12. Esquema de la antena NRF24L01

Una vez eliminada esta se procede a eliminar el ATmega328P, mostrado en la Figura 11, y todas sus conexiones. Por Tanto, tras la eliminación de ambos dispositivos el esquema general del juego de espigas quedaría como se muestra en la Figura 12.

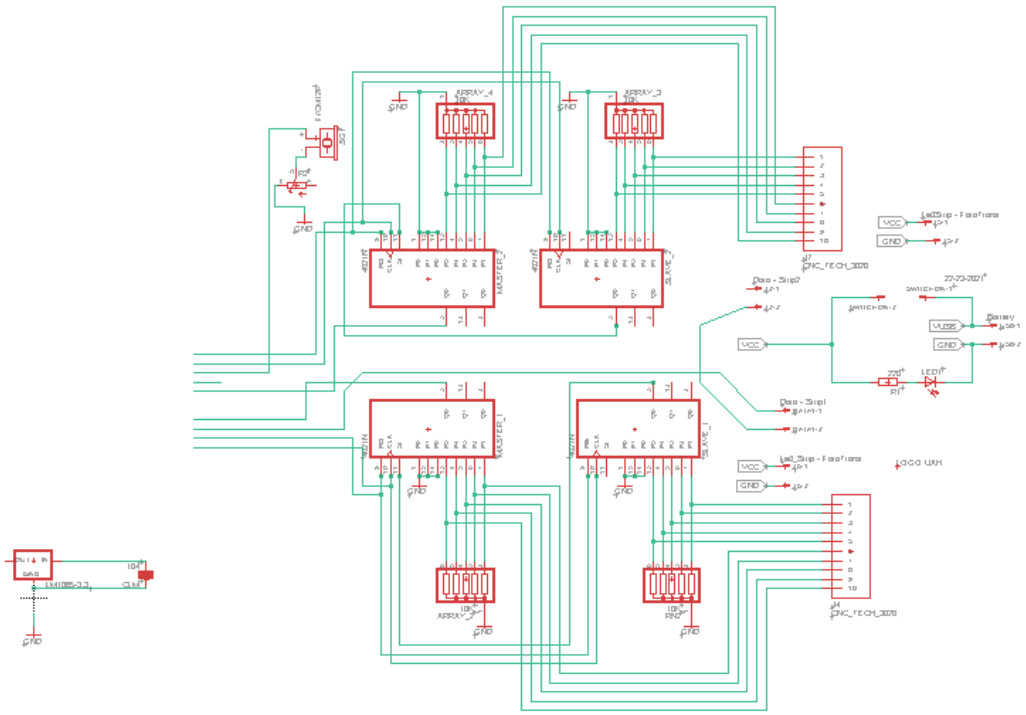


Figura 13. Esquema general con elementos borrados

Como se puede observar se ha dejado en la placa principal el LM1085 de 3.3V que estaba conectado a la antena NRF24L01 y cuyo uso y propósito se comentará más adelante. A continuación, se procede a la instalación del módulo ESP-WROOM-32 en el esquema mostrado en la Figura 13. A la hora de instalar el nuevo microcontrolador será necesario instalar junto a él nuevos componentes que permitan un funcionamiento correcto y una programación sencilla del mismo [15] [16].

Es necesario la instalación de un condensador de 0.1uF en paralelo con la alimentación del microchip cuya función principal es la de filtrado de ruido de la señal DC y otro de 104uF, conectado al regulador de tensión, que se encargará de estabilizar la señal de la alimentación durante periodos de variación de esta. Como hemos mencionado anteriormente el regulador de tensión LM1085 no se eliminará del circuito y se utilizará para que el ESP-WROOM-32 este alimentado a 3.3V. En la figura 14 se puede encontrar un esquemático de cómo quedará el ESP32 configurado con los condensadores y el regulador de tensión antes de unirlo a los demás componentes del juego de espigas.

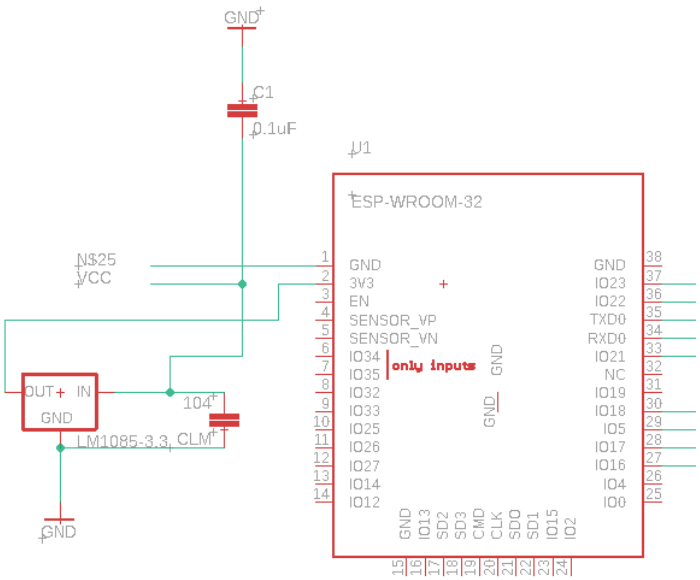


Figura 14. Esquemático general del ESP32

Es necesario conectar el ESP-WROOM-32 a los componentes de la placa principal que no han sido eliminados del juego de espigas para un correcto funcionamiento del mismo. A continuación, se explica pin a pin, tal y como se hizo anteriormente con el Amega328p, como está conectado el microcontrolador a cada uno de los elementos:

- En el IO19 está conectado el buzzer o bocina.
- En el IO23 está conectado el 4021N master2.
- En el IO22 está conectado el 4021N master2.
- En el IO21 está conectado el 4021N master2.
- En el IO18 está conectado el 4021N master1.
- En el IO5 está conectado al *Data Strip 1* y este a su vez conectado a los leds el dispositivo de juego.
- En el IO17 está conectado el 4021N master1
- En el IO16 está conectado el 4021N master1

En la Figura 15 se puede observar un esquemático general de como quedaría la placa principal una vez acoplado el nuevo microcontrolador [17].

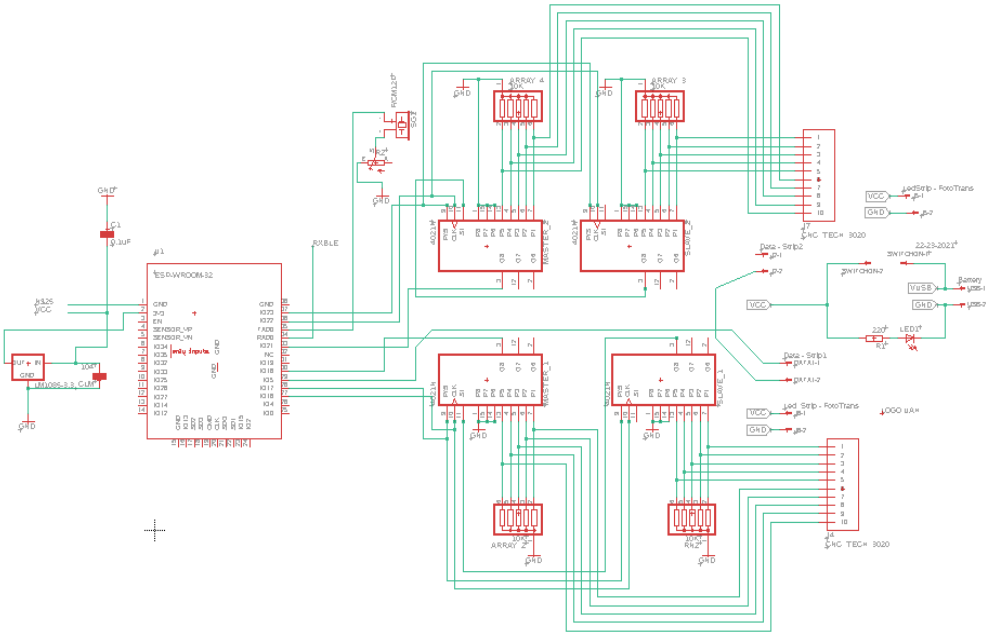


Figura 15. Esquemático general con el ESP32 integrado

El chip ESP32 es compatible con el protocolo TTL para su programación, por tanto, es necesario acoplar al circuito un conector Serie-USB, en nuestro caso utilizaremos un adaptador basado en el chip FT232RL. Este tipo de conversores USB a TTL tienen la función de facilitar la comunicación entre el PC y el microcontrolador por medio de protocolo USB y señales TTL, las cuales se basan en establecer una relación entre las interfaces USB del PC y serie (UART) del microcontrolador. Este adaptador se puede observar en la Figura 16. Para la programación del chip se empelarán los pines TX, RX, 3.3V y GND del convertidor serie-USB para poder establecer la comunicación entre ambos. El ESP-WROOM-32 ya posee en su interior un conversor de USB a TTL por lo que con solo usar el adaptador FT232RL ya será suficiente para poder programar el microcontrolador a través de USB.

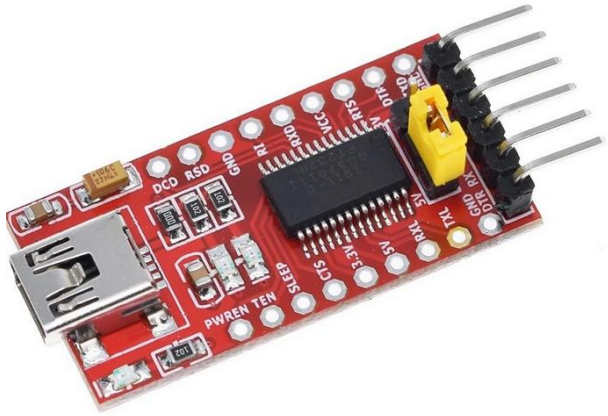


Figura 16. Circuito de programación serie-USB FT232RL

A la hora de acoplar este convertidor serie-USB se seguirá el esquema de la Figura 17, el cuál muestra una idea general de cómo se conectarán entre sí ambos. Se puede observar como también conectamos una resistencia en serie de 10KΩ para proteger al microcontrolador de posibles variaciones en la alimentación o corriente [18].

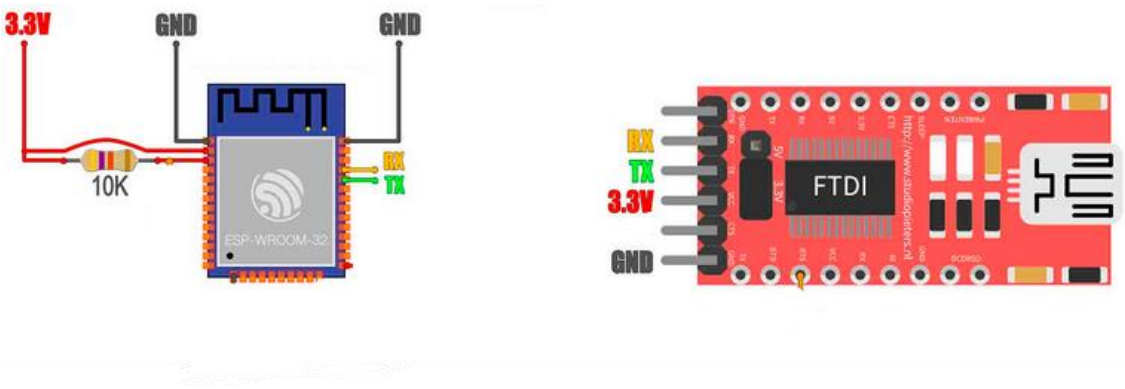


Figura 17. Esquema de montaje de circuito de programación y de ESP32

Aunque anteriormente se había indicado que posiblemente el código se podría cargar a través de OTA o una conexión inalámbrica, finalmente esta es solución empleada debido a que es una de las más económicas y la más sencilla de implementar en el juego de espigas, siendo además esta solución para cargar el código más segura y una vez depurado este no se prevén demasiadas actualizaciones del mismo al ser un código sencillo. Para un funcionamiento correcto y poder cargar el código correctamente es importante que el GPIO 0 esté conectado a GND para habilitar en el microcontrolador el proceso de carga de software. Otra manera que habría para habilitar el proceso de carga sería a través del pin CTS (*Clear to Send Data*), este pin se debería conectar al pin RTS del microcontrolador, debido a que ambos son los pines de control de flujo de datos. RTS es

una salida y CTS es una entrada. Si el pin CTS está a 0, indica que el dispositivo puede aceptar más datos a través del puerto TXD. Si el pin esta a 1 indica justo lo contrario, que el dispositivo no puede aceptar mas datos. A pesar de esto nos quedaremos con la primera forma mencionada para cargar el software en el microcontrolador. En la Tabla 3 se indica a que puertos y de qué manera se han conectado entre sí ambos componentes.

ESP32	**FT232RL**
GND	GND
3.3V	3.3V
U0R	TX
U0T	RX
GPIO 0	GND

Tabla 3. Conexión de ESP32 con FT232RL

El FT232RL permite que el VCC empleado sea tanto de 5V como de 3.3V, en nuestro caso vamos a utilizar la conexión de 3.3V. Más adelante se explicará el proceso que se lleva a cabo para poder cargar el programa en el nuevo microcontrolador. Es necesario integrar este nuevo componente en el circuito principal del juego de espigas siguiendo las conexiones mostradas en la Tabla 3. Una vez integrado, el convertidor serie-USB junto con el ESP32 y el LM1085 de 3.3V quedaría de la manera mostrada en la Figura 18.

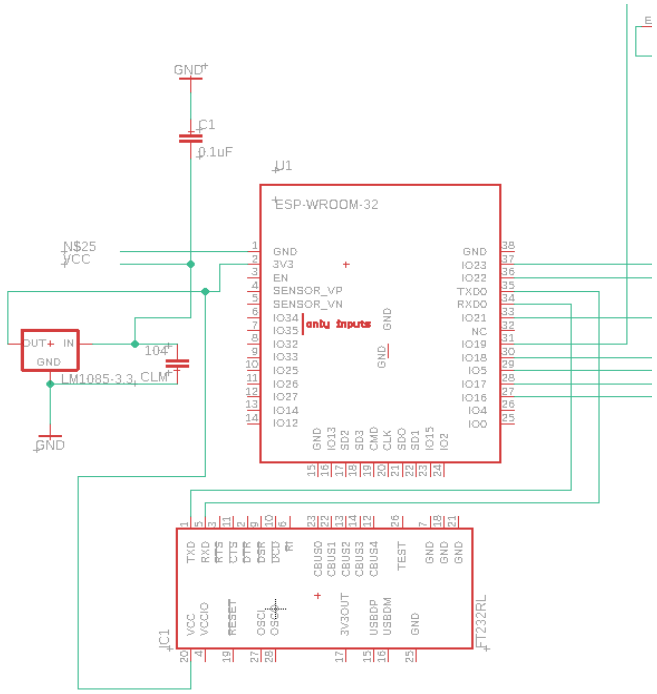


Figura 18. Esquema de conexión del ESP32 con el FT232RL

Una vez integrado el adaptador serie-USB el cambio de microcontrolador principal estaría completo y se puede avanzar a la siguiente mejora del juego de espigas.

5. Actualización de voltaje a 5V y 3.3V.

A continuación, se procede a conectar cada uno de los elementos al voltaje correspondiente y observar si se puede actualizar el voltaje de alimentación de algún componente y conectarlo a 3.3V.

En versiones anteriores del juego de espigas algunos elementos se quedaron sin conectar al realizar el esquema general, provocando que tuvieran que ser conectados manualmente con un soldador una vez se detectó el fallo. Aprovechando que se va a realizar una actualización estos elementos van a ser conectados a VCC como el resto de los elementos eliminando así la necesidad de hacerlo más tarde manera manual. Este es el caso de los registros de desplazamiento 4021N, en anteriores versiones estos se quedaron sin conectar a VCC por lo que hasta que no fueron conectados el juego de espigas no pudo funcionar correctamente. En esta nueva actualización del juego de espigas estos serán conectados a través de su pin 16 (VDD) a 5V, evitando así la necesidad de conectarlos de manera manual. Al conectar estos componentes, todos los componentes que existían en el juego en versiones anteriores quedan conectados a VCC y quedaría hacer una revisión a los

nuevos componentes para que ninguno de ellos se quede sin conectar y vuelva a surgir el problema que existía y estamos intentando subsanar. En cuanto a los nuevos componentes, el ESP-WROOM-32 se conecta a la salida del LM1085, ya que esta produce un voltaje de 3.3 V a partir de los 5V de entrada a los que está conectado el circuito. Lo mismo ocurre con el FT232RL, como hemos comentado anteriormente este componente tiene la opción de poder ser conectado tanto a 5V como a 3.3V, eligiendo en nuestro caso la conexión a 3.3V, por lo que también irá conectado a la salida del LM1085. La batería que se emplea para alimentar el juego de espigas es de 5V, tal y como se explicará más adelante, por eso ese es el voltaje principal que se utiliza en el dispositivo y a partir del cual se obtiene mediante convertidores el voltaje secundario al que va conectado el microcontrolador. Así todos los componentes, tanto los nuevos como los que se van a reutilizar quedarían conectados a VCC generando un esquema general final tal y como se muestra en la Figura 19.

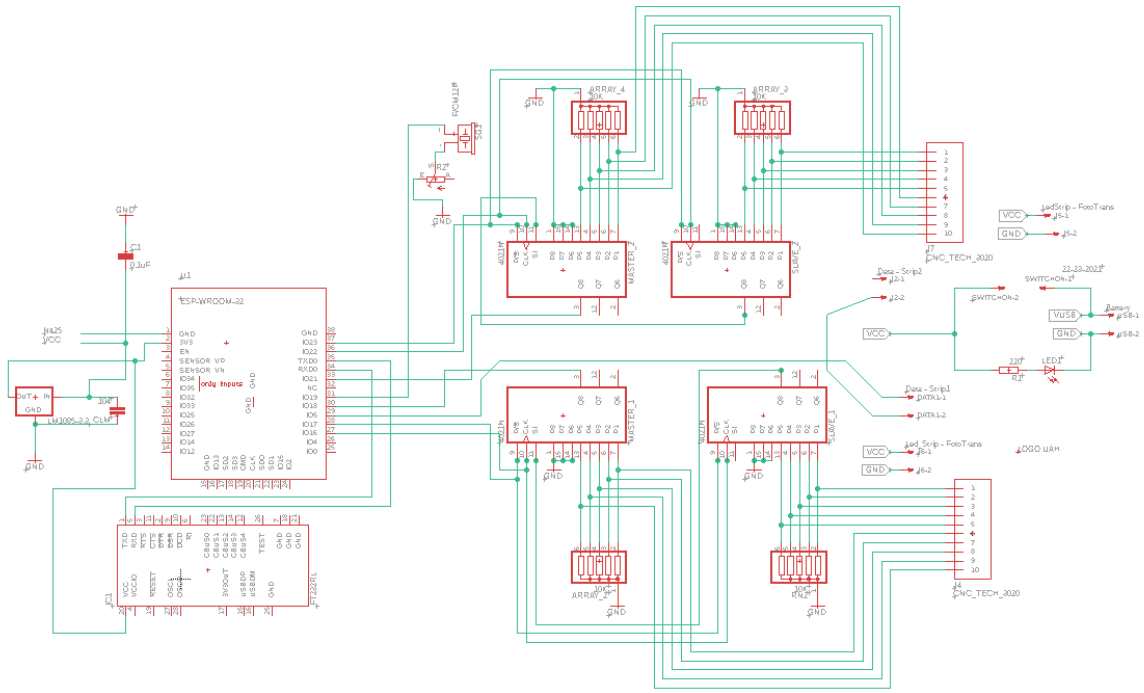


Figura 19. Circuito principal final del juego de espigas

6. Selección y montaje de USB-C y sensor de estado de batería.

Una de las principales mejoras que se pretende hacer es incluir en el dispositivo de espigas un conector de tipo C para la carga de la batería, y que no sea necesario abrir la tapa superior del juego de espigas para proceder a cargar el juego, y un sensor de estado para poder medir el nivel de la misma y saber cuándo es necesario cargarla. A continuación, se procede a explicar de forma detallada que sensor de nivel de carga y que tipo de USB-C se ha escogido y los motivos de esta elección. Tras su selección se procederá a explicar su conexión con el resto de los componentes y todo lo necesario para su montaje en una de las tapas laterales del juego de espigas.

En primer lugar, como ya se ha mencionado, vamos a detallar cuál es el sensor y el USB-C seleccionados para el juego de espigas. Tras revisar la batería utiliza en el dispositivo podemos observar que se trata de una de 5V por lo que tanto la clavija de conexión de tipo C como el sensor de nivel de batería tienen que ser compatibles con este voltaje. En nuestro dispositivo esta batería es una batería externa portátil conectada a los componentes, por lo que su conexionado será mucho más sencillo que si se tratará de otro tipo de fuente de voltaje.

Tras hacer una búsqueda exhaustiva por internet y encontrar varios tipos de modelos seleccionamos el cabezal hembra tipo C mostrado en la Figura 20 [19] y el sensor de nivel de batería mostrado en la Figura 21 [20].

Figura 20. Conexión USB-C

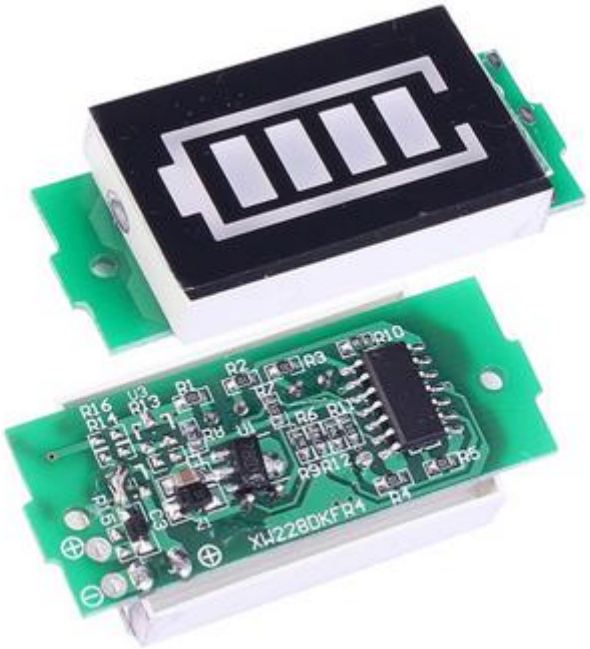


Figura 21. Sensor nivel de batería

Comenzaremos hablando del cabezal hembra tipo C de la Figura 20. Para realizar el cambio de conexión primero debemos tener en cuenta que el tipo de USB-A que ya porta la batería es 3.1 por tanto el cambio para realizar la carga de esta es viable. El tipo C elegido no es de tipo SS (*SuperSpeed*). Este tipo de USB permiten no solo la transferencia de datos y carga del dispositivo, sino también la transferencia de imagen y video. Si hubiéramos seleccionado un tipo C hembra de tipo SS el intercambio de USBs sería mucho más complicado ya que los USB tipo C SS necesitan información sobre si van a ser host o periférico y un mayor número de conexiones con el circuito principal. Como solo nos interesa la carga de la batería seleccionamos un tipo C hembra más barato y con una conexión mucho más sencilla. Para realiza el cambio hay dos formas de hacerlo [21]:

1. En primer lugar, podemos realizar unas conexiones en los pines del USB-A y conectar estos al USB-C hembra de la Figura 20 mediante cables. Como podemos ver el USB-C elegido tiene 4 pines para realizar posibles conexiones: V, D+, D- y G. Estos mismos pines se encuentran en el USB-A de la batería. La conexión entre uno y otro se realizaría de la manera que muestra la Figura 22.

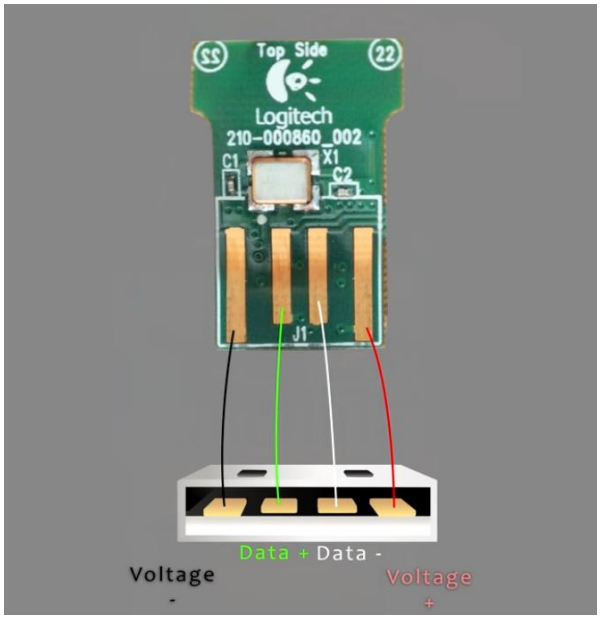


Figura 22. Conexión entre USB-A y USB-C

Como se puede observar en esta figura, los pines Data + y Data – corresponden con D+ y D- del tipo C seleccionado. Voltaje + corresponde con V y Voltaje – con G. Así el tipo C quedaría montado y listo para poder cargar la batería.

2. La segunda manera mediante la cual podríamos cambiar un USB por otro sería desoldando el USB-A de la placa de la batería y soldando el USB-C a la placa. Esta manera de sustituir los USB comparte el mismo principio que la anterior solo que en vez de soldar encima del anterior USB soldaremos encima de la placa. Como se puede observar en la Figura 23 el USB-A, como se ha mencionado anteriormente, tiene las mismas conexiones que el USB-C seleccionado debido a que los dos son hembras, por lo que solo habría que desoldar uno y soldar el nuevo en la misma posición.

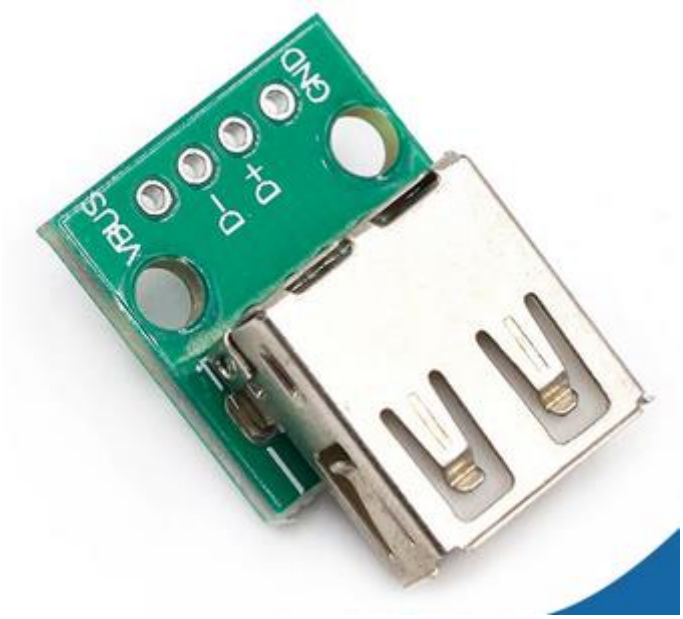


Figura 23. Conexiones de USB-A

Hay que tener en cuenta que tanto para la 1ª manera de cambio como para la 2ª hay que utilizar cables de una longitud determinada para poder llevar la conexión tipo C hasta el lateral del juego de espigas donde se pretende instalar esta conexión. Como se ha mencionado anteriormente esto es importante para eliminar el problema de tener que abrir la tapa cada vez que se quiera cargar el dispositivo.

Instalado el nuevo tipo de conexión de carga se procede a la instalación del sensor de nivel de batería [22].

Como ya hemos mencionado, en la Figura 21 se puede ver el sensor que hemos seleccionado, un sensor económico y sencillo de instalar para evitar posibles complicaciones en el conexionado con la batería. El montaje del mismo es muy sencillo, solo habrá que conectar los terminales positivo y negativo de la batería al sensor.

Los puntos de conexión en el sensor se pueden observar en la Figura 21, en la palca de abajo, siendo representado el positivo con + y el negativo con –. Para un correcto funcionamiento del sensor habría que desmontar la tapa de la batería y observar el número de pilas que lleve incorporadas para poder elegir correctamente el tipo de sensor que habría que comprar, siendo 1 pila 1S, 2 pilas 2S…. El sensor indica el nivel de carga de la batería como se puede observar en la Figura 24, dividiendo el porcentaje de batería en

4 niveles de 25% de carga cada uno, obteniendo un 100% de batería al mostrar las 4 rayas y un 25% al mostrar solo 1.

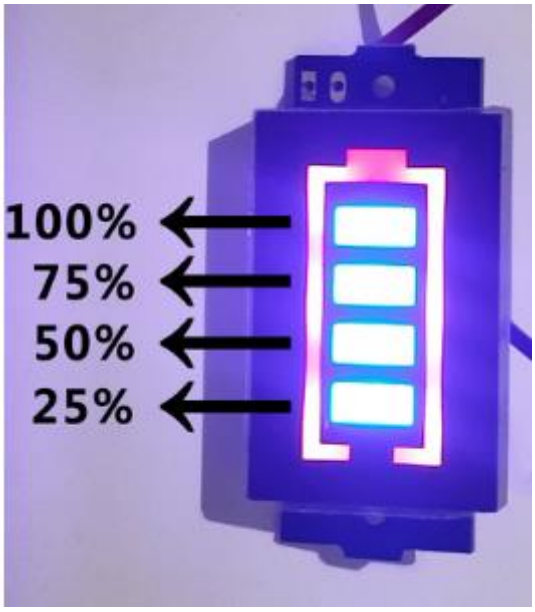


Figura 24. Niveles de carga de batería

Una vez terminado el nuevo puerto de carga y el sensor de batería se procede a seguir avanzando en las mejoras del dispositivo de espigas.

7. Diseño de PCB principal.

Para terminar con la actualización del hardware del juego de espigas hay que hacer un nuevo diseño del PCB principal del mismo. Como se han incluido nuevos elementos y se han eliminado otros a los que ya no se les iba a dar uso es necesario este nuevo diseño.

Para el rediseño del nuevo PCB partiremos del esquema general mostrado en la Figura 19 anteriormente. Gracias a la ayuda del programa EAGLE de la empresa AUTODESK podemos intercambiar entre el esquemático general del que ya hemos hablado y un modo para poder visualizar ese circuito creado en una PCB [23]. Esta visualización del esquema en modo PCB nos permite ver los componentes tal y como se verían en una PCB impresa, como se muestra en la Figura 25, aunque la mayoría de componentes y cables aparecen sin ordenar tal y como estarían en el esquema. En la figura se pueden observar conexiones de color rojo, que harían referencia a cables de cobre en la parte superior de la PCB y conexiones de color azul, que harían referencia a conexiones en la parte posterior de la misma.

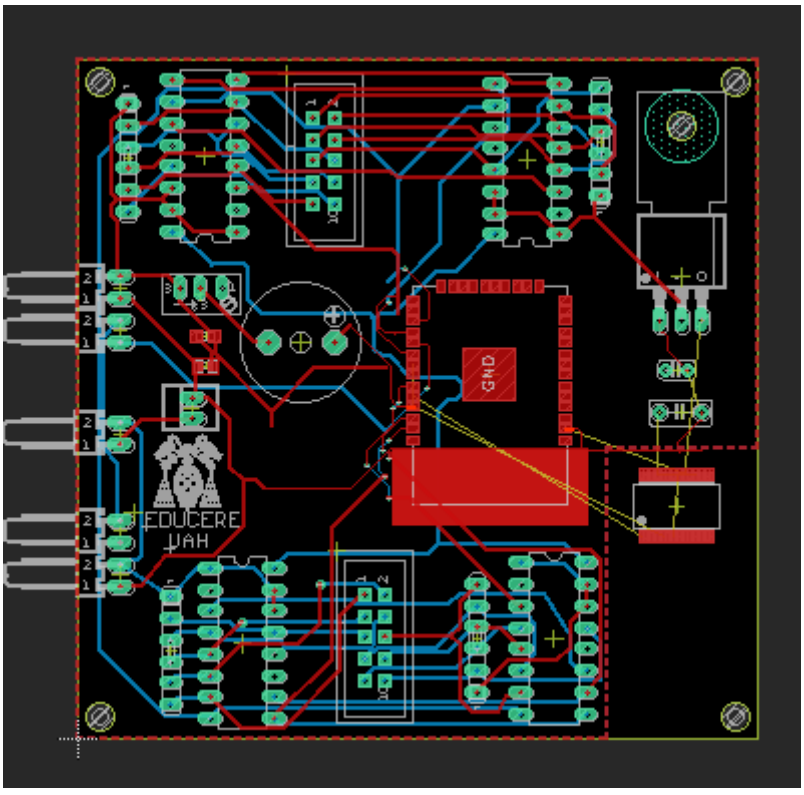


Figura 25. PCB inicial con nuevos elementos

Para tener nuestra PCB final lista el primer paso que haría falta sería ordenar los componentes y los cables para que queden de manera impresa en la placa. Además, es necesario agregarle 4 pines de programación, es decir, un pin de VCC general, uno de GND, uno de RESET para reiniciar la placa al completo y uno de I/O. Estos pines se van a reutilizar la de la PCB anterior, son los pines de conexión que se pueden observar en la parte izquierda de la PCB de la Figura 25. Para que no exista ningún tipo de confusión de cuales son estos pines, se muestran en la Figura 26.

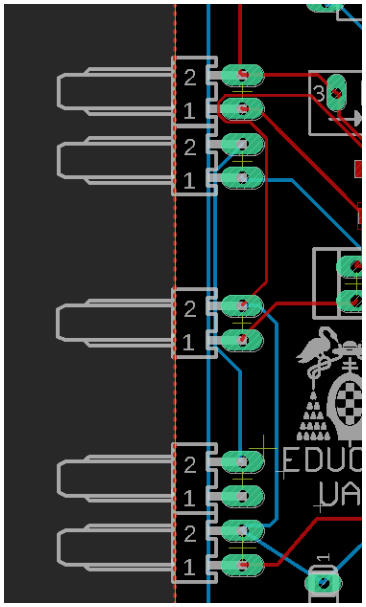


Figura 26. Pines de programación y alimentación

Se puede observar en el diseño inicial (Figura 25) como hay elementos que no estarían conectados en nuestra PCB (cables de color amarillo) por lo que habría que conectar esos componentes tanto a VCC y a GND como al resto de elementos. Una vez añadidos los pines de programación y colocados elementos y conectados entre sí mediante cables nuestra PCB final quedaría tal y como se muestra en la Figura 27.

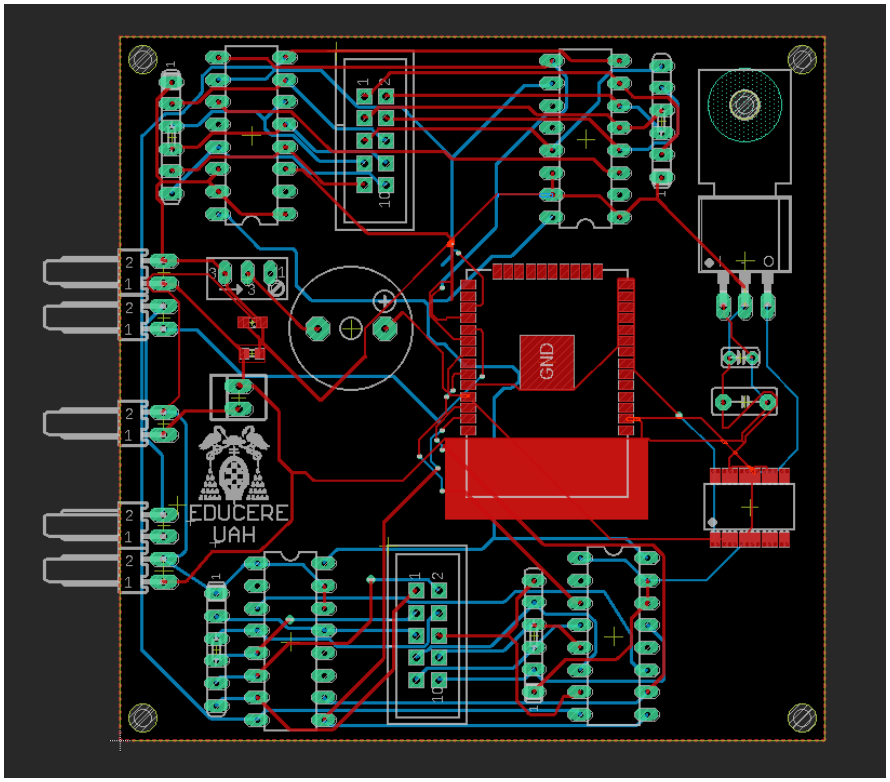


Figura 27. PCB final con nuevos elementos

Se puede observar el diseño de PCB final, tal y como se vería una vez impreso, en el Anexo I.

Capítulo IV: Actualización del software:

1. Estudio de la tecnología BLE (*Bluetooth Low Energy*).
2. Realización del nuevo software de Arduino.

1. Estudio de la tecnología BLE (Bluetooth Low Energy).

Para poder realizar una correcta modificación del software y teniendo en cuenta que vamos a realizar una transmisión de datos a través de Bluetooth usando el BLE que incorpora el ESP-WROOM-32, primero vamos a hacer un pequeño estudio a cerca del BLE y como configurarlo para proceder más adelante a explicar cómo se ha programado el software de Arduino que se va a cargar en el microcontrolador [24] [25].

El BLE se diferencia del bluetooth tradicional principalmente en cuanto a los consumos se refiere, siendo el consumo del BLE 100 veces menor de lo que puede consumir un dispositivo con bluetooth tradicional. Esto es debido a que mientras el bluetooth tradicional se mantiene siempre activo esperando una conexión, el BLE solo se activa al establecer dicha conexión. Además, en emparejamiento de dispositivos con BLE es mucho más rápido, siendo de unos pocos milisegundos, mientras que con bluetooth tradicional puede superar el centenar. EL BLE suele tener enviar datos en paquetes de tamaño pequeño ya que posee un ancho de banda menor que el bluetooth tradicional. Adicionalmente, el BLE no solo admite conexión punto a punto, sino también el modo de difusión de información y la distribución de información a través de una red malla (mesh network). En la Tabla 4 que se muestra a continuación se hace una comparativa un poco más exhaustiva de ambos tipos de bluetooth.

El principal motivo para la selección del BLE sobre el bluetooth tradicional es el consumo. El juego de espigas es un dispositivo en el cual se buscan los consumos más bajos posibles ya que no está conectado a una fuente constante de voltaje como podría ser un enchufe. Al ser su fuente principal de voltaje una batería externa se busca reducir estos consumos para alargar la vida del dispositivo lo máximo posible sin necesidad de cagarla.

	BLE	Bluetooth tradicional
Optimizado para...	Transmisión de datos de corta duración	Transmisión continua de datos
Banda de frecuencia	2.4GHz ISM band	2.4GHz ISM band
Canales	40 canales con 2MHz de espaciamiento	79 canales con 1MHz de espaciamiento
Modulación	GFSK	GFSK, π/4 DQPSK, 8DPSK
Consumo	0.01x a 05x (valor de referencia)	1 (valor de referencia)
Tipos de red	Punto a punto Malla Difusión	Punto a punto
Velocidad de datos	LE 2M PHY:2Mb/s LE 1M PHY:1Mb/s LE Coded PHY (S=2):500Kb/s LE Coded PHY (S=8):125Kb/s	EDR PHY(8DPSK):3Mb/s EDR PHY (π/4 DQPSK):2Mb/s BR PHY (GFSK): 1Mb/s

Tabla 4. BLE vs Bluetooth

El BLE puede tomar dos roles totalmente diferentes: el servidor y el cliente, tal y como se muestra en el esquema de la Figura 28 [26]:

- El servidor: anuncia su existencia, para poder ser encontrado por otros dispositivos, y contiene los datos que el cliente puede leer sobre el chip.
- El cliente: explora dispositivos cercanos, y cuando encuentra el servidor que está buscando, establece una conexión con él para poder leer los datos que le llegan de este.

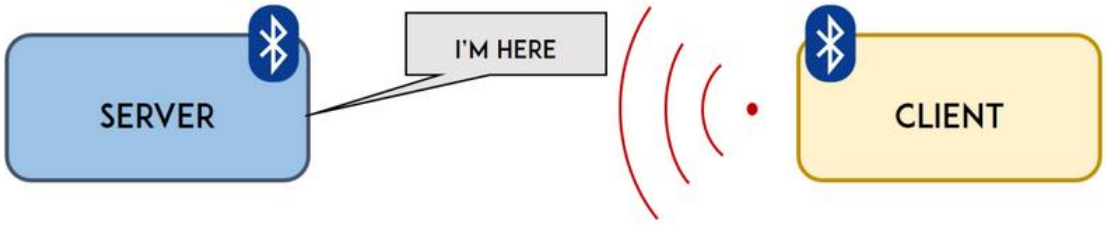


Figura 28. Comunicación punto a punto BLE

Este tipo de comunicación es el denominado, y mencionado anteriormente, comunicación punto a punto.

Como hemos mencionado anteriormente, el BLE también puede soportar transmisión de datos a través de difusión y red de malla:

- Modo difusión: El servidor transmite los datos a todo cliente que esté conectado a él.
- Modo de red de malla: Los datos se transmiten a través de una red de dispositivos conectados entre sí.

El BLE tiene una pila de protocolos divididos en 3 capas distintas: la capa física, la capa de enlace y la capa L2CAP (*Logic Link Control and Adaptation Protocol*):

- La capa física contiene todos los circuitos de comunicaciones para poder realizar los procesos tanto de modulación como de demodulación de las señales analógicas que recibe el cliente y transformarlas en señales digitales.
- La capa de enlace (*link layer*) es la encargada de gestionar diversas características necesarias de este tipo de bluetooth como puede ser los requerimientos temporales, el chequeo de la información recibida y su reenvío, gestión y filtrado de esa información… Además, se encarga de determinar los distintos roles de cada uno de los dispositivos que intervienen en la comunicación (cliente, servidor, master, slave). El nivel LL propio de esta capa es el encargado de controlar los procesos que conllevan un cambio de parámetro. Esta capa cuenta además con protocolo HCI que permite la comunicación entre un host (*o anfitrión*) y un controlador para poder llevar a cabo la comunicación interfaz serie entre ambos.

- Por último, la capa L2CAP es la responsable de dos de las tareas más importantes de dispositivos con BLE: la multiplexación y la fragmentación y recombinación de información. En primer lugar, el proceso de multiplexación es la capacidad del microchip de dar un formato a la distinta información que viene de las otras dos capas y comprimirlas en paquetes de datos BLE. En segundo lugar, la capacidad de fragmentación y de recombinación es la característica mediante la cual el microcontrolador es capaz de tomar grandes paquetes de información y fragmentarlos y reorganizarlos adecuándose a los protocolos predefinidos del BLE. Está capa es la encargada de permitir y gestionar los dos protocolos fundamentales en los cuales nos vamos a centrar: GAP y GATT

El protocolo GAP (*Generic Acces Profile*) es el encargado de permitir que nuestro dispositivo sea visible para el resto de dispositivos y que por tanto se puedan conectar a él. Este proceso ocurre cuando dos dispositivos han pasado la primera fase, que es la encargada de el establecimiento de la conexión para tener comunicación (controlada por el protocolo GAP), y da comienzo el intercambio de información pudiendo ser de forma unidireccional o bidireccional.

El protocolo GATT (*Generic Attribute Profile***)** ocurre una vez los dispositivos han superado el anterior protocolo. Este protocolo determina como los dispositivos intercambian información. El protocolo GATT sigue una estructura jerárquica, mostrada en la Figura 29, mediante la cual se conectan los dispositivos con BLE. Esta estructura es importante debido a que facilita el entendimiento del BLE y como desarrollar los programas que sirven para utilizarlo.

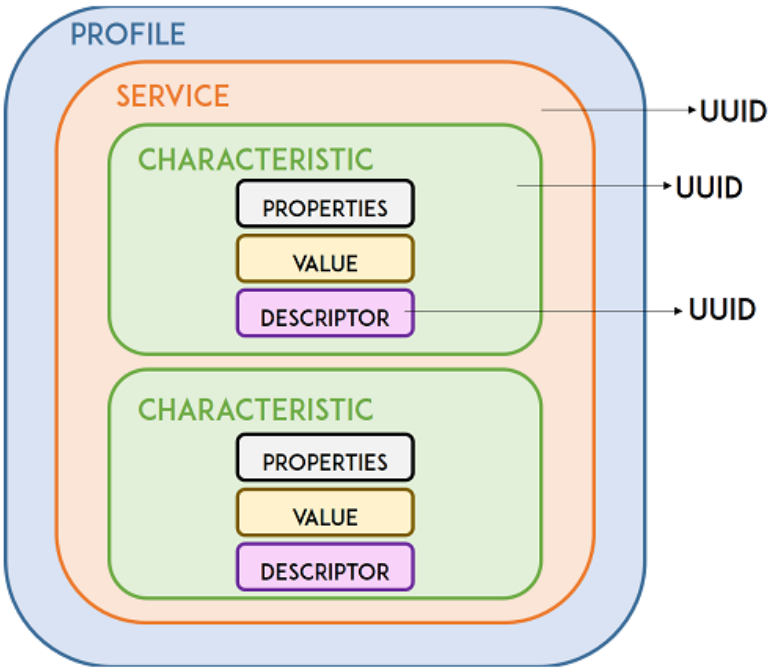


Figura 29. Jerarquía de protocolo GATT

En el nivel más alto de la pirámide jerárquica encontramos el perfil (*Profile*), el cual está formado por uno o más servicios (*Service*). Usualmente, el BLE contine más de un solo servicio. Cada servicio contiene en su interior al menos una característica (*characteristic*) o pude referenciar a otros servicios del perfil. Un servicio es simplemente una colección de información, donde las características contienen propiedades, valores y descriptores de esa información. En las propiedades se describe la forma en la que pueden interactuar las características, como las operaciones que estas pueden realizar. Todo servicio, característica y descriptor contine un UUID (*Universally Unique Identifier*), un número único de 128 bits (*16 bytes*) que sirve como identificador. Existen diversos tipos de UUID acortados para todo tipo de servicio, característica o descriptor que lo necesite los cuales vienen definidos en el SIG (*Bluetooth Special Interest Group*) [27]. Centrándonos en el ESP-WROOM-32 y su uso del BLE en este microcontrolador, en el caso del proyecto que estamos llevando a cabo y como se ha mencionado anteriormente, vamos a programarlo utilizando el IDE Arduino. El proceso de configuración se explica a continuación.

2. Realización del nuevo software de Arduino.

Es necesario explicar cómo se ha desarrollado el nuevo software que se va a utilizar para poder programar el ESP-WROOM-32 y que sea capaz de recibir y enviar los datos. El ESP-WROOM-32 va a actuar de servidor de datos ya que es el rol que adopta en el dispositivo que estamos actualizando. De todo lo mencionado anteriormente solo se creará un perfil, dos características y dos servicios ya que para el envió de los datos no necesitamos más. A la vez que vamos comentado el proceso de configuración del nuevo software se explicará el motivo de cambiar ciertas partes del código que antes eran necesarias y ahora ya no se utilizarán. Se va a comentar en primer lugar la creación del servidor y características del microcontrolador, para más adelante comentar otras partes fundamentales del programa completo. Para ello hay que instalar la placa ESP-32 en Arduino [28] y comprender como funciona este IDE [29] [31].

En primer lugar, tras estudiar como enviar y recibir datos con el ESP-32 y el BLE [30][32] [33], se van a incluir las librerías necesarias para que funcione correctamente el BLE y a suprimir aquellas que ya no van a ser utilizadas. En la Figura 30 se pueden observar tanto las librerías nuevas que hemos incluido como aquellas que hemos eliminado.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLE2902.h>
#include <SPI.h>
#include <FastLED.h>
```

```
#include <FastLED.h>
//#include <nRF24L01.h>
#include <RF24.h>
//#include <RF24_config.h>
```

Figura 30. Librerías BLE vs NRF24L01

En la parte izquierda de la imagen podemos ver las nuevas librerías características del BLE que se van a utilizar en esta nueva versión del programa, mientras que en la parte derecha se observan aquellas librerías que hacían referencia a la antena de radio frecuencia NRF24L01 y que, por tanto, al ser eliminada y cambiar el tipo de comunicación ya no van a ser necesarias. Se mantienen las librerías <SPI.h> y <FastLED.h> [34] para mantener ciertas funciones necesarias en el dispositivo, como pueden ser la iluminación de los leds o la reproducción de sonidos. Tras esto, es necesario definir los UUIDs que se van a utilizar en el proyecto para poder establecer la comunicación vía wifi.

Estos UUIDs se muestran en la Figura 31 y han sido obtenidos de manera aleatoria en una web que sirve para generarlos, ya que solo son números de identificación [35].

```
#define SERVICE_UUID_TX        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define SERVICE_UUID_RX        "47160bda-2783-11ed-a261-0242ac120002"
#define CHARACTERISTIC_UUID_TX "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define CHARACTERISTIC_UUID_RX "4e92ce02-2783-11ed-a261-0242ac120002"
```

Figura 31. UUIDs del servicio y de la característica

Solo se han definido UUIDs para dos servicios y dos características, una para enviar datos y otra para recibirlos, ya que, como hemos mencionado anteriormente, para este tipo de proyectos con definir el menor número de cada uno es suficiente para una ejecución correcta de los protocolos. Se puede observar cómo se crean dos UUIDs para recibir datos y dos para enviarlos. Una vez configurados estos UUIDs se procede a la creación en sí de los servicios y de las características. Comenzamos iniciando el dispositivo BLE que se va a utilizar, tal y como se muestras en la Figura 32.

```
BLEDevice::init("MyESP32 Juego de Espigas");
```

Figura 32. Inicialización del BLE

El nombre elegido para que el ESP32 sea reconocido a través del BLE es el mostrado en la figura anterior: “MyESP32 Juego de Espigas”. Una vez iniciada la comunicación por BLE procedemos a crear el servidor para que este sea capaz de enviar y de recibir datos. Creamos este servidor del dispositivo BLE, con el comando mostrado en la Figura 33, para después proceder a crear los servicios necesarios para el envió y el recibimiento de estos datos con las sentencias mostrada en la Figura 34.

```
BLEServer *pServer = BLEDevice::createServer();
```

Figura 33. Creación del servidor del BLE

```
BLEService *pService1 = pServer->createService(SERVICE_UUID_TX);
BLEService *pService2 = pServer->createService(SERVICE_UUID_RX);
```

Figura 34. Creación de los servicios del BLE para envió y el recibimiento de datos

Por último, en cuanto a configuración se refiere, y dejar el BLE del dispositivo totalmente configurado, configuramos las características del mismo. Para ello utilizamos en primer lugar las sentencias mostradas en la Figura 35 donde, además de crearla, definimos las

propiedades necesarias para que funcione correctamente. Esta característica será la utilizada para poder enviar los datos de manera correcta.

```
BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                       CHARACTERISTIC_UUID_TX,
                                       BLECharacteristic::PROPERTY_READ |
                                       BLECharacteristic::PROPERTY_WRITE |
                                       BLECharacteristic::PROPERTY_NOTIFY |
                                       BLECharacteristic::PROPERTY_INDICATE
                                     );
```

Figura 35. Creación de la característica para enviar datos y definición de sus propiedades

Sin embargo, es necesario configurar otra característica para el recibimiento de los datos del cliente y que este pueda recibirlos de manera correcta una vez procesados por el microcontrolador, como ya se había mencionado anteriormente. Las sentencias utilizadas para configurar esta segunda característica se muestran en la Figura 36.

```
BLECharacteristic *pCharacteristic2 = pService->createCharacteristic(
                                        CHARACTERISTIC_UUID_RX,
                                        BLECharacteristic::PROPERTY_WRITE
                                      );
```

Figura 36. Creación de la característica para recibir datos y definición de sus propiedades

Para finalizar y tras haber terminado todas las configuraciones, inicializamos el servicio y comenzamos a anunciar el servidor del dispositivo para que pueda ser encontrado por cualquier cliente tal y como se muestra en los comandos utilizados de la Figura 37. Además, podemos observar cómo se añade el descriptor a la característica de envió de datos, el descriptor 2902, el cual permite al cliente activar las notificaciones cuando él quiera, de esta manera se evita el envío innecesario de datos.

```
pCharacteristic->addDescriptor(new BLE2902());
pService->start();
pServer->getAdvertising()->start();
```

Figura 37. Descriptor de envió de datos, activación de servicio e inicio de aviso de servidor

Una vez terminada la configuración del BLE e iniciado el mismo se procede a explicar otras partes del código importantes para el correcto funcionamiento del software. En primer lugar, vamos a especificar en el software en que pines están conectados los distintos elementos del hardware, tal y como se muestra en la Figura 38, para que el dispositivo sea capaz de funcionar correctamente.

La conexión seguida es la explicada anteriormente en el capítulo referente a la actualización del hardware, en la sección referente al cambio de microcontrolador principal [36].

```
int latchPin1 = 17;
int dataPin1 = 18; //Master/slave1 in hardware - Fila 1
int clockPin1 = 2;
int latchPin2 = 16;
int dataPin2 = 21;  //Master/slave2 in hardware - Fila 2
int clockPin2 = 22;
int speakerPin = 19;     // speaker  GND and pin 19
```

Figura 38. Definición de pines en Software

Con los pines definidos, vamos a realizar una función en bucle la cual nos permita iniciar la actividad del juego de espigas una vez un dispositivo esté conectado a través de la variable *deviceConnected*. Si hay un dispositivo conectado esta variable se pondrá a valor *True* y permitirá iniciar la ejecución del programa del juego de espigas y la lectura de datos del mismo. En caso de que no haya ningún cliente conectado a nuestro servidor la variable estará a valor *False* y seguirá anunciado nuestro servidor hasta que finalmente se conecte un cliente. Todas las sentencias y variables utilizadas para esta función se encuentran definidas en la Figura 39.

```
bool deviceConnected = false;
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* MyServer) {
    deviceConnected = true;
  };

  void onDisconnect(BLEServer* MyServer) {
    deviceConnected = false;
  }
```

Figura 39. Función para conexión de dispositivo

Es necesario también comentar las funciones que se van a encargar de el recibimiento y el envió de datos a cliente, las cuales funcionarán a través de interrupciones [37]. Así, comenzaremos comentando la función para poder recibir los datos. En primer lugar, se muestran las sentencias utilizadas para esta lectura de los datos en la Figura 40.

```
unsigned int readData( byte *buff_rx)
{
  unsigned int response = 0;
  if (deviceConnected = true) {
    class CharacteristicCallbacks: public BLEcharacteristicCallbacks{
    void onWrite (BLEcharacterisitc *pcharacteristic2){
    byte c =  std::string rxValue = characteristic->getValue();
    response = c;
    Serial.print( buff_rx, c);
  }
  }
  }
  return response;
}
```

Figura 40. Función para lectura de datos

Esta función solo iniciará en caso de que haya un dispositivo conectado, siendo posible esto gracias a la variable *deviceConnected* que ya habíamos mencionado anteriormente. En caso de que el juego de espigas esté conectado a un dispositivo este será capaz de leer datos gracias a la característica *CHARACTERISTIC_UUID_RX* la cuál hemos denominado en nuestro programa *pcharacteristic2*. Está recibirá los datos a través de la función *getValue()* y los guardará en la variable *C*, la cual más adelante la denominaremos *Response* para almacenar estos datos que se reciben de cliente.

Para el envió de datos utilizaremos los comandos mostrados en la Figura 41 que se muestra a continuación.

```
void sendData(byte device, byte sensor, byte sec , byte len, byte *data)
{
  byte packet[32];
  byte resp[2];
  int counter = 0;
  int i = 0;
  packet[i++] = device;  //device_id
  packet[i++] = sensor;   //data_id
  packet[i++] = sec;   //data_id
  packet[i++] = len;   //data_id
  for (int j = 0; j < len; j++) {
    packet[i++] = data[j];
  }
  pcharacteristic->setValue(&packet[0]);
  pcharacteristic->notify();
  Serial.print(&packet[0], i);
}
```

Figura 41. Función para el envió de datos

Como se puede observar vamos a recibir 4 tipos de datos distintos: *device* (hace referencia al dispositivo), *sensor* (el sensor que esta enviado los datos), *sec* (instante de tiempo después del inicio de la actividad) y *len* (que es la longitud del vector de los datos que se van a enviar). Una vez compuesto este vector en la variable *packet* se guardará en la variable *data* y será enviado al cliente a través de la característica *CHARACTERISTIC_UUID_TX* la cual hemos denominado *pcharcteristic*. Este envió lo hará gracias a dos funciones:

- *setValue():* la cual guarda el valor del vector en la característica para prepararlo para su envió al cliente.
- notify(): envía el vector obtenido al cliente para que este pueda observar los datos recogidos del juego de espigas.

Del anterior software programado para el Atmega328p heredaremos el resto de código ya que salvo por la forma de enviar y recibir los datos, además de conexiones, el juego de espigas va a seguir funcionando de la misma manera. Cabe mencionar la manera en la cual se van a reproducir los sonidos a través del buzzer. Para ello primero se configurarán las notas musicales a utilizar tal y como se muestra en la Figura 42. Estas notas serán utilizadas para reproducir una canción la cual entrara en reproducción cuando el usuario del juego introduzca o saque la espiga del agujero correspondiente. Para reproducir la música se emplearán los comandos utilizados en la Figura 43, función que entrará en funcionamiento cada vez que en el código se inicie la variable *sound*, la cual hará que entre en funcionamiento la función startWarsMusic que aparece en la Figura 43 ya mencionada.

```
int c[5] = {131, 262, 523, 1046, 2093}; // Do
int cs[5] = {139, 277, 554, 1108, 2217}; // Do#
int d[5] = {147, 294, 587, 1175, 2349}; // Re
int ds[5] = {156, 311, 622, 1244, 2489}; // Re#
int e[5] = {165, 330, 659, 1319, 2637}; // Mi
int f[5] = {175, 349, 698, 1397, 2794}; // Fa
int fs[5] = {185, 370, 740, 1480, 2960}; // Fa#
int g[5] = {196, 392, 784, 1568, 3136}; // Sol
int gs[5] = {208, 415, 831, 1661, 3322}; // Sol#
int a[5] = {220, 440, 880, 1760, 3520}; // La
int as[5] = {233, 466, 932, 1866, 3729}; // La#
int b[5] = {247, 494, 988, 1976, 3951}; // Si
```

Figura 42. Notas musicales del juego de espigas

```
void startWarsMusic(void)
{
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin, ds[2]); delay(500); noTone(speakerPin); delay(1);
  tone(speakerPin, as[2]); delay(125); noTone(speakerPin); delay(25);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin, ds[2]); delay(500); noTone(speakerPin); delay(1);
  tone(speakerPin, as[2]); delay(125); noTone(speakerPin); delay(25);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin);
}
};
```

Figura 43. Canción utilizada para el juego de espigas

Para poder observar el resto de funciones del nuevo software, como puede ser la actualización del estado de los sensores de las espigas con la función *updateCylinderStatus*, se puede observar el código completo en el Anexo II.

Capítulo V: Actualización del diseño del juego de espigas:

1. Revisión de conectores y resistencias del juego de espigas
2. Actualización del diseño 2D de las tapas del dispositivo.

1. Revisión de conectores y resistencias del juego de espigas.

Se procede ahora a actualizar el diseño actual del juego de espigas. En primer lugar, vamos a revisar los conectores y las resistencias empleadas actualmente y en caso de encontrar una mejor opción, ya sea por la calidad del componente o por ser más económico, se sustituirá por el que se está utilizando actualmente.

Comenzamos con la revisión de las resistencias. Actualmente se utilizan resistencias de tipo normal, es decir de película metálica como las mostradas en la Figura 44, por lo que vamos a hacer un estudio de mercado para ver si hay alguna solución más óptima que soldar varias resistencias entre sí en paralelo. Estas resistencias se utilizan tanto en los optoacopladores del circuito como en la placa principal del mismo, soldadas varias entre sí para obtener el valor buscado, por lo que solo sería necesario sustituirlas en estos lugares en caso de que encontremos alguna resistencia cuyo uso y aplicación sea mejor para este proyecto. El valor de resistencia que buscamos sustituir y que aportan las resistencias actuales es de 40KΩ siendo estas resistencias de película metálica.

Figura 44. Resistencia de película metálica

Hay dos posibles sustitutos para las 4 resistencias actuales:

- Utilizar una resistencia como la mostrada en el Figura 44, pero cuyo valor sea directamente de 40KΩ, de tal manera que simplifiquemos tanto el coste del dispositivo al reducir el número de componentes a comprar, como el número de elementos a soldar en la PCB y por tanto un posible menor número de errores que pueden aparecer por soldar mal un componente. Además, al tener menos elementos la probabilidad de que uno falle también disminuye.

- Utilizar una resistencia más moderna y que también sea capaz de entregar los 40KΩ que buscamos. Este tipo de resistencia son como las mostradas en la Figura 45, cuyo nombre es resistencias de película delgada. Este tipo de resistencias son de menor tamaño y se integran directamente en la PCB de los circuitos electrónicos. Suelen ser más precisas que las resistencias de película gruesa o que las resistencias de película metálica, que junto con su menor tamaño como hemos mencionado antes, hace que sean las principales resistencias escogidas hoy en día para las PCBs electrónicas.

Figura 45. Resistencia de película delgada

La solución que se va a emplear es la primera que se ha comentado. Este motivo es debido a que es posible encontrar resistencias de 40KΩ de película metálica en el mercado actual muy económicas y de múltiples distribuidores distintos. Otro problema por el cual se opta por esa solución es debido a que la sustitución por posible fallo de la resistencia de película delgada es mucho más complicada ya que esta va directamente soldada a la PCB, pero de una manera más directa y cercana que la resistencia de película metálica, por lo que al sustituirla las probabilidades de dañar la PCB principal con mucho mayores.

A continuación, se procede al estudio de mercado para ver si hay algún conector de mejor calidad y más económico que el que se está usando actualmente. En la Figura 46 se puede observar el conector que se estamos buscando sustituir, es el conector de las tiras de led del juego de espigas.

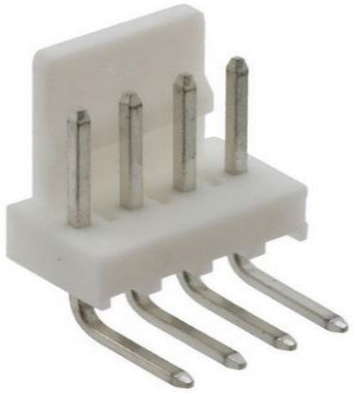


Figura 46. Conector tiras de led MOLEX 22-05-7048

El principal problema de estos conectores es su debilidad y la debilidad de su conexión, además de la facilidad con la que tienden a desconectarse las tiras de led ante cualquier movimiento. Por ello se busca poder sustituirlos. Por tanto, para evitar estos problemas y que la conexión sea más robusta y estable se seleccionan los conectores mostrados en la Figura 47 [38]. La ventaja de estos conectores, aparte de que son más económicos y por tanto disminuyen el precio del dispositivo, es que poseen una tapa que protege la conexión y evita que se suelte con facilidad. Además, son conectores que no necesitan soldadura, aunque cabe mencionar que los que ya se estaban utilizando tampoco la necesitaban.

Figura 47. Conector seleccionado para las tiras de led

El resto de conectores del dispositivo no se van a sustituir ya que proporcionan una conexión estable y robusta de los elementos del juego de espigas.

2. Actualización del diseño 2D de las tapas del dispositivo.

Para terminar con la actualización del juego de espigas vamos a hacer una actualización de las tapas del dispositivo para poder incluir el sensor del nivel de batería y el cabezal tipo C hembra para la carga de esta. Este rediseño se va a hacer de manera muy sencilla, se cogen los diseños de las tapas que ya existían y se acoplan en una de las caras laterales el hueco para el cabezal tipo C y el sensor del nivel de batería. Las medias utilizadas para el señor de tipo C son las extraídas de la Figura 48 que aparece a continuación, son medidas estándar ya que todos los cabezales de tipo C hembra que sean del mismo tipo del que hemos escogido van a tener esas medidas.

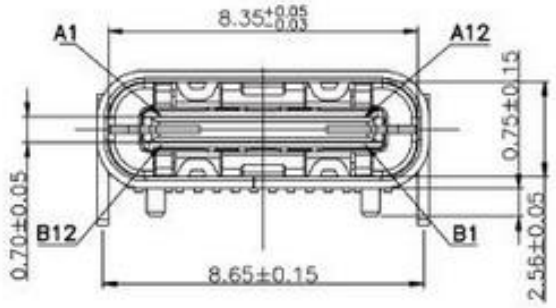


Figura 48. Medias cabezal tipo C hembra

En cuanto a las medidas del sensor de nivel de batería seleccionado se han utilizado las medidas sacadas de la página oficial de compra del señor. Estas medidas se pueden observar en la Figura 49. Esta figura ha sido modificada de manera manual para incluir las medidas ya que el vendedor solo las incluye de manera teórica. Este sensor se va a montar de manera horizontal.

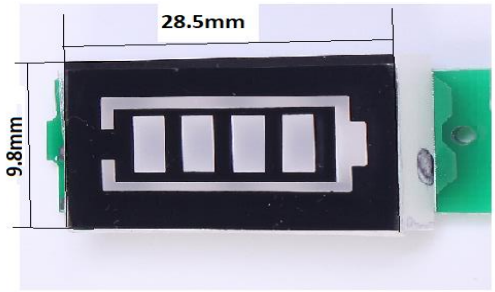


Figura 49. Medidas sensor de nivel de batería

Una vez conocidas las medidas lo integramos en la tapa lateral del dispositivo, la cual quedaría con ambos integrados como se muestra en la Figura 50.

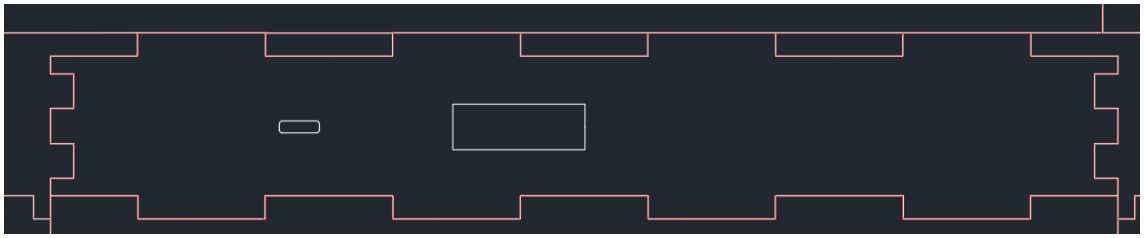


Figura 50. Tapa lateral con tipo C y sensor de nivel de batería

Tras colocar ambos elementos, y terminar el rediseño de la tapa lateral nuestro juego de espigas quedaría finalmente actualizado y listo para que el usuario pruebe el sistema. El plano completo 3D de todas las caras del juego de espigas se puede observar en el Anexo III.

Capítulo VI: Conclusión y líneas futuras:

1. Conclusión del proyecto.
2. Líneas futuras.

1. Conclusión del proyecto.

Durante todo el desarrollo del proyecto se ha observado como ha ido evolucionando un juego de espigas ya funcional a través de unas propuestas que introducen ciertos cambios tanto a nivel de hardware, como de software, como de diseño externo del dispositivo, para que su uso sea más intuitivo, más sencillo y este sea de mayor calidad.

Estas propuestas parten de la idea de que ningún dispositivo creado desde cero es perfecto en sus primeras versiones, si no que todo dispositivo necesita actualizaciones para no quedarse desfasado y siempre tener incorporada la última tecnología del mercado necesaria para un correcto funcionamiento del mismo. Además, estas actualizaciones sirven para subsanar fallos detectados tras el montaje del dispositivo y mejorar características con ideas posteriores a las primeras versiones del mismo.

En primer lugar, se ha realizado una actualización del hardware del dispositivo que viene motivada por la mejora de las comunicaciones y de la forma de recibir los datos que tiene el usuario, además de intentar abaratar los costes del juego de espigas. Este cambio se ha hecho de manera ordenada y, como se ha mencionado a lo largo de dispositivo, intentando mantener en la medida de lo posible un diseño más parecido al juego de espigas original. Con este cambio conseguimos el objetivo propuesto, ya que ahora el juego de espigas enviará los datos a través de BLE y no por radiofrecuencia, siendo su uso mucho más sencillo para el usuario, además de conseguir eliminar algún componente de gran tamaño de la PCB como puede ser la antena de radiofrecuencia.

En segundo lugar, dentro de los objetivos de un carácter más importante, se abordó la actualización del software a través del IDE Arduino para que el nuevo microcontrolador fuera capaz de enviar y recibir los datos. La premisa de este cambio era la misma que en la actualización del hardware, actualizar el programa de comunicaciones manteniendo la esencia del original ya que esté ya funcionaba correctamente. Este cambio se ha realizado con éxito, permitiendo con ambas actualizaciones que el usuario del juego tenga menos dificultades a la hora de conectarse al microcontrolador para poder recibir los datos.

Cabe destacar que este proyecto se ha realizado bajo la idea de una actualización teórica, por lo que el mismo no es totalmente funcional hasta que no se implemente en el juego de espigas y se observen fallos que, de momento y como suele ser normal en este tipo de

trabajos, no se han podido detectar. Esto es debido a que muchos de esos fallos solo se manifiestan una vez el dispositivo ha sido creado en forma de prototipo.

En cuanto a la planificación seguida para desarrollar el proyecto, cabe destacar el hecho de que no se respetó la distribución de tiempo planteada al inicio del mismo, en el anteproyecto. Esto es debido a que todo proyecto cuenta con dificultades para ser desarrollado y rara vez se cumple una planificación inicial. Estas dificultades y la aparición de épocas donde predominan los tiempos muertos por diversos motivos hacen que el cumplimiento de los tiempos de desarrollo del proyecto se vuelva complicado. No obstante, y a pesar de esto, la propuesta de actualización ha sido terminada con éxito.

Las mayores dificultades que se han podido encontrar a lo largo del desarrollo del TFG han sido relacionadas sobre todo con los programas utilizados para la actualización de este juego de espigas [39] [40]. Cabe destacar la dificultad a la hora de utilizar EAGLE, el software de la empresa Autodesk, que se ha utilizado tanto para desarrollar los esquemáticos como para las nuevas PCBs. Estas dificultades son debidas a que no se conocía el software con anterioridad por lo que ha sido necesario dedicar horas de estudio a aprender el manejo del mismo antes de empezar a realizar esta propuesta de actualización. También cabe destacar la falta de experiencia en algunos campos aquí abordados como puede ser el módulo ESP-WROOM-32 y los dispositivos relacionados con IoT.

Por último y para finalizar con la propuesta de actualización se ha desarrollado un presupuesto de lo que costaría la implementación de esta en el juego de espigas. Este presupuesto se puede observar en el Anexo IV.

2. Líneas futuras

Se han abierto nuevas líneas de trabajo a partir de este TFG, que son las siguientes:

- Implementación de la propuesta desarrollada en un prototipo funcional.
- Estudio del prototipo una vez construido y búsqueda de mejoras.
- Implementación de carga de software a través de OTA (*Over The Air*).
- Desarrollo de aplicación móvil específica para el envió y recibimiento de datos del juego de espigas.
- Mejora del diseño de la caja del dispositivo pudiendo a hacerse de un material más barato en vez de madera para reducir el peso y abaratar costes.

Con todos estos trabajos futuros se pretende ir perfeccionando cada vez más el juego de espigas de manera que se termine con un dispositivo casi perfecto llegando incluso a poder comercializarlo para que más público tenga acceso a él y pueda disfrutar de las ventajas que ofrece.

Capítulo VII: Bibliografía:

1. Bibliografía

[1] «PortalWeb de BricoGeek» [En línea]. Available: https://tienda.bricogeek.com/varios/906-transceptor-inalambrico-nrf24l01-24ghz.html.

[2] «PortalWeb de Wikipedia» [En línea]. Available: https://es.wikipedia.org/wiki/Registro_de_desplazamiento

[3] «PortalWeb de Mouser» [En línea]. Available: https://www.mouser.es/c/semiconductors/power-management-ics/voltage-regulators-voltage-controllers/ldo-voltage-regulators

[4] «PortalWeb de Arrow» [En línea]. Available: https://www.arrow.com/es-mx/research-and-events/articles/shedding-light-on-the-importance-of-photosensors

[5] «PortalWeb de Mouser» [En línea]. Available: https://www.mouser.es/new/microchip/atmelatmega328/

[6] «PortalWeb de Alldatasheet» [En línea]. Available: https://www.alldatasheet.com/view.jsp?Searchword=Atmega328p

[7] «PortalWeb de Electroallweb» [En línea]. Available: https://www.electroallweb.com/index.php/2020/02/12/funcionamiento-de-todos-los-pines-del-microcontrolador-atmega328p/

[8] «PortalWeb de WordPress» [En línea]. Available: https://edgardosilvi.wordpress.com/2016/02/29/acamica-ventajas-y-desventajas-de-arduino/

[9] «PortalWeb de naylampmechatronics» [En línea]. Available: https://naylampmechatronics.com/espressif-esp/382-modulo-esp-wroom-32-esp32-wifi.html

[10] «PortalWeb de RandomNerdTutorials» [En línea]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

[11] «PortalWeb de RandomNerdTutorials» [En línea]. Available: https://lastminuteengineers.com/handling-esp32-gpio-interrupts-tutorial/

[12] «PortalWeb de RandomNerdTutorials» [En línea]. Available: https://microcontrollerslab.com/esp32-external-interrupts-tutorial-arduino-ide/

[13] «PortalWeb de Espressif» [En línea]. Available: https://www.espressif.com/en/support/documents/technical-documents

[14] «PortalWeb de Eagle» [En línea]. Available: http://eagle.autodesk.com/eagle/libraries

[15] «PortalWeb de Espressif» [En línea]. Available: https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf

[16] «PortalWeb de Esp32» [En línea]. Available: https://esp32.com/viewtopic.php?t=7004

[17] «PortalWeb de Github» [En línea]. Available: https://github.com/espressif/esp-idf/issues/1096

[18] «PortalWeb de Mischianti» [En línea]. Available: https://www.mischianti.org/2021/05/30/esp32-wroom-32-esp32-s-flash-pinout-specs-and-ide-configuration-1/

[19] «PortalWeb de Aliexpress» [En línea]. Available: https://es.aliexpress.com/item/1005002795893679.html

[20] «PortalWeb de Aliexpress» [En línea]. Available: https://es.aliexpress.com/item/1005002387807888.html

[21] «PortalWeb de Youtube» [En línea]. Available: https://www.youtube.com/watch?v=V-vFtiDYiIw

[22] «PortalWeb de Slideshare» [En línea]. Available: https://es.slideshare.net/Jomicast/montaje-de-un-indicador-de-bateria

[23] «PortalWeb de Hetpro-store» [En línea]. Available: https://hetpro-store.com/images/Tutoriales/pcb_eagle/hetpro_tutorial_pcb_eagle.pdf

[24] «PortalWeb de Welivesecurity» [En línea]. Available: https://www.welivesecurity.com/la-es/2020/03/17/como-funciona-bluetooth-low-energy/

[25] «PortalWeb de Elt» [En línea]. Available: https://www.elt.es/ble-bluetooth-low-energy

[26] «PortalWeb de RandomNerdTutorials» [En línea]. Available: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

[27] «PortalWeb de Bluethooth» [En línea]. Available: https://www.bluetooth.com/specifications/assigned-numbers/

[28] «PortalWeb de RandomNerdTutorials» [En línea]. Available: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

[29] «PortalWeb de Hetpro-store» [En línea]. Available: https://hetpro-store.com/TUTORIALES/arduino-serial/

[30] «PortalWeb de Arduino» [En línea]. Available: https://forum.arduino.cc/t/sending-data-from-esp32-with-ble-in-my-phone/676015

[31] «PortalWeb de Arduino» [En línea]. Available: https://www.arduino.cc/en/Tutorial/Foundations/ShiftIn

[32] «PortalWeb de Stackoverflow» [En línea]. Available: https://stackoverflow.com/questions/58936651/how-to-send-and-receive-data-with-esp32-over-ble

[33] «PortalWeb de Arduino» [En línea]. Available: https://forum.arduino.cc/t/esp32-arduino-ble-how-to-read-hex-data/893030/5

[34] «PortalWeb de Github» [En línea]. Available: https://github.com/FastLED/FastLED/releases

[35] «PortalWeb de Uuidgenerator» [En línea]. Available: https://www.uuidgenerator.net/version1

[36] «PortalWeb de Tecnotizate» [En línea]. Available: https://tecnotizate.es/esp32-mapeo-de-pines-y-sensores-internos/

[37] «PortalWeb de Dronobotworkshop» [En línea]. Available: https://dronebotworkshop.com/interrupts/

[38] «PortalWeb de Dronobotworkshop» [En línea]. Available: https://es.aliexpress.com/item/4000959466150.html

[39] «PortalWeb de Github» [En línea]. Available: https://github.com/espressif/esp-idf/issues/1096

[40] «PortalWeb de Github» [En línea]. Available: https://github.com/poruloh/Eagle/issues

[41] «PortalWeb de Opcionis» [En línea]. Available: https://opcionis.com/blog/sueldo-medio-en-espana-2021/

Capítulo VIII: Anexos:

Anexo I: Modelo final 2D de la PCB del circuito

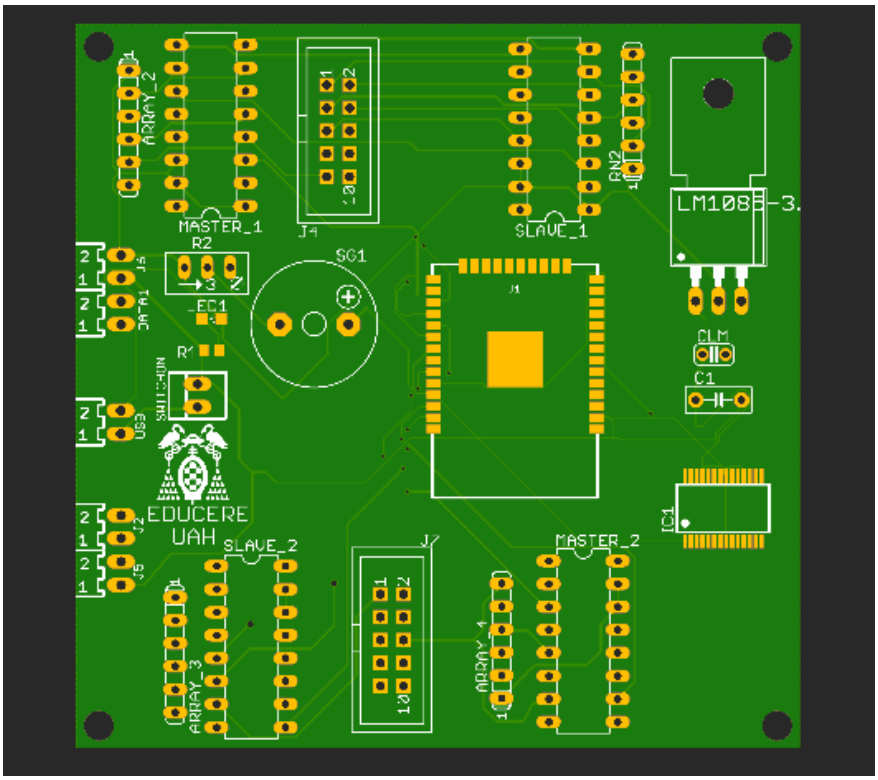


Figura 51. Modelo final 2D de la cara superior de la nueva PCB

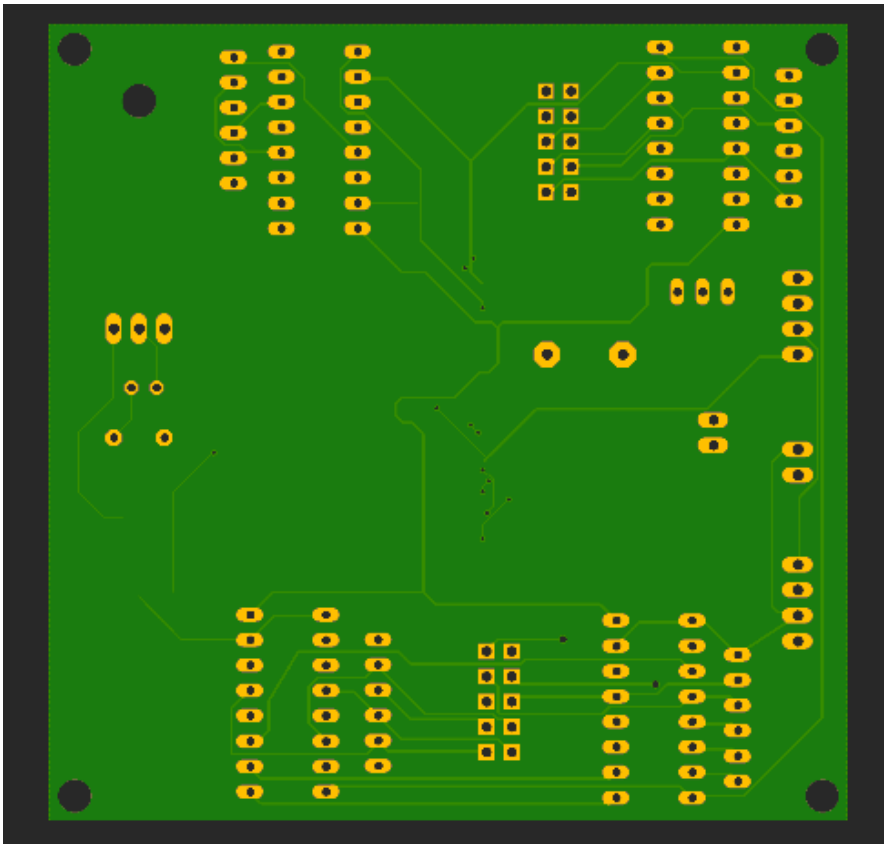


Figura 52. Modelo final 2D de la cara inferior de la nueva PCB

Anexo II: Código completo del nuevo software de comunicaciones

```
//IMPORTANTE PARA PROGRAMAR:
//Instalar Minicore en el IDE: https://github.com/MCUdude/MiniCore
//Usar Herramientas/programador: USBasp (MInicore)
// Herramientas Configuración: ES-Wroom-32, 16MHZ, BOD 2.7V, EEPROM
retained, LTO disabled, 328P/328PA, Not Bootloader
//PARA PROGRAMAR -> SUBIR USANDO PROGRAMADOR
//Nuevas espigas de 2022: con respecto a la versión anterior, se ha
quitado la inversión de los bits de la lectura de la
//fila 2 de sensores (ver en versión 9 la función
updateCylinderStatus)
//Se ha reognanizado nombres de las variables de los sensores


#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLE2902.h>
#include <SPI.h>
#include <FastLED.h>



#define COMMAND 0x00
#define START 0x0F  //start 00ff
#define STOP  0xF0 // stop 0000
#define ACK   0xFF //ack dev-ff
#define CYLINDERS_SENSOR 1
#define CYLINDERS_DEVICE 4
#define NUM_LEDS 20
#define ledsPin  5
#define Brightness 30

//definir UUID
#define SERVICE_UUID_TX        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define SERVICE_UUID_RX        "47160bda-2783-11ed-a261-0242ac120002"
#define CHARACTERISTIC_UUID_TX "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define CHARACTERISTIC_UUID_RX "4e92ce02-2783-11ed-a261-0242ac120002"

int speakerPin = 19;     // speaker  GND and pin 19

// frequencies 5 octaves
int c[5] = {131, 262, 523, 1046, 2093}; // Do
int cs[5] = {139, 277, 554, 1108, 2217}; // Do#
int d[5] = {147, 294, 587, 1175, 2349}; // Re
int ds[5] = {156, 311, 622, 1244, 2489}; // Re#
int e[5] = {165, 330, 659, 1319, 2637}; // Mi
int f[5] = {175, 349, 698, 1397, 2794}; // Fa
int fs[5] = {185, 370, 740, 1480, 2960}; // Fa#
int g[5] = {196, 392, 784, 1568, 3136}; // Sol
int gs[5] = {208, 415, 831, 1661, 3322}; // Sol#
int a[5] = {220, 440, 880, 1760, 3520}; // La
int as[5] = {233, 466, 932, 1866, 3729}; // La#
int b[5] = {247, 494, 988, 1976, 3951}; // Si


int latchPin1 = 17;
int dataPin1 = 18; //Master/slave1 in hardware - Fila 1
```

```
int clockPin1 = 2;
int latchPin2 = 16;
int dataPin2 = 21;  //Master/slave2 in hardware - Fila 2
int clockPin2 = 22;

//byte cylinderMap1A = 25; //00011001
//byte cylinderMap1B = 209; //11010001
//byte cylinderMap2A = 72;  //01001000
//byte cylinderMap2B = 159; //10011111
byte cylinderMap1A = 255; //Fila 1
byte cylinderMap1B = 255;
byte cylinderMap2A = 255; //Fila 2
byte cylinderMap2B = 255;
byte cylinderMap2Ainv = 255; //Fila 2 invertida
byte cylinderMap2Binv = 255;
byte lastCylinderMap1A = 255;
byte lastCylinderMap1B = 255;
byte lastCylinderMap2Ainv = 255;
byte lastCylinderMap2Binv = 255;
int sec = 0;
unsigned long startTime = millis();
unsigned long rxTime;


boolean sound = true;
boolean activity = false;
boolean initTimer = false;

boolean binary1A[5];
boolean binary1B[5];
boolean binary2A[5];
boolean binary2B[5];
byte bufferRX[3] = {0x55, 0x55, 0x55};

int n, i;
CRGB leds[NUM_LEDS];
CRGB colorIn = CRGB::Green;
CRGB colorOut = CRGB::Orange;


union {
  unsigned long v;
  byte b[4];
} timeStamp;

  bool deviceConnected = false;
  class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* MyServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* MyServer) {
      deviceConnected = false;
    }

void setup() {

  FastLED.addLeds<WS2812B, ledsPin, GRB>(leds, NUM_LEDS);
  FastLED.setBrightness(Brightness);
```

```
// Configurar BLE
  Serial.begin(115200);
  Serial.println("Starting BLE work!");

  BLEDevice::init("MyESP32 Juego de Espigas");
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *pService1 = pServer->createService(SERVICE_UUID_TX);
  BLEService *pService2 = pServer->createService(SERVICE_UUID_RX);
  BLECharacteristic *pCharacteristic = pService1-
>createCharacteristic(
                                         CHARACTERISTIC_UUID_TX,
                                         BLECharacteristic::PROPERTY_R
EAD |
                                         BLECharacteristic::PROPERTY_W
RITE |
                                         BLECharacteristic::PROPERTY_N
OTIFY |
                                         BLECharacteristic::PROPERTY_I
NDICATE
                                       );

  pCharacteristic->addDescriptor(new BLE2902());

  BLECharacteristic *pCharacteristic2 = pService2-
>createCharacteristic(
                                         CHARACTERISTIC_UUID_RX,
                                         BLECharacteristic::PROPERTY_W
RITE
                                       );

  pCharacteristic2->setCallbacks(new MyCharacteristicCallbacks());

  pService->start();
  pServer->getAdvertising()->start();



  //define pin modes
  pinMode(latchPin1, OUTPUT);
  pinMode(clockPin1, OUTPUT);
  pinMode(dataPin1, INPUT);
  pinMode(latchPin2, OUTPUT);
  pinMode(clockPin2, OUTPUT);
  pinMode(dataPin2, INPUT);

  for (i = 0; i <= NUM_LEDS; i++)
  {
    leds[i] = CRGB::Orange;
  }
  FastLED.show();
  delay(200);
  for (i = 0; i <= NUM_LEDS; i++)
  {
    leds[i] = CRGB::Black;
  }
  FastLED.show();
  delay(200);
  for (i = 0; i <= NUM_LEDS; i++)
  {
```

```
 leds[i] = CRGB::Green;
  }
  FastLED.show();
  //startWarsMusic();
   if (sound) {
    tone(speakerPin,  c[2]);
    delay(100);
    noTone(speakerPin);
    delay(200);
    tone(speakerPin,  c[2]);
    delay(100);
    noTone(speakerPin);
   }

}

void loop() {


// Check if the device receives from controller COMMAND + START or
STOP activity

unsigned int len = 0;

if (readData(&bufferRX[0]))
{
    if (bufferRX[0] == COMMAND && bufferRX[1] == START)
    {
      lastCylinderMap2Ainv = cylinderMap2Ainv;
      lastCylinderMap2Binv = cylinderMap2Binv;
      lastCylinderMap1A = cylinderMap1A;
      lastCylinderMap1B = cylinderMap1B;
      sound = true;
      colorIn = CRGB::Green;
      colorOut = CRGB::Orange;
      if (sound)
      {
        tone(speakerPin, g[2]);
        delay(250);
        noTone(speakerPin);
      }
      activity = true;
      initTimer = false;
      rxTime = millis();
      timeStamp.v = 0;
    }

    if (bufferRX[0] == COMMAND && bufferRX[1] == STOP)
    {
      if (sound)
      {
        tone(speakerPin,  g[2]);
        delay(500);
        noTone(speakerPin);
      }
      rxTime = millis();
      activity = false;
      sec = 0;
    }
 }
```

```
updateCylinderStatus();
if (activity)
{
  if (lastCylinderMap1A != cylinderMap1A || lastCylinderMap1B !=
cylinderMap1B|| lastCylinderMap2Ainv != cylinderMap2Ainv||
lastCylinderMap2Binv != cylinderMap2Binv)
  {
  if (!initTimer) //only first time in a activity period start time
counter and first change has taken place
  {
    initTimer = true;
    startTime = millis();
  }
    timeStamp.v = millis() - startTime;
    byte data[32];
    int c = 0;
    lastCylinderMap2Ainv = cylinderMap2Ainv;
    lastCylinderMap2Binv = cylinderMap2Binv;
    lastCylinderMap1A = cylinderMap1A;
    lastCylinderMap1B = cylinderMap1B;
    data[c++] = timeStamp.b[3];
    data[c++] = timeStamp.b[2];
    data[c++] = timeStamp.b[1];
    data[c++] = timeStamp.b[0];
    data[c++] = cylinderMap2Ainv;
    data[c++] = cylinderMap2Binv;
    data[c++] = cylinderMap1A;
    data[c++] = cylinderMap1B;
    sendData(CYLINDERS_DEVICE, CYLINDERS_SENSOR, sec++ , 8, &data[0]);
    if (((cylinderMap1A & cylinderMap1B & cylinderMap2Ainv &
cylinderMap2Binv) == 0B00011111) && sound) startWarsMusic();
}
}
}


void updateCylinderStatus(void)
{

  //********** READ THE STATUS OF CYLINDRE´S HOLES

  //Pulse the latch pin:
  //set it to 1 to collect parallel data
  digitalWrite(latchPin2, 1);
  digitalWrite(latchPin1, 1);

  //set it to 1 to collect parallel data, wait
  delayMicroseconds(20);
  //set it to 0 to transmit data serially
  digitalWrite(latchPin2, 0);
  digitalWrite(latchPin1, 0);

  cylinderMap2A = shiftIn(dataPin2, clockPin2);  //read 5 bits from
master 2, hole: 10, 9, 8, 7, 6
  cylinderMap2B = shiftIn(dataPin2, clockPin2);  //read 5 bits from
slave 2, hole: 5, 4, 3, 2, 1
  cylinderMap1A = shiftIn(dataPin1, clockPin1);  //read 5 bits from
master 1, hole: 10, 9, 8, 7, 6
  cylinderMap1B = shiftIn(dataPin1, clockPin1);  ////read 5 bits from
slave 1, hole: 5, 4, 3, 2, 1
```

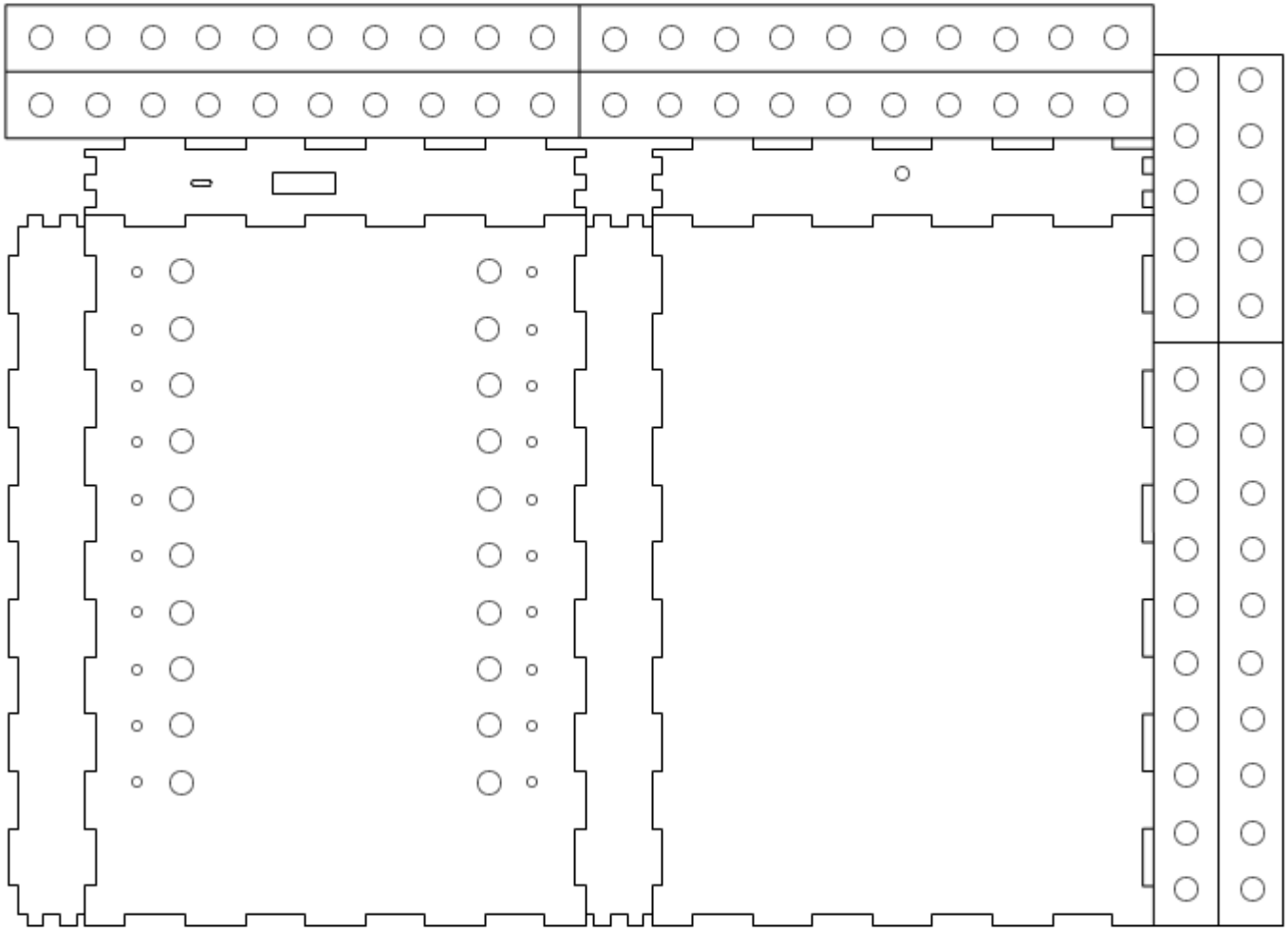
```
  //Invierte mapa de fila 2
  for (i = 0; i < 5; i++)
  {
    bitWrite(cylinderMap2Ainv, i,bitRead(cylinderMap2A, 4-i));
    bitWrite(cylinderMap2Binv, i,bitRead(cylinderMap2B, 4-i));
  }

  //************  UPDATE LEDS

  for (n = 4; n >= 0; n--)
  {
    binary2A[n] = cylinderMap2A & (1 << n);
    binary2B[n] = cylinderMap2B & (1 << n);
    binary1A[n] = cylinderMap1A & (1 << n);
    binary1B[n] = cylinderMap1B & (1 << n);

    //light leds 0..7
    if (binary2A[n] == 1) leds[n + 15] = colorIn;
    if (binary2A[n] == 0) leds[n + 15] = colorOut;
    if (binary2B[n] == 1) leds[n + 10] = colorIn;
    if (binary2B[n] == 0) leds[n + 10] = colorOut;
    if (binary1A[n] == 1) leds[n + 5] = colorIn;
    if (binary1A[n] == 0) leds[n + 5] = colorOut;
    if (binary1B[n] == 1) leds[n] = colorIn;
    if (binary1B[n] == 0) leds[n] = colorOut;
  }

  FastLED.show();

}

byte shiftIn(int dataPin, int clockPin) {
  int i;
  int inputStatus = 0;
  byte dataIn = 0;

  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, INPUT);

  for (i = 7; i >= 0; i--)
  {
    digitalWrite(clockPin, 0);
    delayMicroseconds(2);
    inputStatus = digitalRead(dataPin);
    if (inputStatus) {
      dataIn = dataIn | (1 << i);
    }
    digitalWrite(clockPin, 1);
  }
  return dataIn;
  delay (50);
}


unsigned int readData( byte *buff_rx)
{
  unsigned int response = 0;
  if (deviceConnected = true) {
    class CharacteristicCallbacks: public BLEcharacteristicCallbacks{
    void onWrite (BLEcharacterisitc *pcharacteristic2){
```

```
  byte c =  std::string rxValue = characteristic->getValue();
    response = c;
    Serial.print( buff_rx, c);
  }
  }
  }
  return response;
}

void sendData(byte device, byte sensor, byte sec , byte len, byte
*data)
{
  byte packet[32];
  byte resp[2];
  int counter = 0;
  int i = 0;
  packet[i++] = device;  //device_id
  packet[i++] = sensor;   //data_id
  packet[i++] = sec;   //data_id
  packet[i++] = len;   //data_id
  for (int j = 0; j < len; j++) {
    packet[i++] = data[j];
  }
  pcharacteristic->setValue(&packet[0]);
  pcharacteristic->notify();
  Serial.print(&packet[0], i);
}


void startWarsMusic(void)
{
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin, ds[2]); delay(500); noTone(speakerPin); delay(1);
  tone(speakerPin, as[2]); delay(125); noTone(speakerPin); delay(25);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin); delay(100);
  tone(speakerPin, ds[2]); delay(500); noTone(speakerPin); delay(1);
  tone(speakerPin, as[2]); delay(125); noTone(speakerPin); delay(25);
  tone(speakerPin,  g[2]); delay(500); noTone(speakerPin);
}
};
```

Figura 53. Código completo del Software de comunicaciones en Arduino

Anexo III: Plano 2D final de la caja del dispositivo

El plano que se muestra a continuación es el relativo a la caja entera del dispositivo visto desde la parte superior:



	Propuesta de actualización del Hardware y Software de un juego de espigas	Villarejo Crespo, Jaime
	Plano 2D final de la caja del dispositivo	2022
		Plano 1
		Grado en Ingeniería Electrónica y Automática Industrial

Anexo IV: Presupuesto

A continuación, se adjunta el presupuesto para la realización del prototipo explicado a lo largo de todo el proyecto, comentando tanto los costes de mano de obra del dispositivo, como el precio de compra de casa uno de los materiales:

- **Costes de mano de obra**

En la Tabla 5 se muestran las horas que han sido dedicadas a cada una de las partes del proyecto, así como el precio de cada hora para obtener un coste final de mano de obra. Las horas que aquí se reflejan son una estimación.

ACTIVIDAD	DURACIÓN (Horas)
Documentación y estudio del juego de espigas actual y búsqueda de soluciones	150
Actualización del Hardware	120
Actualización del Software	120
Actualización del diseño del juego de espigas	40
Elaboración del trabajo final	150
TOTAL	580

Tabla 5. Costes por actividad

Haciendo una rápida búsqueda por internet podemos saber que el sueldo medio por hora de España es de 11.95€/h [41]. Por tanto, los costes finales de mano de obra son:

HORAS	SUELDO POR HORA	TOTAL
580	11.95€/h	6931€

Tabla 6. Costes de mano de obra totales

- **Costes materiales**

A continuación, en la Tabla 7 se muestra una estimación de los costes materiales para actualizar el dispositivo, teniendo en cuenta que muchas piezas del juego de espigas se van a reutilizar.

MATERIAL	PRECIO
Ordenador Hp ProBooK	750€
ESP-WROOM-32	3.63€
USB-C	1.58€
Sensor de nivel de batería	1.58€
Conectores de tira de led	1€
Licencia anual Autodesk	1960€
Licencia Microsoft	279€
Reserva para otros materiales	50€
Impresora 3D	150€
TOTAL	3196.69€

Tabla 7. Costes por material

Conociendo los costes materiales y los costes de la mano de obra podemos realizar un presupuesto general tal y como se muestra en la Tabla 8.

TIPO DE COSTE	PRECIO
Coste mano de obra	6931€
Costes materiales	3196.69€
TOTAL	10127.69€

Tabla 8. Coste total del prototipo

Con esto tendríamos un presupuesto general para la realización del prototipo.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR