Universidad Zaragoza
1542

Trabajo Fin de Grado

Prototipo funcional de despertador basado en el análisis del sueño

Functional prototype of an alarm clock with sleep analysis

Autor

Germán Viñao Torralba

Director

José Elías Herrero Jaraba

Grado en Ingeniería Electrónica y Automática

Escuela de Ingeniería y Arquitectura de Zaragoza

Enero 2020

ÍNDICE

1 INTRODUCCIÓN

El objetivo principal de este trabajo es el desarrollo de un prototipo electrónico funcional capaz de detectar los ciclos de sueño del usuario y levantarlo en una hora que facilite su despertar, mediante sonidos y luces que creen un despertar más agradable. Otro objetivo es crear un dispositivo para interaccionar con el usuario, que muestre información en tiempo real y sea capaz de conectarse inalámbricamente a otro dispositivo externo.

La motivación principal de este proyecto es mejorar la calidad de vida del usuario, gracias a un producto final desarrollado a partir de este prototipo. Tras el estudio de los proyectos que incorporan la tecnología de análisis de sueño y después de mostrar los requisitos más importantes del proyecto, se diseñará el prototipo. Para ello se han usado herramientas informáticas de verificación electrónica para asegurar su correcto funcionamiento. A continuación, tras la creación de la placa electrónica con todos sus componentes colocados, se verificará de nuevo el funcionamiento. Para finalizar, se crearán diagramas de flujo del software como antesala a la programación del microcontrolador. Una vez completado el diseño del dispositivo y comprobado su correcto funcionamiento, se creará una aplicación como apoyo para el manejo de este.

2 ESTADO DEL ARTE

Durante la segunda mitad del siglo XX, la llegada de nuevos instrumentos electrónicos aplicados a la medicina permitieron un mejor estudio del sueño. Además, la electrofisiología y la polisomnografía se convirtieron en el estándar del análisis del sueño [1]. En 1968, Rechtschaffen y Kales publicaron el "Manual de Terminología Estandarizada, Técnicas y Sistemas de Calificación para los Estadios de Sueño en el Humano" [2] el cual se sigue usando en la actualidad y entre otras cosas, define las diferentes fases del sueño que se detallan en el apartado 3.

Hoy en día, tras el estudio acerca de la importancia del sueño para mantener una buena salud y tras el desarrollo de nueva tecnología, surgen estudios, como [9], [10] y [12], en los que se trata de detectar las fases del sueño de manera más sencilla. Estos estudios se aprovechan para crear software en dispositivos como "smartphones" y pulseras inteligentes para detectar los ciclos del sueño y aprovecharlo para despertarte en una franja horaria en la que no cueste tanto despertarse. Además, no solo hay aplicaciones tipo despertador sino muchas más que monitorean diferentes parámetros corporales como el pulso, el movimiento, etc. Que aprovechan para dar información sobre la calidad del sueño y ayudar a tener un mejor descanso.

"Sleep cycle" [3] es una aplicación para móviles que analiza el movimiento durante el sueño mediante un micrófono o un acelerómetro para detectar las fases del sueño y despertar en la primera o segunda fase (donde el sueño es más ligero). Consigue despertar sin que el usuario se sienta cansado. Otro ejemplo en esta línea es "Pillow" [4], una aplicación para iOS un tanto más compleja que la anterior ya que hace uso del acelerómetro y el pulsómetro. Es capaz de diferenciar las fases del sueño para dar una información más precisa del descanso que se ha tenido.

En la rama de las comercializaciones que no son exclusivamente software se encuentra "Phillips wake up light". Un despertador de mesilla inspirado en el amanecer natural que presta una combinación de luz y sonido para despertar de una manera menos intrusiva, este despertador está basado en estudios como [8] que explican el beneficio de la luz momentos antes de despertar.

3 ESTUDIO DEL SUEÑO

El sueño es una sucesión cíclica de las diferentes fases de aproximadamente de 90 a 120 minutos que se repite durante todo el sueño con el siguiente orden:

La primera etapa es la conocida como S0 (la etapa de adormecimiento). Comprende los primeros minutos desde que el sujeto está despierto hasta que se adormece, es una etapa de transición. La siguiente fase es la denominada SI o etapa de sueño ligero. Esta etapa puede ser fácilmente perturbada causando despertares y es la etapa de introducción al sueño más profundo en esta fase se suelen producir los famosos espasmos o sustos causados por sensaciones de caída libre. En la tercera fase SII, el sueño es más profundo y es más difícil despertar ya que la actividad cerebral y el ritmo cardiaco van disminuyendo. Es una etapa larga que comprende casi el 60% del tiempo total de sueño. La penúltima fase es la denominada SIII+SIV, también conocida como sueño profundo. Es en esta etapa donde se lleva a cabo la reparación del cuerpo, en esta fase es difícil despertar y si esto ocurre la sensación de aturdimiento es muy grande, con lo cual hay que evitar despertar en esta fase ya que un despertador convencional si sería capaz de interrumpir el sueño en este estado. La última es la llamada fase REM en la cual suceden el 90% de los sueños. Esta etapa se caracteriza por el rápido movimiento de los ojos de donde toma su nombre, en inglés “rapid eye movement”. Ocupa entre un 20 y un 30% del total del sueño y suele aparecer entre 4 y 6 veces cada noche.

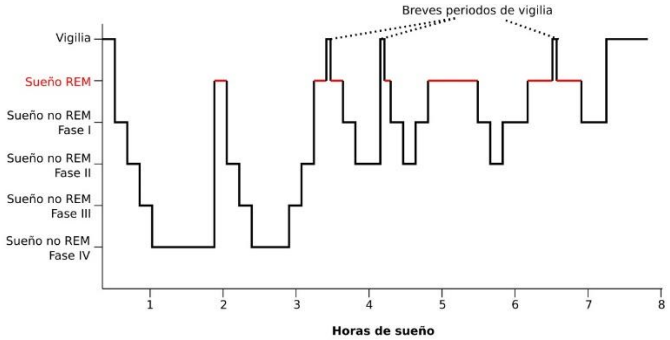


Figura 1. Alternancia entre fases durante el descanso.

El sueño suele dividirse en dos mitades de unas cuatro horas en adultos. En la primera, la mayoría de tiempo están las fases 2 y 3 con breves interrupciones de la fase 1 y de la fase REM. En la segunda mitad, es más común observar estados más cortos de la fase 3

manteniéndose la 1 y la 2 y aumentando el tiempo en fase REM. Normalmente se experimentan 5 fases REM [5]. En la figura 1 puede observarse de manera más clara la alternancia entre fases durante el sueño.

3.1 Detección de las fases

Para la correcta detección de las fases del sueño es necesario tener en cuenta diferentes variantes tales como la respiración, el pulso, las ondas cerebrales, el movimiento de los ojos y la temperatura corporal. Es lo que se conoce como polisomnografía y para llevarla a cabo es necesario el uso de un laboratorio especializado donde se conecta al sujeto con una numerosa cantidad de aparatos como electrodos y sensores. Con estos datos mediante la observación de un experto usando técnicas complejas de análisis se lleva a cabo la realización del diagrama de las fases del sueño que ha atravesado el investigado.

Como este procedimiento está al alcance de muy pocas personas, lo que se desea es reducir la complejidad y conseguir la detección de las fases de manera, aunque no perfecta, si satisfactoria. Cada fase, como se ha introducido antes, tiene una serie de características diferentes y de todas ellas, este apartado se centra en la de los movimientos del cuerpo ya que, es la forma más fácil que se tiene de conseguir información con sensores de fácil alcance (como los usados en el apartado 5).

En el artículo [6], se realiza un estudio con adultos conectados a sensores para realizarles una polisomnografía. Además, se incluye un sensor electromecánico para tener información complementaria. Una vez detectadas las fases con ayuda de las polisomnografías, se extrapola la información a los datos recibidos por el sensor electromecánico para encontrar relaciones entre ellos. Es entonces cuando se comprueba una clara relación entre el movimiento de la persona y la fase del sueño en la que se encuentra. Las etapas de sueño SI y REM son las etapas en las que más movimiento se detecta en discordancia con las etapas SII y SIII. En la figura 2 se muestra una gráfica en la que se puede ver cómo cambia el movimiento del cuerpo en función de la fase.

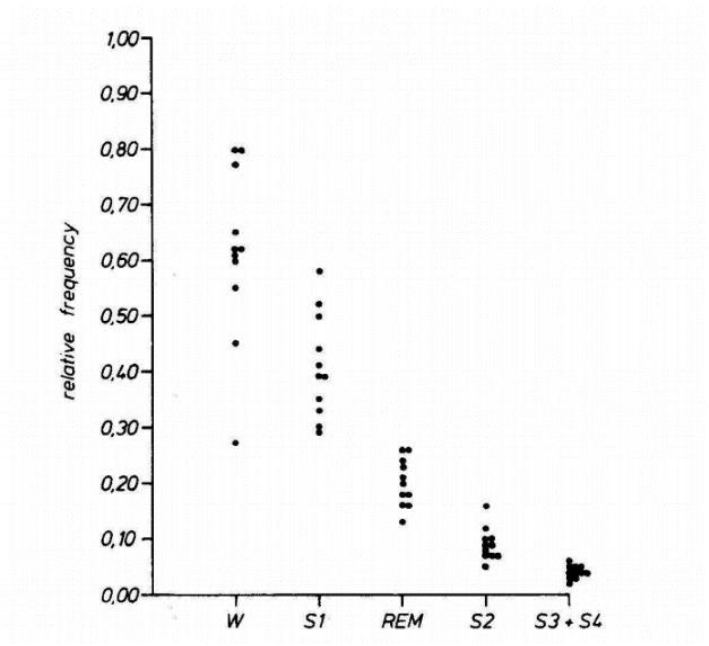


Figura 2. Movimiento del cuerpo en cada fase. Nota: Recuperado de "Rate and distribution of body movements during sleep in Humans", por J. Wilde-Frenz y H. Schulz, 1983, ResearchGate, [Fig 1], p. 277.

3.2 Mejor momento para despertar

Para conocer cuál es el mejor momento para despertarse es importante hablar del término "inercia del sueño". [7] Se trata del aturdimiento que disminuye las capacidades psicomotrices de la persona, generalmente disminuye entre los 15 y 60 minutos posteriores al despertar. Aunque no tiene una explicación todavía segura, se cree que se trata de un mecanismo para poder volver al sueño rápidamente tras haber despertado por alguna causa externa indeseada. La inercia del sueño además aumenta con factores como: la privación de sueño acumulado, la vigilia, el despertar durante horas que deberían ser de sueño y la fase del sueño en la que se encuentre el sujeto. Siendo mayor el aturdimiento experimentado al despertarse de etapas de sueño profundo que en etapas de sueño ligero.

3.3 Mecanismos para eliminar la inercia del sueño

Varios estudios, como [8], postulan que hay diferentes formas de minimizar la incidencia de la inercia del sueño. Una de ellas es la exposición a la luz antes del despertar, es decir, simulando un amanecer real. Esto aporta beneficios tanto a nivel físico como mental. El sonido es otro mecanismo que ayuda a reducir este tiempo de somnolencia. Varios artículos de investigación usan diferentes tipos de sonido con diferentes intensidades para demostrar esta hipótesis.

4 DISEÑO DEL PROTOTIPO

4.1 Requisitos

Para poder detectar las fases del sueño es necesario que este dispositivo electrónico sea capaz de detectar información del usuario por medio de sensores que puedan de forma directa o indirecta captar el movimiento. Además, dependiendo de la complejidad del algoritmo para extraer la información de esos sensores, puede ser preciso dotar al dispositivo con un ordenador o en su defecto ser capaz de conectarse de forma inalámbrica a este. Además, esta conexión nos permitirá manejar parámetros almacenados en el dispositivo electrónico. Para una mejor experiencia de usuario el dispositivo electrónico deberá incorporar una pantalla donde se muestre información. Además, se deberá hacer uso de botones físicos que también permitan su control en caso de no disponer de conexión inalámbrica. Asimismo, aprovechando la información adquirida en el apartado 3.3 serán necesario tanto la incorporación de luces como la incorporación de un sistema de reproducción de audio muestreado a frecuencia de 44,1Khz para la correcta reproducción de gran parte del espectro audible.

4.2 Posibilidades

Para la creación del dispositivo electrónico es importante especificar los componentes que se pueden usar. Para ello se crea una tabla con dispositivos que puedan ser introducidos en el proyecto debido a factores como, sus prestaciones, facilidad de adquisición y precio.

Captura de movimiento	Procesado de datos	Información	Procesador dispositivo	Reproducción de sonido	Conexión	Luces
Micrófono	MAC o Windows	Pantallas 7 segmentos	MC9S08QG8	Modulo DFPlayer	Módulo bluetooth Hc-05	Relé
Sparkfun Accelerometer	Raspberry	Pantalla TFT	ATmega328	Circuito amplificador de audio	Modulo Wifi Esp8266	Leds RGB
Sensor PIR	Móvil	Leds	Raspberry	Modulo de almacenamiento y lectura de datos		

Tabla 1. Posibles incorporaciones al prototipo

4.3 Elección de componentes

Dado que el microcontrolador es un chip muy limitado en cuanto a salidas y entradas digitales hay que tenerlo muy presente a la hora de elegir los componentes, se dará prioridad a aquellos que hagan uso de protocolos serie. Otra alternativa puede ser el uso de decodificadores para subsanar esta carencia.

Para elegir entre los diferentes módulos de captura de movimiento es importante hacer una comparación basada en los resultados arrojados tras las pruebas en el apartado 5.2, 5.3 y 5.4. En los que se llega a la conclusión que el acelerómetro es el sensor con el que se puede extraer más información ya que el PIR no es preciso al estar el sujeto cubierto

con mantas ni el micrófono tiene la calidad suficiente de ganancia de audio y eliminación de ruido para detectar sonidos producidos por el movimiento del sujeto durante la noche.

Para el procesado de datos es necesario un dispositivo con cierta capacidad de cálculo ya que los algoritmos que detectan las fases del sueño pueden ser algo complejos o pesados para procesadores poco potentes. Además, si se desea crear una aplicación de apoyo para el control del dispositivo es preciso hacerlo en un entorno que sea ampliamente usado como pueden ser ordenadores o móviles. Sin embargo, aunque no atañe a este proyecto es necesario conocer que dependiendo en que plataforma móvil se desee programar, si es Android o bien iOS el código deberá ser completamente distinto ya que se usan códigos totalmente diferentes, con lo cual la decisión final será el uso de un ordenador.

La información puede ser mostrada con cualquiera de los dispositivos mencionados en la tabla sin ningún problema. Ya que se quería trabajar con conocimientos adquiridos en asignaturas como SEP (Sistemas electrónicos programables) [23] se optó por incluir una pantalla creada artesanamente para mostrar información en ella por medio de la visualización dinámica, en la que un total de 32 leds muestran información relacionada con la hora actual, la hora programada de la alarma, el estado actual del sistema, el volumen y la indicación de si la hora mostrada es “post meridiem” o “ante meridiem”.

Para el control de los dispositivos hardware la idea principal era aprovechar el ordenador Raspberry para tener centralizado en una misma placa un procesador capaz, tanto de interactuar a nivel hardware con elementos como la pantalla y los botones, como de tener la capacidad para procesar el algoritmo de detección de fases. Finalmente, se descartó esta idea por dos razones, la primera debido a la carencia de un conversor analógico-digital para la inclusión de un micrófono, lo cual complicaba el diseño porque se debía añadir externamente un conversor que ocuparía muchos puertos de la raspberry. La segunda razón fue por la velocidad con la que se ejecuta código a nivel hardware, comparado con el *ATmega* [14] resultó mucho menor, ya que este último no dispone de un sistema operativo que maneja tareas. Así pues, se decidió usar un microcontrolador de 8 bits el “ATMEGA328” programable con el entorno de Arduino.

Para la reproducción de sonido es necesario almacenamiento y lectura de datos que en el caso de la raspberry viene incluido, pero como se comenta anteriormente queda descartada, con lo cual la alternativa que aún sigue es usar un módulo de almacenamiento y un sistema de amplificación de audio. La inclusión de estos sistemas añadía una

complejidad extra que se entiende innecesaria para el propósito final de este trabajo con lo cual, buscando una relación entre complejidad, funcionalidad y coste, se elige el módulo MP3 [19], capaz de leer y almacenar datos de una tarjeta "microSD" reproduciéndolos en un altavoz de hasta 2W de forma autónoma a nuestro procesador.

Como se detalla en el apartado 4.1 es necesaria la capacidad de comunicación inalámbrica en el dispositivo electrónico, para ello se dota con un módulo Bluetooth capaz de establecer una comunicación serie con cualquier dispositivo que disponga de esta funcionalidad.

4.4 Diseño de esquemas

A continuación, se divide el diseño en dos apartados que representan la totalidad del dispositivo electrónico que se diseña. Además de las dos placas de circuito que se observan en las figuras 3 y 5.

4.4.1 Pantalla

El diseño de la pantalla está inspirado en los relojes analógicos y no es más que una cuestión estética. Para mostrar las horas sin confusión es necesario incluir al menos 12 leds para las horas y 12 para los minutos, representando cada uno los múltiplos de 5 ya que hay en total 60 minutos cada hora. Como no tenemos tantos pines en nuestro microcontrolador es inevitable el uso de decodificadores que seleccionen una salida mediante un dato numérico de entrada. Se usan 2 decodificadores MC74HC138A [13] los cuales decodifican un código de 3 bits cada uno. Se pueden ver en la segunda imagen de la figura 15. Como la mayoría de los decodificadores permiten la puesta en paralelo de otro decodificador mediante unas entradas de selección. Así pues, se consigue la decodificación de un numero de 4 bits dando como resultado 16 salidas. De todas formas, si no se hiciera nada más se necesitarían más salidas hasta llegar a 24, el total de leds [20] para mostrar información de la hora. Para solucionarlo se deben usar dos transistores [16] que conmuten entre los leds de las horas y los minutos de forma cíclica a una velocidad tal que el ojo no pueda distinguir el parpadeo, aproximadamente 10 milisegundos, de esta forma se puede con las 16 salidas, seleccionar cualquier led de las horas activando el

primer transistor y lo mismo con los minutos con su respectivo transistor. En la figura 3 se puede observar la conexión.

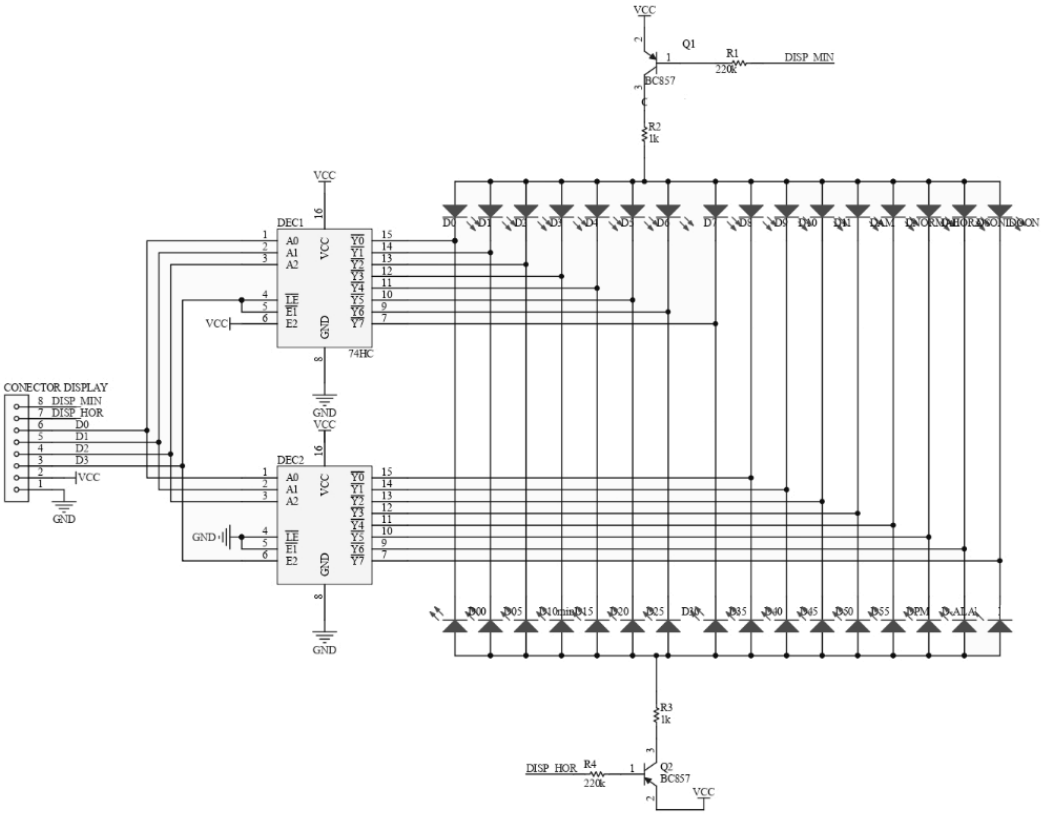


Figura 3. Esquema de la pantalla

Para comenzar se debe asegurar que ningún valor de ningún componente sobrepase los máximos expresados en el *datasheet*[1] el cual se adjunta en la tabla 2. Para la alimentación se usan los 5V del bus USB el cual está en el rango admisible de alimentación de todos los componentes usados. La corriente de salida tampoco excede los ±25mA ya que cada led se alimenta con una corriente de no más de 1 mili amperio.

[1] Documento técnico que recoge el rendimiento y especificaciones de un producto.

Symbol	Parameter	Value	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	− 0.5 to + 7.0	V
V_{in}	DC Input Voltage (Referenced to GND)	− 0.5 to V_{CC} + 0.5	V
V_{out}	DC Output Voltage (Referenced to GND)	− 0.5 to V_{CC} + 0.5	V
I_{in}	DC Input Current, per Pin	± 20	mA
I_{out}	DC Output Current, per Pin	± 25	mA
I_{CC}	DC Supply Current, V_{CC} and GND Pins	± 50	mA
P_D	Power Dissipation in Still Air, Plastic DIP† SOIC Package† TSSOP Package†	750 500 450	mW
T_{stg}	Storage Temperature	− 65 to + 150	°C
T_L	Lead Temperature, 1 mm from Case for 10 Seconds (Plastic DIP, SOIC or TSSOP Package)	260	°C

Tabla 2. Valores máximos. Nota: Recuperado de ON Semiconductor. MC74HC138A, p. 3

Los valores indicados en las tablas 3 y 4 son también importantes ya que ayudan al diseño y dan información sobre la compatibilidad con otros dispositivos. Los valores a la entrada del decodificador deben superen un cierto umbral para que sigan teniendo la misma información en otro dispositivo. Por ejemplo, si a la entrada del decodificador tenemos un circuito digital que envía estados en alto con valores de v voltios debemos validar si el decodificador entiende ese valor de voltaje como un estado en alto o en bajo. Esta información se obtiene con los símbolos V_{IH} (mínima tensión alta de entrada) V_{IL} (máxima tensión baja de entrada).

Symbol	Parameter	Test Conditions	V_{CC} V	Guaranteed Limit			Unit
				−55°C to 25°C	≤ 85°C	≤ 125°C	
V_{IH}	Minimum High−Level Input Voltage	V_{out} = 0.1 V or V_{CC} − 0.1 V $\|I_{out}\| \le$ 20 μA	2.0 3.0 4.5 6.0	1.5 2.1 3.15 4.2	1.5 2.1 3.15 4.2	1.5 2.1 3.15 4.2	V
V_{IL}	Maximum Low−Level Input Voltage	V_{out} = 0.1 V or V_{CC} − 0.1 V $\|I_{out}\| \le$ 20 μA	2.0 3.0 4.5 6.0	0.5 0.9 1.35 1.8	0.5 0.9 1.35 1.8	0.5 0.9 1.35 1.8	V
V_{OH}	Minimum High−Level Output Voltage	V_{in} = V_{IH} or V_{IL} $\|I_{out}\| \le$ 20 μA	2.0 4.5 6.0	1.9 4.4 5.9	1.9 4.4 5.9	1.9 4.4 5.9	V
		V_{in} = V_{IH} or V_{IL} $\|I_{out}\| \le$ 2.4 mA $\|I_{out}\| \le$ 4.0 mA $\|I_{out}\| \le$ 5.2 mA	3.0 4.5 6.0	2.48 3.98 5.48	2.34 3.84 5.34	2.20 3.70 5.20	
V_{OL}	Maximum Low−Level Output Voltage	V_{in} = V_{IH} or V_{IL} $\|I_{out}\| \le$ 20 μA	2.0 4.5 6.0	0.1 0.1 0.1	0.1 0.1 0.1	0.1 0.1 0.1	V
		V_{in} = V_{IH} or V_{IL} $\|I_{out}\| \le$ 2.4 mA $\|I_{out}\| \le$ 4.0 mA $\|I_{out}\| \le$ 5.2 mA	3.0 4.5 6.0	0.26 0.26 0.26	0.33 0.33 0.33	0.40 0.40 0.40	
I_{in}	Maximum Input Leakage Current	V_{in} = V_{CC} or GND	6.0	± 0.1	± 1.0	± 1.0	μA
I_{CC}	Maximum Quiescent Supply Current (per Package)	V_{in} = V_{CC} or GND I_{out} = 0 μA	6.0	4	40	160	μA

Tabla 3. Tensiones lógicas del decodificador. Nota: Recuperado de ON Semiconductor. MC74HC138A, p. 4.

Como a la entrada del decodificador está el ATmega cuyo valor alto mínimo de salida (V_{OH}) es 4.1V el cual es mayor que el valor alto mínimo de entrada (V_{IH}) del decodificador podemos asegurar que en cualquier circunstancia un valor alto del microprocesador va a ser reconocido como un valor alto por el decodificador.

$$V_{OH\text{-}MICRO} > V_{IH\text{-}DECODIFICADOR}$$

Ecuación 1. Condición para la detección correcta de niveles altos por el par microcontrolador/decodificador

Lo mismo para los valores bajos. Debemos comprobar si el valor bajo máximo de salida del microprocesador es más bajo que el valor bajo máximo de entrada del decodificador.

$$V_{OL\text{-}MICRO} < V_{IL\text{-}DECODIFICADOR}$$

Ecuación 2. Condición para la detección correcta de niveles bajos por el par microcontrolador/decodificador

Symbol	Parameter	Condition		Min.	Typ.	Max.	Units
V_{OH}	Output High Voltage[3] except Reset pin	I_{OH} = -20mA, V_{CC} = 5V	T_A=85°C	4.2			
			T_A=105°C	4.1			
		I_{OH} = -10mA, V_{CC} = 3V	T_A=85°C	2.3			
			T_A=105°C	2.1			V
V_{OL}	Output Low Voltage[4] except RESET pin	I_{OL} = 20mA, V_{CC} = 5V	T_A=85°C			0.9	
			T_A=105°C			1.0	
		I_{OL} = 10mA, V_{CC} = 3V	T_A=85°C			0.6	
			T_A=105°C			0.7	V

Tabla 4. Tensiones lógicas del ATmega. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 323

Los siguientes valores a tener en cuenta son las tensiones a la salida del decodificador para el diseño de la visualización dinámica, se detallan por medio de V_{OH} y V_{OL} en la tabla 3. El primero se tiene en cuenta para establecer los transistores PNP Q1 y Q2[2] en

[2] Q1 y Q2 hacen referencia a los transistores de la figura 3

modo corte y el segundo para calcular el valor de las resistencias de base R1 y R4[3] para una correcta saturación. Los cálculos se detallan en las ecuaciones 3 y 4:

$$R1 \leq \frac{V_{CC} - V_{CE_{SAT}} - V_{OL\_MICRO}}{I_D} * h_{FE} = \frac{5 - 0.7 - 1}{2_{mA}} * 125 = 206k\Omega$$

Ecuación 3. Cálculo de la resistencia de base de los transistores

Se elije R1=R4=200kΩ

Ahora se calcula la resistencia de alimentación de los leds teniendo en cuenta los valores de la figura 4. Y los datos del *datasheet* [16].

$$R2 = \frac{V_{CC} - V_{CE\_SAT} - V_D - V_{OL\_DECO}}{I_D} = \frac{5 - 0.7 - 2.2 - 0.1}{2_{mA}} = 1K\Omega$$ [4]

Ecuación 4. Cálculo de la resistencia de alimentación de los leds

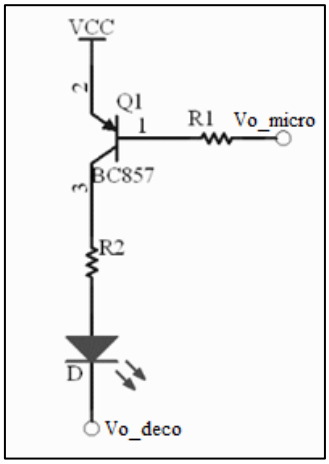


Figura 4. Detalle del circuito de alimentación de los leds

[3] R1 y R4 hacen referencia a las resistencias de la figura 3

[4] VCC, VCE_SAT, VD, VOL_DECO, ID hacen referencia a: tensión del bus, tensión de saturación entre emisor y colector, tensión directa del diodo, máximo voltaje bajo de entrada del decodificador y corriente del diodo respectivamente.

4.4.2 Circuito de control

Una vez que se conocen todos los componentes y se ha comprobado que tienen cabida en el microcontrolador se procede a asignar cada componente en su entrada o salida del ATmega. Este chip dispone de tres puertos lógicos, puerto B, puerto C y puerto D. Solo el último dispone de sus ocho salidas/entradas ya que el primero usa los bits 6 y 7 para el circuito del oscilador de cuarzo y el segundo puerto usa el bit 6 para la entrada del botón del *reset* que como en todo proyecto electrónico debe existir. En la figura 5 se puede ver el esquema diseñado para este proyecto.

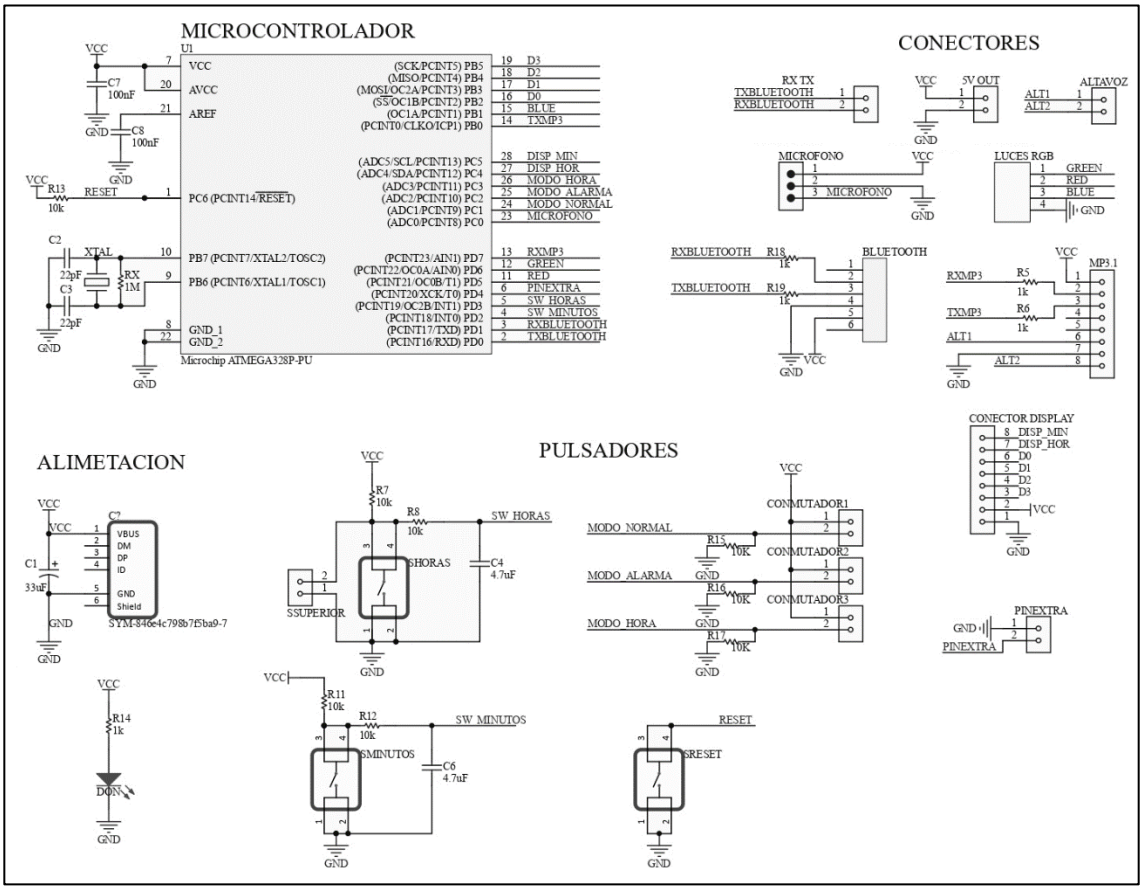


Figura 5. Esquema del circuito de control

4.4.2.1 Puerto Serie

Se empieza asignando los puertos más restrictivos que solo pueden ser usados para una tarea, en este caso el puerto serie es uno de ellos, en él se ensamblará el módulo bluetooth. Así pues, se asignan PD0 y PD1[5] que equivalen a RXD y TXD del protocolo serie a TX y RX del módulo bluetooth respectivamente.

Como indica el fabricante de los módulos MP3 [19] y Bluetooth [15] es necesaria una resistencia limitadora de corriente en los terminales RX y TX de valor 1kΩ.

4.4.2.2 Filtrado de los pulsadores

Lo siguiente de lo que hay que preocuparse es de las interrupciones hardware debido a que este microcontrolador solo posee dos, concretamente en PD2 y PD3 como se observa en la figura 11.

Puesto que la interrupción hardware se procesa instantáneamente es indeseable que el mecanismo encargado de activarla, en este caso un botón, tenga rebotes[6]. Estos generarían varias activaciones en lugar de una sola, con lo cual es indispensable un filtrado hardware o software. En este caso se opta por uno del hardware ya que se elimina peso computacional y no se incrementa el coste del prototipo.

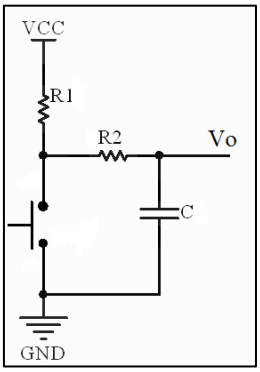


Figura 6. Esquema del filtro anti-rebotes

Como se puede observar en la figura 6 cada filtrado solo depende de tres componentes, dos resistencias y un condensador. Para cada estado del pulsador el circuito resultante es diferente, con el pulsador cerrado detallado en la Figura 7 el condensador descarga su carga por la resistencia R2 mediante una constante temporal τ_d, cuyo valor equivale a

[5] PD0 y PD1 son los bits bajos del puerto D, su conexión a los pines del microcontrolador se pueden ver en la figura 11

[6] Rebote: Alternación rápida entre niveles de tensión debido a la desconexión mecánica de un elemento sometido a corriente.

τ_d=R2*C, para la carga la constante temporal cambia ya que lo hace el circuito como se puede ver en la Figura 9, τ_c=(R1+R2)*C.

Como se quieren filtrar rebotes producidos por la conexión y desconexión de las pletinas del elemento mecánico, basta con que la constante del tiempo del filtrado sea mayor que la duración de dichos rebotes que es del orden de los milisegundos.

Se elije la constante de tiempos más pequeña, la de descarga, y se le atribuye un valor de 47 milisegundos con el que se asegura un más que suficiente filtrado de tensión, Ahora tiene libertad de elegir tanto el condensador como la resistencia R2 ya que el producto de ambos debe ser el anteriormente dicho. El resultado es R2=10kΩ y C=4,7µF.

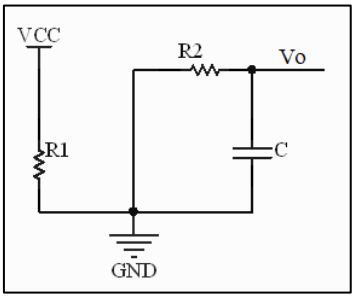


Figura 7. Filtro anti rebotes pulsador cerrado

Para la elección de R1 se tiene un compromiso entre la potencia disipada cuando el pulsador está cerrado, equivale a $P_{R1}=V_{cc}^2/R1$, y entre la contente de tiempo de carga.

Se busca un valor con el cual no se disipe mucha potencia con el pulsador cerrado, pero tampoco incremente en demasía la constante de carga del condensador, ya que haría que la detección del pulsado fuera demasiada lenta.

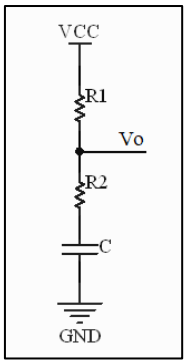


Figura 8. Filtro anti rebotes pulsador abierto

Para un valor de R1=10kΩ se obtiene una potencia, P_{R1}=2,5mW y una constante de carga de Cc= 94 milisegundos. Son valores muy adecuados ya que la perdida energética es muy pequeña, recordemos que solo se produce en el momento del pulsado. Además, el tiempo total de carga y descarga suman 141 milisegundos un valor suficiente ya que una persona no es capaz de pulsar con esa frecuencia.

Para comprobar el correcto funcionamiento se utiliza el software simulador de circuitos electrónicos [17], con el cual se obtiene la gráfica de la figura 9.

Cabe mencionar que la frecuencia a la que se podrá pulsar será mayor que la teórica mencionada anteriormente ya que el microcontrolador detectaría un nivel bajo cuando la señal fuera inferior a 0,3*VCC y detectaría un nivel alto cuando supere un nivel de 0,7*VCC siguiendo las tablas de características DC en [14].

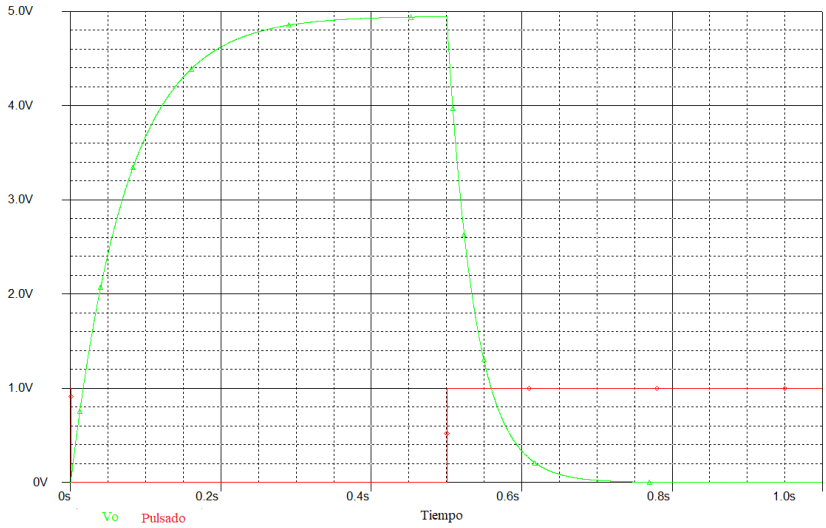


Figura 9. Simulación carga y descarga

4.4.2.3 Iluminación

Como se puede ver en la figura 10 el control de la luminosidad de cada color se produce mediante pulsos PWM que activan y desactivan un transistor [21] alimentado mediante una resistencia de base calculada en la ecuación 5 con una frecuencia suficiente para que el ojo humano no detecte este constente encendido y apagado, para ello es preciso hacer uso de un componente hardware del microcontrolador, los timers, estos se programan como se verá en el apartado 4.6.2. El ATmega solo dispone de unos pocos pines que se pueden conectar a estos timer. En la figura 11 se observa que PD5, PD6 y PB1 se pueden conectar a los timers OC0B, OC0A y OC1A respectivamente.

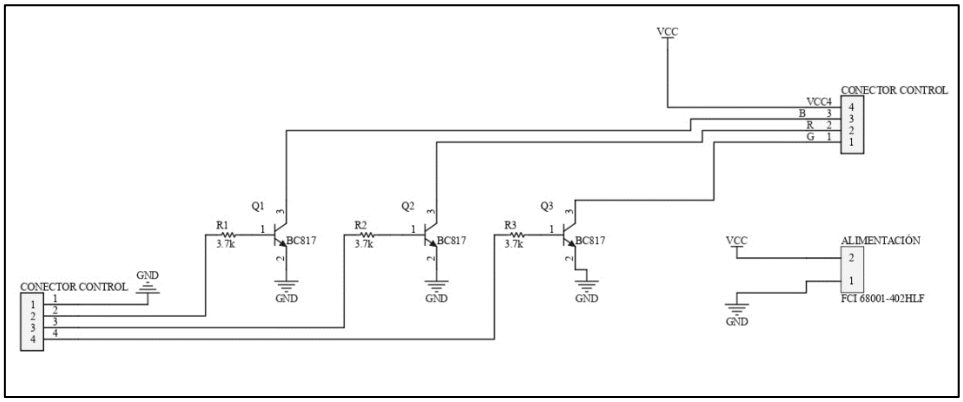


Figura 10. Esquema del circuito de encendido de leds RGB

$$R \leq \frac{V_{CC} - V_{CE_{SAT}}}{I} * h_{FE} = \frac{5 - 0.65}{100_{mA}} * 100 = 4{,}3k\Omega$$

$$\text{R} = 3{,}7\text{k}\Omega$$

Ecuación 5. Cálculo de la resistencia de base de los transistores[7]

[7] Valores mostrados en la tabla 5.
I: Corriente consumida por los leds que circula por el colector del transitor.

PARAMETER		SYMBOL	MIN.	TYP.	MAX.	UNIT
Collector-Emitter Breakdown Voltage (I_C=10mA, I_B=0)		$V_{(BR)CEO}$	45	-	-	V
Collector-Emitter Breakdown Voltage (V_{EB}=0V, I_C=10uA		$V_{(BR)CES}$	50	-	-	V
Emitter-Base Breakdown Voltage (I_E=1.0uA,I_C=0)		$V_{(BR)EBO}$	5.0	-	-	V
Emitter-Base Cutoff Current (V_{EB}=5V)		I_{EBO}	-	-	100	nA
Collector-Base Cutoff Current (V_{CB}=20V,I_E=0)	 T_J=150°C	I_{CBO}	- -	-	100 5.0	nA uA
DC Current Gain (I_C=100mA,V_{CE}=1V)	BC817-16 BC817-25 BC817-40	h_{FE}	100 160 250	- - -	250 400 600	-
(I_C=500mA,V_{CE}=1V)			40	.	-	
Collector-Emitter Saturation Voltage (I_C=500mA ,I_B=50mA)		$V_{CE(SAT)}$	-	-	0.7	V
Base-Emitte Voltage (I_C=500mA,V_{CE}=1.0V)		$V_{BE(ON)}$	-	-	1.2	V
Collector-Base Capacitance (V_{CB}=10v,I_E=0,f=1MHz)		C_{CBO}	-	5.0	-	pF
Current Gain-Bandwidth Product (I_C=10mA,V_{CE}=5V,f=100MHz)		f_T	100	-	-	MHz

Tabla 5. Nota: Recuperado de ZETEX. BC817 Datasheet

4.4.2.4 Micrófono

Para tratar la señal que llega desde el módulo es necesario usar un pin específico que recoja el valor de forma analógica, para ello se emplea el PC0 conectado al conversor analógico digital de 10 bits del que dispone el microcontrolador.

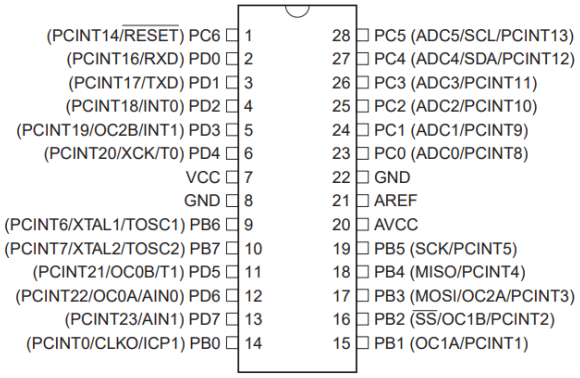


Figura 11 Esquema de pines ATmega.. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 12.

En la figura 12 se puede ver el diseño de la etapa acondicionadora del micrófono. Para realizarlo se usan esquemas propuestos por el fabricante [18]. Se trata de una etapa analógica que acopla en un nivel de continua la onda alterna generada por el micrófono

pasando esta señal por un filtro paso banda generado por los componentes, C1, R5, R6 y C4 a fin de eliminar frecuencias indeseadas, como el acoplamiento de la red. El cambio que se hace a los esquemas propuestos por el fabricante es la incorporación de un comparador de niveles ajustable mediante RD y el ajuste de la ganancia de la etapa amplificadora estableciendo R5=1.2k y R6=330k. En la figura posterior, 13, se puede observar el circuito acabado en una placa de cobre.

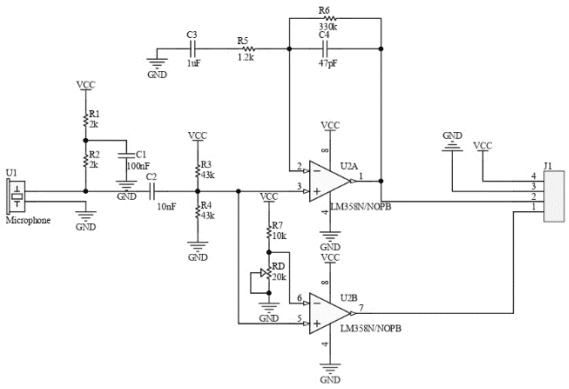


Figura 12.Esquema del amplificador del micrófono

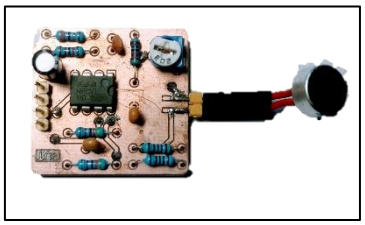


Figura 13. Placa de circuito impreso del micrófono

4.4.2.5 Cristal de cuarzo

Tal y como se define en [14] para obtener una medición de tiempos más precisa es necesario usar un cristal de cuarzo. Este se conecta, mediante el esquema representado en la figura 14, en el puerto PB6 y PB7. Atendiendo a la tabla 6 se conecta un oscilador de 16MHz con C1=C2=22pF

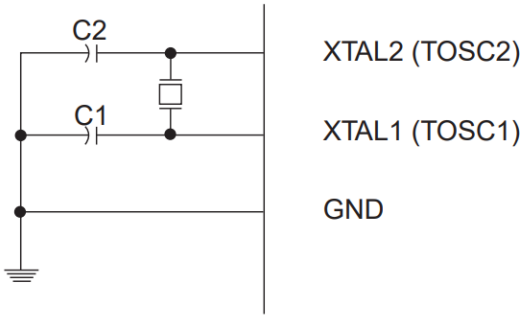


Figura 14. Conexiones del reloj de cuarzo. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 38

Frequency Range[1] (MHz)	Recommended Range for Capacitors C1 and C2 (pF)	CKSEL3...1
0.4 - 20	12 - 22	011

Tabla 6.Diseño reloj de cuarzo. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 39

Por último, para brindar un mejor funcionamiento ya que se trata de un circuito propenso a sufrir acoplamiento de ruido debido a su elevada frecuencia de funcionamiento, 16Mhz. Es necesario colocar los componentes lo más cercanos posible unos a otros y a los puertos del microcontrolador. De esta forma se consigue una mejor precisión en la medición del tiempo.

4.4.2.6 Alimentación

Se utiliza el bus USB de 5V con lo cual no es necesario el empleo de ninguna fuente extra para la conversión de niveles. Durante toda la etapa de diseño se ha tenido en cuenta para hacer uso de componentes adecuados para este valor. No obstante, dependiendo de que fuente se elija para alimentar el prototipo, ya sea un ordenador, un cargador de móvil, etc. Es preciso filtrar componentes armónicas para lograr un nivel más estable que favorezca el buen funcionamiento de la placa. Esto podría afectar particularmente a los circuitos lógicos, como los decodificadores, ya que si la señal tiene ruido podría desencadenar en la muestra de niveles bajos cuando se requieren altos y viceversa.

Como no se tiene conocimiento de que fuente se usará en el futuro el cálculo del filtrado ha de ser genérico atendiendo este a la mayoría de los casos. Se supone que es preciso

filtrar componentes armónicas tanto de la red eléctrica como de circuitos de conmutación con lo cual se usarán condensadores de desacoplo a la entrada de la alimentación del microcontrolador, pines AREF y AVCC.

Al igual que en el apartado anterior estos condensadores de desacoplo tendrán una mayor efectividad al colocarse lo más cerca posible a las conexiones del ATmega. Lo mismo con el botón de "reset" ya que el acoplamiento de ruido puede hacer que el programa dentro del microcontrolador se reinicie constantemente, para ello se sitúa lo más cerca posible del pin del ATmega como se puede ver en la primera imagen de la figura 15, situado en la parte inferior de la placa.

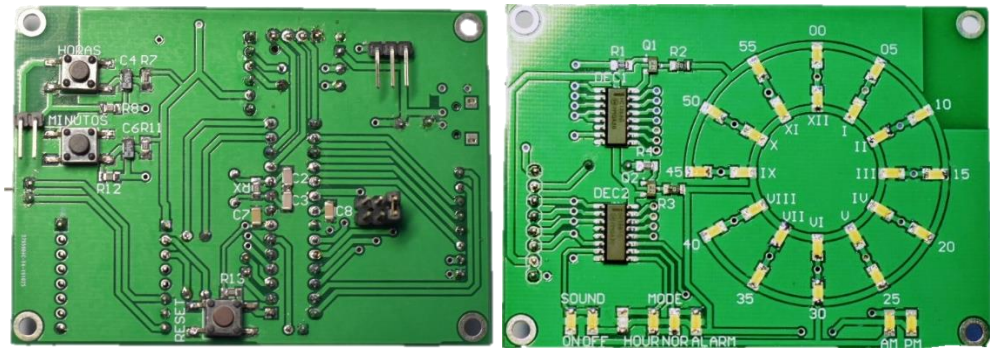


Figura 15. Placas de circuito impreso del dispositivo

4.5 Inicializaciones

Tan necesario como el diseño de los esquemas del hardware serán las inicializaciones de los registros del microcontrolador, tanto para activar componentes hardware internos como para proveer una correcta conexión con los componentes externos.

4.6.1 Interrupciones hardware

Para el correcto funcionamiento de las activaciones de las interrupciones es forzosa la correcta asignación de los bits de los registros adecuados. En este caso según el *datasheet* hay que modificar el valor de los registros "EICRA" y "EIMSK" ambos en la página 80 del dicho manual, mostrados en la tabla 7 y figura 16.

El primero de ellos sirve para establecer el modo en el que la interrupción se detecta, como puede ser con un estado en bajo de la tensión de entrada al pin, cualquier cambio lógico en esta, el flanco de bajada o el de subida de la misma. Se desea que detecte los flancos de bajada de cada una de las dos interrupciones con lo cual se tiene el siguiente valor de registro de la forma en la que se escribiría en código C:

EICRA = B00001010;

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Tabla 7. Registro de configuración de interrupciones. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 80

El siguiente registro para editar es el “EIMSK” el cual establece que interrupción esta activa o inactiva mediante el uso de un 1 o un 0 respectivamente en los bits “INT1” e “INT0”. En código C se escribiría lo siguiente:

EIMSK = B00000011;

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 16. Registro de configuración de interrupciones. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 80

Por último, faltara especificar al microcontrolador que hacer tras la activación de dichas interrupciones. Para lo cual, se define una función que es llamada por la interrupción, para ello el microcontrolador dispone de una tabla con los vectores de interrupción, véase la tabla 8. Cada vector corresponde a una dirección de memoria donde se puede almacenar un vector con la dirección de una función. Una vez activada la interrupción el puntero del programa cambiará a esta nueva dirección de memoria para ejecutar inmediatamente, en el orden de prioridad de las interrupciones, el fragmento de código.

Para ello existe una función llamada ISR (*Interrupt Service Routine*). La cual acepta como parámetro un vector de interrupción. Para ejecutar código atribuido a la interrupción INT0 se usaría el siguiente código.

```
ISR(INT0_vect){

//Código a ejecutar

}
```

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1

Tabla 8. Detalle de los vectores de interrupción. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 74

Para facilitar todo este proceso el entorno de programación de Arduino brinda una función llamada "attachInterrupt()" en la que se añaden como parámetros, el pin de la placa de Arduino al que nos vamos a conectar, la función la cual queremos que active y el modo detección del flanco de tensión.

4.6.2 Interrupciones temporales

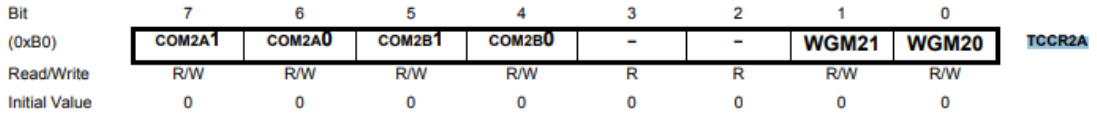
Para crear un contador de tiempos fiable es necesario el uso de los *timers*[8]. El microcontrolador en uso dispone de tres timers tal cual se expone en la página primera de [14].

Para el contaje del tiempo se usa un "timer" que no afecta a los que hacen falta en las patillas donde están conectadas las luces RGB ya que estas también harán uso de timers. Para ello a continuación se ajusta el número 2.

[8] Timers: Mecanismo Hardware que disponen la mayoría de microcontroladores, para el incremento de un contador a una frecuencia determinada que se usa para la activación de funciones.

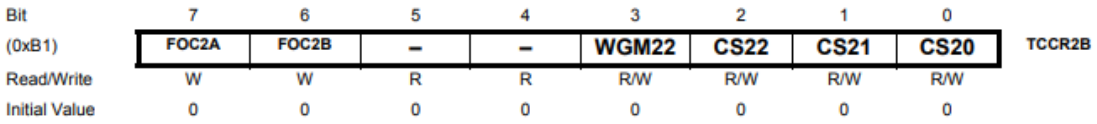
Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1][2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Tabla 9. Registro de configuración del modo de las interrupciones. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 115



Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 17.Registro de configuración de las interrupciones. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 162



Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 18. Registro de configuración de las interrupciones. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 165

Table 18-9. Clock Select Bit Description

CS22	CS21	CS20	Description
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

Tabla 10. Registro del prescaler. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 166

Siguiendo las indicaciones de la sección 18 de [14] lo primero es establecer el modo CTC para un mejor control de la frecuencia de la interrupción, para ello según la tabla 9 es necesario establecer los bits WGM22 de la figura 18 a valor cero, WGM21 de la figura 17 a valor 1 y WGM20 a valor cero.

Como se quiere una frecuencia lo más baja posible se establece el valor del "prescaler" a su máximo valor, 1024, añadiendo tres valores de uno a los bits CS22, CS21 y CS20 mostrados en la tabla 10.

Lo primero que se debe hacer es iniciar los registros de la tabla 9 y figura 17 a cero. Después, se establece a 1 el bit WGM21. Posteriormente, se ajusta el registro del valor del comparador calculado en la ecuación 6. Cabe resaltar que el "timer" que se está programando tiene una resolución de 8 bits con lo cual el valor calculado para el "output compare" no puede ser superior a 255. En este caso no lo es y no hay problema.

$$OutputCompare = \frac{f_{CLK}}{f_{PWM} * Prescaler} = \frac{16 * 10^6}{125 * 1024} - 1 = 125$$

Ecuación 6. Cálculo del valor comparador

Para finalizar se activan las interrupciones asignadas a este comparador estableciendo un uno en el bit OCIE2A.Gracias a esto se podrá ejecutar un código cada vez que el contador alcance el valor almacenado en el comparador.

En el caso de la iluminación led RGB se usan las funciones "analogWrite()" que son cedidas por el entorno de Arduino. De esta manera se facilita la creación y lectura del código.

4.6.4 Configuración de los puertos

Para configurar las entradas digitales de nuestro microcontrolador debemos hacer uso de los registros que se muestran en la figura 19. Para establecer un pin como salida basta con asignar un "1" a la dirección del registro asociado a ese pin. En el caso contrario para establecer una entrada se asigna un "0".

Mediante comparadores lógicos bit a bit: | o &, se asignan estos valores al registro para no cambiar valores ya almacenados previamente. Ejemplo en código C:

```
DDRD = DDRD | B01110001;
```

```
DDRD = DDRD & B11110001;
```

La primera línea añade unos a las posiciones 0, 4, 5 y 6 del registro sin importar que hubiera y deja en el mismo estado las demás posiciones. La segunda garantiza que en las posiciones 1, 2 y 3 haya ceros sin alterar el valor de lo que haya en otras posiciones.

Así pues, los pines PD0, PD4, PD5 y PD6 pertenecientes al pin de transmisión del protocolo serie, el pin extra de conexiones, el pin de control del color rojo y el pin de control del color verde del módulo RGB respectivamente, se configuran como salidas. El resto de los pines: PD1, PD2 y PD3 como entradas ya que pertenecen al pin de recepción del protocolo serie y a los dos botones de interrupción. Tanto PD7 como PB0 no se configuran todavía.

En el caso del puerto C se tienen tanto salidas como entradas. Los pines atribuidos a los bits PC4 y PC5 son las conexiones a los transistores que controlan la visualización dinámica de la pantalla con lo cual son salidas que se inicializan poniendo a 1 los bits requeridos del registro DDRC.

Para finalizar los bits PC0, PC1, PC2, PC3 y PC6 son la entrada del micrófono y del conmutador de estado con lo cual se debe asignar un cero a cada bit.[9]

Para facilitar la futura programación de la visualización dinámica la cual necesita 4 bits para enviar la información a la pantalla es preciso que esta información "viaje" por el mismo puerto, de esta forma se puede crear un código mucho más eficiente operado con operadores "bitwise". A fin de enviar los datos a la pantalla se usan los 4 bits más

[9] Se puede comprobar en la figura 5

significativos disponibles del puerto B, estos son los bits: PB5, PB4, PB3 y PB2. De esta forma añadiendo un numero decimal de 4 bits sobre estas posiciones podemos enviar la información a los decodificadores de la pantalla mediante las dos líneas siguientes de código.

```
PORTB = PORTB & B11000011
```

```
PORTB = PORTB | (B00000000+num<<2);
```

La primera borra el dato que hay en el bus de 4 bits anteriormente mencionado y la segunda añade el dato en la misma posición. De esta simple forma ahorramos tiempo y varias líneas de código al microprocesador.

Finalmente, las tres siguientes líneas son un ejemplo de la programación en C de los registros pertinentes para configurar salidas y entradas.

```
DDRB = DDRB | B00111110;
```

```
DDRC = DDRC | B00110000;
```

```
DDRC = DDRC & B10110000;
```

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 19. Registros de las direcciones de los puertos. Nota: Recuperado de Microchip. ATmega48A/PA/88A/PA/168A/PA/328/P, p. 100

4.6.5 Puerto serie

El microcontrolador ATmega dispone de un solo puerto serie real, es decir, por hardware y el diseño del prototipo incluye dos módulos (MP3 y Bluetooth) que se comunican de esta forma. Sin embargo, el entorno de Arduino nos permite hacer uso de librerías que mediante interrupciones temporales crean puertos serie Software, en cualquier pin. El principal problema de esto es que son significantemente más lentos como se ha podido comprobar con el uso de osciloscopio. Esto desemboca en un problema de diseño en base a pensar en una futura programación lo más rápida posible que posibilite una visualización correcta de los leds de la pantalla. Para eso la medida ideal es escoger el puerto serie hardware para el módulo bluetooth ya que su uso será más frecuente.

Para el puerto serie software se escogen los pines atribuidos a los bits PD7 y PB0, los cuales se inicializan como entrada y salida respectivamente. Bastaría pues incluir la librería en nuestro código y llamar a la función de inicialización mostradas en las siguientes líneas.

```
#include <SoftwareSerial.h>

SoftwareSerial MP3Serial(PB0,PD7);
```

4.6.6 Watchdog

En todo proyecto electrónico debe haber un control de la estabilidad del programa y "una opción b" en el caso de que puedan haber fallos que colapsen el programa. Casi todos los microcontroladores tienen una opción que se llama *watchdog*. Se trata de una funcionalidad que sirve para dar respuesta a cuelgues inesperados. Tras uno de estos y dependiendo de cómo se ha programado puede interrumpir el código y llamar a una función que guarde el estado actual y posteriormente reinicie el programa. Primero sería necesario inicializarlo fuera del bucle para posteriormente dentro de este ir reiniciando el contador que usa para activar el *flag* de *reset*.

En la página 60 [14] se exponen códigos para el uso y configuración de esta funcionalidad, aun así, se usa una librería de código abierto llamada "wdt" mucho más fácil de programar.

4.6 Diagramas de flujo

En este apartado se muestran diferentes diagramas de flujo de las tareas más importantes de control del dispositivo electrónico, la similitud entre todas ellas reside en que son funciones que han sido fraccionadas para que no tengan un tiempo de cómputo excesivo, En vez que ejecutar un bucle dentro de cada función para procesar toda la información lo que se hace es que cada vez que se llame desde el bucle del ejecutivo cíclico estas ejecutan una parte de código distinta cada vez.

4.7.1 Visualización dinámica

Cada vez que se llama a esta función ejecuta un código diferente que prepara el hardware para mostrar diferente información. En el siguiente orden: la hora, el minuto, “post” o “ante meridiem”, el modo de ajuste y el nivel de sonido. Con cada llamada se enciende un único led, al elevarse la frecuencia el ojo confunde que led está activo y los ve los 5 a la vez.

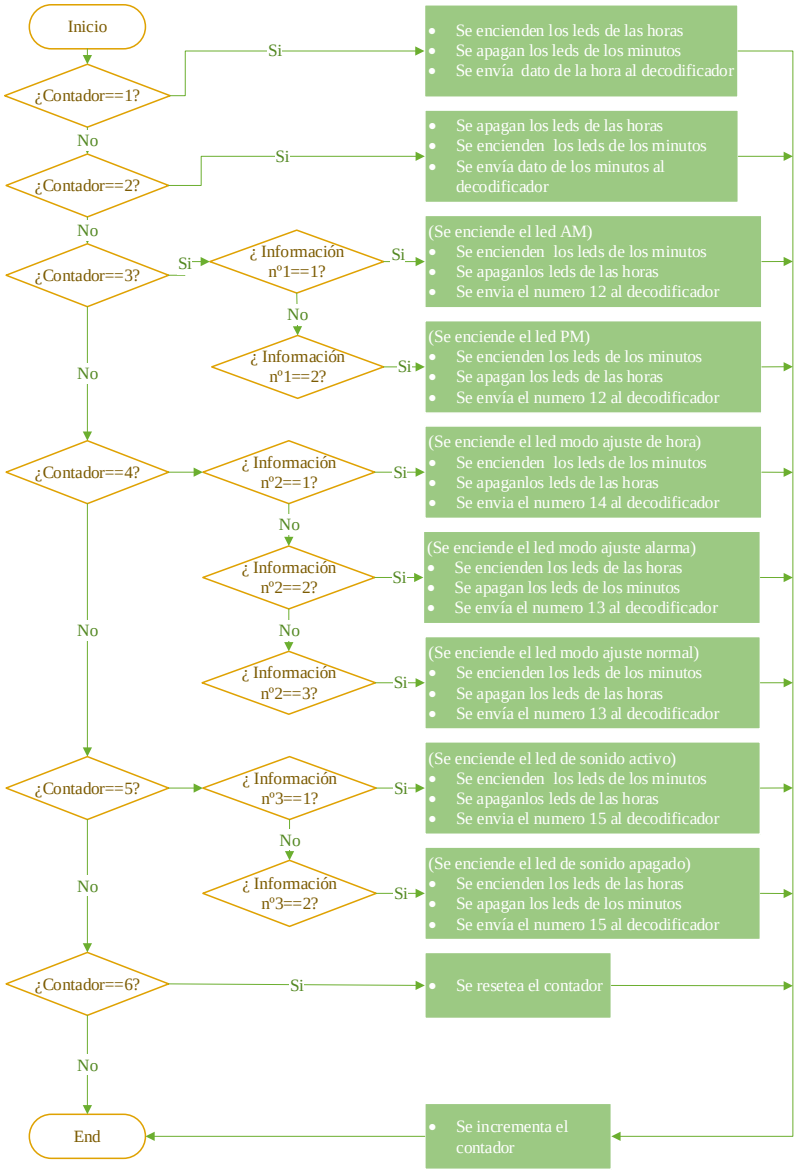


Figura 20. Visualización dinámica

4.7.2 Iluminación led

Mediante los timers mencionados en el apartado 4.6.2 se crean ondas PWM para cada canal de color de los leds RGB.

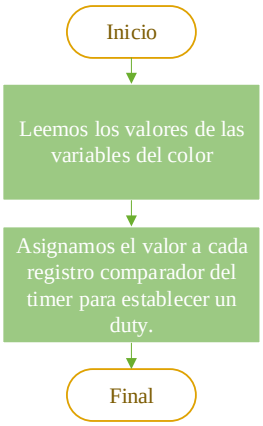


Figura 21.Diagrama de flujo de luces led

La dependencia de la luminosidad de los leds varia de forma no lineal con la tensión con lo cual, se puede pensar que hasta no alcanzar un valor elevado en la señal de salida los leds no lucirán. Esto hubiera sido un problema si se hubiese usado una salida analógica de tensión, pero en el caso del PWM la regulación de intensidad en el brillo se produce por la relación entre tiempo en alto y tiempo en bajo de la salida digital. Como en la visualización dinámica esto es un efecto óptico.

4.7.3 Comunicación con el módulo MP3

El módulo MP3 se comunica con el microcontrolador por medio de comandos serie que activan diferentes funcionalidades del módulo. Cada comando es una sucesión de 10 bytes incluyendo códigos de inicio de cadena, suma de verificación, final de cadena y datos. Para poder ejecutar en tiempo real se divide la función que envía los comandos para enviar solo un byte cada vez que se llama. De esta forma se consigue un cómputo mucho menor que ayudará en el ejecutivo cíclico. Para crear una abstracción mayor a la hora de llamar a esta función se crea una variable llamada "función" que ayuda para saber que comando se debe enviar.

A modo de ejemplo. Si la variable función almacena el valor 1, se enviará por serie el comando "parar la reproducción". Eso sí, cada vez que se llama a la función solo se envía un byte de dicho comando. Una vez que se envían los 10 bytes ya se puede cambiar el valor almacenado en la variable función.

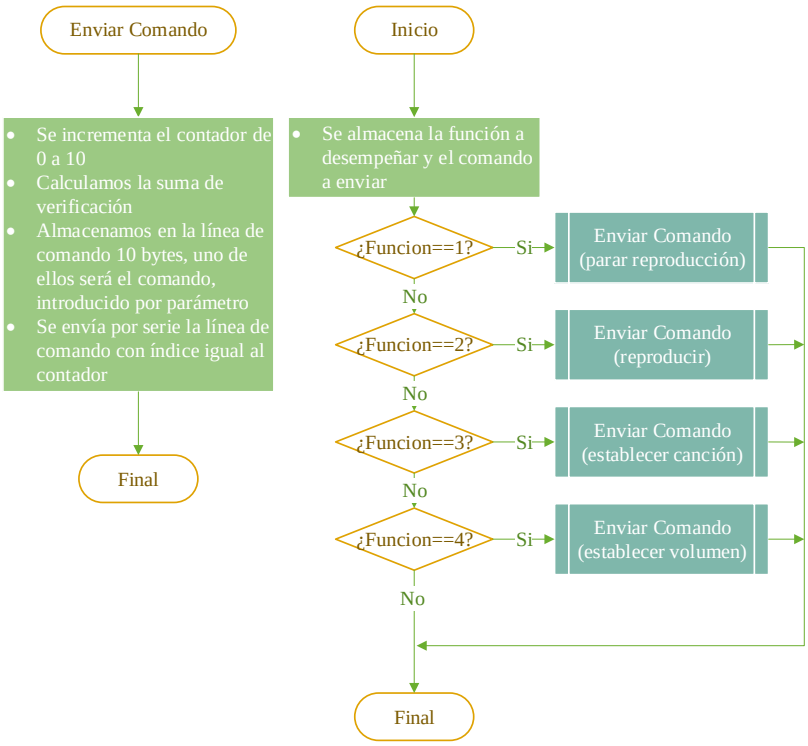


Figura 22. Diagrama de flujo de la función MP3

4.7.4 Envío de información

La siguiente función hace uso del puerto serie hardware del micrófono con lo cual no interfiere con el protocolo serie Software de la función del MP3, también cabe destacar que la información que se envía es el estado de la máquina de estados, usado por la aplicación de ordenador y el valor de 10 bits del módulo analógico conectado.

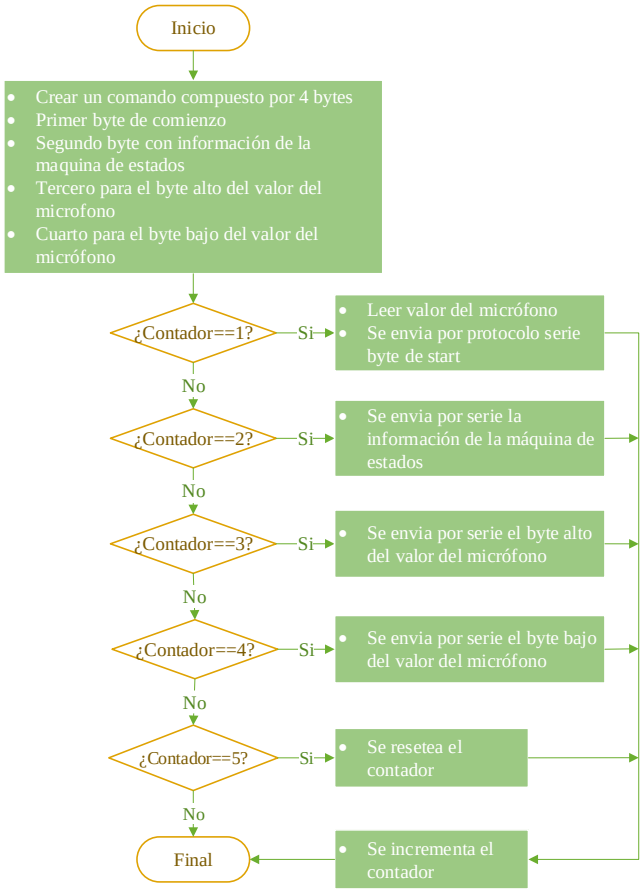


Figura 23.Diagrama de flujo de envió de información

4.7.5 Ajuste de la hora

La siguiente función hace el mismo proceso tanto en la hora de la alarma como en la hora actual. Sirve para actualizar correctamente el valor de estas horas cuando son incrementadas externamente por el uso de los botones o de la interrupción temporal. Sin esta función, el reloj no mostraría correctamente la hora ya que pasados 60 minutos no se incrementarían las horas. Además, esta función tiene una misión fundamental que es el quitar peso computacional en las interrupciones, para que estas no se demoren demasiado en acabar la tarea.

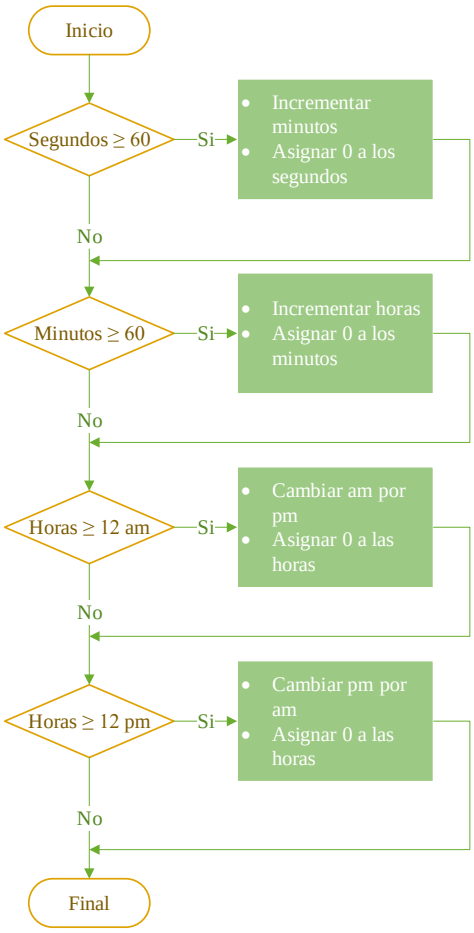


Figura 24. Diagrama de flujo del ajuste de hora

4.7.6 Leer entradas

Para la correcta comunicación entre dispositivos es necesario programar una función que lea periódicamente el puerto serie. Este tiene un "buffer" de 64 bytes con lo cual hay que tener presente que si no se lee de forma periódica puede haber datos que se pierdan ya que el buffer se llenaría. Para ello se tiene en cuenta la velocidad a la que le llegan esos datos que tal como está configurado son 9600bps lo que equivale a 1200 bytes por segundo. Haciendo un cálculo sencillo se llega a que en 53milisegundos el buffer de 64 bytes se llena. Con lo cual hay que llamar a la función a una frecuencia mayor para asegurar no perder ningún dato.

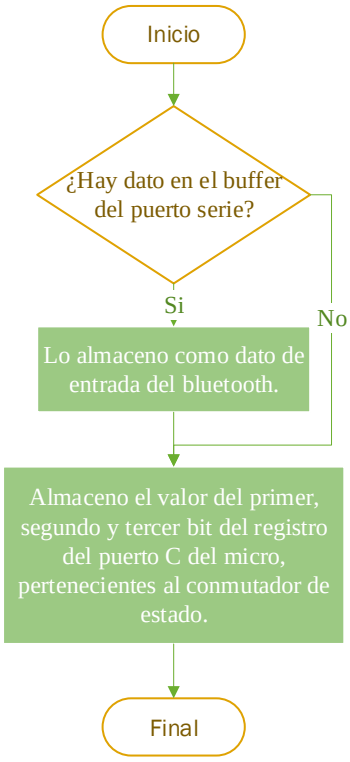


Figura 25. Diagrama de flujo de la función leer entradas.

4.7.6 Botones

Se trata de una función activada mediante una interrupción hardware por lo cual se intenta que el código a realizar sea lo menos pesado posible.

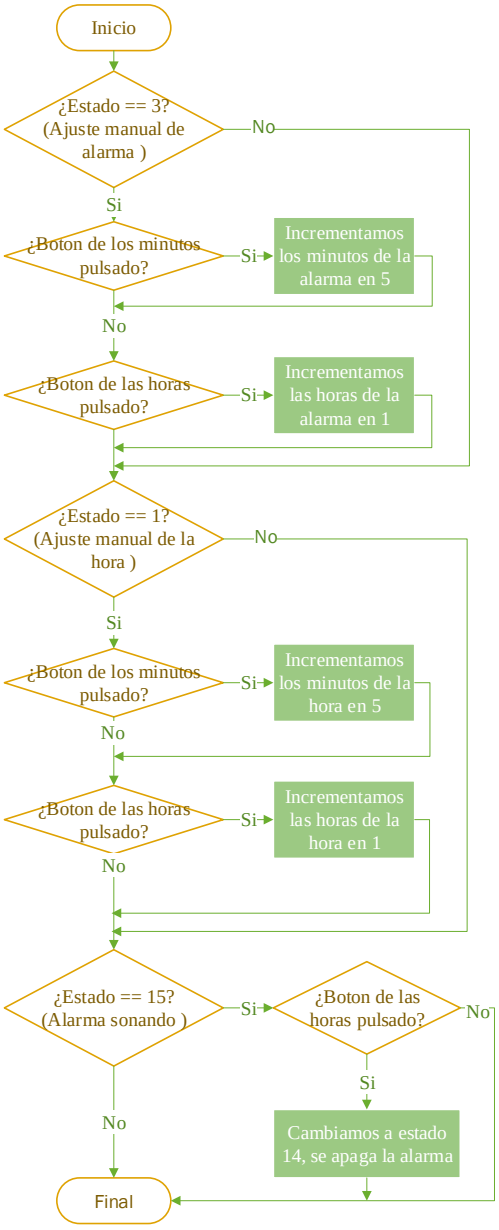


Figura 26. Diagrama de flujo de la interrupcion de los botones

4.7 Máquina de estados

La máquina de estados o máquina de estados finitos representa una forma rudimentaria pero eficaz para la ejecución y el orden de tareas con una cierta complejidad. Lo que se consigue es obtener una respuesta ante una entrada determinada. Se usa la maquina Mealy.

Las entradas de la máquina de estados pueden venir del dato leído proveniente del módulo bluetooth, del estado del conmutador, de la pulsación de los botones o de un flag dedicado al estado de la alarma. Las entradas se leen cada vez que se llama a la función donde se programa la máquina de estados a excepción de los botones, que activan una interrupción la cual mediante un breve código es capaz de interactuar con la máquina de estados.

Para la explicación de la máquina se detalla un pequeño ejemplo: Si está en el estado E13 (Ajuste del volumen) y recibe por bluetooth una entrada con valor 3 la máquina de estados asignará un volumen de 15, si en ese mismo estado se recibe una entrada con valor 20 el estado cambiará a E8 (Ajuste del color).

[10]

		ESTADOS BLUETOOTH OFF			ESTADOS BLUETOOTH ON										ESTADO ALARMA	
		AJUSTE HORA	AJUSTE NORM.	AJUSTE ALARM.	AJUSTE HORA	AJUSTE NORM.	AJUSTE ALARM.	AJUSTE TONO	COLOR	HORA ACT.	MIN. ACT.	HORA ALARM.	MIN. ALARM.	VOLUMEN	APAGAR BLUET.	LUZ ON	SONIDO ON
		E1	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**	**E8**	**E9**	**E10**	**E11**	**E12**	**E13**	**E14**	**E15**	**E16**
ENTRADA BLUETOOTH	1				E9		E11	TONO T1	L1	1	5	1	5	5			
	2				E10		E12	TONO T2	L2	2	10	2	10	10			
	3							TONO T3	L3	3	15	3	15	15			
	4							TONO T4	L4	4	20	4	20	20			
	5									5	25	5	25	25			
	6									6	30	6	30	30			
	7									7	35	7	35				
	8									8	40	8	40				
	9									9	45	9	45				
	10									10	50	10	50				
	11									11	55	11	55				
	12									12	00	12	00				
	13									AM	AM	AM	AM				
	14									PM	PM	PM	PM				

Tabla 11. Máquina de estados, primera parte.

[10] ROJO →Salidas NEGRO →Cambios de estado

		ESTADOS MANUALES			ESTADOS BLUETOOTH											ESTADO ALARMA	
		AJUSTE HORA	AJUSTE NORM.	AJUSTE ALARM.	AJUSTE HORA	AJUSTE NORM.	AJUSTE ALARM.	AJUSTE TONO	COLOR	HORA ACT.	MIN. ACT.	HORA ALARM.	MIN. ALARM.	VOLUMEN	APAGAR BLUET.	LUZ ON	SONIDO ON
		E1	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**	**E8**	**E9**	**E10**	**E11**	**E12**	**E13**	**E14**	**E15**	**E16**
ENTRADA BLUETOOTH	15									E10		E12					
	16										E9		E11				
	17	E4	E4	E4		E4	E4	E4	E4	E4	E4	E4	E4	E4			
	18	E5	E5	E5	E5		E5	E5	E5	E5	E5	E5	E5	E5			
	19	E6	E6	E6	E6	E6		E6	E6	E6	E6	E6	E6	E6			
	20	E7	E7	E7	E7	E7	E7		E7	E7	E7	E7	E7	E7			
	21	E8	E8	E8	E8	E8	E8	E8		E8	E8	E8	E8	E8			
	22	E13	E13	E13	E13	E13	E13	E13	E13	E13	E13	E13	E13				
	23	E14	E14	E14	E14	E14	E14	E14	E14	E14	E14	E14	E14	E14		E14	
	24 (IA)															E16	
	POR DEFECTO														TODO OFF	LEDS ON	E15 MUSIC. ON
CONMUTADOR	1		E1	E1											E1		
	2	E2		E2											E2		
	3	E3	E3												E3		
BOTON F.S	HORA	HORA ACT ++		HORA ALARM ++												E14	
	MIN.	MIN ACT ++		MIN ALARM++													
FLAG ALARM	1		E15			E15											
	2															E16	

Tabla 12. Máquina de estados, segunda parte.

4.8 Ejecutivo Cíclico

Para que el microprocesador monoprocesador pueda realizar todas las tareas de forma ordenada es preciso crear un ejecutivo cíclico que entrelace varias tareas periódicas durante un periodo de tiempo de tal forma que su ejecución cíclica garantice el cumplimiento de los plazos de estas, de esta forma se crea un paralelismo virtual.

4.8.1 Diseño temporal

A continuación, se detallan los criterios a la hora de elegir el periodo de cada función usada en el ejecutivo cíclico.

4.8.1.1 Pantalla con visualización dinámica

Para la elección del periodo hay que conocer la frecuencia a la cual el ojo deja de observar un parpadeo, que puede incluso llegar a ser molesto, esta frecuencia está en torno a 50 y 60 Hz, aunque se podría usar una de estas se usara una de 100Hz ya que así será menos notorio el parpadeo.

$$P = \frac{1}{f} = \frac{1}{100\ Hz} = 2\ ms$$

Ecuación 7. Periodo para la visualización dinámica

4.8.1.2 Conexión con el módulo de MP3

Debido a la escasa información anexada en el "datasheet" de este módulo el periodo tuvo que ser calculado mediante experimentación. Con un periodo comprendido entre 1 milisegundo y 5 milisegundos el módulo era capaz de recibir la información de forma exitosa en el 100% de las pruebas.

Para simplificar los cálculos en el ejecutivo cíclico se establece un periodo del doble de duración que para la pantalla, 4 milisegundos.

4.8.3 Funciones usadas cíclicamente:

Para calcular el periodo de la función “pantallaDinámica()” se debe tener en cuenta que el cálculo en la ecuación 7. Por eso se establece un tiempo de 2 milisegundos. La función “progresivo()” requiere un tiempo con el cual no se note diferencia del cambio entre colores, como este cambio se realiza de manera muy progresiva, 255 valores como máximo en 20 minutos, se establecen 100 milisegundos como periodo de la función con el cual cambiarán de color los leds. El módulo de MP3 requiere un tiempo entre bytes recibidos de 4 milisegundos con lo cual ese será el periodo de la función que los envía. La función “enviarInfo()” envía los datos recogidos por el sensor a la cuarta vez que se le llama, si queremos muestrear a 125Hz necesitamos que el periodo de esta función sea $\frac{1}{125*4}$ segundos, es decir 2 milisegundos. Por último el periodo de la función “estados()” debe ser 10 veces superior a enviar MP3() de esta forma se envían los 10 bytes que necesita el módulo MP3.

Código	Nombre	Dependencia	Computo C (µs)	Periodo P (µs)	Plazo D (ms)
F1	onMin()	-		-	-
F2	offMin()	-			
F3	onHor()	-			
F4	offHor()	-			
F5	onDisplay()	-			
F6	offDisplay()	F2,F4			
F7	numDisplay()	-			
F8	**pantallaDinamica()**	F1,F2,F3,F4,F7	25	2000	1600
F9	setRGB()	-			
F10	onRGB()	-	40		
F11	offRGB()	-	40		
F12	progresivo()	F10	80	100000	
F13	pause()	F17			
F14	play()	F17			
F15	setVolume()	F17			
F16	file()	F17			
F17	ejecutar_CMD()	-	1500	1000-5000	
F18	**enviarMp3()**	F13,F14,F15,F16	1510	4000	3200
F19	leerEntrada()	-	15		
F20	**enviarInfo()**		200	2000	2000
F21	**laHora()**	-	10	40000	32000
F22	alarma()	-			
F23	diferenciaHoras()	-	5		
F24	**estados()**	F10,F11,F12,F19,F22	200	40000	32000

Tabla 13. Tiempos de las funciones del ejecutivo cíclico

Código	Nombre	Dependencia	Cómputo (µs)	Periodo (ms)
IN1	botones()		20	100 [1]
IN2	TIMER2_COMPA_vect		20	1000

Tabla 14. Tiempos de las interrupciones

4.8.4 Cálculos

Para simplificar el diseño el plan principal de planificación es dividido en uno o más planes secundarios llamados marcos (m).

Tamaño de hiper-periodo:

$$M = m.c.m(Pi) = 40000\ \mu s$$

$$M = K \cdot m$$

Tamaño de marco:

$$m \leq \min(Di) = 2000\ \mu s$$

$$m \geq \max(Ci) = 1510\ \mu s$$

Se elije: $m = 2000\ \mu s$

Comprobamos el cumplimiento de plazos:

$$m + (m - m.c.d(m, Pi)) \leq Di \quad [11]$$

Ecuación 8. Ecuaciones para el tamaño del ejecutivo

Se comprueba que todas las tareas cumplen la condición es decir cumplen plazos y pueden ser planificadas.

Cabe destacar que inicialmente la función enviarMP3() tenía un cómputo de 15.1 milisegundos ya que se enviaban los 10 bytes seguidamente uno tras otro en un bucle, al crear la planificación esta función no cabía en ningún marco por lo que no quedó opción para dividirla en 10 funciones iguales que ejecutaran una parte de código diferente en cada llamada.

[11] Se utiliza i como subíndice de la tarea, siendo Pi y Di el periodo y el plazo de la tarea i respectivamente. m.c.d(a, b) hace referencia al máximo común divisor.

Además, los tiempos de cómputo se midieron todos con osciloscopio usando una salida digital, que se ponía en estado alto y bajo cuando empezaba y terminaba la tarea.

4.8.5 Planificación:

En el siguiente gráfico podemos ver las activaciones y plazos de respuesta de cada tarea en una línea temporal. Se observa como el periodo está dividido en 20 marcos de 2000 µs de duración en los que sendas tareas se activan después de cada activación y terminan de ejecutar antes de cada plazo. Con lo cual la planificación queda concluida. En la figura X se puede observar de forma gráfica mejor el funcionamiento.

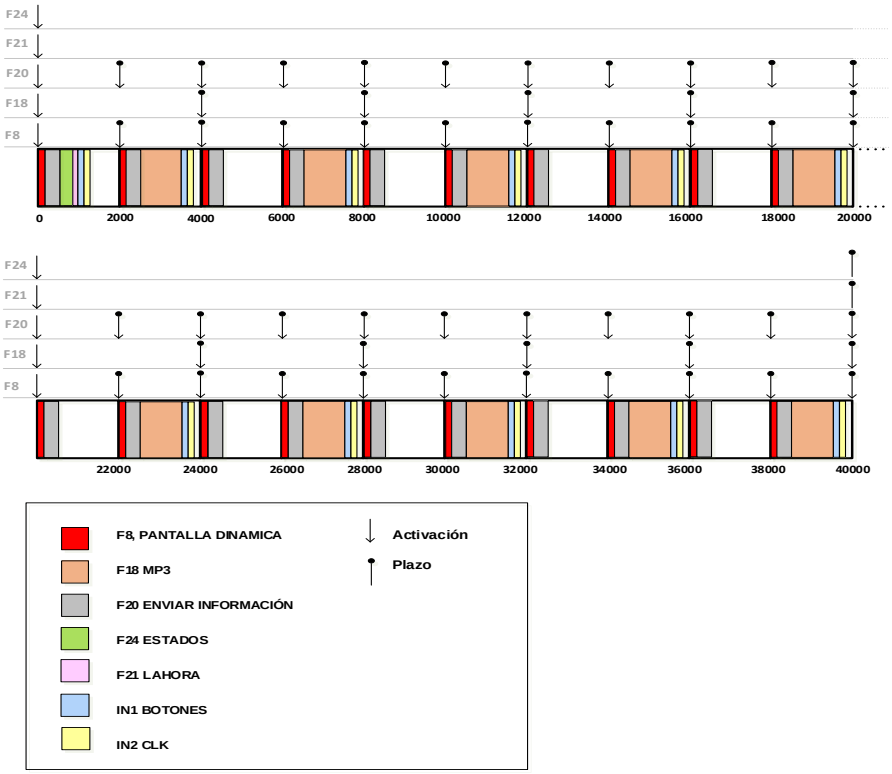


Figura 27. Planificación de las tareas

5 OBTENCIÓN DE DATOS

Una vez recogidos datos de varias noches, se intenta extraer información de varios sensores para posteriormente, decir cuál es más útil. Para ello disponemos de tres tipos de sensores para medir movimientos de forma directa o indirecta.

5.1 Aplicación

Mediante código Python y con ayuda de la librería “tkinter”, se diseña una aplicación de ordenador capaz de comunicarse con el dispositivo electrónico vía comunicación bluetooth. Gracias a esto se puede controlar la placa ajustando la información en ella guardada (la hora de encendido de la alarma, el color con el que se encienden los leds, el tono que suena y el volumen de este). Además, se puede recibir la información de los sensores conectados a ella, así como de los tres que se detallan en el apartado 5.2.

Para la conexión entre la aplicación y la placa se utiliza un protocolo bluetooth para enviar datos a través de los puertos serie de ambos. Por lo cual es preciso el uso de un ordenador que disponga de esta conexión. En el caso de la placa, como se detalla en el apartado 4.3, se usa un módulo que recibe y envía información en forma de códigos de tamaño byte. Estos códigos como se ve en el apartado 4.7 son capaces de cambiar el estado en el que se encuentra el dispositivo.

5.2 Sensor PIR

Se trata de un sensor para medir movimiento de forma directa sin contacto. Es capaz de detectar la radiación infrarroja que emiten los objetos mediante un sensor dividido en dos zonas en las que en condiciones normales de no movimiento reciben la misma radiación y se encuentran en equilibrio, tras el paso de un objeto emisor de luz infrarroja el equilibrio desaparece y se dispara.

Debido al diseño hardware con el que el sensor viene incluido, entrega una señal de tensión alta cuando detecta movimiento durante cinco segundos. Transcurrido este tiempo proporciona una señal de tensión baja si no vuelve a detectar movimiento.

Este aspecto obliga a diseñar un algoritmo de muy baja frecuencia ya que no se puede obtener información con más frecuencia. Por ello se crean intervalos de 1 minuto en los

que se realiza un recuento de cuantas veces se ha disparado en dicho intervalo. De esta forma, se sigue sucesivamente hasta completar la noche entera.

El aspecto de estos datos expuestos en una gráfica temporal se puede observar en la figura 28. Además, se puede observar como las detecciones de movimiento siguen un patrón similar en los tres casos. Por ello se puede intuir que se puede extraer información de la fase del sueño.

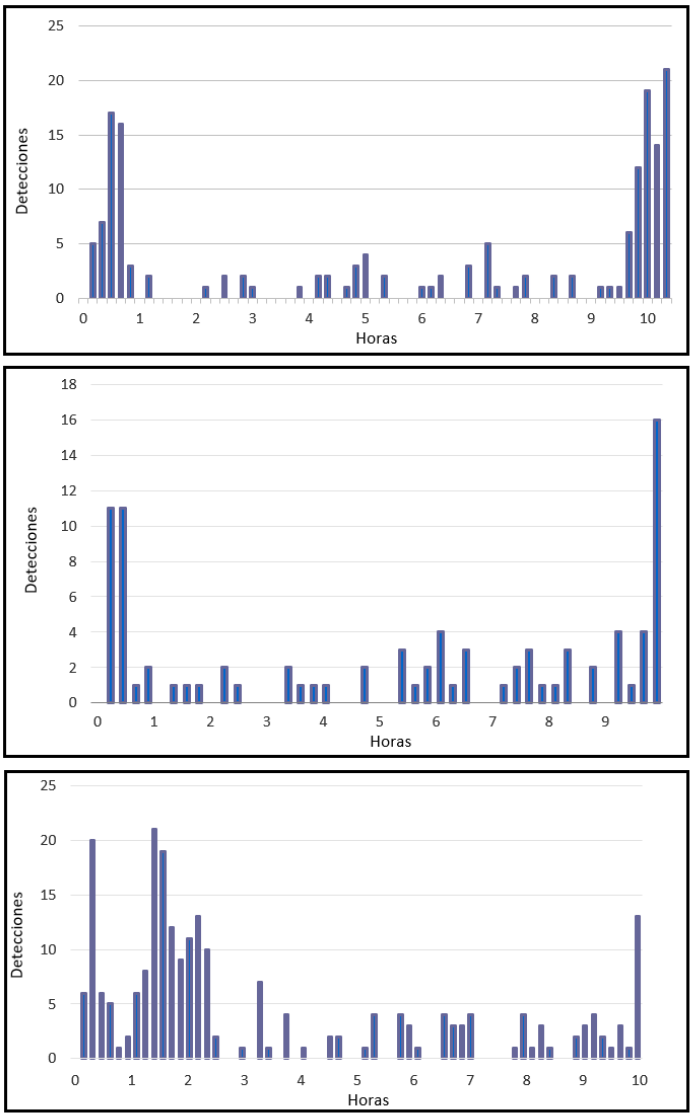


Figura 28. Muestra de los datos recogidos con el sensor PIR en tres noches diferentes.

5.3 Micrófono

Gracias a este dispositivo se puede extraer información a una frecuencia mayor. Se lleva a cabo la extracción de datos a frecuencias de cien hercios resultando en los ejemplos de las figuras 29 y 30.

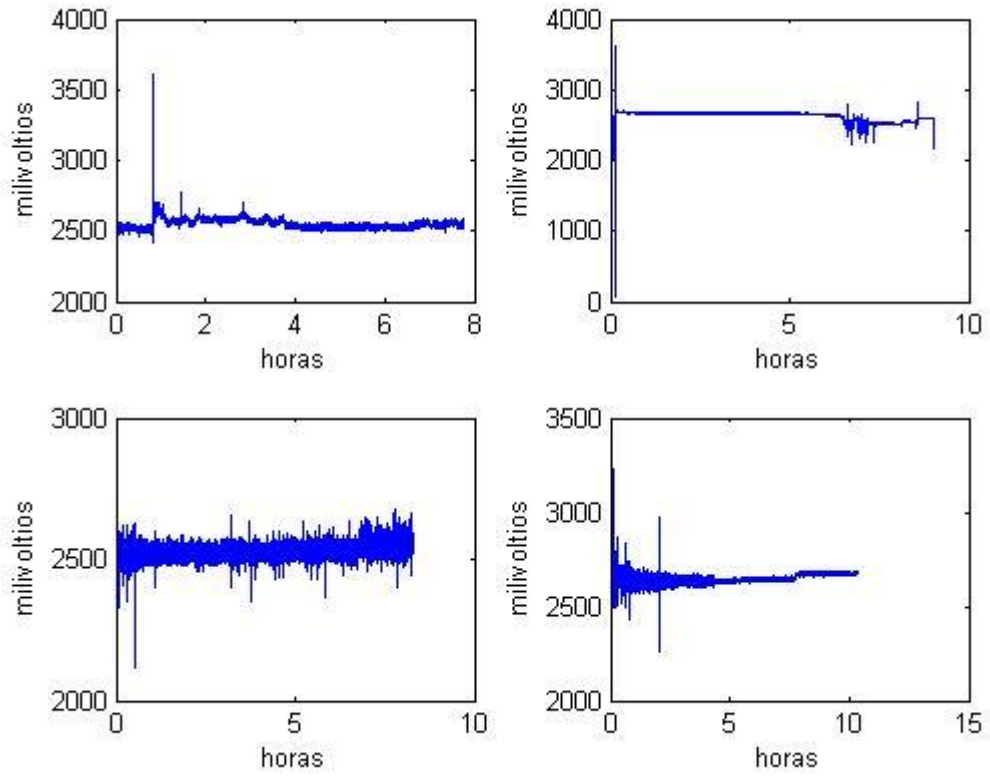


Figura 29. Grabación de audio a frecuencia de 1Hz

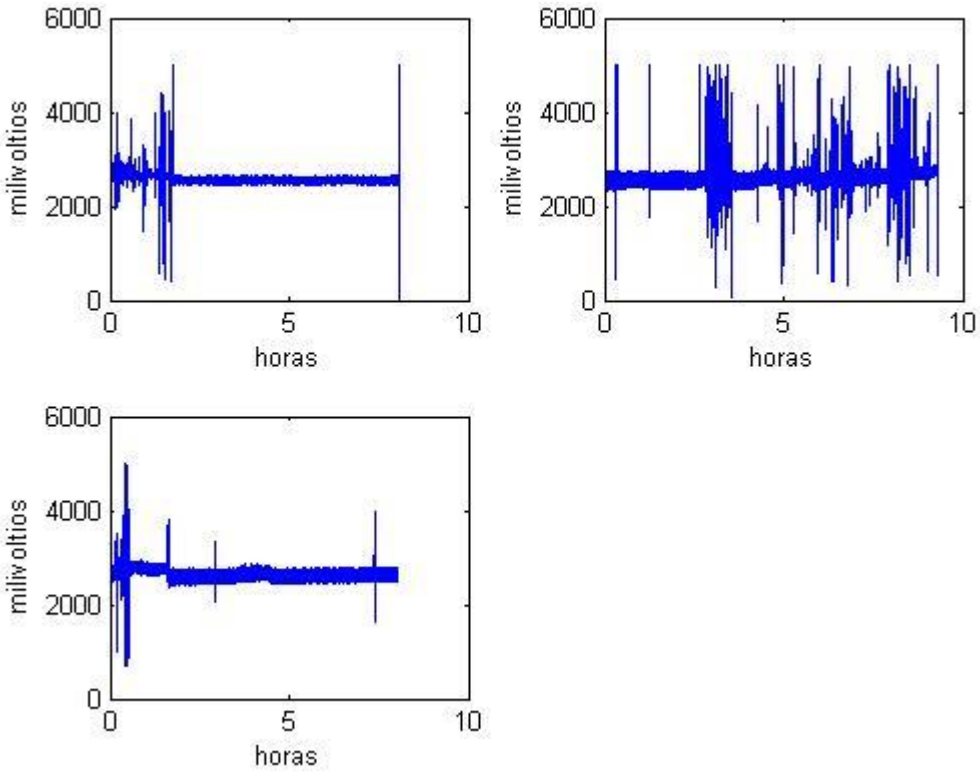


Figura 30. Grabación de audio a frecuencia de 100Hz

Aparentemente no se observa una clara relación entre ellas, lo cual hace pensar en problemas debidos con el módulo de grabación como: limitación en la ganancia de salida, el elevado ruido, fallos técnicos durante la grabación o elección de la frecuencia de muestreo errónea.

Para comprobar si estos datos pueden aportar más información de la observada a simple vista apoyándose en los estudios [9], [10] y [12], se opta por la creación de una inteligencia artificial para extraer información. Ya que estos programas son capaces de detectar características a simple vista invisibles.

Para el desarrollo de una red neuronal es necesario una amplia base de datos [11] la cual aporte tanto datos de la misma naturaleza que los que se van a medir como de su respectivo análisis, es decir la estadificación de cada sueño.

Esta base de datos ofrece los valores recogidos mediante un micrófono a frecuencia de un hercio. Se procede a la creación de una red neuronal inspirada en el trabajo [12]

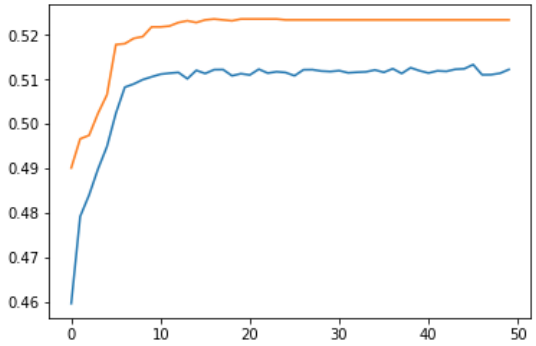


Figura 31. Datos de entrenamiento de la red neuronal [12]

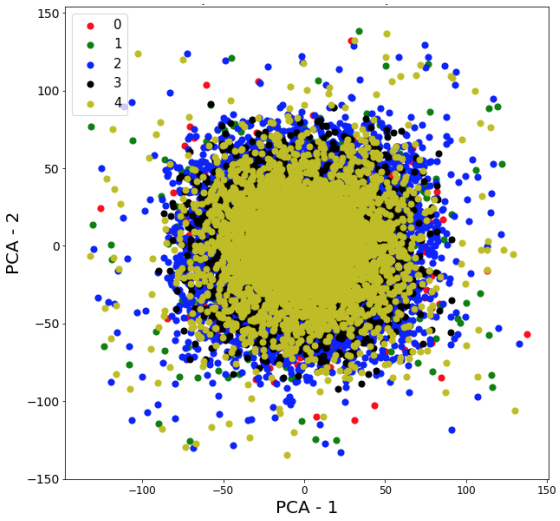


Figura 32. Análisis PCA de la red neuronal

Para comprobar si la red neuronal es capaz de diferenciar entre fases del sueño se entrena con la base de datos mediante una sucesión de epocas que mediante la salida que ofrece la red se compara con el dato que debería haber dado para informarle mediante una realimentación de si ha acertado o no, este proceso ajusta unos valores llamados “pesos” de las neuronas.

Para comprobar si la red está haciendo el trabajo bien se genera la figura 31 la cual nos ayuda a ajustar los hiperparametros como la cantidad de veces que se repite la sucesión

[12] En azul, validación con datos de entrenamiento. En naranja, validación con datos externos.

anterior o con que factor ajusta los pesos mediante la realimentación. Tras el ajuste se repite la gáfica que nos da información del acierto,vaalorado sobre 1.

Despues de hacer pruebas, ajustando todos los parametros el valor máximo que se obtiene es del 52% lo cual no deja de ser un proceso azaroso. Para comprobar si los datos son realmente separables en categorias se realiza un analisis PCA donde se puede ver que todas las fases del sueño, de la 0 a la 5, se encuentran solapadas entorno a las dos principales componentes. Con lo cual resulta imposible discriminar entre fases con este metodo o con estos datos.

5.4 Acelerómetro

El acelerómetro es un dispositivo capaz de detectar cambios en la velocidad en tres ejes, debido a ello resulta idóneo para medir movimiento de forma directa. De igual manera es capaz de medir el ángulo formado por el eje del acelerómetro respecto a la línea horizontal del suelo. Para la realización del experimento este dispositivo se coloca encima de la manta de la cama. En la ecuación 9 se pueden ver estas dos influencias en la medida.

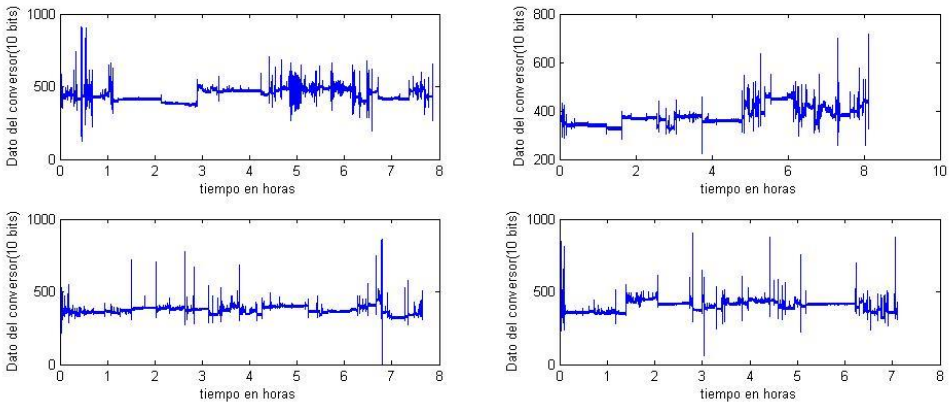


Figura 33. Datos en crudo recogidos mediante el acelerómetro a 100Hz

En las gráficas de la figura 32 se puede observar que la señal en determinados momentos cambia su componente de continua, esto indica un cambio en la inclinación del dispositivo respecto al suelo, si este estuviera anclado al cuerpo mediante alguna sujeción nos daría información sobre la postura, también información valiosa para el análisis. Sin embargo, al estar el transductor encima de la manta es información que se desprecia. En

contraposición no se hará así con el nivel de alterno acoplado a la continua, el cual da una clara información sobre el movimiento del sujeto durante la noche.

Igualmente, como se expone en [6] [5] que hay una clara diferencia de movimiento entre la primera mitad de la noche y la segunda. Asimismo, periodos de movimiento casi nulo e instantes de movimiento, lo cual puede dar una pista de que se está ante unos datos con información relevante.

$$S = C_1 * \sin(\alpha) + C_2 + C_3 * a$$

Ecuación 9: Salida de la medida del acelerómetro[13]

Para empezar, como se ha mencionado anteriormente se debe eliminar el nivel de continua de la señal mediante un filtro paso alto. El resultado se observa en la figura 8.

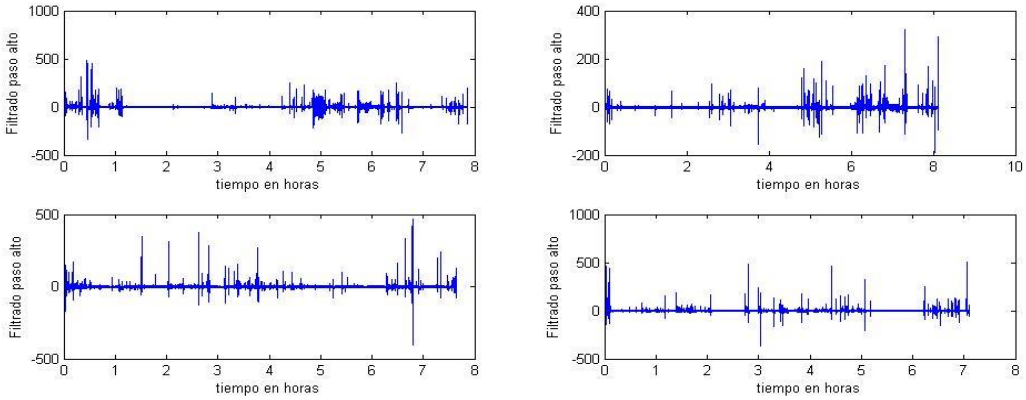


Figura 34. Datos del acelerómetro filtrados con un paso alto

Se tiene presente que el mismo sensor dependiendo de donde esté localizado puede arrojar unos resultados de detección diferentes, para ello es importante la normalización de la información. Con ese fin, se detecta el nivel de la intensidad de los datos una vez filtrados con el paso alto y se acumula ese valor que se usará para ajustar un umbral que se define a continuación.

[13] S, hace referencia al dato de salida del acelerómetro en un valor decimal. C_1, da constancia de una relación proporcional debido tanto a la propia amplificación del dispositivo como a la resolución en bits del conversor analógico digital que se use. α , es el ángulo que forman el eje del acelerómetro con el que se mide y la línea tangente a la tierra. C_2, afecta al nivel de continua sobre el que esté acoplada la señal. C_3 , es la constante que influye en la medida de la aceleración, suele ser la masa del detector junto a la ganancia de la circuitería. Finalmente, a es el termino de aceleración en m/s^2

Al tratarse de una onda con componente únicamente alterna podemos despreciar los valores negativos y detectar los máximos positivos, ya que nos aporta información sobre la intensidad de ese movimiento. Esto se puede llevar a cabo mediante la inclusión de un detector de pico creado por software. Para evitar que solo detecte el pico más grande y mantenga por tiempo indefinido ese valor, se modifica el código para que el valor almacenado se vaya atenuando con el tiempo. En la figura 35 se puede ver el resultado para cada una de las gráficas anteriores.

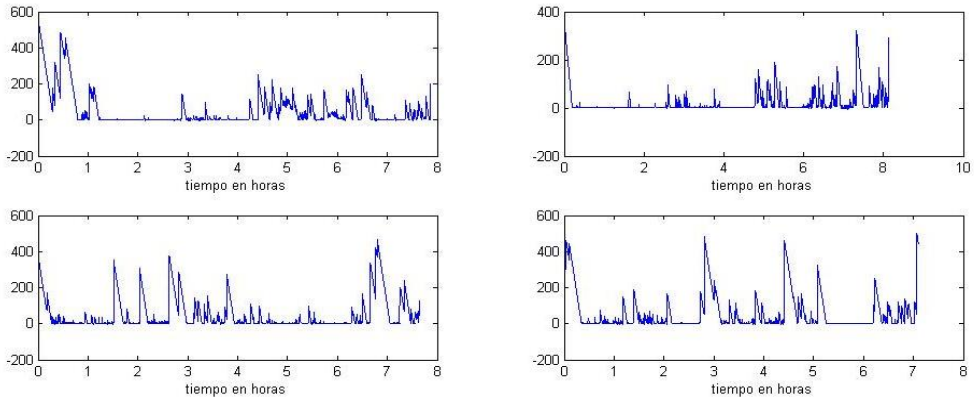


Figura 35.Detector de pico para los datos del filtro paso alto

Posteriormente gracias al valor de intensidad nombrado anteriormente se ajusta un umbral. Este valor se compara en intervalos de diez minutos con el valor de la señal del paso alto. Así pues, si en la totalidad de ese marco de diez minutos la señal ha estado por encima del umbral se define que el sujeto ha estado con un movimiento relativo del cien por cien. En el caso contrario cuando esta señal no ha superado en ningún momento el umbral se establece que ha tenido un movimiento relativo del cero por ciento. Cuando el caso es intermedio y la señal a veces está por encima y otras veces por debajo se realiza una ponderación temporal con los límites anteriormente mencionados. En la figura 36 se expone mediante las gráficas el resultado.

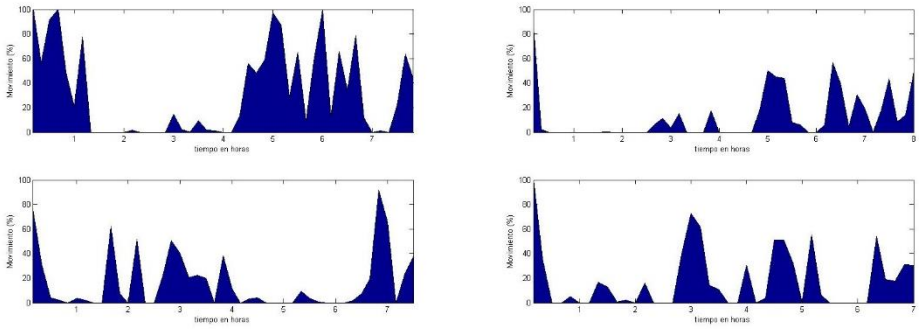


Figura 36.Movimiento detectado

El último paso es detectar en que momentos el sujeto atraviesa la fase de REM para que al salir de ella el sistema lo reconozca y el algoritmo envié una señal para activar el sonido del despertador, ya que como se dice en el apartado 3.2 es el mejor momento para despertar. Para ello haciendo uso de la información en los artículos [5] y [6], se establece una lógica que dicta que la fase rem es aquella en el que el movimiento relativo oscila entre un 10 y un 30 por ciento. Además, cada fase suele encontrarse a mínimo 90 minutos de otra ya que la duración media de un ciclo completo oscila entre 90 y 120 minutos, debido a esto entre cada fase de rem se encuentra un periodo de sueño profundo, aquel en el que casi no hay movimiento. Usando esta información se crea a partir de todos los datos recabados anteriormente un esquema de las fases que el sistema reconoce como REM el cual se puede observar superpuesto a la gráfica de movimiento en la 37.

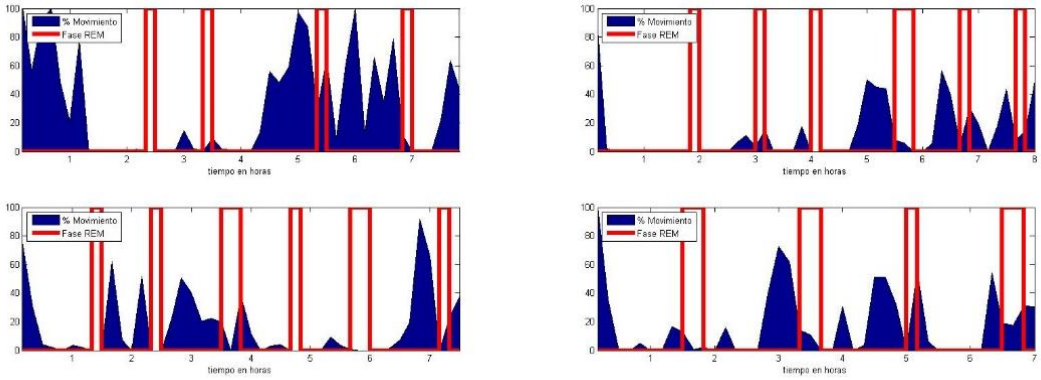


Figura 37. Detecciones de fase rem

Como se puede observar en las pruebas con datos de cuatro noches diferentes se puede observar que el algoritmo detecta entre 4 y 6 fases rem en noches de 8 horas lo cual coincide con la información obtenida de los artículos mencionados anteriormente. Además, el tiempo total en fase rem en cada prueba ronda el 25% del total lo cual es otro dato indicativo de que el algoritmo está funcionando. También cabe destacar la detección de una fase REM en los primeros 90 minutos de sueño lo cual también encaja en los estudios.

Para finalizar se muestra en la figura 38 una comparación entre una aplicación a la venta y los datos obtenidos durante la realización de este proyecto. Ambos datos fueron recogidos la misma noche por sendos dispositivos y se puede observar la clara relación entre las dos detecciones de movimiento, coincidiendo en cada hora los picos de actividad e inactividad.

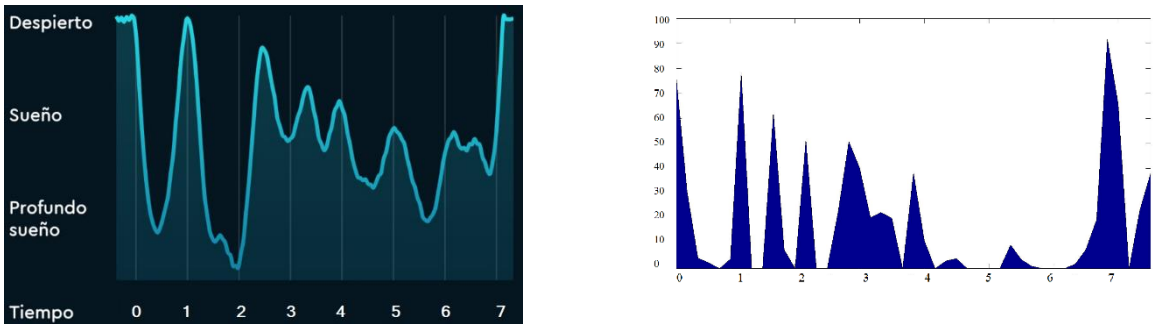


Figura 38. Comparación entre aplicación y código propio

6 LIMITACIÓNES, MEJORAS FUTURAS Y CONCLUSIONES

En cuanto a las limitaciones, se considera que los sensores usados para el presente estudio no han tenido la calidad suficiente para una buena extracción de los datos. Así mismo, la dificultad para acceder a información en bases de datos ha limitado la capacidad de este trabajo. Sin embargo, estas limitaciones han hecho pensar en alternativas que podrían ser incluso más adecuadas que las pensadas originalmente.

Para futuras investigaciones es necesario el uso de sensores más precisos que aporten más información. Además, sería oportuno el desarrollo de una inteligencia artificial para mejorar la interpretación de los datos, ya que estos algoritmos son capaces de detectar patrones difícilmente reconocibles por otros métodos.

Pensando en un uso práctico de este proyecto se podría no solo enfocar en un usuario sino en varios, como por ejemplo, en un centro hospitalario donde sería necesario el uso de una infraestructura IoT (*Internet of things*). Todos los dispositivos se conectarían con un único nodo central y almacenarían la información en "la nube" con posibilidad de un análisis posterior de esos datos para la obtención de información sobre la calidad del sueño de cada paciente. También se podría extrapolar a un hotel, ya que daría información valiosa al propietario como por ejemplo cual es la habitación en la que mejor duermen sus clientes.

Como conclusión, es posible diferenciar las etapas del sueño mediante el uso de sensores de fácil alcance como los usados en este proyecto. Sin embargo, para afirmar si este prototipo mejora el despertar del usuario, sería necesario el desarrollo de un estudio que encontrara diferencias entre el uso o no uso del presente prototipo.

8 BIBLIOGRAFÍA

[1] K. A. A. Rechtschaffen A, Manual of Standardized Terminology, Techniques and Scoring System for Sleep Stages of human Subjects, Los Ángeles: UCLA, 1968.

[2] D. V. Escobar, «Evolución histórica de los métodos de investigación,» 2000. Disponible en: http://www.scielo.org.pe/pdf/rmh/v11n4/v11n4ce2.

[3] Sleep Cycle AB, «Sleep Cycle,» 2020. Disponible en: https://www.sleepcycle.com/.

[4] Neybox Digital Ltd., «Pillow,» 2020. Disponible en: https://apps.apple.com/es/app/pillow-automatic-sleep-tracker/id878691772.

[5] TUCK.COM LLC, «Stages of Sleep and Sleep Cycles,» Octubre 2019. Disponible en: https://www.tuck.com/stages/.

[6] J. Wilde-Frenz y H. Schulz, «Rate and distribution of body movements during sleep in Humans,» Marzo 1983. Disponible en: https://www.researchgate.net/publication/16355093_Rate_and_distribution_of_body_movements_during_sleep_in_Humans.

[7] C. J. Hilditch y A. W. McHill, «Sleep inertia: current insights,» Agosto 2019. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6710480/.

[8] C. J. HILDITCH, . J. DORRIAN y S. BANKS, «Time to wake up: reactive countermeasures to sleep inertia,» Mayo 2016. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5136610/#bib_035.

[9] E. Eliran Dafna, A. Tarasiuk y Y. Zigel, «Sleep staging using nocturnal sound analysis,» Septiembre 2018. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6128888/.

[10] E. Dafna, A. Tarasiuk y Y. Zigel, «Sleep-Wake Evaluation from Whole-Night Non-Contact Audio Recordings of Breathing Sounds,» Febrero 2015. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4339734/.

[11] «Sleep-EDF Database Expanded,» Octubre 2013. Disponible en: https://alpha.physionet.org/content/sleep-edfx/1.0.0/#ref1.

[12] S. Mousavi y F. Afghah, «SleepEEGNet: Automated Sleep Stage Scoring with,» Marzo 2019. Disponible en: https://arxiv.org/pdf/1903.02108.pdf.

[13] ON Semiconductor, «MC74HC138A Datasheet,» Junio 2005. Disponible en: https://pdf1.alldatasheet.es/datasheet-pdf/view/172770/ONSEMI/MC74HC138A.html.

[14] MICROCHIP, «ATmega328 Datasheet,» 2018. Disponible en: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf.

[15] CORE ELECTRONICS, «JY-MCU Bluetooth to UART Wireless Serial Port,» Disponible en: https://core-electronics.com.au/attachments/guides/Product-User-Guide-JY-MCU-Bluetooth-UART-R1-0.pdf.

[16] SGS-THOMSON, «BC857 Small Signal PNP Transistors,» 1997. Disponible en: https://www.alldatasheet.com/datasheet-pdf/pdf/21971/STMICROELECTRONICS/BC857.html.

[17] cadence,«OrCAD,»2019. Disponible en:https://www.orcad.com/products/orcad-capture/overview.

[18] MAXIM, «Low-Cost, Microphone Preamplifiers with complete Shutdown,» Disponible en: https://www.alldatasheet.com/datasheet-pdf/pdf/460080/MAXIM/MAX4466.html.

[19] Flyron Technology, «FN-M16P Embedded MP3 Audio Module,» Disponible en: http://www.flyrontech.com/uploadfile/download/20184121510393726.pdf.

[20] WÜRTH ELEKTRONIK, «WL-SMCW SMT Mono-color Chip,» Disponible en: http://www.alldatasheet.es/datasheet-pdf/pdf/1089670/WURTH/150060AS75000.html.

[21] ZETEX, «BC817 Datasheet,» Disponible en: https://www.alldatasheet.com/datasheet-pdf/pdf/34269/ZETEX/BC81725.html.

[22] V. Fernández Escartín, C. Bernal Ruiz y F. J. Perez Cebolla, «Electónica Analógica,» Universidad de Zaragoza, Zaragoza, 2005.

[23] B. Martín del Brío y A. Bono Nuez, <Sistemas Electrónicos Programables>Universidad de Zaragoza,2017

9 ÍNDICE DE FIGURAS

10 ÍNDICE DE TABLAS

11 ANEXOS

11.1 LIBRERÍA PARA LA PANTALLA

11.1.1 ARCHIVO DE CUERPO

```
/*
  DisplayTFG.cpp - Libreria para controlar pantalla.
  Creado por Germán Viñao,2 de Noviembre, 2019.

*/

#include "Arduino.h"
#include "DisplayTFG_Libreria.h"

int contDinamico=0;
byte estadoPantalla=1;

void onMin(){//Activa el transistor de encendido de los leds superiores
    if(estadoPantalla==1) {
        PORTC = PORTC & B11011111;
    }
 }

void offMin(){//Desactiva el transistor de encendido de los leds superiores
    PORTC = PORTC | B00100000;
 }

void onHor(){//Activa el transistor de encendido de los leds inferiores
    if(estadoPantalla==1){
        PORTC = PORTC & B11101111;
    }
 }

void offHor(){//Desactiva el transistor de encendido de los leds inferiores
    PORTC = PORTC | B00010000;
}

void iniPuertos(){//Establece los pines como salidas o entradas
    DDRD = DDRD | B01110001; //Output PD6,PD5,PD4,PD0
    DDRD = DDRD & B11110001; //Input PD3,PD2,PD1
    DDRB = DDRB | B00111110; //Output PB1-PB5
    DDRC = DDRC | B00110000; //Output PC4 Y PC5 solo cambiando esos a 1
    DDRC = DDRC & B10110000; //Input PC0,PC1,PC2,PC3,PC6 se cambian a 0
    PORTB = PORTB & B11000011; //Se inicializa a 00, (las 12)
    offMin();
    offHor();
}

void offDisplay(){//Apaga la pantalla por completo
```

```
    offMin();
    offHor();
    estadoPantalla=0;
 }

void onDisplay(){//Permite encender los leds superiores o inferiores
    estadoPantalla=1;
}

void numDisplay(int num){
    PORTB = PORTB & B11000011; //Borra lo que habia en la posicion 2-5
    PORTB = PORTB | (B00000000+num<<2); //Añade el dato en posicion 2-5
}

void bienvenida(){//Juego de luces
    onMin();
    for(int j=0;j<2;j++){
        for (int i=0;i<12;i++){
            numDisplay(i);
            delay(40);
        }
    }
    for(int j=0;j<2;j++){
        offMin();
        onHor();
        for (int i=0;i<12;i++){
            numDisplay(i);
            delay(20);
        }
        offHor();
  }
}


/*
*Función para mostrar información completa en la pantalla
*Tiempo de computo 25us D<=2ms
*/
void pantallaDinamica(int hora,int minuto,int info1,int info2,int info3){


    contDinamico = (contDinamico % 5)+1; //Crea un bucle con el valor del contador
desde uno hasta 5

    if(contDinamico==1){  //Mostramos la hora
        onHor();
        offMin();
        numDisplay(hora);
    }
```

```
 if(contDinamico==2){  //Mostramos el minuto
      onMin();
      offHor();
      numDisplay(minuto/5);
  }

  if(contDinamico==3){  //Muestra la información 1 Leds AM y PM
     switch(info1){
         case 1:
         onMin();offHor();numDisplay(12);break;
         case 2:
         onHor();offMin();numDisplay(12);break;
         case 3:
         offMin();offHor();break;

    }
  }
  if(contDinamico==4){  ////Muestra la información 2  Leds MODO
     switch(info2){
         case 1:
         onMin();offHor();numDisplay(14);break;
        case 2:
        onHor();offMin();numDisplay(13);break;
         case 3:
        onMin();offHor();numDisplay(13);break;
        case 4:
        offMin();offHor();break;
     }
  }
  if(contDinamico==5){  ////Muestra la información 3 Leds Sonido ON y OFF
      switch(info3){
       case 1:
                onMin();offHor();numDisplay(15);break;
       case 2:
       onHor();offMin();numDisplay(15);break;
      }
   }
}
```

11.1.2 ARCHIVO DE ENCABEZADO

```
#ifndef DisplayTFG_Libreria_h
#define DisplayTFG_Libreria_h

#include "Arduino.h"

#define AM 1
#define PM 2
#define Ajuste_Hora 1
#define Ajuste_Alarma 2
#define Normal 3
#define Info_Extra 6
#define Sonido_On 1
#define Sonido_Off 2

void DisplayTFG();
void iniPuertos();
void offDisplay();
void onDisplay();
void bienvenida();
void pantallaDinamica(int hora,int minuto,int info1,int info2,int info3);
void onMin();
void offMin();
void onHor();
void offHor();
void numDisplay(int num);
```

11.2 LIBRERÍA PARA EL MÓDULO MP3

11.2.1 ARCHIVO DE CUERPO

```
/*
  MP3TFG.cpp - Libreria para controlar MP3.
  Creado por Germán Viñao,4 de Noviembre, 2019.
  */
 #include "MP3TFG_Libreria.h"
#include <Arduino.h>
```

```
#include <SoftwareSerial.h>

//Variables
byte kexe=0;
byte dato;
word checksum ;
byte Command_line[10] = { Byte_Comienzo, Byte_Version, Longitud_Comando, 0,
Acknowledge,
0, 0, 0, 0, Byte_Fin};
static SoftwareSerial * _software_serial = NULL;

//Funciones

void iniMP3(SoftwareSerial &theSerial){//Inicializa el puerto serie software
        _software_serial = &theSerial;
        for(int ini=0;ini<10;ini++){
                ejecutar_CMD(0x06, 0, 10);
                delay(1);
         }
          delay(200);
          for(int ini=0;ini<10;ini++){ //Muestra un sonido de Bienvenida
                file(10);
                delay(1);
         }
        delay(800);
        for(int ini=0;ini<10;ini++){ //Activa la función repetir
                  ejecutar_CMD(0x08,0,2);
                 delay(1);
          }
          delay(100);
          for(int ini=0;ini<10;ini++){ //Pausa el sonido
                  ejecutar_CMD(0x0E,0,2);
                   delay(1);
          }
```

```
  }


void pause(){//Envia un comando que el modulo entiende como parar la cancion
          ejecutar_CMD(0x0E,0,0);
  }
void play(){
            ejecutar_CMD(0x0D,0,1);
  }

void setVolume(byte volume){//Envia el comando seleccionar volumen
  ejecutar_CMD(0x06, 0, volume); // Seleccionar el volumen (0x00~0x30)
  }

void file(byte file){//Envia el comando eproducir la canción especificada
            ejecutar_CMD(0x03,0,file);
  }

void ejecutar_CMD(byte CMD, byte Par1, byte Par2){//Funcion para enviar
//Computo a 9600baud 1,5ms
// Periodo de 1ms a 5ms
kexe = (kexe%10)+1;
checksum = -( Byte_Version + Longitud_Comando + CMD + Acknowledge + Par1 +
Par2);
          switch (kexe){
                    case 4:
                    Command_line[3] = CMD;
                    case 6:
                    Command_line[5] = Par1;
                    case 7:
                    Command_line[6] = Par2;
                    case 8:
                    Command_line[7] = highByte(checksum);
                    case 9:
```

```
                Command_line[8] = lowByte(checksum);
        }
        _software_serial->write ( Command_line[kexe-1]);
 }

/*
*Función que se llamará desde el programa principal, envia comandos y datos según lo establecido en los parámetros, esta dividida con lo cual cada vez que se llama solo envia uun byte de los 10 que hacen falta para un comando completo.
*/
void enviarMP3(byte fnc,byte cancion,byte volumen){
        switch(fnc){
                        case 1:
                pause(); break;
                case 2:
                play();  break;
                case 3:
                  file(cancion); break;
                case 4:
                setVolume(volumen); break;
                default: break;
        }
  }
```

11.2.2 ARCHIVO DE ENCABEZADO

```
#ifndef MP3TFG_Libreria_h
#define MP3TFG_Libreria_h
#include "Arduino.h"
#include "SoftwareSerial.h"
# define Byte_Comienzo 0x7E
# define Byte_Version 0xFF
# define Longitud_Comando 0x06
# define Byte_Fin 0xEF
```

```
# define Acknowledge 0x00
# define enviarPausa 1
# define enviarPlay 2
# define enviarCancion 3
# define enviarVolumen 4
# define enviarNada 5

void iniMP3(SoftwareSerial &theSerial);
void pause();
void play();
void setVolume(byte volume);
void file(byte file);
void ejecutar_CMD(byte CMD, byte Par1, byte Par2);
void enviarMP3(byte fnc,byte cancion,byte volumen);//FNC:1 pausa, 2 play, 3 cancion,
4 volumen
#endif
```

11.3 LIBRERÍA PARA EL MÓDULO DE LUCES RGB

11.3.1 ARCHIVO DE ENCABEZADO

```
/*
  Luces_TFG_Libreria.cpp - Libreria para controlar los leds rgb.
  Forma de conectarlos, red->pin5, green->pin6, blue->pin9
  Creado por Germán Viñao, 2 de Noviembre, 2019.

*/

#include "Arduino.h"
#include "Luces_TFG_Libreria.h"

byte pinGreen=6;
byte pinRed=5;
byte pinBlue=9;
byte* vRed=0;
```

```
byte* vBlue=0;
byte* vGreen=0;
byte encendido=1;
float Red=0;
float Green=0;
float mBlue;
float mRed;
float mGreen;
float Blue=0;
int jLuz=1;
int pasos;

void iniLuces(){//Establece como salidas los pines necesarios
        DDRD = DDRD | B01100000;
 DDRB = DDRB | B00000010;
}

void setRGB(byte *R, byte *G, byte *B){//Almacena los punteros a los valores de color
        vRed=R;
        vBlue=B;
        vGreen=G;
}

void onRGB(){// Computo=40uS
        analogWrite(pinRed,*vRed);
        analogWrite(pinGreen,*vGreen);
        analogWrite(pinBlue,*vBlue);
}
void offRGB(){
Red=0;Green=0;Blue=0;
        analogWrite(pinRed,0);
        analogWrite(pinGreen,0);
        analogWrite(pinBlue,0);
        jLuz=1;
```

```
}

/*
        *Función que enciende poco a poco los leds al color establecido, varia de
        intensidad según el tiempo que queda
*/
void progresivo(byte minutos){// Cmax 80uS, Periodo 100ms=0.1segundos
        if(jLuz==1){
                pasos=((int)minutos*600); // 60segundos/0.1segundos
                if(minutos>60) minutos=60;
                if(minutos<=1) minutos=1;
                mRed=(float)*vRed/pasos;
                mGreen=(float)*vGreen/pasos;
                mBlue=(float)*vBlue/pasos;
        }
        if(jLuz==pasos){
                onRGB();
        }
        else{
                jLuz = (jLuz % pasos)+1;
                analogWrite(pinRed,(byte)Red);
                Red=Red+mRed;
                analogWrite(pinGreen,(byte)Green);
                Green=Green+mGreen;
                analogWrite(pinBlue,(byte)Blue);
                Blue=Blue+mBlue;
        }
}
```

11.3.2 ARCHIVO DE CABECERA

```
#ifndef Luces_TFG_Libreria_h
#define Luces_TFG_Libreria_h
```

```
#include "Arduino.h"
void iniLuces();
void setRGB(byte *R, byte *G, byte *B);
void onRGB();
void offRGB();
void progresivo(byte minutos);
#endif
```

11.4 CÓDIGO DEL PROGRAMA PRINCIPAL

```
#include <MP3TFG_Libreria.h>
#include <DisplayTFG_Libreria.h>
#include <Luces_TFG_Libreria.h>
#include <SoftwareSerial.h>

//PARA EL EJECUTIVO CICLICO
unsigned long siguiente=0;
unsigned int marco=0;
unsigned int num_marcos=20;
unsigned int tiempoMarco=2000; //microsegundos

//LUCES
byte r=0;
byte g=0;
byte b=0;
byte demoraLuces=20; //minutos antes de que suene la alarma, para iniciar los leds rgb

//VARIABLES PARA LOS ESTADOS
byte estado=1;
byte tono=0;

//VARIABLES PARA EL RELOJ
byte horaAlarma=10;
byte minutoAlarma=0;
```

```
byte infoAlarma1=AM;
byte horaActual=10;
byte minutoActual=13;
byte infoActual1=AM;
byte segundoActual=40;
int auxSegundoActual=0;

//VARIABLES PARA LA PANTALLA
//byte info1=1;
byte info2=1;
byte info3=1;
byte* infoMuestra1=0;
byte* horaMuestra=0;
byte* minutoMuestra=0;

//VARIABLES PARA LAS ENTRADAS
byte entrada[4];
byte auxEntrada;
byte ajustePorBluetooth;
byte estadoAlarma=0; //0 nada,1 encendido de luces, 2 encendido  de música.
byte alarmaActivada=0; //Para no entrar varias veces al estado 16,  se da constancia de que ya se ha activado
byte alarmaAjustada=0;//Cuando se accede a ajustes se pone a 1 si se apaga la alarma se pone a 0 para que no vuelva a sonar al no estar ajustada

//VARIABLES PARA EL MICROFONO
byte j;
word sumaVerificacion;
int valor=0;

//VARIABLES PARA EL MP3
SoftwareSerial MP3Serial(8,7);//inicializa 7 y 8
//funcionMP3MP3 1 pausa, 2 play, 3 cancion, 4 volumen,5 nada
byte funcionMP3=5,cancion=1,volumen=15;
```

```
/* Se repite una vez. Inicializamos*/
void setup() {

        //PIN PARA MEDIR TIEMPOS
        pinMode(4,OUTPUT);

        //INTERRUPCIONES
        attachInterrupt(digitalPinToInterrupt(3),botones,FALLING);
        attachInterrupt(digitalPinToInterrupt(2),botones,FALLING);



        //AJUSTE INTERRUPCION TIMER2 A 125Hz
 TCCR2A = 0;// TCCR2A registro a 0
         TCCR2B = 0;// Lo mismo para TCCR2B
 TCNT2  = 0;//Inicializar contador a 0
 //Ajustamos registro comparador
 OCR2A = 124;// = (16*10^6) / (125*1024) - 1 (deberia ser  <256)
 // modo CTC
 TCCR2A |= (1 << WGM21);
 // Preescaler del reloj de 16Mhz, 7=1024
 TCCR2B |= 7  ;
 // Activar interrupcion comparador
 TIMSK2 |= (1 << OCIE2A);

//INICIALIZA LAS LUCES
          setRGB(&r,&g,&b);

        //INICIALIZA EL MP3
          MP3Serial.begin(9600);
          iniMP3(MP3Serial);

        //INICIALIZA PUERTOS
```

```
 Serial.begin(9600);
 iniPuertos();  //INICIALIZA TODOS LOS PUERTOS menos 7 y 8 que se inicializar con
 mp3Ini()
 bienvenida();

  }//Final setup

//FUNCION CICLICA. CODIGO A EJECUTAR
void loop() {

siguiente=micros(); //Cada 70 minutos reinicia, se tiene en cuenta
marco = (marco % num_marcos)+1 ;
   switch (marco) { //Esto es el ejecutivo cíclico
          case 1:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);est
          ados();enviarInfo();laHora(); break;
          case 2:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
          viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
          case 3:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
          viarInfo();break;
          case 4:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
          viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
          case 5:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
          viarInfo(); break;
          case 6:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
          viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
          case 7:
          pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
          viarInfo(); break;
```

```
case 8:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
case 9:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo(); break;
case 10:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
case 11:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo(); break;
case 12:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
case 13:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo(); break;
case 14:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
case 15:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo(); break;
case 16:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
case 17:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo(); break;
case 18:
pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;
```

```
        case 19:
        pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
        viarInfo(); break;
        case 20:
        pantallaDinamica(*horaMuestra,*minutoMuestra,*infoMuestra1,info2,info3);en
        viarInfo();enviarMP3(funcionMP3,cancion,volumen); break;

  }
  siguiente+=tiempoMarco;//El siguiente tiempo es el inicio del siguiente marco
  while(micros()<siguiente){ //Se espera hasta el siguiente marco
         asm("sleep\n"); //Para ahorrar energía, la mayor parte del tiempo está apagado
   }
}//Final del bucle

//FUNCIONES

/*funcionMP3 para leer la entrada de bluetooth y conmutador*/
void leerEntrada(){ //Computo 15us
 if(Serial.available()>0){
    auxEntrada = Serial.read();//leo el buffer
            if(auxEntrada!=0){
        entrada[0]=auxEntrada;
   }
 }
         entrada[1]=(PINC & (1 << 1 ))>>1; //SELECCIONO ENTRADA PC1 ( MODO
NORMAL)
         entrada[2]=(PINC & (1 << 2 ))>>2; //SELECCIONO PC2 (ALARMA)
        entrada[3]=(PINC & (1 << 3 ))>>3; //SELECCIONO PC3 (HORA)
 }




/*funcionMP3 para enviar datos al pc de forma segura con verificación de dato correcto*/
```

```
void enviarInfo(){
        j = (j % 4)+1;
         if(j==1){valor = analogRead(A0);}//LEO SOLO UNA VEZ
        byte comando_info[4] = {0,estado,highByte(valor),lowByte(valor)};
         Serial.write(comando_info[j-1]);
 }

// Interrupcion
 void botones(){ //C=40us con serial

        if(estado==3){// alarma
                if((PIND & (1 << 2 ))>>2 == 0) minutoAlarma+=5;
                if((PIND & (1 << 3 ))>>3 == 0) horaAlarma++;
         }
         if(estado==1){//modo hora actual
                if((PIND & (1 << 2 ))>>2 == 0) minutoActual+=5;
                if((PIND & (1 << 3 ))>>3 == 0) horaActual++;
         }
        if(estado==15){
                if((PIND & (1 << 3 ))>>3 == 0) estado=14;
         }
   }

//Interrupción temporal a 1Hz
 ISR(TIMER2_COMPA_vect){//timer1 interrupt 1Hz
          if(estado==2 || estado==5 || estado==15){
                auxSegundoActual++;
          }
          if(auxSegundoActual>=125){
                segundoActual++;auxSegundoActual=0;
          }
}
```

```
void laHora(){
        if(segundoActual>=60) {minutoActual++; segundoActual=0;}
         if(minutoActual>=60) {horaActual++; minutoActual=0;}
         if(horaActual>=12 && infoActual1==AM) {horaActual=0; infoActual1=PM;}
         if(horaActual>=12 && infoActual1==PM) {horaActual=0; infoActual1=AM;}
 }

void alarma(){
         int                                    a                                    =
        diferenciaHoras(horaAlarma,minutoAlarma,infoAlarma1,horaActual,minutoAct
        ual,infoActual1);
         if(a==0) estadoAlarma=2;
         else if(a<=demoraLuces) estadoAlarma=1;
        else estadoAlarma=0;
  }

/*
 * Mide en minutos la diferencia entre la primera y la segunda hora
*/
int diferenciaHoras(byte h1,byte m1,byte i1,byte h2,byte m2,byte i2){//C=5us
         int difHoras,difMinutos;
         if(m1-m2<0){
                 difHoras=(h1+12*(i1-1))-1; //se resta una hora a la inicial,12*(i1-1) sirve
                para añadir 12 si es pm y 0 si es am asi tenemos de 0 a 24 horas
                  difMinutos=m1+60;
                   difMinutos-=m2;
                   if(difHoras-(h2+12*(i2-1))>=0){
                           difHoras=difHoras-(h2+12*(i2-1));
                  }
                else{
                        difHoras=difHoras+24-(h2+12*(i2-1));
                }
         }
         else{
```

```
                difMinutos=m1-m2;
                 if(h1-h2>=0){
                        difHoras=(h1+12*(i1-1))-(h2+12*(i2-1));
                }
                 else{
                          difHoras=(h1+12*(i1-1))+24-(h2+12*(i2-1));
                 }
          }
           return (int)difHoras*60+difMinutos;
  }


void estados(){
   leerEntrada();
   alarma();
   funcionMP3=5;
   switch(estado){
     case 1: //Ajuste manual HORA
       horaMuestra=&horaActual;
       minutoMuestra=&minutoActual;
       info2=Ajuste_Hora;
       infoMuestra1=&infoActual1;

       if(entrada[0]==17)estado=4;
       else if(entrada[0]==18)estado=5;
       else if(entrada[0]==19)estado=6;
       else if(entrada[0]==20)estado=7;
       else if(entrada[0]==21)estado=8;
       else if(entrada[0]==22)estado=13;
       else if(entrada[0]==23)estado=14;
       if(entrada[2])estado=2;
       else if(entrada[3]) estado=3;

       break;
```

```
case 2:
  horaMuestra=&horaActual;
  minutoMuestra=&minutoActual;
  info2=Normal;
  infoMuestra1=&infoActual1;

  if(entrada[0]==17)estado=4;
  else if(entrada[0]==18)estado=5;
  else if(entrada[0]==19)estado=6;
  else if(entrada[0]==20)estado=7;
  else if(entrada[0]==21)estado=8;
  else if(entrada[0]==22)estado=13;
  else if(entrada[0]==23)estado=14;
  if(entrada[1])estado=1;
  else if(entrada[3])estado=3;

  if(estadoAlarma==1 && alarmaAjustada)estado=15;
  break;
case 3:


  horaMuestra=&horaAlarma;
  minutoMuestra=&minutoAlarma;
  info2=Ajuste_Alarma;
  infoMuestra1=&infoAlarma1;

  alarmaAjustada=1;

  if(entrada[1])estado=1;
  else if(entrada[2])estado=2;
  if(entrada[0]==16)estado=4;
  else if(entrada[0]==18)estado=5;
  else if(entrada[0]==19)estado=6;
```

```
  else if(entrada[0]==20)estado=7;
  else if(entrada[0]==21)estado=8;
  else if(entrada[0]==22)estado=13;
  else if(entrada[0]==23)estado=14;
  break;

case 4://Ajuste Hora Actual
  switch(entrada[0]){
    case 1:
    estado=9;
    break;
    case 2:
    estado=10;
    break;
    case 18:
    estado=5;
    break;
    case 19:
    estado=6;
    break;
    case 20:
    estado=7;
    break;
    case 21:
    estado=8;
    break;
    case 22:
    estado=13;
    break;
    case 23:
    estado=14;
    break;
    }
```

```
  horaMuestra=&horaActual;
  minutoMuestra=&minutoActual;
  info2=Ajuste_Hora;
  infoMuestra1=&infoActual1;
  break;

case 5:
  switch(entrada[0]){
    case 17:
    estado=4;
    break;
    case 19:
    estado=6;
    break;
    case 20:
    estado=7;
    break;
    case 21:
    estado=8;
    break;
    case 22:
    estado=13;
    break;
    case 23:
    estado=14;
    break;
    }
  if(estadoAlarma==1 && alarmaAjustada)estado=15;

  horaMuestra=&horaActual;
  minutoMuestra=&minutoActual;
  info2=Normal;
  infoMuestra1=&infoActual1;
  break;
```

```
case 6://AJUSTE ALARMA
  switch(entrada[0]){
    case 1:
    estado=11;
    break;
    case 2:
    estado=12;
    break;
    case 17:
    estado=4;
    break;
    case 18:
    estado=5;
    break;
    case 20:
    estado=7;
    break;
    case 21:
    estado=8;
    break;
    case 22:
    estado=13;
    break;
    case 23:
    estado=14;
    break;

    }
  horaMuestra=&horaAlarma;
  minutoMuestra=&minutoAlarma;
  info2=Ajuste_Alarma;
  infoMuestra1=&infoAlarma1;
```

```
      alarmaAjustada=1;
      break;

case 7://AJUSTE CANCION
     switch(entrada[0]){
      case 1:
      cancion=1;
      funcionMP3=3;
      break;
      case 2:
      cancion=2;
      funcionMP3=3;
      break;
      case 3:
      cancion=3;
      funcionMP3=3;
      break;
      case 4:
      cancion=4;
      funcionMP3=3;
      break;
      case 5:
      cancion=5;
      funcionMP3=3;
      break;
      case 6:
      cancion=6;
      funcionMP3=3;
      break;
      case 7:
      cancion=7;
      funcionMP3=3;
     break;
      case 8:
```

```
cancion=8;
funcionMP3=3;
 break;
case 9:
cancion=9;
funcionMP3=3;
break;
case 10:
cancion=10;
funcionMP3=3;
break;
case 17:
estado=4;
funcionMP3=1;
break;
case 18:
estado=5;
funcionMP3=1;
break;
case 19:
estado=6;
funcionMP3=1;
break;
case 21:
estado=8;
funcionMP3=1;
break;
case 22:
estado=13;
funcionMP3=1;
break;
case 23:
estado=14;
funcionMP3=1;
```

```
 break;
 }

 break;

case 8://AJUSTE DE LUZ
 onRGB();
 switch(entrada[0]){
  case 1: //NARANJA
    r=255;
    g=40;
    b=10;
    break;
  case 2: //amanecer
    r=255;
    g=100;
    b=70;
    break;
  case 3: //amarillo
    r=255;
    g=130;
    b=0;
    break;
  case 4://verde
    r=40;
    g=255;
    b=0;
    break;
  case 5://verde azulado
    r=145;
    g=255;
    b=170;
    break;
  case 6://azul verdoso
```

```
  r=0;
  g=255;
  b=255;
  break;
case 7://azul intenso
  r=20;
  g=0;
  b=255;
  break;
 case 8://violeta
  r=255;
  g=0;
  b=255;
  break;
 case 9://rosa
  r=255;
  g=20;
  b=20;
  break;
 case 10://rojo
  r=255;
  g=5;
  b=5;
  break;

 case 17:
  offRGB();
  estado=4;
 break;
 case 18:
  offRGB();
  estado=5;
 break;
 case 19:
```

```
    estado=6;
    offRGB();
  break;
  case 20:
    estado=7;
    offRGB();
  break;
  case 22:
    estado=13;
    offRGB();
  break;
  case 23:
    estado=14;
    offRGB();
  break;
  }

  break;

case 9:
  switch(entrada[0]){
    case 1:
    horaActual=1;
    break;
    case 2:
    horaActual=2;
    break;
    case 3:
    horaActual=3;
    break;
    case 4:
    horaActual=4;
    break;
    case 5:
```

```
horaActual=5;
break;
case 6:
horaActual=6;
break;
case 7:
horaActual=7;
break;
case 8:
horaActual=8;
break;
case 9:
horaActual=9;
break;
case 10:
horaActual=10;
break;
case 11:
horaActual=11;
break;
case 12:
horaActual=0;
break;
case 13:
infoActual1=AM;
break;
case 14:
infoActual1=PM;
break;
case 15:
estado=10;
break;
case 17:
estado=4;
```

```
    break;
    case 18:
    estado=5;
    break;
    case 19:
    estado=6;
    break;
    case 20:
    estado=7;
    break;
    case 21:
    estado=8;
    break;
    case 22:
    estado=13;
    break;
    case 23:
    estado=14;
    break;


    }
    break;
case 10:
  switch(entrada[0]){
    case 1:
    minutoActual=5;
    break;
    case 2:
    minutoActual=10;
    break;
    case 3:
    minutoActual=15;
    break;
```

```
case 4:
minutoActual=20;
break;
case 5:
minutoActual=25;
break;
case 6:
minutoActual=30;
break;
case 7:
minutoActual=35;
break;
case 8:
minutoActual=40;
break;
case 9:
minutoActual=45;
break;
case 10:
minutoActual=50;
break;
case 11:
minutoActual=55;
break;
case 12:
minutoActual=0;
break;
case 13:
infoActual1=AM;
break;
case 14:
infoActual1=PM;
break;
case 16:
```

```
    estado=9;
    break;
    case 17:
    estado=4;
    break;
    case 18:
    estado=5;
    break;
    case 19:
    estado=6;
    break;
    case 20:
    estado=7;
    break;
    case 21:
    estado=8;
    break;
    case 22:
    estado=13;
    break;
    case 23:
    estado=14;
    break;
    }
    break;
case 11:
  switch(entrada[0]){
    case 1:
    horaAlarma=1; break;
    case 2:
    horaAlarma=2; break;
    case 3:
    horaAlarma=3; break;
    case 4:
```

```
horaAlarma=4; break;
case 5:
horaAlarma=5; break;
case 6:
horaAlarma=6; break;
case 7:
horaAlarma=7; break;
case 8:
horaAlarma=8; break;
case 9:
horaAlarma=9; break;
case 10:
horaAlarma=10; break;
case 11:
horaAlarma=11; break;
case 12:
horaAlarma=0; break;
case 13:
infoAlarma1=AM; break;
case 14:
infoAlarma1=PM; break;
case 15:
estado=12; break;
case 17:
estado=4; break;
case 18:
estado=5; break;
case 19:
estado=6; break;
case 20:
estado=7; break;
case 21:
estado=8; break;
case 22:
```

```
    estado=13; break;
    case 23:
    estado=14; break;
    }
    break;
case 12:
 switch(entrada[0]){
    case 1:
    minutoAlarma=5;
    break;
    case 2:
    minutoAlarma=10;
    break;
    case 3:
    minutoAlarma=15;
    break;
    case 4:
    minutoAlarma=20;
    break;
    case 5:
    minutoAlarma=25;
    break;
    case 6:
    minutoAlarma=30;
    break;
    case 7:
    minutoAlarma=35;
    break;
    case 8:
    minutoAlarma=40;
    break;
    case 9:
    minutoAlarma=45;
    break;
```

```
case 10:
minutoAlarma=50;
break;
case 11:
minutoAlarma=55;
break;
case 12:
minutoAlarma=0;
break;
case 13:
infoAlarma1=AM;
break;
case 14:
infoAlarma1=PM;
break;
case 16:
estado=11;
break;
case 17:
estado=4;
break;
case 18:
estado=5;
break;
case 19:
estado=6;
break;
case 20:
estado=7;
break;
case 21:
estado=8;
break;
case 22:
```

```
    estado=13;
    break;
    case 23:
    estado=14;
    break;
    }
    break;

case 13:
  switch(entrada[0]){
  case 1:
  volumen=0;
  funcionMP3=4;
  info3=2;//OFF
  break;
  case 2:
  volumen=5;
  funcionMP3=4;
  info3=1;
  break;
  case 3:
  volumen=10;
  funcionMP3=4;
  info3=1;
  break;
  case 4:
  volumen=15;
  funcionMP3=4;
  info3=1;
  break;
  case 5:
  volumen=20;
  funcionMP3=4;
  info3=1;
```

```
break;
case 6:
volumen=25;
funcionMP3=4;
info3=1;
break;
case 7:
volumen=30;
funcionMP3=4;
info3=1;
break;
case 17:
estado=4;
funcionMP3=1;
break;
case 18:
estado=5;
funcionMP3=1;
break;
case 19:
estado=6;
funcionMP3=1;
break;
case 20:
estado=7;
funcionMP3=1;
break;
case 21:
estado=8;
funcionMP3=1;
break;
case 23:
estado=14;
funcionMP3=1;
```

```
    break;
    }
   break;

  case 14://SALIR
    offRGB();
    if(entrada[1])estado=1;
    else if(entrada[2])estado=2;
    else if(entrada[3]) estado=3;
    funcionMP3=1;
    alarmaActivada=0;
    alarmaAjustada=0;
    break;

 case 15:
    if(entrada[0]==24 && !alarmaActivada)estado=16;
    if(estadoAlarma==2 && !alarmaActivada)estado=16;
    progresivo(demoraLuces);
    funcionMP3=5;
    if(entrada[0]==23)estado=14;

    break;
 case 16:

    funcionMP3=3;//Reproduce la canción
    estado=15;//Vuelve al otro estado para no enviar la cancion una y otra vez
    alarmaActivada=1;
  }//Fin estados


}
```