Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería electrónica industrial y automática.

Trabajo de fin de grado:

# DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE BAJO COSTE PARA EL REGISTRO DE ACTIVIDAD EN APARCAMIENTOS 'DE 20 MINUTOS'

Autor: Sergio del Rosario Gutiérrez.

Tutora: Marta Sigut Saavedra.

Abstract ……1

Resumen ……1

1. Memoria descriptiva ……1

1.1. Antecedentes……1

1.2 Objetivos ……1

1.3. Ubicación……..2

1.4. Funcionamiento……3

1.5. Metodología ……..4

1.6. Especificaciones del prototipo …… 4

1.7. Hardware ……4

1.7.1. Arduino Pro-Mini ……4

1.7.1.1. Descripción ……4

1.7.1.2. Configuración de pines ……5

1.7.1.3. Definición de pines ……..6

1.7.1.4. Alimentación……..8

1.7.2. Arduino Mega R3……8

1.7.2.1. Descripción…… 8

1.7.2.2. Configuración de pines…….. 8

1.7.2.3. Definición de pines ……..9

1.7.2.4. Alimentación ……..11

1.7.3. Sensor de ultrasonido HC-SR04 ……12

1.7.3.1 Funcionamiento ……12

1.7.3.2 Descripción ……13

1.7.3.3. Conexionado ……..14

1.7.4. Cámara VC0706 ……..15

1.7.4.1. Descripción……15

1.7.4.2. Funcionamiento ……16

1.7.4.3. Comunicación. ……17

1.7.4.4. Conexionado. ……..17

1.7.5. SP3232 ……19

1.7.6. Módulo Bluetooth HC-06……22

1.7.7. Módulo microSD. ……24
1.7.7.1. Descripción. ……24
1.7.7.2. Comunicación……..25
1.7.7.3. Conexión. ……..26
1.7.8. Tarjeta microSD. ……27
1.7.9. Conversor CP2102. ……27
1.7.9.1. Descripción. ……27
1.7.9.2. Conexión. ……28
1.7.10. RTC. ……..29
1.7.10.1. Funcionamiento. ……..29
1.7.10.2. Descripción……29
1.7.10.3. Comunicación. ……..30
1.7.10.4. Conexión……31
1.8. Software. ……33
1.8.1. Librerías utilizadas. ……..33
1.8.2. IDE Arduino……33
1.8.1. Código……33
1.8.3. Kicad……35
1.9. Alimentación y consumo……..35
1.10. Pruebas reales y resultados……37
2. Memoria justificativa……38
2.1. Hardware. ……38
2.1.1. Selección del microcontrolador. ……38
2.1.2. Sensor de ultrasonido HC-SR04 ……38
2.1.3. Cámara VC0706. ……..39
2.1.4. SP3232. ……..39
2.1.5. Módulo Bluetooth HC-06. ……39
2.1.6. Módulo microSD. ……39
2.1.7. Tarjeta microSD. ……..39
2.1.8. Conversor CP2102. ……39

2.1.9. RTC. ……..39

2.2. Alimentación y consumo. ……40

3. Presupuesto. ……41

4. Conclusiones. ……42

5. Líneas Abiertas……42

6. Anexos ……..44

6.1. Anexo I ……..44

6.2. Anexo II ……46

7. Referencias. ……..56

## Abstract

In the last years, the improvement of open commercial areas has been one aim of the different town halls of the Tenerife island. With this intention, several town halls have decided to create, some car parks with a limited time. Normally the maximum time is 20 minute, although it can be 25 minutes too.

The problem is that, due to the amount of these car park type, the police can not check the time that a vehicle has been parked. So, the goal of this project is to create a low-cost device that can register the time that the car has been parked.

To solve the problem, we have designed a first prototype that through an ultrasonic sensor it is able to detect the arrival/exit of the car to the parking lot. Once the car is detected a camera take a picture and save it in a microSD card. When the car goes out, if the parked time is more than 20 minute, the device takes the second picture and save it, having as filename the date. If the parked time is less, the first photo taken is deleted.

## Resumen

En los últimos años, el número de aparcamientos de tiempo limitado ha aumentado en los municipios del norte de la isla de Tenerife. El tiempo de estacionamiento no puede ser controlado por la policía local debido al número de aparcamientos que hay de este tipo. Para conseguir que se respete el tiempo máximo permitido en ellos se ha diseñado un primer prototipo capaz de detectar la entrada y salida del vehículo capturando una imagen y almacenándola en una memoria (tarjeta microSD) con la hora de entrada y salida.

# 1. Memoria descriptiva

## 1.1. Antecedentes

En los últimos años, se ha venido desarrollando por los diferentes Ayuntamientos de la isla de Tenerife la mejora de las zonas comerciales abiertas. Con esa intención, una de las medidas adoptadas ha sido la implantación de aparcamientos de tiempo limitado. Este periodo de tiempo puede variar dependiendo del municipio en el que se encuentre. Esta restricción se aplica solamente durante el día, es decir, generalmente desde las 8:00 a las 20:00.

El problema es que el tiempo de estacionamiento de los vehículos en este tipo de aparcamiento no es controlado por las autoridades ya que requeriría un agente de policía por cada zona en la que se encuentran este tipo de plazas de aparcamiento.

## 1.2. Objetivos

Con este Trabajo de Fin de Grado se intenta dar una posible solución al problema. Para ello hemos diseñado e implementado un sistema empotrado capaz de detectar y registrar la entrada y salida de un vehículo de un aparcamiento en batería.

El prototipo se ha desarrollado en base a los siguientes requerimientos:

- Bajo coste.
- Bajo consumo.
- Tamaño reducido.
- Fácil programación.

Para ello se hecho uso de los siguientes componentes electrónicos:

- Arduino Mega 2560 R3
- Sensor de ultrasonido HC-SR04.
- Cámara VC0706.
- Módulo de almacenamiento microSD.
- RTC DS3231 (Módulo ZS-042).
- SP3232

## 1.3. Ubicación.

El dispositivo se colocará al lado del bordillo de la acera, a la misma distancia entre las dos líneas laterales amarillas que señalan el aparcamiento, enfocando hacia la entrada del vehículo.

**Figura 1. Ubicación del dispositivo.**

## 1.4. Funcionamiento

El funcionamiento del dispositivo será el siguiente:

A la llegada de un vehículo, se tomará una foto de este. Esta foto será almacenada en una memoria microSD de 2 GB de almacenamiento en formato JPG dentro de una carpeta con nombre V_XXX donde XXX va de 000 a 999. El archivo tendrá como nombre la fecha y la hora a la que ha aparcado el vehículo.

Para que el dispositivo sea capaz de detectar el acercamiento de un vehículo se hace uso de un sensor de ultrasonido (HC-SR04) que tomará cinco medidas, de manera que si la distancia obtenida entre medición y medición es cada vez menor podremos decir que el vehículo está aparcando. Cuando el vehículo de disponer a abandonar el aparcamiento esas distancias son cada vez mayores

A la salida del vehículo se compara el tiempo en el que se obtuvo la imagen de entrada y la hora actual y si es superior a un tiempo determinado (definido en la programación), en este caso 20 minutos, se toma la segunda foto registrando en ella el tiempo de salida y se almacena en la tarjeta microSD. En caso de que no se supere el tiempo límite se eliminará el directorio con la imagen de entrada.

Para que la hora este sincronizada se ha hecho uso de un RTC (DS3231) de alta precisión. Además de funcionar como reloj, utilizaremos una de las dos alarmas de las que dispone para despertar al dispositivo a las 09:00. A las 20:00 el dispositivo entrará en modo sleep, disminuyendo así notablemente el consumo de energía y aumentando la autonomía.

Para una mejor comprensión del funcionamiento global del sistema, en la figura 2 se muestra un diagrama con las acciones que realiza el dispositivo.

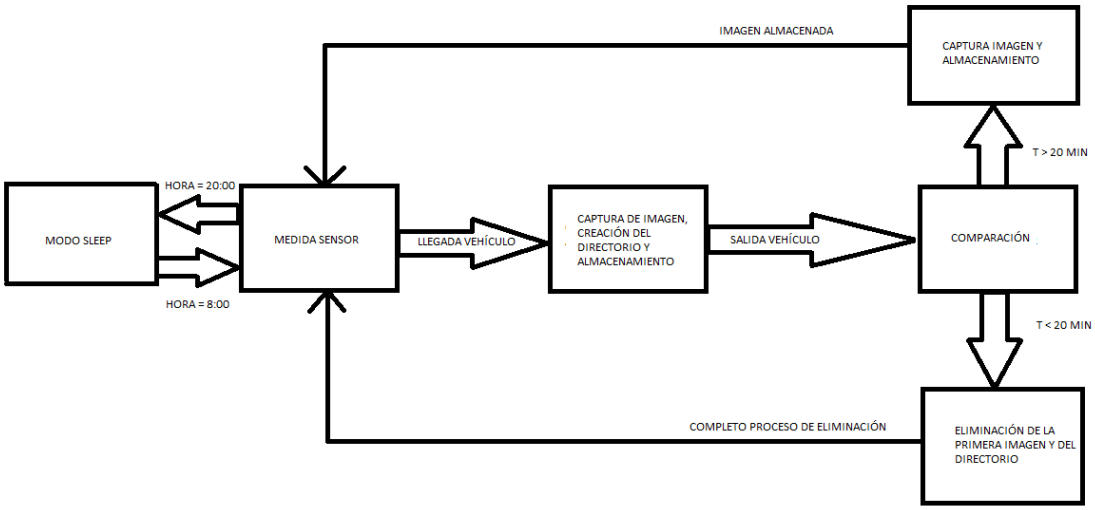



**Figura 2: descripción del funcionamiento.**

## 1.5. Metodología

El procedimiento seguido para la realización de este TFG ha sido el siguiente:

a) Investigación de proyectos similares.
b) Identificación de los componentes necesarios.
c) Lectura de los diferentes datasheets de los diferentes componentes.
d) Prueba y testeo del correcto funcionamiento de cada componente.
e) Montaje completo en protoboard.
f) Medición de consumo.
g) Diseño de PCB.
h) Realización de la programación del dispositivo.
i) Diseño de la carcasa donde se introducirá el dispositivo.
j) Pruebas reales.

## 1.6. Especificaciones del prototipo

Las especificaciones del prototipo diseñado e implementado son las siguientes:

- Se alimenta con 3 pilas AA de 1,5 V en serie con una capacidad de 2800 mAh.
- Autonomía de 5 días y 15 horas.
- El dispositivo toma un vector de medias cada 600 ms aproximadamente. Teniendo un tiempo entre medida y medida de 80 ms.
- Se obtiene una imagen en formato QQVGA (160 x 120 pixeles).
- Peso de la imagen 4 KB.
- Tiempo de almacenamiento de la imagen:
    - Máximo 2 segundos.
    - Típico 1,7 segundos.
    - Mínimo 1,2 segundos.
- Es capaz de almacenar imágenes de hasta 1000 vehículos que han superado el tiempo máximo establecido.

Estas características de funcionamiento son el resultado de una serie de decisiones tomadas tanto a nivel hardware como a nivel software, que se describen y justifican a continuación.

## 1.7. Hardware

A continuación, describiremos los diferentes elementos hardware que se han empleado en el desarrollo del prototipo diseñado.

### 1.7.1. Arduino Pro-mini

Comenzamos con la descripción del Arduino Pro-mini.

#### 1.7.1.1. Descripción

Para la realización de este proyecto hemos empleado preliminarmente un Arduino Pro-mini, figura 3, con un cristal de cuarzo de 8 MHz que opera a 3.3 V.

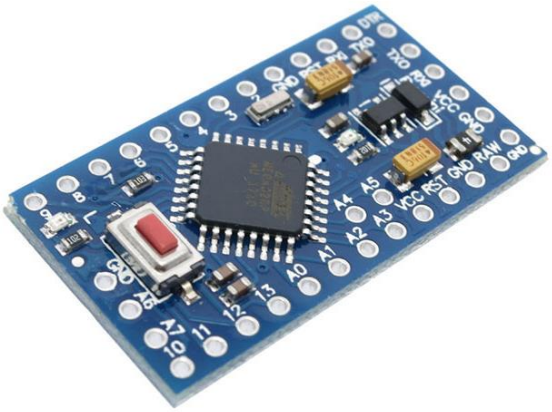


**Figura 3: Arduino Pro-Mini**

Esta placa fue desarrollada por SparkFun Electronics para aplicaciones permanentes donde se dispone de un espacio reducido o donde el espacio ocupado por el dispositivo a desarrollar está limitado.

Arduino Pro-Mini es un microcontrolador basado en ATmega328P. Tal y como se muestra en la **tabla 1** [1], tiene 14 pines de entrada/salida de los cuales 6 pueden ser usamos como salida PWM. Además, dispone de 6 pines de entrada analógica y un botón de reset. Los pines disponen de conexiones Berg (pin headers). En uno de los lados disponemos de 6 pin headers para la conexión con un cable FTDI. En nuestro caso necesitaremos un convertidor (CP2102) para la conexión.

**Tabla 1: Características del Arduino Pro-Mini.**

| Microcontrolador | ATmega328P |
|---|---|
| Alimentación de la placa | 3.35 -12 V |
| Voltaje de funcionamiento de los circuitos | 3.3V |
| Pines digitales de E/S | 14 |
| Pines PWM | 6 |
| UART | 1 |
| SPI | 1 |
| I2C | 1 |
| Pines de entradas analógicas | 6 |
| Interrupciones externas | 2 |
| Corriente DC por pin de E/S | 40 mA |
| Memoria Flash | 32KB of which 2 KB used by bootloader |
| SRAM | 2 KB |
| EEPROM | 1 KB |
| Velocidad de reloj | 8 MHz |
| Temperatura de trabajo | -40$^{o}$C - 105$^{o}$ C |
| Dimensiones | 33 mm x 17 mm |

### 1.7.1.2. Configuración de pines

Los pines de Arduino Pro-Mini se muestran en la figura 4:

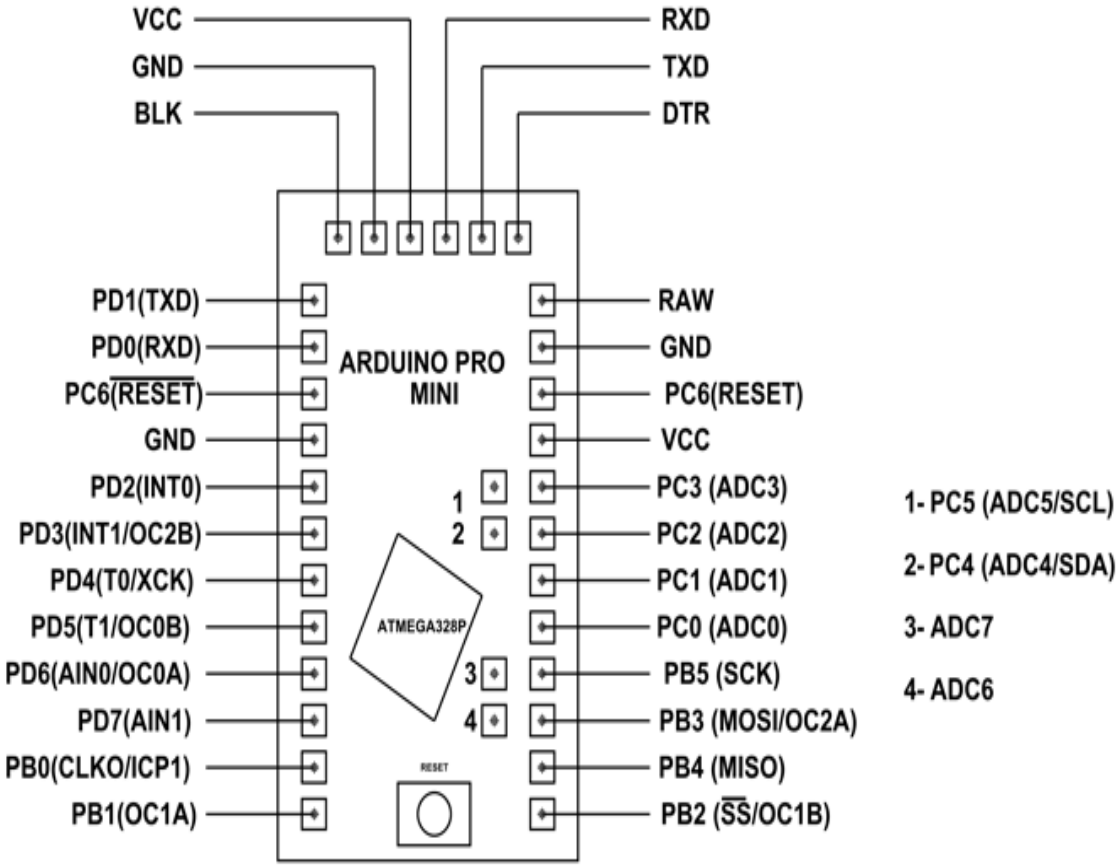



**Figura 4: Configuración de pines.**

### 1.7.1.3. Definición de los pines

A continuación, se muestra una tabla con los grupos de pines y su descripción.

**Tabla 2: pines de Arduino Pro-Mini.**

| Grupo de pines | Nombre | Descripción |
|---|---|---|
| Alimentación | VCC | Conexión a 3,3 V |
| | GND | Conexión a tierra |
| | RAW | Conexión a voltaje no regulado entre 3,3 y 12 V |
| Comunicación | UART (RxD, TxD) | UART (Universal Asynchronous Receiver Transmitter) |
| | SPI (MOSI, MISO, SCK, SS) | SPI (Serial Peripheral Interface) |
| | I2C (SDA, SCL) | I2C (Inter-Integrated Circuit) también conocido como TWI |
| Pines de entrada salida | PD0 to PD7 (8 pins of PORTD) | Aunque estos pines pueden tener otras funciones son considerados pines de entrada salida |
| | PB0 to PB5 (6 pins of PORTB) | |
| | PC0 to PC6 (7 pins of PORTC) | |
| Pines ADC | ADC0, ADC1, ADC2,…ADC7 | Estos pines se usan como entradas analógicas. La resolución del ADC es de 10 bits |
| PWM | OC0A,OC0B,OC1A,OC1B,OC2A,OC2B | A través de estos pines se puede generar una salida PWM (Pulse Width Modulation). Tienen una resolución de 8 bits |
| Reset | Reset | Resetea el controlador |
| Interrupciones externas | T0 y T1 | Estos dos pines han sido diseñados para interrupciones externas por hardware |
| Comparador analógico | AIN0 y AIN1 | Pines conectados a un comparador interno |

### 1.8.1.4 Alimentación.

El microcontrolador puede ser alimentado a través de:

1. Una placa adaptadora con un chip convertidor de RS232 a TTL (el mismo componente usado para la programación del microcontrolador) conectado a los 6 “pin headers”.

2. Un voltaje regulado de 3.3 V.

3. La entrada RAW, en la que podemos introducir un voltaje no regulador siempre que sea mayor que 3.3 V y menor que 12 V.

## 1.7.2. Arduino Mega R3

Finalmente se ha elegido como microcontrolador para realizar este dispositivo un Arduino Mega con ATmega2560. La razón de esta elección se explicará con detalle en la memoria justificativa.

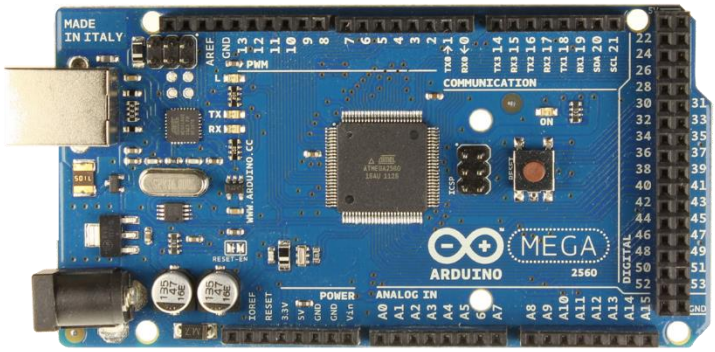



**Figura 5: Arduino Mega**

### 1.7.2.1. Descripción

Como se dice en [2] Arduino Mega 2560 es un microcontrolador basado en ARmega2560. Tiene 54 pines digitales de entrada/salida del cuales 15 pueden ser usados como salidas PWM, 16 pines analógicos y 4 UARTs (puerto hardware serial). Además, tiene un oscilador de cristal de 16 MHz, una conexión USB, una conexión Jack de alimentación, un ICSP header y un botón de reset.

**Tabla 3: características de Arduino Mega.**

| Microcontrolador | ATmega2560 |
|---|---|
| Alimentación recomendada de la placa | 7 -12 V |
| Voltaje de funcionamiento de los circuitos | 5 V |
| Pines digitales de E/S | 54 |
| PWM Pins | 16 |
| UART | 4 |
| SPI | 1 |
| I2C | 1 |
| Pines de entradas analógicas | 16 |
| Interrupciones externas | 5 |
| Corriente DC por pin de E/S | 20 mA |
| Corriente DC para pin de 3.3 V | 40 mA |
| Memoria Flash | 256 KB of which 8 KB used by bootloader |
| SRAM | 8 KB |
| EEPROM | 4 KB |
| Velocidad de reloj | 16 MHz |
| Largo | 101,52 mm |
| Ancho | 53,3 mm |
| Peso | 37 g |

### 1.7.2.2. Configuración de pines

En las figuras 5 y 6 podemos ver donde se sitúan cada uno de los pines a los que tenemos acceso en Arduino Mega. Estas imágenes han sido creadas por Geek Factory.

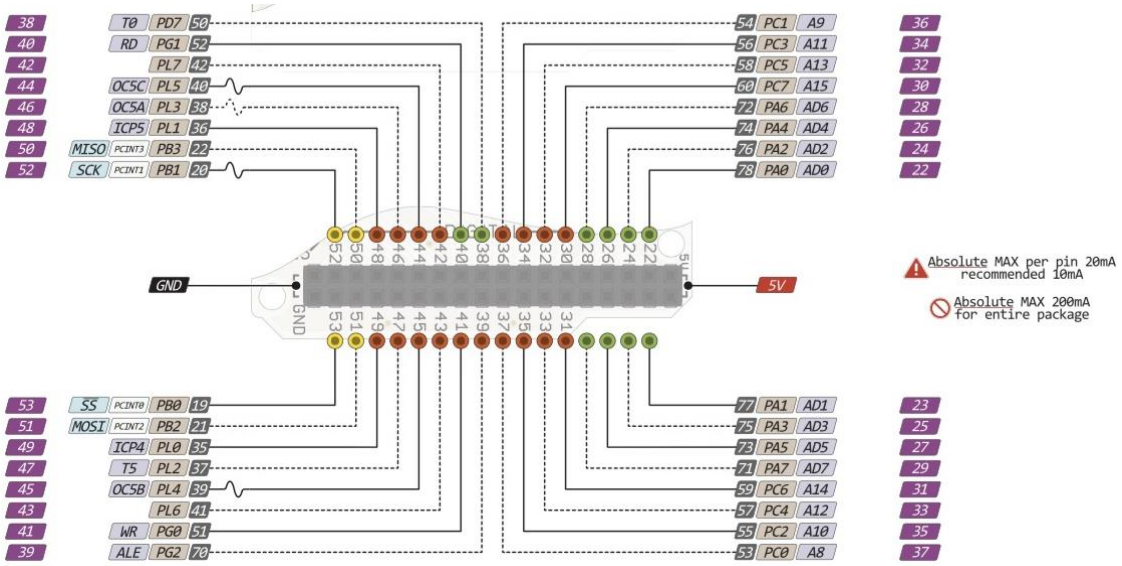



**Figura 6. Pines de Arduino Mega.**

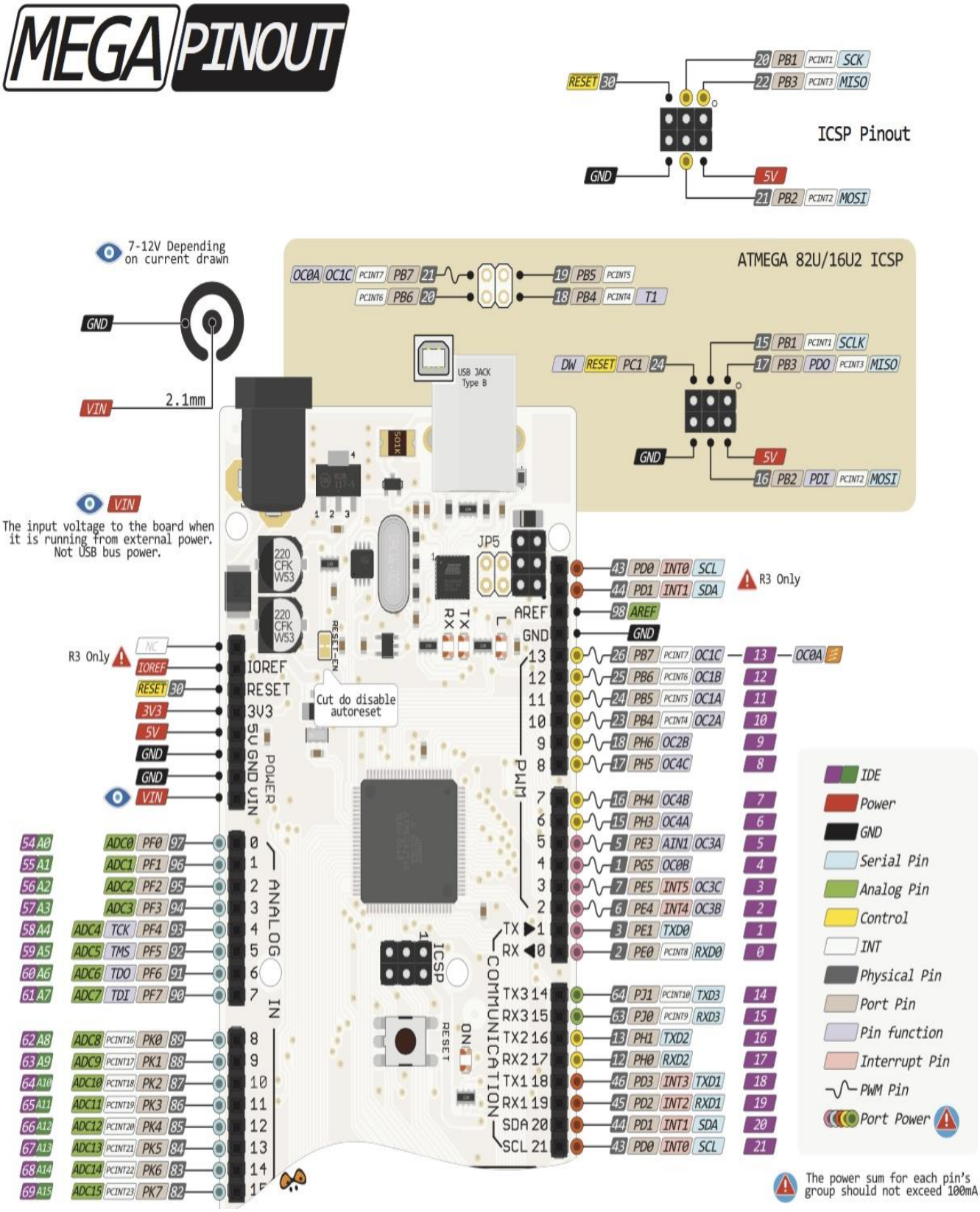



**Figura 7: Pines de Arduino Mega.**

#### 1.7.2.3. Definición de los pines

Como se indica en [2] cada uno de los 54 pines digitales en el Mega pueden utilizarse como entradas o como salidas usando las funciones pinMode(), digitalWrite(), y digitalRead(). Las E/S operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad máxima de 40 mA (20 mA recomendado) y tiene una resistencia interna de pull-up (desconectada por defecto) de 20-50kOhms. Además, algunos pines tienen funciones especializadas:

- Serie: 0 (RX) y 1 (TX), Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX). Usados para recibir (RX) transmitir (TX) datos a través de puerto serie TTL. Los pines Serie: 0 (RX) y 1 (TX) están conectados a los pines correspondientes del chip FTDI USB-to-TTL.
- Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2). Estos pines se pueden configurar para lanzar una interrupción en un valor LOW(0V), en flancos de subida o bajada (cambio de LOW a HIGH(5V) o viceversa), o en cambios de valor. Ver la función attachInterrupt() para más detalles.
- PWM: de 0 a 13. Proporciona una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución (valores de 0 a 255) a través de la función analogWrite().
- SPI: 50 (SS), 51 (MOSI), 52 (MISO), 53 (SCK). Estos pines proporcionan comunicación SPI.
- LED: 13**.** Hay un LED integrado en la placa conectado al pin digital 13.
- I2C: 20 (SDA) y 21 (SCL). Soporte para el protocolo de comunicaciones I2C (TWI).
- AREF. Voltaje de referencia para las entradas analógicas.
- Reset. Suministrar un valor LOW (0V) para reiniciar el microcontrolador.

El Mega tiene 16 entradas analógicas, y cada una de ellas proporciona una resolución de 10bits (1024 valores). Por defecto se mide desde 0V a 5V, aunque es posible cambiar la cota superior de este rango usando el pin AREF y la función analogReference().

#### 1.7.2.4. Alimentación

El Arduino Mega puede ser alimentado vía la conexión USB o con una fuente de alimentación externa. La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V, el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable; si se usan más de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- VIN. La entrada de voltaje a la placa Arduino cuando se está usando una fuente externa de alimentación (en opuesto a los 5 voltios de la conexión USB). Se puede proporcionar voltaje a través de este pin, o, si se está alimentando a través de la conexión de 2.1 mm, acceder a ella a través de este pin.
- 5V. La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de VIN a través de un regulador integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V.
- 3V3. Una fuente de voltaje de 3.3 voltios generada por un regulador integrado en la placa. La corriente máxima soportada 50mA.
- GND. Pin de tierra.

La placa en nuestro caso está alimentada mediante 3 pilas de AA de 1,5 V conectadas al pin Vin. Se volverá sobre ello en el apartado 2.2 dedicado a la alimentación.

### 1.7.3. Sensor de ultrasonido HC-SR04

Para la detección de los vehículos se ha utilizado el sensor de ultrasonido HC-SR04.

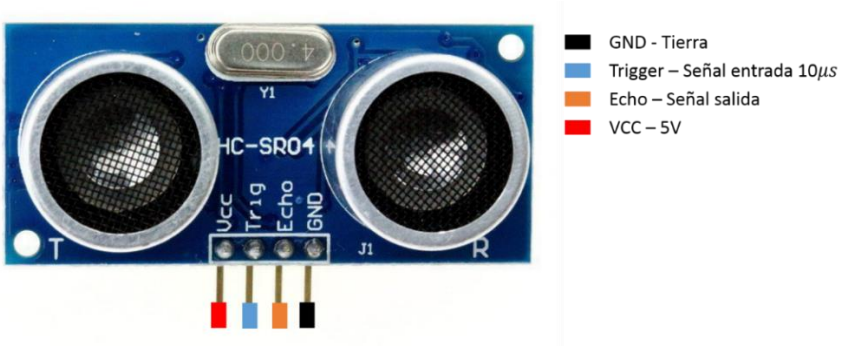



**Figura 8: Sensor de ultrasonido HC-SR04**

Utilizaremos este sensor para medir la distancia a la que se encuentra el vehículo respecto a la ubicación del dispositivo.

#### 1.7.3.1. Funcionamiento.

Tal y como se explica en [3] el funcionamiento del sensor de ultrasonido HC-SR04 se basa en las propiedades magnetostrictivas de determinados materiales. Una lámina de material magnetostrictivo o membrana tiene la propiedad de deformarse mecánicamente y generar ultrasonidos al ser excitada por una corriente eléctrica. El efecto contrario también se produce, es decir, que una vibración mecánica produce una corriente eléctrica. Por tanto, estos sensores emiten una radiación ultrasónica que rebota en los obstáculos del entorno y captan los ecos recibidos.

Los sensores de ultrasonidos se utilizan para medir distancias. A partir de la excitación de la membrana magnetostrictiva con una serie de impulsos eléctricos, se genera un tren de ondas ultrasónicas. Por otro lado, la membrana magnetostrictiva, recibe los ecos de las emisiones recibidas y las transforma en impulsos eléctricos.

Existen típicamente dos tipos de configuraciones: una con sistema de emisor/receptor en el mismo transductor, y otra con un sistema de **e**misor/receptor separados que es el caso de nuestro sensor, el HC-SR04.

### 1.7.3.2. Descripción.

La técnica de medición de distancia que hemos usado es la conocida como *tiempo de vuelo.* Esta técnica consiste en emitir un tren de 8 impulsos a 40 KHz y poner un temporizador en marcha. Cuando se recibe el eco de los impulsos emitidos, el tiempo transcurrido es proporcional al doble de la distancia al obstáculo (tiempo de impacto + tiempo de eco). Si en un determinado margen de tiempo no se ha recibido eco, se considera que no hay obstáculo**.** En la figura 9 se muestra el diagrama de tiempos de las señales. En la programación hay que tener en cuenta que no se puede disparar dos veces el trigger en un tiempo menor de 50 µs debido a los 8 impulsos de 25 µs de duración cada uno que se emiten y se reciben.

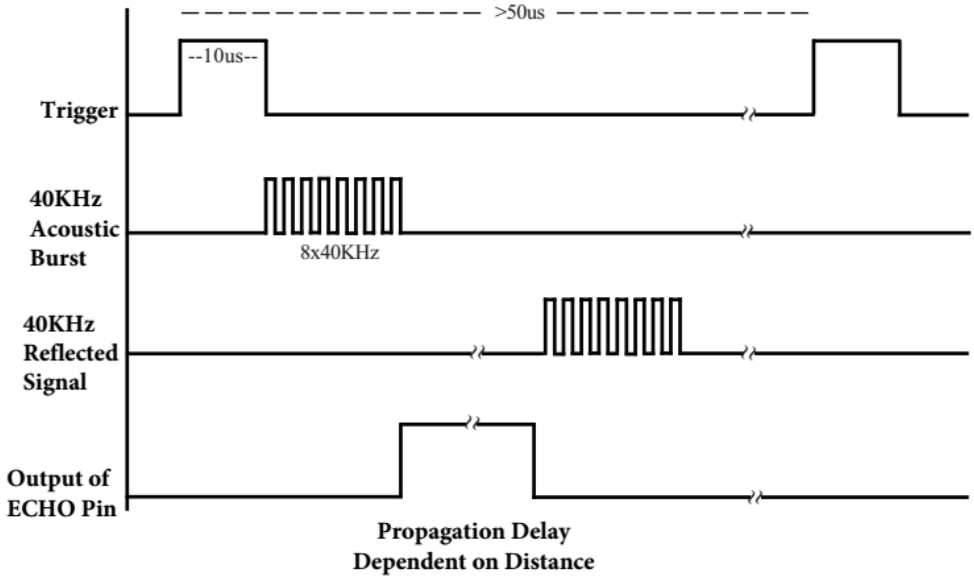



**Figura 9: Diagrama de tiempos del HC-SR04**

Sabiendo la velocidad del sonido a 20 ºC es de 343,2 m/s, midiendo el tiempo entre pulsos podemos estimar la distancia a la que se encuentra el vehículo utilizando la expresión 1.

$$Distancia = \frac{tiempo/2}{velocidad\ sonido} \quad (1)$$

Las variaciones de temperatura tienen cierta influencia, pero para nuestra aplicación podemos despreciarlas ya que se estima que el tiempo de vuelo tiene una dependencia de un 7% con la temperatura para un rango entre 0 y 40 $^{o}$C, tal y como se explica en [3].

Hay que tener en cuenta que para activar el sensor necesitamos como mínimo generar un pulso eléctrico en el pin del trigger (disparador) de al menos 10 µs. Esto se mencionará en el apartado Software.

En cuanto al ángulo de medida efectivo, de acuerdo con la hoja de características del HC-SR04 es de 15$^{o}$, por lo que se podrá detectar el vehículo en un rango de 30$^{o}$ (15$^{o}$ hacia la derecha e izquierda del ángulo cero en la horizontal y lo mismo ocurre verticalmente). Fuera de este rango, los valores obtenidos de tiempo recorrido por la onda serán erróneos.

### 1.7.3.3. Conexionado.

A continuación, se muestra el conexionado para el HC-SR04.

I. Para Arduino Pro-Mini.

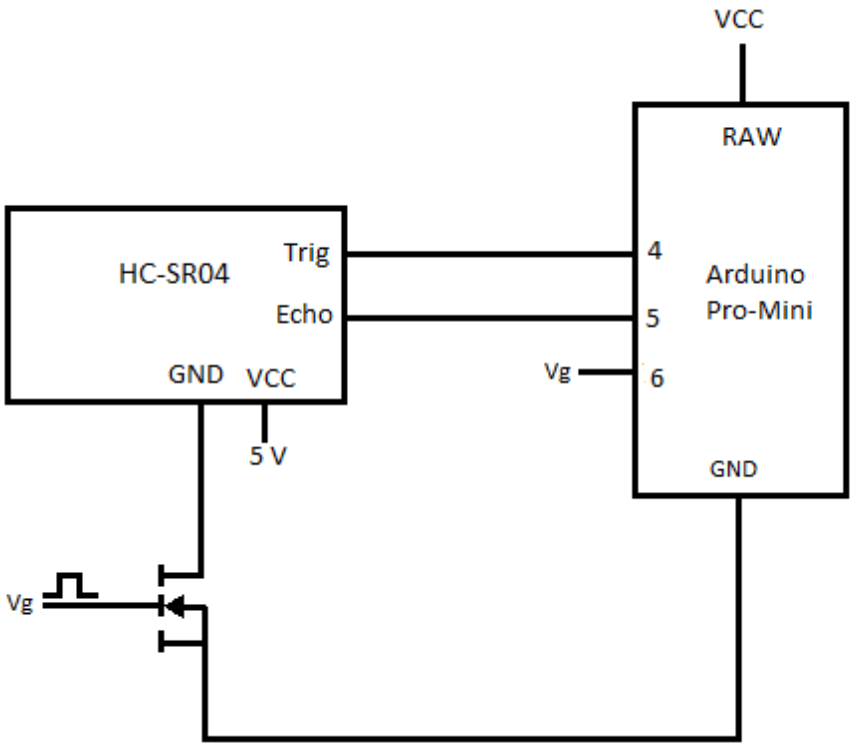



**Figura 10: Conexionado HC-SR04 con Arduino Pro-Mini**

II. Para Arduino Mega.

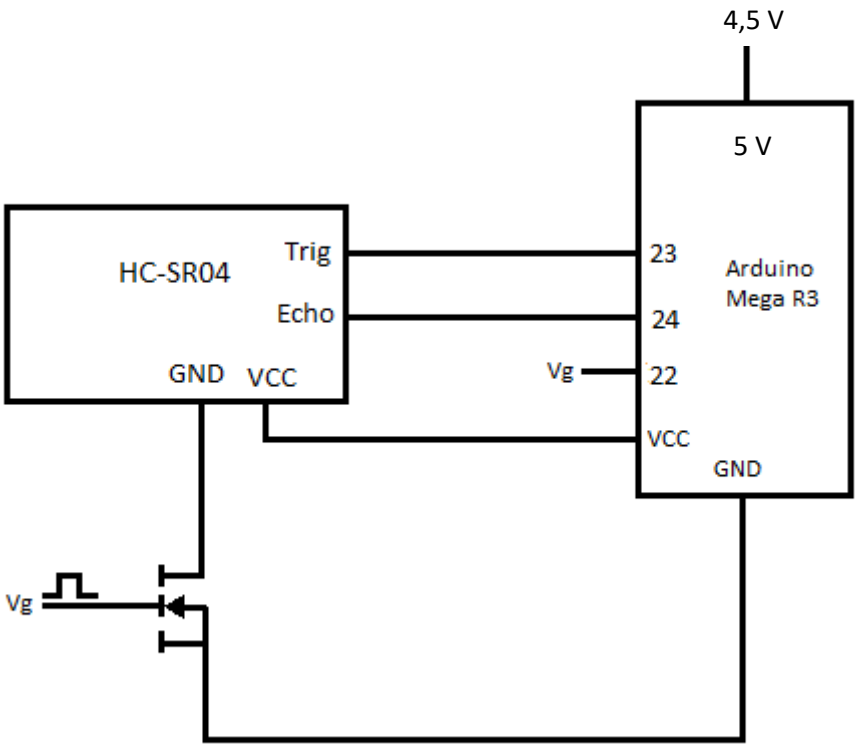



**Figura 11: conexionado HC-SR04 con Arduino Mega**

### 1.7.4. Cámara VC0706

Para poder registrar la entrada y salida de vehículos del aparcamiento hemos utilizado una cámara, concretamente la VC0706 de la marca VDUNSKY cuyo aspecto se muestra en la **f**igura 12

**Figura 12. Cámara VC0706.**

#### 1.7.4.1. Descripción.

Se trata de un tipo de cámara que está diseñada con la idea de ser utilizada para la videovigilancia, aunque puede ser perfectamente utilizada en nuestro proyecto. Nosotros no utilizaremos la salida de video, simplemente capturaremos una imagen y la almacenaremos en la tarjeta SD.

Esta cámara utiliza un sensor de tipo CMOS de 1/4 de pulgada que vienen a ser 4.5 mm de diagonal, 3.6 mm de ancho y 2.7 mm de alto, teniendo un factor de recuadro de 10.81.

En cuanto a la lente utilizamos una de 2 mm de distancia focal con un FOV (field of view) que cubre desde los 60 hasta los 120 grados.

La cámara VC0706 tiene una resolución de 0.3 MP ya que permite obtener imágenes en VGA (640 x 480 píxeles). También se pueden obtener imágenes en QVGA (320 x 240 pixeles) o en QQVGA (160 x 120 píxeles). Nosotros utilizaremos el formato QQVGA ya que con él podemos captar perfectamente la matrícula del vehículo, que lo único que necesitamos en la aplicación que nos ocupa. Además, cuanto menor es la resolución menor es el peso de la imagen. Por tanto, ocupará un menor espacio en la memoria flash conectada y será mucho más rápido tanto la transmisión de los datos a la hora de leerlos del buffer que tiene el módulo de la cámara como la escritura en la tarjeta microSD.

#### 1.7.4.2. Funcionamiento

Los sensores CMOS son uno de los sensores utilizados en las cámaras, aparte de los CCD. Los CMOS son en realidad sensores de píxeles activos (active pixel sensor, APS) aunque, como este tipo de sensor detecta la luz basado en la tecnología CMOS, es conocido como Sensor CMOS.

Tal y como se menciona en [4], el APS, al igual que el sensor CCD, se basa en el efecto fotoeléctrico. Está formado por numerosos fotositos, uno para cada píxel, que producen una corriente eléctrica que varía en función de la intensidad de luz recibida. En el CMOS, a diferencia del CCD, se incorpora un amplificador de la señal eléctrica en cada fotosito y un conversor digital en el propio chip.

Como se menciona en [5], para reconocer los colores de la imagen captada se utiliza un filtro de Bayer, que es un tipo de matriz de filtros, rojos, verdes, azules (RGB) que se sitúa sobre el sensor CMOS para hacer llegar a cada fotodiodo la información de luminosidad correspondiente a una sección de los distintos colores primarios. Interpolando las muestras de cada cuatro fotodiodos vecinos se obtiene un pixel de color.

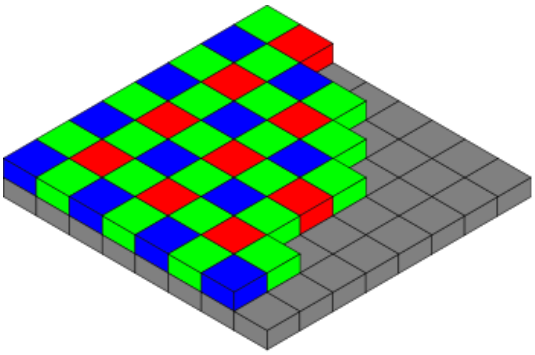


**Figura 13: mosaico de Bayer**

El mosaico de Bayer está formado por un 50% de filtros verdes, un 25% de rojos y un 25% de azules. La razón de que se use mayor cantidad de puntos verdes es que el ojo humano es más sensible a ese color.

El funcionamiento de la cámara a la hora de adquirir una imagen se describe en la figura 13. Dado que se trata de una cámara de CCTV, se captura la luz y se convierte en señales digitales,

pero para obtener una imagen es necesario procesarla. Esto se hace a través de un DSP. El módulo utiliza el DSP VC0706PREB de VIMICRO.

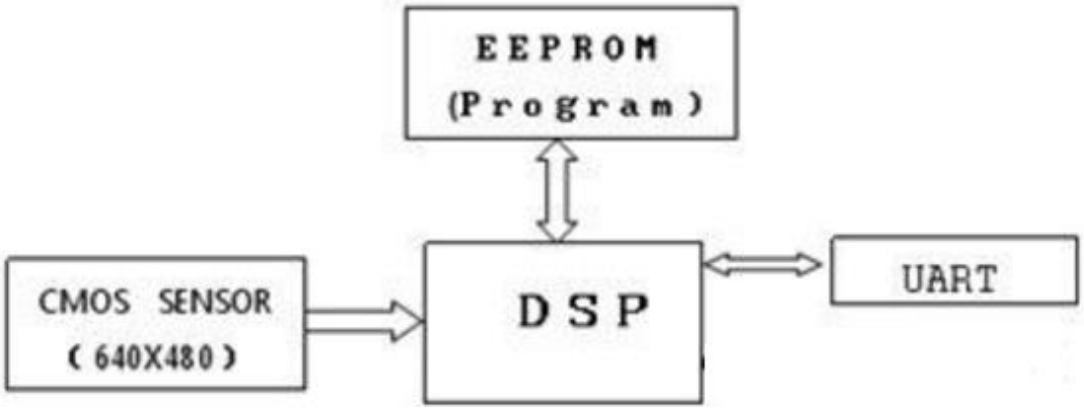



**Figura 14: adquisición de la imagen**

El DSP (Digital Signal Processor) es un sistema que consta de un microprocesador, que tiene una serie de instrucciones y un software. Estas instrucciones y este software están grabados en una memoria EEPROM que forma parte del módulo VC0706 que estamos utilizando. El DSP regula y controla todo el sistema electrónico de la cámara y se encarga de procesar la imagen que le llega desde el sensor CMOS para posteriormente poder transmitir la imagen a través de la UART.

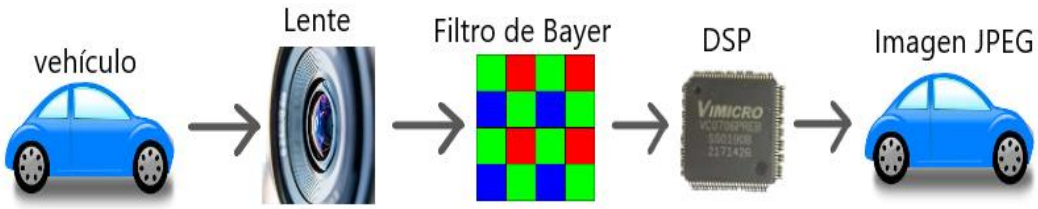



**Figura 15: descripción de la obtención de la imagen**

### 1.7.4.3. Comunicación

Dado que esta cámara está pensada para videovigilancia, envía los datos utilizando el protocolo RS232, aunque en realidad el DSP se comunica con señales CMOS. El problema es que solo tenemos acceso al pin la comunicación RS232 y Arduino no trabaja con este tipo de señales. Por eso hemos optado por utilizar convertidor el SP3232E para volver a transformar la señal RS232 en una señal TTL para poder comunicarnos con nuestro microcontrolador (Arduino) a través de un puerto serie. El esquemático del MAX3232 que tiene implementado el módulo de la cámara se muestra en la figura 15.

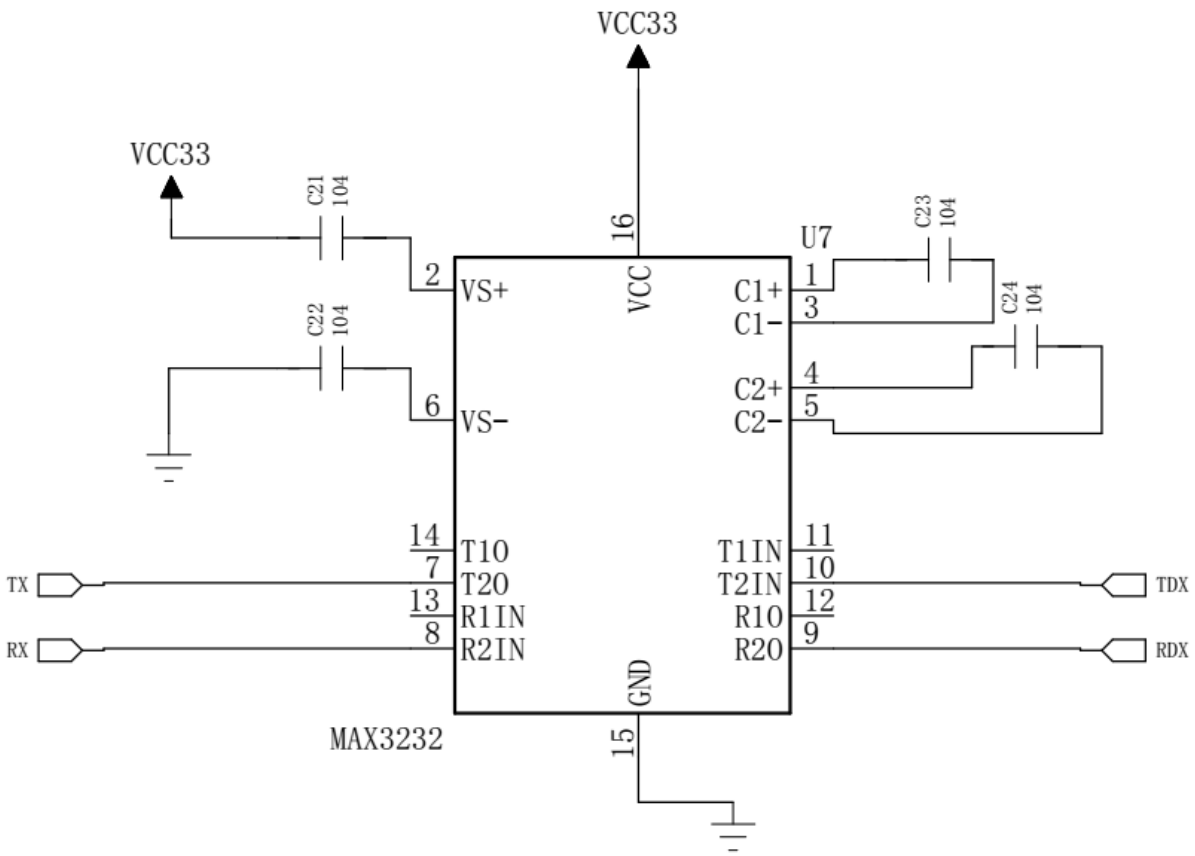



**Figura 16: esquemático MAX3232**

Los pines TX, y RX (señales en RS232) son las salidas del MAX3232 y TDX y RDX (señales TTL 3.3 V) son las entradas que provienen del VC0706PREB.

La cámara además nos permite un baud rate entre los 9600 bps y los 115200 bps. El baud rate de nuestra cámara lo hemos fijado en 38400 bps.

### 1.7.4.4. Conexionado

A continuación, se muestra el conexionado de la cámara con el SP3232.

I. Para Arduino Pro-Mini.

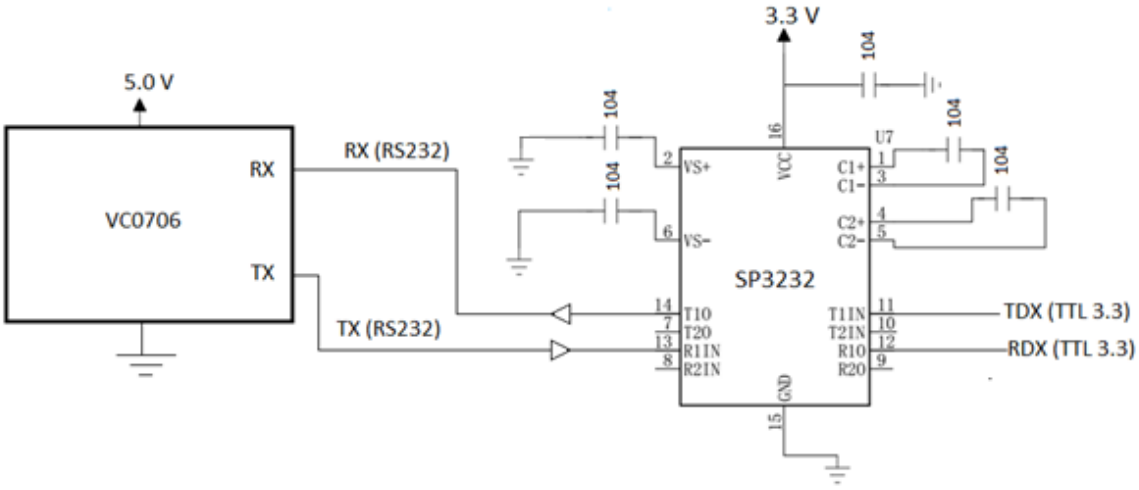



**Figura 17: conexionado cámara con SP3232 para Arduino Pro-Mini.**

II. Para Arduino Mega.

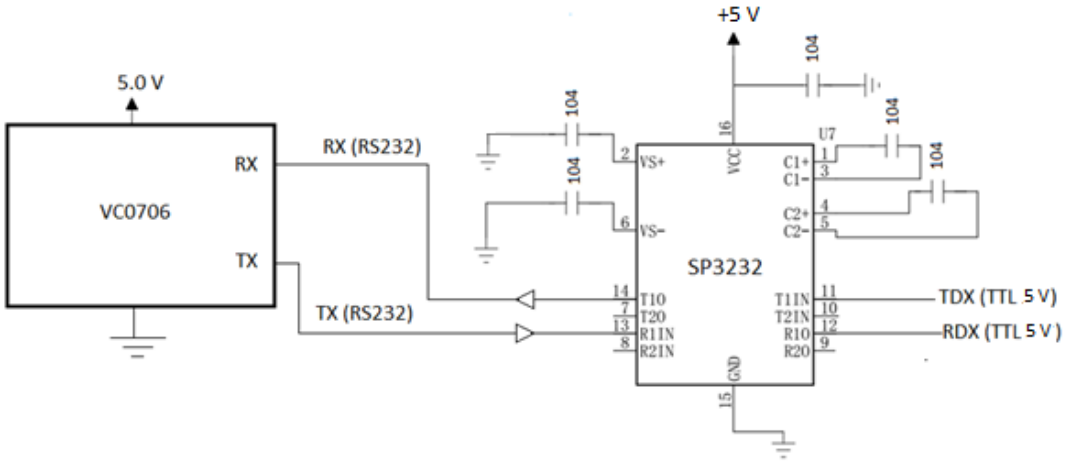



**Figura 18: conexionado VC0706 con SP3232 para Arduino Mega.**

### 1.7.5. SP3232E

Tal y como se mencionó en el apartado 1.7.3, se utiliza un SP3232E que es un circuito integrado que nos permite transformar una señal RS232 a una señal TTL, de forma que podamos comunicar adecuadamente el módulo de la cámara con el microcontrolador.

En las señales RS232 se identifica como 1 lógico niveles de voltaje entre -3 y -25 V y como 0 lógico los valores ente +3 y +25 V. Los valores entre -3 y +3 V quedan como no definidos. En cambio, en una señal TTL con VCC de 3.3 V se considera alto desde 2.4 V hasta VCC y bajo desde 0 hasta 0.8 V.

Este dispositivo puede ser alimentado a partir de 3.3 V hasta 5 V y su velocidad máxima de transferencia de datos es de 235 Kbps, que es mucho mayor a la utilizada por la cámara.

En la figura 19, obtenida de la hoja de característica del integrado, podemos ver la configuración de pines para el SP3232E.

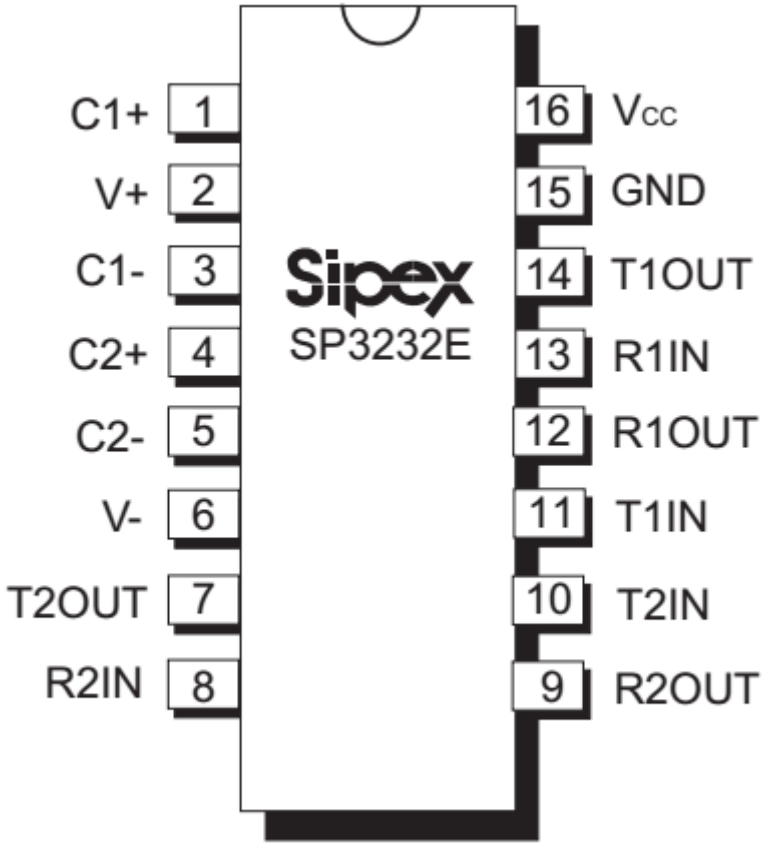



**Figura 19: pines SP3232**

En la tabla 4 podemos ver la definición de sus pines.

**Tabla 4: definición de pines del SP3232.**

| Nombre | Función | Número Pin |
|---|---|---|
| C1+ | Conexión positiva del condensador C1 del doblador de voltaje | 1 |
| V+ | +5.5 V generados por condensador de carga | 2 |
| C1- | Terminal negativa del condensador doblador de voltaje | 3 |
| C2+ | Terminal positiva del condensador inversor de voltaje | 4 |
| C2- | Terminal negativa del condensador inversor de voltaje | 5 |
| V- | -5.5 V generados por el condensador de carga | 6 |
| T1OUT | Salida del driver RS232 | 14 |
| T2OUT | Salida del driver RS232 | 7 |
| R1IN | Entrada RS232 | 13 |

| | | |
|---|---|---|
| R2IN | Entrada RS232 | 8 |
| R1OUT | TTL/CMOS salida | 12 |
| R2OUT | TTL/CMOS salida | 9 |
| T1IN | TTL/CMOS entrada | 11 |
| T2IN | TTL/CMOS entrada | 10 |
| GND | Tierra | 15 |
| Vcc | 3.3 V a 5.5 V | 16 |

A continuación, se muestra el conexionado del SP3232 al microcontrolador.

I. Para Arduino Pro-Mini.

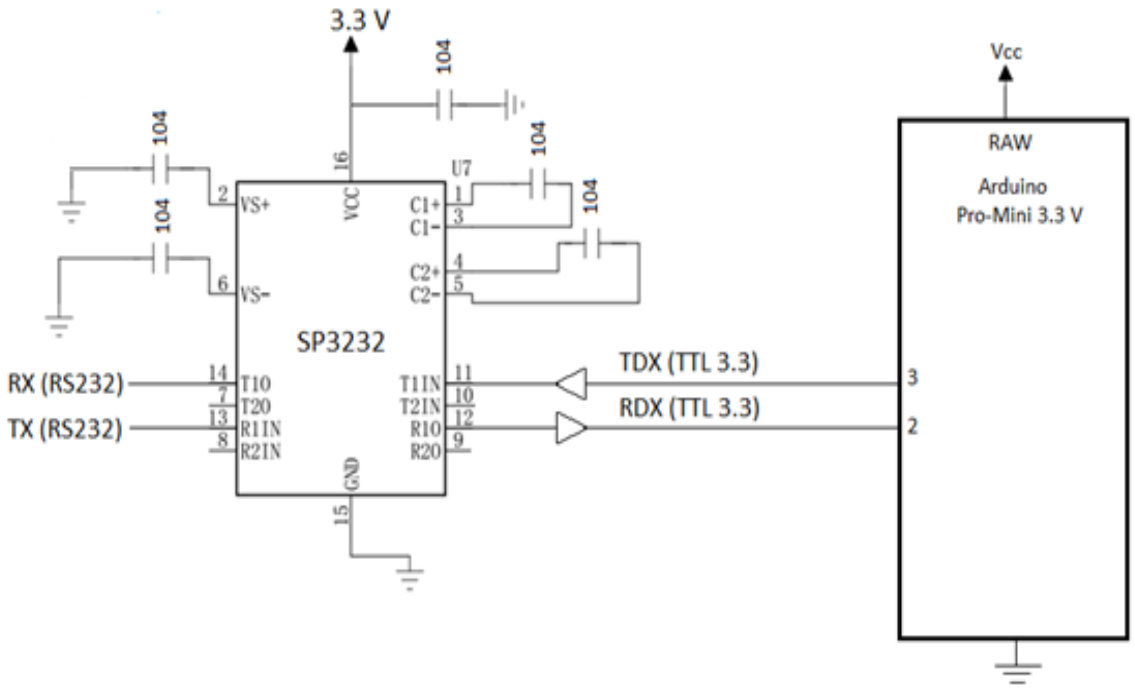



**Figura 20: conexionado SP3232 con Arduino Pro-Mini.**

II. Para Arduino Mega.

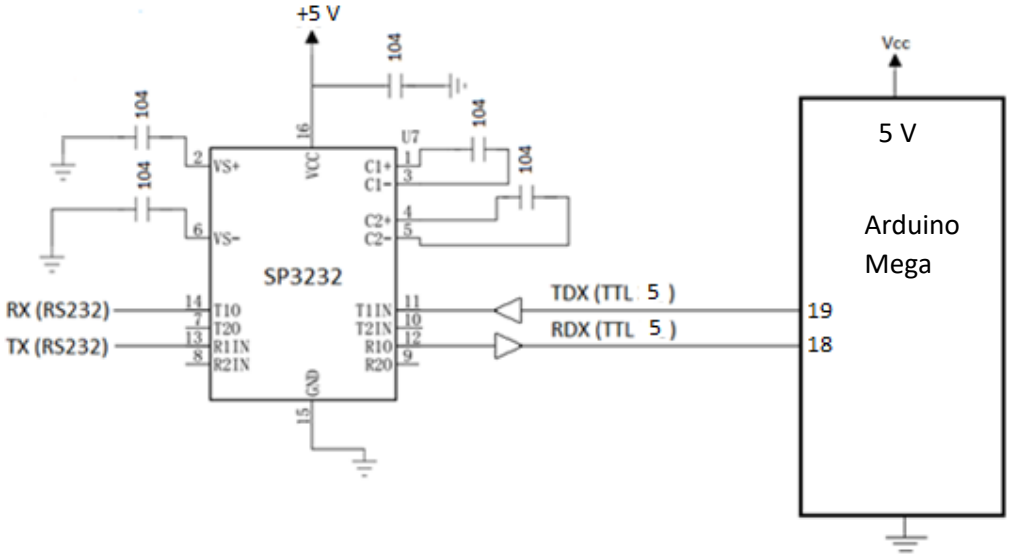



**Figura 21: conexionado SP3232 con Arduino Mega**

### 1.7.6. Módulo Bluetooth HC-06.

Se plantea el uso del módulo bluetooth HC-06 para poder obtener transmitir de manera inalámbrica las imágenes al smartphone de la policía. De esta manera, la policía local del municipio solo tendría que acercarse al dispositivo, conectar el bluetooth y esperar hasta recibir todas las imágenes. De esta manera se evita tener que extraer la tarjeta microSD del dispositivo, insertarla en el móvil y cortar los archivos a la carpeta correspondiente en la memoria interna del smartphone.

Como se menciona en [6], Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) creado por Bluetooth Special Interest Group, Inc. que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM (bandas reservadas para el uso no comercial de la radiofrecuencia) de los 2.4 GHz.

Tal y como se indica en [7], el estándar bluetooth, del mismo modo que wifi, utiliza la técnica FHSS (Frequency Hopping Spread Spectrum, espectro ensanchado por saltos de frecuencia), que consiste en dividir la banda de frecuencia de 2.402 - 2.480 GHz en 79 canales (denominados "saltos") de 1 MHz de ancho cada uno y, después, transmitir la señal utilizando una secuencia de canales que sea conocida tanto para la estación emisora como para la receptora. Por lo tanto, al cambiar de canales con una frecuencia de 1600 veces por segundo, el estándar bluetooth puede evitar la interferencia con otras señales de radio.

El estándar bluetooth se basa en el modo de operación maestro/esclavo. El establecimiento de una conexión entre dos dispositivos bluetooth sigue un procedimiento relativamente complicado para asegurar un cierto nivel de seguridad: Modo pasivo > Solicitud > Búsqueda de puntos de acceso > Paginación > Sincronización con los puntos de acceso > Descubrimiento del

servicio del punto de acceso > Creación de un canal con el punto de acceso > Emparejamiento mediante el PIN (seguridad) > Utilización de la red.

Durante el uso normal, un dispositivo funciona en "modo pasivo", es decir, que está escuchando la red. El establecimiento de una conexión comienza con una fase denominada solicitud, durante la cual el dispositivo maestro envía una solicitud a todos los dispositivos que encuentra dentro de su rango, denominados puntos de acceso. Todos los dispositivos que reciben la solicitud responden con su dirección. El dispositivo maestro elige una dirección y se sincroniza con el punto de acceso mediante una técnica denominada paginación, que principalmente consiste en la sincronización de su reloj y frecuencia con el punto de acceso.

El módulo HC-06 que podemos ver en la figura 22 es un módulo mediante el cual podemos enviar datos desde Arduino o recibir datos desde un dispositivo externo, como puede ser un Smartphone, a través de bluetooth.

**Figura 22: módulo Bluetooth HC-06**

Este dispositivo, que solo puede ser utilizado como esclavo, tiene una tasa de transferencia de 2,1 Mbit/s.

En un primer momento se pensó en transmitir las imágenes de manera inalámbrica. Se trató en utilizar el protocolo de Bluetooth para poder enviar las imágenes almacenadas en la tarjeta de memoria flash (tarjeta microSD) a dispositivos Android.

Para ello es necesario el desarrollo de una aplicación para Android capaz de almacenar las imágenes transmitidas de Arduino a través de bluetooth.

Inicialmente, se pensó en trabajar a muy alto nivel para poder desarrollar este tipo de aplicación, pero nos encontramos con que utilizando un lenguaje de bloques solo podíamos utilizar ficheros de texto (.txt o .dat). Además, es necesario saber con anterioridad el número de archivos que deseamos transmitir, para así tener definido el número de archivos que se van a guardar en la memoria del dispositivo Android. Esto hizo que optásemos por una programación a bajo nivel. Se utilizó como entorno de desarrollo Android Studio y se tomó como base para programar la aplicación una aplicación de la Play Store, BlueTerm, que es de código abierto. Después de varias semanas intentando desarrollar esta aplicación, se desistió debido a su complejidad.

### 1.7.7. Módulo microSD.

Para poder almacenar las imágenes captadas por la cámara se hace uso de un módulo microSD como el que se muestra en la figura 23.

**Figura 23: módulo microSD**

#### 1.7.7.1. Descripción.

El módulo (adaptador de tarjeta MicroSD) permite la comunicación con una tarjeta microSD, ya sea para leer datos de ella o para escribir en ella. Tiene unas dimensiones de 42 mm de largo y 24 mm de ancho y puede soportar tarjetas microSD de hasta 2 GB y tarjetas micro SDHC (High Capacity) de hasta 32 GB.

La tensión de funcionamiento es de 3.3 V, pero el módulo trae incorporado un regulador de tensión que permite alimentar el módulo a 5 V para luego operar a 3.3 V.

Además, el módulo microSD tiene implementado un circuito regulador de nivel (regulador de bajo diferencial (LDO)) permitiendo así, que las señales de bus SPI de 5 V sean convertidas a señales de 3.3 V.

En la figura 24 se puede ver al regulador de tensión y el circuito convertidor de señal.

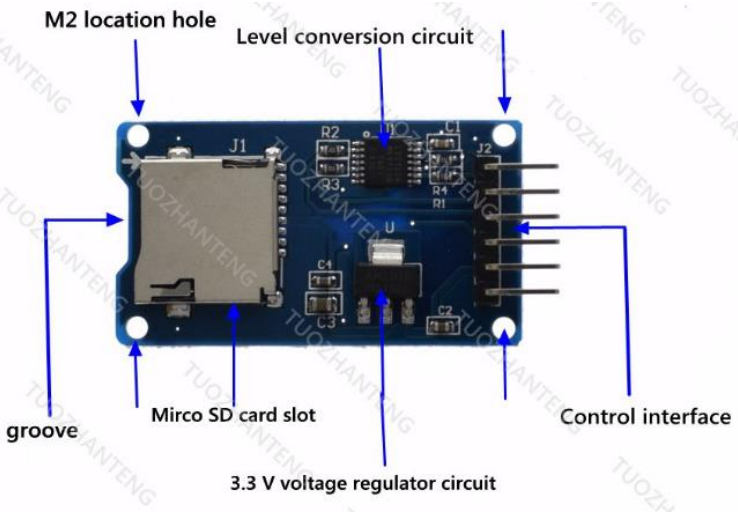



**Figura 24: componentes del módulo microSD.**

#### 1.7.7.2. Comunicación.

Como parte del desarrollo de este TFG ha sido necesario utilizar bus SPI con el cual conseguimos comunicar nuestra tarjeta microSD con Arduino.

El bus SPI (Serial Peripheral Interface), tal y como se explica en [8], fue desarrollado por Motorola en 1980. Tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro (Arduino Pro-Mini) puede iniciar la comunicación con uno o varios esclavos (slave), y enviar o recibir datos de ellos. En este caso solo está conectado a través de SPI este módulo. Los dispositivos esclavos no pueden iniciar la comunicación, ni intercambiar datos entre ellos directamente.

La comunicación de datos entre maestro y esclavo se realiza en dos líneas independientes, una del maestro a los esclavos, y otra de los esclavos al maestro. Es decir, tiene una comunicación Full Duplex, el maestro puede enviar y recibir datos simultáneamente.

Otra característica de SPI es que es un bus síncrono. El dispositivo maestro proporciona una señal de reloj (SCK), que mantiene a todos los dispositivos sincronizados. Esto reduce la complejidad del sistema frente a los sistemas asíncronos.

Por tanto, el bus SPI requiere un mínimo de 3 líneas:

- MOSI (Master-out, slave-in) para la comunicación del maestro al esclavo.
- MISO (Master-in, salve-out) para la comunicación del esclavo al maestro.
- SCK (Signal Clock), que es la señal de reloj enviada por el maestro.

Además, es necesario una línea adicional llamada SS (Slave Select) para cada dispositivo esclavo conectado, para poder así indicar con que dispositivo se va a iniciar la comunicación. En la figura 25 podemos ver una imagen con las líneas de comunicación necesarias.

**Figura 25. Conexión entre maestro y esclavo. [8]**

Por defecto, el maestro mantiene en estado HIGH todas las líneas SS. Cuando el maestro quiere establecer comunicación con el esclavo, pone a LOW la línea SS correspondiente, lo que indica al esclavo que debe iniciar la comunicación.

En cada pulso de la señal de reloj, en el flanco de subida, el dispositivo maestro envía un bit al esclavo y a la vez recibe un bit del esclavo seleccionado. En la figura 26 podemos ver las señales MISO, MOSI, SCK tanto del maestro como del esclavo enviando una ‘a’ y recibiendo una ‘z’.

**Figura 26: ejemplo de comunicación SPI. [8]**

### 1.7.7.3. Conexión.

A continuación, se muestra el conexionado del módulo microSD al microcontrolador.

I. Para Arduino Pro-Mini.

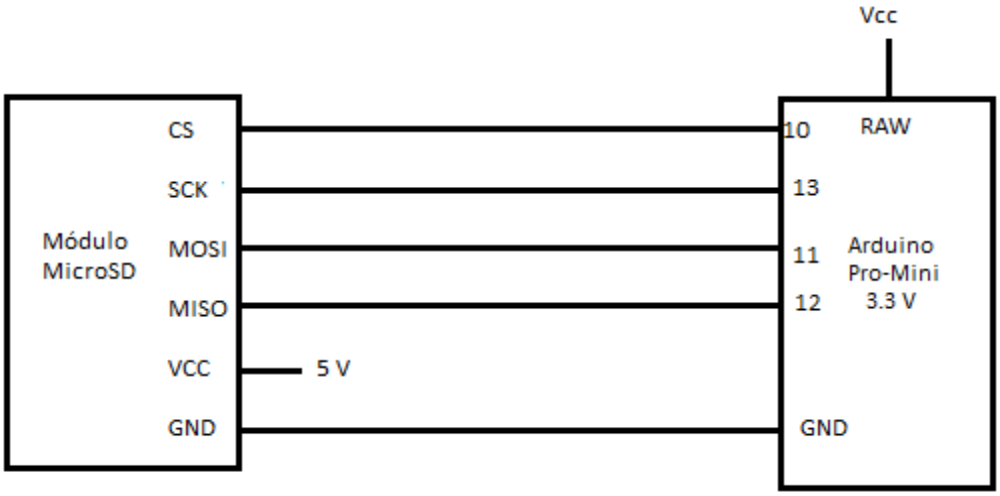



**Figura 27: conexionado con Arduino Pro-Mini.**

II. Para Arduino Mega.

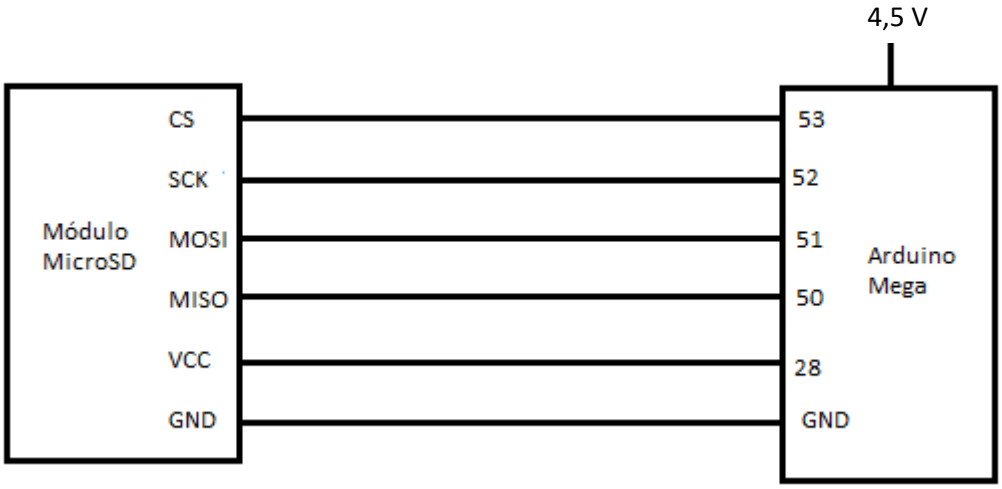



**Figura 28: conexionado del módulo microSD con Arduino Mega.**

### 1.7.8. Tarjeta microSD.

Se ha utilizado una tarjeta microSD de 2 GB de almacenamiento. Esta tarjeta se coloca en la ranura del módulo microSD.

### 1.7.9. Conversor CP2102.

Se ha hecho uso del módulo conversor CP2102, que lleva nombre del chip CP2102, que es un conversor que nos permite convertir las señales de RS232 (USB) a TTL. Con él hemos programado el Arduino Pro-Mini.

**Figura 29: conversor CP2102.**

### 1.7.9.1. Descripción.

El módulo nos permite conectarnos a un puerto USB de un PC a través de un puerto macho USB 2.0 (12 Mbps de velocidad) convirtiendo las señales de comunicación de RS232 a TTL. El módulo tiene 6 pines: dos de alimentación (3.3 V y 5 V) que sirven para alimentar al Arduino Pro-Mini (en nuestro caso solo utilizaremos el pin de 3.3 V), uno de tierra (GND), dos de comunicación (TX y RX) y el DTR que sirve para resetear el Arduino al momento de programarlo, ya que justo antes de cargar el programa es necesario resetear el microcontrolador.

Además, dispone de 3 LEDs:

- TXD. Se enciende cuando se transmiten datos desde el USB.
- RXD. Se enciende cuando se reciben datos desde el Arduino.
- Un LED que nos indica si el módulo está alimentado.

El buffer del CP2102 tiene una capacidad de 640 bytes, siendo la velocidad máxima de comunicación 1,5 Mbps.

Las dimensiones de la PCB son de 26,5 mm de largo x 15,6 mm de ancho.

### 1.7.9.2. Conexión.

A continuación, se muestra el conexionado del CP2102 a Arduino Pro-Mini.

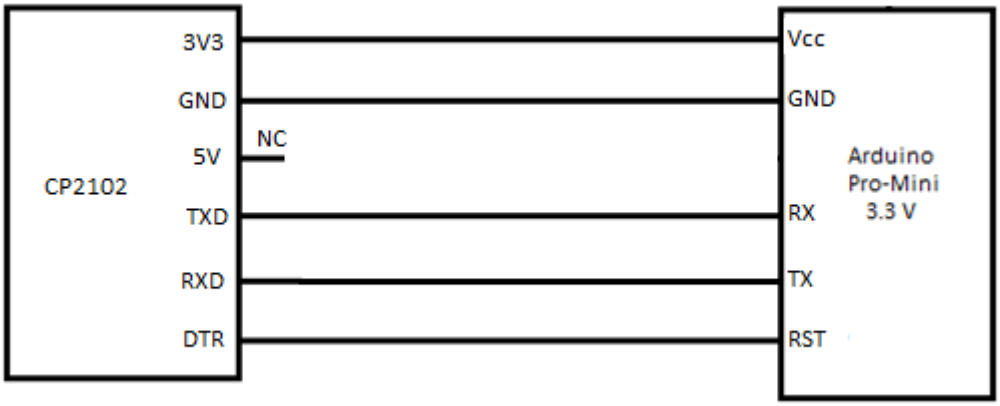



**Figura 30: conexionado del CP2102 con Arduino Pro-Mini.**

### 1.7.10. RTC

Se hace uso del módulo ZS-042 que incorpora el integrado DS3231, un RTC (Real Time Clock) que nos permite registrar con una alta precisión la hora de entrada y salida de los vehículos del aparcamiento. Además, el módulo incorpora una pequeña memoria no volátil (EEPROM) de 32 Kbits (AT24C32) de la cual no se hace uso en este diseño. En la figura 31 podemos ver el módulo ZS-042.

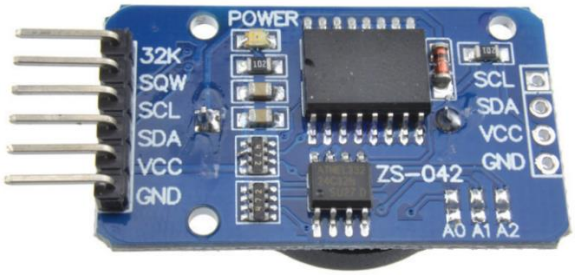



**Figura 31: RTC (ZS-042).**

#### 1.7.10.1. Funcionamiento

EL RTC funciona con un oscilador de cuarzo, material que se caracteriza por su constancia y estabilidad en la oscilación. De esta forma, habiendo fijado previamente la hora en el RTC, a partir del número de oscilaciones este dispositivo es capaz de proporcionarnos el tiempo transcurrido, conociendo así la hora actual.

#### 1.7.10.2. Descripción

El DS3231 es un RTC de bajo coste, extremadamente preciso, con un oscilador de cristal integrado compensador por temperatura (TCXO). El dispositivo incorpora una entrada de batería permitiendo así mantener la cuenta con precisión a pesar de desconectar la fuente de alimentación principal. La integración del cristal mejora la precisión a largo plazo del dispositivo, así como reduce el número de componentes de la línea de producción.

El RTC mantiene información de segundos, minutos, horas, días, fecha, mes y año. La fecha al final del mes es ajustada automáticamente para meses que tienen menos de 31 días, incluyendo correcciones para los años bisiestos. El reloj opera tanto en formato de 24 horas como de 12 horas, teniendo un indicador para saber si nos encontramos en AM o PM. Además, el DS3231 nos proporciona dos alarmas y una señal cuadrada programable. La dirección y los datos son transferidos al reloj por el bus bidireccional $I^2C$.

Además, se monitorea el estado de VCC para detectar fallos de potencia, proporcionando una salida de reinicio. Además, el primer PIN (RST) es monitoreado como un pulsador de entrada para generar externamente una señal de reset. En nuestro módulo este pin no está conectado.

Cuando el microcontrolador se desconecta de la alimentación, el RTC sigue funcionando ya que dispone de una pila de botón (LIR2032) que le permitirá seguir llevando la cuenta de la hora.

### 1.7.10.3. Comunicación

Como se mencionó en el apartado descripción (1.7.10.2), la comunicación se realiza a través del protocolo $I^2C$.

Tal y como se menciona en [9], el estándar $I^2C$ (Inter-Integrated Circuit) fue desarrollado por Philips en 1982 para la comunicación interna de dispositivos electrónicos en sus artículos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.

Este estándar también es conocido como TWI (Two Wired Interface). Esto se debe a que había una patente, que caducó en el año 2006. Actualmente no hay restricción sobre el uso del término $I^2C$.

El bus $I^2C$ requiere dos cables para su funcionamiento, uno para la señal de reloj (SDL) y otro para el envío de datos (SDA).

En la figura 32 podemos ver como se conectarían varios esclavos al maestro a través del bus $I^2C$.

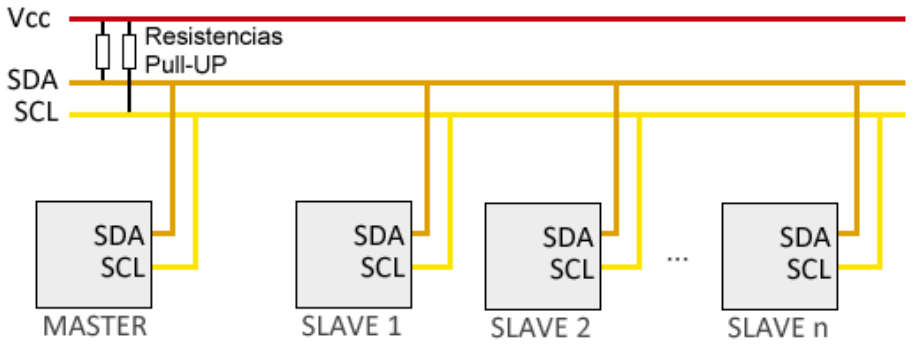



**Figura 32:conexión al bus I2C. [9]**

En el bus cada dispositivo dispone de una dirección que se emplea para acceder a cada uno de ellos de forma individual. Esta dirección puede ser fijada por hardware o por software. Por tanto, cada dispositivo conectado al bus tiene una dirección única. Si se utilizan varios dispositivos similares se tendrá que cambiar la dirección o implementar un bus secundario.

El bus tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro inicia la comunicación (síncrona) y proporciona una señal de reloj que mantiene sincronizados a todos los dispositivos del bus.

Se utilizan resistencias Pull-up de las líneas a VCC. Estas resistencias, de 4,7 kΩ, están ya integradas en el módulo ZS-042.

El protocolo de comunicación consta de:

- 7 bits corresponden a la dirección del dispositivo esclavo con el que queremos comunicar.
- Un bit restante indica si queremos enviar o recibir información.
- Un bit de validación.
- Uno o más bytes de datos enviados o recibidos del esclavo.
- Un bit de validación.

En la figura 33 podemos ver una imagen con un ejemplo de esta comunicación.

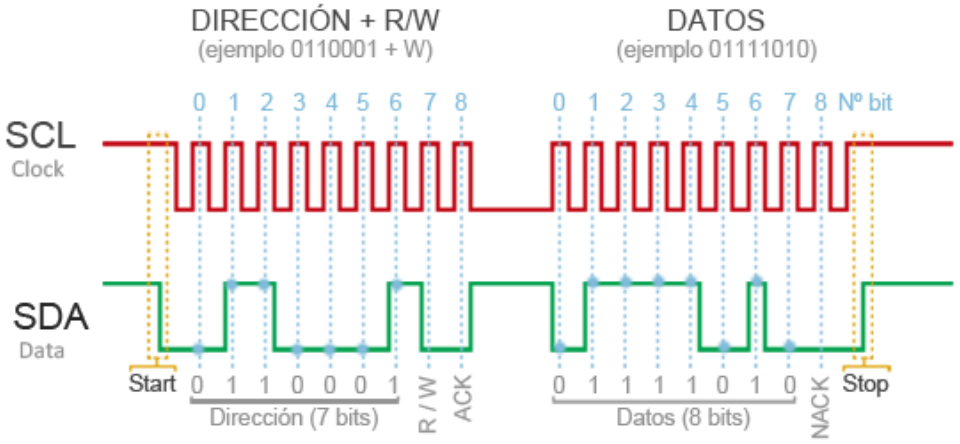



**Figura 33: ejemplo de comunicación. [9]**

Con los 7 bits de dirección es posible acceder a 112 dispositivos en un mismo bus (16 direcciones se reservan para usos especiales).

El envío de 18 bits por cada 8 bits de datos hace que la velocidad del bus $I^2C$ sea reducida. La velocidad estándar de transmisión es de 100 MHz, con un modo de alta velocidad de 400 MHz.

En Arduino disponemos de soporte $I^2C$ por hardware vinculado a ciertos pines. En la tabla 4 podemos ver a qué pines corresponde este bus dependiendo del modelo.

**Tabla 5: Pines de conexión al bus I2C.**

| Modelo | SDA | SCL |
|---|---|---|
| Arduino Pro-Mini | A4 | A5 |
| Arduino Mega | 20 | 21 |

Para poder utilizar el bus se ha utilizado la librería Wire.h. Volveremos sobre ello en el apartado 1.8 dedicado al software.

#### 1.7.10.4. Conexionado

A continuación, se muestra el conexionado el módulo microSD al microcontrolador.

I. Arduino Pro-Mini.

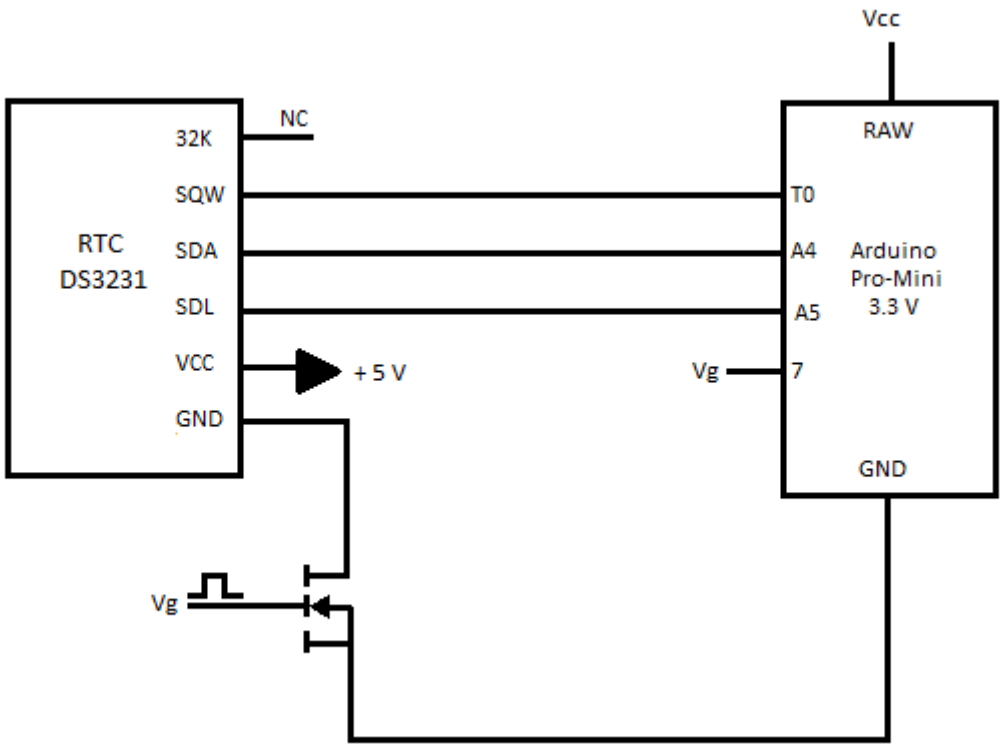



**Figura 34: conexionado RTC a Arduino Pro-Mini.**

II. Arduino Mega R3.

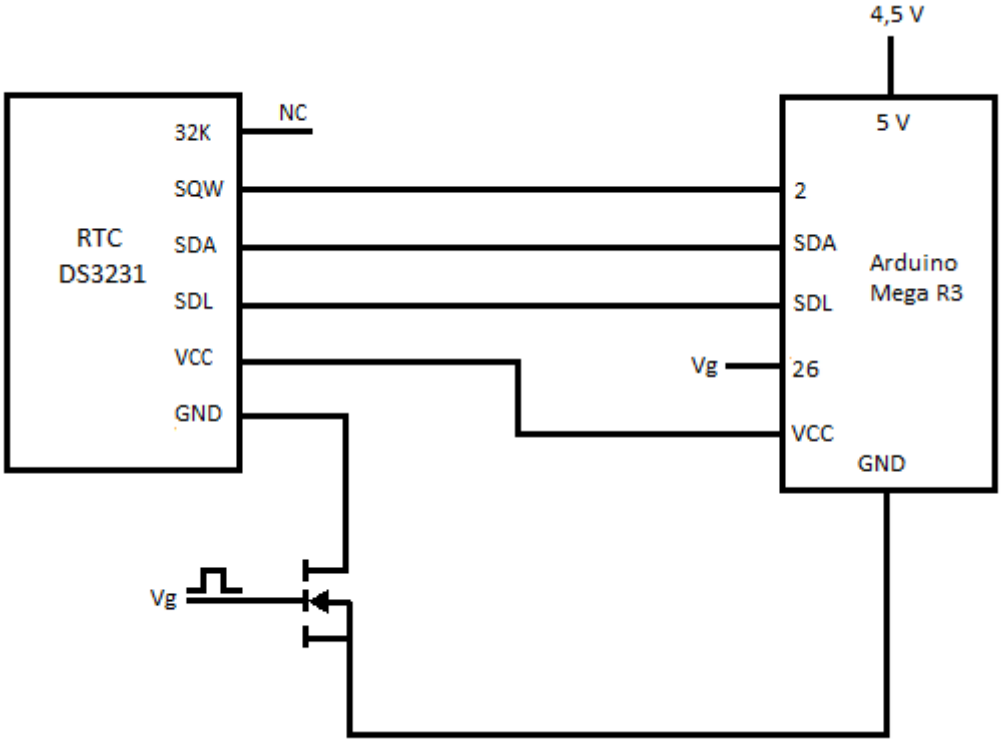



**Figura 35: conexionado RTC a Arduino Mega.**

## 1.8. Software

En este apartado incluimos la descripción del software empleado en el desarrollo del presente trabajo de fin de grado.

### 1.8.1. IDE Arduino

Para la programación de Arduino se ha utilizado el Arduino IDE 1.8.5, que es un entorno de desarrollo en el que se realiza la programación de estas placas. Este entorno en realidad está basado en C++, ofreciendo unas librerías que facilitan la programación de los pines de entrada/salida y de los puertos de comunicación.

### 1.8.2. Librerías utilizadas

Para facilitar la programación hemos utilizado las siguientes librerías:

- Adafruit TTL Serial Cámara JPEG. Se trata de una librería creada por Limor Fried bajo una licencia BSD. Se puede encontrar está librería en Github.

- Librería Serial Peripheral Interface (SPI), ya incluida en el IDE.

- Librería SD incluida en el IDE.

- DS3232RTC es la librería utilizada para programar el RTC DS3231. Creada por Jack Christensen bajo la licencia GNU GPL v3.0, en ella se incluye la librería "Wire" que nos permite comunicarnos a través del bus $I^2C$, y la librería "TimeLib". Con esta librería tenemos implementado el calendario hasta el año 2100 teniendo en cuenta años bisiestos y meses con diferentes números de días.

- La librería "avr/sleep" nos permite ahorrar energía poniendo a Arduino en diferentes estados. Nosotros la usamos para "dormir" el Arduino por la noche en el modo SLEEP_MODE_PWR_DOWN, que es el modo de mayor reducción de consumo.

### 1.8.3. Código.

En este apartado se describirán las funciones implementadas en el código.

- En la función **setup** definimos la configuración de los pines, iniciaremos la comunicación del puerto serial 0 para poder ver mensajes de error por pantalla, iniciamos la comunicación con la cámara, y con el módulo microSD. Además, se define el tamaño de la foto, se fija la alarma para despertar al microcontrolador y se fija la hora en el controlador y en el RTC.
- El programa es necesario cargarlo dos veces en el microcontrolador. La primera para fijar la hora en el RTC, comentando las líneas de sincronización. Y la segunda, una vez que el RTC está en hora, se comentan las líneas que fijan la hora en el RTC y se descomentan las líneas de sincronización. Esto se hace así para que cuando haya una pérdida de alimentación, por ejemplo, en el momento del cambio de las pilas, al volver

a suministrar la alimentación la hora del sistema se fije automáticamente mediante el RTC.

- Se ha creado la función **carpeta**, en la cual se da nombre y se crea directorio donde se almacenarán las imágenes. El nombre de la carpeta será "V_XXX" donde "XXX" va desde 000 hasta 999 por lo que se podrán almacenar las imágenes de hasta 1000 vehículos que han superado el límite de tiempo fijado para el estacionamiento.
- Por otro lado, se ha creado la función llamada **saveImage()** en la cual se da nombre al fichero JPG con el siguiente formato: hora-minuto-segundo.JPG. Un ejemplo de nombre de archivo sería 09-10-40.JPG. Una vez que se da nombre al fichero, este es creado y abierto indicando la ruta con la función SD.open(). Para almacenar la imagen vamos leyendo el buffer de la cámara de 32 en 32 bytes y escribiendo en la tarjeta microSD. Es decir, leemos 32 bytes y los escribimos en la tarjeta microSD y así hasta hayamos copiado toda la información. Para saber el peso de la foto utilizamos la función frameLenth() de la librería de la cámara. Para leer el buffer de la cámara utilizamos la función readPicture().
- La función **ping()** almacena la distancia detectada por el ultrasonido en un vector de tamaño cinco. El tiempo entre medida y medida es aproximadamente de 80 ms. El tiempo para obtener el vector de distancias es aproximadamente 600 ms.
- Por último, la función **loop()**, que es la función principal de nuestro código, en la que hemos puesto todas aquellas tareas que se tienen que realizar o comprobar repetidamente. En ella llamamos a la función ping para realizar la medición de distancias, y tenemos las condiciones en las cuales se toma una imagen y se almacena. Además de tener las condiciones en las que Arduino entra en modo sleep y el control del encendido y apagado de los componentes mediante la puesta a HIGH o LOW de los pines digitales que están conectados a la puerta de los diferentes MOSFET.

Dentro de la función loop dormimos Arduino a las 20:00h. Para ello apagamos todos los componentes excepto el RTC que debe permanecer alimentado para poder recibir enviar la alarma a Arduino. Una vez despierto Arduino sincronizamos la hora del sistema con la del RTC, apagamos el RTC y encendemos el ultrasonido.

Hay diferentes modos para dormir Arduino, los cuales se pueden consultar en la figura 36. Nosotros hemos utilizado el modo Power Down ya que es el modo de mayor ahorro. En este modo, cuando Arduino está en modo sleep solo se tienen activadas las interrupciones, la comunicación $I^2C$, y el watchdog. En este modo el consumo del Arduino es menor a 1 microamperio.

**Table 9-1.** Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

| | Active Clock Domains | | | | | Oscillators | | Wake-up Sources | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sleep Mode** | $clk_{CPU}$ | $clk_{FLASH}$ | $clk_{IO}$ | $clk_{ADC}$ | $clk_{ASY}$ | Main Clock Source Enabled | Timer Oscillator Enabled | INT1, INT0 and Pin Change | TWI Address Match | Timer2 | SPM/EEPROM Ready | ADC | WDT | Other I/O | Software BOD Disable |
| Idle | | | X | X | X | X | X(2) | X | X | X | X | X | X | X | |
| ADC Noise Reduction | | | | X | X | X | X(2) | X(3) | X | X(2) | X | X | X | | |
| Power-down | | | | | | | | X(3) | X | | | | X | | X |
| Power-save | | | | | X | | X(2) | X(3) | X | X | | | X | | X |
| Standby(1) | | | | | | X | | X(3) | X | | | | X | | X |
| Extended Standby | | | | | X(2) | X | X(2) | X(3) | X | X | | | X | | X |

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

**Figura 36: distintos modos de sleep.**

## 1.8.4. Kicad.

Para el diseño de la PCB hemos utilizado KiCad, un entorno de software libre utilizado para el diseño de circuitos impresos. En la figura 37 podemos ver como se han hecho las conexiones, mostrándose en ella las pistas “forward” y “bottom” además de los drills y la capa Edge.

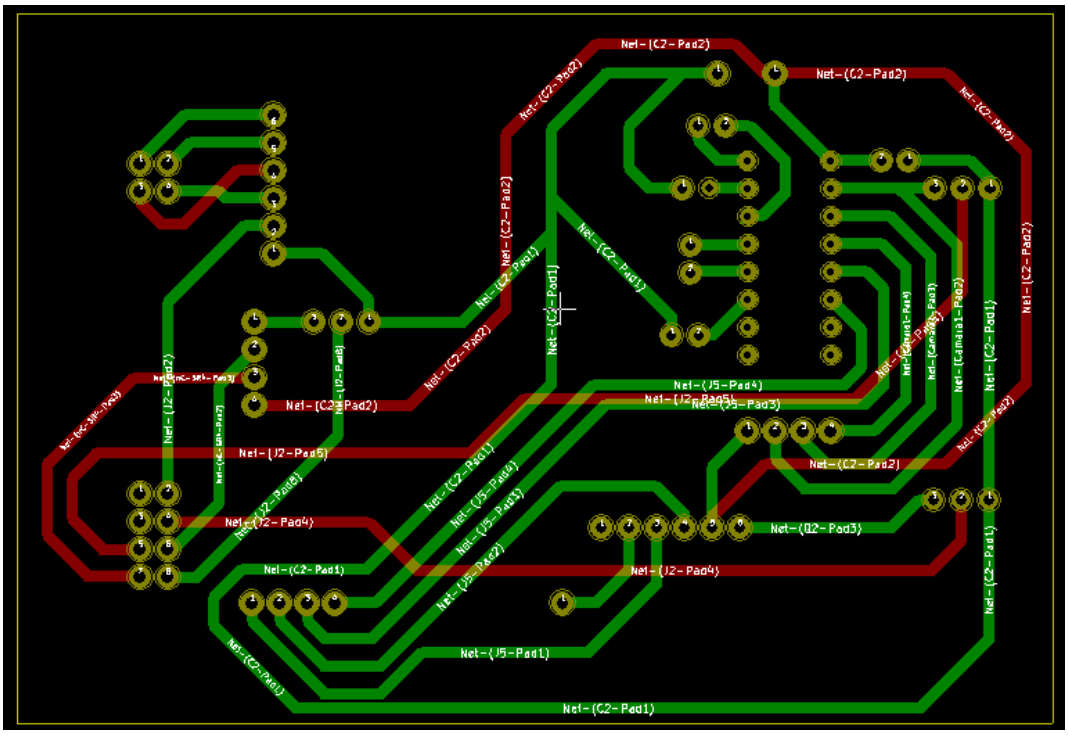



**Figura 37: diseño de la PCB**

En la figura 38, vemos la placa ya hecha y con los componentes soldados.

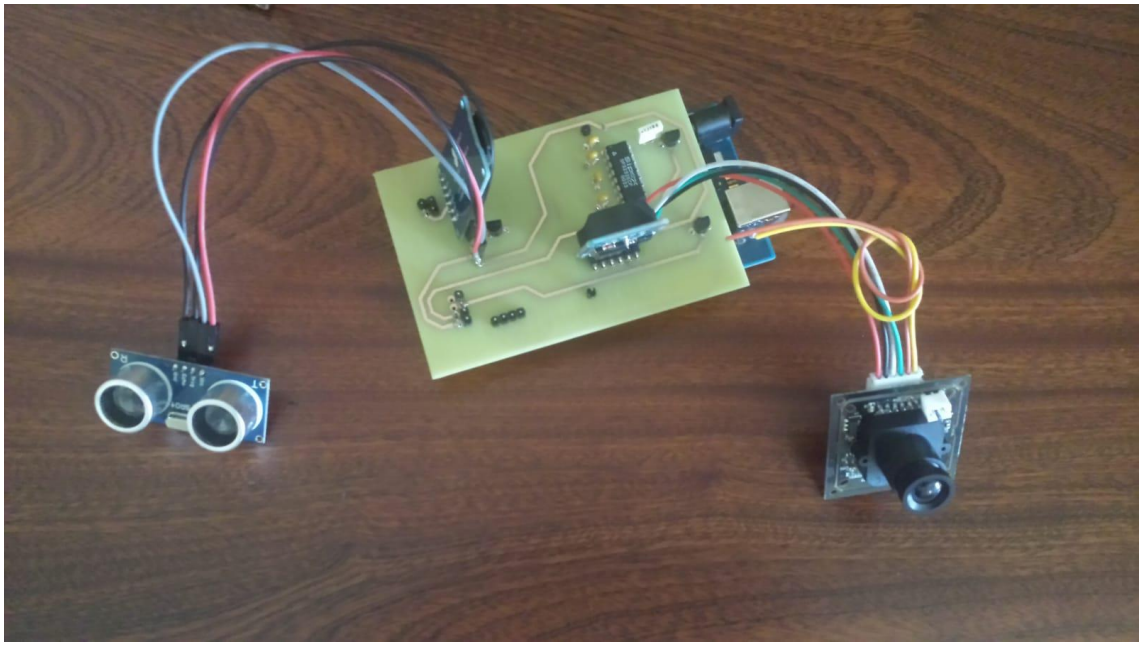


**Figura 38. PCB conectada a Arduino Mega.**

## 1.9. Alimentación y consumo.

Para saber con exactitud el consumo de global de nuestro prototipo y de cada uno de sus componentes hemos medido la intensidad total consumida y la intensidad de cada componente:

- Para el ultrasonido hemos obtenido una intensidad de 2,75 mA, aunque se pueden producir picos de hasta 15 mA en el proceso de medida.
- Para el RTC se produce un consumo de 3.4 mA cuando está alimentado a Vcc. Esto se debe al regulador de tensión. Cuando desconectamos la alimentación del Arduino y se alimenta por la pila de botón, tiene un consumo de 110 µA de acuerdo a la hoja de datos del DS3231.
- El módulo microSD tiene un consumo de 4 mA en reposo y en torno a los 10 mA cuando se almacena la imagen.
- La cámara tiene un consumo de 85 mA, ya sea a la espera o en la captura de la imagen.
- El circuito integrado SP3232 tiene un consumo de 6.0 mA.

En total se tiene un consumo de 181 mA, de los cuales 101.15 mA corresponden a los componentes conectados a Arduino y 38 mA corresponden al propio Arduino. Los 41 mA restantes son lo que consumen el regulador de tensión y el USB.

Para alargar la autonomía del prototipo se han utilizado 3 MOSFET, concretamente el modelo TN0702N3-G. El drenador se conecta a la tierra del componente, la puerta a un pin digital que controla si el MOSFET trabaja en corte o en saturación y el surtidor a tierra.

Este MOSFET entra en conducción a partir de un Vgs de 1.75 V. En saturación la caída de tensión que se produce entre drenador y surtidor es de 1.6 mV, por lo que la disipación de potencia es casi despreciable debido a la pequeña corriente que circula a través del transistor.

Utilizando 3 pilas AA de 1.5 V y 2800 mAh en serie, conectadas al pin de 5 V se llega a una autonomía de 5 días y 15 horas. Esto es debido al gran consumo que tiene Arduino (38 mA).

Sin duda, se trata de un consumo importante que dota a nuestro dispositivo de una autonomía inferior a la deseada. El aumento de dicha autonomía es, por tanto, una de las principales líneas en las que este trabajo podría ser mejorado, tal y como se indica al final de la memoria, en el apartado dedicado a las líneas abiertas.

## 1.10. Pruebas reales y resultados.

Una vez terminado el prototipo se ha creado una carcasa para el dispositivo, como podemos ver en la figura 38, y se han realizado distintas pruebas.

**Figura 39: Prototipo dentro de la carcasa en cartón.**

En primer lugar, se colocó el dispositivo y se conectó al ordenador por puerto serie para ver por pantalla los resultados que íbamos a obtener. Una vez hecho esto se comenzó con las pruebas.

La primera prueba que se realizó fue para determinar el ángulo óptimo que debe tener la cámara y el sensor de ultrasonido. La prueba se realizó con un Toyota Yaris. El ángulo de la cámara se ajustó de forma que la matrícula saliese en la parte inferior de la fotografía, para que en caso de que aparque un coche que tenga la matrícula más alta, esta salga en la imagen. La cámara se encuentra a una altura de 7,5 cm respecto al suelo y tiene un ángulo de inclinación de 72º.

A continuación, se trabajó con el sensor de ultrasonidos para ajustar el tiempo entre medida y medida del vector de distancias. Para ello probamos a aparcar a distintas velocidades. Inicialmente habíamos fijado este tiempo en 300 ms, pero tras la realización de estas pruebas nos dimos cuenta de que en caso de que el vehículo aparque rápido, el dispositivo no es capaz de detectar adecuadamente la entrada del mismo. Para solucionar este problema y establecer el tiempo óptimo entre medidas consecutivas se realizaron varios estacionamientos a distintas velocidades,

encontrando que cuando el coche estaciona lentamente el dispositivo detecta la entrada y la salida del vehículo sin dificultad. En cambio, cuando el estacionamiento se produce a una velocidad elevada, la maniobra de entrada y/o salida del vehículo no siempre es detectada.

Se decidió bajar el tiempo entre medida a 80 ms y se repitieron las pruebas. En este caso se realizaron cinco intentos de estacionamiento a distintas velocidades. En los cinco intentos el dispositivo fue capaz de detectar correctamente tanto la entrada y la salida del vehículo.

Durante estas pruebas también nos dimos cuenta de que es necesario obtener más de un vector de distancias ya que, en muchos casos, el dispositivo detectaba la entrada del vehículo en el segundo o tercer vector de distancias, en algunos intentos incluso en el cuarto.

Además, durante estas pruebas se comprobó el tiempo de captura y almacenamiento de la imagen en la tarjeta microSD. Este tiempo variaba entre 1,2 y 2 segundos, teniendo un valor típico de 1,7 s.

**Figura 40: Imagen de entrada y de salida del vehículo.**

# 2. Memoria justificativa.

A continuación se justificarán las decisiones tomadas en el transcurso del desarrollo de este Trabajo de Fin de Grado que afectan tanto a la parte hardware como a la parte software.

## 2.1. Hardware

En primer lugar, se justificará la elección de los distintos elementos hardware que componen el dispositivo construido.

### 2.1.1. Selección microcontrolador

En un primer momento se eligió el Arduino Pro-Mini como el microcontrolador ideal para este Trabajo de Fin de Grado debido a su bajo consumo. Se llegó a hacer el montaje completo de todos los componentes, pero en la fase de programación vimos que se ocupaba el 80 % de la memoria SRAM del microcontrolador, que son 2 KB. Debido a esto surgieron problemas de estabilidad y el programa no funcionaba correctamente ya que la memoria estática era insuficiente.

Por este motivo, decidimos optar por un modelo superior, el Arduino Mega. La ventaja de este microcontrolador en que posee una mayor memoria estática (8 KB). Además, incluye 4 UARTs, por lo que no es necesario utilizar la librería SoftwareSerial, disminuyendo así el consumo de SRAM. La gran desventaja de este modelo es el consumo, su consumo es mayor que el Arduino Pro-Mini, lo cual disminuye la autonomía del prototipo.

Otro motivo para descartar Arduino Pro-Mini ha sido, que, si en un futuro se desean implementar nuevas funcionalidades al dispositivo, la escasa SRAM de la que dispone el microcontrolador nos haría imposible llevarlas a cabo. Una de estas nuevas funciones puede ser la creación de un fichero .dat que indique el tiempo de estacionamiento de cada vehículo para así poder saber si el límite de tiempo asignado al parking es adecuado o no.

Por otro lado, al utilizar el Arduino Mega evitamos tener que utilizar un regulador de tensión para obtener un voltaje de 5 V para alimentar los componentes.

### 2.1.2. Sensor de ultrasonido HC-SR04.

Los sensores de ultrasonidos son sensores de baja precisión. La orientación de la superficie a medir puede provocar que la onda se refleje, falseando la medición. Además, no resultan adecuados en entornos con un gran número de objetos, dado que el sonido rebota en las superficies de estos generando ecos y falsas mediciones. Tampoco son apropiados, en general, para el funcionamiento en el exterior y al aire libre. Pero en nuestro caso, dado que el objeto a medir es un vehículo y, por tanto, tiene un tamaño considerable, no hay ningún problema en utilizar este tipo se sensor. Esto es así porque cuanto mayor es el área del objeto a detectar menos falsas mediciones se producen.

El principal motivo por el que se ha elegido el HC-SR04 es por el coste. Es uno de los sensores más baratos que podemos encontrar en el mercado.

### 2.1.3. Cámara VC0706

Para fotografiar la matrícula de los vehículos que aparcan en las plazas de tiempo limitado se empezó a trabajar inicialmente con la cámara OV7670, pero su programación resultó ser bastante compleja. Se llegó a programar y a obtener imágenes de ella, pero no eran imágenes “claras”, presentando saturación y bastante ruido. Por este motivo se descartó este modelo y se optó por la cámara VC0706, que tenía la ventaja de traer implementada una librería y resultaba mucho más sencilla de utilizar. Otro argumento para elegir esta cámara es su bajo coste, ya que nos permite obtener una resolución de 0,3 MP a un bajo precio.

El gran inconveniente del modelo elegido es que la comunicación sigue el protocolo RS232.

En cuanto a la máxima velocidad de comunicación que hemos conseguido con la VC0706, es de 38400 bps. Si la excedemos se producen fallos de lectura del buffer.

### 2.1.6. Módulo microSD

Se seleccionó este módulo, aparte de por ser el único que se encontró en el mercado, porque ya incorpora la ranura microSD y dispone de un regulador para alimentar la tarjeta a 3,3 V.

### 2.1.9. RTC

Se seleccionó el DS3231, un reloj de alta precisión, por su coste. Realmente en este Trabajo de Fin de Grado no es necesaria una precisión tan alta, debido a que solo se tiene en cuenta el tiempo de estacionamiento. Es otras palabras, no nos importa que haya una diferencia de segundos entre la hora real y la hora del sistema. A pesar de ello, se eligió este componente debido a que no hay diferencia de precio entre el DS3231 y un DS1307, por ejemplo, cuya precisión es menor.

## 2.2 Alimentación y consumo.

Como se mencionó en la memoria descriptiva, usando pilas AA de 2800 mAh se consigue una autonomía de 5 días y 15 horas. En la tabla 6 se puede ver el cálculo desglosado de la autonomía.

**Tabla 6: Consumo.**

| Componente | Consumo (mA) | horas activo/día | mAh/día | Total(días) |
|---|---|---|---|---|
| Cámara + SP3232 | 91 | 0,0458 | 4,17 | |
| microSD | 10 | 0,0458 | 0,46 | |
| RTC | 3,4 | 13 | 44,20 | |
| Regulador y USB | 41 | 0 | 0 | |
| Arduino (sin tener en cuenta regulador y USB) | 38 | 11 | 418 | |
| HC-SR04 | 2,75 | 11 | 30,25 | |
| Total | | | 497,08 | 5,63 |

Se ha calculado el consumo del dispositivo en mAh/día. Sabiendo esto y que la pilas son de 2800 mAh, dividiendo 2800 entre el consumo diario, obtenemos cada cuántos días tendrían que reemplazarse las pilas.

Para esto se ha calculado las horas activas de cada componente y se ha medido el consumo de cada uno de ellos.

Viendo la autonomía de la que dispone este diseño, que está por debajo de lo que sería deseable, llegamos a la conclusión de que hay que disminuir el consumo o bien encontrar otra fuente de alimentación.

## 3. Presupuesto.

| Producto | Unidades | Precio/unidad(€) | Precio total (€) |
|---|---|---|---|
| HARDWARE | | | |
| RTC DS3231 | 1 | 0,91 | 0,91 |
| Cámara VC0706 | 1 | 14,49 | 14,49 |
| Sensor HC-SR04 | 1 | 0,69 | 0,69 |
| Módulo microSD | 1 | 0,38 | 0,38 |
| Arduino Mega R3 | 1 | 8,26 | 8,26 |
| SP3232 | 1 | 4,95 | 4,95 |
| CR2032 | 1 | 1,5 | 1,5 |
| Pila 9V 6LR61 | 3 | 2 | 6 |
| Condensadores | 5 | 0,25 | 1,25 |
| PCB | 1 | 15 | 15 |
| Total | | | 53,43 |
| | | | |
| MANO DE OBRA | horas | Precio/hora | Precio total (€) |
| Ingeniero técnico | 300 | 22 | 6600 |
| | | | |
| TOTAL | | | 6653,43 € |

En este primer presupuesto se incluye las horas aproximadas dedicadas al proyecto, pero hay que tener en cuenta que muchas de esas horas se dedicaron a buscar documentación sobre el proyecto, a buscar los elementos que mejor se ajustaban a los requerimientos de nuestra aplicación (el Arduino, la cámara, etc.… ), a familiarizarte con la programación, con el diseño de la PCB en KiCAD.

Teniendo esto en cuenta, estimamos que para crear el prototipo de nuevo se necesitan 25 horas de trabajo aproximadamente. Obteniendo así el siguiente presupuesto.

| Producto | Unidades | Precio/unidad(€) | Precio total (€) |
|---|---|---|---|
| HARDWARE | | | |
| RTC DS3231 | 1 | 0,91 | 0,91 |
| Cámara VC0706 | 1 | 14,49 | 14,49 |
| Sensor HC-SR04 | 1 | 0,69 | 0,69 |
| Módulo microSD | 1 | 0,38 | 0,38 |
| Arduino Mega R3 | 1 | 8,26 | 8,26 |
| SP3232 | 1 | 4,95 | 4,95 |
| CR2032 | 1 | 1,5 | 1,5 |
| Pila 9V 6LR61 | 3 | 2 | 6 |
| Condensadores | 5 | 0,25 | 1,25 |
| PCB | 1 | 15 | 15 |
| Total | | | 53,43 |
| | | | |
| MANO DE OBRA | horas | Precio/hora | Precio total (€) |
| Ingeniero técnico | 25 | 22 | 550 |
| | | | |
| TOTAL | | | 603,43 € |

## 4. Conclusiones.

In this Final Degree Project, we have achieved a first prototype that fulfills perfectly the required functions in this problem. The device can detect the entry and exit of a vehicle regardless of the speed with it parks. In addition, it is possible to store the entry and exit images of the car with the entry and exit time, thus, registering vehicles that have exceeded the maximum time allowed.

The low-cost objective has been got because, the hardware budget is 53,43 €. The only drawback of this first prototype is the autonomy. Only an autonomy of 5 days and 15 hours have been achieved.

From a personal point of view, this project has allowed me to develop my knowledge in programming with the Arduino IDE, to better understand how a microcontrollers work, familiarize with digital cameras and learn how to design a PCB.

On the other hand, it has allowed me to develop, what at the beginning was only an idea, in a first prototype. With it, we have obtained better results that the ones we have expected at the beginning of the project. So I am very proud of making this project.

## 5. Líneas Abiertas.

Como se ha comentado en las conclusiones, el prototipo que hemos diseñado e implementado cumple, en general, con las expectativas que se plantearon en la fase inicial de este trabajo. No obstante, hay ciertos aspectos, como los que se mencionan a continuación, que deberían mejorarse de cara a la utilización real de este dispositivo para el fin para el que se ideó.

- Disminución del consumo. En el estado actual apenas se consiguen 6 días de autonomía. Este aspecto del dispositivo puede y debe mejorarse.
    - Una posible mejora pasa por un diseño con un microcontrolador de 3.3 V de bajo consumo. Además, hemos visto que no es necesario utilizar un reloj de 16 MHz. Cuando se utilizó el Arduino Pro-Mini comprobamos que con un cristal de 8 MHz el diseño es perfectamente válido.

    - También se puede plantear la opción de que una vez aparcado el vehículo se ponga a Arduino en modo sleep, asumiendo así un tiempo mínimo de uso del estacionamiento. Esto disminuiría enormemente el consumo del dispositivo.

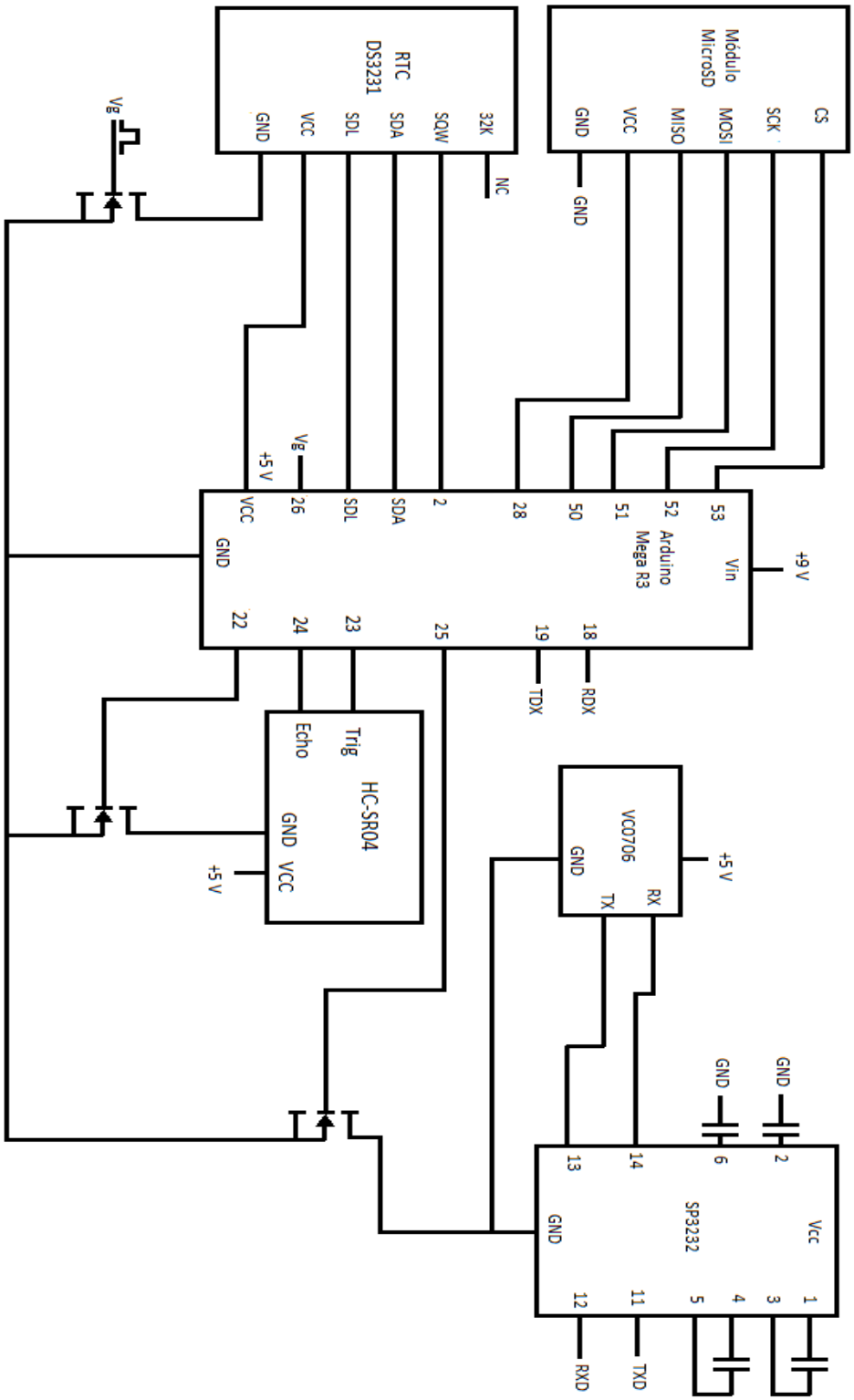
    - Para poder mejorar la autonomía otra posible opción sería la colocación de un pequeño panel solar que iría situado en la señalización vertical del estacionamiento. Este panel solar proveería de energía a las baterías de al menos dos dispositivos ya que en este tipo de aparcamientos suelen haber al menos dos aparcamientos de este tipo uno al lado del otro.

- Implementación de un sistema inalámbrico para el traspaso de las imágenes, evitando así, abrir la carcasa para extraer la tarjeta microSD para cortar los ficheros. Este sistema puede ser el Bluetooth.

- Diseñar una carcasa con un material más resistente como puede ser el acero, ya que, en este proyecto, esta carcasa se ha hecho de cartón con la única intención de poder hacer pruebas con el dispositivo.

# 5. Anexos

## 5.1. Anexo I



## 5.2 Anexo II

```
//*******Librerías***********

#include <Adafruit_VC0706.h> //Librería para la utilización de la cámara

#include <SPI.h> //Librería para comunicación asíncrona

#include <SD.h> //Librería del módulo microSD

//#include <TimeLib.h> Librería para manejar calendarios. Se incluye en el RTC.

//#include <Wire.h> Para poder comunicarnos con el protocolo I2C. Se incluye en el RTC.

#include <DS3232RTC.h> //Librería del RTC

#include <avr/sleep.h>


//******Definición de pines*******

const uint8_t Trig = 23;

const uint8_t Echo = 24;

const uint8_t VgSonido = 22; //Control de la puerta del MOSFET para alimentar el HC-SR04.

const uint8_t chipSelect = 53;

const uint8_t VccSD = 28; //Pin digital de alimentación del módulo microSD.

const uint8_t VgRTC = 26; //Control de la puerta del MOSFET para alimentar el RTC.

const uint8_t VgCamara = 25; //Control de la puerta del MOSFET para alimentar la cámara y el
//SP3232.


//******Variables Globales*********

int distcoche[]={0,0,0,0,0};//Vector de aproximación o de alejamiento del coche

bool haycoche = 0; //Se utiliza para saber si hay un vehículo estacionado

String dirname; //Nombre del directorio donde se almacenará la imagen o imágenes de los
//vehículos que estaciones en el aparcamiento

String ruta; // Ruta en la que se guardaran las imágenes. En ella se indica el directorio y el
//nombre del archivo.

uint16_t t_entrada; //Se utiliza para almacenar el tiempo en minutos de la hora de entrada.

uint16_t t_salida; // Se utiliza para almacenar el tiempo en minutos de la hora de salida.

uint16_t total; //Se guarda el tiempo que ha estado el vehículo estacionado.
```

```
Adafruit_VC0706 cam = Adafruit_VC0706(&Serial1);

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  pinMode(VgSonido, OUTPUT); // Vgs. Interruptor HC-SR04.
  digitalWrite(VgSonido, HIGH);
  pinMode(VgRTC, OUTPUT);//RTC
  digitalWrite(VgRTC, HIGH);
  pinMode(VccSD, OUTPUT);//SD
  digitalWrite(VccSD, HIGH);
  pinMode(VgCamara, OUTPUT);//Cámara + integrado
  digitalWrite(VgCamara, HIGH);
  pinMode(Trig, OUTPUT);

  pinMode(Echo, INPUT);
  pinMode(chipSelect, OUTPUT); // El pin de selección debe estar como una salida para no causar
 //ningún problema en la comunicación
  pinMode(2,INPUT_PULLUP); // Pin de la interrupción--Alarma.

  Serial.begin(9600);

  // Iniciamos la comunicación con la cámara
  if (cam.begin()) {
    Serial.println("Camara encontrada:");
  } else {
    Serial.println("Camara no encontrada?");
    return;
  }
  // Iniciamos la comunicación con la tarjeta SD y comprobamos si está la microSD
```

```
if (!SD.begin(chipSelect)) {
  Serial.println("Tarjeta no encontrada");
  return;
}
digitalWrite(VccSD,LOW);

//cam.setImageSize(VC0706_640x480);        // biggest
//cam.setImageSize(VC0706_320x240);        // medium
cam.setImageSize(VC0706_160x120);          // small. Tenemos seleccionado QQVGA
/*uint8_t imgsize = cam.getImageSize();
Serial.print("Image size: ");
if (imgsize == VC0706_640x480) Serial.println("640x480");
if (imgsize == VC0706_320x240) Serial.println("320x240");
if (imgsize == VC0706_160x120) Serial.println("160x120");*/
digitalWrite(VgCamara,LOW);

//************************ALARMAS*********************************
RTC.setAlarm(ALM1_MATCH_DATE, 0, 0, 0, 1);
RTC.alarm(ALARM_1);
RTC.alarmInterrupt(ALARM_1, false);
RTC.setAlarm(ALM2_MATCH_HOURS, 0, 0, 8, 0);
RTC.alarm(ALARM_2);
RTC.squareWave(SQWAVE_NONE);
RTC.alarmInterrupt(ALARM_2, true);//Habilita la interrupción de alarma2
attachInterrupt(0, Alarma, LOW);
//*******************************************************************
```

```
  //******************Solo para fijar fecha y hora en el RTC***************
  setTime(12,00,00,21,05,2018); // Las 12:00:00 del día 21 de mayo de 2018
  if(! RTC.set(now())){ //Devuelve 0 si es correcto
    Serial.println("Se ha fijado tiempo en RTC");
  }
  //*************************************************************************


 //****************Sincronizamos con el RTC*********************************
 /*setSyncProvider(RTC.get);      // Vamos a usar el RTC
 if (timeStatus() != timeSet){ //Devuelve timeSet si se ha ajustado correctamente
   Serial.println("No se ha podido sincronizar con el RTC");
 }else{
   Serial.println("El RTC ha fijado la hora del sistema");
   delay(200);
   digitalWrite(26,LOW);
 }*/
 //*****************************************************************************
}

void carpeta(){
  //Creamos el nombre con el que se llamara la carpeta
  dirname = "V_XXX";
  for (int i = 0; i < 999; i++) {
    dirname[2]  = '0' + i/100;
    dirname[3] = '0' + (i - 100*((int)(dirname[2]-48)))/10;
    dirname[4] = '0' + i%10;
    if (! SD.exists(dirname)) {
      break;
    }
  }
```

```
  //Creamos la carpeta
  if(! SD.mkdir(dirname)){
    Serial.println("No se ha creado el directorio");
    return;
  }else{
    Serial.println("El directorio se ha creado correctamente");
  }
}



void saveImage(){
  time_t t_captura = now();
  String s,m,h;
  if(second(t_captura) < 10){
    s = '0' + String(second(t_captura));
  }else{
    s = String(second(t_captura));
  }
  if(minute(t_captura) < 10){
    m = '0' + String(minute(t_captura));
  }else{
    m = String(minute(t_captura));
  }
  if(hour(t_captura) < 10){
    h = '0' + String(hour(t_captura));
  }else{
    h = String(hour(t_captura));
  }
  String filename = h + '-' + m + '-' + s;
  Serial.println(filename);
  ruta = dirname + '/' + filename + ".JPG";
```

```
Serial.print("La ruta es:" );//Para comprobar que el nombre de la ruta es correcto
Serial.println(ruta);



File imgFile = SD.open(ruta, FILE_WRITE);
//Comprobamos si se ha abierto el archivo.
if(imgFile == false){
  Serial.println("ERROR. NO SE HA PODIDO ABRIR EL ARCHIVO");
  return;
}else{
  Serial.println("Todo correcto con la ruta");
}
// Get the size of the image (frame) taken
uint16_t jpglen = cam.frameLength();
Serial.print("Storing ");
Serial.print(jpglen, DEC);
Serial.print(" byte image.");
uint16_t time = millis();
byte wCount = 0; // Para contar el número de bytes escritos
while (jpglen > 0) {
  // Lee 32 bytes cada vez;
  uint8_t *buffer;
  uint8_t bytesToRead = min(32, jpglen);
  buffer = cam.readPicture(bytesToRead);
  imgFile.write(buffer, bytesToRead);
  if(++wCount >= 64) { // Imprime por pantalla un punto cada 64 bytes.
    Serial.print('.');
    wCount = 0;
  }
```

```
    jpglen -= bytesToRead;



  //Cerramos el fichero
  imgFile.close();

  //Calculamos cuanto tiempo ha tardado para escribir la imagen
  time = millis() - time;
  Serial.println("done!");
  Serial.print(time); Serial.println(" ms elapsed");
}
void ping() {
   uint32_t duration;
   int distancecm;
   uint8_t i=0;
   for(i=0;i<5;i++){
     digitalWrite(Trig, LOW);  //para generar un pulso limpio ponemos a LOW 10us
     delayMicroseconds(10);
     digitalWrite(Trig, HIGH);  //generamos Trigger (disparo) de 10us
     delayMicroseconds(10);
     digitalWrite(Trig, LOW);

     duration = pulseIn(Echo, HIGH);  //medimos el tiempo entre pulsos, en microsegundos
     distancecm = (duration * 100) /(2914*2);   //Calculamos la distancia en cm

     distcoche[i] = distancecm;
     delay(80); //Tiempo entre medición.
   }
}
```

```
void loop() {
int16_t time = millis();//Esto es para ver cuánto tiempo se tarda en tomar el array de distancias
  //Estamos llamando a la función ping continuamente.
  ping();

  time = millis() - time; //Tiempo en tomar las mediciones

  Serial.println("Distancia:");
  Serial.println(distcoche[0]); //Para ver los valores
  Serial.println(distcoche[1]);
  Serial.println(distcoche[2]);
  Serial.println(distcoche[3]);
  Serial.println(distcoche[4]);
  Serial.print(time); Serial.println(" ms elapsed");

  uint8_t i = 0, entrada = 0, salida =0;
  for(i=0;i<4;i++){
    if(distcoche[i+1] > distcoche[i]){
      salida++;
    }
    if(distcoche[i+1] < distcoche[i]){
      entrada++;
    }
  }
  //ESTO ES SI APARCA UN COCHE MIENTRAS EL ARDUINO ESTA DORMIDO
  if((entrada != 4) && (hour(now()) == 9) && (minute(now()) == 0) && second(now()) < 10){
    carpeta(); //Creamos el directorio
    haycoche = 1;
    t_entrada =  540; //9*60
  }
```

```
//Si entrada vale 4 entonces se acerca un coche. Lo mismo con salida
if(entrada == 4){
  digitalWrite(VgCamara,HIGH);
  digitalWrite(VccSD,HIGH);
  delay(50);
  cam.begin();
  SD.begin(chipSelect);
  delay(200);
  cam.setImageSize(VC0706_160x120);
  Serial.println("COCHE ENTRANDO");
  cam.takePicture();
  carpeta();
  saveImage();
  t_entrada = hour(now())*60 + minute(now());
  Serial.println(t_entrada);
  haycoche = 1;
  digitalWrite(VgCamara,LOW);
  digitalWrite(VccSD,LOW);
}

if(salida == 4 && haycoche ==1){
   Serial.println("COCHE SALIENDO");
   t_salida = hour(now())*60 + minute(now());
   Serial.print("tiempo de salida: ");
   Serial.println(t_salida);
   total = (t_salida - t_entrada);
   Serial.println(total);
   if((t_salida - t_entrada) > 20){
   digitalWrite(VgCamara,HIGH);
   digitalWrite(VccSD,HIGH);
   delay(50);
```

```
    cam.begin();
    SD.begin(chipSelect);
    delay(200);
    cam.setImageSize(VC0706_160x120);
    cam.takePicture();
    saveImage();
    haycoche = 0;
    digitalWrite(VgCamara,LOW);
    digitalWrite(VccSD,LOW);
    Serial.println("Se han guardado las dos imagenes");
  }else{
    if(!SD.remove(ruta)){
      Serial.println("ERROR AL ELIMINAR EL FICHERO");
    }else{
      Serial.println("SE HA ELIMINADO EL FICHERO");
    }
    if(!SD.rmdir(dirname)){

  Serial.println("ERROR AL ELIMINAR LA CARPETA");
    }else{
      Serial.println("SE HA ELIMINADO LA CARPETA");
    }
  }
}

//*****************Dormir Arduino************************************
if((hour(now()) == 20) && (minute(now()) == 0)){
  pinMode(Trig, INPUT);
  digitalWrite(VgSonido, LOW);
  digitalWrite(VgRTC, HIGH);//RTC
  digitalWrite(VccSD, LOW);
```

```
  digitalWrite(VgCamara, LOW);
  attachInterrupt(0, Alarma, LOW); // Activamos la interrupción pin 2.
  set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Fijamos el modo sleep
  Serial.println("DORMIMOS");
  sleep_enable(); //Habilitamos el bit de sleep en el registro del microcontrolador
  sleep_mode(); //Se duerme Arduino
  sleep_disable(); //Deshabilitamos el bit sleep
  detachInterrupt(0);//Deshabilitamos la interrupción 0
  setSyncProvider(RTC.get); // Sincronizamos de nuevo con el RTC.
  if (timeStatus() != timeSet){ //Devuelve timeSet si se ha ajustado correctamente
   Serial.println("No se ha podido sincronizar con el RTC");
  }else{
   Serial.println("El RTC ha fijado la hora del sistema");
  }

  pinMode(Trig, OUTPUT);
  digitalWrite(VgSonido, HIGH);
  digitalWrite(VgRTC, LOW);//RTC
}
```

## 7. Referencias.

[1] https://store.arduino.cc/arduino-pro-mini

[2] https://store.arduino.cc/usa/arduino-mega-2560-rev3

[3] http://wiki.robotica.webs.upv.es/wiki-de-robotica/sensores/sensores-proximidad/sensor-de-ultrasonidos/

[4] https://es.wikipedia.org/wiki/Sensor_CMOS

[5] https://es.wikipedia.org/wiki/Mosaico_de_Bayer

[6] https://es.wikipedia.org/wiki/Bluetooth

[7] https://es.ccm.net/contents/69-como-funciona-el-bluetooth

[8] https://www.luisllamas.es/arduino-spi/

[9] https://www.luisllamas.es/arduino-i2c/