



ESTUDIOS DE INGENIERÍA DE ELECTRÓNICA

PROYECTO DE FIN DE CARRERA

DISPOSITIVO DE GEOLOCALIZACIÓN Y BALIZA  
PARA EL PROYECTO DE INNOVACIÓN DOCENTE  
UGRASP-ICARUS

CURSO: 2014/2015

JOSÉ ANTONIO CÁRDENAS MONTERO



ESTUDIOS DE INTENIERÍA ELECTRÓNICA

DISPOSITIVO DE GEOLOCALIZACIÓN Y BALIZA  
PARA EL PROYECTO DE INNOVACIÓN DOCENTE  
**UGRASP-ICARUS**

REALIZADO POR:

José Antonio Cárdenas Montero

DIRIGIDO POR:

Christian Agustín Morillas Gutiérrez

Rodrigo Carlos Agís Melero

DEPARTAMENTO:

Arquitectura y Tecnología de Computadores

Granada, Julio de 2015

# DISPOSITIVO DE GEOLOCALIZACIÓN Y BALIZA PARA EL PROYECTO DE INNOVACIÓN DOCENTE UGRASP-ICARUS

José Antonio Cárdenas Montero

## PALABRAS CLAVE:

Geolocalización, GPS, GSM, PIC32, ARDUINO, SMS, SIM900, UBLOX, NEO-6M, CHIPKIT, MPIDE, MICROCHIP, PCB, UGRASP, ICARUS.

## RESUMEN:

Este proyecto forma parte del programa UGRASP-ICARUS con el que se lanzará un globo sonda hasta la estratosfera con el objetivo de recoger imágenes de la curvatura de la tierra así como recoger datos durante el viaje. Este proyecto propone el diseño de un dispositivo de geolocalización basado en tecnología GPS que recogerá información sobre la posición y la altura de la barquilla así como su monitorización en tiempo real usando un sistema de comunicación GSM con el que se enviarán mensajes de texto a un teléfono móvil indicando la localización. Para controlar estos dispositivos se usará un microcontrolador PIC32 del fabricante Microchip. Para el desarrollo del programa se usará el entorno de desarrollo MPIDE, que es una plataforma basada en Arduino adaptada para microcontroladores PIC32. Durante el desarrollo de la memoria se explicarán todas las fases de desarrollo desde la selección de componentes para cada placa, las consideraciones a tener en cuenta en el diseño de las PCBs y la implementación del código del programa que hará funcionar el conjunto de módulos de forma correcta y ordenada.

# GEOLOCATION AND MARKER DEVICE FOR TEACHING INNOVATION PROJECT UGRASP-ICARUS

José Antonio Cárdenas Montero

## KEYWORDS:

Geolocation, GPS, GSM, PIC32, ARDUINO, SMS, SIM900, UBLOX, NEO-6M, CHIPKIT, MPIDE, MICROCHIP, PCB, UGRASP, ICARUS..

## ABSTRACT:

This project is part of UGRASP-ICARUS program with which a balloon will be thrown to the stratosphere in order to collect images of the curvature of the earth and collect data during the journey. This project proposes the design of a geolocation device based on GPS technology that will collect information about the position and height of the balloon and its real-time monitoring using a GSM communication system with which they sent text messages to a mobile phone indicating the location. To control these devices a PIC32 microcontroller of the manufacturer Microchip will be used. For program development will be used MPIDE development environment, which is a Arduino based platform adapted to be used with PIC32 microcontrollers. During the development of this memory will be discussed all phases of development from the selection of components for each board, the considerations to be taken into account in the design of PCBs and implementation of the program code that will run the set of modules correctly and orderly.

D. Christian A. Morillas Gutiérrez, profesor del departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada, y Rodrigo C. Agís Melero, responsable del Servicio de Mecatrónica y Sistemas Electrónicos del Centro de Instrumentación Científica de la Universidad de Granada, como directores del Proyecto de fin de Carrera de D. José Antonio Cárdenas Montero, informan que el presente trabajo, titulado:

Dispositivo de geolocalización y baliza para el proyecto de innovación docente UGRASP-ICARUS.

Ha sido realizado y redactado por el mencionado alumno bajo nuestra dirección, y con esta fecha autorizamos a su presentación.

Granada, a \_\_\_\_\_ de Julio de 2015

Fdo. Christian A. Morillas y Rodrigo A. Agís

Los abajo firmantes autorizan a que la presente copia del Proyecto de Fin de Carrera se ubique en la Biblioteca de Centro y/o departamento para ser libremente consultado por las personas que lo deseen.

Granada, a \_\_\_\_\_ de \_\_\_\_\_ de 2015

*(Firmas y número de DNI del alumno y de los tutores)*

# Índice

<b>1. Introducción</b>	<b>15</b>
1.1. Objetivos . . . . .	16
1.2. Estructura de la memoria . . . . .	16
<b>2. El proyecto UGRASP (UGR AeroSpace Program)</b>	<b>19</b>
2.1. Programa ICARUS . . . . .	20
2.2. Módulo de gelocalización . . . . .	23
2.3. Tarjetas de Circuito Impreso . . . . .	23
<b>3. Introducción a la Arquitectura PIC32</b>	<b>27</b>
3.1. Características de los PIC32 . . . . .	27
3.2. El CORE PIC32 (M4K) . . . . .	29
<b>4. Herramientas de desarrollo</b>	<b>33</b>
4.1. MPLABX . . . . .	33
4.2. Arduino . . . . .	33
4.3. La plataforma CHIPKIT . . . . .	35
4.4. Bootloader . . . . .	35
4.5. MPIDE . . . . .	36
<b>5. Placa de Control</b>	<b>37</b>
<b>6. Módulo de GPS</b>	<b>41</b>
6.1. El límite COCOM . . . . .	42
6.2. Características de la tarjeta de navegación: . . . . .	43
6.3. Hardware . . . . .	43
6.4. Módulo NEO-6M . . . . .	44
6.4.1. Timepulse . . . . .	45
6.4.2. Antena . . . . .	46
6.4.3. Power Management . . . . .	46
6.4.4. Modos de funcionamiento . . . . .	46
6.4.5. Descripción de los protocolos de comunicación serie . . . . .	48
6.4.6. Puertos UART . . . . .	48
6.5. Configuración de las interrupciones . . . . .	49
6.6. Protocolo NMEA . . . . .	49
6.7. Formato de la Latitud y la Longitud . . . . .	53
6.8. Implementación . . . . .	53
6.9. Tarjeta MicroSD . . . . .	54

<b>7. Módulo de Comunicación GSM</b>	<b>57</b>
7.1. Introducción . . . . .	57
7.2. Características del módulo GSM . . . . .	59
7.3. Consideraciones de diseño de la alimentación . . . . .	59
7.4. Escenarios de encendido y apagado . . . . .	60
7.5. Ahorro de energía . . . . .	63
7.5.1. Modo de funcionalidad mínima . . . . .	63
7.5.2. Modo Sleep (Slow Clock Mode) . . . . .	63
7.6. Interfaz Serie . . . . .	64
7.6.1. Interfaz de la Tarjeta SIM . . . . .	64
7.7. Led NETLIGH . . . . .	65
7.8. Antena . . . . .	65
7.9. Implementación . . . . .	65
<b>8. Módulo de alimentación</b>	<b>67</b>
<b>9. Validación y resultados</b>	<b>69</b>
<b>10. Fabricación y estimación económica</b>	<b>77</b>
10.1. Proceso de fabricación de las PCB . . . . .	77
10.2. Estimación Económica . . . . .	78
<b>11. Conclusiones</b>	<b>81</b>
11.1. Futuras líneas de trabajo . . . . .	82
<b>Referencias</b>	<b>85</b>
<b>Apéndice A: Esquemáticos de la placa de desarrollo.</b>	<b>87</b>
<b>Apéndice B: Esquemáticos de la placa GPS.</b>	<b>93</b>
<b>Apéndice C: Esquemáticos de la placa de alimentación.</b>	<b>97</b>
<b>Apéndice D: Esquemáticos de la placa de GSM.</b>	<b>101</b>
<b>Apéndice E: Implementación.</b>	<b>107</b>
<b>Apéndice F: Listas de materiales.</b>	<b>116</b>

# Índice de figuras

1.	Peñón del embalse de canales . . . . .	19
2.	Water Rocket . . . . .	20
3.	Sonda estratosférica . . . . .	21
4.	Globo estratosférico en ascenso . . . . .	22
5.	Trayectoria globo . . . . .	23
6.	Diagrama de bloques del sistema . . . . .	25
7.	Arquitectura del PIC32MX795F512L . . . . .	28
8.	Logo de arduino . . . . .	34
9.	Placa Arduino UNO . . . . .	34
10.	Ilustración del entorno de desarrollo MPIDE . . . . .	36
11.	Configuración de pines de la placa de control . . . . .	38
12.	Pickit 3 . . . . .	39
13.	Cable FTDI TTL-232R-3V3 . . . . .	40
14.	Monitor serie MPIDE . . . . .	40
15.	Shield Arduino Ublox NEO-6M . . . . .	41
16.	Diagrama de la Shield Ublox NEO-6M . . . . .	44
17.	Diagrama de bloques del módulo Ublox Neo-6M . . . . .	45
18.	Señal TimePulse . . . . .	45
19.	Configuración de las interrupciones de la Shield Ublox NEO-6M . . . . .	50
20.	Sentencia NMEA . . . . .	51
21.	Sentencia GGA . . . . .	52
22.	Diagrama de pines SD . . . . .	56
23.	Diagrama de la placa GSM . . . . .	57
24.	Diagrama Funcional del SIM900 . . . . .	58
25.	Encendido del dispositivo: (arriba) Modo PWRKEY; (abajo) Modo PWRKEY y PWRKEY_OUT . . . . .	61
26.	Apagado del dispositivo: (arriba) Modo PWRKEY; (abajo) Modo PWRKEY y PWRKEY_OUT . . . . .	62
27.	Fuente de alimentación estabilizada . . . . .	69
28.	Placa de alimentación . . . . .	70
29.	Resultados de los voltajes de salida . . . . .	70
30.	Placa GPS . . . . .	71
31.	Tramas NMEA recibidas . . . . .	71
32.	Placa GSM . . . . .	72
33.	Comprobación de leds de la placa GSM . . . . .	73
34.	Envío de mensaje y comprobación . . . . .	73
35.	Montaje final y envío de mensajes activado . . . . .	74

36.	Comprobación del envío de mensajes . . . . .	75
37.	Información en tarjeta SD . . . . .	75
38.	Medida del consumo del dispositivo a máximo rendimiento . . . . .	76
39.	Máquina de control numérico para fabricación de PCBs . . . . .	77
40.	Fabricación de la PCB . . . . .	78
41.	Placas tras el galvanizado . . . . .	79
42.	Placa cortada . . . . .	79
43.	Diagrama de flujo del sistema . . . . .	109

## Índice de cuadros

1.	Correspondencia de pines . . . . .	37
2.	Descripción de pines de la Shield Ublox NEO-6M . . . . .	43
3.	Asignación de los “target numbers” . . . . .	48
4.	Relación de baud rates configurables . . . . .	49
5.	Algunas sentencias NMEA . . . . .	52



## Glosario de términos

**GSM** Global System for Mobile (Sistema Global para las comunicaciones Móviles)

**GSMA** Asociación GSM

**GPS** Global Positioning System (Sistema de Posicionamiento Global)

**SD** Secure Digital

**PCB** Printed Circuit Board (Placa de Circuito Impreso)

**SIM** Subscriber Identity Module (Módulo de Identidad de Abonado)

**ICSP** In Circuit Serial Programmer

**OTG** On The Go

**COCOM** Coordinating Committee for Multilateral Export Controls (Comité de Coordinación para el Control de Exportación Mutilateral)

**NMEA** National Marine Electronics Association (Asociación de Electrónica de Marina Nacional )

**RTC** Real Time Clock (Reloj de Tiempo Real)

**UGR** Universidad de Granada

**GPRS** General Packet Radio Service (Servicio General de Paquetes vía Radio)

**NASA** National Aeronautics and Space Administration (Administración Nacional de la Aeronáutica y del Espacio)

**RAM** Random Memory Access (Memoria de acceso Aleatorio)

**CPU** Central Processing Unit (Unidad Central de Procesamiento)

**JTAG** Joint Test Action Group

**EJTAG** Enhanced Joint Test Action Group

**SPI** Serial Peripheral Interface (Interfaz de comunicación Serie de Perifericos)

**IIC(I2C)** Inter-Integrated Circuit (Inter-Circuitos Integrados)

**ALU** Arithmetic Logic Unit (Unidad Aritmético Lógica)

**MAC** Instrucciones tipo Multiplica y Acumula

**MDU** Unidad de Multiplexación y División

**IDE** Integrated Development Environment (Entorno de Desarrollo Integrado)

**CAD** Computer-Aided Design (Diseño Asistido por Ordenador)

**MCU** Microcontroller Unit (Unidad Microcontroladora)

**USART** Universal Asynchronous Receiver-Transmitter (Transmisor-Receptor Asíncrono Universal)

**RF** Radio Frecuencia

**DDC** Direct Digital Control (Control Digital Directo)

**PVT:** Posición, Velocidad, Tiempo

**ADC** Analogic-Digital Converter (Convertidor de Analógico a Digital)

**ICD** (In Circuit Debugger)

## 1. Introducción

Este proyecto de fin de carrera está enmarcado dentro del proyecto de innovación docente titulado UGR Aaerospacial, del que hablaremos con más detalle en el Capítulo 2, y más concretamente en el programa ÍCARUS, que pretende desarrollar un globo sonda que será enviado a la estratosfera y con el que se podrán tomar fotos de la curvatura de la tierra así como la toma de datos de la trayectoria seguida o de las condiciones atmosféricas que se dan durante todo el viaje. Esta actividad supone una práctica muy atractiva tanto para los estudiantes de ingenierías, debido a su contenido tecnológico, como para los estudiantes de ciencias físicas, por los datos meteorológicos que se pueden obtener durante el viaje.

Uno de los problemas que se presentan a la hora de lanzar una sonda estratosférica es su localización. Este problema puede solucionarse haciendo uso de la tecnología GPS, que aprovechando una red de satélites que orbitan la tierra, proporcionan información en tiempo real sobre posición, velocidad, hora,... Esta información puede ser muy útil si la usamos junto con un cliente GSM, que proporciona una conexión a una red de telefonía móvil digital, puede utilizarse para enviar información sobre la localización de nuestro globo mediante internet o servicio de mensajes cortos SMS. Según la Asociación GSM (GSMA)[1], GSM es el estándar en telecomunicaciones móviles más extendido del mundo, con un 82 % por ciento de los terminales mundiales en uso, por lo que su uso en nuestro proyecto queda más que justificado.

El sistema GSM tiene ciertas limitaciones, ya que la cobertura de este sistema de comunicación deja de ser efectiva a partir de cierta altura. Esta limitación no nos afecta mucho, ya que lo que realmente nos interesa es poder recuperar la barquilla una vez que ésta llegue al suelo. Una vez que vuelva a la zona de cobertura GSM, nos enviará información sobre su posición y podremos acudir a recogerla.

En cuanto al sistema GPS, tampoco está libre de limitaciones. Existe un límite impuesto por la organización COCOM (Coordinating Committee for Multilateral Export Controls) [3] llamado “límite COCOM”, el cual se describirá más detalladamente en la sección 6.1, que está presente en muchos de los receptores GPS actuales y que no permite que éstos sigan funcionando por encima de los 18000 metros de altitud. Este límite es un problema bastante importante para nuestro proyecto ya que no nos dejaría recoger datos sobre la trayectoria seguida por encima del límite, por lo que hay que encontrar un dispositivo que no disponga de este límite o, en su defecto, que lo tenga más alto. Por ello encontramos un receptor GPS del fabricante UBLOX que tiene el límite de altura impuesto a 50000 metros, lo cual es más que suficiente para nuestro propósito.

Aunque no podamos monitorizar la posición de la barquilla durante toda su travesía, sí que nos interesa conservar información sobre ésta para su posible estudio. Para ello se implementará un módulo que gestione una tarjeta SD como dispositivo de almacenamiento donde se irá volcando

periódicamente información sobre su posición y altura, además de otras variables de interés.

Para controlar toda esta tecnología se propone usar un microcontrolador PIC32MX795F512 [7, 8] del fabricante Microchip usando la plataforma de desarrollo MPIDE desarrollada por DIGILENT[11]. Esta plataforma de desarrollo es una adaptación del lenguaje “wiring” de Arduino a los PIC32 de microchip. Arduino es un entorno que está muy extendido entre la comunidad de desarrolladores de sistemas embebidos y existe gran cantidad de material disponible en la red con cantidad de proyectos en formato Open Hardware y Open Software que nos puede facilitar mucho la tarea a la hora de buscar referencias de diseño. Además, presenta un método de programación muy intuitivo basado en C++ que es ideal para todas aquellas personas que están iniciándose en el mundo de la electrónica.

A parte del control de los dispositivos de GSM y GPS, sería interesante desarrollar una plataforma que sirviera de base para posibles ampliaciones o que fuera reutilizable para otra misión, debido a la naturaleza educativa del proyecto. De esta manera tendríamos un kit de desarrollo personalizado y acorde a nuestras necesidades.

### 1.1. Objetivos

- Diseño de una plataforma de desarrollo basada en PIC32MX795F512 con un factor de forma propio y con multitud de pines de entrada/salida, botones, leds y conectores como USB o tarjeta SD de propósito general para posibles ampliaciones u otros proyectos.
- Diseño de un módulo de comunicación GSM compatible con nuestro factor de forma.
- Diseño de un módulo receptor GPS compatible con nuestro factor de forma
- Diseño de un programa que reciba la información GPS y la procese de manera que sea enviada de forma legible por mensaje de texto y en un dispositivo de almacenamiento SD.

### 1.2. Estructura de la memoria

En el capítulo 2 se explicará de qué trata el proyecto UGRASP y los distintos programas que lo componen así como los objetivos que proponen. Luego se hará una presentación del programa Ícarus, que es en el que está enmarcado este proyecto, entrando en detalle un poco más en sus características y haciendo un análisis del entorno en el que trabajará nuestro dispositivo, el cual deberemos tener en cuenta a la hora de la selección de componentes. Finalmente se verá de forma breve el objetivo principal de este programa junto con una presentación de las placas que vamos a diseñar.

Los capítulos 3 y 4 harán una introducción teórica de los microcontroladores PIC32. En el capítulo 3 se mostrarán las características más importantes de estos microcontroladores y se verá en detalle como está construida su arquitectura y su funcionamiento. Posteriormente, en el capítulo 4 se hará un análisis de las herramientas de desarrollo disponibles y cual se ha escogido para trabajar en función de sus características.

Los capítulos 5, 6, 7 y 8 explicarán en detalle cada una de las placas de circuito impreso que se han fabricado, sus características principales, consideraciones de diseño y las funciones que implementan su funcionamiento.

En el capítulo 9 se muestra cómo ha sido el proceso de puesta en marcha de cada uno de los módulos por separado y el funcionamiento del conjunto. Pueden verse en este capítulo imágenes de los resultados obtenidos en cada una de las comprobaciones mostrando el correcto funcionamiento de cada uno de ellos y el funcionamiento del módulo en general.

El capítulo 10 está dedicado al proceso de fabricación. Este apartado explica las máquinas que han sido necesarias para fabricar las placas y el proceso que se ha seguido para dicha fabricación. Posteriormente se hace una estimación económica teniendo en cuenta el número de horas de trabajo, los servicios contratados y los materiales necesarios para la fabricación de cada una de las tarjetas.

En el capítulo 11 se presentarán las conclusiones del proyecto, especificando en gran parte los componentes que se han utilizado para la fabricación de las placas y por qué se han decidido usarse éstos.

Finalmente tendremos una guía de referencias bibliográficas y 6 apéndices. Los apéndices A,B,C y D contienen todos los esquemáticos y los diseños de las PCBs que se han fabricado; El apéndice E es una referencia de la implementación del código de programa; Por último, el apéndice F contiene las listas de materiales separadas en tablas por placa.



## 2. El proyecto UGRASP (UGR AeroSpace Program)

El proyecto **UGRASP** [12] es un proyecto de innovación docente organizado por el **Departamento de Arquitectura y Tecnología de Computadores** y coordinado por el profesor **Samuel Romero**. Esta iniciativa tiene como objetivo crear un “programa espacial simulado” que suponga un gran reto al estilo del programa “Apollo” de la NASA, pero a pequeña escala. Esta propuesta está orientada a los estudiantes de las Ingenierías en Informática, Telecomunicaciones Electrónica.

El proyecto UGRASP consta de tres programas en los que se abordan distintos objetivos:

**UGRASP-ICARUS:** Este programa, que es en el que se centra este proyecto de fin de carrera, tiene como objetivo la construcción de una **sonda estratosférica** que incorporará un dispositivo de seguimiento y un dispositivo de vídeo con el que se tomarán imágenes del viaje hasta la estratosfera de la tierra y se medirán las condiciones ambientales.

**UGRASP-ULYSSES:** Este programa tiene como objetivo la exploración de localizaciones remotas con vehículos robotizados (rover). Se propone desplegar una serie de misiones de exploración sobre un lugar de difícil acceso. Inicialmente, se ha seleccionado el púlpito o peñón de Canales, en la orilla del embalse del mismo nombre por su compleja orografía. (Figura 1)



Figura 1: Peñón del embalse de canales

**UGRASP-HYDRA:** En este programa se diseñará un cohete propulsado por agua y aire a presión con el objetivo de superar retos de diseño como un **control de actitud** en vuelo para permitir obtener una trayectoria lo más vertical posible, y por tanto, mayor altitud; **control de potencia** del cohete, midiendo la presión interna y administrando la liberación del chorro de propulsión mediante una boquilla regulable; finalmente un automatización de todas las tareas: disponer de un centro de control de la misión con registro en tiempo real, automatizar el lanzamiento y los actuadores en las distintas fases (lanzamiento, desprendimiento de etapas



Figura 2: Water Rocket

consumidas, liberación de la “ carga útil ” o supuesto satélite , despliegue del paracaídas). (Figura 2)

## 2.1. Programa ICARUS

Este proyecto se centra en el programa **ICARUS** del proyecto UGRASP el cual propone la fabricación de una **sonda estratosférica**.

Las sondas estratosféricas son dispositivos que se utilizan para tomar datos o realizar comunicaciones a una altitud de hasta 50 Km sobre el nivel del mar, por encima de donde suelen viajar los vuelos comerciales y por debajo de la capa de ozono.

Habitualmente, las sondas estratosféricas utilizan globos de Helio para subir hasta dicha altitud y así realizar tareas de medición de datos ambientales (temperatura, humedad, presencia de diferentes gases, radiación) y tomar fotografías o vídeo, además de servir en ocasiones como enlaces de telecomunicaciones.

Las sondas estratosféricas son usadas con frecuencia por servicios meteorológicos y por agencias espaciales para medir condiciones atmosféricas a distintas alturas (Figura 3).



Figura 3: Sonda estratosférica

Normalmente, la sonda se compone de:

- Un globo de Helio de grandes dimensiones
- Un paracaídas
- Una barquilla o góndola, con los instrumentos científicos y todos los subsistemas necesarios.

El globo suele ser de un material muy ligero y delgado derivado del látex, y de gran tamaño. Un referente de este material suele ser el Totex[13], que es especial para sondeos meteorológicos. Inicialmente se llena con una cantidad de gas lo suficiente como para proporcionar un ascenso moderado. Conforme el globo sube, la disminución de la presión atmosférica sobre las paredes del globo hará que éste se hinche progresivamente, hasta que a determinada altura (en torno a 30 o 40 Km) explote (Figura 4). Una vez que explote, el paracaídas que hace sujeción entre la barquilla y el globo se desplegará y amortiguará el descenso.

La barquilla, habitualmente de forma cúbica y de material ligero (corcho blanco, por ejemplo) contiene los dispositivos necesarios para la misión: cámaras, GPS, baterías, sensores de diverso tipo, etc. Es normal rellenar con espuma aislante (como poliuretano) todo el espacio dentro de la barquilla una vez ubicados los instrumentos para reducir al mínimo el aire dentro, dado que se pueden alcanzar temperaturas de -60 °C en el exterior. Igualmente, es típico recubrir el exterior de la barquilla con mantas térmicas como aislante. La razón de utilizar una espuma aislante como el poliuretano reside en su bajo coeficiente de conductividad térmica. Al tener poca conductividad, el calor fluirá más lentamente a través de las paredes de la barquilla, por lo que mientras mayor sea



Figura 4: Globo estratosférico en ascenso

el espesor de la capa de espuma, mas tiempo tardará en aumentar la temperatura. Esta propiedad es esencial en nuestro proyecto, ya que la mayoría de los circuitos integrados soportan una temperatura de funcionamiento mínima de -40°C y de esta forma no tendremos este problema durante un largo periodo de tiempo. Además, la fuente de alimentación genera calor, por lo que el flujo de calor desde el exterior puede verse contrarrestado en cierto grado.

Uno de los problemas más habituales en estas sondas es la localización de la barquilla una vez que ha descendido a la superficie y concluye su misión. Durante el ascenso y el descenso, ésta se ve sometida a los vientos de las diferentes capas atmosféricas, que la desviarán de la vertical del punto de lanzamiento hasta varias decenas de kilómetros (Figura 5). Como hemos dicho en el capítulo introductorio, uno de los intereses de este experimento es poder tener estos datos de desviación disponibles para su posible estudio. Estos datos quedarán almacenados en las barquillas por lo que es esencial poder recuperarlas.

Existe una relación directa entre la altura a la que llegará el globo y la velocidad de ascenso. En el momento del lanzamiento deberemos tener esta relación en cuenta dependiendo de cual sea nuestro objetivo, si llegar más alto o que el desvío del globo sea el menor posible. Si el globo se hincha mucho, realizará un ascenso muy rápido y por tanto se desviará menos, pero explotará a menor altura. Si se hincha muy poco, lo justo para ascender, conseguiremos alcanzar mayor altitud, pero el desvío será muy grande. Sería interesante proponer un experimento en el que se analice el grado de desviación en función de la presión del globo y también la altura a la que llegan a explotar. Además, estas variables dependen de las condiciones atmosféricas y de la presión atmosférica, por lo que sería otro parámetro interesante con el que jugar.

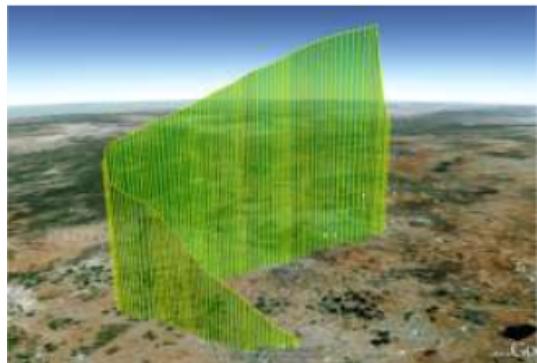


Figura 5: Trayectoria globo

Aunque como hemos dicho, distintos organismos suelen lanzar este tipo de sondas, últimamente se ha popularizado su uso por parte de aficionados y estudiantes, incluso de niveles educativos preuniversitarios ya que es una actividad sencilla, económica y de gran interés educativo debido a la cantidad de información que se puede extraer de los distintos experimentos que se puedan proponer, desde hacer fotos de la estratosfera a tomar datos de presión, temperatura y humedad de las distintas capas atmosféricas puede suponer una actividad muy lucrativa y de gran motivación.

## 2.2. Módulo de geolocalización

Como se ha dicho en el apartado anterior, uno de los principales problemas de las sondas estratosféricas es su localización una vez lanzada y sobre todo al caer, ya que de lo contrario no podríamos recuperar la barquilla perdiendo así la información almacenada en la sonda y la inversión realizada. Este proyecto se centra en el diseño de un **módulo de geolocalización basado en tecnología GPS y GSM**.

Aparte del principal objetivo que es el diseño del localizador, la intención de este proyecto también es la de **diseñar una placa de desarrollo propia que tenga mutitud de entradas/salidas de propósito general disponibles para aplicaciones futuras del proyecto**. Una de las aplicaciones propuestas es el diseño de un **módulo de Control de Actitud** que, una vez en el aire, pueda estabilizar la barquilla mediante el principio de acción y reacción con un control electrónico de unas válvulas que regulan la expulsión de aire a presión.

## 2.3. Tarjetas de Circuito Impreso

Para este proyecto se han diseñado un total de 4 tarjetas para cumplir con los objetivos propuestos en el capítulo 1.1. Éstas se interconectan entre sí a través de una interfaz de pines que proporcionan una comunicación directa entre el microcontrolador y las distintas tarjetas, así como

la alimentación que estas necesitan que proviene de la placa de alimentación. No importa el orden en el que se interconecten ya que un pin de una determinada tarjeta será compartido en las demás que tenga conectadas. Esta interfaz de pines se encuentra mucho más detallada en el capítulo 5.

**Placa de alimentación general:** Tarjeta de propósito general diseñada mediante reguladores lineales. El objetivo de esta placa es alimentar todos los componentes de las tarjetas a los distintos voltajes que requieren cada uno, en nuestro caso 3.3V, 4.2V y 5V. La razón principal de llevar reguladores lineales y no convertidores DC/DC es el ruido que estos generan, ya que en su diseño está implícito el uso de conmutadores que generan mucho ruido y pueden contaminar los demás módulos del sistema así como por ejemplo medidas analógicas, niveles de tensión de alimentación, etc... Este ruido es muy difícil de controlar y puede meterse por los planos de masa y las pistas de alimentación afectando los niveles de voltaje que alimentan los integrados y provocando problemas de reseteos debido a picos de baja tensión. Además, interesa que la tarjeta de alimentación pueda proporcionar la mayor cantidad de corriente posible ya que, aparte de sus futuras aplicaciones que pueden requerir más potencia, componentes de esta misma placa como puede ser el módulo GSM requieren de una fuente capaz de proporcionar hasta 2 amperios de corriente. Ésto en una fuente conmutada se traduce en bobinas grandes, pesadas y caras. El incluir elementos pesados en el sistema es contraproducente ya que cada gramo que lleve la barquilla de más puede ser crucial en el viaje del globo. Por esta razón, las fuentes conmutadas, aunque son más eficientes, necesitan de un estudio más profundo para eliminar efectos de ruido y se reservan para aplicaciones más concretas. Los reguladores lineales por tanto, aunque son menos eficientes, son más flexibles para nuestro proyecto y más fáciles de usar.

**Placa control:** Esta tarjeta es el corazón del módulo de geolocalización. Está basada en un microcontrolador **PIC32MX795F512** que pertenece a la familia PIC32MX del fabricante Microchip. Inspirada en el kit de desarrollo PIC32 USB Started kit II, esta tarjeta tiene todo lo necesario para trabajar con los protocolos de comunicación típicos como el USART, SPI, I2C o CAN así como entradas/salidas de propósito general, convertidores ADC y botones y leds para control de usuario, que la convierten una base perfecta para desarrollar cualquier proyecto que se proponga. Proporciona todo lo necesario para desarrollar aplicaciones USB embebidas host/device/OTG combinando esta placa con el software USB gratuito de microchip. Además dispone de un puerto ICSP (In Circuit Serial Programmer) con el que se puede programar la placa y depurar el código de programa haciendo uso de un dispositivo externo como un pickit 3 o un ICD (In Circuit Debugger).

**Placa GPS:** Para el módulo GPS se ha utilizado una shield de Arduino ya fabricada por la empresa **ITEAD STUDIO**. El motivo de comprar esta tarjeta y no hacerla es porque el módulo GPS que incluye es un poco especial debido a que sobrepasa la altitud establecida por el límite COCOM. Este módulo era un poco desconocido y se encontró poca información de diseño, por lo que se optó por comprarla. Esta placa tiene un factor de forma compatible

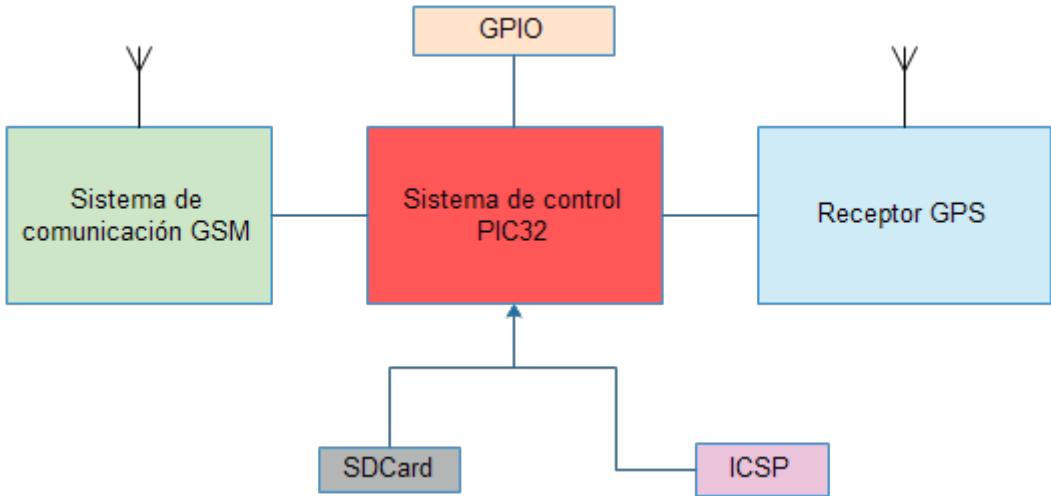


Figura 6: Diagrama de bloques del sistema

con **Arduino** por lo que se ha diseñado un tarjeta adaptadora del factor de forma propio que se ha acogido en nuestro proyecto al factor de forma de Arduino. Además incluye un slot microSD que podremos aprovechar y que resulta muy interesante, ya que es capaz de funcionar con una alimentación de 5V o 3.3V.

**Placa GSM:** Esta placa está basada en un módulo de comunicaciones GSM/GPRS **SIM900** del fabricante SIMCOM. El SIM900 es un módulo muy extendido entre la comunidad de Arduino, por lo que existen muchos proyectos de referencia en internet disponibles que pueden ser muy útiles a la hora de realizar cualquier diseño. La razón del éxito de este módulo es en gran parte por su relación calidad-precio. Tiene todas las características necesarias que se le pueden requerir a un módulo de este tipo como es una conexión a Internet a velocidad 3G y el envío de mensajes de texto, todo esto por un precio que ronda los 9€ (dependiendo del distribuidor). En nuestro proyecto, éste módulo será el encargado de enviar un SMS cada cierto periodo de tiempo predefinido con la información sobre la posición de la barquilla.

La figura 6 muestra un diagrama de bloques general de nuestro dispositivo.



### **3. Introducción a la Arquitectura PIC32**

En el año 2007 Microchip introduce su gama de microcontroladores más potente que la compañía había diseñado hasta la fecha.

Su primera familia fue el PIC32MX3, cuyos dispositivos eran en montaje superficial y contaban con un pinout de 68 y 100 terminales.

El PIC32 es el primer microcontrolador de la historia de Microchip no diseñado por la compañía sino que fue encargado a otra empresa llamada MIPS Technologies, la cual es muy conocida en el campo de los núcleos de 32 y 64 bits, por ser una desarrolladora de núcleos para terceros de 32 y 64 bits.

Dicho núcleo se incluyó en el interior de un PIC24H128, al cual Microchip lo adaptó para poder colocar su nuevo CORE, de esta forma el usuario goza de un núcleo de 32 bits de muy altas prestaciones junto con los periféricos que tienen de los PIC24H.

Actualmente la familia está integrada por los PIC32MZ, que es la última familia de microcontroladores liberados por la compañía y son capaces de funcionar a una frecuencia de hasta 200MHz.

#### **3.1. Características de los PIC32**

Para comenzar con nuestra descripción diremos que un PIC32 tiene internamente una arquitectura mixta, ya que dentro de su CORE, su arquitectura es Harvard, es decir, que existe un bus de instrucciones y otro de datos. Pero si lo vemos desde la memoria de programa y la memoria de datos, el PIC tiene una arquitectura Von Newman, por lo que utilizan el mismo dispositivo de almacenamiento tanto para las instrucciones como para los datos.

Esta dualidad termina con el viejo problema de Microchip de los accesos a la memoria de programa para la lectura de tablas de datos, por existir dos buses diferenciados, y que las arquitecturas anteriores habían salvado por medio de la implementación de una herramienta complementaria; en el caso de PIC18, se había utilizado un mecanismo de “punteros de tabla” y un registro TABLAT; en los micros de 16 bits, PIC24 y dsPIC, se usaba un recurso mediante el cual se podía visualizar parte de la memoria de programa desde la RAM. Estos mecanismos, aunque eficientes, perdían ciclos de máquina y fue por ello por lo que MIPS decidió que ambas teorías podían enlazarse con un mismo BUS de datos, ya que éste era de 32 bits y tenía el espacio suficiente para poder transferir las instrucciones.

Pero Microchip, para acelerar el procesamiento de las mismas solicitó a MIPS que en el interior

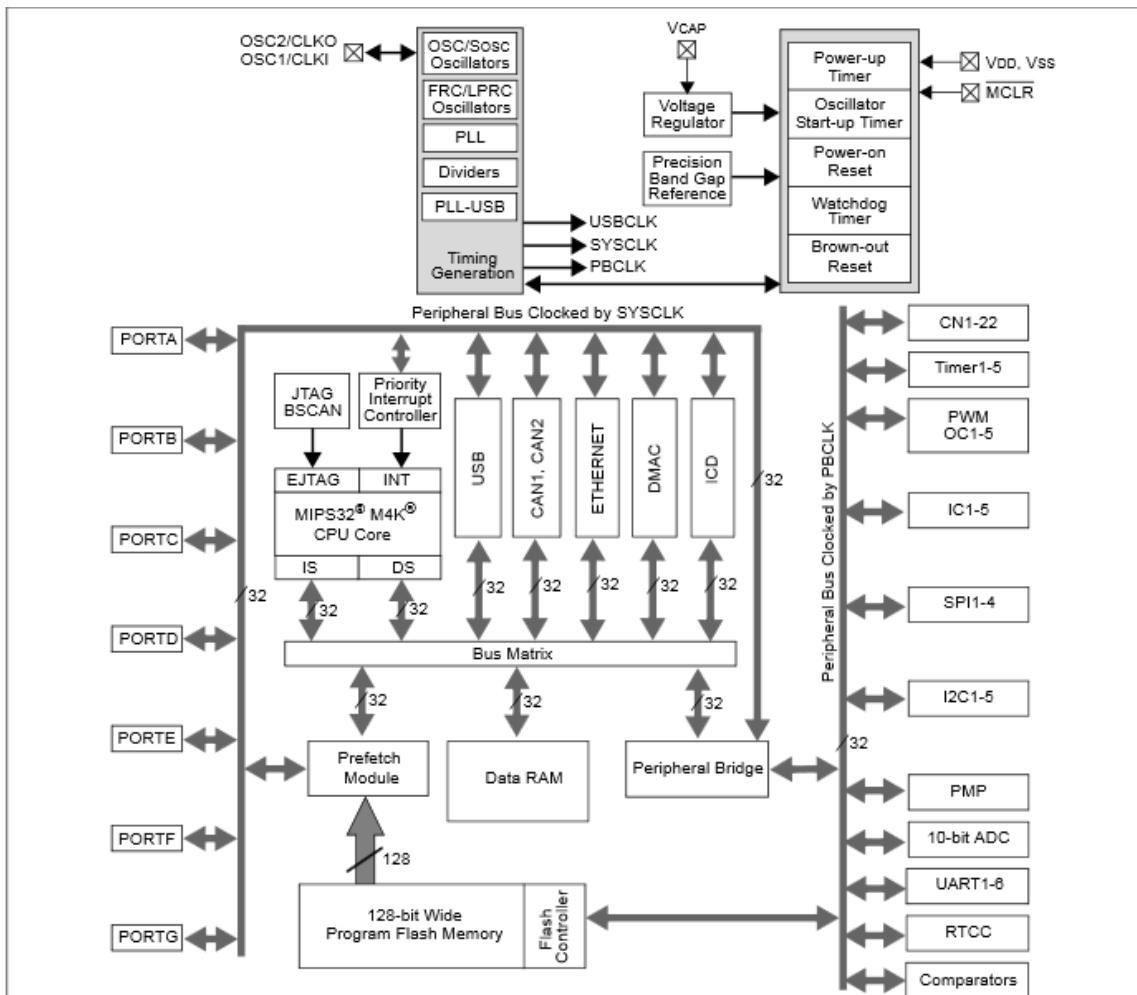


Figura 7: Arquitectura del PIC32MX795F512L

de CORE, las instrucciones y los datos se separasen. De esta manera las instrucciones podían ser procesadas por un nuevo pipeline de 5 niveles. Esto mejoró el rendimiento del procesador permitiendo ejecutar las instrucciones en un pulso de reloj, con lo cual, estando el pipeline cargado, el PIC32 puede procesar 80MIPS a 80MHz.

El corazón del PIC32 es el núcleo **M4K**. Una interfaz JTAG mejorada (EJTAG) está implementada con acceso directo al núcleo para soportar las funciones de debugging. El núcleo tiene 2 buses, uno usado para buscar instrucciones y el otro para acceder a los datos.

A fin de garantizar el rendimiento mas alto posible de transferencia de datos e instrucciones se introdujo un “Bus Matriz” en lugar de la típica arquitectura “dato y direcciones”, bus común en las CPU de menor porte. El bus matriz puede visualizarse como una autopista con múltiples carriles

en comparación con el bus de vía única de un microcontrolador típico.

Debido a que hay disponibles muchos carriles para el tráfico, los datos y las instrucciones pueden ser transmitidos desde muchas fuentes y hacia muchos destinos en un mismo ciclo máquina. Naturalmente hay limitaciones en cuanto a lo que es posible, pero el hardware se asegura de que todo el mundo tenga su tiempo justo en el BUS. De hecho el programador puede seleccionar entre diferentes variantes de prioridad de acceso. El bus matriz está funcionando a la misma frecuencia que el core de la CPU.

La memoria FLASH almacena constantes e instrucciones del programa y la RAM almacena los datos y las variables, estando ambas conectadas al bus matriz. En la memoria también se puede observar un elemento nuevo: un módulo de prebúsqueda.

La FLASH implementada en los PIC32 no puede entregar la información a la CPU más rápido de lo que la CPU puede consumir dicha información. Con el fin de resolver este desequilibrio el módulo de prebúsqueda ofrece 2 características:

1. Un mecanismo de CACHE que puede almacenar 64 instrucciones de 32 bits. La propia CACHE es una memoria RAM de tecnología de alta velocidad que puede entregar datos e instrucciones al núcleo a la misma frecuencia que el micro está ejecutando el código.
2. Mecanismo de captación previo que usa un algoritmo de “predicción de salto”, mediante el cual se carga la CACHE con instrucciones antes de su tiempo de ejecución.

Ambas características ayudan a que el código se ejecute con la mayor velocidad posible. Además cabe señalar que cada ciclo de búsqueda lee un bloque de datos de 128bits, lo que equivale a 4 instrucciones de 32 bits (u 8 para las instrucciones de 16bits llamadas MIPS16e).

El resto de periféricos están conectados a un puente de periféricos. Dicho puente puede trabajar a la misma velocidad que el núcleo o a otra alternativa a través de un divisor de frecuencia.

En la Figura 7 puede verse un esquema general de la arquitectura de nuestro PIC32.

### **3.2. El CORE PIC32 (M4K)**

El núcleo es capaz de ejecutar una instrucción por cada ciclo reloj. Está compuesto por 32 registros de 32 bits disponibles para el almacenamiento de datos locales, y otro conjunto completo de 32 registros de 32 bits que pueden ser utilizados por la rutina de interrupciones, reduciendo así la sobrecarga de entrada y salida de la interrupción.

La ALU es de 32 bits de ancho y soporta instrucciones del tipo MAC (Multiplica y Acumula) en un solo ciclo. Cuando el espacio del código generado es muy crítico, es posible compilar la totalidad

o parte del código siguiendo las instrucciones MIPS16e donde las instrucciones son de 16 bits en lugar de 32. Esto permite ahorrar espacio en memoria de programa, pero aumenta el tiempo de ejecución de forma drástica.

Como en todos los microcontroladores PIC, los datos se almacenan en forma Little Endian, es decir, el byte menos significativo se encuentra en la dirección mas baja de la memoria. El corazón del núcleo es la unidad de ejecución, la cual se ocupa de la aritmética estándar, las operaciones de rotación y las instrucciones de desplazamiento.

El propio núcleo utiliza un pipeline de 5 etapas necesario para cargar, ejecutar y guardar el resultado de cada instrucción, lo que requiere de un total de 5 ciclos de reloj, sin embargo cuando el pipeline está totalmente cargado, la CPU es capaz de ejecutar cada instrucción en un solo ciclo de reloj.

Las instrucciones de salto implican un retardo adicional, conocido como “retardo de salto”, debido a la estructura secuencial del pipeline, sin embargo la mayoría de los compiladores y ensambladores llenan ese retardo con una instrucción útil (siempre que sea posible), eliminando así cualquier pérdida de rendimiento en la mayoría de los casos.

Se suma al núcleo la Unidad de Multiplicación y División (MDU), la cual realiza cálculos de 16x16 y 16x32 bits en un solo ciclo máquina, mientras que las multiplicaciones de 32x32 bits usan 2 ciclos máquina. Las instrucciones de división necesitan de 35 ciclos pero de forma inteligente usa tan solo 13 ciclos si se detecta que la división es de datos de menos de 32 bits.

El compilador debe reordenar las instrucciones necesarias para garantizar que el resultado estará disponible cuando el pipeline del núcleo lo requiera. Dado que esta operación se ejecuta en paralelo al núcleo y con el reordenamiento inteligente de las instrucciones por parte del compilador, no debería haber pérdidas de rendimiento.

En el momento del arranque el núcleo no es consciente de muchos de los datos necesarios para su funcionamiento como por ejemplo cómo se conecta la memoria disponible y no tiene ningún registro de estado implementado. Con el fin de remediar esta situación existe lo que se denomina el coprocesador cero (copro 0).

El coprocesador 0 sirve para implementar el registro de estado del núcleo, configurar el contador de memoria caché y programar otras características.

Antes de que cualquier código se ejecute, los registros del coprocesador 0 deben ser inicializados apropiadamente. Esta acción está realizada por un código conocido como “código de arranque” y

se vincula automáticamente al código de la aplicación.



## 4. Herramientas de desarrollo

### 4.1. MPLABX

El fabricante Microchip ofrece un entorno de desarrollo profesional llamado **MPLABX**. Este entorno de desarrollo está basado en el IDE Netbeans creado por Oracle para el desarrollo de java y funciona sobre un compilador propio desarrollado por Microchip llamado XC32. Este compilador ofrece una versión gratuita y otra de pago. Esta versión gratuita no ofrece la **compilación de código C++** ni la **optimización de código**, características que si ofrece la versión de pago. Esta herramienta es de las mejores opciones que se puede encontrar en el mercado de los microcontroladores ya que, pese a tener limitaciones de pago, se pueden desarrollar proyectos profesionales completos en C sin prácticamente ninguna limitación. Además de esto, todas las librerías proporcionadas por microchip están escritas en C.

Hasta hace relativamente poco tiempo, Microchip tenía todas las librerías y la documentación disgregada y costaba un poco iniciarse en el desarrollo con estos microcontroladores. Pero desde la salida de la última familia de microcontroladores (PIC32MZ), Microchip ha desarrollado **MPLAB Harmony**. Harmony es una unificación de todas las librerías que tenía Microchip y la creación de algunas nuevas, proporcionando un framework desarrollado en C eficiente con una extensa documentación unificada que hace que, junto con la versión gratuita del compilador XC32, tengamos una herramienta muy potente para desarrollar prácticamente cualquier tipo de proyecto. Entre otras características muy llamativas de este framework se encuentra un plugin (MPLAB Harmony Configurator) que permite configurar todos los pines, relojes y periféricos del microcontrolador de forma gráfica. Cabe destacar también que Harmony implementa una filosofía de programación orientada a máquinas de estados basado en interrupciones. Es muy importante leer el **getting started** de la ayuda de este framework para iniciarse en él.

Como se ha indicado anteriormente, este framework es relativamente joven y está sobre todo enfocado a la última familia de microcontroladores de Microchip. Por lo que se han encontrado algunos problemas con la antigua familia de microcontroladores de Microchip que es con la que estamos usando en este proyecto (PIC32MX). Es por esto que nos hemos inclinado por usar un entorno de desarrollo basado en **Arduino** y que está más centrado en esta familia de microcontroladores.

### 4.2. Arduino

A día de hoy, Arduino es la plataforma de desarrollo más extendida entre las personas que se inician en el mundo de la electrónica.

Arduino es una plataforma de prototipos electrónicos de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como



Figura 8: Logo de arduino

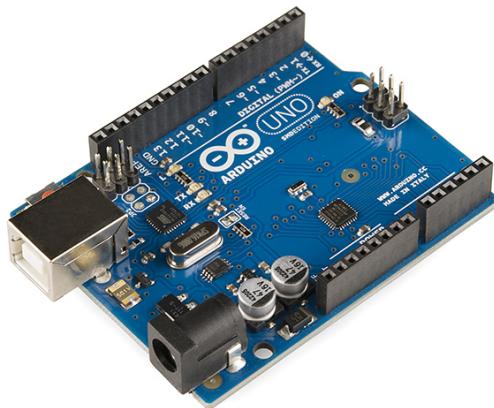


Figura 9: Placa Arduino UNO

hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Arduino puede monitorizar el entorno y puede actuar en el medio mediante el control de luces, motores y otros dispositivos. El microcontrolador de la placa se programa usando el **Arduino Programming Language** (Basado en Wiring) y el Arduino Development Environment (Basado en Processing).

Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en un ordenador (Por ejemplo Flash, Processing, MaxMSP,etc...).

Las placas se pueden ensamblar a mano o encargarlas preensambladas; el software se puede descargar gratuitamente. Los diseños de referencia del Hardware (Archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas (Figura 9).

El verdadero éxito que de esta plataforma de desarrollo se debe a su lenguaje de programación

**Wiring.** Este lenguaje está basado en compilador GCC, por lo que está escrito en C++ y orientado a objetos. Esto hace que junto a la librería que trae consigo sea muy intuitivo para una persona que se está iniciando en la programación de microcontroladores. Si a esto le sumamos la gran aceptación de la comunidad y la cantidad de proyectos diferentes publicados en forma de código abierto junto con la cantidad de módulos de todo tipo diseñados para el factor de forma Arduino, hacen de esta plataforma una apuesta segura y sencilla para desarrollo rápido y para iniciados.

#### 4.3. La plataforma CHIPKIT

La empresa DIGILENT desarrolló toda una plataforma de placas Arduino compatibles basadas en microcontroladores PIC32 de Microchip. La placa en la que nos vamos a basar es la MAX32 de este fabricante, ya que está basada en el mismo microcontrolador que estamos usando y aprovecha al máximo sus periféricos.

La placa MAX32 está pensada para ser usada con el Multi-Platform IDE (MPIDE). Digilent produjo la plataforma de desarrollo MPIDE por medio de la modificación del IDE de Arduino. Por este motivo es totalmente compatible con Arduino, por lo que cualquier proyecto realizado en MPIDE puede ser portado fácilmente a Arduino.

Para programar esta placa se necesita un convertidor de Serial TTL 3.3V a USB. Esto es así ya que la placa viene equipada con un código **bootloader** o cargador de arranque que se ejecuta en la placa y que le permite autoprogramar al microcontrolador a partir de los códigos que se descargan desde el MPIDE.

Antes de utilizar MPIDE hay que instalar los drivers que vienen con el adaptador. Posteriormente se especificarán las conexiones del adaptador Serie-USB.

Al poner el pin MCLR en bajo, se resetea el microcontrolador, y se ejecuta con la gestión del arranque.

#### 4.4. Bootloader

El bootloader o gestor de arranque de los dispositivos PIC se utiliza para actualizar el firmware en un dispositivo sin la necesidad de un programador externo.

Se inicia el código del gestor de arranque cuando se produce el RESET del dispositivo. Si no existen eventos para entrar en el modo de actualización del firmware, el gestor de arranque comienza la ejecución de la aplicación del usuario. El gestor de arranque realiza operaciones de borrado y auto programación de la memoria flash cuando se encuentra en el modo de actualización del firmware.



Figura 10: Ilustración del entorno de desarrollo MPIDE

Este programa es muy pequeño y utiliza recursos mínimos de memoria del microcontrolador. En el momento del RESET se toma el control del arranque del microcontrolador, de ahí su nombre “gestor de arranque” o bootloader, y si no se detecta la “puesta en modo auto programación” salta a la primera instrucción del programa de aplicación realizado por el usuario. A partir de ese momento el control del microcontrolador queda a cargo del programa de usuario y el bootloader queda inactivo hasta que ocurra un RESET del micro.

#### 4.5. MPIDE

Es el entorno de desarrollo propio que ha desarrollado DIGILENT para trabajar con su placa Arduino compatible para PIC32. Este entorno es totalmente compatible con los entornos IDE de otras plataformas para otros MCU compatibles (Figura 10).

## 5. Placa de Control

Como se ha mencionado anteriormente, la placa de control funciona con un **PIC32MX795F512** de la familia MX de Microchip. En el momento de diseño inicial, esta placa se basó en el diseño de la placa de desarrollo Microchip Ethernet Starter Kit, pero como se ha mencionado en el Apartado 4.1, este chip está dando problemas con el framework MPLAB Harmony, por lo que finalmente se buscó una placa equivalente entre la familia de placas del fabricante DIGILENT, por lo que se ha hecho una equivalencia de pines que se muestran en el **Cuadro 1** de nuestra placa con la “**CHIPKIT MAX32**”.

MAX32	PIC32	MAX32	PIC32	MAX32	PIC32	MAX32	PIC32
0 USART0RX	RF2	24	VBUS	48	RD8	72	RA4
1 USART0TX	RF8	25	USBID	49	RD11	73	RA5
2	RE8	26	USBD-	50 SDMISO	RG7	74	RD9
3 PWM	RD0	27	USBD+	51 SDMOSI	RG8	76 BUT 1	RD13
4	RC14	28	RG15	52 SDSCK	RG6	77 BUT 3	RD7
5 LEDV	RD1	29	RG7	53 SDSS	RG9	78	RG1
6 LEDA	RD2	30	RE7	AN0	RB0	79	RG0
7	RE9	31	RE6	AN1	RB1	80	RA6
8	RD12	32	RE5	AN2	RB2	81	RA7
9 LEDR	RD3	33	RE4	AN3	RB3	82	RG14
10	RD4	34	RE3	AN4	RB4	83	RG12
11	RC4	35	RE2	AN5	RB5	84	RG13
12	RA2	36	RE1	AN6	RB6	85	RA9
13	RA3	37	RE0	AN7	RB7		
14	RF13	38	RD10	AN8	RB8		
15	RF12	39	RD5	AN9	RB9		
16	RF5	40	RB11	AN10	RB10		
17	RF4	41	RB13	AN11	RB11		
18	RD15	42	RB12	AN12	RB12		
19	RD14	43	RG8	AN13	RB13		
20 SDA 1	RA15	44	RA10	AN14	RB14		
21 SCL1	RA14	45	RF0	AN15	RB15		
22	RC2	46	RF1	70	RA0		
23	RC3	47 BUT 2	RD6	71	RA1		

Cuadro 1: Correspondencia de pines

La Figura 11 muestra el diagrama de pines de nuestra placa de desarrollo. Como se puede ver en la figura, se han incluido tres **Leds** y tres **Botones de propósito general** para usarlos como indicadores o añadir funcionalidades.

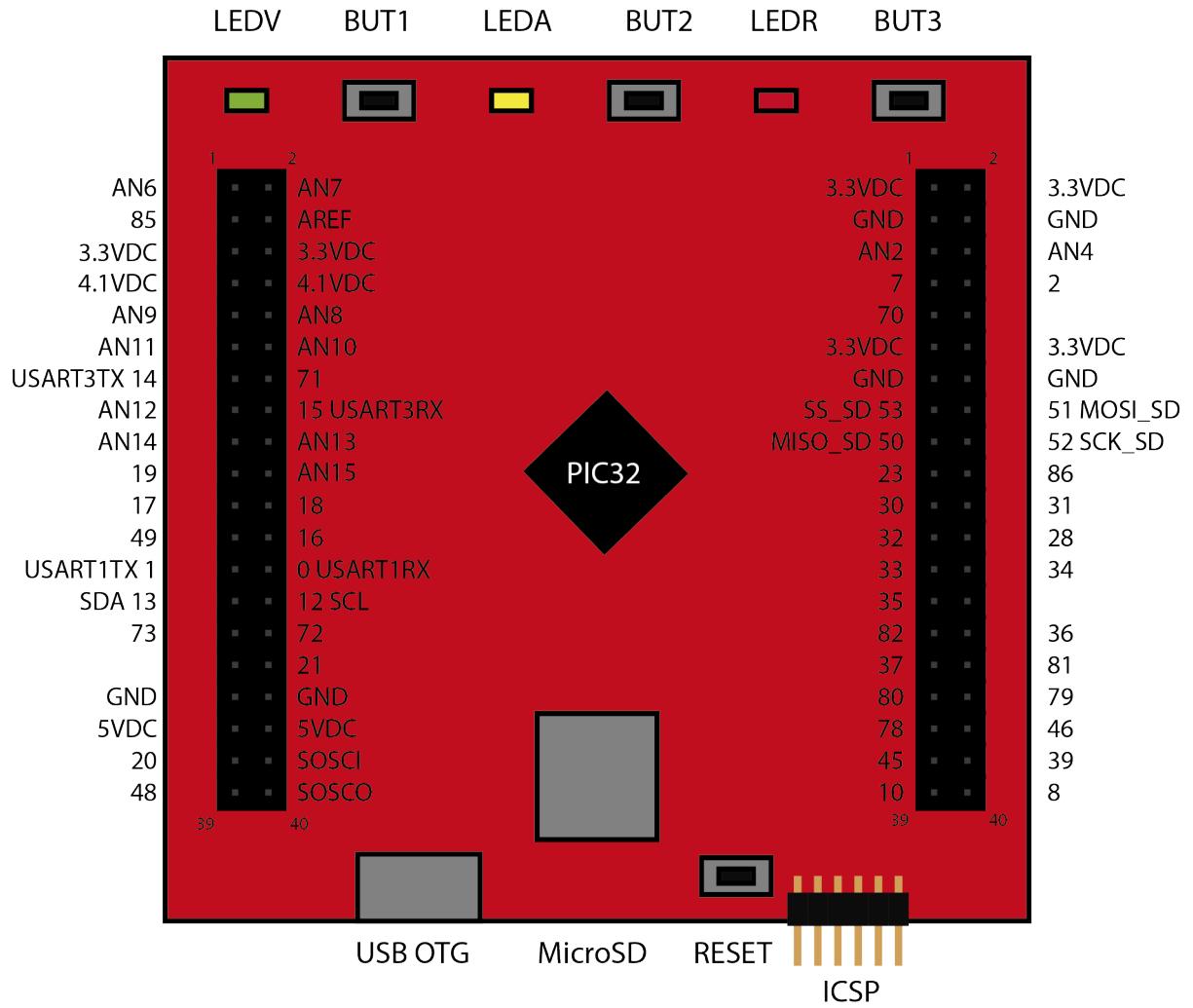


Figura 11: Configuración de pines de la placa de control



Figura 12: Pickit 3

Aunque no se hayan usado en este proyecto, quedan disponibles una **ranura para tarjetas MicroSD** y un **puerto USB** con capacidad On The Go (OTG) para otras aplicaciones.

La ranura de la tarjeta SD estuvo concebida en un principio para ser usada con el framework Harmony, pero al dar tantos problemas y cambiar de plataforma se ha optado por usar la SD que trae consigo la placa GPS, la cual no ha dado ningún fallo. Los pines usados para esta conexión son el 50, 51, 52 y el 53, los cuales están especificados en el **Cuadro 1**.

También se disponen en un lateral de la placa 6 pines que están destinados a usarse con un programador externo como el **PICKIT 3** (Figura 12) de Microchip. Este programador permite programar la placa y depurar en tiempo de ejecución. Este programador se puede usar **sólo** a través del entorno de desarrollo MPLABX y con el compilador de Microchip. Para la plataforma MPIDE **no es necesario usar este programador** ya que la programación del chip se hace a través del **Puerto Serie USART 0** mediante cualquier adaptador USB-TTLSerial 3.3V. En nuestro caso hemos usado un **Cable FTDI TTL-232R-3V3** (Figura 13).

El programador **Pickit 3** es una muy buena herramienta cuando se está usando la plataforma MPLAB X. Esta herramienta permite programar la placa directamente sin ninguna necesidad de usar un **Bootloader**. Además permite depurar el código en tiempo de ejecución pudiendo hacer paradas en los puntos del código que queramos mientras se está ejecutando el código y poder así ver los valores de las variables a través de la herramienta **Visor de Variables** de MPLABX. La plataforma MPIDE utiliza un bootloader para cargar las aplicaciones en el microcontrolador, por lo que el único uso del Pickit 3 en esta plataforma ha sido para grabar el bootloader en el chip. En cuanto al debug en la plataforma MPIDE, al igual que en Arduino, se hace a través del monitor serie que incluye la plataforma. Esta herramienta permite mostrar información por pantalla a través de los comandos “Serial.print()” “Serial.println()” o “Serial.write()”. Así mismo, permite leer caracteres o



Figura 13: Cable FTDI TTL-232R-3V3

cadenas que se introducen en el campo de texto. En definitiva, el monitor serie muestra toda la información que está pasando por el bus serie. Tener en cuenta que debemos ajustar la velocidad de comunicación del puerto serie para observar correctamente lo que está pasando por el (Figura 14)

Para programar la aplicación con MPIDE hay que **conectar los pines Rx y Tx del cable FTDI con los Pines Tx y Rx del puerto USART 0 de la placa de desarrollo respectivamente**. Para que el bootloader entre en modo programación tiene que aplicarse un reset a la placa al iniciar la programación. Esto lo hace MPIDE de forma automática, pero tenemos que **conectar el pin RTS# (Verde) con el pin izquierdo del conector ICSP**. Para que MPIDE muestre información del proceso de compilación/programación hay que pulsar dichos botones en el programa **pulsando la tecla SHIFT**.

En el **Apéndice A: Esquemáticos de la placa de desarrollo** se pueden encontrar los esquemáticos completos de esta placa.

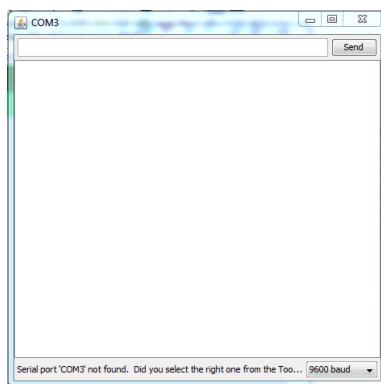


Figura 14: Monitor serie MPIDE

## 6. Módulo de GPS

En esta parte del proyecto se va a programar un shield de Arduino para GPS de la compañía Ublox cuyo modelo es el NEO-6M. La razón para usar este componente y no otro GPS cualquiera del mercado es debido a que el límite COCOM de altura a la cual el dispositivo deja de funcionar está a 50000 metros de altura frente a los 18000 metros que normalmente ofrecen otros dispositivos. Otra de las razones por las que se ha escogido este dispositivo es que posee una sensibilidad de recepción de señal de -160dBm frente a los -140dBm que trae por ejemplo el SIM908 que fue nuestra primera opción. Además trae consigo una antena GPS adecuada para este dispositivo. La Figura 15 muestra una imagen de la placa que vamos a usar.



Figura 15: Shield Arduino Ublox NEO-6M

Al tratarse de una shield de Arduino, no es compatible con nuestra placa de desarrollo por lo que se ha diseñado una placa adaptadora que hace de interfaz entre nuestra placa y la placa de GPS. En el **Apéndice B: Esquemáticos de la placa GPS** puede verse el esquemático de esta placa adaptadora.

## 6.1. El límite COCOM

Muchas de las unidades GPS comerciales que hay por el mundo se desactivan automáticamente si van más rápido de 1.900 kilómetros por hora o si se van por encima de los 18.000 metros de altura.

Es lo que se conoce como el “límite COCOM”, que se utiliza para referirse a un límite impuesto a los dispositivos GPS que debe deshabilitar el seguimiento cuando el dispositivo se da cuenta de que se mueve más rápido de 1.000 nudos (1.900 km / h; 1,200 mph) y a una altitud mayor de 60.000 pies (18.000 m).

Con ello se pretendía evitar el uso de GPS para que funcionaran en los misiles balísticos intercontinentales.

Muchos fabricantes siguen aplicando este límite, literalmente, mientras que otros han optado por directamente deshabilitar el seguimiento simplemente cuando un único límite es alcanzado.

Este límite está poniéndole las cosas difíciles a algunos usuarios, entre ellos, los **aficionados a los globos de gran altitud**, que suben muy alto, pero no tienen nada que ver con un peligro para la seguridad nacional.

Efectivamente, el límite fue promulgado a petición de los EE.UU., que apadrinó la creación de COCOM (acrónimo de Coordinating Committee for Multilateral Export Controls), una organización internacional para controlar el comercio internacional de productos estratégicos y de datos técnicos de los países miembros hacia países prohibidos (considérados no aliados).

COCOM era el némesis del famoso COMECON, y fue establecido por los poderes del bloque occidental en los primeros cinco años después del final de la Segunda Guerra Mundial, durante la incipiente Guerra Fría, para ejecutar el embargo de armas a los países del Pacto de Varsovia.

Aunque la organización COCOM dejó de funcionar el 31 de marzo de 1994, para dejar paso al Acuerdo de Wassenaar, y su lista de tecnología restringida a estos países se ha ido diluyendo en el tiempo, siguen vigentes algunos aspectos como, por ejemplo, los receptores GPS.

Arduino PIN	Descripción
D0	Data
D1	Din
D2-D9	-
D10	CSN
D11	MOSI
D12	MISO
D13	SCK
A0-A3	Breakout
A4	IIC_SDA
A5	IIC_SCL

Cuadro 2: Descripción de pines de la Shield Ublox NEO-6M

### 6.2. Características de la tarjeta de navegación:

- Interfaz MicroSD con la que se podrá ir grabando la trayectoria que sigue el dispositivo.
- Compatible con niveles de tensión de 5V o 3.3V.
- Antena de alta sensibilidad.
- Interfaz UART (baud rate: 38400bps)
- Interfaz SPI
- Interfaz IIC
- Rango de temperatura de operación: -40°C hasta 85°C.

### 6.3. Hardware

En la Figura 16 puede verse un esquema general de la PCB con la situación de todos los componentes, así como la disposición de las entradas/salidas. Esta disposición queda descrita en el Cuadro 2, donde se muestra una especificación mas detallada de la funcionalidad de cada pin de la interfaz de interconexión. En el **Apéndice B: Esquemáticos de la placa GPS** de este documento podrá encontrar los esquemáticos completos.

Como esta placa tiene una interfaz de conexiones compatible con Arduino, la cual es incompatible con la nuestra, la tarea de diseño con este módulo se reduce a diseñar un adaptador que interconecte las dos interfaces. Se ha diseñado una tarjeta adaptadora que conecta el puerto serie del módulo GPS al puerto “Serial3” de la placa de control así como la interfaz del slot microSD con los pines 50, 51, 52 y 53 de la placa de control. El esquema de interconexión de esta placa así como su esquemático se encuentra en el Apéndice B.

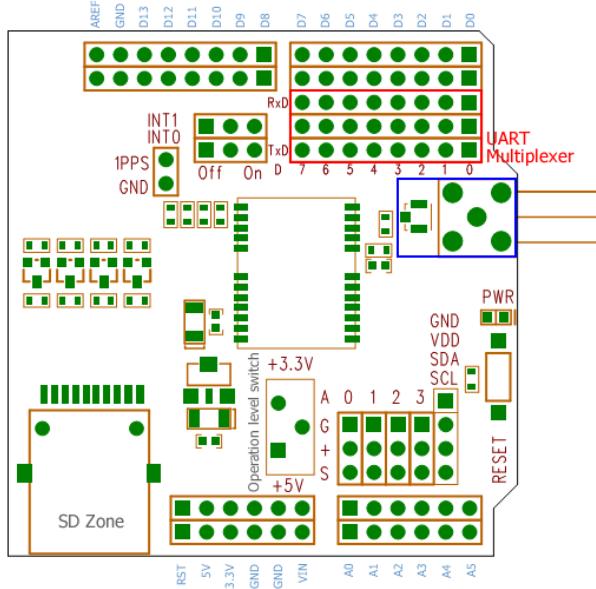


Figura 16: Diagrama de la Shield Ublox NEO-6M

#### 6.4. Módulo NEO-6M

El módulo NEO-6M es un módulo GPS de alto rendimiento diseñado por el fabricante **Ublox**. Mediante este módulo obtendremos, una vez capturada la señal de al menos 4 satélites, información precisa sobre la posición, altura, velocidad y hora. Algunas de sus características son:

- Comunicación GPS por protocolo NMEA, UBX (Protocolo propio) y RTCM.
- Alimentación desde 2.7 V hasta 3.6 V.
- Protocolo de comunicación USART, USB, I2C y SPI.
- Sensibilidad de recepción de señal -160dBm.
- Límites operacionales:
  - Aceleración  $\leq 4g$
  - Altitud  $\leq 50000$  metros
  - Velocidad  $\leq 500$  m/s

La **Figura 17** muestra un diagrama de bloques del módulo NEO-6M

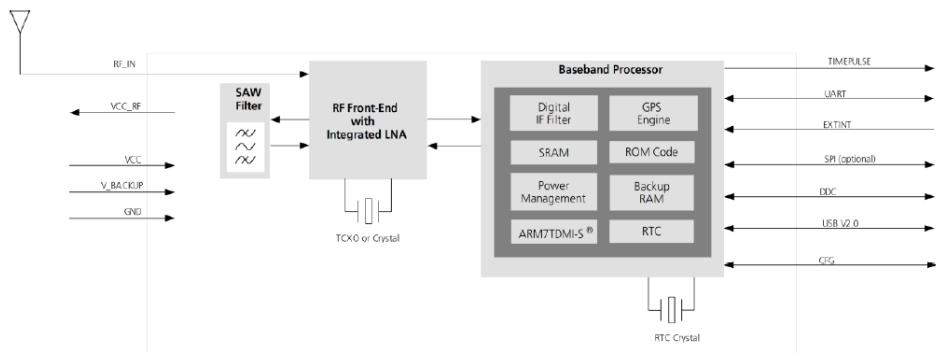


Figura 17: Diagrama de bloques del módulo Ublox Neo-6M

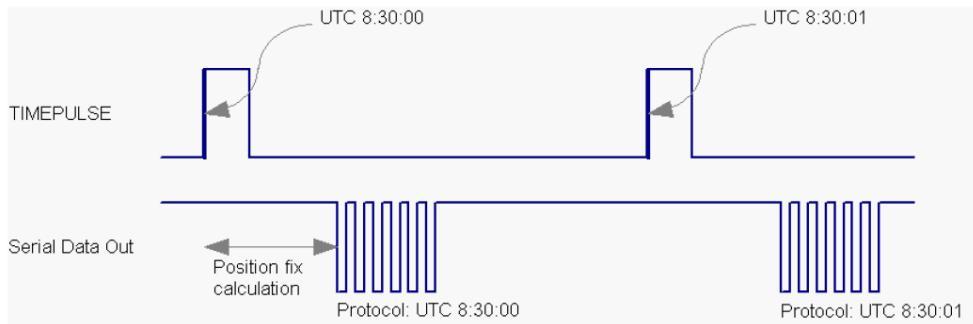


Figura 18: Señal TimePulse

#### 6.4.1. Timepulse

El NEO-6M tiene una salida llamada **TimePulse** que proporciona una señal de pulso de frecuencia variable. Esta salida se puede usar para la sincronización o como frecuencia de referencia. Por defecto está configurado a un pulso por segundo. Esta señal está relacionada con la frecuencia a la que se envían las tramas NMEA. En este caso, al estar configurado a un segundo, la serie de tramas se enviará cada segundo. En el caso de que cambiáramos su frecuencia a 100 ms por ejemplo, la serie de tramas se enviaría cada 100 ms. El problema de ésto es que puede que no haya terminado de enviar la información cuando vuelva a empezar otra vez a enviarla. La figura 18 muestra este proceso. En nuestro caso, lo dejaremos con la configuración por defecto y la utilizaremos para tener un contador de un segundo de periodo que usaremos para enviar un mensaje cada minuto por ejemplo.

El mensaje del protocolo UBX “**UBX-TIM-TP**” proporciona información temporal para el siguiente pulso, fuente temporal y cuantización de error del pin de salida.

Para mas información sobre como configurar este parámetro consultar la **Sección 11 de u-blox 6 Receiver Description Including Protocol Specification [18]**

#### 6.4.2. Antena

El módulo NEO-6M está diseñado para usarse con antenas pasivas y activas. En nuestro caso utilizaremos antenas pasivas por simplicidad y ser más baratas. Algunas de las características son:

- Ganancia mínima: 15 dB (para compensar pérdidas de señal en el cable RF)
- Ganancia Máxima: 50 dB
- Máxima figura de ruido: 1.5 dB
- Frecuencia: 1.575,42 Mhz. +/- 10 Mhz

#### 6.4.3. Power Management

El u-blox soporta diferentes **Modos de Alimentación**. Estos modos de alimentación representan estrategias sobre cómo controlar la adquisición de datos y el motor de seguimiento para conseguir el mejor rendimiento posible o el menor consumo de potencia. Estos modos de potencia se seleccionan mediante el mensaje UBX “**CFG-RXM**”.

**Maximum Performance Mode:** Un receptor en modo de **máximo rendimiento** hace uso del motor de adquisición de datos buscando todos los satélites. Una vez que el receptor tiene la posición fijada (Position Fix) o si la información de pre-posicionamiento está disponible, el motor de adquisición continúa buscando satélites que no han sido captados.

**Eco Mode:** Un dispositivo en Eco-mode actúa exactamente igual que en modo de máximo rendimiento durante el inicio del dispositivo. Una vez que la posición puede ser calculada con el suficiente número de satélites, el motor de adquisición de datos se apaga resultando en un significativo ahorro de energía.

**Power save mode:** Este modo permite una reducción de consumo activando o desactivando selectivamente partes del receptor. Para más información sobre este modo mirar el punto 9.3 del “u-blox 6 Receiver Description including Protocol Specification”

En nuestro caso estamos trabajando en el modo de máximo rendimiento ya que el consumo no es excesivamente grande y uno de nuestros objetivos principales es la adquisición constante de datos, por lo que el ahorro de energía no es tan relevante como el perder información sobre la trayectoria.

#### 6.4.4. Modos de funcionamiento

La tecnología de posicionamiento de u-blox soporta diferentes modelos para ajustar el motor de navegación al entorno de navegación esperado. Estos ajustes pueden ser cambiados dinámicamente sin tener que realizar un reset al dispositivo. Los ajustes mejoran la interpretación de las medidas y por lo tanto proporcionan una adquisición de la posición más precisa. Ajustando el módulo a un modelo de navegación inapropiado puede resultar en una pérdida de rendimiento del dispositivo y

de precisión.

Estos ajustes están relacionados con el mensaje UBLOX de configuración “**CFG-NAV5**”

Los distintos modelos de navegación se exponen a continuación:

**Portable:** Configuración por defecto. Aplicaciones con poca aceleración como por ejemplo dispositivos portátiles. Esta configuración es apropiada para la mayoría de situaciones. Max altitud: 12000 metros; Max velocidad: 310m/s; Máxima velocidad vertical: 50m/s.

**Stationary:** Usada en aplicaciones de timing (la antena debe ser estacionaria) u otra situación estacionaria. Velocidad limitada a 0m/s; Se asume una dinámica 0 ; Máxima altitud: 9000 metros; Máxima velocidad: 10m/s; Máxima velocidad vertical: 6 m/s.

**Pedestrian:** Aplicaciones con poca velocidad y aceleración como por ejemplo una persona moviéndose. Se asume poca aceleración. Max altitud: 9000 metros; Máxima velocidad: 30m/s; Máxima velocidad vertical: 20m/s.

**Automotive:** Configuración por defecto para ADR (Automotive Dead Reckoning). Usado para aplicaciones con una dinámica equivalente al transporte de pasajeros. Se asume una baja aceleración vertical. Máxima altitud: 6000 metros; Máxima velocidad: 84 m/s; Máxima velocidad vertical: 15 m/s.

**At sea:** Recomendado para aplicaciones en el mar, Con velocidad vertical 0. Máxima altitud: 500 metros; Máxima velocidad 25 m/s, Máxima velocidad vertical: 5m/s.

**Airborne<1g:** Usado para aplicaciones con un alto rango dinámico y aceleración vertical como un coche de pasajeros. No se soportan los 2D Position Fix, sólo 3D. Máxima altitud: 50000 metros; Máxima velocidad: 100 m/s; Máxima velocidad vertical: 100 m/s.

**Airborne<2g:** Recomendado para entornos típicos aéreos. No se soportan los 2D Position Fix, sólo 3D. Máxima altitud: 50000 metros; Máxima velocidad: 250 m/s; Máxima velocidad vertical: 100 m/s.

**Airborne<4g:** Solo recomendado para entornos aéreos extremos. No se soportan los 2D Position Fix, solo 3D. Máxima altitud: 50000 metros; Máxima velocidad: 500 m/s; Máxima velocidad vertical: 100 m/s.

Este parámetro es el más importante del mensaje de configuración CFG-NAV5. Pero existen algunos parámetros más de filtrado que puede ser configurados con este mensaje. Para más información consultar el “**u-blox 6 Receiver Description Including Protocol Specification**” [18]

Target #	Electrical Interface
0	DDC (I2C Compatible)
1	UART 1
2	UART 2
3	USB
4	SPI
5	Reservado

Cuadro 3: Asignación de los “target numbers”

#### 6.4.5. Descripción de los protocolos de comunicación serie

La tecnología de posicionamiento de u-blox viene con una interfaz de comunicación serie altamente flexible. Soporta el protocolo NMEA y el protocolo propietario UBX, y es multipuerto y se pueden usar varios puertos a la vez. Cada protocolo (UBX, NMEA) puede ser asignado a varios puertos al mismo tiempo con ajustes individuales (como el baud rate, messages rate, ...) para cada puerto. También es posible asignar más de un protocolo a un mismo puerto, esto es particularmente útil para depuración o monitorización.

Para activar un mensaje en un puerto al protocolo UBX y/o NMEA se debe activar ese puerto usando el mensaje propietario **CFG-PRT**. Este mensaje también permite cambiar los ajustes específicos de puerto (baud rate, address, etc...). Ver la especificación del mensaje CFG-PRT para una descripción mas detallada [18].

Un **target** en el contexto de los sistemas de entrada/salida es un puerto de entrada/salida. La tabla 3 muestra los “target numbers” usados.

#### 6.4.6. Puertos UART

Como en este proyecto sólo se van a usar los puertos UART, sólo se va a hablar de estos, pero en “u-blox 6 Receiver Description Including Protocol Specification” [18] se podrá encontrar una especificación de los demás puertos de comunicación usados.

Uno o dos puertos UART pueden usarse para transmitir las medidas GPS, monitorizar información de estado y configurar el receptor.

Los puertos serie constan de una línea RX y TX. No se proporciona ninguna señal de Handshaking ni de control de flujo por hardware. Los baudrates pueden ser configurados individualmente para cada puerto serie. Sin embargo, no existe mecanismo para ajustar diferentes velocidades a la línea de transmisión y la de recepción de un mismo puerto. El Cuadro 4 muestra una relación de todos los baudrates que se pueden configurar.

Baud Rate	Data Bits	Paridad	Stop bits
4800	8	none	1
9600	8	none	1
19200	8	none	1
38400	8	none	1
57600	8	none	1
115200	8	none	1

Cuadro 4: Relación de baud rates configurables

Si la cantidad de datos configurados es demasiado grande para ciertos anchos de banda del puerto (como los mensajes de salida UBX en un puerto con un baud rate de 9600), el buffer se llenará. Una vez que el buffer esté lleno, los nuevos mensajes serán descartados, perdiéndose información.

Si el número de bytes a ser transmitidos y el baudrate está seleccionado de tal forma que la transmisión no pueda ser completada con éxito dentro de un periodo de tiempo, entonces el host no recibirá algunos mensajes. Para prevenir la pérdida de mensajes debido al timeout, el baudrate y la velocidad de comunicación o el número de mensajes activados debe ser seleccionado de tal forma que el número de bytes esperados pueda ser transmitido en menos de un segundo.

Si el Host no se comunica sobre SPI o DDC por más de aproximadamente 2 segundos, el dispositivo asume que el host no va a seguir usando esta interfaz y no serán enviados más paquetes por este puerto.

## 6.5. Configuración de las interrupciones

**INT0** Cuando este jumper está en ON activa las interrupciones externas del pin del módulo GPS.

Si no se va a usar hay que ponerlo en OFF en la posición que indica la imagen de la izquierda de la Figura 19.

**INT1** es el Jumper para la salida 1PPS. Cuando el módulo GPS ha captado el satélite GPS, este pin enviará un pulso por segundo. La configuración de esta interrupción se muestra en la imagen de la derecha de la Figura 19.

## 6.6. Protocolo NMEA

En esta sección se va a explicar un poco del código utilizado para trabajar con este módulo.

Como se ha dicho en las características de la placa, este módulo GPS soporta diferentes protocolos de comunicación. Por ser el más extendido vamos a usar el protocolo NMEA.

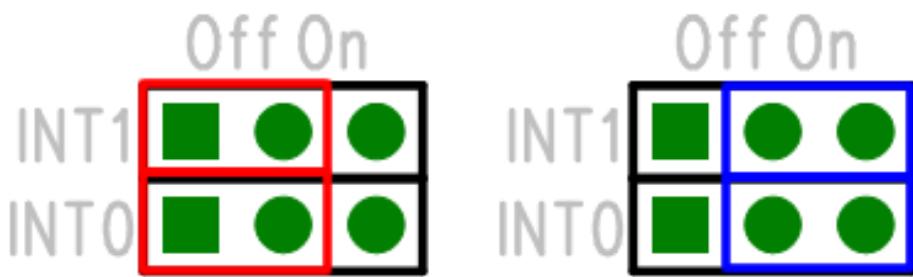


Figura 19: Configuración de las interrupciones de la Shield Ublox NEO-6M

El protocolo NMEA fue diseñado por la **National Marine Electronics Association (NMEA)** y define la interfaz entre varias piezas del equipamiento electrónico de la marina. El estándar permite a la marina enviar información a ordenadores y otro equipamiento. La comunicación de un receptor GPS está definida dentro de esta especificación. La mayoría de ordenadores que proporcionan posicionamiento en tiempo real entiende y espera que la información le llegue en formato NMEA. Esta información incluye la solución completa PVT (Posición, Velocidad y tiempo) computada por el receptor GPS.

La idea de NMEA es enviar una línea de datos llamada **sentencia** y que es independiente de otras. Existen sentencias estándar para cada **categoría de dispositivo** y existe también la capacidad de definir sentencias propietarias para uso de cada compañía individualmente. Todas las sentencias estándar tienen al menos dos letras de prefijo que definen el tipo de dispositivo a usar (por ejemplo GP para receptores GPS) las cuales están seguidas de una secuencia de tres letras que define el contenido de la sentencia. Además NMEA permite a los fabricantes Hardware definir sus propias sentencias para el propósito que deseen. Todas las sentencias propietarias comienzan con la letra P y están seguidas de 3 letras que identifican al fabricante que controla tal sentencia. Por ejemplo una sentencia del fabricante Garmin podría empezar con PGRM y una de Magellan podría empezar con PMGN.

**Cada sentencia empieza con un '\$' y acaba con un retorno de carro '\r\n' y no puede ser más larga de 80 caracteres de texto visible (sin contar los terminadores de línea).** Los datos contenidos dentro de esta simple línea son ítems de información separados por comas. La información en sí misma está representada como **texto ascii**. La información puede variar en función de la precisión contenida en cada mensaje. Por ejemplo el tiempo debe indicarse como parte decimal de un segundo o la posición como parte decimal de un grado con tres o 4 dígitos de precisión. Los programas que leen la información deben usar las comas como información para delimitar los campos. Se proporciona un checksum al final de cada sentencia la cual puede ser o no chequeada por la unidad que lee la información. El **Checksum** consiste en un '\*', y dos

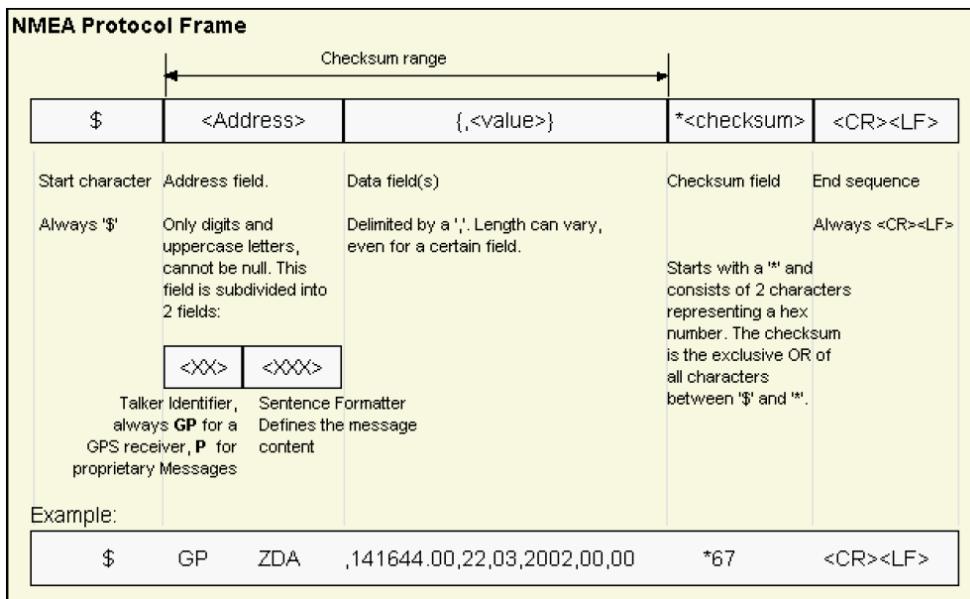


Figura 20: Sentencia NMEA

cifras hexadecimales que representan un EXOR de 8 bits de todos los caracteres de la sentencia comprendidos entre '\$' y '\*' pero sin incluirlos.

NMEA consiste en sentencias. La primera palabra de ellas, llamada **Data Type** define la interpretación del resto de la sentencia. Cada Data Type podría contener una única interpretación y está definida en el estándar NMEA. La sentencia GGA (descrita en la Figura 21) muestra un ejemplo que proporciona información esencial sobre el posicionamiento. Otras sentencias pueden repetir información pero proporcionarán información adicional. Sin embargo, el dispositivo o programa que lee la información puede observar para la información de la sentencia en la que está interesada y simplemente ignorar otras sentencias que no le interesan. En el estándar NMEA no existen comandos para indicar al receptor GPS que debe hacer algo diferente. En su lugar cada receptor envía toda la información en tramas esperando que la mayoría sea ignorada. Algunos receptores tienen comandos dentro de la unidad que puede dejar seleccionar un subconjunto de sentencias a mostrar o, en algunos casos, enviar sólo la sentencia de interés. No existe ningún modo de decirle al receptor que la trama está mal o que la vuelva a mandar. En su lugar, cada trama proporciona un checksum para corrección de errores. Si una trama se detecta que está mal, se desecha y se espera a la próxima.

Existen muchas sentencias en el estándar NMEA para todos los tipos de dispositivos que pueden ser usados en un entorno de la marina. Una breve descripción de algunas sentencias se muestra en el Cuadro 5. Para una descripción más completa se puede consultar el **u-blox 6 Receiver Description Including Protocol Specification** incluido en la bibliografía. [18]

```

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:
    GGA          Global Positioning System Fix Data
    123519       Fix taken at 12:35:19 UTC
    4807.038,N   Latitude 48 deg 07.038' N
    01131.000,E   Longitude 11 deg 31.000' E
    1             Fix quality: 0 = invalid
                  1 = GPS fix (SPS)
                  2 = DGPS fix
                  3 = PPS fix
                  4 = Real Time Kinematic
                  5 = Float RTK
                  6 = estimated (dead reckoning) (2.3 feature)
                  7 = Manual input mode
                  8 = Simulation mode
    08            Number of satellites being tracked
    0.9           Horizontal dilution of position
    545.4,M       Altitude, Meters, above mean sea level
    46.9,M        Height of geoid (mean sea level) above WGS84
                  ellipsoid
(empty field) time in seconds since last DGPS update
(empty field) DGPS station ID number
*47            the checksum data, always begins with *

```

Figura 21: Sentencia GGA

Sentencia NMEA	Descripción
DTM	Datum Reference
GBS	GNSS Satellite Fault Detection
GGA	Global positioning system fix data
GLL	Latitude and longitude, with time of position fix and status
GPQ	Poll message
GRS	GNSS Range Residuals
GSA	GNSS DOP and Active Satellites
GST	GNSS Pseudo Range Error Statistics
GSV	GNSS Satellites in View
RMC	Recommended Minimum data
THS	True Heading and Status
TXT	Text Transmission
VTG	Course over ground and Ground speed
ZDA	Time and Date

Cuadro 5: Algunas sentencias NMEA

## 6.7. Formato de la Latitud y la Longitud

De acuerdo con el estándar NMEA, las latitudes y longitudes de salida son expresadas en formato de Grados, Minutos y Fracciones de minutos. Para convertir a grados y fracciones de grados, o grados, minutos, segundos y fracciones de segundos, los 'Minutos' y las 'Fracciones de minutos' necesitan ser convertidos. En otras palabras: Si el GPS nos da un valor de Latitud de 4717.112671 Norte y Longitud de 00833.9148443 Este, esto es:

Latitud 47 Grados, 17.112671 Minutos

Longitud 8 Grados, 33.914843 Minutos

O

Latitud 47 Grados, 17 Minutos, 6.76026 Segundos

Longitud 8 Grados, 33 Minutos, 54.89058 Segundos

O

Latitud 47.285211188 Grados

Longitud 8.56524738 Grados

## 6.8. Implementación

Como se ha explicado anteriormente en el capítulo del protocolo NMEA, el receptor GPS lo que hará es enviarnos por puerto serie una serie de tramas delimitadas por los caracteres '\$' y el retorno de carro. Estos delimitadores son los que nos ayudarán a capturar cada trama que llegue al microcontrolador. Una vez capturada la trama se identifica si es la trama de interés y se almacena en un buffer preparado para este propósito para posteriormente ser decodificada. Las demás tramas se desecharán.

Como se puede ver en el código adjunto en el "Apéndice E: Implementación" se han declarado dos funciones llamadas **getNMEA1** y **getNMEA2**. Estas funciones son las que se encargan de sondear el puerto serie para capturar las tramas de interés. "getNMEA1" tiene un comportamiento **bloqueante** y "getNMEA2" tiene un comportamiento **no bloqueante**. Esto es que cuando se llama a getNMEA1, el microcontrolador espera a que la trama que busca aparezca y no sale de la función hasta que la encuentra y la decodifica. Sin embargo, getNMEA2 captura un byte del buffer de entrada del puerto serie en cada ciclo del loop principal, dando así opción a realizar más tareas durante una pasada del loop principal. También se puede ver que se ha declarado una función **descifrarTrama** que es llamada dentro de las funciones getNMEA y que se encargará de descifrar el contenido de la trama capturada.

En cuanto a la trama que nos interesa, capturaremos la sentencia **GGA** ya que es la que da información completa sobre posicionamiento tridimensional además de la hora UTC.

Como se puede ver en la implementación de getNMEA1 y getNMEA2, la diferencia entre ellas

es que en getNMEA1 no se sale del bucle while hasta que la trama está capturada por completo, mientras que en getNMEA2 cambia el while por un if que simplemente chequea si hay información disponible en el buffer del puerto serie y lo almacena en la cadena de caracteres NMEA. La ventaja de usar getNMEA2 frente a getNMEA1 es que permite al microcontrolador realizar más tareas durante cada ciclo del loop principal mientras que getNMEA1 nos da la certeza de que no se perderá información por estar el microcontrolador trabajando en otra tarea. En nuestro caso, ya que no estamos trabajando con muchas tareas hemos usado getNMEA 2

Como se puede ver en la implementación de la función **descifrarTrama**, lo más destacable es el scanf que hace a la cadena de caracteres que contiene la trama para extraer la información proporcionada por el receptor GPS. Una vez obtenida la información de la trama se preparan dos tramas mediante el comando sprintf: una que será escrita en un archivo en la tarjeta SD de la placa y otra trama adaptada al formato de un enlace web a google maps que será enviada vía SMS para poder localizar el dispositivo.

## 6.9. Tarjeta MicroSD

Uno de los puntos principales del proyecto es poder hacer un estudio de la trayectoria recorrida de la barquilla durante el vuelo. Uno de los problemas que se nos presentan es que con la tecnología GSM no podemos hacer un seguimiento en tiempo real de la barquilla ya que la cobertura GSM empieza a perderse a partir de los 1000 metros de altura. Para solucionar este problema en cierto modo, se propone registrar los datos de la posición de la barquilla en una tarjeta SD. De esta manera, aunque no podamos saber la posición sobre el nivel del suelo de la barquilla en todo momento, la barquilla tendrá almacenados esos datos cuando la recojamos.

En un primer momento se diseñó un slot microSD en la placa controladora para ser usada con MPLAB Harmony. Pero debido a los problemas que éste causa con nuestro microcontrolador y del cambio propuesto a la plataforma MPIDE se ha tenido que hacer uso del Slot microSD que viene con la placa GPS ya que este entorno sólo nos deja usar la tarjeta SD con los pines 50, 51, 52, 53. Por supuesto las dos ranuras están funcionales por si se desea volver a usar MPLAB Harmony en algún momento.

Como es bien conocido, **Secure Digital (SD)** es un formato de tarjeta de memoria para dispositivos portátiles tales como cámaras fotográficas digitales, computadoras portátiles e incluso videoconsolas. Fue desarrollado por SanDisk, Panasonic y Toshiba e introducido como una mejora evolutiva de las tarjetas MMC.

El formato Secure Digital incluye cuatro versiones de tarjetas, disponibles en tres tamaños. Las cuatro familias son la original “Standard Capacity” (SDSC), “High-Capacity” (SDHC), “Extended-Capacity” (SDXC) y SDIO (Secure Digital Input/Output). Los tres tamaños son el SD estándar

original, el Mini SD, y el Micro SD. Este último es muy utilizado en la actualidad en tabletas y smartphones. Por medio de adaptadores eléctricamente pasivos es posible utilizar tarjetas en ranuras más grandes.

Todas las tarjetas de memoria SD y SDIO necesitan soportar el antiguo modo SPI/MMC que soporta la interfaz serie de cuatro cables ligeramente más lenta (reloj, entrada serial, salida serial y selección de chip), pero que es compatible con los puertos SPI de muchos microcontroladores. El modo MMC no proporciona acceso a las características propietarias de cifrado de las tarjetas SD y la documentación libre de SD no describe dichas características. La información del cifrado es utilizada primordialmente por los productores de medios y no es muy utilizada por los consumidores quienes típicamente utilizan tarjetas SD para almacenar datos no protegidos.

Existen 3 modos de transferencia soportados por SD:

- Modo SPI: entrada y salida serie separados.
- Modo un-bit SD: separa comandos, canales de datos y un formato propietario de transferencia.
- Modo cuatro-bit SD: utiliza terminales extra más algunos terminales reasignados para soportar transferencias paralelas de cuatro bits.

En la **Figura 22** se puede ver el pinout de los distintos modos disponibles de comunicación disponibles en cada tarjeta SD. Para este proyecto vamos a usar el Modo SPI de 4 pines ya que es el único que soporta la librería que viene con MPIDE. En el **Cuadro 1** se puede ver la conexión de los distintos pines a la placa.

En la implementación que se encuentra en el Apéndice E encontraremos algunas de las funciones necesarias para manejar la SD. Para poder usar estas funciones hay que incluir el archivo “SD.h”. Además se han incluido varias librerías estándar de C para poder manejar mejor cadenas de caracteres. Para poder trabajar con archivos tenemos que crear un objeto File al que hemos llamado “myFile”. A través de este objeto podemos transferir información a un archivo alojado en la SD así como seleccionar los modos de escritura. La configuración del pin Chip Select de la tarjeta SD es el único parámetro que se puede cambiar en cuanto a selección de pines de la SD, los demás han de ser los pines 50, 51 y 52. Para comprobar si está disponible la tarjeta SD se hace un chequeo de la misma . Para ello se recurre a dos métodos de la clase SD que son **begin** y **exists**. El método **begin()** inicializa la librería SD y la tarjeta. Como parámetro puede recibir el número del pin asignado a ChipSelect de la SD o nada en caso de que éste se encuentre conectado al pin por defecto (pin 53). Como valor de retorno devuelve un TRUE si todo ha ido bien o un FALSE en caso contrario. El método **exists(“archivo”)** chequea si existe el archivo que se le ha pasado como parámetro. En caso afirmativo devuelve un TRUE o un FALSE en caso contrario.

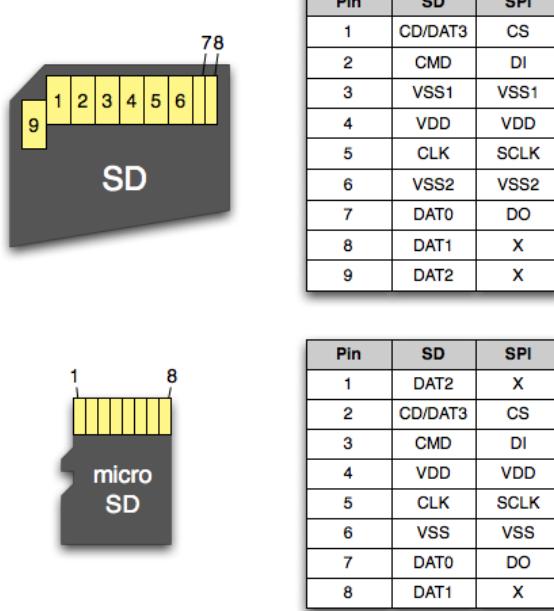


Figura 22: Diagrama de pines SD

La función que se encarga de escribir en la SD es **writeFileSD(\*char mensaje)**. Pasándole una cadena de caracteres como argumento, ésta se encarga de escribirla en la SD. Se puede observar que esta función se utiliza dentro de la función descifrarTrama() dentro de getNMEA2(). Como se puede ver en la implementación de esta función, antes de escribir en la tarjeta hay que hacer un **open("archivo",modo)** para indicarle al programa el archivo al que se desea escribir y el modo. En el caso de que el archivo no exista, lo crea. El método **write(mensaje)** escribe la cadena de caracteres enviada como argumento a la tarjeta SD. Finalmente se hace un **Close()** para finalizar la escritura del archivo.

Como se puede ver, la librería que trae consigo MPIDE es muy sencilla de usar y es mas que suficiente para nuestro propósito.

## 7. Módulo de Comunicación GSM

### 7.1. Introducción

Para enviar la información sobre posicionamiento de forma telemática vamos a hacer uso de un módulo de comunicación GSM. Este módulo nos permitirá enviar mensajes de texto a un móvil cada cierto tiempo con un enlace a Google Maps que nos indicará la posición de la barquilla. El problema que presenta esta tecnología es que a partir de cierta altura deja de ser funcional por lo que su uso está limitado a zonas de baja altitud.

El módulo que vamos a usar es un **SIM900** de la compañía **SIMCOM**. Este dispositivo se ha elegido por estar muy extendido entre la comunidad de desarrolladores de Arduino, por lo que existe gran cantidad de información disponible en Internet. Como ya se dijo en el capítulo 2.3, el SIM900 tiene una buena relación calidad-precio ya que tiene todas las características que se le pueden pedir a un módulo de este tipo, como envío de mensajes de texto y una comunicación a velocidad 3G, por un precio que ronda los 9 euros. Es por ésto que su uso en nuestro proyecto queda más que justificado. Las Figuras 23 y 24 muestran un diagrama funcional del módulo y un esquema general de la placa GSM.

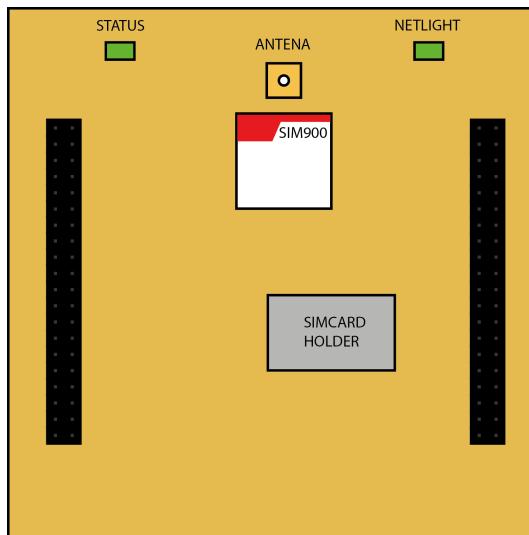


Figura 23: Diagrama de la placa GSM

El funcionamiento de este dispositivo está basado en el uso de **comandos AT** a modo de consola y su protocolo de comunicación con otros dispositivos está implementado principalmente por **Puerto Serie**. Esto tiene una gran ventaja y es que no solo podemos probar el dispositivo sin necesidad de implementar ningún código para probar su funcionamiento, sino que con solo conectar la tarjeta a la alimentación y el convertidor usb-serial al puerto serie de ésta, ya podemos empezar

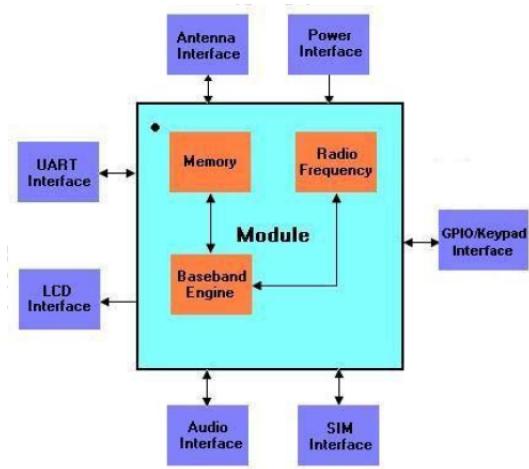


Figura 24: Diagrama Funcional del SIM900

a trabajar con él a través de algún terminal de puerto serie como puede ser “RealTerm” en nuestro caso [14].

## 7.2. Características del módulo GSM

- Motor de 4 Bandas GSM/GPRS
  - GSM: 850 MHz
  - EGSM: 900MHz
  - DSC: 1800 MHz
  - PCS: 1900 MHz
- Dimensiones de empaquetado de 24x24x3 mm.
- 68 pines
  - Keypad y SPI Display Interface
  - Serial Port y Debug Port para el desarrollo de aplicaciones.
  - Un canal de audio con Micrófono y Altavoz.
  - Entradas/Salidas de propósito general.
- Modo de Ahorro de Energía: 1.5 mA en modo sleep.
- Protocolos:
  - TCP/IP
  - Extended TCP/IP: Los comandos AT están desarrollados para que los usuarios usen fácilmente TCP/IP.
- Voltaje de alimentación: 3.4 V a 4.5 V.

## 7.3. Consideraciones de diseño de la alimentación

Este módulo GSM puede llegar a tener picos de consumo de hasta 2 Amperios, por lo que es conveniente diseñar la Fuente de Alimentación de tal modo que sea capaz de proporcionar esta corriente cuando se requiera. Para estabilizar la tensión de alimentación a un valor fijo se recomienda usar dos capacidades de bypass de 100nF y 100 uF con una resistencia interna de pérdidas (ESR) lo más baja posible por lo que se recomienda usar capacitores cerámicos o de tantalio.

El SIM900 tiene 3 pines de alimentación principales **VBAT** que, como se ha dicho en el Apartado 7.2, su tensión debe estar comprendida entre 3.4 V y 4.5 V. Además se dispone de un pin **VRTC** que puede usarse para alimentar el RTC interno (Real Time Clock). Esta alimentación nos da la posibilidad de no perder configuraciones establecidas tras una desconexión completa de la placa. Si se desea usar esta funcionalidad deberemos conectar a este pin una pila de 3V. En caso

contrario debe dejarse sin conectar.

Durante el funcionamiento normal del dispositivo, tenemos disponible el Comando AT **AT+CBC** que devuelve información sobre el nivel del voltaje, por lo que podremos monitorizar su estado en cualquier momento. El voltaje que se muestra es el valor sobre el último periodo de medida antes de que el comando AT+CBC sea ejecutado.

#### 7.4. Escenarios de encendido y apagado

En general, hay que asegurarse de no encender el SIM900 hasta que no esté más allá de los límites seguros de voltaje y temperatura. El módulo puede llegar a apagarse inmediatamente si observa que no existen condiciones adecuadas de alimentación.

Para encender y apagar el dispositivo existen dos formas distintas de hacerlo:

- Mediante un **pulso negativo** en el pin **PWRKEY**. La duración de este pulso debe ser mayor que 1 segundo. Pasado este pulso, el dispositivo tardará alrededor de 1.7 segundos en encenderse.
- Mediante un **pulso positivo** conjuntamente en los pines **PWRKEY** y **PWRKEY\_OUT**. Al igual que en el otro caso, este pulso debe ser mayor que un segundo, y tras 1.7 segundos, el dispositivo estará encendido.

Tras este proceso, el led conectado al pin status del módulo se mantendrá encendido. Cuando el led conectado al pin NETLIGH parpadee lentamente, el módulo estará listo para enviar información. Estos dos modos sirven tanto para encender como para apagar el dispositivo. En las figuras 25 y 26 se ilustra este procedimiento. Tras realizar estos procesos, el módulo nos enviará el mensaje “**NORMAL POWER DOWN**”. Tras este mensaje el puerto de comunicación se desconectará y el dispositivo no podrá seguir recibiendo información. Lo único que sigue funcionando tras este comando es el RTC (Real Time Clock).

Para que se produzca un apagado del dispositivo existen tres escenarios posibles más aparte de los citados antes:

- Mediante el comando AT “**AT+CPOWD=1**”. Este método invita al módulo a desconectarse de la red y le permite entrar en un **estado seguro** y guardar datos antes de apagar el dispositivo completamente. Antes de completar el proceso, el dispositivo enviará el mensaje de comprobación “**NORMAL POWER DOWN**”. Tras este mensaje, los comandos AT no pueden ser ejecutados igual que en los casos anteriores.
- Caso de **sobrevoltaje** o **subvoltaje** detectado. El módulo está constantemente monitorizando el voltaje aplicado en el pin VBAT. Cuando se presente un voltaje menor a 3.5V el

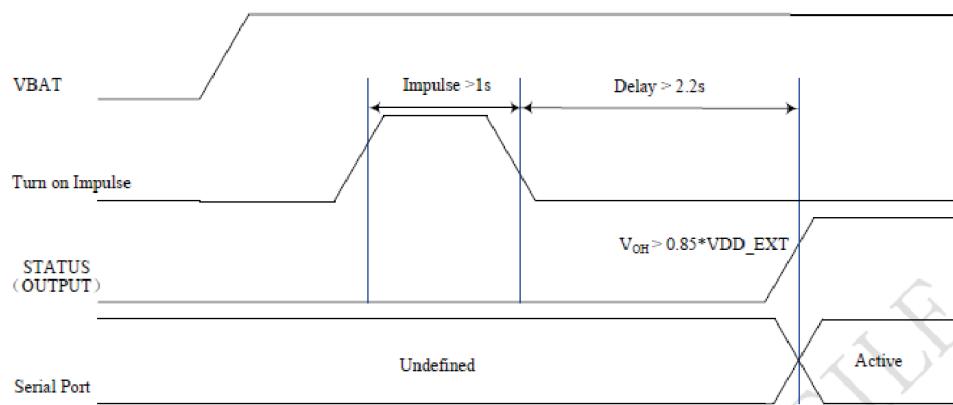
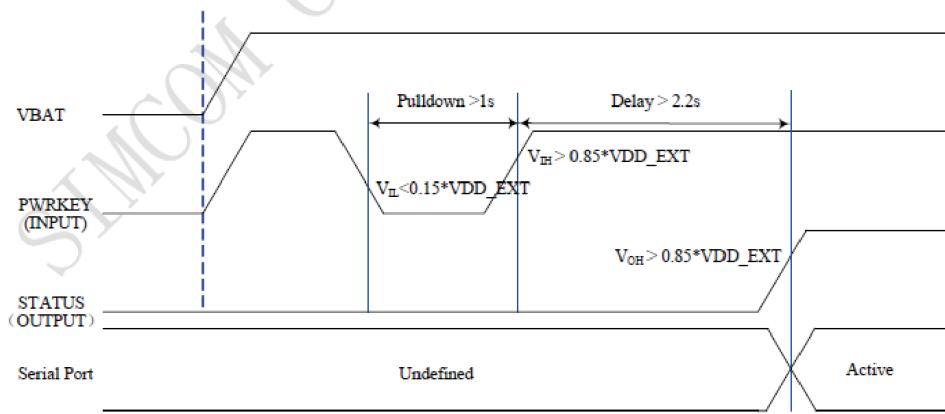


Figura 25: Encendido del dispositivo: (arriba) Modo PWRKEY; (abajo) Modo PWRKEY y PWRKEY\_OUT

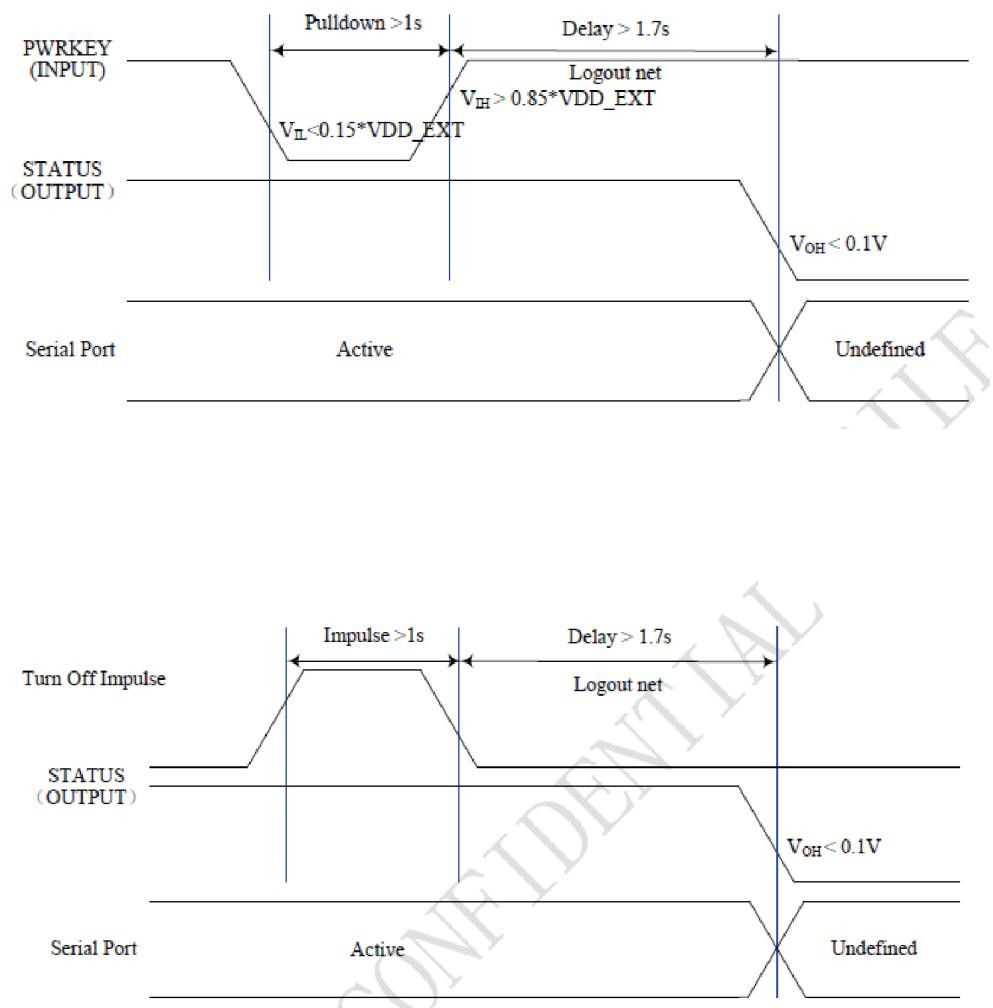


Figura 26: Apagado del dispositivo: (arriba) Modo PWRKEY; (abajo) Modo PWRKEY y PWRKEY\_OUT

módulo enviará el mensaje “**UNDER-VOLTAGE WARNING**” y cuando detecte un voltaje mayor a 4.5V enviará el mensaje “**OVER-VOLTAGE WARNING**”. Si el nivel de tensión llega a ser mayor que 4.6V o menor que 3.4V, el dispositivo se apagará automáticamente. En ambos casos aparecerán los mensajes “**OVER-VOLTAGE POWER DOWN**” o “**UNDER-VOLTAGE POWER DOWN**” respectivamente. Después de este momento el módulo dejará de ejecutar comandos AT.

- Caso de nivel de **sobretemperatura** o **subtemperatura** detectado. El módulo está siempre monitorizando la temperatura del módulo, si la temperatura de éste es mayor que 80°C se presentará el mensaje “**+CMTE:1**”. Si la temperatura cae por debajo de los -30°C, se mostrará el mensaje “**+CMTE:-1**”. Los valores de temperatura críticos son 85°C y -40°C. En el caso de que estos valores sean sobrepasados se mostraran los mensajes “**+CMTE:2**” y “**+CMTE:-2**” respectivamente. Después de este momento los Comandos AT no podrán ser ejecutados y el módulo entrará en modo Power Down. Lo único que seguirá funcionando es el RTC. Para **monitorizar la temperatura** del módulo puede usarse el Comando AT “**AT+CMTE**” cuando éste esté encendido.

## 7.5. Ahorro de energía

El módulo SIM900 soporta dos modos principales destinados al ahorro de energía. En nuestro proyecto, esta funcionalidad sería muy interesante cuando la barquilla se encuentre fuera de la zona de cobertura, ya que en este espacio no podremos transmitir por GSM:

### 7.5.1. Modo de funcionalidad mínima

Este modo reduce la funcionalidad del módulo al mínimo y, por tanto, minimiza el consumo de corriente a un nivel mínimo. Para activar el modo de mínima funcionalidad hay que usar el comando AT “**AT+FUN**”. Este comando proporciona 3 niveles de funcionalidad:

- 0: Funcionalidad mínima. Si el módulo ha sido configurado para esta funcionalidad “**AT+FUN=0**”, la función RF y la función de la SIM Card estarán bloqueadas. En este caso el puerto de comunicación serie estará activo todavía, pero los comandos relativos a RF y SIM no.
- 1: Alta funcionalidad (Por defecto)
- 4: Desactiva los circuitos emisores y receptores del teléfono. Cuando el módulo se encuentra en esta funcionalidad (“**AT+FUN=4**”), sólo el módulo de RF estará desbloqueado. El puerto de comunicación serie sigue activo.

### 7.5.2. Modo Sleep (Slow Clock Mode)

Cuando el SIM900 entra en modo sleep puede recibir SMS de la red, pero el puerto serie no será accesible. Podemos controlar la entrada y salida de este modo mediante el pin **DTR**. Cuando

DTR está en alto y no existe una interrupción hardware (tal como una interrupción en un pin GPIO o datos en el puerto serie), el módulo entrará en modo sleep. Cuando el módulo está en modo sleep, los siguientes métodos pueden sacarlo de este estado:

- Activar el pin DTR. Si DTR se lleva a un nivel bajo, esta señal sacará al módulo del modo sleep. El puerto serie estará activo después de que DTR haya cambiado a nivel bajo durante alrededor de 50 ms.
- Recibir una llamada de voz
- Recibir un SMS.

## 7.6. Interfaz Serie

En este proyecto hemos elegido trabajar con el puerto serie del SIM900 ya que está muy bien implementado y tiene líneas de estado y control que nos pueden ayudar a controlar mejor la comunicación con el dispositivo. La sencillez de uso con el puerto serie que tiene MPIDE también ha influido a la hora de elegir este puerto de comunicación para trabajar. El puerto serie del SIM900 posee las siguientes características:

- Dispositivo módem
- Contiene líneas de transmisión TXD y RXD, líneas de estado RTS y CTS y líneas de control DTR, DCD, DSR, y RI
- El Puerto Serie puede usarse para CSD FAX, servicio GPRS y enviar comandos AT para controlar el módulo. También puede usarse para funciones de Multiplexación. Solo soporta el modo básico de multiplexación.
- Soporta los siguientes ratios de comunicación: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 (defecto) bps.
- Posee una función de autobauding que soporta los siguientes ratios: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 (defecto) bps.

La función de autobauding está activada por defecto. Esta función detecta automáticamente el baud rate configurado por el host. Con el autobauding activado el puerto serie debe de operar con 8 bits de datos, sin paridad y un bit de stop (Ajustes de fábrica).

### 7.6.1. Interfaz de la Tarjeta SIM

El SIM900 ofrece la posibilidad de usar los comandos AT para obtener información de la tarjeta SIM.

La interfaz SIM soporta las siguientes funcionalidades:

- Funcionalidad de la especificación GSM Phase 1
- Funcionalidad de la especificación GSM Phase 2+ para fast 64kbps SIM (Intencionado para uso con una SIM application tool-kit)
- Soportadas 1.8V y 3.0V SIM Cards
- La interfaz sim está alimentada desde un regulador interno del módulo y tiene un voltaje normal de 3V.
- El comando AT “**AT+CSMT**” puede usarse para cambiar la configuración de la SIMCARD

## 7.7. Led NETLIGH

El led Netlight mostrado en el esquema de la Figura 23 puede usarse para conocer el estado de conexión del dispositivo. Ésto nos será muy útil en tierra antes del lanzamiento ya que de esta manera podemos conocer el estado de la comunicación en todo momento. Los modos de comportamiento que puede tener este led son los siguientes:

**off:** Sim900 no está encendido

**64msON/800msOFF:** Sim900 no encuentra la Red

**64msON/3segOFF:** Sim900 conectado a la Red

**64msON/300msOFF:** Comunicación GPRS

## 7.8. Antena

El SIM900 tiene una entrada para la conexión de una antena RF(Radio Frecuencia). Un conector de antena debe estar colocado en la placa y conectado al módulo mediante una línea microstrip o cualquier otro tipo de traza RF cuya impedancia deber estar controlada a 50 Ohmios. Para evitar problemas de este tipo se ha situado el conector de la antena muy cerca del módulo de manera que las pérdidas por problemas de impedancia controlada sean mínimas.

Para más información sobre consideraciones de diseño y comandos AT puede consultarse la bibliografía de esta memoria [20, 21].

## 7.9. Implementación

En el **Apéndice E: Implementación** puede verse el código completo del proyecto. A continuación se explicarán algunos de los comandos utilizados y algunas de las funciones usadas para un mejor manejo de estos comandos.

Para empezar decir que en el setup del programa se ha configurado el puerto “Serial” como puerto de comunicación del módulo GSM y el puerto “Serial2” ha sido asignado como monitor

serial para llevar una depuración del programa. A ambos puertos se le ha ajustado el baud rate a 9600bps. Como se ha explicado en el apartado 7.6, el módulo GSM tiene una función de autobauding, por lo que no es necesario configurar el baudrate del módulo GSM.

Para poder observar qué es lo que está ocurriendo en el puerto “Serial” a través del puerto “Serial2” se ha creado la función **check()**. Esta función hace de puente entre ambos puertos de manera que lo que escriba en un puerto redirigirá hacia el otro.

Dentro del bucle principal podemos encontrar tres condicionales if relacionados con el uso de módulo GSM:

- El primer condicional implementa un servicio de envío de mensajes cada cierto tiempo. Este intervalo de tiempo está controlado por la variable “tiempo”. Como se ha explicado en el capítulo del módulo GPS, cada segundo aproximadamente el receptor GPS envía una serie de tramas a través del puerto serie. La variable “tiempo” aumenta de valor cada vez que se envía una trama por lo que se podría decir que cada segundo aumenta en una unidad el contador “tiempo”. Cuando el contador llega a cierto valor establecido por el condicional if, se accede a una función que envía un mensaje de texto con un enlace a google maps con la posición obtenida del receptor GPS una vez ya decodificada. La función encargada de enviar el mensaje se llama **enviarMensaje(char \*cadena)** a la que se le pasa como argumento una cadena de caracteres que es la cadena a enviar. En la implementación de esta función se puede modificar el número de teléfono al que se desea enviar el mensaje dentro de la llamada al comando **AT+CMGS** que es el encargado de enviar los mensajes de texto.
- El segundo condicional está relacionado con el encendido y apagado del módulo GSM. Cuando se pulsa el botón 1 de la placa de control, éste pone a bajo el pin PWRKEY del módulo y cuando se deja de pulsar se pone en alto. Como se explicó en el apartado 7.4 habrá que dejar el botón pulsado más de un segundo para que el módulo lleve a cabo la orden de encenderse o apagarse.
- Por último, el tercer condicional está relacionado con la activación del servicio de envío de SMS. Cuando pulsamos el botón 2 de la placa de control, se desactiva el servicio de envío de mensajes activando un led de la placa de control como testigo de que el servicio está desactivado.

## 8. Módulo de alimentación

Para poder alimentar todos los módulos de este proyecto y las posibles ampliaciones hay que diseñar una fuente de alimentación que sea capaz de proporcionar los voltajes y las corrientes adecuadas. Esta fuente de alimentación se ha diseñado con dos modelos distintos de reguladores lineales: El **LM7805** y el **LM317**. La razón principal de llevar reguladores lineales y no convertidores DC/DC es el ruido que estos generan, ya que en su diseño está implícito el uso de conmutadores que generan mucho ruido y pueden contaminar los demás módulos del sistema así como por ejemplo medidas analógicas, niveles de tensión de alimentación, etc... . Este ruido es muy difícil de controlar y puede meterse por los planos de masa y las pistas de alimentación afectando los niveles de voltaje que alimentan los integrados y provocando problemas de reseteos debido a picos de baja tensión. Además, interesa que la tarjeta de alimentación pueda proporcionar la mayor cantidad de corriente posible ya que, aparte de sus futuras aplicaciones que pueden requerir más potencia, componentes de esta misma placa como puede ser el módulo GSM requieren de una fuente capaz de proporcionar hasta 2 amperios de corriente. Ésto en una fuente conmutada se traduce en bobinas grandes, pesadas y caras. El incluir elementos pesados en el sistema es contraproducente ya que cada gramo que lleve la barquilla de más puede ser crucial en el viaje del globo. Por esta razón, las fuentes conmutadas aunque son más eficientes, necesitan de un estudio más profundo para eliminar efectos de ruido y se reservan para aplicaciones más concretas. Los reguladores lineales por tanto, aunque son menos eficientes, son más flexibles para nuestro proyecto y más fáciles de usar.

El LM7805 es un componente estándar muy conocido en el mundo de la electrónica que es capaz de proporcionar 5V de tensión y una corriente máxima de 1.5A. Aunque actualmente no se dispone de ningún dispositivo que opere a esta tensión, se ha dejado disponible por su posible uso en expansiones ya que existen muchos componentes en el mercado que funcionan a esta tensión.

El LM317 es un regulador lineal ajustable desde 1.2V hasta 37V y una corriente de 1.5A. Es un componente bastante estandarizado y se ha elegido especialmente porque pueden conectarse varios reguladores en paralelo para proporcionar más corriente. Estos reguladores se han usado para tener una salida de voltaje de 3.3V para alimentar la gran mayoría de los componentes de este proyecto y una salida de 4.2V a 3A para el módulo de GSM ya que lo requerían sus especificaciones, por lo que en este caso se ha usado una configuración de dos reguladores en paralelo. Notar que cada configuración del LM317 lleva una resistencia variable que puede usarse para ajustar la tensión de salida de la configuración.

Para más información puede consultarse el esquemático del **Apéndice C: Esquemáticos de la placa de alimentación** y los datasheets de los dos módulos indicados en la bibliografía[22, 23]



## 9. Validación y resultados

En este capítulo se va a explicar el procedimiento seguido para la validación de cada una de las placas del proyecto por separado y se expondrá mediante fotos y capturas los resultados obtenidos.

Contamos con una fuente de alimentación estabilizada a 14V que es capaz de proporcionar hasta 5A, más que suficiente para nuestro propósito. La figura 27 muestra dicha fuente.



Figura 27: Fuente de alimentación estabilizada

Para empezar conectamos nuestra placa de alimentación a la fuente (Figura 28). Como se explicó en el capítulo de la placa de alimentación, ésta estaba formada por reguladores lineales, por lo que se espera que a partir de una entrada de 14V nos proporcione salidas de 5, 4.2 y 3.3V. La figura 29 muestra el resultado al conectar el polímetro con los distintos pines de alimentación. Como se puede observar en las imágenes, se obtiene el resultado esperado.

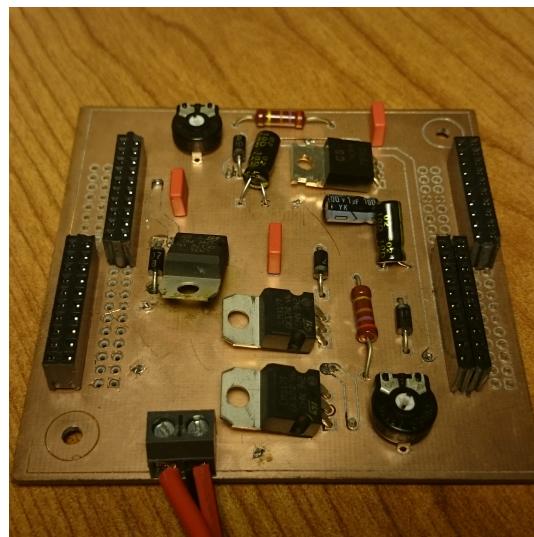


Figura 28: Placa de alimentación

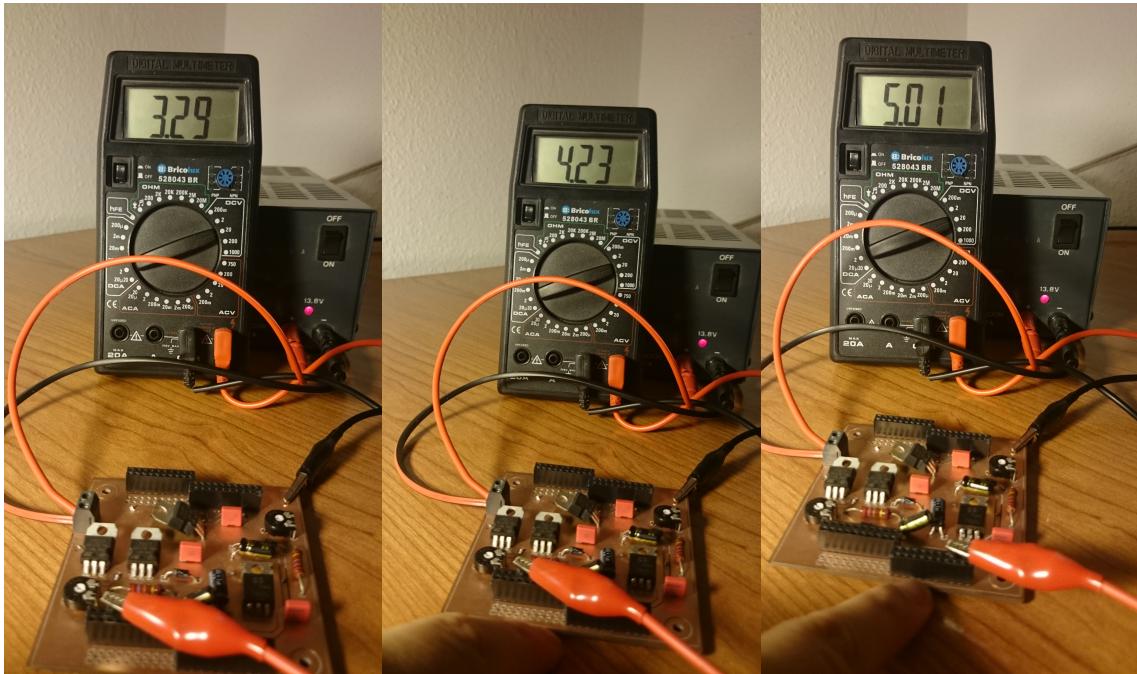


Figura 29: Resultados de los voltajes de salida

Una vez comprobado que funciona la placa de alimentación, estamos listos para probar el resto de las placas montadas sobre ésta. La primera placa que vamos a probar va a ser la placa GPS (Figura 30). Para ello vamos a hacer uso del cable TTL Serial-USB conectándolo directamente a la salida del puerto serie de la placa. Como se explicó en el capítulo de la placa de GPS, se espera

que el receptor envíe cada segundo una serie de tramas encabezadas por un signo '\$'. Para verlo usaremos el monitor serie de MPIDE. Como se puede observar en la figura 31 tenemos una serie de líneas que se repiten periódicamente. Éste era el resultado que esperábamos, entre el que se encuentra la trama GPGGA que es la que nos interesa para nuestro proyecto.

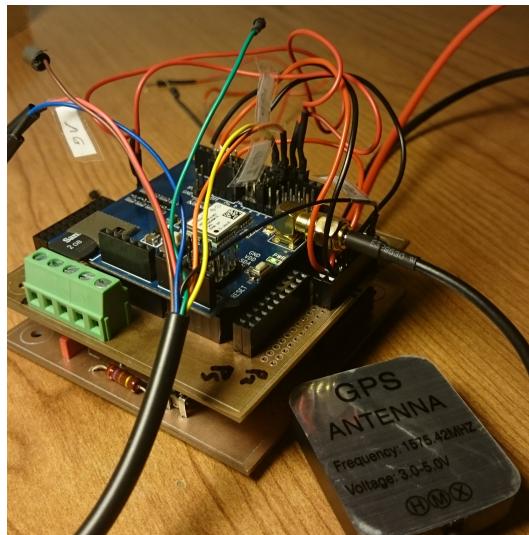


Figura 30: Placa GPS

```

$GPGSV, 2, 2, 06, 28, 16, 255, 43, 32, 45, 046, 30*76
$GPGLL, 3712.20817, N, 00336.73380, W, 220850.00, A, A*74
$GPRMC, 220851.00, A, 3712.20817, N, 00336.73376, W, 0.143,, 300615,,, A*62
$GPVTG,, T,, M, 0.143, N, 0.265, K, A*24
$GPGGA, 220851.00, 3712.20817, N, 00336.73376, W, 1, 05, 1.95, 762.1, M, 46.8, M,, *49
$GPGSA, A, 3, 28, 17, 01, 32, 09, , , , , , 4.68, 1.95, 4.26*00
$GPGSV, 2, 1, 06, 01, 66, 066, 28, 03, 74, 320, 27, 09, 15, 195, 43, 17, 34, 311, 36*73
$GPGSV, 2, 2, 06, 28, 16, 255, 43, 32, 45, 046, 29*7E
$GPGLL, 3712.20817, N, 00336.73376, W, 220851.00, A, A*7C
$GPRMC, 220852.00, A, 3712.20816, N, 00336.73371, W, 0.050,, 300615,,, A*64
$GPVTG,, T,, M, 0.050, N, 0.093, K, A*2C
$GPGGA, 220852.00, 3712.20816, N, 00336.73371, W, 1, 05, 1.95, 761.9, M, 46.8, M,, *47
$GPGSA, A, 3, 28, 17, 01, 32, 09, , , , , , 4.68, 1.95, 4.26*00
$GPGSV, 2, 1, 06, 01, 66, 066, 28, 03, 74, 320, 26, 09, 15, 195, 43, 17, 34, 311, 37*73
$GPGSV, 2, 2, 06, 28, 16, 255, 43, 32, 45, 046, 29*7E
$GPGLL, 3712.20816, N, 00336.73371, W, 220852.00, A, A*79
$GPRMC, 220853.00, A, 3712.20817, N, 00336.73366, W, 0.070,, 300615,,, A*60
 Autoscroll
    
```

Figura 31: Tramas NMEA recibidas

Una vez que hemos comprobado que la placa de GPS funciona correctamente, vamos a compro-

bar la placa de GSM (Figura 32). Antes de probar ningún comando AT, conectamos la placa GSM y comprobamos que se encienden los leds “Status” y “Netlight”. Como se puede ver en la figura 33, el led de estado se enciende y, aunque no se pueda apreciar en la imagen, el led netlight parpadeaba muy lentamente, lo que significa que tiene cobertura de red. Una vez comprobado que enciende correctamente y se conecta a la red vamos a probar a mandar un SMS mediante comandos AT. En este caso abrimos el programa “Realterm”. Este programa es otro monitor serie parecido al monitor serie de MPIDE o Arduino pero tiene más opciones. El caso es que el monitor serie de MPIDE o Arduino no detecta la combinación de teclado “CTRL+Z” (carácter ascii 0x1A ) necesaria que hay que introducir al terminar de escribir el texto que se va a enviar por SMS. La figura 34 muestra el mensaje escrito a través del comando AT “AT+CMGS” con la confirmación “OK” de que se ha enviado correctamente y el correspondiente mensaje que ha llegado al móvil de destino.

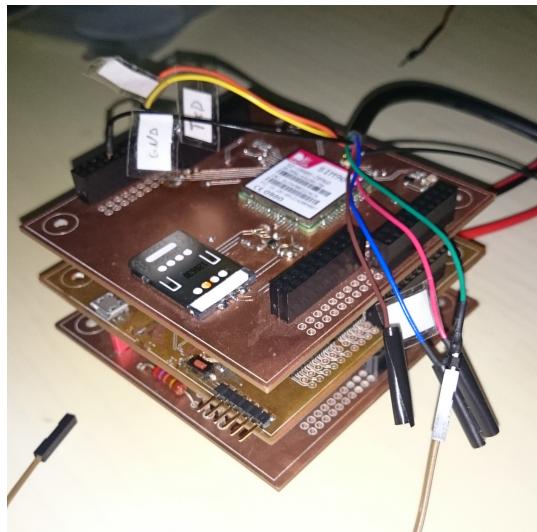


Figura 32: Placa GSM



Figura 33: Comprobación de leds de la placa GSM

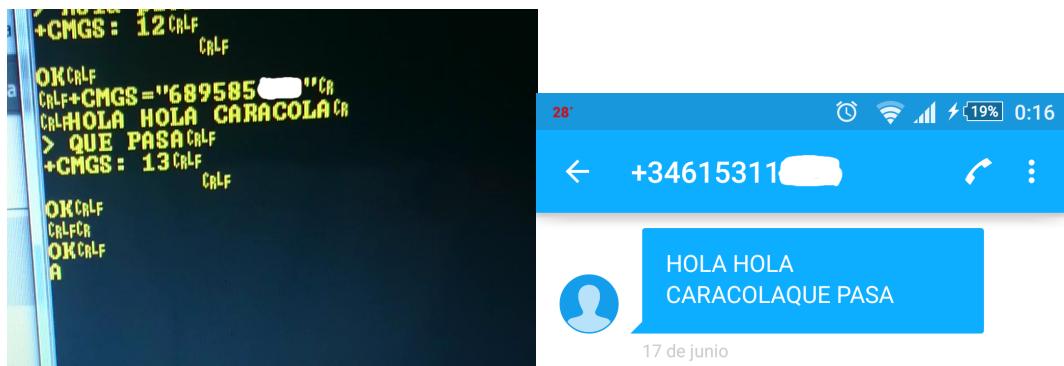


Figura 34: Envío de mensaje y comprobación

Finalmente, ya que sabemos que las placas de GPS y GSM funcionan por separado, se han montado todas las placas conjuntamente y se ha comprobado que funciona el conjunto.

Tras montar todas las placas y conectarlas a la corriente, el programa arrancará los módulos en orden correcto. Primero inicia los puertos de comunicación serie y luego comprobará la tarjeta SD. Se comprueba que se ha encendido el led de Status de la placa de GSM, que el de Netlight parpadea lentamente y que no se ha encendido el led rojo de la placa de desarrollo, lo que significaría que no se ha conseguido leer la tarjeta SD. Si esto ocurriera habría que hacer un reset a la placa del microcontrolador y, si es necesario, encender la placa de GSM con el botón central de la placa de control, hasta llegar al estado deseado. Una vez que estamos en este estado el servicio de envío de mensajes está desactivado, el GPS estará buscando satélites y descodificando la trama GGA para dividirla en las diferentes variables que la componen y transformarla en el enlace de google maps. Tras esperar un rato a que el GPS esté recibiendo correctamente (alrededor de un par de minutos) pulsamos el botón de la derecha de la placa de control que activa el envío de mensajes. Tras

pulsar el botón se activará el led verde (Figura 35) lo cual indica que cada 60 segundos (según lo programado) nos llegará un mensaje al móvil con un enlace a google maps que nos dará la posición del dispositivo (Figura 36). La razón de implementar el servicio de envío de mensajes se debe a que si empezáramos a enviar los mensajes desde que arranca el dispositivo, nos llegarían enlaces no válidos ya que el GPS tarda un rato en encontrar los satélites y empezar a recibir información.

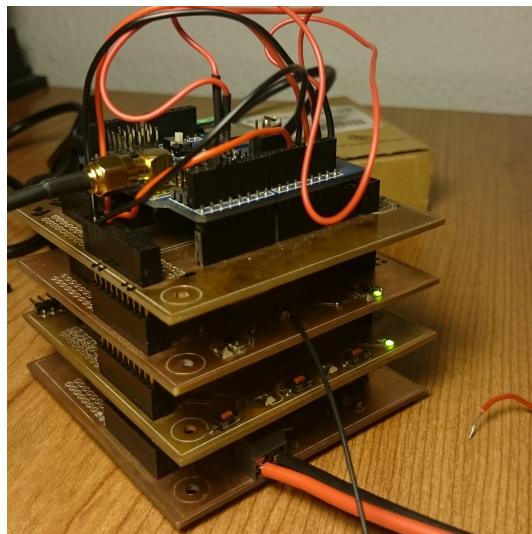


Figura 35: Montaje final y envío de mensajes activado

Tras haber comprobado el envío de mensajes comprobamos que se han ido escribiendo en la tarjeta SD (según lo implementado en el programa) una línea con los parámetros de la sentencia GPGGA decodificada y el enlace de google maps con la posición decodificada. Para ello insertamos la tarjeta en el PC y comprobamos el archivo LOG.txt que se ha creado (Figura 37). Como se puede ver en la imagen, el programa ha ido volcando la información en la tarjeta SD de forma correcta.

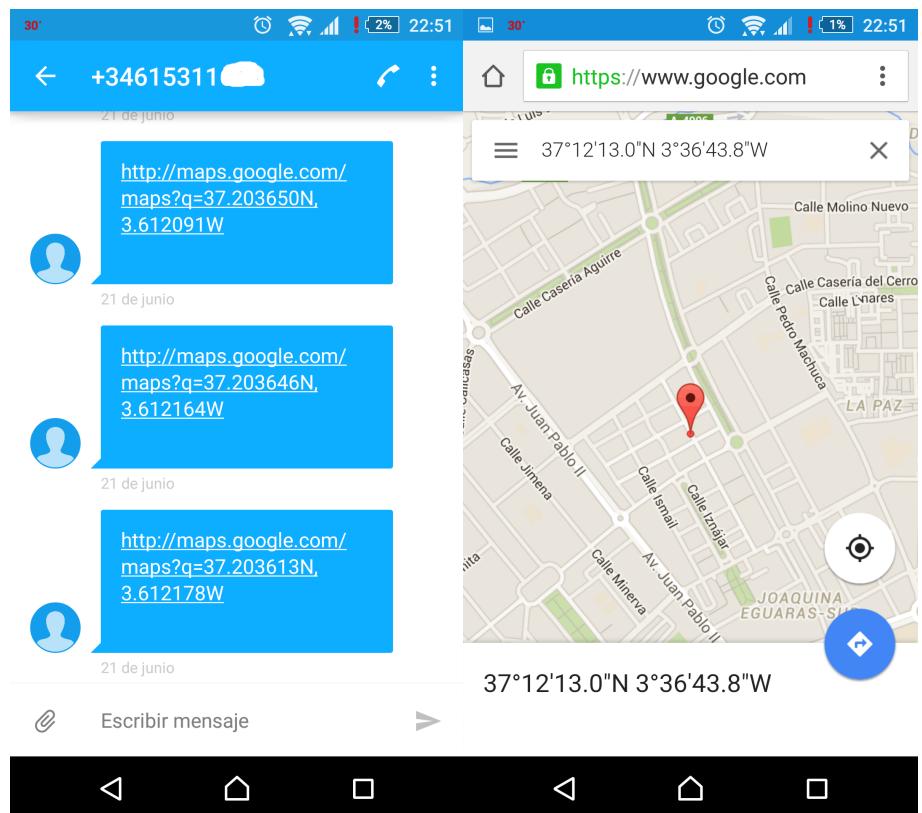


Figura 36: Comprobación del envío de mensajes

```

Board_Defs.h LOG.TXT
138
139 164336.000000,3712.227051,N,336.720795,W,2,4,3.480000,680.799988,M
140 http://maps.google.com/maps?q=37.203784N,3.612013W
141
142 164337.000000,3712.226562,N,336.720856,W,2,4,3.480000,682.900024,M
143 http://maps.google.com/maps?q=37.203776N,3.612014W
144
145 164338.000000,3712.226318,N,336.721100,W,2,4,3.480000,684.599976,M
146 http://maps.google.com/maps?q=37.203772N,3.612016W
147
148 164339.000000,3712.226074,N,336.721191,W,2,5,3.480000,685.500000,M
149 http://maps.google.com/maps?q=37.203768N,3.612020W
150
151 164340.000000,3712.226318,N,336.721680,W,2,4,3.480000,686.299988,M
152 http://maps.google.com/maps?q=37.203772N,3.612028W
153
154 164341.000000,3712.226318,N,336.721985,W,2,4,3.480000,686.700012,M
155 http://maps.google.com/maps?q=37.203772N,3.612033W
156
157 164342.000000,3712.226318,N,336.722382,W,2,4,3.490000,686.799988,M
158 http://maps.google.com/maps?q=37.203772N,3.612040W
159

```

Figura 37: Información en tarjeta SD

Por último, una vez que se ha montado el dispositivo entero y se ha comprobado su funcionamiento, se ha medido su consumo real en funcionamiento a máximo rendimiento (todas las funciones activadas: envío de mensajes almacenamiento en SD, búsqueda constante de satélites,...). Como se puede ver en la figura 38, el dispositivo consume alrededor de 230 mA. Este resultado es bastante bueno ya que con una batería lipo pequeña de unos 2200mAh como por ejemplo la “Turnigy 2200mAh 3S 20C Lipo Pack”[15] tendríamos una autonomía de aproximadamente **9.5 horas**. Por supuesto, si jugamos con los modos de ahorro de energía podremos aumentar esta autonomía bastante más.

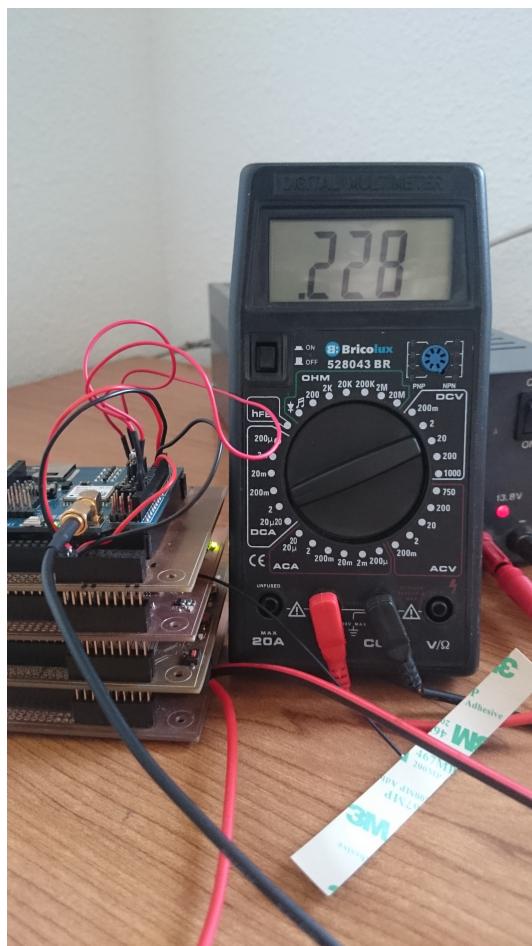


Figura 38: Medida del consumo del dispositivo a máximo rendimiento

## 10. Fabricación y estimación económica

En este capítulo se va a explicar cómo se realizó la fabricación y el montaje de la PCB cuyo diseño se ha detallado en cada capítulo. Además, en los apéndices del final de esta memoria se pueden encontrar todos los esquemáticos de cada placa así como sus correspondientes diseños PCB.

### 10.1. Proceso de fabricación de las PCB

Para fabricar las placas que hemos diseñado mediante el software Altium, se hará uso de una máquina de control numérico del fabricante LPKF Protomat S103[24] que está disponible en el laboratorio de electrónica del Centro de Instrumentación Científica de la Universidad de Granada (Figura 39). El software Altium nos permite generar los archivos de salida necesarios para la fabricación de la PCB a ser: los archivos de NCdrills y Gerber correspondientes a las capas Top, Bottom y mechanical 1.

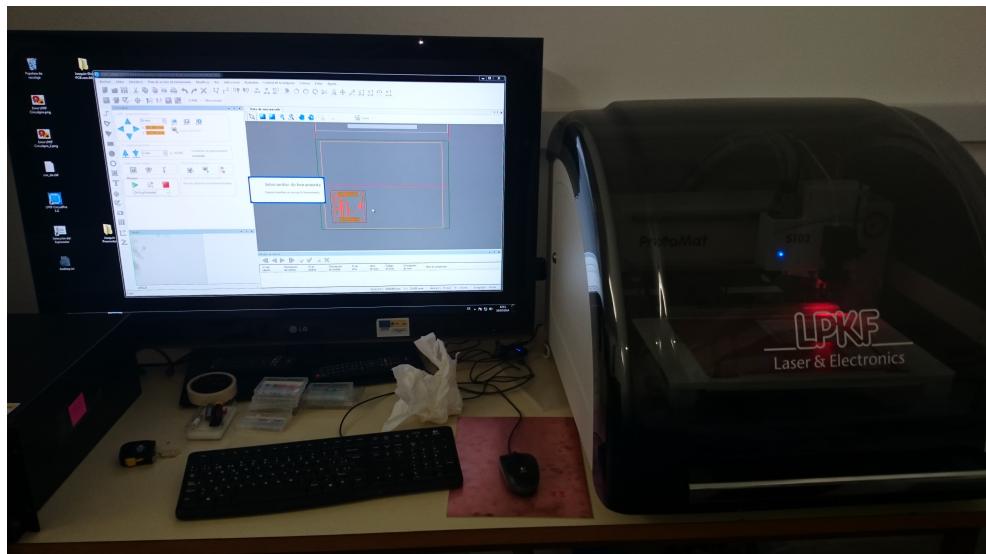


Figura 39: Máquina de control numérico para fabricación de PCBs

El proceso de fabricación se realiza de la siguiente forma:

1. Una vez generados con Altium los archivos necesarios, éstos se cargan un programa propio del fabricante de la máquina. Este programa, además de cargar los archivos de Altium, también te avisa sobre el estado de las brocas. Una vez que se comprueba que todo está bien, se delimita el área de la PCB a través del software y se inicia el proceso de fabricación. Una vez iniciado la maquina hará los taladros de la placa y marcará las pistas del circuito por

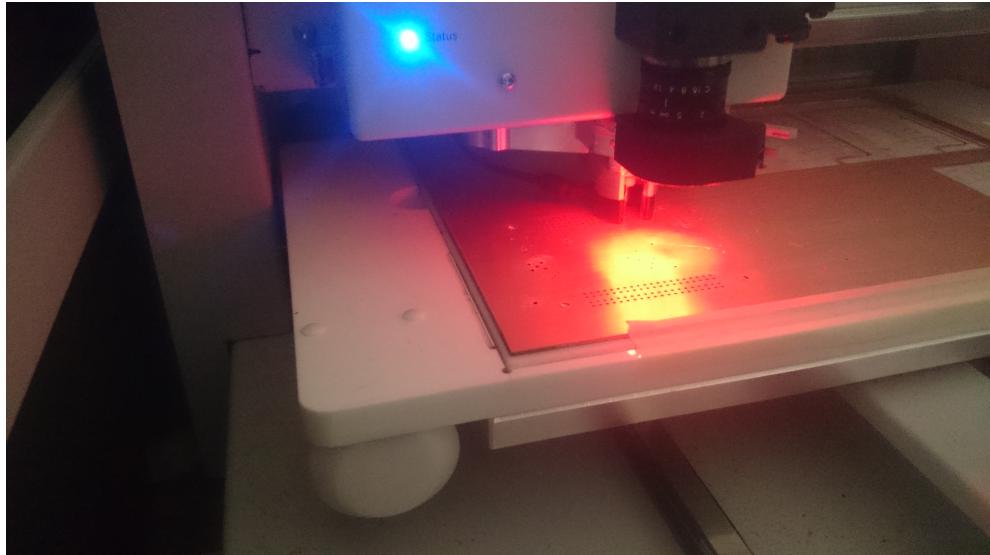


Figura 40: Fabricación de la PCB

una sola cara. Una vez acabada la primera cara habrá que darle la vuelta para que termine el proceso. La figura 40 muestra una imagen de la máquina en funcionamiento.

2. Una vez terminado el proceso de fabricación de la PCB hay que someterla a un proceso de galvanizado para metalizar todos los taladros de la placa. Para este propósito, el centro de instrumentación científica cuenta con una máquina galvanizadora LPKF modelo MiniContac RS del fabricante FPKF[25]. La figura 41 muestra las placas una vez que han salido de la galvanizadora.
3. Por último se cortan las placas para que queden listas para su posterior montaje. La figura 42 muestra una de las placas ya cortada.

En cuanto al **listado de materiales**, en el **Apéndice F: Listas de materiales** se adjunta una tabla para placa. La columna de designadores de cada placa coincide con los componentes de los esquemáticos y las PCB.

## 10.2. Estimación Económica

Como se puede observar en las listas de materiales del apéndice F, el coste total del material de componentes vale aproximadamente 71,96 €. Estos precios son del distribuidor de material electrónico Farnell. Estos precios pueden variar con el distribuidor y en función de la disponibilidad y el volumen de pedido. Generalmente será difícil poder comprar algunos componentes sueltos tales como las resistencias o los condensadores ya que al distribuidor no le saldrá a cuenta vender una sola resistencia de un céntimo de euro. Por esta razón no sería descabellado poner un margen de un 20 o

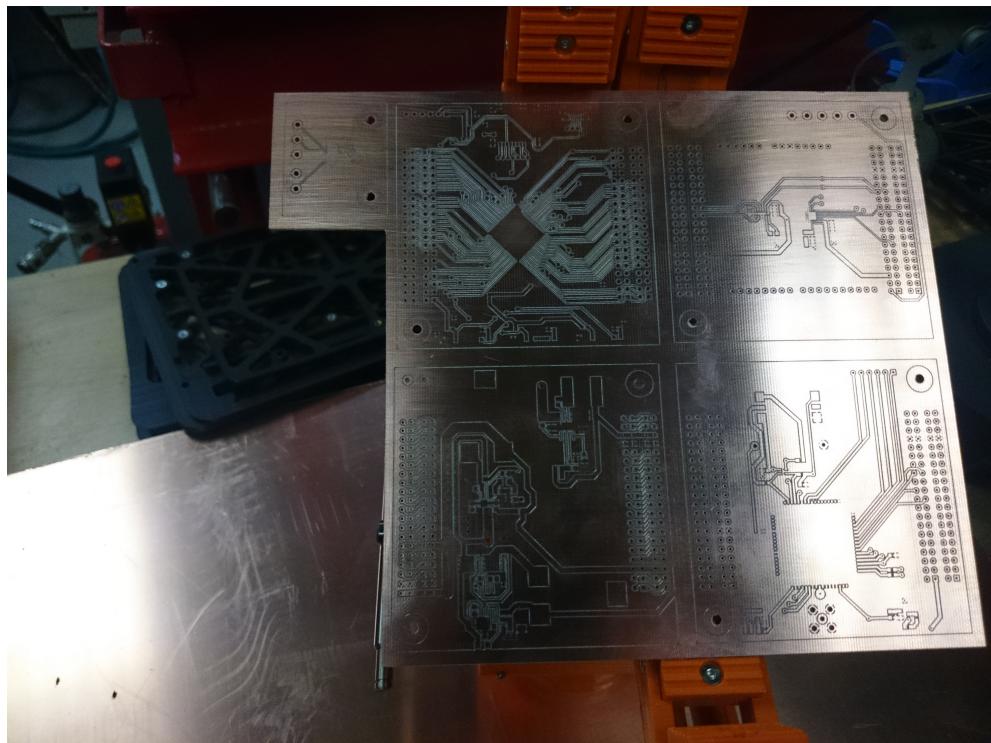


Figura 41: Placas tras el galvanizado

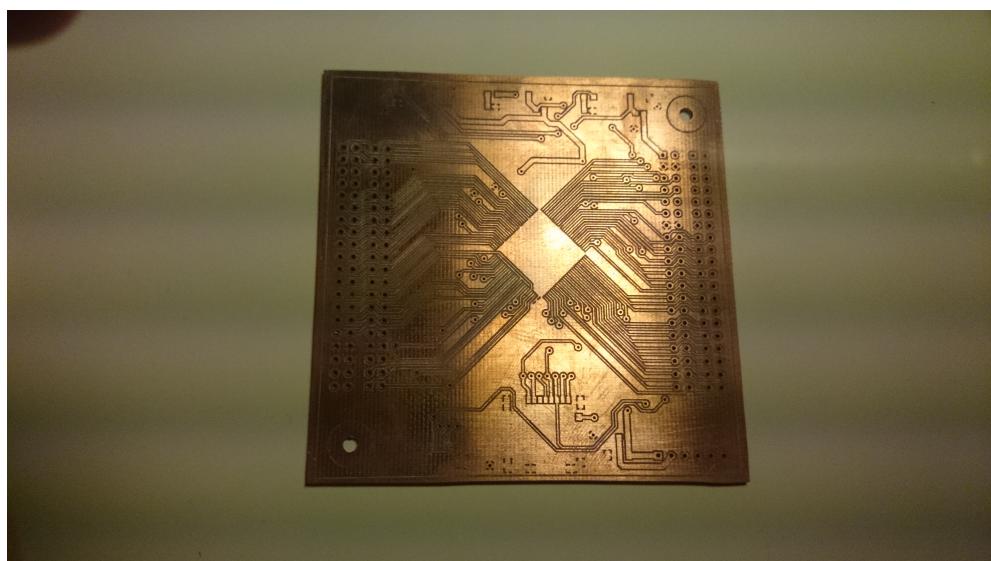


Figura 42: Placa cortada

30 por ciento más, por lo el coste de material ascendería a los 90 o 100 euros por conjunto de placas.

En cuanto al tiempo de diseño, se estima que se han realizado por placa de media unas 15 horas, 4 horas de montaje y unas 10 horas de programación del conjunto entero. Las horas de diseño incluyen selección de componentes y diseño de los esquemáticos y rutado de las PCBs. Se podría estimar que se han dedicado un total de 86 horas de diseño. Para realizar una estimación económica de lo que este trabajo costaría de mano de un ingeniero profesional se han hecho los cálculos basándonos en el convenio colectivo de la industria siderometalúrgica[27] que establece que el sueldo anual de un ingeniero debe ser de 22763€ brutos anuales dividido entre 1760 horas/año que serían de 12 euros por hora de trabajo. Por lo que el trabajo de diseño realizado se podría valorar en 1032 euros aproximadamente.

Junto a este coste de diseño habría que considerar el coste de producción que tiene la fabricación de las placas. Se estima que el proceso de fabricación de la placa fue de unas 3 horas en la máquina de control numérico para fabricar un panel entero y otras 3 horas para su galvanizado. Tomando como referencia las tarifas de los servicios del centro de instrumentación científica [28], el precio por hora de trabajo Técnico de grado medio es de 53.50 euros, por lo que el precio total aproximado del coste de fabricación sería de 321 euros.

Teniendo en cuenta las estimaciones realizadas se puede asumir que el coste de producción del sistema completo estaría valorado en torno a 1500 euros contando con los costes de material, diseño y fabricación.

Por supuesto, haciendo una fabricación en serie se puede conseguir reducir el coste unitario de cada dispositivo. Es difícil dar una estimación económica de este valor unitario ya que las empresas que fabrican y montan los circuitos suelen tener sus propios distribuidores, los cuales les hacen precio por volumen, por lo que la mejor manera de obtener una estimación del coste unitario sería pedir un presupuesto directamente a la compañía fabricante.

## 11. Conclusiones

Para concluir con este proyecto hablaremos un poco sobre como ha sido el proceso de diseño y de fabricación, así de cómo se podría mejorar este proyecto y posibles ampliaciones.

Este proyecto se ha presentado como parte del proyecto ICARUS del programa UGRASP del departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada. Se proponía la fabricación de una sonda estratosférica que fuera capaz de llegar a la estratosfera y poder tomar datos de la trayectoria seguida durante el viaje.

Los objetivos principales del proyecto eran obtener una plataforma de desarrollo propia que tuviera la opción de admitir posibles ampliaciones, poder localizar la sonda una vez cayera y almacenar información sobre la trayectoria seguida durante el viaje, por lo que se han diseñado un total de 4 tarjetas para cumplir con este propósito: una placa de control, una placa de comunicación GSM, una placa de receptor GPS y una placa de alimentación.

Para el diseño de la plataforma de desarrollo se diseñó una placa cuyo componente principal sería un microcontrolador PIC32MX del fabricante Microchip. En un principio la intención era trabajar con el compilador XC32 propio del fabricante y el framework Harmony, pero posteriormente se decidió cambiar al entorno de desarrollo MPIDE basado en Arduino debido a problemas de funcionalidad de este framework con la familia de los microcontroladores MX. Además, se ha planteado un factor de forma que permite la interconexión de las demás placas con esta del mismo modo que lo hacen las shields de arduino. Este factor de forma deja un buen número de entradas/salidas de propósito general para futuras posibles ampliaciones.

Para el desarrollo de la unidad de localización se optó por hacer un diseño basado en tecnología GSM y GPS. Para la placa de GSM se tuvo que optar por comprar una shield de arduino basada en un receptor GPS Ublox-Neo 6M. La razón por la que se optó por comprar el módulo es que existían ciertas limitaciones en cuanto a la altitud máxima en la que los receptores de GPS pueden funcionar por lo que el diseño relacionado con esta placa se ha limitado a hacer una placa adaptadora de un factor de forma de Arduino a nuestro factor de forma. En cuanto al módulo de comunicación GSM se ha optado por un SIM900 del fabricante SIMCOM. La elección de este módulo ha sido principalmente debido a la cantidad de información disponible en Internet ya que es un componente muy utilizado sobre todo por la comunidad de Arduino.

Finalmente se ha diseñado una placa de alimentación que proporciona los niveles de tensión requeridos por los distintos componentes del sistema así como el consumo que estos requieren. Esta placa está formada principalmente por un regulador lineal de 5V LM7805 y un dos regulares lineales ajustables LM317. La razón de la elección de estos componentes ha sido básicamente la sencillez y la flexibilidad que éstos proporcionan.

## **11.1. Futuras líneas de trabajo**

Como se ha dicho en el apartado anterior, se tuvo que cambiar al entorno de desarrollo MPI-DE debido a problemas de funcionalidad con el framework Harmony. Esto se debe principalmente porque este framework está enfocado a la última familia de microcontroladores que Microchip ha liberado, por lo que se propone un rediseño de la placa de control basado en un microcontrolador de esta nueva familia ya que así se evitarían los problemas de funcionalidad que se nos presentaron y tendría un mayor rendimiento. Se propone buscar en la gama de productos del fabricante DIGILENT una placa que esté basada también en un microcontrolador PIC32MZ y hacer el diseño para que sea pin compatible y poder tener así la opción de trabajar con el entorno de Microchip o con MPIDE.

Una de las principales preocupaciones cuando se trabaja con sistemas electrónicos es su eficiencia, por lo que se propone un desarrollo más profundo del programa investigando los distintos modos de ahorro de cada componente para hacer al dispositivo lo más eficiente posible así como el diseño de una tarjeta de alimentación basada en convertidores DC/DC basándose en el tipo de fuente de alimentación que llevará la placa durante el vuelo de la misión (tipo de baterías, etc..). Se propone por ejemplo apagar el módulo GSM cuando éste entre en la zona sin cobertura.

Siguiendo en la línea de la mejora de la alimentación también podría diseñarse un sistema de alimentación solar. Este sistema de alimentación solar aunque fuese parcial, podría alimentar algunos de los dispositivos más importantes como puede ser el microcontrolador.

Sería interesante diseñar una tarjeta que incluyera módulo GSM y GPS. De esta manera conseguiríamos reducir bastante el espacio y el peso. Éstos dos parámetros son cruciales a la hora de lanzar el globo ya que influye mucho en el empuje del globo y, por tanto, su velocidad de subida.

En el estudio de la trayectoria de la barquilla es interesante recoger datos sobre la temperatura exterior. En un principio se propuso fabricar un módulo digital de sensor de temperatura, pero las temperaturas extremas que pueden llegar a alcanzarse en el exterior hacen este dispositivo inviable, por lo que se podría usar una sonda de temperatura. Esta sonda consiste en un resistencia que varía con la temperatura. Éste sistema es muy fácil de montar y no necesita del diseño de ninguna tarjeta especial para controlarlo, tan solo formar un divisor de tensión y conectarlo a una entrada ADC de la tarjeta controladora. Estas sondas tienen la ventaja de que no van a dejar de funcionar con la temperatura, tan solo puede que no midan bien cuando se sale del rango, aun así, la temperatura mínima que pueden llegar a medir ronda los -80°C.

Finalmente, como propone uno de los objetivos del programa UGRASP-ICARUS, se puede

diseñar un sistema de control de actitud basado en sensores de movimiento como acelerómetros y giróscopos que accionen una serie de válvulas que regulan un gas expulsado por una pequeñas toberas situadas fuera de la barquilla y que tengan una orientación vertical y horizontal. De esta manera podrían controlarse la orientación horizontal de la barquilla y la estabilización durante el descenso o el ascenso.



## Referencias

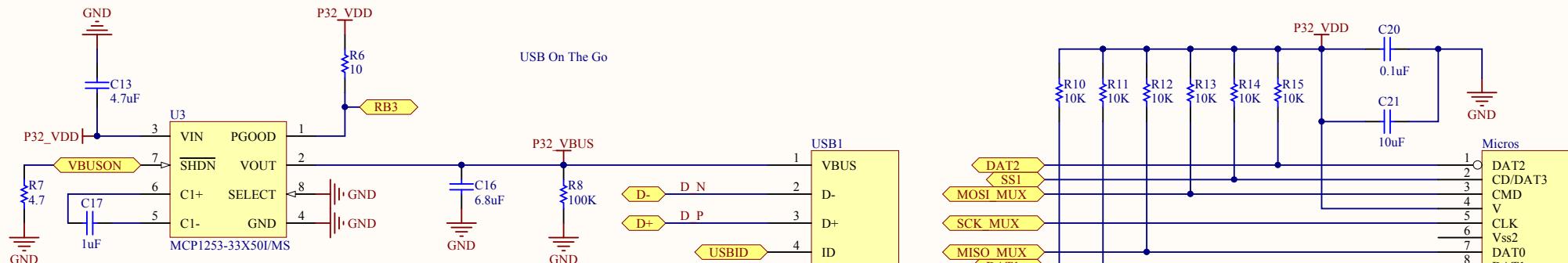
- [1] **Asociación GSM**  
(<http://www.gsma.com/>)
- [2] MPLAB Harmony Help  
(<http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB%20Harmony%20Help%20and%20Release%20Notes%20v1.04.02%29.pdf>)
- [3] Coordinating Committee for Multilateral Export Controls  
(<https://en.wikipedia.org/wiki/CoCom>)
- [4] Diseño e implementación de sistemas embebidos con PIC. (MC Electronics)
- [5] CHIPKIT MAX32 Board Reference  
([http://ww1.microchip.com/downloads/en/DeviceDoc/chipKIT%20Max32\\_rm.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/chipKIT%20Max32_rm.pdf))
- [6] Esquemáticos de la placa MAX32  
([http://www.digilentinc.com/Data/Products/CHIPKIT-MAX32/chipKIT%20Max32\\_bysa\\_sch.pdf](http://www.digilentinc.com/Data/Products/CHIPKIT-MAX32/chipKIT%20Max32_bysa_sch.pdf))
- [7] PIC32 MX Family reference manual  
([http://hades.mech.northwestern.edu/images/2/21/61132B\\_PIC32ReferenceManual.pdf](http://hades.mech.northwestern.edu/images/2/21/61132B_PIC32ReferenceManual.pdf))
- [8] Datasheet PIC32MX795F512  
(<http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf>)
- [9] TTL to USB Serial Converter  
([http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS\\_TTL-232R\\_CABLES.pdf](http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_CABLES.pdf))
- [10] u-blox 6 Receiver Description Including Protocol Specification  
([http://www.u-blox.com/images/downloads/Product\\_Docs/u-blox6\\_ReceiverDescriptionProtocolSpec\\_%28GPS.G6-SW-10018%29.pdf](http://www.u-blox.com/images/downloads/Product_Docs/u-blox6_ReceiverDescriptionProtocolSpec_%28GPS.G6-SW-10018%29.pdf))
- [11] Fabricante DIGILENT  
(<https://www.digilentinc.com/>)
- [12] Proyecto UGR Aerospace  
(<http://atcproyectos.ugr.es/ugrasp/blog/>)
- [13] Totex  
(<http://www.totex.jp/>)
- [14] RealTerm  
(<http://realterm.sourceforge.net/>)

- [15] Turnigy 2200mAh 3S 20C Lipo Pack  
([http://www.hobbyking.com/hobbyking/store/\\_/8932\\_Turnigy\\_2200mAh\\_3S\\_20C\\_Lipo\\_Pack.html](http://www.hobbyking.com/hobbyking/store/_/8932_Turnigy_2200mAh_3S_20C_Lipo_Pack.html))
- [16] www.gpsinformation.org  
(<http://www.gpsinformation.org/dale/nmea.htm>)
- [17] TTL to USB Serial Converter  
([http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS\\_TTL-232R\\_CABLES.pdf](http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_CABLES.pdf))
- [18] u-blox 6 Receiver Description Including Protocol Specification  
([http://www.u-blox.com/images/downloads/Product\\_Docs/u-blox6\\_ReceiverDescriptionProtocolSpec\\_%28GPS.G6-SW-10018%29.pdf](http://www.u-blox.com/images/downloads/Product_Docs/u-blox6_ReceiverDescriptionProtocolSpec_%28GPS.G6-SW-10018%29.pdf))
- [19] www.gpsinformation.org  
(<http://www.gpsinformation.org/dale/nmea.htm>)
- [20] Especificaciones de diseño GSM  
([http://www.simcom.us/act\\_admin/supportfile/SIM900\\_HD\\_V1.01%28091226%29.pdf](http://www.simcom.us/act_admin/supportfile/SIM900_HD_V1.01%28091226%29.pdf))
- [21] Protocolo AT SIM900  
([http://www.propox.com/download/docs/SIM900\\_AT.pdf](http://www.propox.com/download/docs/SIM900_AT.pdf))
- [22] Datasheet LM7805  
(<https://www.sparkfun.com/datasheets/Components/LM7805.pdf>)
- [23] Datasheet LM317  
(<http://www.ti.com/lit/ds/symlink/lm117.pdf>)
- [24] Máquina de control numérico Protomat S103  
(<http://www.lpkf.com/products/rapid-pcb-prototyping/circuit-board-plotter/protomat-s103.htm>)
- [25] Máquina de galvanizado LPKF modelo MiniContac RS  
([http://www.lpkfusa.com/RapidPCB/ThroughHolePlating/minicontac\\_rs.htm](http://www.lpkfusa.com/RapidPCB/ThroughHolePlating/minicontac_rs.htm))
- [27] Convenios colectivos de la Industria Siderometalúrgicas  
(<http://www.camaragranada.org/descargas/corporacion/publicaciones/convenios-colectivos-de-la-provincia-de-granada/industrias-siderometalurgicas.aspx#go>)
- [28] Tarifas del laboratorio de electrónica del centro de instrumentación científica de la Universidad de Granada  
(<http://cic.ugr.es/servicios-y-unidades/ficha.php?unidad=35&codServicio=8&verTarifas=S>)

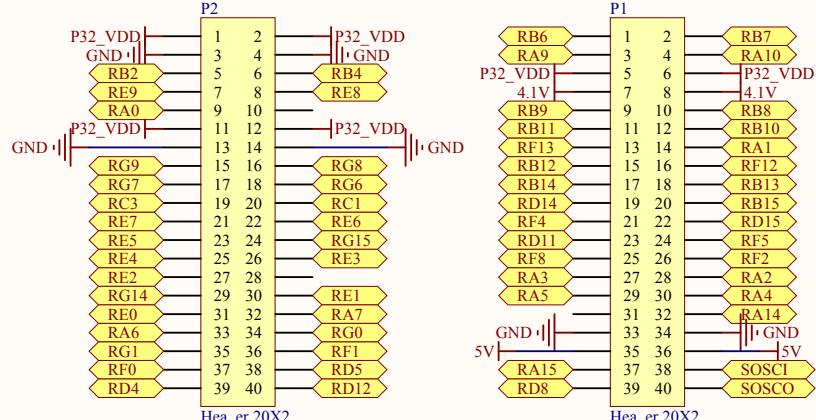
## **Apéndice A: Esquemáticos de la placa de desarrollo.**

1 2 3 4

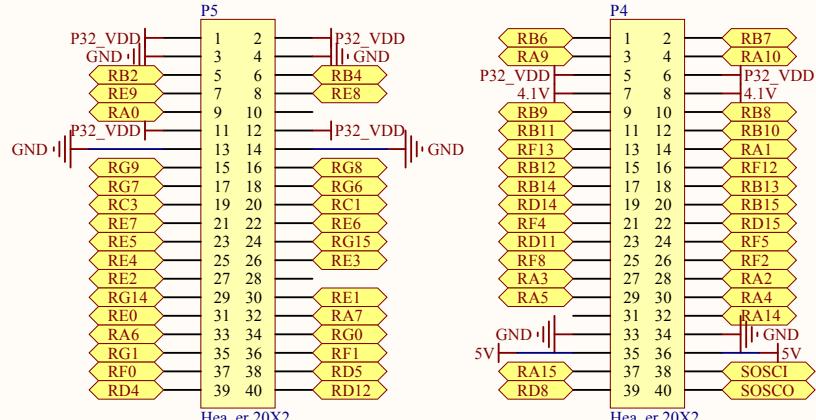
A



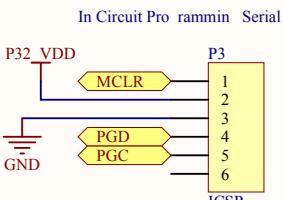
B



C

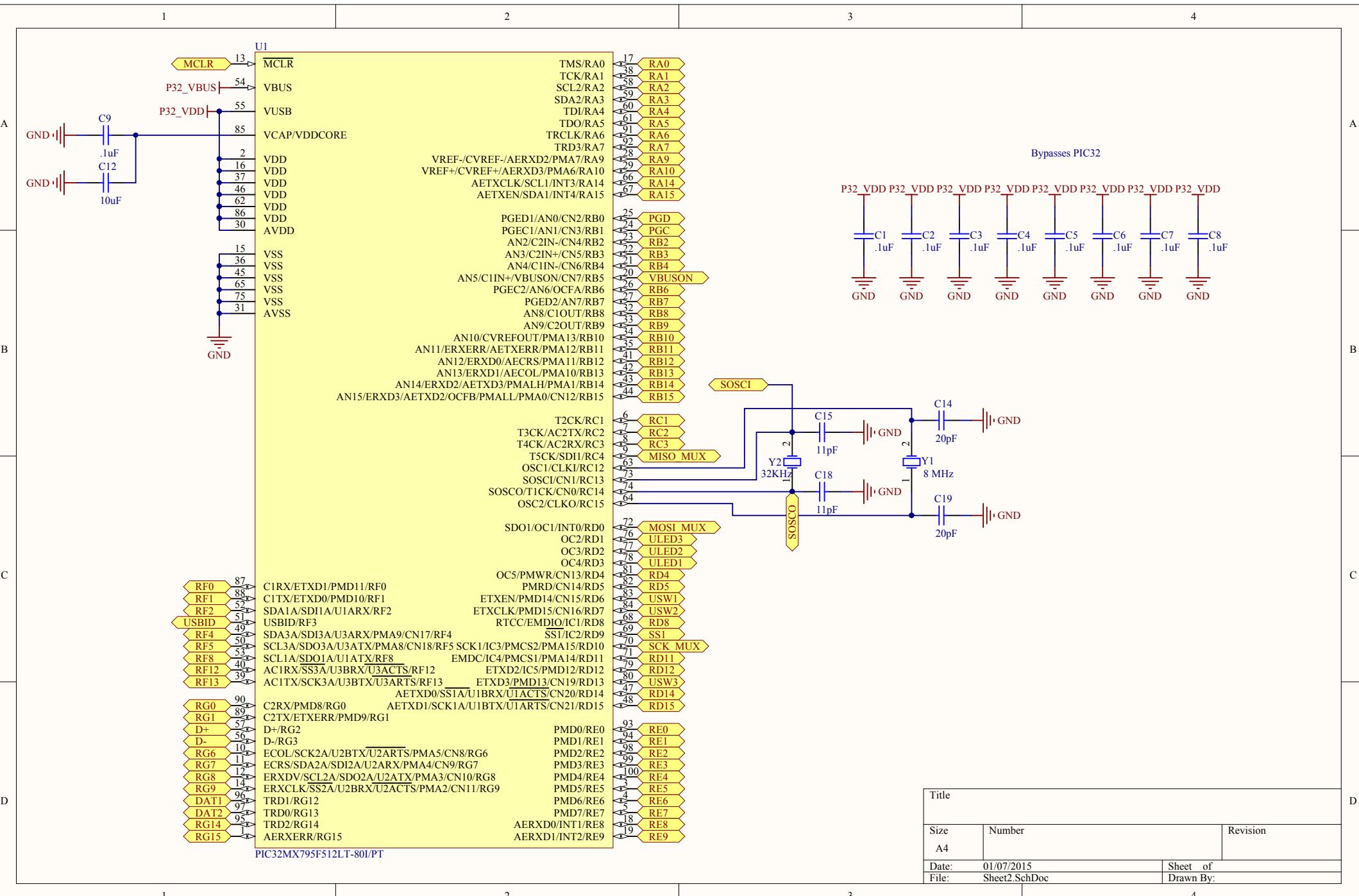


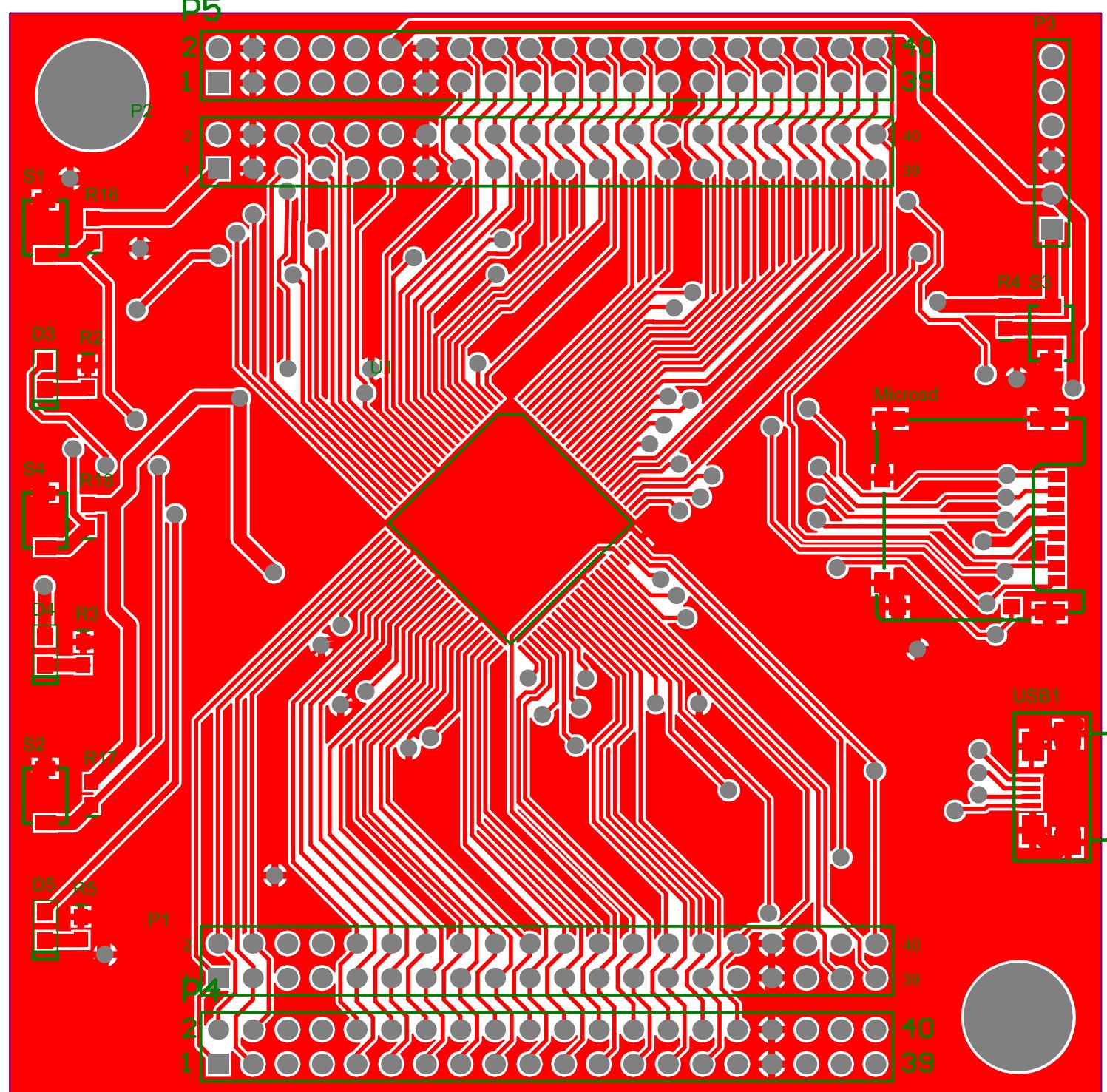
D

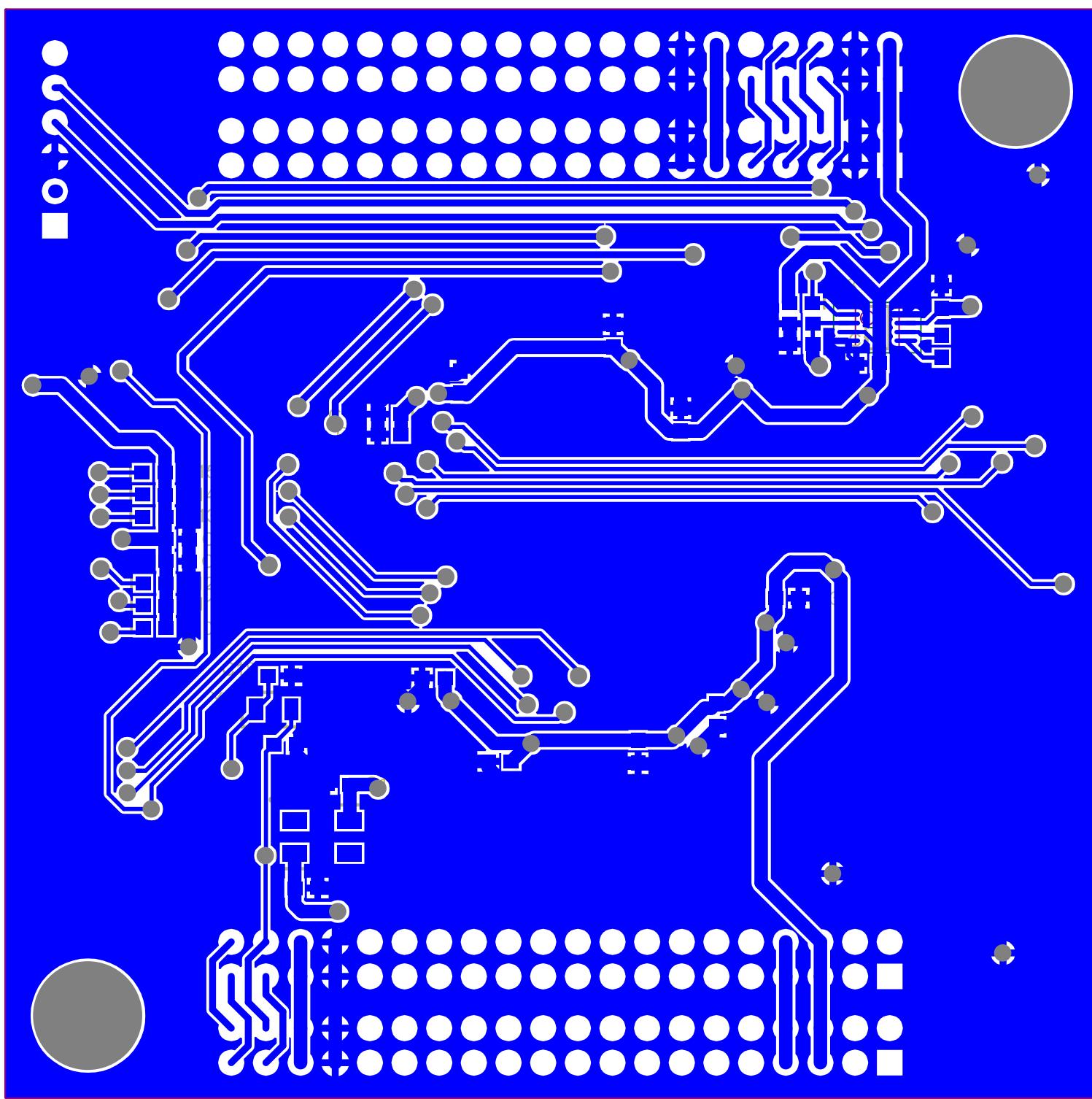


Title		
Size	Number	Revision
A4		
Date:	01/07/2015	Sheet of
File:	Sheet1.SchDoc	Drawn By:

1 2 3 4

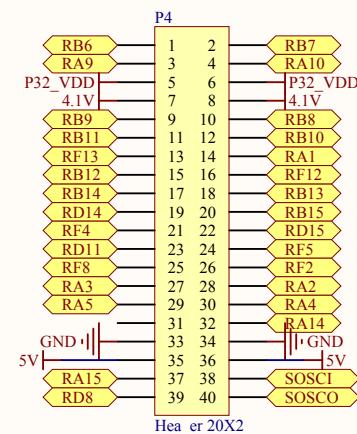
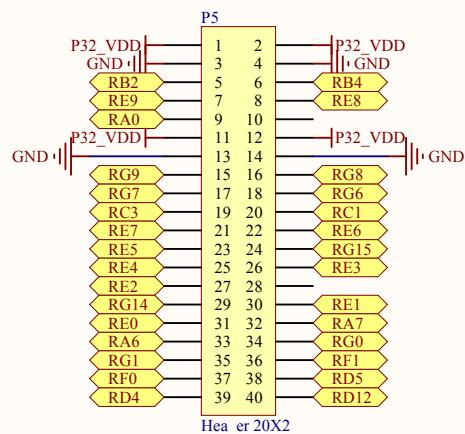
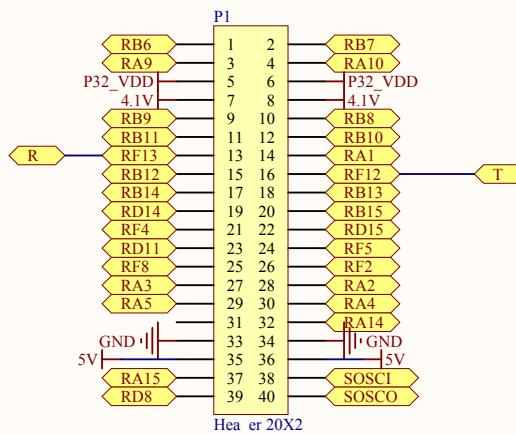
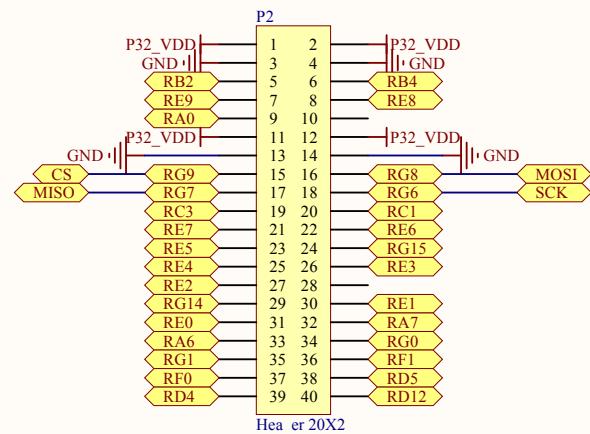




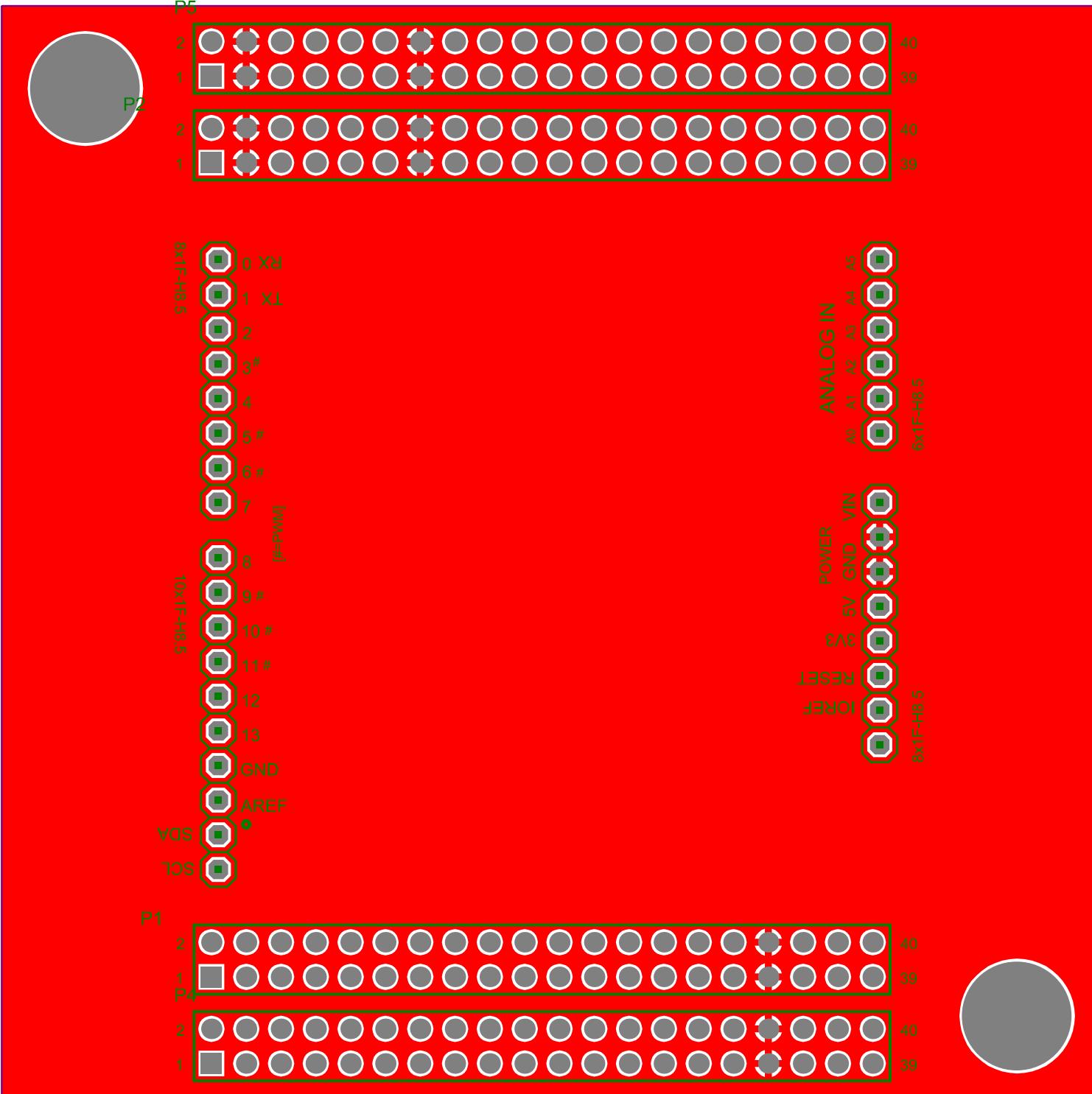


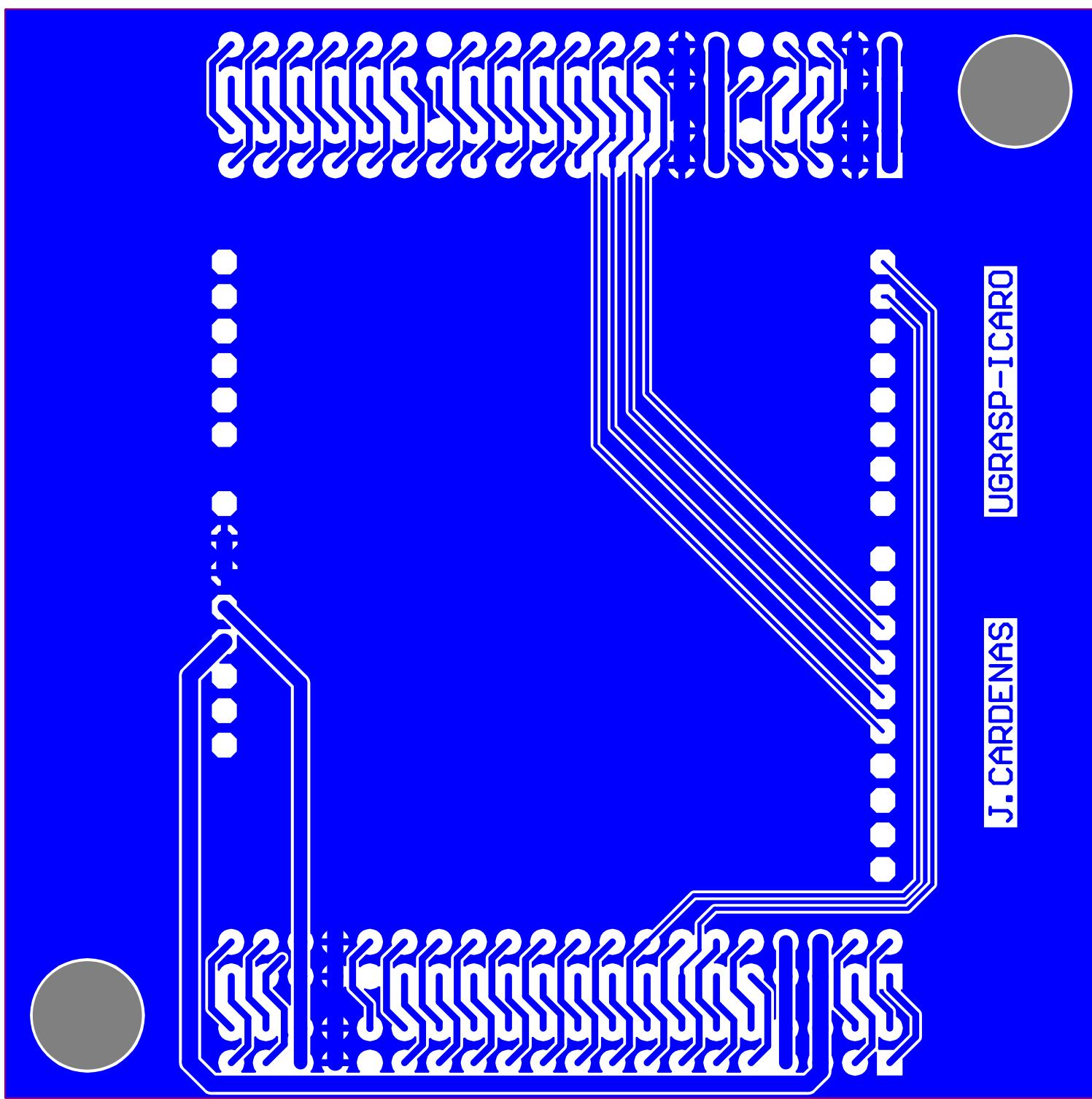


## Apéndice B: Esquemáticos de la placa GPS.



Title		
Size A4	Number	Revision
Date: 01/07/2015	Sheet of	
File: C:\Users..GPS.SchDoc	Drawn By:	





## Apéndice C: Esquemáticos de la placa de alimentación.

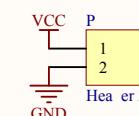
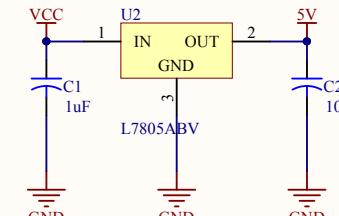
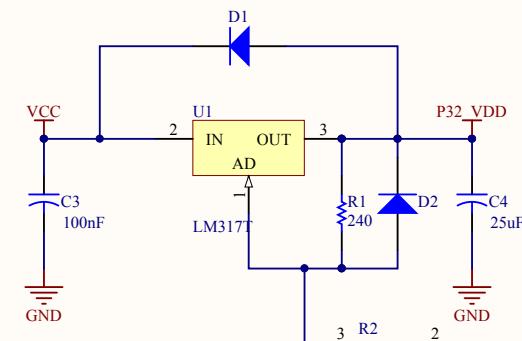
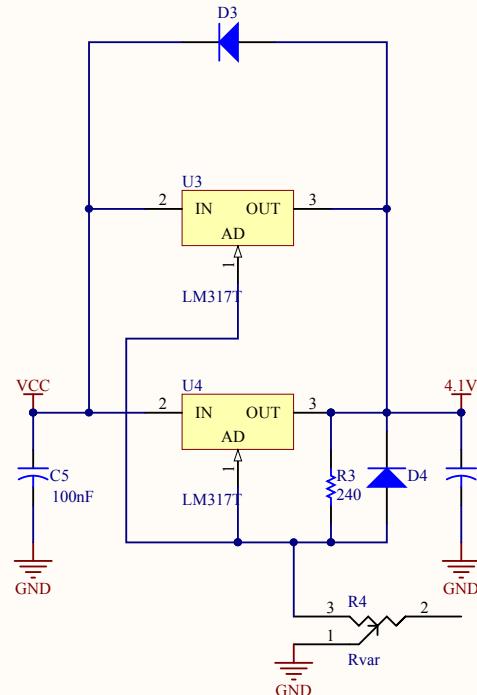
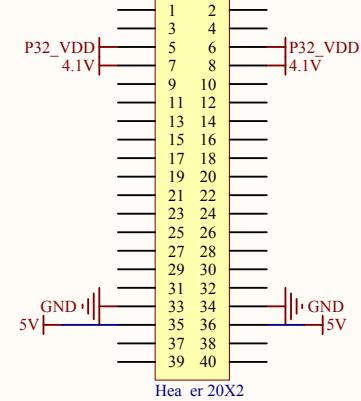
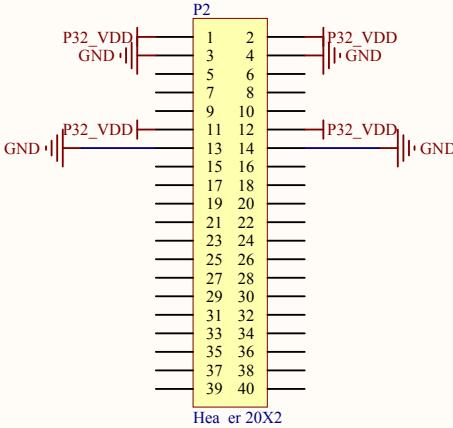
1

2

3

4

A



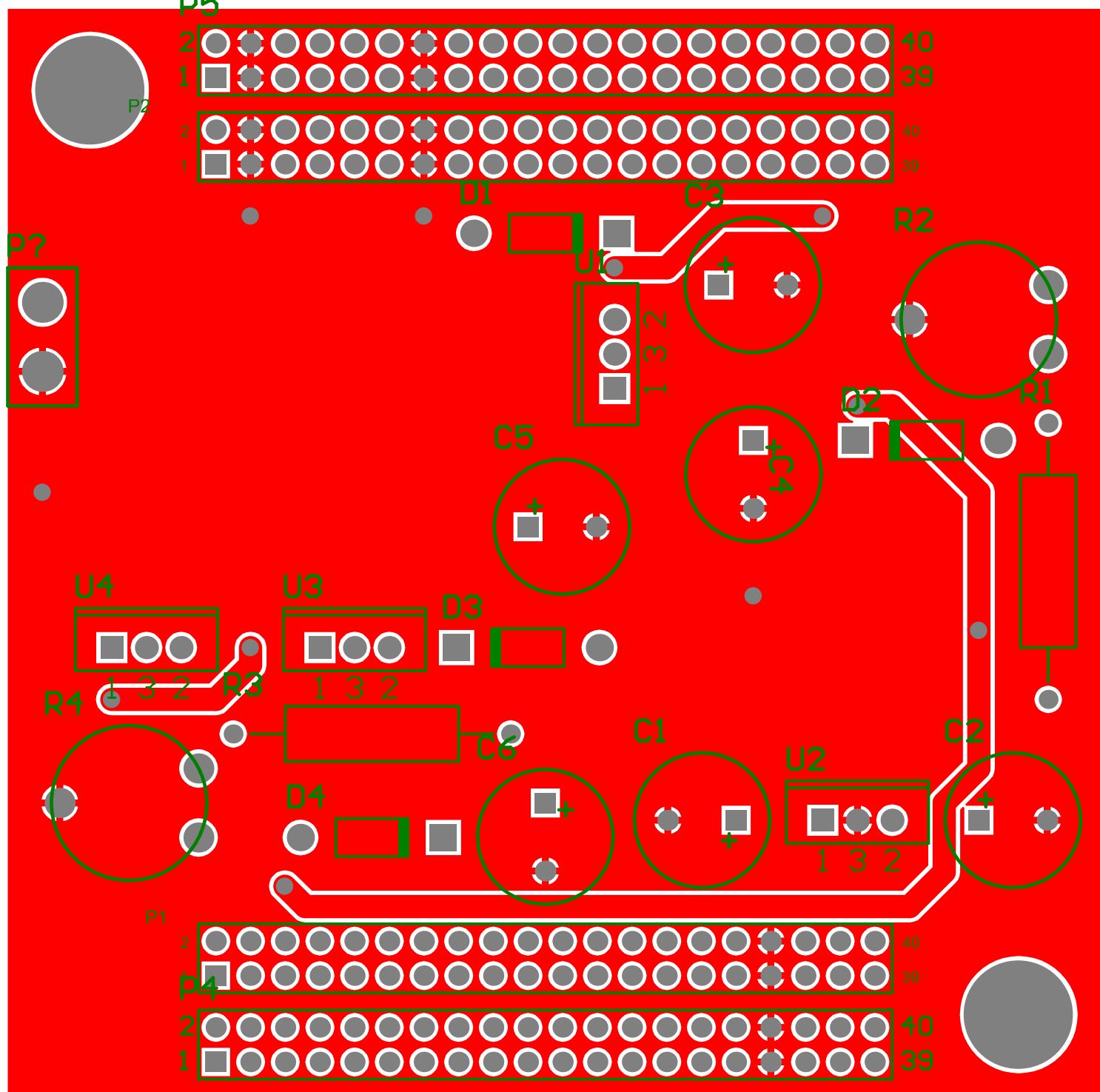
Title		
Size	Number	Revision
A4		
Date: 01/07/2015	Sheet of	
File: C:\Users...POWERPCB.SchDot		Drawn By:

1

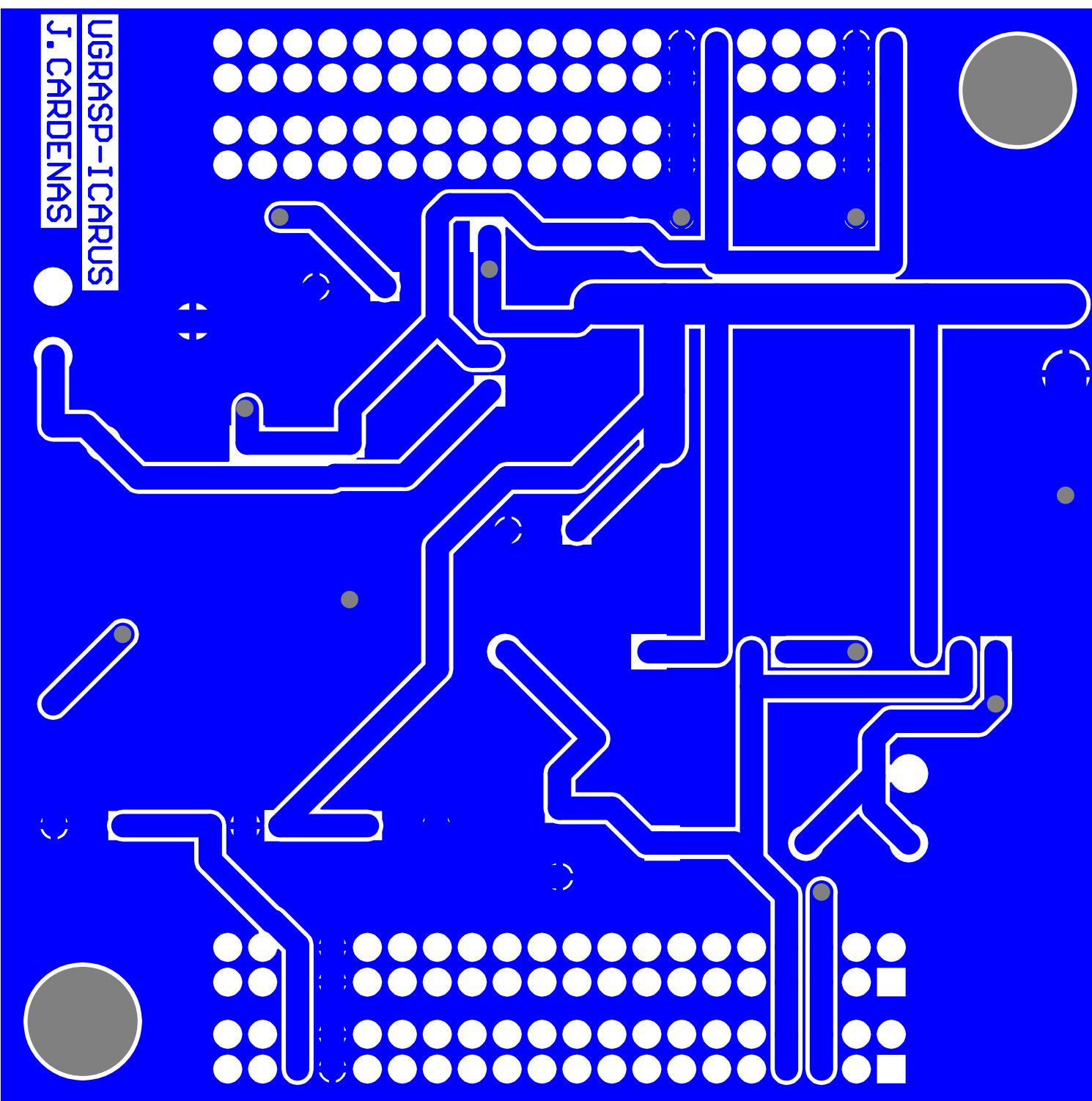
2

3

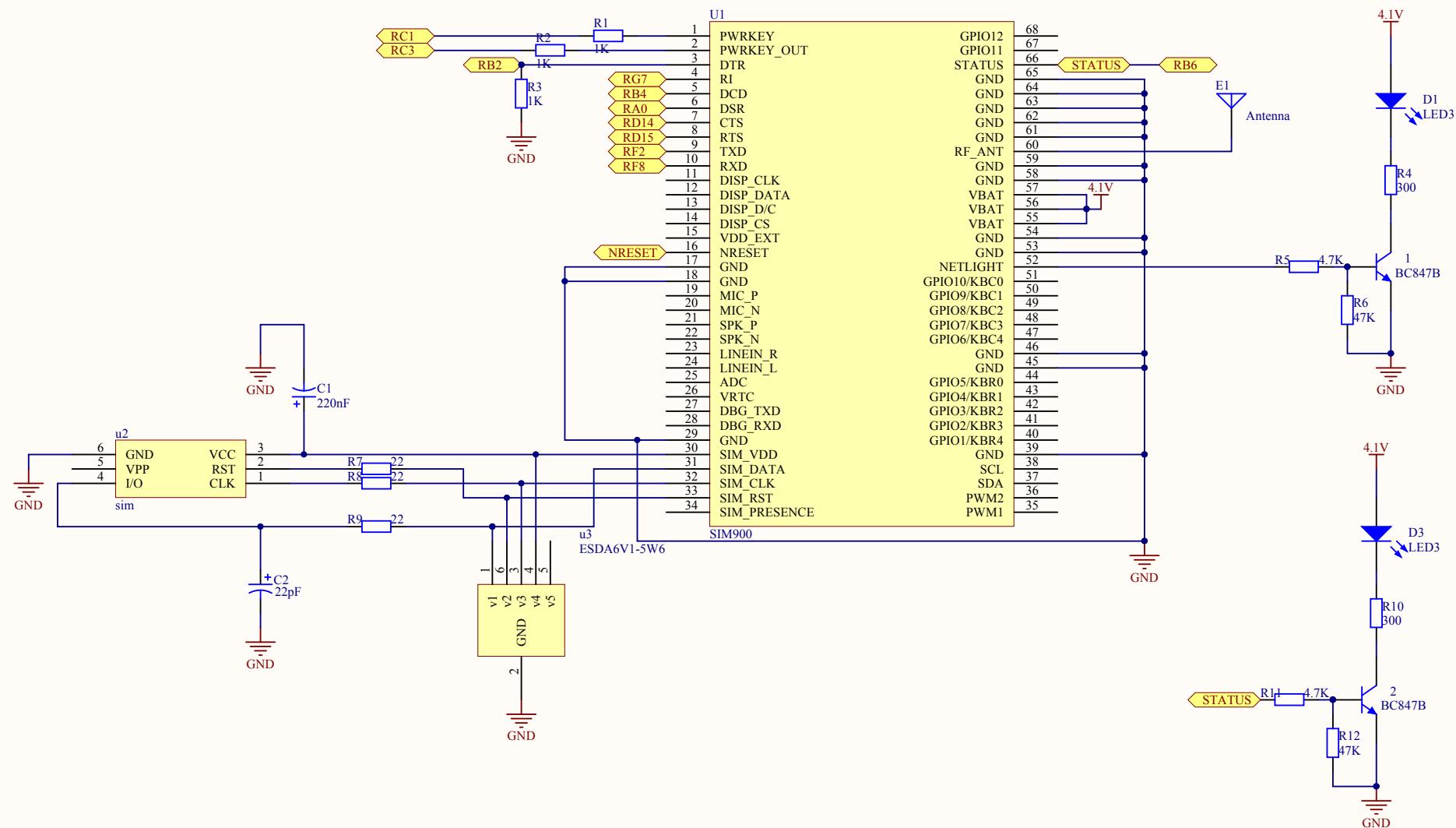
4



UGRASP-ICARUS  
J. CARDENAS

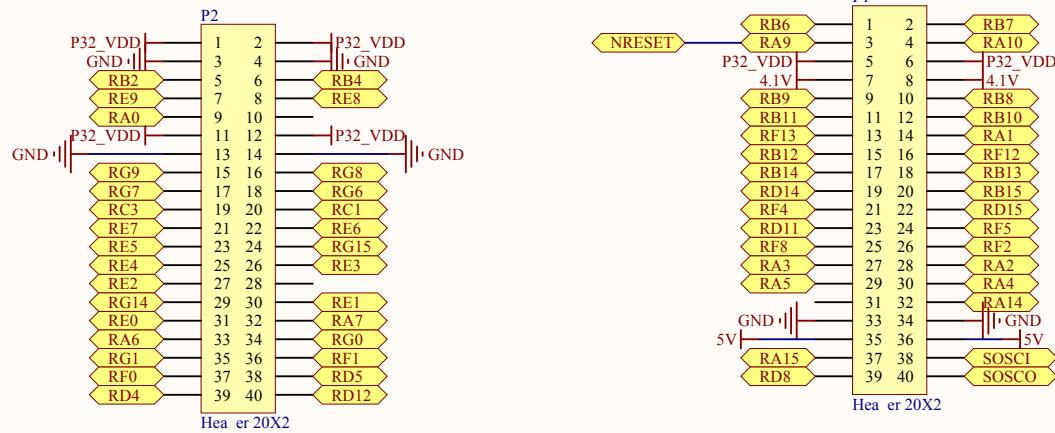


## Apéndice D: Esquemáticos de la placa de GSM.

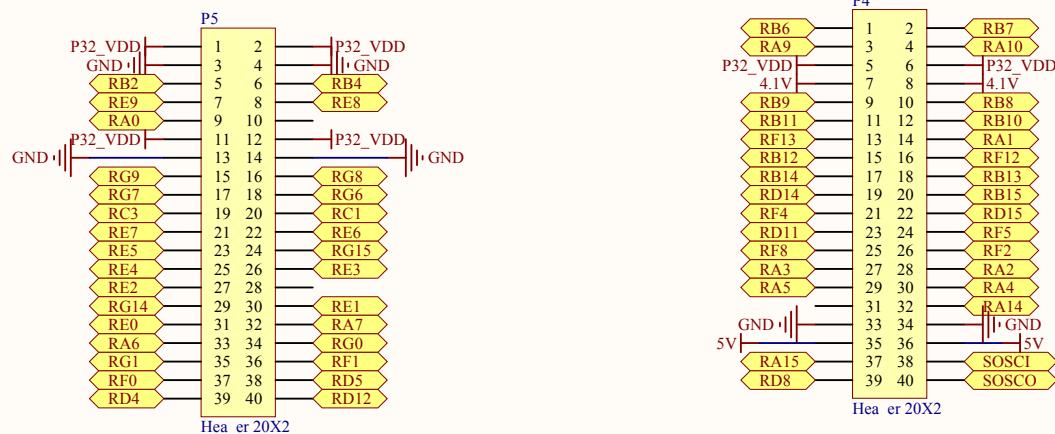


Title		
Size A4	Number	Revision
Date:	01/07/2015	Sheet of
File:	Sheet1.SchDoc	Drawn By:

A



B



C

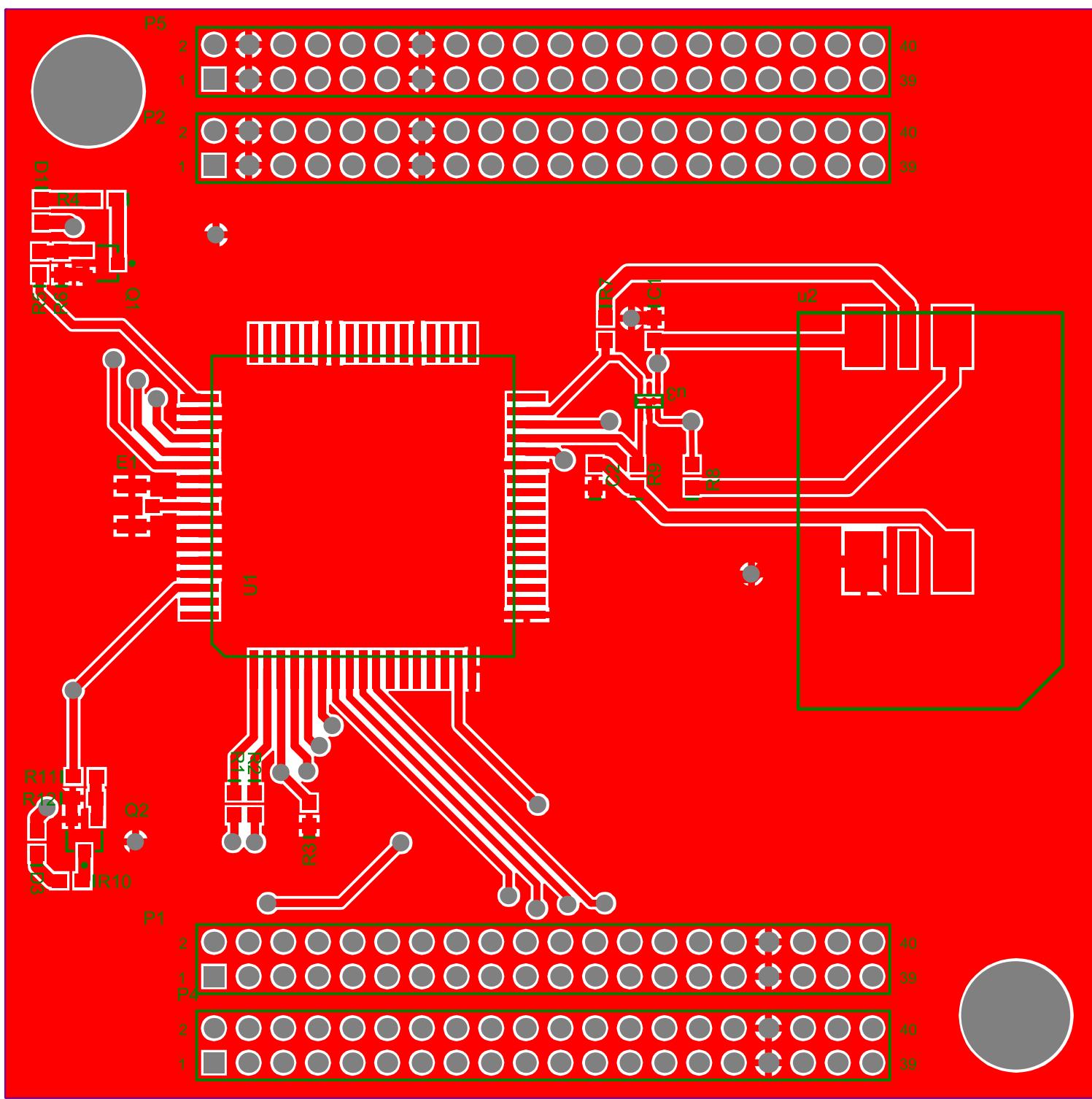
Title		
Size	Number	Revision
A4		
Date:	01/07/2015	Sheet of
File:	Sheet1.SchDoc	Drawn By:

A

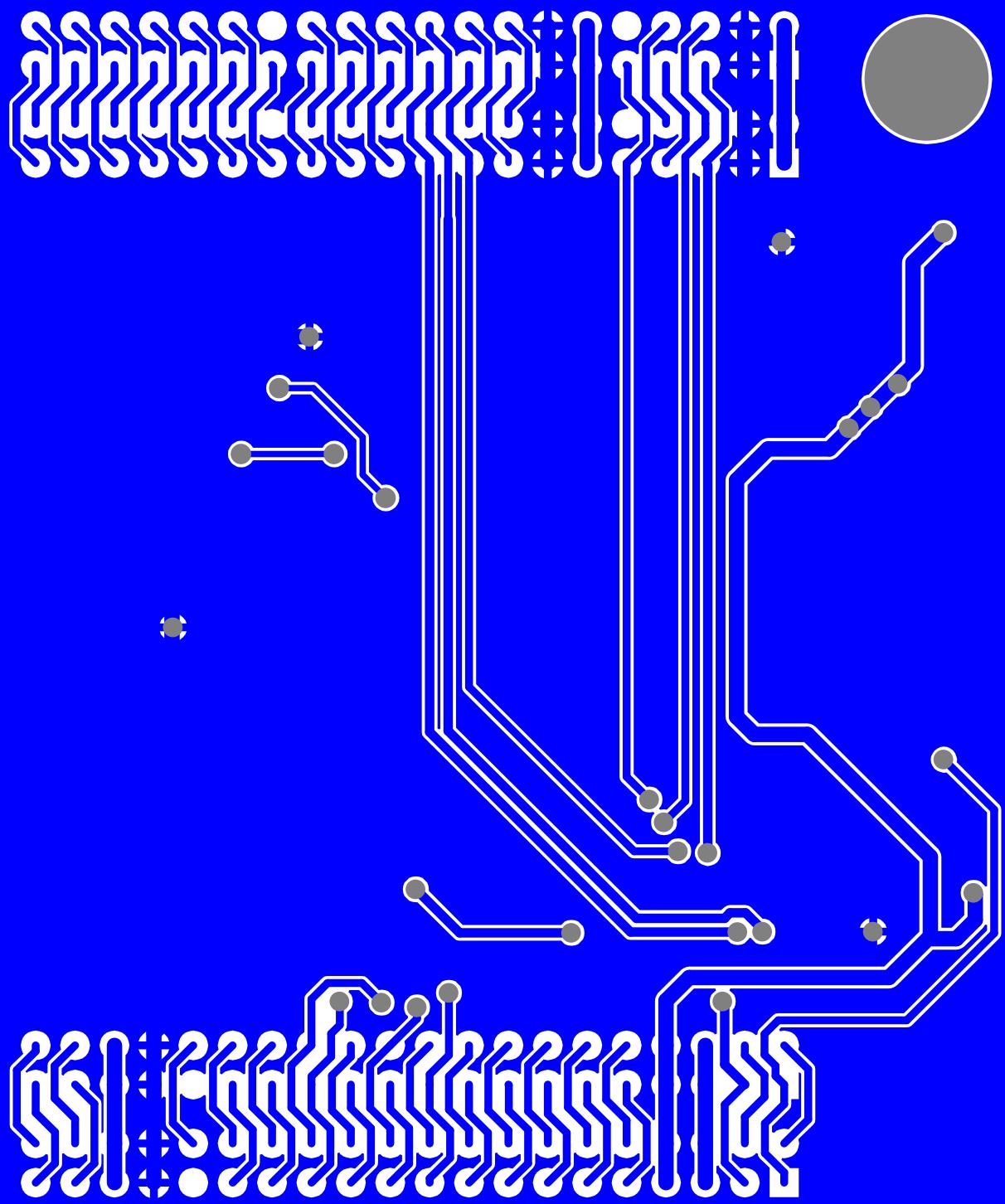
B

C

D



UGRASP-ICARUS  
J.CARDENAS





## Apéndice E: Implementación.

Este apéndice pretende servir como referencia de las funciones implementadas en el código para cada uno de los módulos, así como una descripción de cada una de ellas. El código completo puede consultarse al final de esta sección o en el CD que acompaña a este trabajo junto con el resto de material que se ha desarrollado. Además, y para apoyar la iniciativa del software libre, se ha publicado un repositorio público en **GitHub** con el objetivo de que más gente pueda participar en el proyecto aportando su granito de arena y ayudando a que el proyecto crezca. Se puede acceder a esta fuente a través del enlace <https://github.com/josancardenasm/UGRASP-ICARUS>

**void getNMEA1 (void)** Esta función se encarga de sondear todas las tramas que nos envía el receptor GPS y capturar aquella que nos interese. Ésta tiene un comportamiento **bloqueante**, lo que quiere decir que el programa estará dentro de la función permanentemente mirando las tramas que llegan del receptor hasta que se encuentre con la que queremos decodificar. Una vez encontrada la enviamos a la función **descifrarTrama** a través de la cadena de caracteres “TRAMA” para su descomposición en distintas variables. El comportamiento bloqueante de esta función tiene una clara ventaja y es que de esta manera tenemos la seguridad de que recibimos la trama sin que se pueda perder información debido a que el microcontrolador esté ocupado en otras tareas. La desventaja en este caso sería que el programa puede pasarse demasiado tiempo en esta función y no atender a tiempo otras funciones. Las variables con las que trabaja esta función son:

**flag** Esta variable actua como indicador de que se ha encontrado un inicio de trama.

**NMEA** Es una cadena de caracteres que hace buffer donde se almacena temporalmente la trama que se está recibiendo para luego comprobar si es la nuestra.

**TRAMA** En esta cadena de caracteres se almacenará la trama capturada que nos interesa para después ser procesada.

**tiempo** Variable temporal que aumenta cada vez que encontramos nuestra trama. Como se explicaba en el capítulo 6.8, esto suele ocurrir cada segundo según la configuración del módulo. La finalidad de esta variable es tener una noción temporal aproximada para que cuando llegue a cierta cuenta, por ejemplo 60 (1 minuto), realizar una acción determinada como puede ser el envío con la posición de la barquilla en nuestro caso.

**void getNMEA2 (void)** Esta función tiene el mismo objetivo que la función getNMEA1 con la única diferencia de que su comportamiento es **no bloqueante**. Ésto quiere decir que en lugar de estar observando constantemente el puerto serie del receptor GPS hasta encontrar nuestra trama, en cada ciclo del loop principal se comprueba si existe información disponible en el buffer de entrada del puerto serie para ser procesada. Al contrario que con getNMEA1, la desventaja de usar esta función está en que se puede perder información al estar el programa trabajando en otra tarea. Por el contrario, se pueden atender otras tareas de manera más

frecuente. Las variables con las que trabaja esta función son las mismas que las que trabajan con getNMEA1.

**void descifrarTrama(void)** Esta función hace un escaneo de la cadena TRAMA y la descompone en distintas variables. Luego escribe un mensaje con formato en la cadena de caracteres “respuesta”. Este mensaje será escrito en la memoria SD mediante una función “writeFileSD(respuesta)”. Tras escribir el mensaje en la SD transformará las variables lon (longitud) y lat (latitud) a un formato legible por google maps. Escribirá el enlace con los datos transformados y lo guardará en la cadena de caracteres “gleMapsMesg”. Este mensaje es que se enviará por mensaje de texto cuando el programa lo solicite.

**void writeFileSD(char \*)** Esta función es la que se encarga de escribir en la tarjeta SD. Se le pasa como parámetro una cadena de caracteres y la guarda en memoria. No devuelve ningún valor. La implementación de esta función está basada en las funciones Open, write y close de la librería “SD.h“.

**void enviarMensaje(char \*)** Esta función se encarga de realizar la secuencia de comandos AT necesaria para enviar un mensaje un mensaje de texto. Se le pasa como parámetro la cadena de caracteres que se desea enviar. No devuelve ningún valor. Los comandos usados son “AT+CMGF=1” que activa el modo de mensajes de texto, y el comando “AT+CMGS=”nº de teléfono” que se utiliza para enviar el mensaje de texto. Una vez mandado el carácter ascii \n\r, el módulo esperará que se le envíe la cadena de texto que se desea enviar. El final de la cadena de texto se delimita con el carácter ascii “0x1A”.

**void check(void)** Esta función hace de puente entre el puerto serie “Serial” y el puerto serie “Serial2”. Todo lo que se escriba en un puerto serie se recibirá en el otro. No necesita ningún parámetro para funcionar.

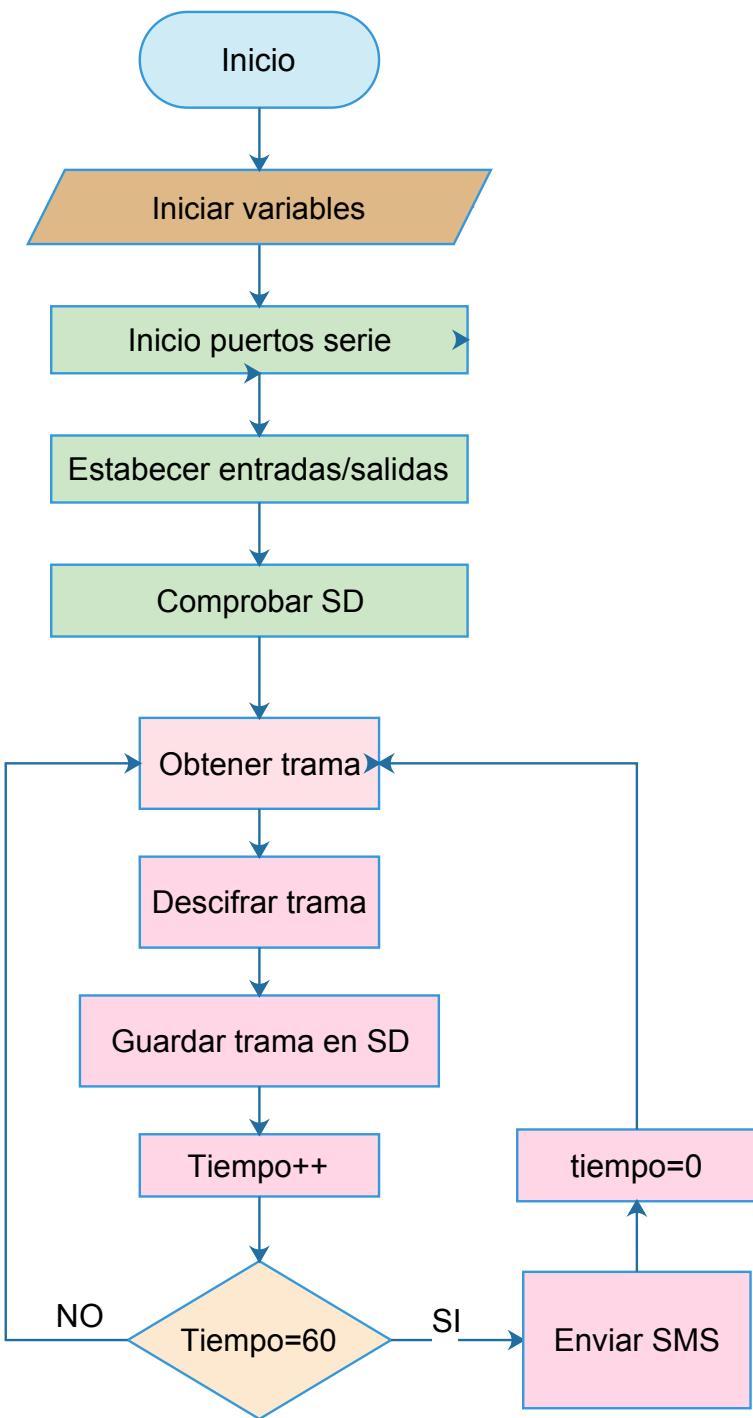


Figura 43: Diagrama de flujo del sistema

```

1
2
3  /*****INCLUDES*****/
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <SD.h>
9 #include <math.h>
10
11 /*****DEFINES*****/
12
13 //LEDS
14 #define LEDV 5
15 #define LEDA 6
16 #define LEDR 9
17
18 #define BUT1 47
19 #define BUT2 76
20 #define BUT3 77
21
22 #define PWRKEY 86 //pin de apagado y encendido del GSM
23
24
25 /*****VARIABLES*****/
26
27 //GPS
28 char NMEA[100]; //Buffer para la trama NMEA
29 char TRAMA[100]; //Buffer para la trama NMEA Capturada
30 int i=0;
31 char c='0';
32 int flag=0;
33 int tiempo=0;
34 float hora; //Hora UTC
35 float lat; //Latitud
36 char ilat; //Indicador norte/sur
37 float lon; //Longitud
38 char ilon; //Indicador longitud este/oeste
39 int fix; //Flag de posición fijada
40 int sat; //Número de satélites usados
41 float HDOP; //No se que es
42 float alt; //Altitud
43 char unit; //Unidad de altitud
44
45 //GSM
46
47 int enviarSMSFlag=0;
48
49
50 //SD
51 File myFile; //Manejador para la SD
52 const int chipSelect_SD_default = 53;
53 const int chipSelect_SD = chipSelect_SD_default;
54 int fileExist=0;
55 char mensaje[200];
56 char ggleMapsMesg[200];
57
58 /*****PROTOTIPOS*****/
59

```

```

60 void getNMEA1( void ) ; //Modo Bloqueante
61 void getNMEA2( void ) ; //Modo No bloqueante
62 void descifrarTrama( void ) ;
63 void writeFileSD( char * ) ;
64 void enviarMensaje ( char * ) ;
65 void check( void ) ;
66
67
68 /******APLICACION*****/
69
70 void setup() {
71
72     delay ( 5000 ) ;
73
74     //GPS
75     Serial.begin ( 9600 ) ;
76     Serial2.begin ( 9600 ) ;
77     Serial3.begin ( 38400 ) ;
78
79
80     Serial2.println ( " Aqui empiezo " ) ;
81
82     //LEDS
83     pinMode ( LEDV, OUTPUT ) ;
84     pinMode ( LEDA, OUTPUT ) ;
85     pinMode ( LEDR, OUTPUT ) ;
86     digitalWrite ( LEDV,LOW ) ;
87     digitalWrite ( LEDA,LOW ) ;
88     digitalWrite ( LEDR,LOW ) ;
89
90     //GSM
91
92     pinMode ( PWRKEY,OUTPUT ) ;
93     pinMode ( BUT1,INPUT ) ;
94     pinMode ( BUT2,INPUT ) ;
95     pinMode ( BUT3,INPUT ) ;
96     digitalWrite ( PWRKEY,HIGH ) ;
97
98     //SD ( Para ayuda mirar ejemplo Files )
99     pinMode ( chipSelect_SD_default , OUTPUT ) ;
100    digitalWrite ( chipSelect_SD_default , HIGH ) ;
101    pinMode ( chipSelect_SD , OUTPUT ) ;
102    digitalWrite ( chipSelect_SD , HIGH ) ;
103
104    //CHequeo de la SD
105    if ( !SD.begin ( chipSelect_SD ) ) {
106        Serial2.println ( " initialization failed ! " ) ;
107        digitalWrite ( LEDR,HIGH ) ;
108    }
109    if ( SD.exists ( " log.txt " ) ) {
110        Serial2.println ( " example.txt exists . " ) ;
111    }
112
113 }
114
115 void loop() {
116
117     //Desciframos trama
118     getNMEA2() ;

```

```

119
120 //Enviamos mensaje de texto con coordenadas
121 if (tiempo==60){
122     Serial2.println("enviar SMS");
123
124     if (enviarSMSFlag==1){
125         enviarMensaje(ggleMapsMsg);
126     }
127     tiempo=0;
128 }
129
130 //chequeamos el buffer del GSM
131 check();
132
133 //Encendido del modulo GSM
134 if (digitalRead(BUT1)==LOW){
135     digitalWrite(LEDA,HIGH);
136     digitalWrite(PWRKEY,LOW);
137 } else {
138     digitalWrite(LEDA,LOW);
139     digitalWrite(PWRKEY,HIGH);
140 }
141
142 //Activación del envío de sms
143 if (digitalRead(BUT2)==0){
144     if (enviarSMSFlag==0){
145         enviarSMSFlag=1;
146         digitalWrite(LEDV,HIGH);
147         delay(1000);
148     } else {
149         enviarSMSFlag=0;
150         digitalWrite(LEDV,LOW);
151         delay(1000);
152     }
153 }
154
155 }
156
157
158
159 void getNMEA1 (void){
160
161     unsigned int flag=0;
162     char c='0';
163     int i=0;
164
165     if (Serial3.available()>0){           //Observo si hay algo en el buffer y lo comparo
166         para ver si es el caracter '$'
167
168         c=Serial3.read();
169
170         if (c=='$'){
171             flag=1;
172             NMEA[i]=c;
173             i++;
174         }
175
176     while (flag==1){

```

```

177     if ( Serial3 . available () >0){
178         c=Serial3 . read () ;
179         NMEA[ i]=c ;
180         i++;
181     }
182 }
183
184 if ( c=='\r '){
185
186     if ( strncmp (NMEA, "$GPGGA" , 6 )==0){
187         memset (TRAMA, 0 , 100 ) ;
188         strcpy (TRAMA, NMEA) ;
189         // Serial2 . println (TRAMA) ;
190         descifrarTrama () ;
191         tiempo++ ;
192     }
193
194     memset (NMEA, 0 , 100 ) ;
195     flag=0 ;
196 }
197
198 }
199
200 }
201
202 void getNMEA2 ( void ){
203
204     if ( Serial3 . available () >0){
205         c=Serial3 . read () ;
206         if ( c=='$'&& flag==0){
207             flag=1 ;
208         }
209
210         if ( flag==1){
211
212             NMEA[ i]=c ;
213             i++;
214
215         }
216
217         if ( c=='\r '){
218             NMEA[ i]=c ;
219
220             if ( strncmp (NMEA, "$GPGGA" , 6 )==0){
221                 memset (TRAMA, 0 , 100 ) ;
222                 strcpy (TRAMA, NMEA) ;
223                 // Serial2 . println (TRAMA) ;
224                 descifrarTrama () ;
225                 tiempo++ ;
226             }
227
228             memset (NMEA, 0 , 100 ) ;
229             flag=0 ;
230             i=0 ;
231
232         }
233
234     }
235

```

```

236
237    }
238
239 void descifrarTrama( void ){
240
241     sscanf(TRAMA,"%f,%f,%c,%f,%c,%i,%i,%f,%f,%c",& hora,& lat ,& i lat ,& lon ,& ilon ,& fix ,&
242             sat ,& HDOP,& alt ,& unit );
243     char respuesta [200];
244     memset( respuesta ,0 ,200 );
245     sprintf( respuesta ,"%f,%f,%c,%f,%c,%i,%i,%f,%f,%c\n",hora ,lat , ilat ,lon ,ilon ,fix ,sat ,HDOP
246             , alt , unit );
247     //Serial.print( respuesta ); //Debug de lo que se escribe en la sd
248     writeFileSD( respuesta );
249
250     //Mensaje para localizar el dispositivo
251     double latitud , longitud , latpartint , lonpartint , latpartdec , lonpartdec ;
252     latitud = (double) lat ; //Latitud formato google
253     latitud=latitud /100;
254     latpartdec=modf( latitud ,& latpartint );
255     latpartdec=latpartdec *100/60;
256     latitud=latpartint+latpartdec ;
257
258     longitud = (double) lon ; //Longitud formato google
259     longitud=longitud /100;
260     lonpartdec=modf( longitud ,& lonpartint );
261     lonpartdec=lonpartdec *100/60;
262     longitud=lonpartint+lonpartdec ;
263     memset( ggleMapsMesg ,0 ,200 );
264     sprintf( ggleMapsMesg , " http://maps.google.com/maps?q=% .6f %c ,% .6f %c \n\r ",latitud , ilat ,
265             longitud , ilon );
266     //writeFileSD( ggleMapsMesg ); //Debug
267
268 }
269
270 void writeFileSD( char *mens ){
271     myFile = SD.open(" log .txt " , FILE_WRITE );
272     myFile . write( mens );
273     myFile . close ();
274 }
275
276 void check ( void ){
277
278     while ( Serial . available ()>0){
279         Serial2 . write( Serial . read () );
280     }
281
282     while ( Serial2 . available ()>0){
283         Serial . write( Serial2 . read () );
284     }
285 }
286
287 void enviarMensaje( char *mensaje ){
288
289     Serial . println( "AT+CMGF=1" );
290     check ();
291     delay (1000 );
292     Serial . println( "AT+CMGS=\\" 654123789 \\ " );
293     check ();

```

```
292     delay(1000);  
293     Serial.println(mensaje);  
294     check();  
295     delay(1000);  
296     Serial.write(0x1A);  
297     check();  
298     delay(1000);  
299 }
```

## Apéndice F: Listas de materiales.

## Placa de Alimentación

Componente	Descripción	Designador	Cantidad	Precio Unidad (€)	Precio total(€)
1uF	Capacitor	C1	1	0,03	0,03
100nF	Capacitor	C2, C3, C5	3	0,03	0,09
25uF	Capacitor	C4, C6	2	0,03	0,06
Diode 1N4007	1 Amp General Purpose Rectifier	D1, D2, D3, D4	4	0,2	0,8
Header 20X2	Header, 20-Pin, Dual row	P1, P2, P4, P5	4	1	4
Header 2	Header, 2-Pin	P3	1	0,5	0,5
240	Resistor	R1, R3	2	0,03	0,06
Rvar	R variable 1K	R2, R4	2	0,5	1
LM317T	1.2V to 37V Voltage Regulator	U1, U3, U4	3	0,57	1,71
LM7805	Precision 1A Regulator	U2	1	0,5	0,5
					Total (€) 8,75

**Placa de desarrollo**

Componente	Descripción	Designador	Cantidad	Precio Unidad (€)	Precio total(€)
100nF	Capacitor SMD 0603	C1, C2, C3, C4, C5, C6, C7, C8, C9,C20	9	0,01	0,09
10uF	Capacitor SMD 0603	C12, C21	2	0,3	0,6
4.7uF	Capacitor SMD 0603	C13	1	0,027	0,027
20pF	Capacitor SMD 0603	C14, C19	2	0,009	0,018
11pF	Capacitor SMD 0603	C15, C18	2	0,003	0,006
6.8uF	Capacitor SMD 0603	C16	1	0,23	0,23
1uF	Capacitor SMD 0603	C17	1	0,01	0,01
LED3	Typical BLUE SiC LED SMD 0603	D3, D4, D5	3	0,29	0,87
Microsd	Ranura MicroSD	Microsd	1	2,59	2,59
Header 10X1		P1, P2, P4, P5	8	0,5	4
ICSP	Header, 6-Pin	P3	1	0,6	0,6
40R	Resistor SMD 0603	R2, R3, R5	3	0,003	0,009
4.7K	Resistor SMD 0603	R4, R7, R16, R17, R18	5	0,006	0,03
10k	Resistor SMD 0603	R6, R10, R11, R12, R13, R14, R15	7	0,003	0,021
100K	Resistor SMD 0603	R8	1	0,003	0,003
SW-PB	Switch	S1, S2, S3, S4	4	0,5	2
PIC32MX795F512LT-80I/PT	Microcontrolador	U1	1	10,61	10,61
MCP1253-33X50I/MS	Low Noise, Positive-Regulated Charge Pump, 1MHz	U3	1	1,39	1,39
USB	Conector USB	USB1	1	1,64	1,64
8 MHz	Crystal Oscillator	Y1	1	1,11	1,11
32KHz	Crystal Oscillator	Y2	1	0,7	0,7
					Total 26,554

## Placa adaptadora GPS

Componente	Componente	Componente	Componente	Componente	Componente
Header 10X1		P1, P2, P4, P5	8	0,5	4
header 10	Arduino header 10		1	0,5	0,5
header 8	Arduino header 8		2	0,5	1
header 6	Arduino header 6		1	0,5	0,5
			Total	6	

### Placa de GSM

Componente	Componente	Componente	Componente	Componente	Componente
220nF	Capacidad SMD 0603	C1	1	0,04	0,04
22pF	Capacidad SMD 0603	C2	1	0,009	0,009
LED3	LED SMD 0603	D1, D3	2	0,932	1,864
Antenna	Generic Antenna connector	E1	1	0,5	0,5
Header 10X1	Header, 20-Pin, Dual row	P1, P2, P4, P5	8	0,5	4
BC847B	Transistor NPN	Q1, Q2	2	0,161	0,322
1K	Resistor	R1, R2, R3	3	0,053	0,159
300	Resistor	R4, R10	2	0,0319	0,0638
4.7K	Resistor	R5, R11	2	0,006	0,012
47K	Resistor	R6, R12	2	0,005	0,01
22	Resistor	R7, R8, R9	3	0,01	0,03
sim	Zócalo SIM	u2	1	2,39	2,39
ESDA6V1-5W6	Dispositivo de protección ESD	u3	1	0,512	0,512
SIM900	Módulo GSM	U1	1	13,51	13,51
Antena	Antena GSM		1	7,29	7,29
				Total	30,7118