Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Desarrollo de un sistema sensorial de la calidad del aire con Arduino

Autor: Víctor Manuel García Melgar

Tutor/es: Philip Siegmann

2015

NOMBRE GARCIA MELGAR VICTOR MANUEL - NIF 10823395D

Firmado digitalmente por NOMBRE GARCIA MELGAR VICTOR MANUEL - NIF 10823395D
Nombre de reconocimiento (DN): c=ES, o=FNMT, ou=FNMT Clase 2 CA, ou=500730011, cn=NOMBRE GARCIA MELGAR VICTOR MANUEL - NIF 10823395D
Fecha: 2015.06.01 23:18:12 +02'00'

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Desarrollo de un sistema sensorial de la calidad del aire con Arduino.

Autor:Víctor Manuel García Melgar

Director: Philip Siegmann

TRIBUNAL:

Presidente: Francisco Javier Acevedo Rodríguez

Vocal 1º: Sancho Salcedo Sanz

Vocal 2º: Philip Siegmann

FECHA:.....

AGRADECIMIENTOS

A mi familia, con especial mención a José Luis Estébanez y su hermano Toño por su ayuda en mi primer proyecto hace 32 años

Tabla de Contenidos

Índice de ilustraciones

Índice de tablas

Índice de Diagramas

1 Resumen. / Abstract

Resumen. En el presente trabajo de fin de grado se ha desarrollado un sistema portátil que permite comunicar un sensor de nanopartículas con un microprocesador y simultáneamente con un GPS y un sistema de almacenamiento de datos. Con ello se pueden obtener medidas en tiempo real (cada 10 segundos) de la concentración de las nanopartículas en el aire y la localización de dicha medida así como almacenar estos datos en una tarjeta SD.
Palabras clave: Contaminación del aire por nanopartículas, medidas en tiempo real, medidas sobre el terreno, hidrocarburos aromáticos policíclicos adsorbidos por nanopartículas, microprocesador Arduino.

Abstract. In this project has been developed a portable system that allows communicating a nanoparticle sensor with a microprocessor and simultaneously with a GPS and a data storage system. This system can acquire both, the nanoparticle concentration and the GPS coordinates from where the measurements was taken, and store these data on a SD card.

Keywords: Nanoparticulate air pollution, real-tyme measurements, on-road measurements, Particle-Bound Polycyclic Aromatic Hydrocarbon,Microprocessor Arduino.

2 Resumen Extendido

Para la realización de este proyecto fue suministrado un equipo PAS2000CE que va provisto de un sensor que mide la concentración de un determinado tipo de nanopartículas que en su superficie tiene adheridos hidrocarburos aromáticos policíclicos (HAP).

- En una primera fase, y ante la ausencia de manuales que lo describieran, se buscó documentación que indicara su funcionamiento y características. Así mismo se hizo un

estudio de las características del microprocesador Arduino. De la documentación obtenida se deducía que dicho sensor debería tener una comunicación analógica y se esbozó un diseño en el que la comunicación entre el Arduino y el sensor fuera por medio de una de sus puertas analógicas. En dicho esbozo se incluyeron las demás partes del proyecto: la tarjeta de almacenamiento que se comunicaría con el arduino vía SPI (Serie Peripheral Interface) y el receptor GPS, que se conectaría con el Arduino con un protocolo de comunicación serie. Y en función de esa suposición, se midieron las características del cable, el cual se describe en la Figura 2., y se midieron los niveles de tensión por medio de un polímetro, viendo que estos eran compatibles con las especificaciones del Arduino. Se intentó conectar una de las salidas del sensor con el microprocesador Arduino siendo éste incapaz de capturar la señal. Fue necesario utilizar un osciloscopio y constatamos (véase Figura 3) que las características de la señal no se correspondían con la documentación que obtuvimos. Se contactó con la casa matriz la cual no disponía de la información requerida debido a su antigüedad. También se contactó con el profesor que diseñó dicho equipo, Dr. Heinz Burtscher, el cual nos indicó que el equipo no tenía salida analógica. Finalmente el Profesor Tutor obtuvo la información de una documentación antigua la cual especificaba que la señal de salida correspondía a una señal tipo RS232.

- Una vez constatado que las comunicaciones entre el sensor y el microprocesador no eran analógicas se modificó el diseño inicial del proyecto y se procedió a adaptar la señal del sensor a los niveles TTL del microprocesador por medio de un conversor que se muestra en la Figura 8 y sus características en el Diagrama 2. Se realizó un pequeño programa utilizando una librería SoftSerial, que permite las comunicaciones serie entre los puertos del microprocesador y el sensor, para capturar la señal y ver sus características. El microprocesador Arduino, cuando detecta que hay una transmisión por parte del sensor, empieza a almacenar byte a byte dicha información hasta que esta finaliza. Utilizamos el programa RealTerm para capturar y almacenar las comunicaciones que transmitía el sensor, dado que el monitor serie de Arduino permite visualizar dichas comunicaciones pero no

almacenarlas. Se observó que el sensor HAP transmite ráfagas de bytes (véase Figura 3) que oscilan entre 4 y 60 bytes cada 10 segundos aproximadamente. Cuando la secuencia tiene mas de 58 bytes es cuando se transmite la medida capturada por el sensor.

- De forma separada de lo anteriormente realizado se desarrolló un programa para el almacenamiento en una tarjeta SD, (Figura 7, Diagrama 4), utilizando las librerías SD y SPI que se describen en el punto 5.3.3. Se probó el circuito para verificar su correcto funcionamiento. Utilizamos una tarjeta SD de 1 gigabyte, ya que los campos a almacenar serían de 90 caracteres por medida lo cual permitiría casi dos años de medidas continuas.

- También, de forma separada se desarrolló un programa que procesara la señal del receptor GPS.(Figura 6, Diagrama 3). Para procesar la señal debíamos primeramente, comunicarnos desde el Arduino vía serie, con el receptor usando la librería SoftSerial (véase 5.3.1) y decodificar dicha información. Para decodificarla, había diferentes librerías NMEA,TINYGPS, TINYGPS++ así como otros tipos de programas que no incluían librerías. Nos decantamos por la librería TINYGPS++ al ser esta más intuitiva para la programación y porque además procesaba cualquier formato de secuencia del receptor y ocupaba un espacio similar de memoria.

- Probadas todas las partes por separado se procedió a la integración de todas ellas (véase Figura 9). Se desarrolló un programa que integrara todas las funcionalidades que habían sido previamente probadas. Se observó que la librería SoftSerial cuando escuchaba el puerto de comunicación con el receptor GPS y debía conmutar al puerto de comunicación del sensor se perdían datos, por lo que hubo que dedicar una librería solo para las comunicaciones con el sensor (SoftSerial) y otra para el receptor GPS(AltSoftSerial, véase 5.3.2). Se rediseñó el software y se hicieron las capturas de datos y pruebas durante cortos espacios de tiempo verificando por medio del programa RealTerm que la captura de los datos del sensor y del receptor GPS eran almacenados en la tarjeta de almacenamiento.

- Se iniciaron medidas (véase punto 6) en el exterior con periodos de más de una hora para observar la consistencia del diseño. Después de un análisis pormenorizado se observó que de forma aleatoria no se capturaban algunas medidas procedentes del sensor. Se estudiaron de nuevo las características del receptor GPS así como la librería utilizada TINYGPS++. Se comprobó que en algunos casos dicha librería no capturaba la información y no permitía que la información del sensor se almacenara. Como consecuencia de esta comprobación se hicieron pruebas con otras librerías. Con la librería TINYGPS hicimos pruebas durante largos periodos de tiempo y constatamos que no impedía la captura de los datos del sensor en ningún caso.

3 Introducción

Las medidas de contaminantes en el aire en las ciudades se suelen realizar en estaciones de medidas fijas distribuidas por la misma [1]. Uno de los principales contaminantes en las ciudades son los vehículos de combustión. A pesar de que actualmente los vehículos de motor están sometidos a estrictos requisitos técnicos que limitan las emisiones NOX, COX, hidrocarburos (HC) y Materia Particulada (MP) [2], aún existen algunos que contaminan considerablemente más que el resto [3, 4]. Así mismo hay carreteras por las que las emisiones aumentan drásticamente como es en el caso de cuestas pronunciadas y vías de aceleración. La realización de medidas en tiempo real a lo largo de las carreteras permitiría hacer una búsqueda de aquellos lugares más contaminados y vehículos más contaminantes de la ciudad. En el presente proyecto se propone desarrollar un equipo portátil de medida en tiempo real de contaminantes (en este caso de partículas ultrafinas) que a su vez proporcione la información de la localización de la medida realizada en cada instante.

La medida de la polución de partículas ultrafinas, o partículas de tamaño nanométrico, es de especial interés ya que su toxicidad es mayor a las de las partículas de mayor tamaño. Cuando son inhaladas, al ser tan pequeñas son capaces de penetrar en el cuerpo humano hasta los alvéolos pulmonares. A través de los alvéolos pueden pasar al torrente sanguíneo y desde allí distribuirse por

el resto del cuerpo [5]. Por otro lado, la relación superficie/volumen de las nanopartículas es muy grande lo que las hace mucho más reactivas que las de mayor tamaño. Adsorbidos sobre la superficie pueden transportar otras sustancias tóxicas, como hidrocarburos aromáticos policíclicos (HAP), que es una conocida sustancia cancerígena [6]. Esto hace que la medida de las nanoparttículas que han adsorbido HAP sean de especial interés, y es por ello que utilizamos en este trabajo un sensor que permite medir este tipo de contaminantes.

El desarrollo de este sistema se expone en el presente trabajo. En el punto 4 se describen los equipos que se han utilizado. En el punto 5 se describen de forma minuciosa el funcionamiento de los equipos utilizados. En el punto 6 se muestran las medidas y los resultados obtenidos así como las diferentes comprobaciones de los resultados. En el punto 7 se muestra un breve resumen del coste de los materiales utilizados en este Trabajo Fin de Grado (TFG). En el punto 8 se describen las limitaciones y futuros trabajos. En el punto 9 se describen las principales conclusiones alcanzadas en este TFG. En los puntos 10, 11y 12 se describen las Referencias, el Apéndice 1: Código del programa, donde se muestra el código utilizado y el Apéndice 2:Diagramas de los equipos utilizados respectivamente.

4 Descripción de los equipos

A continuación se describen detalladamente cada uno de los equipos de los que se componen el sistema que se va a desarrollar. Estos dispositivos son: El sensor de nanopartículas, un microprocesador Arduino, un lector de tarjetas SD, un GPS y un adaptador de las señales RS232C a TTL. El sensor de nanopartículas detecta solo aquellas que han adsorbido en su superficie HAP, y envía la medida cada 10 segundos a un microprocesador Arduino a través de un adaptador RS2323C a TTL. El Arduino toma simultáneamente información de hora, día, latitud, longitud de un GPS y la envía a una Tarjeta SD para su posterior almacenamiento.

4.1 Sensor de nanopartículas

Como medidor de contaminantes se utiliza un equipo que mide la concentración de partículas ultrafinas o nanopartículas en suspensión en el aire (de tamaño entre 30 y 300 nm). En particular, este sensor detecta un tipo especial de nanopartículas que han adsorbido Hidrocarburos Aromáticos Policíclicos (HAP) en su superficie (HAPP: HAP adsorbidos sobre partícula). La medida de este sensor de nanopartículas está calibrado para dar los valores en nanogramos de HAP adsorbidos sobre nanopartículas por m^3 de aire. Este sensor modelo PAS2000CE está fabricado por Ecochem y se muestra en la Figura 1, y es además portátil y permite realizar medidas a intervalos de 10 segundos [7].

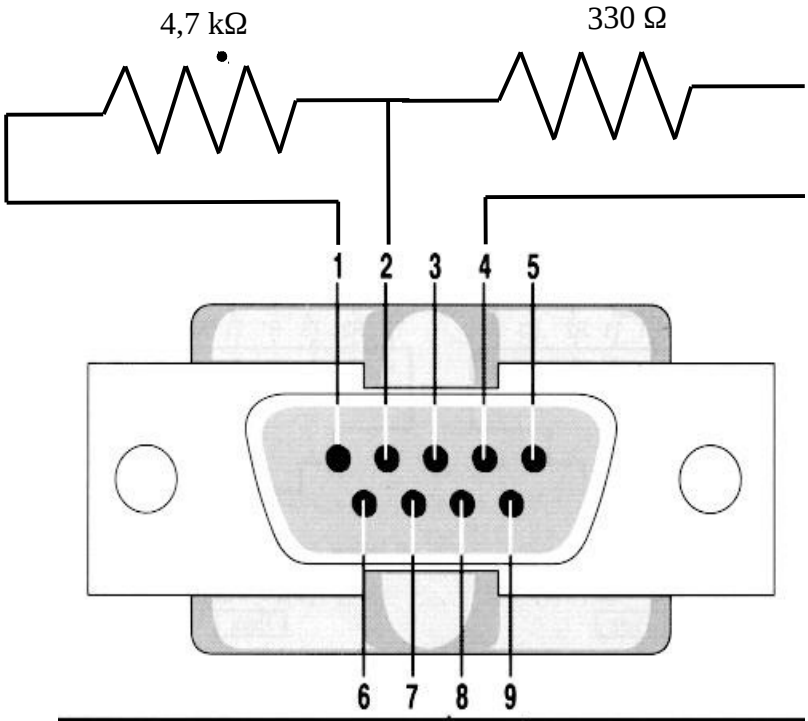
(a) (b)

Figura 1: Imagen del sensor PAS2000CE. a) Imagen frontal, b) Imagen de la parte posterior.

El sensor PAS2000CE del que disponemos es bastante antiguo y no existe una documentación actualizada por lo que se tuvieron que hacer diferentes pruebas para obtener una señal interpretable. El equipo venía con un cable con una salida RS232 que se muestra de forma esquemática en la Figura 2, y tenía entre los pines 2 y 3 un resistor de 4,7 KΩ y entre el pin 3 y el cable de 3,5 mm un resistor de 330Ω.

Cuando se observan las secuencias de la señal se constató que la comunicación (véase Figura 3) consistía en cuatro ráfagas cortas de tres bytes y después otras tres ráfagas de diferente

longitud. Así mismo la comunicación era de 9600 baudios, de 8 bits, un bit de stop y no paridad. Entre las múltiples pruebas realizadas se conectó el pin tres a una entrada digital del Microprocesador Arduino y llevando el 2 y el 5 a tierra. Los datos obtenidos no tenían consistencia por lo que finalmente se conectó un adaptador de RS232 a niveles TTL. Las pruebas se realizaron con los tiempos de salida cada 10 segundos. Los equipos modernos vienen con una salida analógica que permite la captura de la señal de una forma más eficiente, al no tenerla, nuestro equipo de prueba ha añadido un mayor grado de complejidad.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figura 2: Salida conexión RS232

El principio de medida del sensor PAS2000CE se basa en la carga fotoeléctrica (PC: "Photoelectric Charge") de las partículas. También se llama sensor de aerosoles por fotoemisión (PAS, "Photoemission Aerosol Sensor") porque las partículas se irradian con luz UV, produciendo la emisión de un fotoelectrón [8, 9]. Se averiguó que las partículas que han adsorbido HAP tienen una capacidad de carga fotoeléctrica muy alta y por tanto, pueden ser más fácilmente medidos. El origen de HAP y de las partículas que han adsorbido HAP, es la combustión incompleta de material orgánico, como la de los combustibles fósiles en los motores de los vehículos, especialmente los de gasoil.

El sensor de aerosoles por fotoemisión se basa en la fotoionización de las partículas. Resulta que si se irradian con radiación ultravioleta (ver Figura 4) se ionizan sólo aquellas que han adsorbido HAP (al incrementase el rendimiento fotoeléctrico (F) en un factor 2000). De esta forma la partícula ionizada se puede acelerar hacia un filtro que actúa de ánodo produciéndose una corriente (i_{PC}).

No obstante, en este proceso de movimiento hasta el ánodo, la partícula positiva puede volver a chocar con los fotoelectrones, pero esto resulta ser mucho más probable para partículas de mayor tamaño (a presión ambiental), por lo que solo las partículas ionizadas ultrafinas, con una sección activa (S_{active}) del orden de nanómetros, son las que finalmente llegan a un filtro que actúa de ánodo produciendo una corriente (i_{PC}) medible.

La señal que finalmente da el sensor se obtiene de una calibración que relaciona el valor de la corriente i_{PC} en nanogramos de HAP adsorbidos sobre nanopartículas por metro cúbico de aire que circula a través del filtro (ng/m^3).

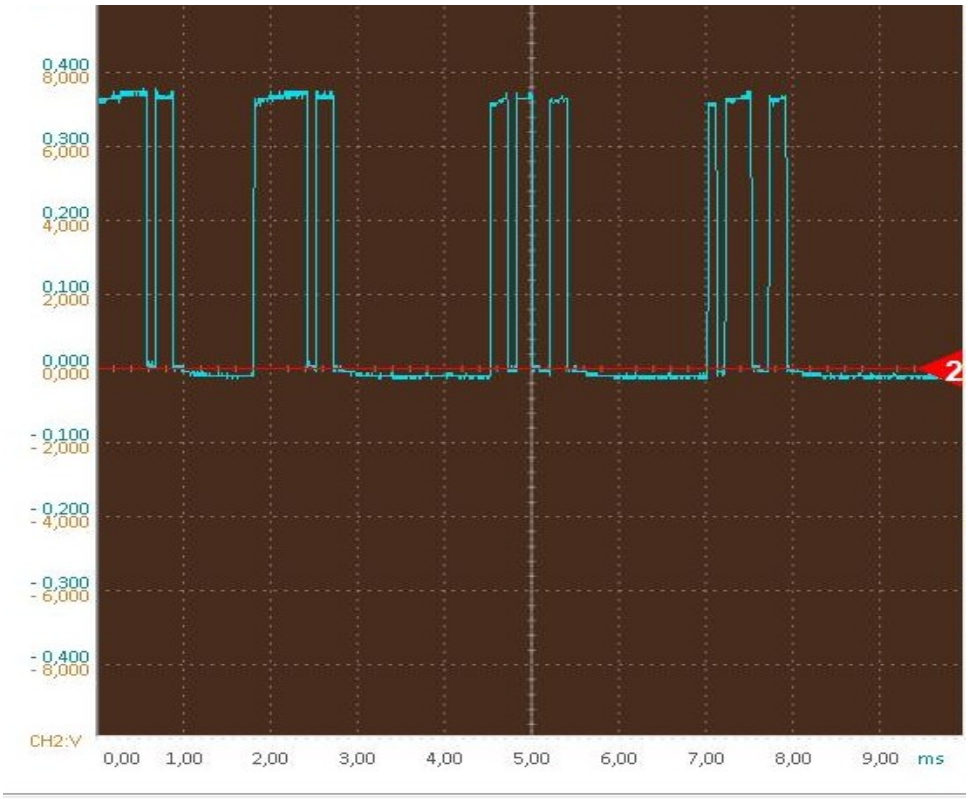


Figura 3: Ejemplo de la señal de salida del sensor de partículas medida con un osciloscopio.

4.2 Microprocesador Arduino

Para obtener los datos de todos los equipos periféricos se hace uso de un microprocesador Arduino Uno que se muestra en la Figura 5. De las características del Microprocesador Arduino existe una amplia documentación y bibliografía, no obstante debe remarcarse la interface periférica serie SPI que puede verse como un registro síncrono de 16 bits que permite comunicarse con la tarjeta SD para el almacenamiento de la Información [10].

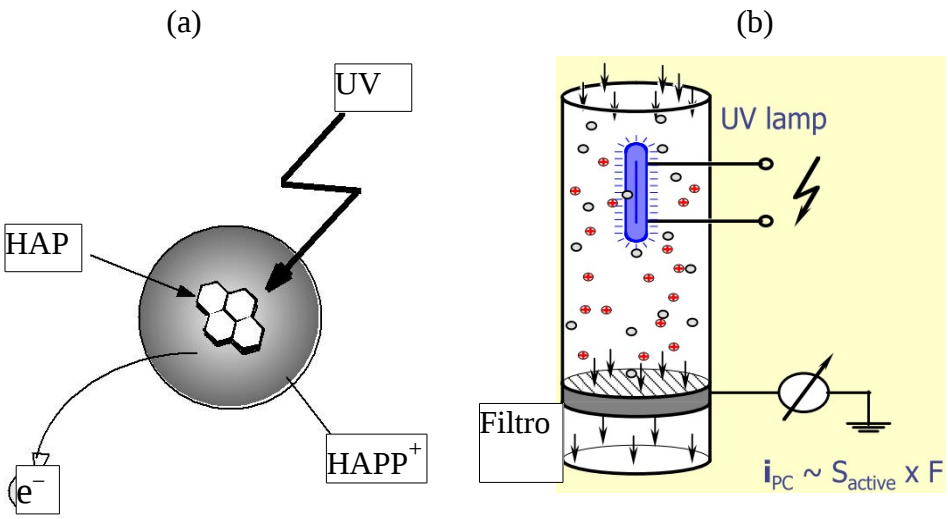


Figura 4: a) Esquema de fotoionización de HAPP. b) Esquema de funcionamiento de medida de los HAPP.

Una de las limitaciones encontradas fue que las librerías para comunicar el Arduino con el equipo PAS2000CE y el GPS no permitían utilizarlas simultáneamente. Por lo que a la librería SoftwareSerial hubo que añadir una nueva librería AltSoftSerial que emula un puerto serie adicional, que le permite comunicarse con otro dispositivo serie cuasi simultáneamente.

4.3 Receptor GPS

El equipo receptor que se va a utilizar es el Venus638FLPx-L y se muestra en la Figura 6. Este equipo permite una alimentación de 3,3 voltios desde el microprocesador Arduino. De este equipo utilizando el protocolo NMEA (Nacional Marine Electronics Association) y utilizaremos la sentencias que permitan obtener la latitud, la longitud, fecha y hora del equipo.

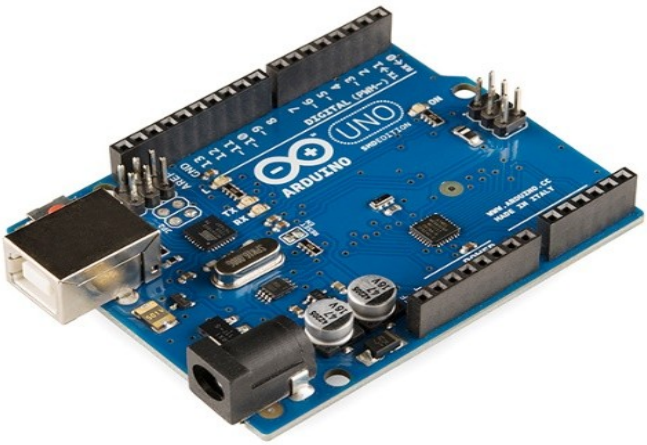


Figura 5: Microprocesador Arduino Uno

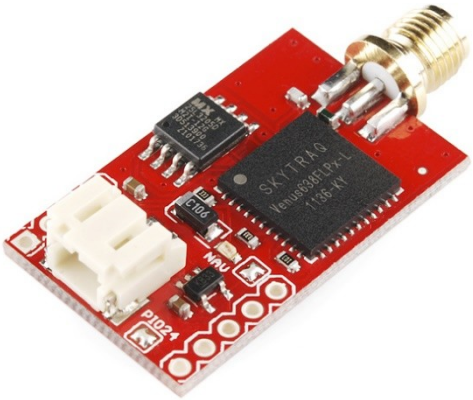


Figura 6: Equipo receptor GPS

4.4 Tarjeta almacenamiento SD

Una de las limitaciones que tiene la tarjeta que utilizamos y que se muestra en la Figura 7, es que no puede almacenar una tarjeta mayor de 16Gbytes. Para nuestro trabajo se ha utilizado una tarjeta de 1Gbyte. Según los cálculos realizados nos permitiría almacenar información durante al menos 2 años de una forma continuada.

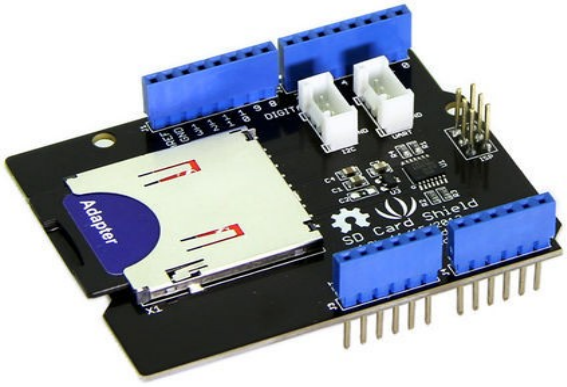


Figura 7: Tarjeta de almacenamiento SD

4.5 Conversor RS232 a TTL

Como comentamos anteriormente el equipo de medida PAS2000CE no tenía una salida normalizada, por lo que fue necesario utilizar dicho conversor (ver Figura 8). Cuando la tensión RS232 es positiva es decir un “0” la tensión de salida es 0 y viceversa.

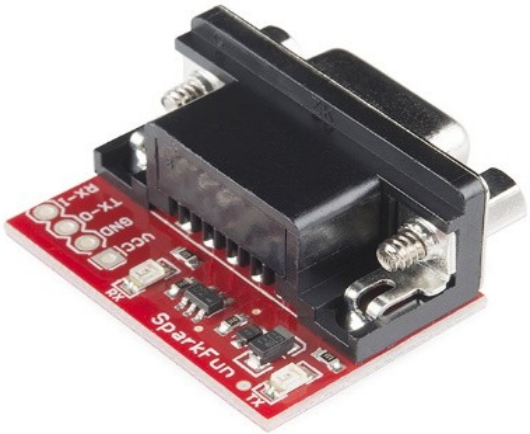


Figura 8: Conversor RS232 a TTL

5 Desarrollo del equipo

5.1 Diagrama del equipo

En la Figura 9 que se muestra a continuación se muestran las conexiones del equipo. Debe comentarse que el circuito donde va montado la tarjeta SD es conectado en la parte superior del equipo Arduino aunque el diagrama se ponga al lado del mismo para una mejor compresión del equipo.

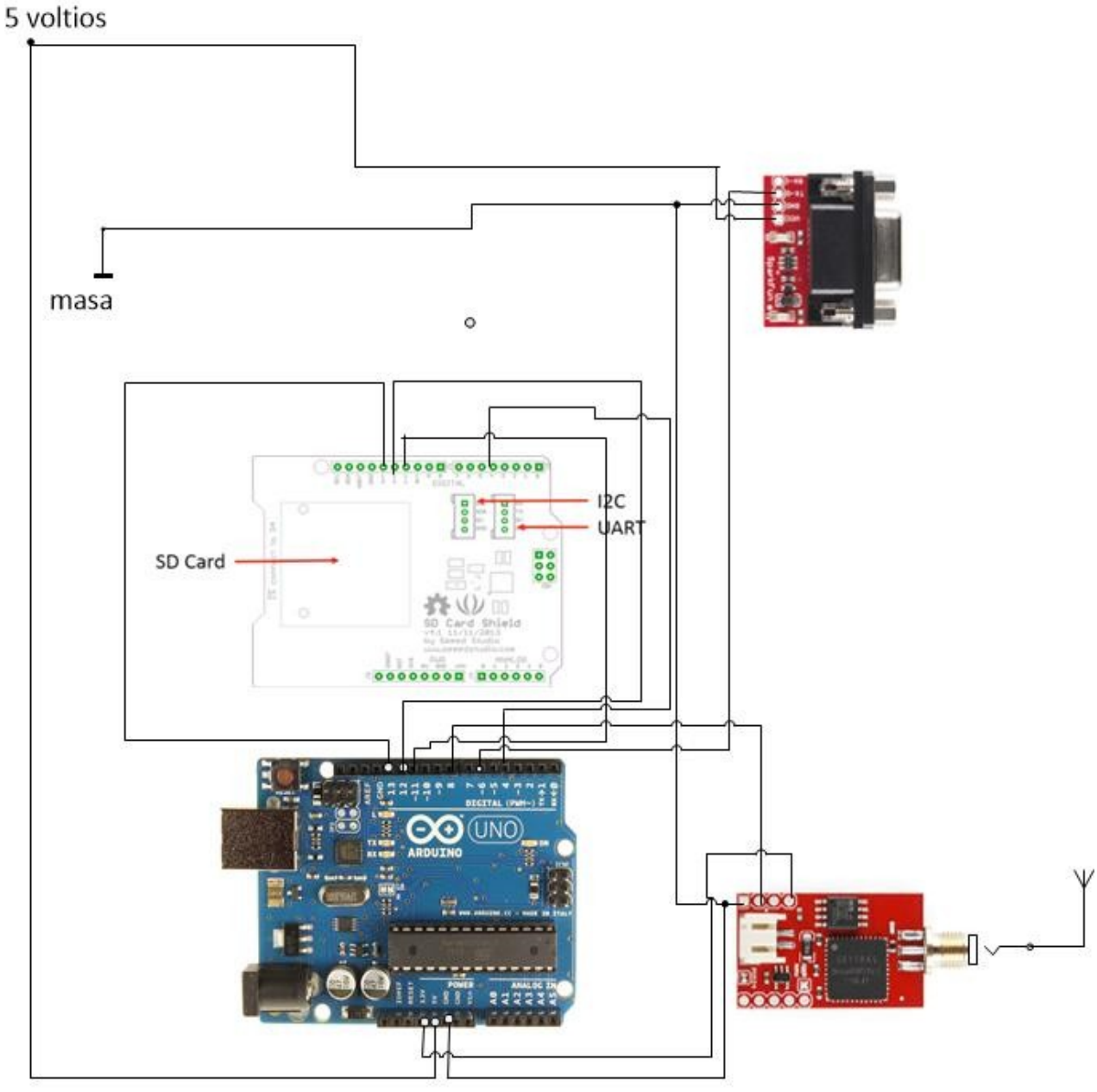


Diagrama de conexión

Figura 9

5.2 Comunicación serie con el microcontrolador Arduino.

En las comunicaciones serie, la transmisión y recepción de un dispositivo deben sincronizarse utilizando un protocolo común y la misma velocidad de transmisión/recepción. La sincronización permite que tanto el transmisor y el receptor estén esperando los datos de transmisión / recepción al mismo tiempo. Hay dos métodos básicos para el mantenimiento de la "sincronización" entre el transmisor y el receptor: asíncrono y síncrono. En un sistema asíncrono la comunicación serie, como USART del ATmega328, los bits trama se utilizan al principio y al final de un byte de datos. Estos bits de entramado alertan al receptor que un byte de datos entrante ha llegado y también señala la finalización de la recepción. La velocidad de transmisión de datos para un sistema de serie asíncrono es típicamente mucho más lento que el sistema síncrono, pero sólo requiere un único cable entre el transmisor y el receptor. Un sistema de comunicación serie síncrona mantiene la "sincronización" entre el transmisor y receptor mediante el empleo de un reloj común entre los dos dispositivos. Los bits de datos son enviados y recibidos en la subida o bajada del pulso del reloj. Esto permite tasas de transferencia de datos más alta que con las técnicas asíncronas pero requiere de dos líneas, datos y reloj, para conectar el receptor y el transmisor.

5.2.1 Velocidad de Transmisión

Las tasas de transmisión de datos se especifican típicamente como baudios o bits por segundo, en nuestro diseño hemos optado por una velocidad standard de 9600 baudios ya que ésta nos venía fijada por el equipo de medida PAS200CE y optamos por usar la misma también para la comunicación con el GPS así como con la comunicación con el PC.

5.2.2 Formato código

No retorno a cero (NRZ): Hay muchas normas de codificación diferentes utilizadas dentro de las comunicaciones en serie. Es menester que el transmisor y el receptor deban utilizar un estándar común de codificación por lo que los datos pueden interpretarse correctamente en el extremo receptor. El Atmel ATmega328 utiliza un no retorno a (NRZ) estándar de codificación de cero. En la codificación NRZ, un uno lógico, se señaliza por un valor lógico alto durante todo el intervalo de tiempo asignado para un único bit; mientras que un cero lógico se señaliza por un valor lógico bajo durante todo el intervalo de tiempo asignado para un único bit.

5.2.3 Protocolo de comunicación RS-232C

Cuando se produce la transmisión en serie a larga distancia son requeridas a veces técnicas que aseguren la integridad de los datos. En distancias largas niveles lógicos se degradan y pueden ser atenuadas/distorsionadas por el ruido. En el extremo receptor, es difícil discernir un valor lógico alto de uno bajo. El RS-232 estándar (EIA-232), un uno lógico se representa con un nivel de -12 VCC mientras que un cero lógico está representado por un nivel 12 VCC. Los chips comúnmente disponibles (por ejemplo, MAX232) convierten los niveles de salida 5 y 0 V a partir de un transmisor RS-232 y convierten 5 V y 0 V niveles a niveles de RS232 en el receptor. El estándar RS-232 especifica también otras características de este protocolo de comunicación.

5.2.4 Paridad

Para mejorar aún más la integridad de los datos durante la transmisión, se pueden utilizar técnicas de paridad. La paridad es un bit adicional (o bits) que puede ser añadido a los bits de datos. El ATmega328 tiene un solo bit de paridad. Con un solo bit de paridad, se puede detectar un solo error de bit. La paridad puede ser par o impar. En

paridad par, el bit de paridad se fija a uno o cero tal que el número de unos en el byte de datos incluyendo el bit de paridad es par. En paridad impar, el bit de paridad se fija a uno o cero tal que el número de unos en el byte de datos, incluyendo el bit de paridad es impar. En el receptor, el número de bits dentro de un byte de datos incluyendo el bit de paridad se cuentan para asegurar que la paridad no ha cambiado, lo que indica un error, durante la transmisión. El equipo PAS2000CE no utiliza el bit de paridad.

5.2.5 ASCII

El Código Estándar Americano para Intercambio de Información o ASCII utiliza 7 bits para la codificación de datos alfanuméricos.

5.3 Software de comunicación serie

5.3.1 Librería SoftwareSerial

El hardware Arduino ha incorporado comunicación serie en los pines 0 y 1 (que también va a la computadora a través de la conexión USB). Esta comunicación serie pasa a través de una pieza de hardware (integrado en el chip) llamado UART. Este hardware permite al chip Atmega recibir la comunicación en serie incluso mientras se trabaja en otras tareas, siempre que haya espacio en la memoria intermedia de serie de 64 bytes. La biblioteca SoftwareSerial ha sido desarrollado para permitir la comunicación en serie en los otros pines digitales del Arduino, utilizando el software para replicar la funcionalidad (de ahí el nombre de "SoftwareSerial"). Es posible tener múltiples puertos serie con velocidades de hasta 115.200 bps. Un parámetro permite la señalización invertida para dispositivos que requieren dicho protocolo. La librería cuenta con la limitación de que si se utiliza software de múltiples puertos serie, sólo uno puede recibir datos a la vez. Aunque puede conmutar a diferentes puertos de forma secuencial, no es capaz de hacerlo a la velocidad que requiere nuestro proyecto. Una vez abierta la comunicación ora está en un puerto ora está en otro. En nuestro caso usaremos dos librerías de comunicaciones para mantener los dos puertos abiertos.

5.3.2 Librería AltSoftSerial

Puede al mismo tiempo transmitir y recibir. Tiene una interferencia mínima con el uso simultáneo de otras librerías. Utiliza un temporizador de 16 bits (y no funcionará con ninguna librería que necesitan ese temporizador) y desactiva algunos pines PWM. Es sensible al uso de otras bibliotecas y puede interrumpirse.

Placa	Pin de Transmisión	Pin de recepción	Pin no utilizable PWM
Teensy 3.0 & 3.1	21	20	22
Teensy 2.0	9	10	(none)
Teensy++ 2.0	25	4	26, 27
Arduino Uno	9	8	10
Arduino Leonardo	5	13	(none)
Arduino Mega	46	48	44, 45
Wiring-S	5	6	4
Sanguino	13	14	12

Tabla 1:Conexiado pines librería Altsoftserial

Como se puede observar en la Tabla 1 los pines 8 y 9 nos vienen ya dados por dicha librería. Los pines PWM "inutilizable" se pueden utilizar normalmente, con digitalRead () o digitalWrite (), pero su función PWM controlado por analogWrite () no funcionará correctamente, porque AltSoftSerial utiliza el temporizador que controla función PWM de ese pin. Los comandos básicos son:

AltSoftSerial mySerial; crea el objeto AltSoftSerial object. Solo un AltSoftSerial puede ser usado con los pines anteriormente mencionados.

mySerial.begin(baud);Inicializa el puerto para comunicar a una determinada velocidad de transmisión.

mySerial.print(anything); Imprime un número o texto. Funciona igual que Serial.print().

mySerial.available(); Retorna el número de bytes recibidos que pueden ser leídos.

mySerial.read();Lee el siguiente byte del Puerto.Si nada ha sido recibido retorna -1

.AltSoftSerial puede soportar aproximadamente 1 bit de tiempo de latencia por interrupción de otras bibliotecas o funciones. La mayoría de las librerías, están diseñadas para minimizar el tiempo de CPU de ejecución interrupciones o acceder a datos con interrupciones deshabilitadas.

Sin embargo, algunas librerías pueden deshabilitar las interrupciones de tiempo superior a 1 bit. Si lo hacen, AltSoftSerial no funcionará correctamente. [12]

5.3.3 Comunicación con la tarjeta SD

El ATmega328 Serie Peripheral Interface o SPI también prevé dos vías de comunicación serie entre un transmisor y un receptor. En el sistema de SPI, el transmisor y el receptor comparten una fuente de reloj común. Esto requiere una línea de reloj adicional entre el transmisor y el receptor pero permite mayores tasas de transmisión de datos en comparación con la USART. El sistema SPI permite un rápido intercambio de datos eficiente entre microcontroladores o dispositivos periféricos.

El SPI [10] puede ser visto como un registro de desplazamiento síncrono de 16 bits con 8 bits permaneciendo en el transmisor y los otros 8 bits residiendo en el receptor como se muestra en la Figura 10

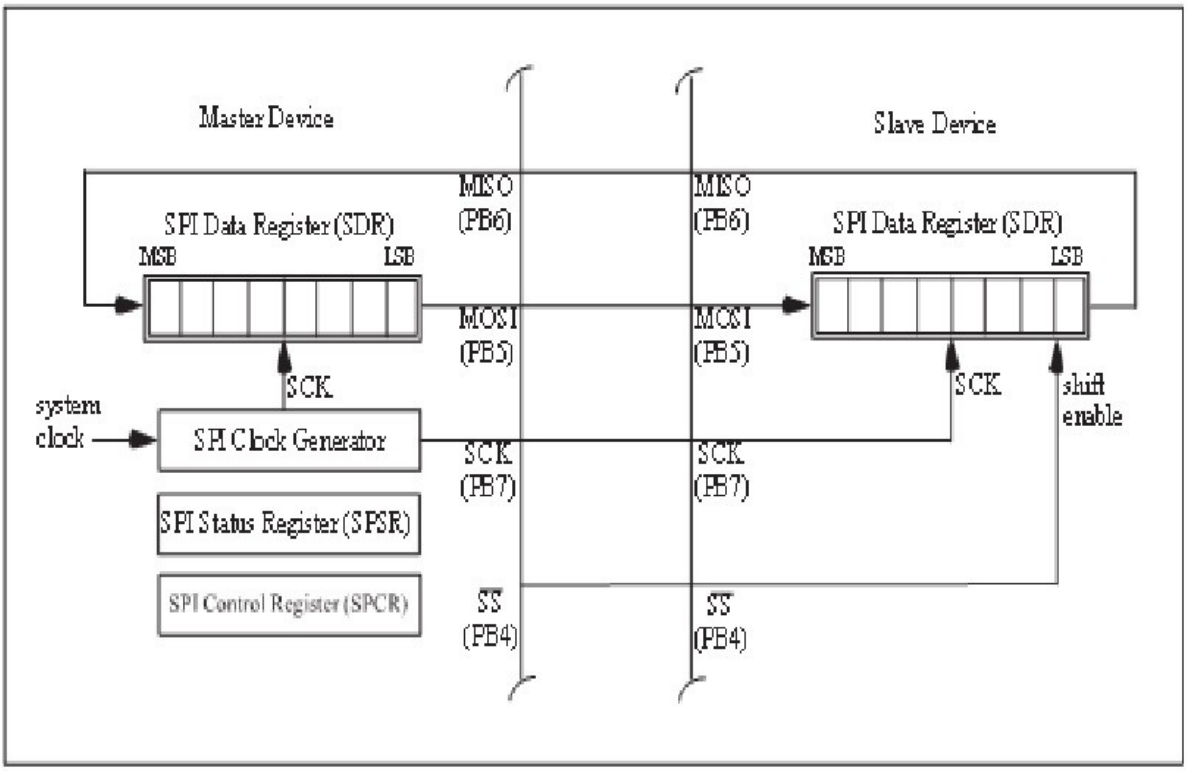


Figura 10: Esquema SPI

El transmisor se designa al maestro ya que está proporcionando la fuente de reloj de sincronización entre el transmisor y el receptor. El receptor se designa como el esclavo. Un esclavo es elegido para la recepción mediante la adopción de su Slave Select (SS) con un nivel bajo. Cuando SS del esclavo tiene un nivel bajo, el desplazamiento en el registro del esclavo está habilitado.

La transmisión SPI se inicia mediante la carga de un byte de datos en el registro maestro configurado como SPI Registro de Datos (SPDR). En ese momento, el generador de reloj SPI proporciona pulsos de reloj para el maestro y también al esclavo a través del pin SCK. Un bit es desplazado desde el maestro al pin del microcontrolador del registro de salida (MOSI) en cada pulso SCK. El dato es recibido en el pin MOSI del esclavo. Al mismo tiempo un bit es desplazado desde el esclavo al maestro (MISO). Después de ocho pulsos de reloj SCK, un byte de datos ha sido intercambiado entre el maestro

y el esclavo. La transmisión entre el maestro y el esclavo es completada cuando el flag SPI de interrupción se active en el maestro y el esclavo. El SPIF flag está direccionado en registro de status SPI (SPSR) de cada dispositivo. A partir de ese momento otro byte puede ser empezado a transmitir.

La tarjeta SD provee de un "disco duro" al microcontrolador Arduino, lo cual le permite una mayor capacidad de almacenamiento. La tarjeta es formateada usando FAT 16 (File Allocation Table).

Antes de escribir en la tarjeta se debe determinar el caudal y fijar la inicialización de la tarjeta SD

5.4 Comunicación GPS

El equipo utilizado Venus638FLPx utiliza el standard NMEA-0183(National Marine Electronics Association la cual es una asociación sin ánimo de lucro de fabricantes, instituciones educativas, etc interesadas en la electrónica marina), con un caudal de 9600 baudios y un formato de salida TTL.

El estándar NMEA 0183 define una interfaz eléctrica y protocolo de datos para las comunicaciones entre equipos marinos, este standard fue diseñado tanto en modo escucha como emisores que utiliza un interfaz asíncrono con caudales mencionados anteriormente y con número de bits de datos: 8 (bit 7 está 0), bits de parada: 1 (o más), no paridad y no handshake.

Dicho equipo envía al microprocesador Arduino una ráfaga de datos, sentencia, en el que va codificada la información. En función del tipo de sentencias enviado, éstas contendrán diferentes datos de los cuales extraeremos los que nos interesan para ese proyecto.

5.4.1 Formato general de las sentencias

Todos los datos transmitidos son en forma de sentencias, solo caracteres imprimibles ASCII son permitidos, además del CR (carriage return) and LF (line feed). Cada

sentencia empieza con un signo de "$" y finaliza con <CR><LF>. Hay tres tipos de sentencias: propietarias, de emisión y de consulta.

Sentencias de emisión: El formato general de la sentencia es: $ttsss,d1,d2,....<CR><LF>

Las dos primeras letras después del símbolo „$" identifican al emisor, los tres siguientes son el identificador de la sentencia (sss) seguidos por campos de datos separados por comas, seguido además por un opcional checksum, y terminado por <CR><LF> . Los campos de datos son inequívocamente por cada tipo de sentencia. Un ejemplo de este tipo es: $HCHDM,238,M<CR><LF>.

Donde "HC" especifica que el emisor es un compás magnético "HDM" indica el rumbo, "238" el valor y "M" define el valor como magnético, la sentencia puede contener hasta 80 caracteres más "$" y CR/LF. Si no son posibles los datos en el campo de datos, se delimitann por comas, sin espacio entre ellas. El campo checksum consiste en "*" y dígitos en hexadecimal que son el resultado de una OR exclusiva de todos los caracteres entre "$" y "*".

Sentencias propietarias: El standard permite a fabricantes individuales definir formatos de sentencias propietarias, dichas sentencias empiezan con "$P", seguido de tres letras que identifican al fabricante seguido además de los datos que desea el fabricante y del formato general de sentencias.

Sentencias de consulta: Una sentencia de este tipo es una petición que emite un determinado tipo de sentencia. El formato es : $ttllQ,sss,[CR][LF]

Los primeros dos caracteres son la identificación del que solicita la consulta y los siguientes dos a quien se le solicita, el quinto carácter es siempre "Q" que define el mensaje como query(consulta). Las siguientes tres letras contienen las letras identificativas de las sentencia que se solicitan. Un ejemplo sería:

$CCGPQ,GGA<CR><LF>

Donde "CC" identifica al ordenador (computer) que pide "GP" (una unidad GPS) la sentencia "GGA" . El GPS transmitirá dicha sentencia una vez por segundo hasta que se solicite una diferente.

5.4.2 Sentencias a utilizar

Hay 19 sentencias diferentes y para este proyecto la librería de software que utilizaremos puede usar aquellas sentencias que permitan obtener los datos que necesitamos para el proyecto. Preferentemente se usará la $GPRMC que indica que pedimos a una unidad de gps el mínimo recomendado para la transmisión de datos (Recommended minimum specific GPS/Transit data). El formato es el que sigue:

$GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,ddmmyy,x.x,a*hh

1 = UTC (tiempo universal coordinado) de la posición
2 = Status de los datos (V=inválido A= válido)
3 = Latitud
4 = N o S (norte o sur)
5 = Longitud
6 = E o W (este u oeste)
7 = velocidad sobre la superficie en nudos
8 = rumbo en grados verdaderos
9 = Fecha
10 = variación magnética en grados (la variación hacia el este debe restarse)
11 = E o W (este u oeste)
12 = Checksum$

No obstante del listado de sentencias que se enumera a continuación la librería de software puede utilizar aquellas sentencias que permitan extraer la información que necesitamos. A continuación se enumeran las que pueden ser utilizadas:

- $ GPGGA-Sistema de Posicionamiento Global Fix Data

- $GPGLL-Posición geográfica, latitud/longitud
- $GPGSA-Satélites GPS y DOP activos
- $GPRMA-Datos específicos mínimos recomendados Loran-C
- $GPRMB-Información mínima de navegación recomendados
- $GPRMC - Recomendado datos Tránsito mínimo específico GPS/
- $ GPZDA - Fecha y hora

5.4.3 Librería TinyGPS

TinyGPS está diseñado para proporcionar la mayor parte de la funcionalidad GPS NMEA, que nos provee de la información requerida para este proyecto; posición, fecha y hora. Para mantener el consumo de recursos bajos, la biblioteca evita cualquier dependencia de punto flotante obligatoria e ignora casi todos los campos de GPS clave. En su momento también se probó la librería TinyGPS++ pero al consumir ésta más recursos, y en las pruebas sucesivas que se realizaron, perdían uno de cada cuatro datos obtenidos. La librería que mejor comportamiento tuvo a la hora de captura de los datos del satélite, fue la que se comenta en el titulo,

Para probar si los datos devueltos son antiguos, se examina el parámetro (opcional) "fix_age" que devuelve el número de milisegundos desde que los datos fueron codificados.

Para este proyecto solo es necesario decodificar un flujo de datos de formato de NEMEA, sentencia.

La librería TinyGPS contiene varios subobjetos, a saber:

- gps.encode (c)

  Cada byte de datos en formato NEMEA son procesados por la librería TinyGPS mediante el uso de encode (). Se devuelve TRUE cuando los nuevos datos han sido totalmente decodificados.

- gps.get_position (latitud,longitud,antigüedad de la señal). Obtiene la posición, donde la latitud y longitud son variables de tipo long,. La variable antigüedad de la señal de tipo unsigned long.
- gps.f_get_position (latitud,longitud, antigüedad de la señal). Obtiene la posición, donde la latitud y longitud son variables de tipo float, y se devuelven los valores reales. Flotador tipos son más fáciles de usar, pero resultan en código más grande y más lento. Dadas las restricciones de capacidad en el microprocesador Arduino, no usaremos este comando.
- gps.stats (char, sentences, failed_checksum). El método stats proporciona una pista si se está consiguiendo buenos datos o no. Proporciona estadísticas que ayudan a solucionar problemas. Chars indica el número de caracteres alimenta al objeto. Sentences- el número de ráfagas $ GPGGA y $ GPRMC válidamente procesados. Failed_checksum - el número de sentencias que no pasarón la prueba de suma de comprobación

5.4.4 Software desarrollado

El diagrama de programación se describe a continuación:

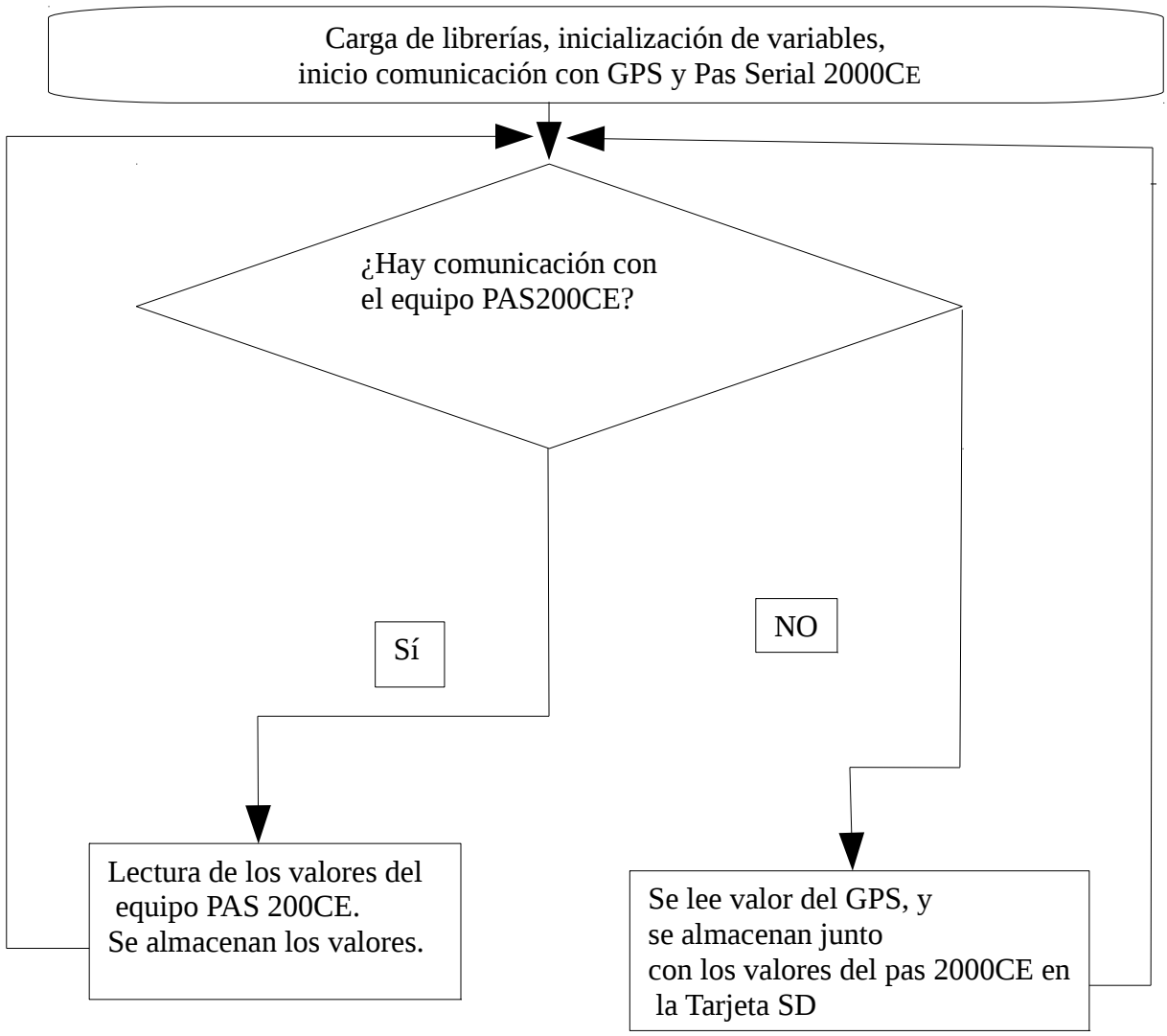


Figura 11:Diagrama de flujo

- Lo primero que haremos será cargar las librerías AltSoftSerial, SD, SPI, TinyGPS y SoftSerial identificando los puertos de salida y entrada.
- Fijamos el caudal de transmisión entre el Arduino y los periféricos. Esta debe ser 9600 baudios, dado que el sensor PAS2000CE viene asi de fábrica y después de varias pruebas nos hemos percatado que no es posible que el microprocesador pueda funcionar a diferentes velocidades con el resto de periféricos.
- Inicializamos la tarjeta SD
- Vemos si hay señal del sensor. Sí no la hay esperamos
- Si la ráfaga tiene más de 58 bytes capturamos el valor de medida de lo contrario es errónea y no la leemos.
- Cuando se termina la ráfaga leemos la señal del GPS
- Capturamos la fecha, la hora, minutos y segundos, la longitud, la latitud
- Abrimos la tarjeta SD y almacenamos la información, cerramos la tarjeta SD
- Empieza el proceso de nuevo.

  Véase Apéndice 1: Código del programa

6 Ejemplo de Medidas Realizadas

Han sido realizadas pruebas para verificar que tanto la captura de valores del sensor como los datos recibidos por el GPS son correctos. Dichas medidas eran sometidas a un doble control: por un lado se instaló en el software un contador que verificara que las medias almacenadas en la tarjeta SD no tenían discontinuidades. Por otro lado se utilizó el programa RealTerm que captura la salida de datos serie del Arduino. En un principio cuando se utilizó la librería TinyGPS++ se observó que había una discontinuidad aleatoria que solo fue observada cuando se hicieron medidas de más de

una hora de duración. Se constató, después de varias pruebas, que dicha discontinuidad aleatoria era debida a una inestabilidad de la librería, ya que necesitaba recursos que en ocasiones, el Arduino no era capaz de proporcionar. Se cambió a la librería TinyGPS la cual consume menos recursos a cambio de una menor precisión de la medida que para el proyecto desarrollado es irrelevante.

6.1 Formato de lo que se almacena en la Tarjeta SD

El formato en el que se almacenan los datos es el que se muestra en la Tabla 2.

valor	seq	latitud	longitud	fecha	hora	tiempo desde la toma de datos
pas2000ce= 3	numero seq=1	LATITUD=40385928	LONGITUD=-3688950	10515	9335329	Fix age:3 ms.

Tabla 2: Formato almacenamiento medidas en tarjeta SD

pas2000ce= 18; indica el valor de la medida obtenida en nanogramos/metro cúbico.
numero seq indica el numero de la medida realizada
40385928 significa 40 grados, 23 minutos 7,7208 segundos norte
-3688950 significa 3 grados, 41 minutos 20,22 segundos oeste
10515 significa 1 de mayo de 2015
9335329 significa que la medida se tomó a las 9 horas, 33 minutos, 53 segundos y 29 centésimas de segundo UTC.
Fix age:3ms indica que la antigüedad de la medida de posicionamiento tiene 3 ms.

6.2 Captura con el programa RealTerm.

Dado que el monitor de Arduino no nos permite capturar la información, es decir no nos permite almacenarla, usaremos el programa Realterm del cual se muestra la captura de la última medida.

numero de iteracciones=60

---

pas2000ce= 18

Cuando hay una ráfaga de más de 58 bytes los últimos bytes contienen la medida de PPAH

6.3 Gráfico de las medidas realizadas

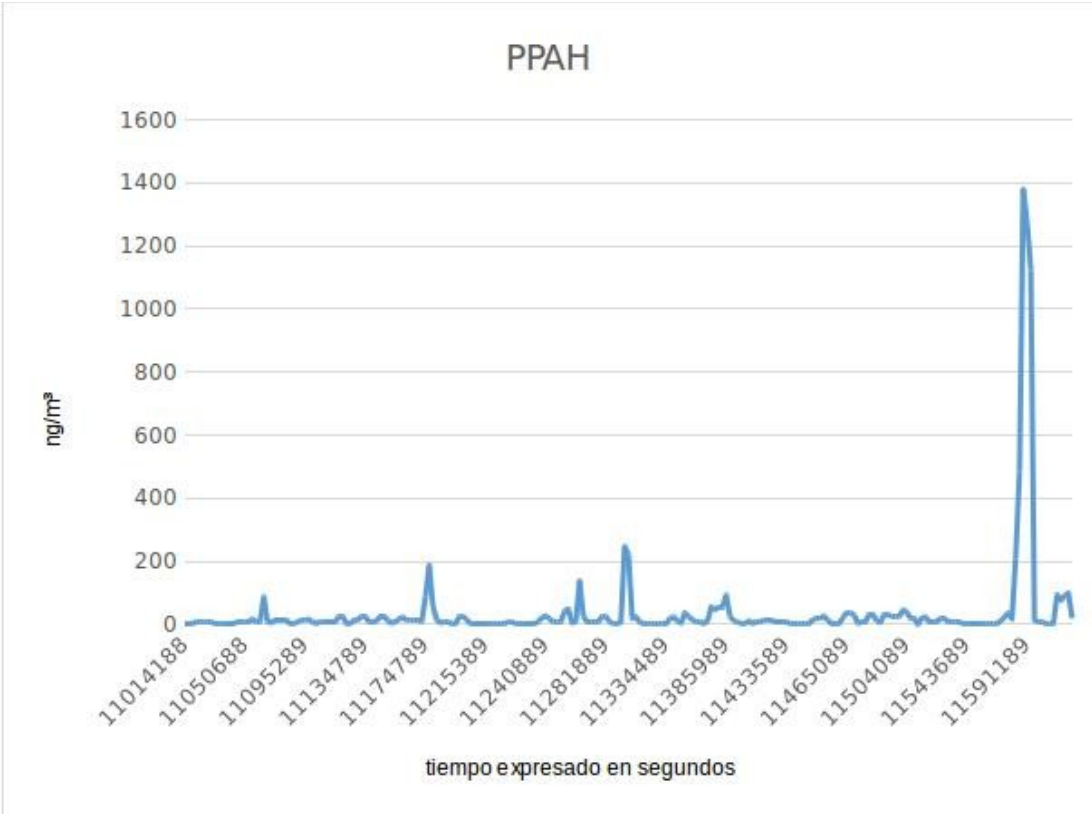


Figura 12: Medidas 25 de abril

El día 25 de abril de 2015 se hizo una captura en la calle como se observa en la Figura 12, en el eje de abscisas se indica a la hora (UTC) que se tomaron y en el de ordenadas el nivel de partículas. Véase Figura 13.

Figura 13: Prueba 25 de abril

Esta medida se hizo en una calle relativamente cerca de un semáforo. El pico que se observa es debido a que en un momento determinado los coches estuvieron parados al lado del sensor. Latitud 40,385540 norte y longitud 3,688372

En esta prueba se constató que la librería TinyGPS++ era inestable por lo que se realizaron más medidas tendentes a asegurarnos de la fiabilidad.

Se hicieron pruebas en la misma calle en sentido contrario donde la circulación no era interrumpida por el semáforo

Se realizaron dichas pruebas el 27 de abril. Los resultados se muestran en la Figura 14.

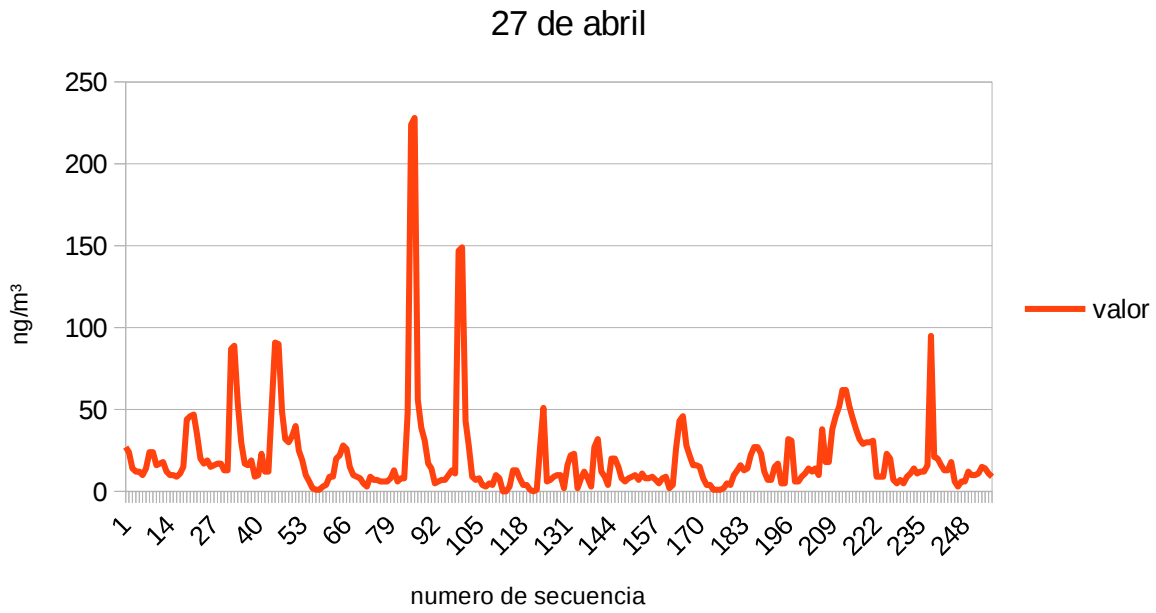


Figura 14: Medidas 27 de abril

En dicha medida se observa que el tráfico, al ser más fluido, los valores son sustancialmente menores.

Se hicieron pruebas en el interior de una vivienda con el propósito de verificar las secuencias del equipo sensor así como del receptor GPS. Véase Figura 15 y Figura 16.

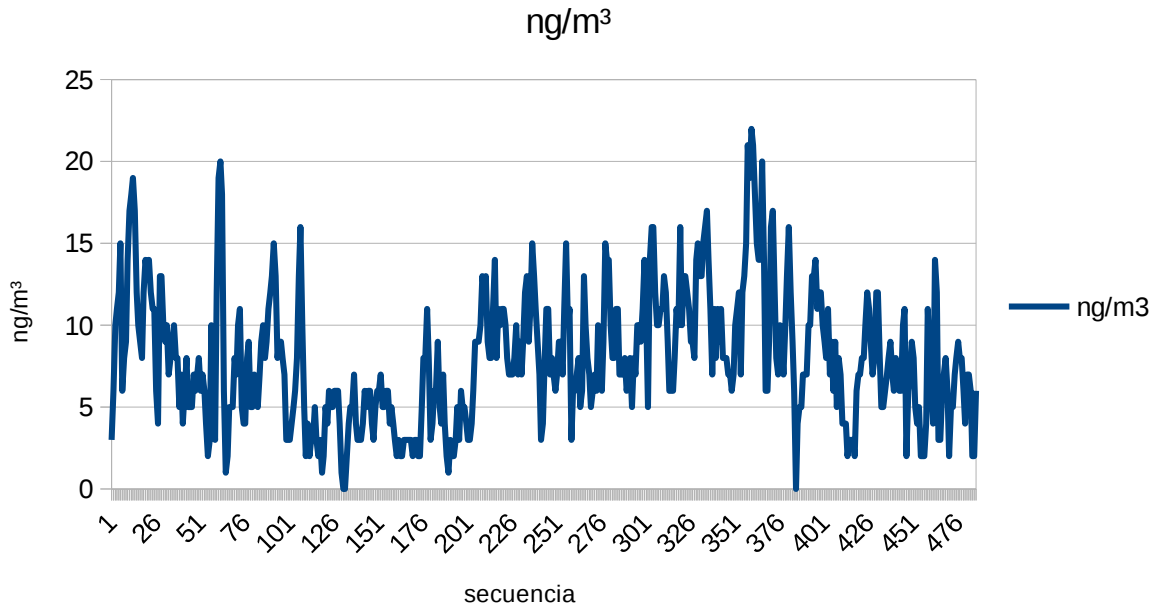


Figura 15: Medidas interior vivienda 29 de abril

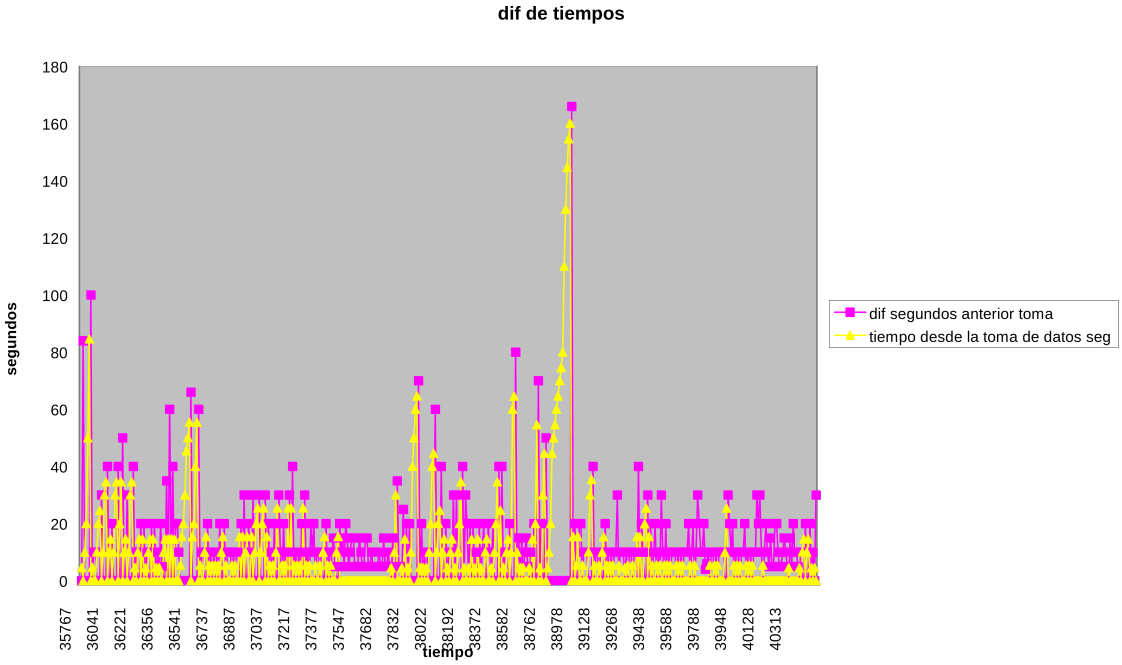


Figura 16: Antigüedad de las medidas del receptor GPS

Dichas medidas se hicieron tomando 487 lecturas el 29 de abril a partir de las 12 horas. En promedio las medidas tomadas por el sensor se realizaron cada 9,66 segundos y una desviación típica de 15 ,25. El GPS tuvo un promedio de retardo en la medida de 9,16 segundos con una desviación típica de 19,41.

Ante estas medidas se hicieron mejoras en el software tendentes a mejorar el tiempo de adquisición de las medidas del GPS, y a consecuencia de ello se realizaron el día 1 de mayo 657 lecturas, y los resultados fueron para el sensor una media entre tomas de 10,15 segundos y una desviación típica de 12,88. Para el GPS fue un promedio de lectura de 5,98 segundos con una desviación típica de 11,2.

7 Presupuesto

Para la realización de este proyecto se han realizado las compras reflejadas en la Tabla 3:

Producto	Precio unitario
MODULO ARDUINO UNO REV 3	19,01 €
MODULO GPS LOGGER VENUS SPARKFUN	39,83 €
MODULO CONVERSOR RS232 / TTL	10,33 €
REGLETA 2,54MM MACHO PIN RECTO 40 CONTACTOS	1,36 €
MODULO ARDUINO SD CARD SHIELD	11,90 €
RESISTENCIA CARBON 1/4W 330 OHMS	0,21 €
SaintSmart DDS120 OSCILOSCOPIO DIGITAL	49,59 €
MODULO BOARD 550 CONTACTOS	8,14 €
2 CONECTORES RS232 MACHO	1,65 €
TOTAL	142,02 €
TOTAL IVA INCLUIDO	171,85 €

Tabla 3:Presupuesto

En dicho presupuesto no está incluida la tarjeta SD de un 1Gbyte ni el software utilizado ni el cableado.

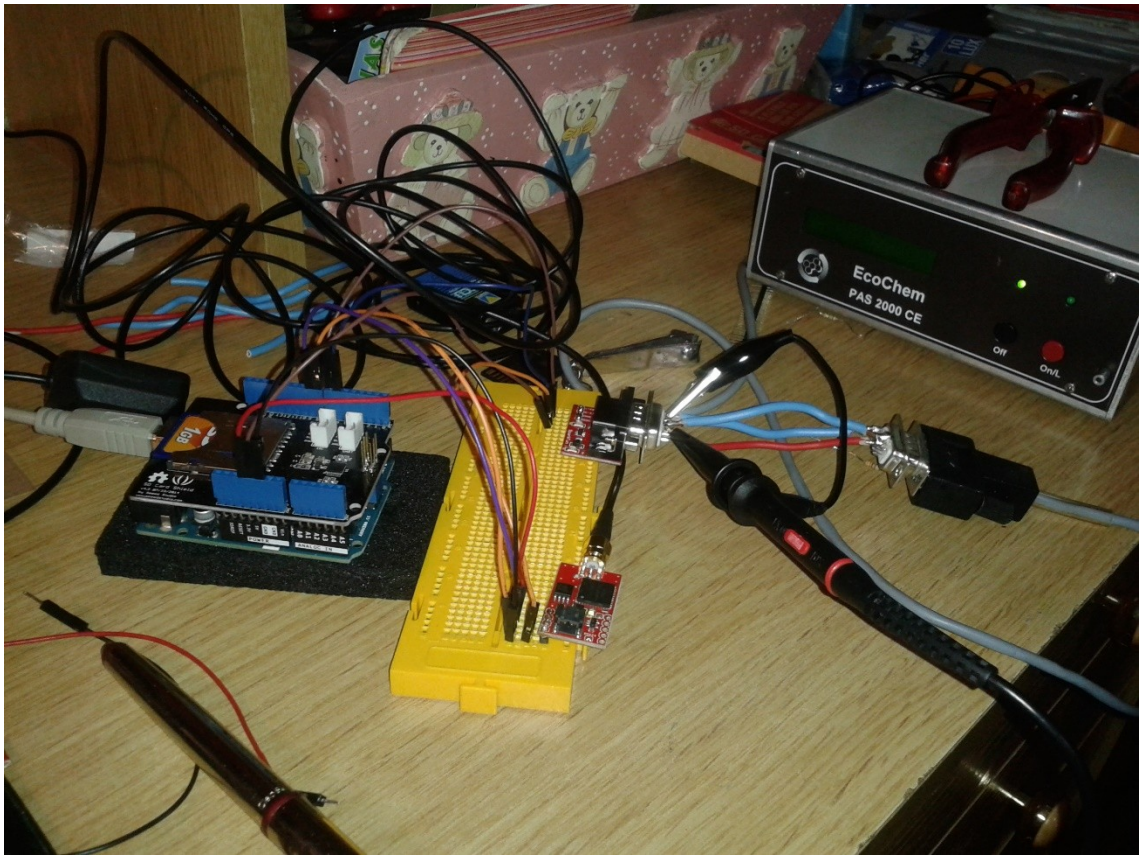


Figura 17:Equipo montado

8 Limitaciones y futuros trabajos.

El microprocesador Arduino tiene tres tipos de memoria, una de las cuales, la memoria FLASH es no volátil y es donde se almacena el programa a ejecutar. Tanto la librería SPI.h (Serial Pheripheral Interface) como la SD.h consumen mucho espacio de memoria siendo la capacidad total de la memoria FLASH de 32Kbytes. Así mismo las capacidades de comunicación serie son bastante limitadas como se ha constatado en capítulos anteriores. Si se desea conectar más de un equipo de

medida que utiliza una comunicación RS232 deberá utilizarse otro tipo de Arduino. En la Tabla 4 que se adjunta se puede obtener una idea del tipo de microprocesador necesario. [13]

	Voltaj e De Operación	Voltaje De Alime	Flash [KB]	SRAM [KB]	I/O digitales /PWM	Pines Analógicos (I/O)	UART	Compatibilidad Con Shields	Notas
Uno	5v	7-12v	32	2	06/14/15	6/0	1	Excelente	
Pro	5v	5-12v	32	2	06/14/15	6/0	1	Excelente	Requiere FTDI para programar
Pro Mini	5v	3.35-12v	32	2	06/14/15	6/0	1	N/A	
Leonardo	5v	7-12v	32	2.5	07/20/15	12/0	1	Decente (diferencias de pines)	USB nativo
Micro	5v	5v	32	2.5	07/20/15	12/0	1	N/A	Compatible con protoboards
Mega	5v	7-12v	256	8	54/15	16/0	4	Buena (Algunas diferencias de pines)	
Mega ADK	5v	7-12v	256	8	54/15	16/0	4	Buena (Algunas diferencias de pines)	Funciona con ADK (Android)
Due	3.3v	7-12v	512	96	54/12	02/12/15	4	Mala (Diferencias de pines y voltaje)	El procesador mas rápido
Ethernet	5v	7-12v	32	2	04/14/15	6/0	-	Muy buena (pocos conflictos con pines)	Requiere FTDI para programar
Lilypad	3.3v	2.7-5.5v	32	2	04/09/15	4/0	-	N/A	Pads para coserse
Esplora	5v	5v	32	2.5	N/A	N/A	-	N/A	Integración nativa con sensores

Tabla 4: Comparativa microprocesadores Arduino

Por otro lado si se desea mantener la configuración del equipo de medida con el Arduino Uno, se puede optar, para evitar periodos de latencia excesivos, prescindir del almacenamiento de la tarjeta SD. El inconveniente es que sería necesario una conexión con un pc y almacenar los datos con el programa Real Term. Además el hecho de tener una conexión GPS, compartiendo el mismo

microprocesador compromete la precisión de la posición, fecha y hora. Si se dispusiera de otro microprocesador exclusivamente para estos menesteres la precisión mejoraría considerablemente.
Debe constatarse así mismo, que las comunicaciones serie de los equipos de medidas no son eficientes, dado que la tecnología actual ya no provee de comunicaciones RS232 por considerarlas obsoletas, dificultando la captura sistemática de los datos.

9 Conclusiones

Se ha desarrollado un dispositivo portátil que permite almacenar en tiempo real (cada 10 segundos aprox.) los datos provenientes de un sensor de partículas junto con la señal de posición de un GPS y almacenar dicha información en una tarjeta SD de una forma autónoma, y siendo éste desarrollo de dimensiones reducidas y de bajo costo. En el desarrollo de este proyecto nos hemos enfrentado a varias dificultades, una de las que llevó más tiempo fue averiguar las características del sensor PAS2000CE, las cuales una vez determinadas, obligarón a modificar el proyecto implicando mayores necesidades de memoria. Ésta modificación, a su vez limitó las prestaciones del microprocesador y conllevó que debieran utilizarse otras librerías para que los datos de recepción del GPS pudieran ser almacenados correctamente, en detrimento del diseño original. Las medidas realizadas muestran que la variación de los valores del sensor en condiciones atmosféricas estable en el interior de una vivienda están en un rango inferior a 25 nanogramos/m³ y que estos valores en el exterior pueden multiplicarse, en función del tráfico observado, de decenas, de centenares de veces más.

10 Referencias

[1] Medidas de la calidad del aire en Madrid: http://www.mambiente.munimadrid.es/opencms/opencms/calaire

[2] Siegmann, P., Acevedo, F.J., Siegmann, K., Maldonado-Bascón, S., “A probabilistic source attribution model for nanoparticles in air suspension applied on the main roads of Madrid and Mexico City,” Atmospheric Environment, 42, 3937–3948, (2008).

[3] Reglamento homologación vehículos de motor Euro 5 y Euro 6: http://register.consilium.europa.eu/doc/srv?l=ES&f=ST%203602%202007%20REV%202

[4] Borrador de Real Decreto por el que se regulan las emisiones de contaminantes atmosféricos procedentes del motor de los vehículos que indica los procedimientos para la inspección de emisiones y valores limite: http://www.magrama.gob.es/es/calidad-y-evaluacion-ambiental/participacion-publica/RD_emisiones_vehiculos_tcm7-310352.pdf

[5] Klara Slezakova, Simone Morais and Maria do Carmo Pereira, “Atmospheric Nanoparticles and Their Impacts on Public Health”, Intech Open Science, Current Topics in Public Health, pp 503-529. 2013. http://dx.doi.org/10.5772/54775

[6] HAP ...

[7] Sensor de partículas. http://www.ecochem.biz/PAH/PAS2000CE.htm

[8] Siegmann y Siegmann, 2000;

[9] Burtscher, 1992

[10] Steven F. Barret Arduino Microcontroller Proccesiing for Everyone

[11] Brian Evans, Beginning Arduino Programming

[12] Librería Altsoftserial. https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html

[13] Tabla comparativahttp://5hertz.com/tutoriales/?p=571

11 Apéndice 1: Código del programa

El código para el microprocesador Arduino es el siguiente:

```
#include <TinyGPS.h>

#include <AltSoftSerial.h>

#include <SD.h>

#include <SPI.h>



#include <SoftwareSerial.h>
// The TinyGPS object
TinyGPS gps;
SoftwareSerial pasSerial(6, 7); // RX, TX (TX not used)but you must connected
// The serial connection to the GPS device
AltSoftSerial ss;// only 8 rx and 9 tx
/*
  SD card read/write

 This example shows how to read and write data to and from an SD card file
 The circuit:
* SD card attached to SPI bus as follows:
** MOSI - pin 11
** MISO - pin 12
```

```
 ** CLK - pin 13
 ** CS - pin 4
 */

const int sentenceSize = 100;

char capture[sentenceSize];


File myFile;
void setup()
{
  Serial.begin(9600);
  pasSerial.begin(9600);
  ss.begin(9600);
  Serial.print("Initializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by default.
  // Note that even if it's not used as the CS pin, the hardware SS pin
  // (10 on most Arduino boards, 53 on the Mega) must be left as an output
  // or the SD library functions will not work.
  pinMode(10, OUTPUT);

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
```

```
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.


}
void loop()
{
  // declarations
  static int i = 0;
  String rita;
  static long  unsigned contador = 0; // only for 4 days
  char filename[13] = "pas2000.txt";
  bool newdata = false;
  long int latitud, longitud;
  long unsigned fix_age, hora, fecha;


  //--------------------------
  if (pasSerial.available())
  {
    char ch = pasSerial.read();
    capture[i] = ch;
    i++;
                                                                                //
Serial.println("_____________________________________________________________
_____________");
    //Serial.print(ch);
```

```
  //Serial.print(capture[i]);

 }


 else
 {
  for (int l = 0; l <= i; l++)
  {
   Serial.print(capture[l]);
   Serial.print("numero de iteracciones="); Serial.println(l);
  }
  if (i >= 58 ) // we eliminated short sequences
  {
   Serial.println("-------");
        Serial.print("pas2000ce="); Serial.print(capture[i - 6]); Serial.print(capture[i - 5]);
Serial.print(capture[i - 4]); Serial.println(capture[i - 3]);

   Serial.println(contador);
   Serial.println("xxxxxxxx");
   myFile = SD.open(filename, FILE_WRITE);
   myFile.print("pas2000ce=");
   myFile.print(capture[i - 6]);
   myFile.print(capture[i - 5]);
   myFile.print(capture[i - 4]);
   myFile.print(capture[i - 3]);
   myFile.print(",");
   myFile.print("numero seq=");
```

```
contador++;
myFile.print(contador);
myFile.print(",");

//--------------------------------------------------------------
while (ss.available() > 0)
{
  int c = ss.read();
  if (gps.encode(c))
  { bool newdata = true;
    break;
}
}

if (newdata = true)
{

  // retrieves +/- lat/long in 100000ths of a degree
  gps.get_position(&latitud, &longitud, &fix_age);
  myFile.print("LATITUD=");
  myFile.print(latitud == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : latitud, 0);
  myFile.print(",");
  myFile.print(" LONGITUD=");
  myFile.print(longitud == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : longitud, 0);
  myFile.print(",");
  // time in hhmmsscc, date in ddmmyy
  gps.get_datetime(&fecha, &hora, &fix_age);
  myFile.print(fecha);
```

```
      myFile.print(",");
      myFile.print(hora);
      Serial.println(fecha);
      Serial.println(hora);


      Serial.print("  Fix age: ");  Serial.print(fix_age); Serial.println("ms.");
      myFile.print(",");
      myFile.print("  Fix age:");
      myFile.print(fix_age);
      myFile.print(" ms.");
      myFile.println(";");
      myFile.close();



    }


   }
   for (int l = i - 6; l <= i - 3; l++)
   { capture[l] = 0;
   }
   i = 0;
  }
}
```

12 Apéndice 2:Diagramas de los equipos utilizados

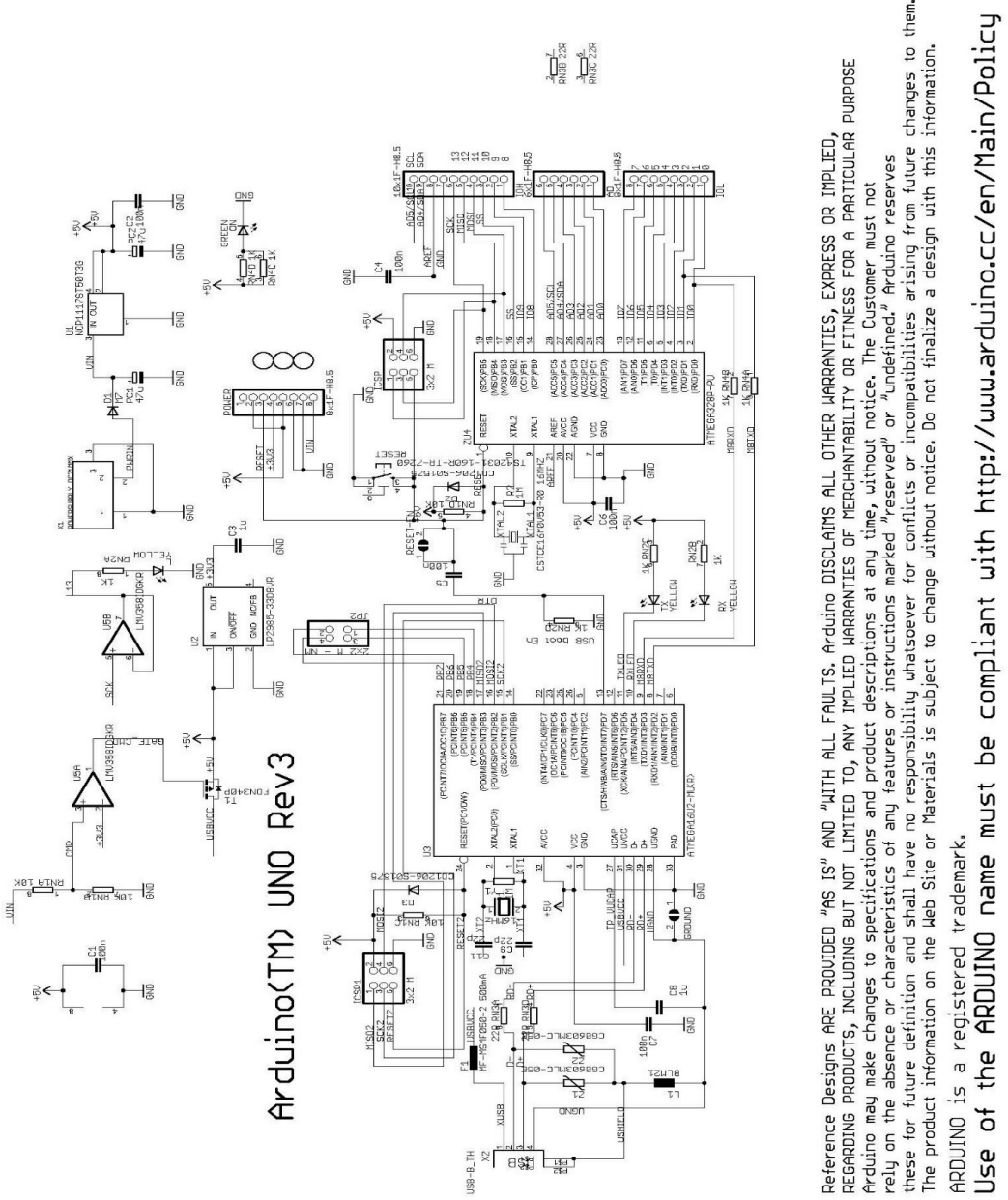


Diagrama 1: Arduino Uno

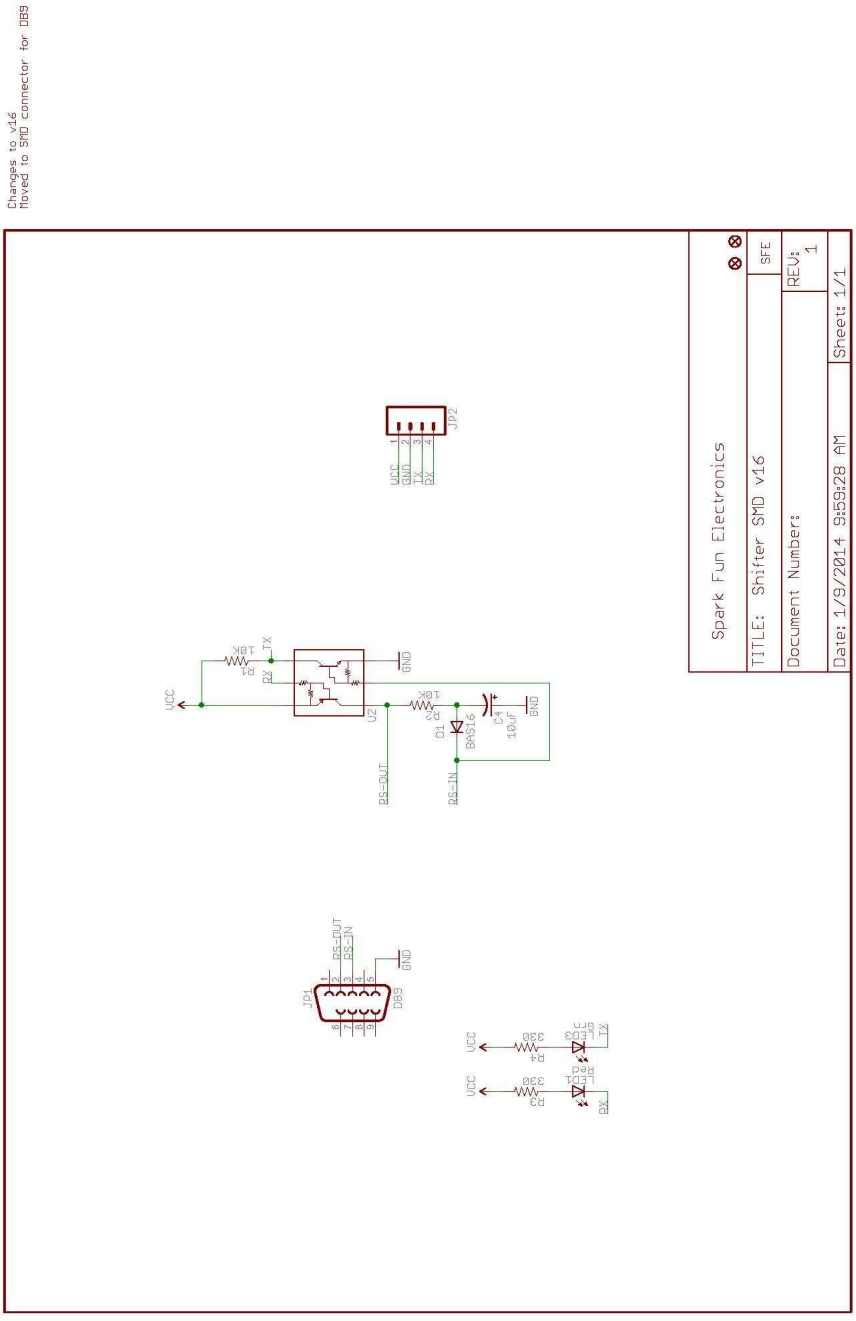


Diagrama 2: Adaptador RS232 TTL

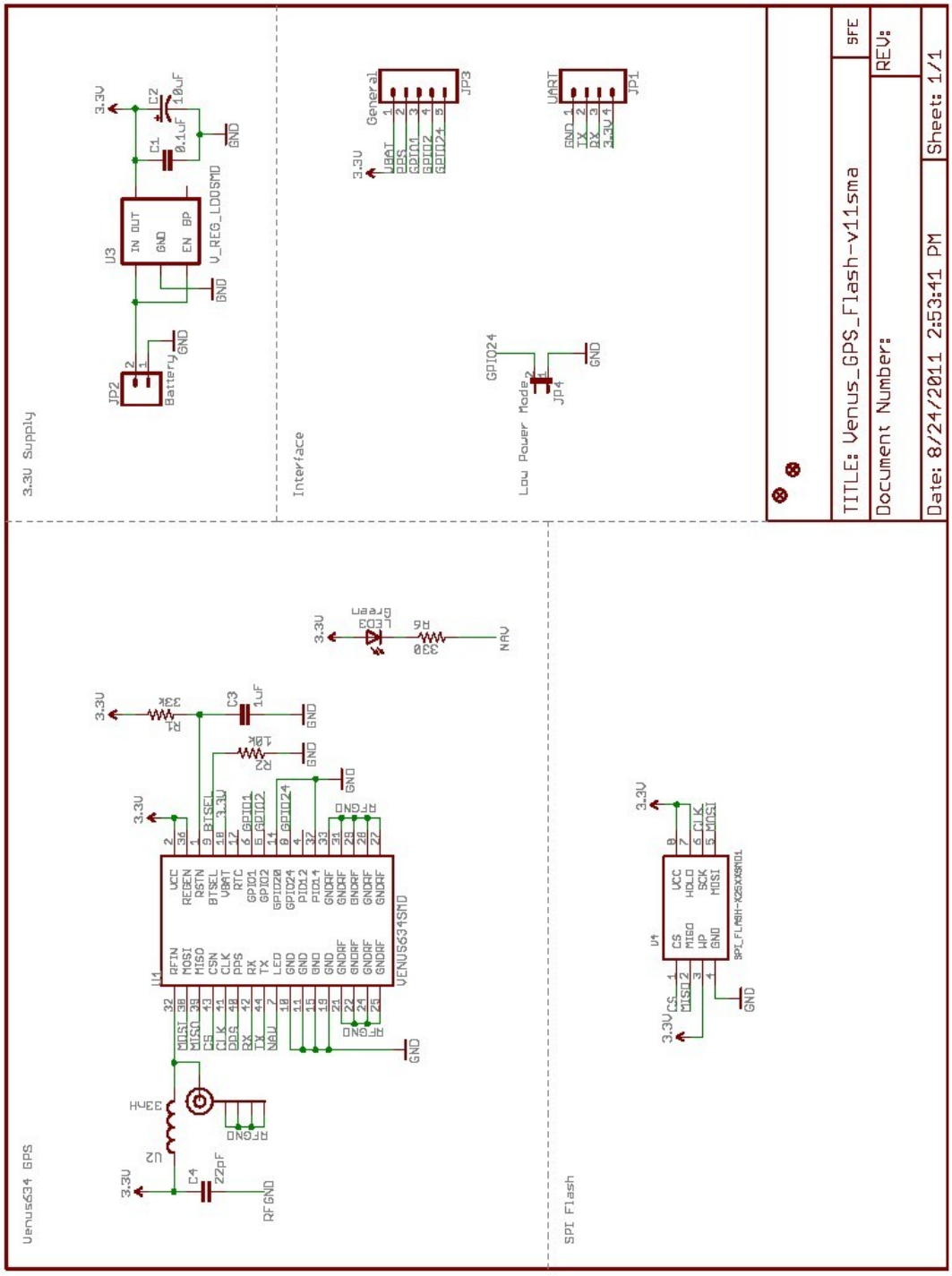


Diagrama 3: GPS

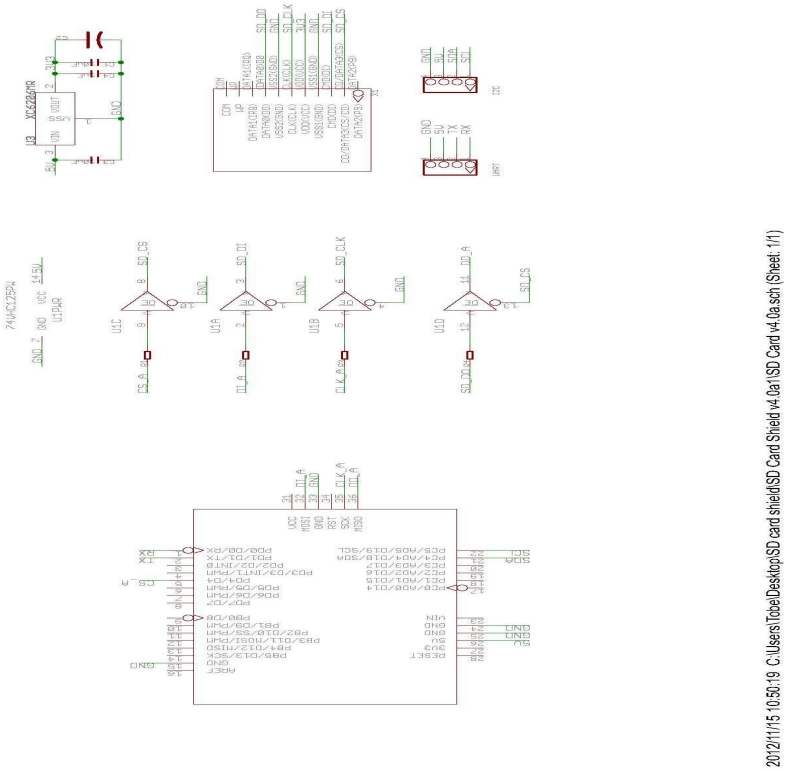


Diagrama 4: Tarjeta SD

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá