PROYECTO PARA LA REALIZACION DE UN ALTIMETRO CON BASE ARDUINO.

David Sariñena

7 Enero 2013

Indice Página

Introducción. Condiciones del proyecto 3

Notas generales del proyecto. 4

Notas sobre placas microcontroladora 4

Notas sobre baterías LiPo 4

Partes del proyecto 5

Diseño del dispositivo. 5

Notas sobre bus I2C 7

Fabricación del dispositivo. 11

Instalación del entorno de trabajo en el ordenador. 11

Programación del dispositivo 12

Notas sobre programación 13

Notas sobre el programa básico 13

Pruebas finales 13

Conclusiones 14

Agradecimientos 14

Enlaces de interés 14

Apéndice 1: Programa básico 15

Apéndice 2: Programa de borrado de memoria 25

Introducción. Condiciones del proyecto.

La idea básica del proyecto consiste en la elaboración de una altímetro de la manera más simple posible, que sea compacto y de poco peso, adecuado para cohetes con motores de tipo D ó E. Quería además volver a recordar un poco la electrónica que hice hace 20 años pero sin comprarme nuevo equipamiento (tengo poco más que un soldador viejo y un tester de 20 euros).

Si el proyecto terminaba bien era mi intención seguir ampliando el dispositivo con más memoria o sensores, pero por el momento estas eran las condiciones iniciales que pretendía cumplir:

1) El altímetro tiene que ser capaz de tomar la altura máxima. En una fase posterior y como mejora, debería guardar todo el perfil de vuelo.

2) El altímetro tiene que caber en un tubo de 40mm de diámetro

3) El peso total, batería incluida, no debe superar el peso de una pila de 9V (menos de 50 gr).

4) El proyecto tiene que ser simple, con el mínimo posible de componentes y de fácil montaje. Se planteaba como opción inicial basar el diseño en la plataforma Arduino.

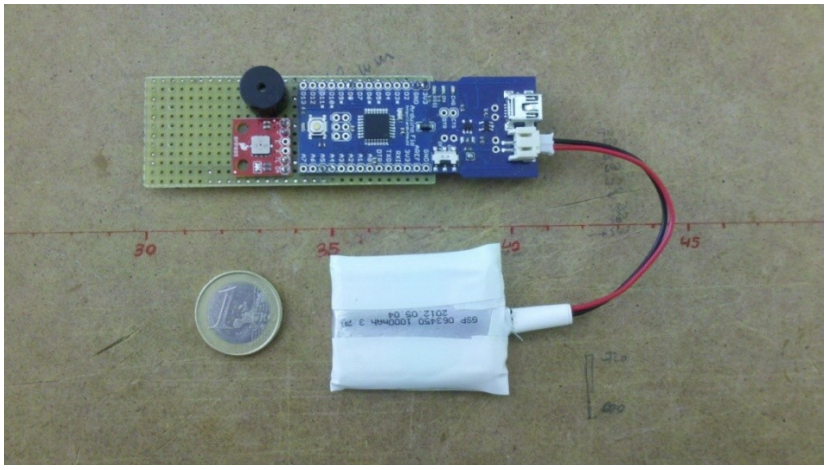

Fig1. Montaje definitivo del proyecto. Se puede ver la placa microcontroladora (azul), placa del sensor altímetrico (rojo), dispositivo acústico (negro), batería (blanco) y placa PCB sobre la que se sueldan los dispositivos (amarillo).

Fig2. Peso de la placa en la que se han añadido algunos componentes para una ampliación en la que estoy trabajando

El documento mostrará que el montaje del hardware fue relativamente sencillo, aunque la parte del software fue más complicada. Así mismo el documento trata todas las etapas del proyecto, por lo cual hay áreas en las que me extenderé para explicar ciertos temas (aparecen en cursiva).

Notas generales del proyecto.

Para desarrollar el proyecto usaré como base una placa microcontroladora Arduino a la que se le conectarán 3 dispositivos:

- Una batería LiPo de 3,7V, capacidad 1000mAh y conector tipo JST-PH de 2mm.
- Un altímetro sensor BMP085, montado sobre una pequeña placa de servicio.
- Un dispositivo acústico que permita dar una lectura in-situ de la altura conseguida.

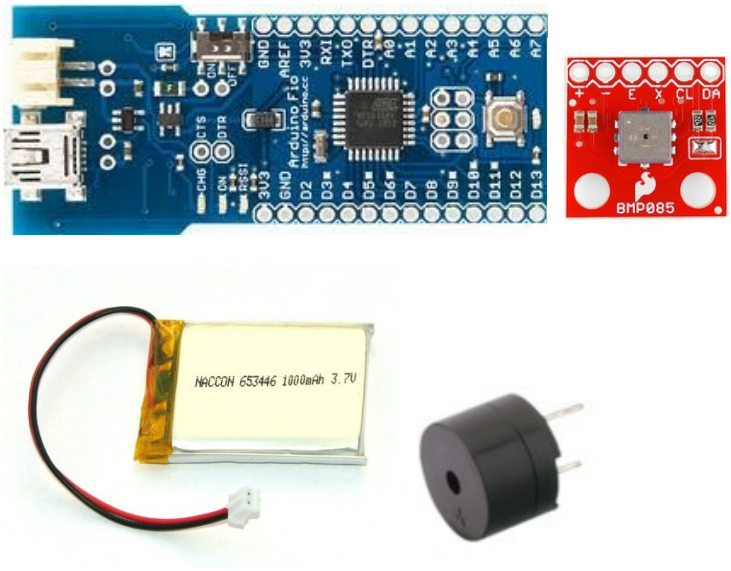

Fig3. Arriba, placa Arduino FIO y sensor BMP085 sobre una placa breakout de Sparkfun. Abajo, batería LiPo y dispositivo acústico con montante PCB (patillas para conectar sobre una placa PCB).

Notas sobre placas microcontroladora

Una placa microcontroladora no deja de ser un pequeño ordenador con CPU, memoria y entradas y salidas analógicas y/o digitales que permite controlar dispositivos y sensores. La fuente de alimentación suele ser externa.

Las placas microcontroladoras tienen memoria sobre la cual es posible insertar un programa; es decir, es posible PROGRAMAR cómo la placa va a controlar los diferentes dispositivos.

La elección de la placa por tanto condiciona el método para cargar el programa. Para el proyecto he escogido una placa de la familia Arduino porque dispone de un entorno simple de programación. Además es un sistema Open source, por lo que esperaba que los componentes fueran de bajo precio.

Hay diferentes placas Arduino, así como extensiones (shields) que mejoran sus capacidades.

Existe una página web en inglés y español con diferentes ejemplos de código para aprender cómo programar la placa, así como un extenso foro donde es posible realizar preguntas y respuestas diversas (ver links al final del documento).

Al ser Arduino Open Source, han aparecido otras placas con prestaciones similares como por ejemplo Netduino, Freeduino, Zigduino, etc.

También existen placas microcontroladoras alternativas como Raspberry Pi.

Notas sobre baterías LiPo

Son una variación de las baterías de iones de litio (Li-ion) y sus características son muy similares (ligereza de sus componentes, elevada capacidad energética y resistencia a la descarga, junto con la ausencia de efecto memoria o su capacidad para funcionar con un elevado número de ciclos de regeneración), pero permiten una mayor densidad de energía, así como una tasa de descarga bastante superior. Estas baterías tienen un tamaño más reducido respecto a las de otros componentes. Su tamaño y peso las hace muy útiles para equipos pequeños que requieran potencia y duración. Cada elemento tiene una tensión nominal de 3,7V (4,2 Ven estado de máxima carga), y se pueden acoplar elementos en serie para obtener tensiones mayores. Así, una batería 3S tiene 3 elementos en serie y su tensión nominal es de 11,1V

Sin embargo son baterías delicadas y deben tomarse precauciones a la hora de manejarlas:

- *La pila nunca debe descargarse por debajo de 3V por elemento (ej, 9V para una batería 3S) o se daña irremediablemente.*
- *Hay que tener mucho cuidado a la hora de recargarlas. Si se le aplica demasiada corriente pueden incendiarse.*
- *Hay que asegurarse de utilizar un cargador capaz de cargar baterías de Polímero de Litio. No usar otro tipo de cargador.*
- *Asegurarse perfectamente de programar correctamente el cargador para el pack que se va a cargar tanto en voltaje como en intensidad.*
- *Inspeccione cuidadosamente el pack especialmente sí el modelo ha sufrido un accidente. Si esta deformado no lo utilice y deshágase de él.*
- *No golpee, pinche doble o deforme el pack de ningún modo.*
- *No seguir utilizando ningún elemento/pack que haya incrementado su volumen (parecido a un globo)*
- *Las baterías de LiPo no deben exceder 60ºC. Durante su uso, si fuera así indicaría que el pack no es el idóneo.*
- *No montar packs de elementos/packs de capacidad desconocida o diferente en serie.*
- *Se debe tener siempre mucho cuidado de no cortocircuitar los elementos/packs de Lipo.*
- *Mantenga sus baterías donde niños o animales no puedan acceder.*

- *Si el electrolito que tiene la batería en su interior toca su piel lavarla con abundante agua y jabón. Si entrase en sus ojos lávelos con agua fría y busque ayuda médica.*

- *Cuando no vaya a utilizar las baterías de Polímero de Litio guárdelas a media carga (3,7/3,8V), nunca vacías o completamente cargadas.*

Partes del proyecto

La realización del dispositivo es algo más que juntar unos componentes y ponerlo a funcionar. Hay que programar su funcionamiento, y eso implica que hará falta un ordenador para realizar el código del programa y cargarlo en la placa (se adjunta al final una copia del código usado).

El despiece del proyecto consistiría pues en:

- Diseño del dispositivo.
- Fabricación del dispositivo.
- Instalación del entorno de trabajo en el ordenador.
- Programación del dispositivo.

Diseño del dispositivo

Como ya se ha indicado, hay 3 componentes que se van a montar sobre una placa PCB, y son la placa Arduino, el altímetro y el aparato acústico.

La placa básica de Arduino es la UNO; es la que aparece en todos los ejemplos de la web y sobre la que se explica cómo instalar el entorno de programación; pero es demasiado grande para el proyecto en curso, así que la placa que he escogido es la Arduino FIO

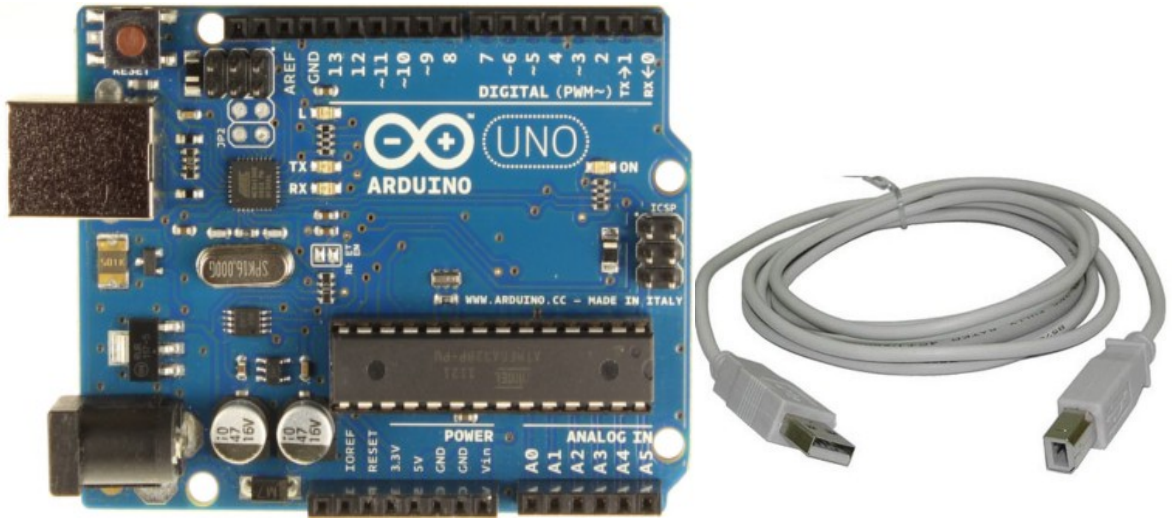

Fig4. Arriba Arduino UNO y cable de alimentación y programacion. Abajo, Arduino FIO y cable USB-FTDI de alimentación y programación. En la placa FIO, los 6 pines de arriba a la izqda son para FTDI (GND-negro, AREF-marrón, 3V3-rojo, RXI-naranja, TXO-amarillo, DTR-verde).

Arduino UNO es bastante más fácil de manejar. Por ejemplo, se puede alimentar y cargar programas con un simple cable USB para impresoras. No es el caso de la Arduino FIO; la mini conexión USB no permite cargar programas, sólo sirve para RECARGAR baterías LiPo. Para cargar datos hay que usar un cable USB – FTDI o bien comprar el modulo wi-fi, y para alimentar de tensión hay que usar el mismo cable FTDI, una batería LiPo o una fuente ya regulada a 3,3V en el pin 3V3. Yo no he encontrado un cable con FTDI macho, así que le puse unos pines al conector FTDI para hacer conexión con la placa (los conectores FDTI de la placa son los 6 agujeros que hay entre GND y DTR).

Aún siendo de manejo más complicado FIO presenta las siguientes ventajas (al menos en mi caso!):

- Es de un tamaño y pesos reducidos
- Permite trabajar a 3,3V, con lo cual puede funcionar con una batería LiPo de 1 célula.
- El conector para la batería LiPo es un JST de 2mm, justo del tamaño de la batería que dispongo.
- Tiene un circuito para recargar baterías LiPo de 1 célula, por lo que no necesito comprarme el cargador para baterías LiPo, aunque sí un cable USB – MiniUSB para que sea el PC el que entregue corriente a la placa y está a la batería.

Es decir, la carga de programa en FIO es farragosa pero el tamaño reducido y su conectividad a la LiPo compensa (ojo, hay muchas baterías LiPo y tienen diferentes conectores; en la mayoría el conector JST es más grande y no entra en esta placa. Tiene que ser un conector JST PH de 2 patas y 2mm).

FIO también dispone de un switch (on off) que permite apagar la batería LiPo cuando está conectada.

Como la carga en FIO por FTDI es farragosa (cuesta hacer bien una buena conexión), para las pruebas trabajé con una Arduino UNO y para el montaje definitivo, una vez probado el programa, con la Arduino FIO.

El sensor elegido es el BMP085 montado sobre una placa breakout de Sparkfun, que facilita su conexión a otros dispositivos (el altímetro es demasiado pequeño para ser manejado fácilmente). La placa incluye también unos condensadores para filtrar ruido y unas resistencias conocidas como pull-up y que son necesarias en todos las conexiones tipo I2C.

Fig5. Sensor BMP085 montado sobre una placa breakout de Sparkfun. El sensor es la pieza cuadrada central.

Notas sobre bus I2C

I2C es un bus serie de datos. Es decir, por tratarse de un bus de datos permite conectar diferentes dispositivos y enviarse datos de unos a otros, pudiendo haber diferentes dispositivos conectados siempre y cuando cada uno de ellos tenga un "identificador" diferente en el bus. Al tratarse de un bus serie, los bits de datos se envían de uno en uno, como en las conexiones a través de USB.

Todos los dispositivos I2C deben estar conectados con 3 conexiones comunes:

- *Tensión positiva*
- *SDA ó DA: Canal de datos (por donde se envían los datos).*
- *SCL ó CL: Canal de reloj o sincronismo*

Además todo dispositivo debe tener una línea de masa común (GND).

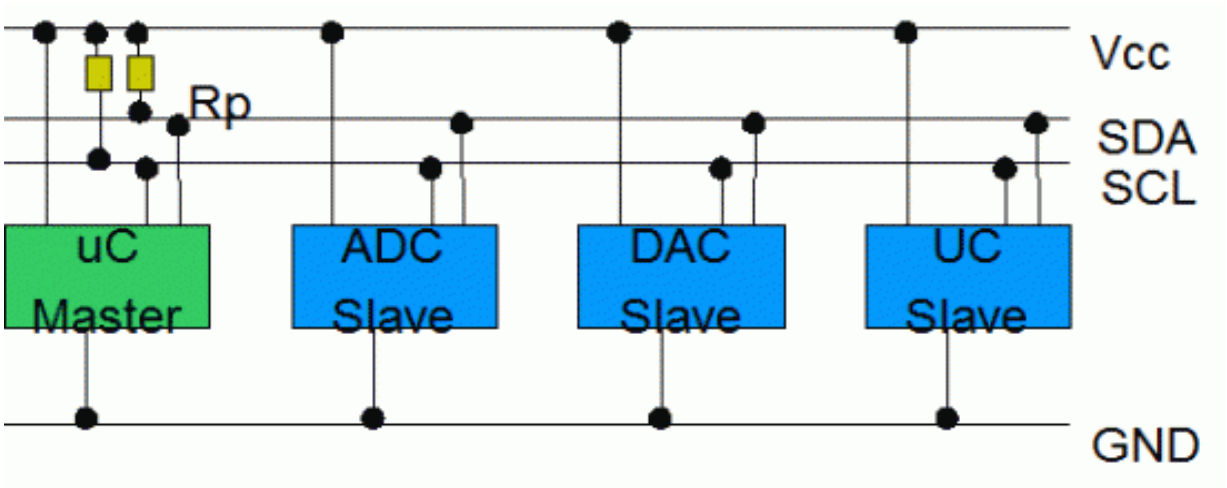

Fig6. Ejemplo de conexión de dispositivos a través de un bus I2C. El master suele ser el microcontrolador, y los dispositivos esclavos (Slave) suelen ser los sensores.

Finalmente, todo bus I2C debe tener 2 resistencias conectadas entre a) Tensión positiva y canal de datos y b) Tensión positiva y canal de reloj. Esas resistencias se conocen como "pull-up" (Rp en la figura anterior). El valor de las resistencias depende del diseño. En nuestro caso, las resistencias están incluidas en la placa roja. Si sólo se conecta el altímetro a la placa Arduino no hay que preocuparse de incluirlas en el diseño porque ya están incluidas, La cosa cambia si desean añadirse más sensores al bus I2C, pero esto no se aborda en el presente documento.

Así pues la placa breakout facilita las conexiones entre el sensor y la placa Arduino y por tanto el bus I2C queda simplificado a la mínima expresión, dejando las conexiones así:

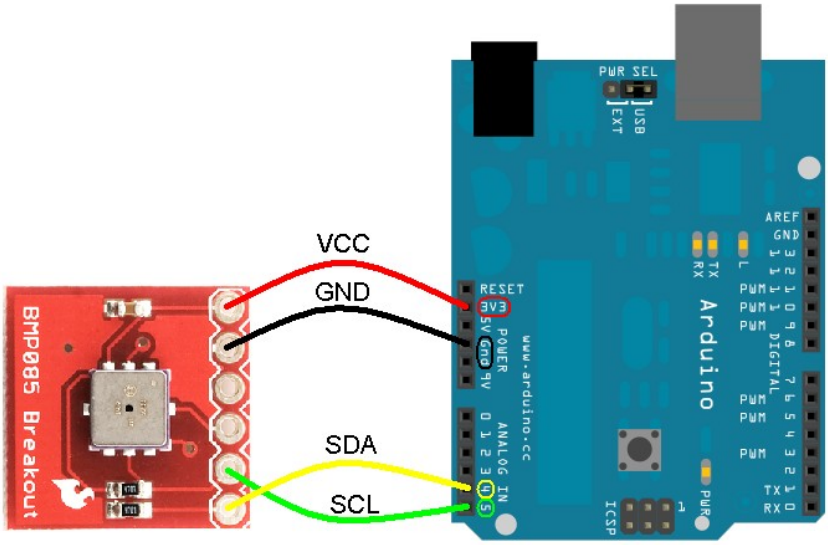

Fig7. Esquema de conexión placa breakout del sensor y placa Arduino

En el esquema se observe una placa Arduino UNO y no una Arduino FIO, pero la conexiones son es las mismas, es decir:

- Conectar VCC (+) de la placa roja a cualquiera de las patas 3V3 de Arduino,
- Conectar GND (-) de la placa roja a cualquier pata GND de Arduino,
- Conectar SDA (DA) de la placa roja a la pata Analógica 4 (A4) de Arduino y
- Conectar SCL (CL) de la palca roja a la pata Analógica 5 (A5) de Arduino.

La pata A4 y A5 de las placas Arduino UNO y FIO son las establecidas para el bus I2C.

Queremos añadir además un dispositivo acústico que nos dará una lectura de la altura alcanzada mediante una serie de tonos. El dispositivo escogido es de tipo de corriente continua sin oscilador interno (transductor electromagnético 1,5VPP para placas PCB). Al ser de corriente continua hay que enviarle una señal variable cuya frecuencia de variación sea el tono escogido (ej, 440Hz). El lenguaje de programación contiene instrucciones para enviar un tono de duración y frecuencia requeridas por cualquiera de las salidas digitales. Baste decir en este punto que las conexiones serán como siguen:

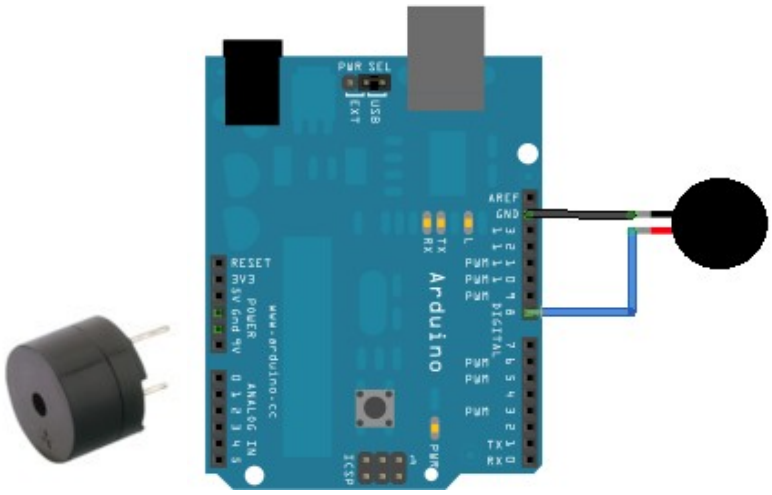

Fig8. Dispositivo acústico y esquema de conexión placa Arduino y el mismo

- Conectar la pata positiva del dispositivo acústico a la entrada digital 8 de Arduino.
- Conectar la pata negativa del dispositivo acústico a cualquier pata GND de Arduino.

Finalmente, vamos a soldar todos los componentes a una placa. Hay varias soluciones, como fabricársela uno mismo (usando luz UV y productos químicos) o usar una placa de tipo PCB y soldar las pistas necesarias, que es el método que he escogido. La PCB debe ser sin conexiones paralelas (la de la primera imagen no vale porque toda la fila está conectada por la misma pista metálica. Las celdas han de ser individuales, como la de la segunda imagen).

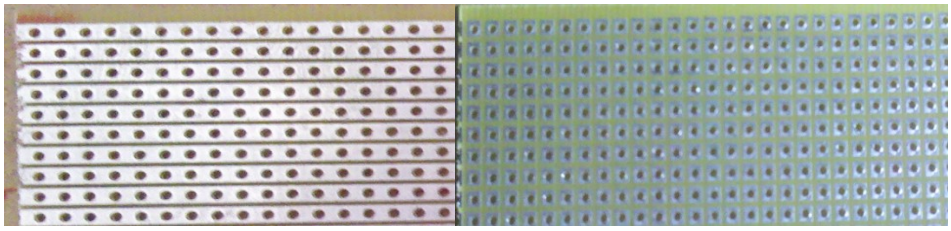

Fig9. Placas PCB de conexión paralela y de celdas individuales

En concreto he usado un trozo de PCB de tipo Placa Standard de fibra de vidrio 1mm topos paso 2,54. Para cortar el trozo necesario (12 x 30 agujeros) he usado un cúter y mucha paciencia.

Para el diseño de las conexiones no disponía de ningún programa de desarrollo, así que como eran pocas conexiones lo he hecho con papel y lápiz. Estoy seguro que con el software apropiado se hubiera conseguido un perfil de conexiones más óptimo.

El diseño de conexiones que he escogido es el siguiente:

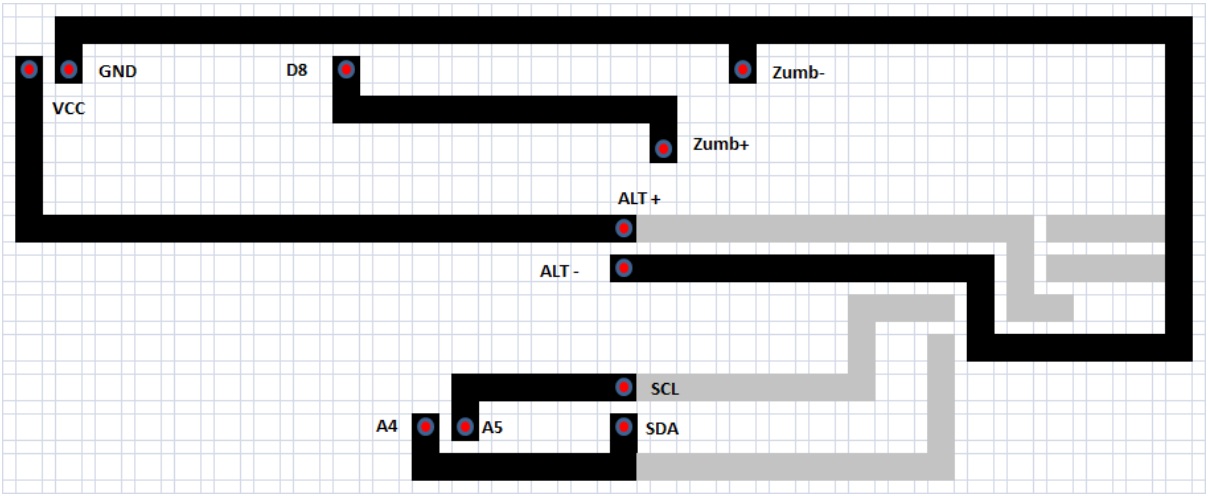

Fig10. Esquema de conexión para el proyecto. En negro las pistas creadas, en gris las pistas para una ampliación que tengo pendiente.

Se puede ver que he dejado un hueco al lado derecho sin conexiones (puntos rojos), y es porque para una segunda fase pretendo añadir memoria externa en la zona derecha y así poder guardar el perfil de vuelo. Las conexiones necesarias para la 2ª fase están en gris, y en este proyecto no son necesarias. Como estaba pensando en la ampliación el diseño de conexiones quedó afectado por ese motivo. Un diseño que no hubiera tenido en cuenta la ampliación hubiera sido algo así:

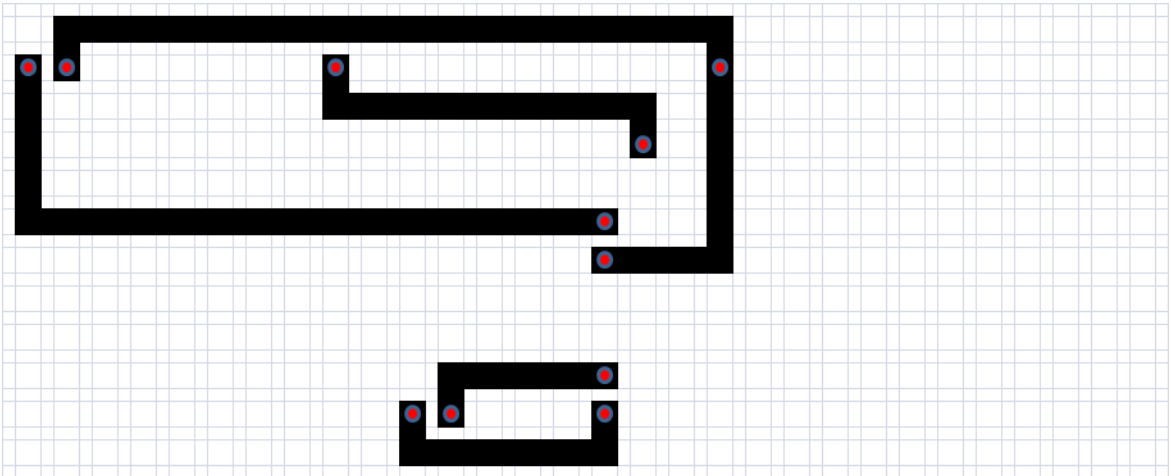

Fig11. Esquema de conexión para el proyecto si no se prevee hacer ampliaciones.

Con un trozo de PCB de 12 x 22 topos en lugar de 12 x 30 hubiera sido suficiente. Las pistas de conexión serán estañadas completamente para mejorar las propiedades eléctricas y la rigidez del diseño, aunque me habrá añadido 2 ó 3 gramos más al producto final.

Otro punto a tratar es el protocolo de tonos proporcionado por el dispositivo acústico, y que se resume en la siguiente tabla:

Concepto	Nº tonos	Duracion	Frecuencia
ERROR memoria llena	7 tonos	0.5 s separados 0.25 s	220 Hz
Presión referencia obtenida	1 tono	2 s	440 Hz
MEDIDA = 5 dígitos			
Señal de inicio de medida	3 tonos	0.5 s separados 0.25 s y 2 s al final	440 Hz
Número 0	1 tono	1 s	600 Hz
Números 1 a 9	1 a 9 tonos	0,25 s separados por 0,25s.	600 Hz
Separación entre dígitos		Un silencio de 0,75s	
Final de medida		Un silencio de 2s	

Ejemplo, para una altura de 00253 metros, la secuencia sería:

- Al inicio, un tono largo de 2s y 440 Hz cuando hubiera obtenido la presión de referencia.
- Una vez lograda la altura máxima, un bucle infinito de:
    - 3 tonos de 0,5s separados 0,25 s + 2s de silencio (Señal de inicio de medida)
    - 1 tono de 1s (Número 0)
    - 0,75s de silencio (Separación entre dígitos)
    - 1 tono de 1s (Número 0)
    - 0,75s de silencio (Separación entre dígitos)
    - 2 tonos de 0,25s separados por silencios de 0,25s (Número 2)
    - 0,75s de silencio (Separación entre dígitos)
    - 5 tonos de 0,25s separados por silencios de 0,25s (Número 5)
    - 0,75s de silencio (Separación entre dígitos)
    - 3 tonos de 0,25s separados por silencios de 0,25s (Número 3)
    - 2s de silencio (Final de medida)

Finalmente unas notas sobre la batería LiPo seleccionada, que es de 1 célula y 1000mAh. No hacía falta una pila de tan alta capacidad, una de 100mAh hubiera sido más que suficiente, pues el sensor consume microamperios y el dispositivo acústico ronda el miliamperio cuando está operativo; pero esta la batería que tenía disponible y además el conector era idéntico al de la placa Arduino FIO, por lo que no me quería complicar más la vida.

Fabricación del dispositivo.

Esta es la parte que más temía dado que hacía 20 años que no montaba nada de electrónica. No dispongo de elementos para fabricar una placa de circuitos, así que como he dicho en el apartado de diseño lo he hecho a partir de una placa PCB y soldaduras. Aquí podéis ver el resultado

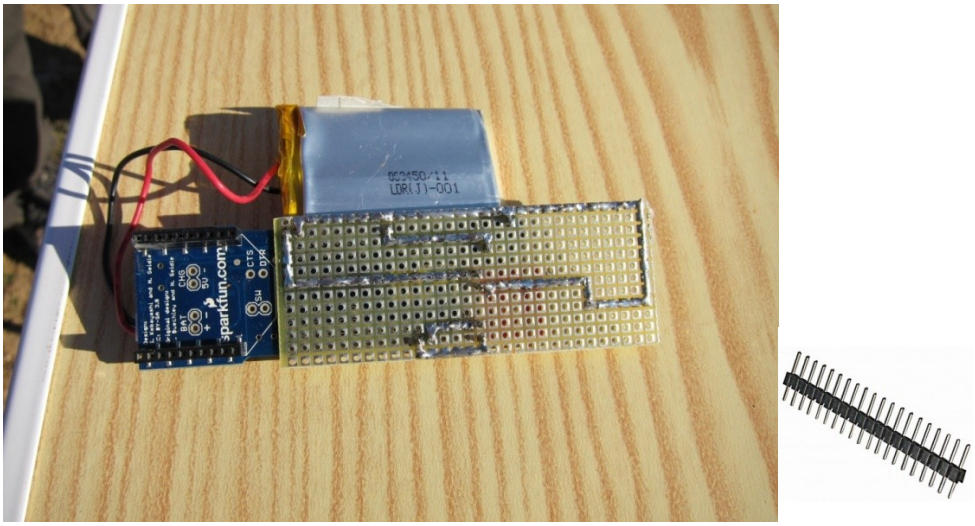

Fig12. Imagen de las pistas soldadas. A la derecha pines usados para afianzar la soldadura.

Notas para los que no habéis usado nunca placas PCB.

1) *La zona metalizada es el reverso.*
2) *Para unir los diferentes componentes a la placa PCB se pueden usar unos pines que los engarcen y luego soldar los pines*
3) *Para hacer las pistas podéis usar cables o estañar toda la ruta. Yo opté por esta opción porque aunque añadía peso el resultado queda más robusto.*
4) *Si no habéis estañado hace mucho tiempo es mejor hacer una prueba previa con un trozo de placa PCB que no vayáis a usar. Ah! Y cuidado con el soldador, que está muy caliente.*
5) *A la hora de soldar es conveniente tener las dos manos libres. Es decir, con una mano se maneja el soldador y con la otra el estaño. Es conveniente fijar la placa a una plataforma que sujete firmemente el proyecto. Yo no tenía ninguna, así que me hice mi propia plataforma con un par de bloques de cajas de CD, una varilla, goma elástica y unas pinzas.*
6) *La placa Arduino FIO sobresale de la placa PCB. Está hecho a propósito; la parte de la placa Arduino que sobresale tiene 2 zócalos negros para montar el dispositivo Wi-fi.*

Entorno de trabajo en PC.

El entorno de trabajo nos permite:

- Crear programas nuevos
- Cargarlos en la memoria de la placa a través de una conexión, que puede ser USB, FTDI, Wi-fi… dependiendo de la placa utilizada.

En primero lugar lo que hay que hacer es descargarse el entorno de trabajo (que permite hacer los programas a cargar en la placa Arduino) y configurarlo.

Es posible descargarse el entorno desde la página web de Arduino. Hay instrucciones para Windows, Mac y Linux (http://arduino.cc/en/Guide/HomePage). La guía de instalación presupone que se va a usar la placa Arduino UNO. Para la placa FIO el procedimiento es el mismo, solo que:

- También hay que instalarse drivers USB – FTDI que no son necesarios en la placa UNO pero sí para la FIO si no se cargan datos via Wi-Fi. Hay un apartado de la guía de instalación que habla de esos drivers.
- Cuando ya esté instalado y se seleccione la placa en el entorno de trabajo (ej, apartado 7 si el sistema operativo es Windows), hay que seleccionar como placa Arduino FIO.

Para comprobar que todo funciona perfectamente es recomendable usar el programa ejemplo básico, conocido como “Blink”. Conectar la placa Arduino mediante el cable correspondiente al PC; abrir el entorno de trabajo y cargar el programa de ejemplo Blink. A continuación, pasarlo a la placa Arduino. Si el programa se carga correctamente en la placa, veremos un pequeño LED cerca del pin Digital 13 (D13) parpadear cada segundo.

Los programas hechos en el entorno de trabajo pueden cargarse en cualquier momento en la placa Arduino, pero si esta no está conectada a los dispositivos esperados por el programa, el funcionamiento puede ser imprevisible. Así pues recomiendo cargar el programa definitivo una vez que ya esté montada la placa Arduino con el sensor, buzzer y batería.

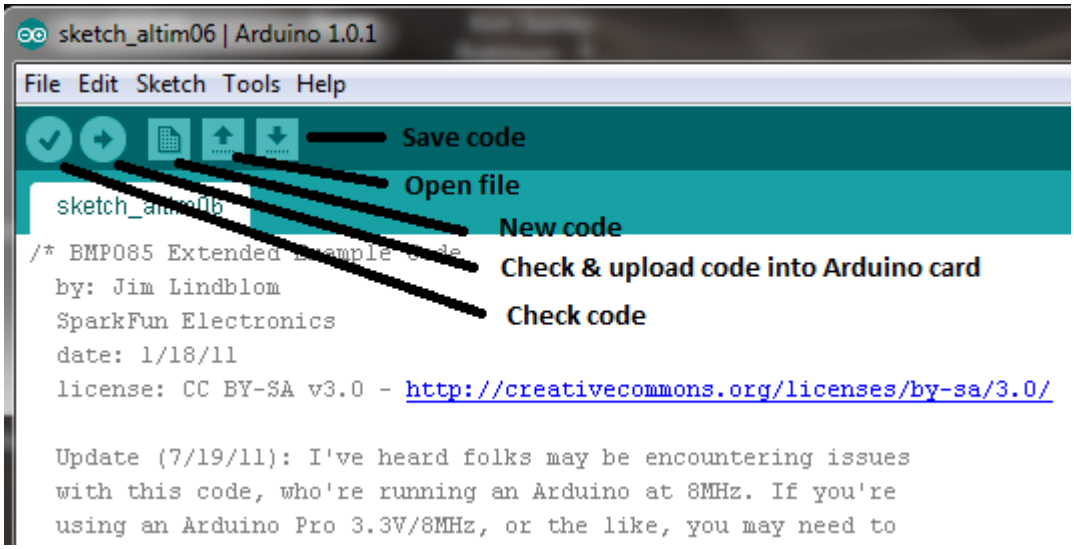

Fig13. Principales funciones del entorno de trabajo

Programación del dispositivo

Esta fue la parte del proyecto que más tiempo me llevó, así que mi idea es que utilicéis el código propuesto tal cual está y así os ahorréis faena.

En verdad, y como el proyecto se ha desarrollado con vistas a una futura ampliación, el código tiene elementos no necesarios en esta etapa. Por ejemplo, hay un algoritmo para guardar la altura máxima en una pequeña memoria EEPROM que contiene la placa. Cada vez que se obtiene una altura máxima, se guarda en la primera zona de memoria libre. El dispositivo puede guardar hasta 100 alturas máximas (lanzamientos), tras lo cual no es operativo (memoria llena). Para limpiar la memoria hay que cargar temporalmente un código de limpieza de memoria, y después volver a cargar el código original. Así pues hay 2 programas desarrollados:

- Programa básico que es el necesario para que el dispositivo funcione (apéndice 1)
- Programa de limpieza de memoria (apéndice 2)

He usado C como lenguaje de programación. Creo que es el básico que propone Arduino para su entorno de programación, desconozco si el entorno funciona con otros lenguajes de programación.
Hay dos maneras de cargar el programa: o se tienen los componentes Wi-fi (y para eso hacen falta 2 módulos Xbee) o se tiene un cable USB – FTDI, que es el método que he usado. Como comentaba tiene el problema que la conexión FTDI es hembra – hembra, así que he usado algunos pins, pero estos bailaban y he tenido que hacer varios intentos para cargar el programa correctamente.

Notas sobre programación

Como he dicho la idea era proporcionar el código para simplificar la fabricación del dispositivo, pero es posible que algún lector esté interesado en modificar el código. Con el fin de aprender cómo programar las placas, la pagina web de Arduino propone una serie de programas de prueba también disponibles en el entorno de programación (este documento no pretende ser un manual de programación en C, así que recomiendo su visualización). Pueden encontrarse en http://arduino.cc/en/Tutorial/HomePage. También hay un pequeño manual de referencia en http://arduino.cc/en/Reference/HomePage, donde pueden verse estructuras, variables, funciones, etc.

Lo básico que hay que conocer es que todo el código a realizar debe estructurarse en 2 funciones: setup() y loop(). El primero se usa para configurar las condiciones iniciales. El segundo contiene el código que se ejecutará continuamente.

Notas sobre el programa básico

El programa se reduce al siguiente funcionamiento.

- ***Revisar que la memoria interna no esté llena.***
- ***Si hay espacio en la memoria***
    - ***Tomar lecturas (rechazando las anómalas). Las primeras 200 lecturas se usarán para calcular la presión (altura) del suelo (presión de referencia).***
    - ***Interpolar los resultados para filtrar ruido***
    - ***Detectar la etapa de vuelo (en rampa, subiendo, bajando, llegando a tierra)***
    - ***Durante la etapa "subiendo", ir registrando la altura máxima.***
    - ***En la etapa "llegando a tierra", dejar de tomar lecturas altimétricas y entonces***
        - ***Guardar en memoria la altura máxima***
        - ***Usar el dispositivo acústico para indicar la altura máxima hasta apagarlo.***

Pruebas finales

El dispositivo fue testeado en mi cohete Pi con motor D12-7, que es el que habitualmente uso. El cohete pesaba 145 sin la electrónica y 186 con la electrónica y la bahía que le preparé.

No atiné en el recálculo de la trayectoria debido al peso extra, y el paracaídas se abrió en plena caída, el cual no aguantó el tirón y la bahía cayó a plomo.

Aun así cuando aterrizó el altímetro seguía funcionando. Openrocket había previsto 230m y el altímetro marcó 220m, lo cual se acerca mucho si se tiene en cuenta que no calculé bien con Openrocket el peso añadido. En ese sentido, creo que la prueba fue un éxito; el altímetro dio una medida correcta aún después de haber recibido un severo golpe.

Conclusiones

Creo haber cubierto los objetivos básicos del proyecto: construcción de un altímetro con pocas piezas y de fácil montaje. La parte software (desarrollo y carga) es algo más complicada, pero es el precio a pagar con componentes open source.
Los próximos pasos serán:

Añadirle memoria suficiente como para guardar la trayectoria de vuelo

Añadirle un switch para que pueda borrar la memoria sin necesidad de recargar programas

Estos dos pasos puedo hacerlos con el modelo presente. Por otro lado me gustaría reducir más el tamaño del dispositivo, añadirle la electrónica para doble despliegue y un dispositivo acústico con más potencia sonora y un acelerómetro. Eso no lo puedo conseguir con la placa FIO, se necesitan voltajes superiores que podrían quemar la placa. Así que como segunda ampliación estoy pensando en alguna otra placa Arduino que soporte las nuevas condiciones.

Otro importante tema a solventar son las instrucciones delay que contiene el programa. El programa se ejecuta de manera secuencial, y la instrucción delay se utiliza para que el código se pare durante unos milisegundos y así tenga tiempo de realizar ciertas operaciones (por ejemplo, cuando se le pide al dispositivo acústico que emita un tono). El problema es que la instrucción delay para TODO el código, así que por ejemplo durante ese tiempo no se toman lecturas altimétricas. Por tanto estoy trabajando en cómo cambiar el código para que se pare lo menos posible, y que sea capaz de hacer sonar el dispositivos acústico al mismo tiempo que el resto del código sigue funcionando.

El coste de los componentes me supuso 45 euros, aunque podría haberlos conseguido por 30 usando componentes clónicos, de segunda mano o páginas web alternativas que usaran portes menores).

Agradecimientos

Debo dar las gracias a Boris du Reau por sus inestimables consejos y por la ayuda prestada en la elaboración de este proyecto.

Enlaces de interés

- Arduino: Pagina web en Ingles: http://arduino.cc/en/
- Arduino: pagina web en español: http://arduino.cc/es/ (desactualizado)

- Arduino: Foro: http://arduino.cc/forum/index.php
- Sparkfun BMP085 breakout https://www.sparkfun.com/products/11282

Otros projectos similares con base Arduino

- Project AltDuino: solución similar, también basada en Arduino, más trabajada y compacta: www.altduino.de
- Boris du Reau (boris.dureau@neuf.fr), está trabajando en un altímetro con doble despliegue con base Arduino, pero en lugar de usar toda una placa Arduino usará sólo la CPU. Se trata de un trabajo muy completo e interesante. También está trabajando de incorporarle algo de memoria. Tiene previsto hacer su primer test a finales de Enero del 2013.

APENDICE 1: PROGRAMA BASICO

(En negro se muestra el código básico propuesto por el proveedor del sensor para su funcionamiento, en azul mis añadidos para el funcionamiento del dispositivo en su conjunto).

```
 /* BMP085 Extended Example Code
  by: Jim Lindblom
  SparkFun Electronics
  date: 1/18/11
  license: CC BY-SA v3.0 - http://creativecommons.org/licenses/by-sa/3.0/

  Update (7/19/11): I've heard folks may be encountering issues
  with this code, who're running an Arduino at 8MHz. If you're
  using an Arduino Pro 3.3V/8MHz, or the like, you may need to
  increase some of the delays in the bmp085ReadUP and
  bmp085ReadUT functions.

  Amendments for rocket flight by David Sariñena

  Version 03 is rocket intended
  * Detects the different stages

  Version 05 debugged and added buzzer
  *  Buzzer tunes
  ** 7 low tunes   at 220Hz --> memory full
  ** 3 tunes       at 440Hz --> start of measure
  ** 1 long tune   at 600Hz --> number zero
  ** n short tunes at 600Hz --> number n

  Version 06.
  * Deleted all Serial Monitor traces.
  * All comments translated to English.
*/

#include <Wire.h>
#include <EEPROM.h>

#define BMP085_ADDRESS 0x77  // I2C address of BMP085

const unsigned char OSS = 0;  // Oversampling Setting

// Calibration values
int ac1;
int ac2;
int ac3;
unsigned int ac4;
unsigned int ac5;
unsigned int ac6;
int b1;
int b2;
```

```
int mb;
int mc;
int md;
unsigned long totalp;

// b5 is calculated in bmp085GetTemperature(...), this variable is also used in
bmp085GetPressure(...)
// so ...Temperature(...) must be called before ...Pressure(...).
long b5;

short temperature;
long pressure;
float prev_pressure;

//data processing
float smooth_press;          //Pressure used
float smooth_press_prev;     //Previous pressure used
float trend;                 //Calculated slope
float trend_prev;            //Previous calculated slope
float alpha;                 //Double exponential smoother parameter
float beta;                  //Double exponential smoother parameter

// Rocket vars
float ref_press;             //Reference preasure calculated in static.
                             //Requires 200 samples without movement (5 seconds)
unsigned long min_press;     //Minimum pressure achieved. THIS IS THE MAIN DATA TO OBTAIN.
float situacion_0;           //Working var
float situacion_1;           //Working var
float situacion_2;           //Working var
float filter;                //To avoid abnormal samples
long stage[4] = {0,0,0,0};   //time when a stage starts in ms (0 = power up the controller).
                             // stage 0: on ramp
                             // stage 1: rocket ascending
                             // stage 2: rocket descending
                             // stage 3: on ground
                             // without an accelerometer it is difficult to get
                             //   other events such us end of thrust or when the parachute
                             //   releases. Moreover it may happen at any of the stages.
int curr_stage;
float threshold;             //IMPORTANT: threshold stage change
                             //(If measure > threshold * measure prev -> change of state)

int led = 13;
int val = 0;
long counter;                //Samples used
long counter_led;            //Controls blinking led to show program is running
long counter_results;        //Controls blinking led to show program is running
int address;                 //EEPROM address memory
int new_address;             //EEPROM address memory


void setup()
```

```
{
  pinMode(led, OUTPUT);               // Pin 13 is output
  digitalWrite(led, LOW);             // Initially turns the LED off by making the voltage LOW
  address = find_free_memory();       // Checks free memory (each measure uses 5 bytes,
                                      // so there is space for 100 measures)
  Wire.begin();                       // Bus I2C
  bmp085Calibration();
  min_press = 200000;
  threshold = 0.999;                  // It is equivalent to 8.5m
  filter    = 0.0016;                 // To prevent abnormal data gathered by the sensor that
                                      // could trigger the stage change.
                                      // This value is equivalent to a speed = Mach 2, so
                                      // I assume rocket speed << Mach 2
  counter   = 0;
  alpha     = 0.2;
  beta      = 0.2;                    // There are other possibilities such as alpha = 0.1,
beta = 0.4;

}


void loop()
{
  blinking_led();
  if (address >=500 ) play_buzzer_memory_full();  // Play 7 low tunes if memory full
  else
  {
    if (curr_stage < 3)
    {
        read_altim();
        counter++;
        if (abs(pressure - prev_pressure) < filter * pressure)
        {
            smooth_data();
            check_stage();
        }
    }
    else play_buzzer();  //only when reaching ground again
  }
}


void read_altim()
{
  temperature = bmp085GetTemperature(bmp085ReadUT());
  prev_pressure = pressure;
  pressure = bmp085GetPressure(bmp085ReadUP());
}


// Stores all of the bmp085's calibration values into global variables
// Calibration values are required to calculate temp and pressure
```

```
// This function should be called at the beginning of the program
void bmp085Calibration()
{
  ac1 = bmp085ReadInt(0xAA);
  ac2 = bmp085ReadInt(0xAC);
  ac3 = bmp085ReadInt(0xAE);
  ac4 = bmp085ReadInt(0xB0);
  ac5 = bmp085ReadInt(0xB2);
  ac6 = bmp085ReadInt(0xB4);
  b1 = bmp085ReadInt(0xB6);
  b2 = bmp085ReadInt(0xB8);
  mb = bmp085ReadInt(0xBA);
  mc = bmp085ReadInt(0xBC);
  md = bmp085ReadInt(0xBE);
}



// Calculate temperature given ut.
// Value returned will be in units of 0.1 deg C
short bmp085GetTemperature(unsigned int ut)
{
  long x1, x2;

  x1 = (((long)ut - (long)ac6)*(long)ac5) >> 15;
  x2 = ((long)mc << 11)/(x1 + md);
  b5 = x1 + x2;

  return ((b5 + 8)>>4);
}



// Calculate pressure given up
// calibration values must be known
// b5 is also required so bmp085GetTemperature(...) must be called first.
// Value returned will be pressure in units of Pa.
long bmp085GetPressure(unsigned long up)
{
  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calculate B3
  x1 = (b2 * (b6 * b6)>>12)>>11;
  x2 = (ac2 * b6)>>11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<<OSS) + 2)>>2;


  // Calculate B4
  x1 = (ac3 * b6)>>13;
```

```
  x2 = (b1 * ((b6 * b6)>>12))>>16;
  x3 = ((x1 + x2) + 2)>>2;
  b4 = (ac4 * (unsigned long)(x3 + 32768))>>15;


  b7 = ((unsigned long)(up - b3) * (50000>>OSS));

  if (b7 < 0x80000000)
    p = (b7<<1)/b4;
  else
    p = (b7/b4)<<1;


  x1 = (p>>8) * (p>>8);
  x1 = (x1 * 3038)>>16;
  x2 = (-7357 * p)>>16;
  p += (x1 + x2 + 3791)>>4;

  return p;
}



// Read 2 bytes from the BMP085
// First byte will be from 'address'
// Second byte will be from 'address'+1
int bmp085ReadInt(unsigned char address)
{
  unsigned char msb, lsb;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 2);
  while(Wire.available()<2)
    ;
  msb = Wire.read();
  lsb = Wire.read();

  return (int) msb<<8 | lsb;
}



// Read the uncompensated temperature value
unsigned int bmp085ReadUT()
{
  unsigned int ut;

  // Write 0x2E into Register 0xF4
  // This requests a temperature reading
```

```
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x2E);
  Wire.endTransmission();

  // Wait at least 4.5ms
  delay(5);

  // Read two bytes from registers 0xF6 and 0xF7
  ut = bmp085ReadInt(0xF6);
  return ut;
}



// Read the uncompensated pressure value
unsigned long bmp085ReadUP()
{
  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Write 0x34+(OSS<<6) into register 0xF4
  // Request a pressure reading w/ oversampling setting
  // El valor del OSS debe escribirse en los 2 bits de mayor peso en el registro 0xF4.
  // El valor base es 0x34 para OSS = 0
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x34 + (OSS<<6));
  Wire.endTransmission();

  // Wait for conversion, delay time dependent on OSS
  delay(2 + (3<<OSS));

  // Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF6);
  Wire.endTransmission();
  Wire.requestFrom(BMP085_ADDRESS, 3);

  // Wait for data to become available
  while(Wire.available() < 3)
    ;
  msb = Wire.read();
  lsb = Wire.read();
  xlsb = Wire.read();

  up = (((unsigned long) msb << 16) | ((unsigned long) lsb << 8) | (unsigned long) xlsb) >>
(8-OSS);

  return up;
}
```

```
// ********************************************************
// HERE STARTS MY CODE (appart from the Setup & Loop cycle)
// ********************************************************

// Double exponential smoothing algorithm is used
// To obtain smoothed data

void smooth_data ()
{
  if (counter == 2)
  {
    smooth_press = prev_pressure;
    ref_press    = prev_pressure;
    trend        = pressure - prev_pressure;
  }
  if (counter > 2)
  {
    smooth_press_prev = smooth_press;
    trend_prev        = trend;
    smooth_press      = alpha * float(pressure) + (1 - alpha)*(smooth_press_prev +
trend_prev);
    trend             = beta * (smooth_press - smooth_press_prev) + (1 - beta)*trend_prev;
  }
}


// Checks flight stage (not necessary for height calculations but I wanted to know
// if it was possible to calculate it without an accelerometer

void check_stage()
{
  switch (curr_stage)
  {
    case 0:  // on ramp
      // Calculate the reference pressure as the average of the first 200 pressure values.
      if (counter < 200)
      {
        ref_press = ((ref_press * (counter - 1)) + smooth_press)/counter;
      }
      if (counter == 200) tone(8,440,2000);

      // Check if threshold has been reached, meaning a stage change
      situacion_0 = float(smooth_press) / float(ref_press);
      if (situacion_0 < threshold)
      {
        stage[1] = millis();
        curr_stage = 1;
      }
    break;

    case 1: //ascending & max height
      if (min_press > smooth_press) min_press = smooth_press;
```

```
      situacion_0 = 0;
      situacion_1 = float(min_press)  / float(smooth_press);
      if (situacion_1 < threshold)
      {
        stage[2] = millis();
        curr_stage = 2;
        write_results();    //Max height stored in this moment
      }
    break;

    case 2: //descending
      situacion_1 = 0;
      situacion_2 = float(smooth_press) / float(ref_press);
      if (situacion_2 > threshold)
      {
        stage[3] = millis();
        curr_stage = 3;
      }
    break;

    case 3: //almost on earth
      situacion_1 = 0;
      situacion_2 = 0;
    break;
  }
}

// Write maximum height in first 5 free EEPROM bytes given by "address"
// Theoretically this function will only be called one, because writing an
// EEPROM takes about 5ms

void write_results()
{
  long heightw;
  int remainder;
  heightw = int(44300 * ( 1- pow(float(min_press / ref_press), (1/5.255))));
  for (int y=address; y<address+5;y++)
  {
    remainder = heightw % 10;
    heightw = int(heightw - remainder) /10;
    EEPROM.write(y,remainder);
    new_address = address + 5;  // New empty address
  }
}

int find_free_memory()
{
  for (int x=0; x<512;x++)
  {
    if (EEPROM.read(x) == 255) return(x);
  }
}
```

```
// Play results with a buzzer.

void play_buzzer()
{
  int value;
  int address_work;
  if (new_address == 0) address_work = address;
  else                  address_work = new_address;

 // step 1 - start signal: 3 tunes
  for (int x=0; x<3; x++)
  {
    tone(8,440,500);
    delay(750);
  }
  delay(2000);
// step 2 - start signal: from 0 to 99999
  for (long y=address_work-1; y>address_work-6;y--)
  {
    value = EEPROM.read(y);
    if (value == 0)
    {
      tone(8,600,1000);
      delay(1750);
    }
    else
    {
      for (int z=1;z<value+1;z++)
      {
        tone (8,600,250);
        delay(500);
      }
      delay(500);
    }
  }
    // step 3 - final silence
  delay (2000);
}

void play_buzzer_memory_full()
{
int value;
 // step 1 - start signal: 3 tunes
  for (int x=0; x<7; x++)
  {
    tone(8,220,500);
    delay(750);
  }
  delay(2000);
}
```

```
void blinking_led()
{
  counter_led++;
  if (counter_led > 50)
  {
    counter_led = 0;
    if (val == 0)
    {
      digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
      val = 1;
    }
    else
    {
      digitalWrite(led, LOW);   // turn the LED on (HIGH is the voltage level)
      val = 0;
    }
  }
}
```

APENDICE 2: PROGRAMA DE LIMPIEZA DE MEMORIA

(A usar si se ha llenado la memoria después de 100 lanzamientos.)

```
/*
 * EEPROM Clear
 *
 * Sets all of the bytes of the EEPROM to 0.
 * This example code is in the public domain.

 */

#include <EEPROM.h>

void setup()
{
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, 0);

  // turn the LED on when we're done
  digitalWrite(13, HIGH);
}

void loop()
{
}
```