



Universidad de Alcalá

Estación meteorológica basada
en LPC1768.



ESCUELA POLITECNICA
SUPERIOR

Autor: Palomo Alonso

Alberto

Asignatura: Sistemas electrónicos digitales avanzados.

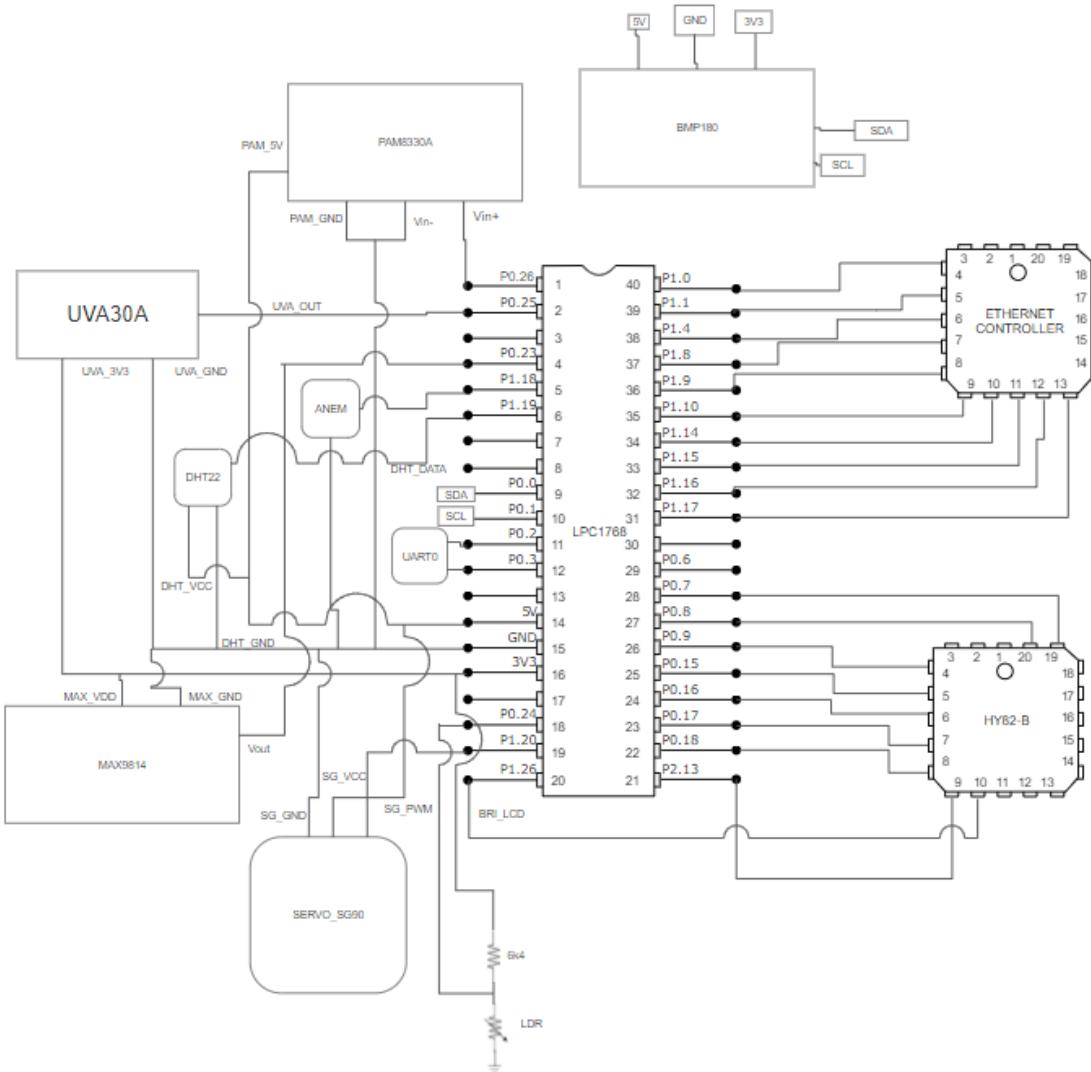
ESCUELA POLITÉCNICA SUPERIOR - UAH.

6 de noviembre de
2019.

Índice de contenido:

| | |
|---|----|
| Abstract. | 5 |
| Capítulo 1: Introducción y objetivos. | 6 |
| Introducción. | 6 |
| Objetivos a cumplir. | 7 |
| Posibles mejoras al sistema en un futuro. | 10 |
| Capítulo 2: Componentes del sistema. | 11 |
| Mini-DK2 | 11 |
| Anemómetro. | 12 |
| LDR. | 12 |
| Sensor UV. | 13 |
| Servo motor. | 13 |
| Amplificador + altavoz. | 14 |
| Micrófono. | 14 |
| Sensor de temperatura. | 15 |
| Sensor de presión. | 15 |
| TFT. | 16 |
| Capítulo 3: Estructura del sistema. | 17 |
| 0. Esquema general. | 17 |
| 1. PWM1 y control de servo. | 18 |
| 2. WEB. | 18 |
| 3. Usuario. | 18 |
| 4. Medio exterior. | 19 |
| 5. Timer controlador de medidas. | 19 |
| 6. Grupo de sensores 1. | 19 |
| 7. Grupo de sensores 2. | 19 |
| 8. WDT. | 19 |
| 9. RTC. | 19 |
| 10. UART. | 20 |
| 11. ADC. | 20 |
| 12. DMA + DAC. | 20 |
| Capítulo 4: Implementación del sistema. | 21 |
| Carpeta MiniDK-2. | 21 |
| Carpeta README. | 21 |
| Carpeta Startup. | 21 |
| Carpeta Principal. | 21 |
| Carpeta Menu. | 22 |
| Carpeta Setup. | 24 |

| | |
|---|-----|
| Carpeta I2C. | 24 |
| Carpeta Anemómetro. | 25 |
| Carpeta ADC. | 25 |
| Carpeta OneWire. | 28 |
| Carpeta PWM. | 29 |
| Carpeta DAC. | 30 |
| Carpeta Database. | 30 |
| Carpeta WDT. | 31 |
| Carpeta UART. | 31 |
| Carpeta TCP/IP. | 32 |
| Carpeta RTC. | 34 |
| Carpeta Timers. | 34 |
| Carpeta SDCARD. | 35 |
| Carpeta CMSIS. | 35 |
| Carpeta GLCD. | 35 |
| Capítulo 5: Pruebas, ejecutabilidad y conclusiones. | 36 |
| Pruebas. | 36 |
| Ejecutabilidad. | 37 |
| Conclusiones. | 38 |
| Capítulo 6: Manual de usuario. | 39 |
| TFT. | 39 |
| WEB. | 42 |
| UART. | 44 |
| Referencias | 45 |
| Anexos. | 46 |
| Anexo I - Esquemático. | 46 |
| Anexo II - Código. | 47 |
| Anexo III - Hojas de datos. | 158 |



Abstract.

Microcontrollers, also known as its acronym MCUs (Micro Controller Unit), are embedded systems which, nowadays, its computational power can overcome microprocessors designed in early 90s as P54C (an old intel processor family) which CPU clock were around 100 MHz and were expensive. Only 20 years later, MCUs overcame, with a remarkable lower cost, its computational power.

In this document, I will talk about an LPC1768 based embedded system which holds a Cortex M3 processor. This MCU has a 100 MHz CPU clock speed and has a Thumb/Thumb-2 subset architecture developed by ARM. In this case, the project to document has been developed in a Mini-DK2 card manufactured by NXP, which contains the following features:

- 100M Ethernet network interface.
 - TFT color LCD with SPI interface.
 - Two USB interfaces.
 - Some sets of buttons, one of which can produce interrupts.
 - Serial ISP download JTAG (the code can also be flashed via USB).
 - SD card with SPI interface.
 - Generic input and output ports (GPIO).
 - A 12 bit ADC and 10 bit DAC.
 - One memory protection unit divided by regions.
 - One real time clock.
 - Interrupt controller.
 - A quadrature encoder interface.
 - System of pulse width modulator (PWM) and other for motor control.
 - An I2C bus and 3 SPI buses.
 - Four timer modules.
 - Four universal asynchronous receiver transmitter buses.
(Reference from: www.hotmcu.com/lpc1768minidk2-development-board-p-55.html)
-
- NXP sponsors the user a manual (UM10360) which can be obtained by typing the following link: <https://www.nxp.com/docs/en/user-guide/UM10360.pdf>
 - The IDE is MDK (Keil uVision4) for the Cortex M3 processor, it can be bought in ARM website.

The project to build in this embedded system consists in a weather station with the capacity to measure some weather conditions such as temperature or air pressure. It will be able to communicate with a computer via UART and it will contain a website reachable via Ethernet interface.

Capítulo 1: Introducción y objetivos.

Introducción.

Una estación meteorológica basada en LPC1768 puede llegar a necesitar un sistema de control medianamente complejo. Pasar de instrucciones máquina que utiliza el procesador de ARM (Cortex M3) a controlar sensores de manera simple y eficaz necesita varias capas de interfaces, esto es, la acción que realizan los sistemas operativos en cualquier tipo de máquina que necesite interacción sencilla con el usuario. Es por ello que definiré más adelante una estructura del sistema operativo que actúa como interfaz, tanto para los sensores como para el usuario, cuyo código estará contenido en memoria flash.

Para empezar, hablaré un poco sobre las características que tiene la placa que vamos a utilizar.

La placa **Mini-DK2** contiene en su interior un procesador de ARM llamado Cortex M3 que funciona a 100MHz de velocidad de reloj. Además, tiene un oscilador de cristal para los periféricos denominado XTAL de 12MHz. A continuación, en la '*Tabla 1.- Características de la placa Mini-DK2*' resumo la gran mayoría de sus características.

| <i>Característica.</i> | <i>Número de elementos.</i> | <i>Notas.</i> |
|---|-----------------------------|---------------|
| 100M interfaz Ethernet | 1 | |
| Interfaz TFT de colores (LCD) con interfaz SPI. | 1 | |
| Puertos de propósito general (GPIO). | 4 x 32 | |
| Interfaz USB. | 2 | |
| Bus I2C | 1 | |
| Bus SPI | 3 | |
| Interfaz UART. | 4 | |
| Reloj en tiempo real. | 2 | |
| Oscilador de cristal. | 1 | 12MHz |
| Módulos temporizadores y de captura. | 4 | |
| Módulo PWM. | 1 | |
| Módulo PWM especializado para motores. | 1 | |
| Acceso directo a memoria (DMA) | 1 | |
| Unidad de protección de memoria. | 1 | 8 regiones |
| Conversor de analógico a digital. | 1 | 12 bits |
| Conversor de digital a analógico. | 1 | 10 bits |
| Memoria RAM. | 1 | 64kB |
| Memoria flash. | 1 | 512kB |
| Interfaz de encoder de cuadratura | 1 | |
| Slot de SD con interfaz SPI | 1 | |
| Interrupciones externas. | 4 | |
| Leds | N/A | |
| Interfaz de carga de código vía JTAG por ISP. | 1 | |
| Controlador de interrupciones anidadas (NVIC). | 1 | |
| Entrada de interrupción no enmascarable (NMI). | 1 | |

Tabla 1.- Características de la placa Mini-DK2.

Para abordar la práctica hablaré de la estructura definida de ese 'sistema operativo' en el **Capítulo 3: Estructura del sistema** y más adelante explicaré cada parte específicamente. Cabe destacar que el código empleado utiliza una estructura jerárquica para controlar el sistema y utiliza variables como señales de comunicación entre diferentes zonas de código.

Más adelante explicaré qué sensores tiene el proyecto, qué miden cada uno de ellos y cómo funcionan a rasgos generales. Todo ello explicado en el **Capítulo 2: Componentes del sistema**.

Objetivos a cumplir.

Los objetivos establecidos para el cumplimiento de la práctica se pueden resumir en la siguiente tabla (Tabla 2.- Objetivos a cumplir):

| <i>Objetivo general.</i> | <i>Objetivos específicos.</i> |
|---|---|
| <i>Crear un menú táctil con el TFT de la tarjeta.</i> | Crear un statechart. Manejar las librerías para la generación de caracteres y zonas en pantalla. Vincular todos los datos y señales de forma coherente. Permitir al usuario modificar variables del sistema. Permitir al usuario interactuar con el sistema. Calibrar la pantalla de manera estática o dinámica. Manejar el brillo de manera manual o automática con el módulo PWM. |
| <i>Controlar un servo motor para la visualización de temperatura.</i> | Ser capaz de alternar dos modos PWM para controlar el servo y el brillo a la vez. Ser capaz de dar valores coherentes a las variables PWM. Interconectar el módulo PWM con los datos. |
| <i>Crear un servidor web accesible vía Ethernet</i> | Manejar las librerías TCP-IP. Configurar variables para realizar una conexión TCP-IP. Escribir con lenguaje de marcado una página web. Crear funciones que interactúen con la página web mediante callbacks. Mantener una conexión TCP. Proporcionar datos a la página web. Recibir datos de solicitudes del usuario mediante la web. |
| <i>Medir la velocidad del viento.</i> | Realizar un análisis físico de cómo realizar las medidas. Realizar un análisis a nivel de electrónica de las medidas. Configurar un módulo temporizador para medir frecuencia de pulsos. Traducir los pulsos a medidas. Conectar los datos recibidos sin que haya interferencias con todo el sistema anterior y posterior, incluyendo gestión de recursos de la tarjeta. |

| | |
|---|--|
| | Examinar si las medidas son precisas. |
| Tener un reloj en tiempo real.* | Configurar el RTC de la tarjeta. |
| | Exportar los datos de manera coherente. |
| | Ser capaz de configurar la fecha, día y hora de manera manual. |
| | Mostrar por el TFT la hora actual. |
| | Conexionar los datos recibidos sin que haya interferencias con todo el sistema anterior y posterior, incluyendo gestión de recursos de la tarjeta. |
| Tener un sistema de control de bloqueo del sistema. | Configurar un Watchdog timer. |
| | Seleccionar una zona del código donde se ejecute con frecuencia, para cualquier estado de funcionamiento, la alimentación del Watchdog sin interferir con el sistema anterior y posterior. |
| | Elegir un valor adecuado para las variables del Watchdog timer. |
| | Examinar que este temporizador no salte de manera espuria. |
| Medir temperatura. | Comprender y manejar el protocolo OneWire. |
| | Comprender documentación en mandarín o en su defecto comprender documentación mal planificada en inglés*. |
| | Configurar un pin para un timer en modo captura. |
| | Gestionar los recursos para no quedarse sin timers ni interferir con el sistema anterior y posterior. |
| | Estructurar y traducir los datos recibidos adecuadamente. |
| Reproducir audio. | Configurar adecuadamente el DAC. |
| | Exportar vía DMA los datos solicitados. |
| | Gestionar la memoria de manera eficaz para tener 3 segundos de audio almacenados en memoria. |
| | Gestionar los recursos para no quedarse sin timers ni interferir con el sistema anterior y posterior. |
| | Configurar adecuadamente otro timer para reproducir el audio guardado en memoria. |
| | Hacer sonar el audio cada vez que una medida supera un umbral. |
| | Establecer dicho umbral para todas las medidas. |
| | Implementar un botón de reproducción del audio cada vez que el usuario lo desee. |
| | Seleccionar las variables adecuadas, así como la frecuencia de exportación del audio. |
| Grabar audio. | Configurar un ADC para tener dos modos de funcionamiento. |
| | Configurar un pin de la placa para leer audio. |
| | Seleccionar adecuadamente las variables, así como la frecuencia de muestreo del audio. |
| | Ser capaz de alternar ambos modos de funcionamiento del ADC sin interferir con |

| | |
|---|--|
| | las demás medidas y ser capaz de bloquearlas mientras se produce la grabación del audio. |
| | Exportar los datos de manera adecuada al sistema. |
| | Entrar en el modo de grabación cuando el usuario lo desee. |
| <i>Medir el brillo.</i> | Traducir de manera adecuada resistencia a voltaje y de voltaje a código del ADC. |
| | Escoger una resistencia de pull-up adecuada para el LDR seleccionado. |
| | Configurar el ADC en modo ráfaga sin que interfiera con la grabación de audio. |
| | Leer el código del ADC para traducirlo a nivel de brillo. |
| | Controlar automáticamente el brillo del LCD en función del dato. |
| | Examinar la validez de las medidas. |
| | Gestionar la frecuencia de muestreo. |
| <i>Medir el índice UV.</i> | Traducir de manera adecuada el código recibido al índice. |
| | Usar el modo del ADC ya configurado para esta medida sin interferir con el sistema anterior y posterior. |
| | Examinar la validez de las medidas. |
| | Exportar los datos al sistema. |
| <i>Medir presión atmosférica.</i> | Crear un controlador que sea capaz de leer registros de un sensor vía I2C. |
| | Examinar la validez de las medidas. |
| <i>Medir posición GPS¹.*</i> | Conseguir leer un sensor GPS vía UART. |
| <i>Conexión WiFi.*</i> | Conseguir conectar un módulo WIFI vía UART a la placa. |
| | Ser capaz de modificar variables. |
| | Comprender los protocolos utilizados para conectar un dispositivo WIFI con el módulo. |
| | Buscar una interfaz adecuada. |
| <i>Conexión UART con un PC.</i> | Conectar el módulo UART del lpc1768 con el controlador de un ordenador convencional. |
| | Modificar variables mediante la conexión serie asíncrona. |
| | Conseguir exportar datos vía conexión seria asíncrona. |

Tabla 2.- Objetivos a cumplir.

Como podemos observar, esta lista de objetivos a cumplir de manera obligatoria para el funcionamiento del sistema, contiene todo lo que debemos de realizar a grandes rasgos y será utilizada como guía para los siguientes capítulos.

¹ Los objetivos marcados con * representan objetivos opcionales pero que incluyen en su implementación modelos marcados como obligatorios.

Posibles mejoras al sistema en un futuro.

Pese a que los objetivos generales son muy completos, los objetivosopcionales son sólo suplementarios y añaden funciones extras a esta estación meteorológica para hacer pequeñas mejoras del sistema y que sea aún más completo. No tabularé los objetivos opcionales, que no prometo para nada cumplir, pero que puede que unas versiones futuras del proyecto aparezcan, son los que enumero a continuación y evalúo su complejidad:

- Añadir memoria extra con la interfaz SPI en la tarjeta SD.
(Complejidad media)
- Añadir funcionalidad a la página web para modificar más variables del sistema.
(Complejidad baja)
- Hacer un seguimiento histórico de los datos.
(Complejidad baja)
- Exportar los datos a la comunidad.
(Complejidad media)
- Mejorar la interfaz TFT.
(Complejidad baja)
- Comunicarse con más estaciones del mismo tipo a gran distancia.
(Complejidad alta)
- Realizar los objetivos marcados como * en el apartado anterior.
(Complejidad media)

Capítulo 2: Componentes del sistema.

Mini-DK2

El componente fundamental de sistema es la tarjeta antes mencionada en el *Capítulo 1: Introducción y objetivos a cumplir*. Será la unidad de control y todos los sensores y aparatos electrónicos del sistema se conectan a este componente. Su función es ejecutar el sistema operativo antes mencionado en el Capítulo 1 y proporcionar control sobre todo el sistema. En la *Figura 1* podemos ver el esquema de pines de la tarjeta con la que vamos a trabajar. En la *Figura 2* podemos ver una imagen de la tarjeta.

Si desea ver las características generales de este componente puede recurrir a la Tabla 1 o si desea más información, puede recurrir a su fabricante; NXP.

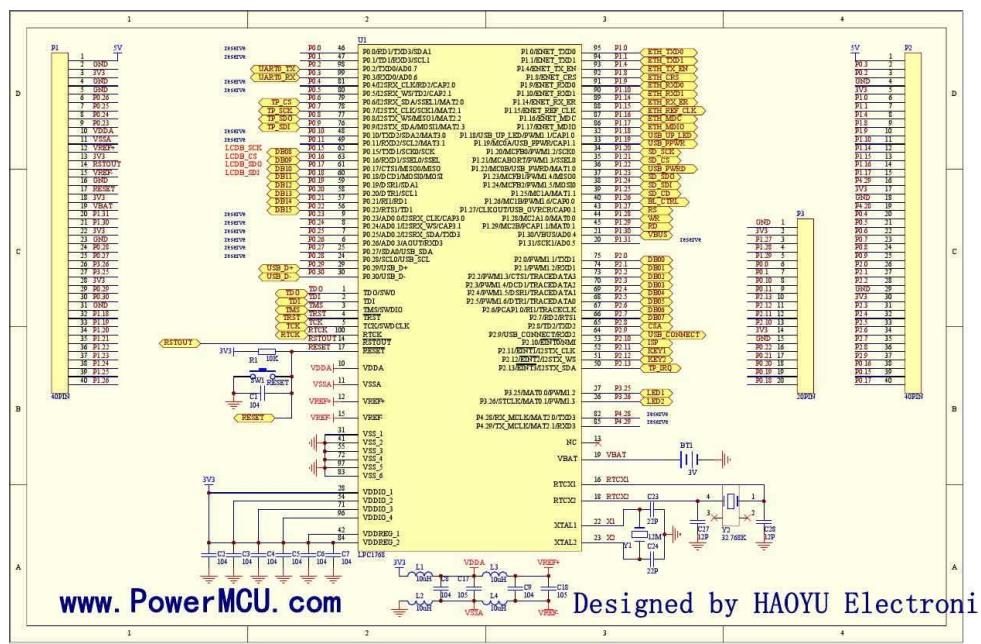


Figura 1.- Esquema general de la tarjeta Mini-DK2.

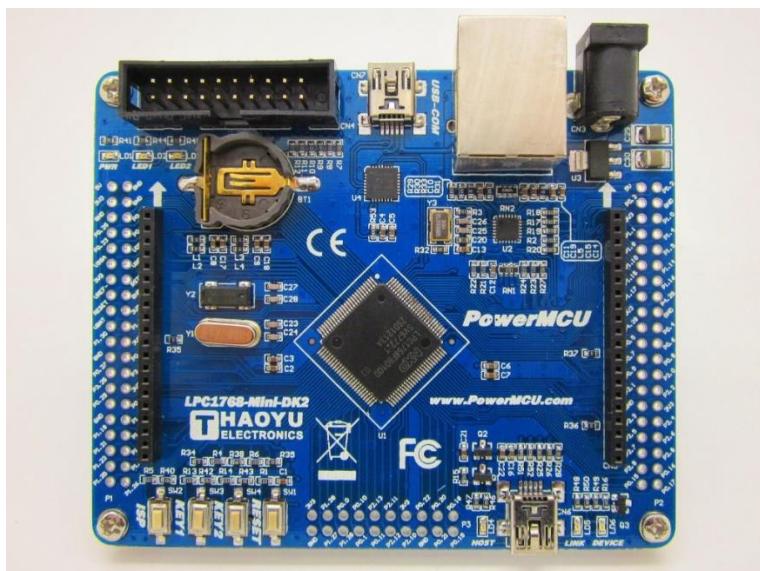


Figura 2.- Imagen de la tarjeta Mini-DK2.

Anemómetro.

Este componente nos permitirá medir la velocidad del viento. Se trata de un eje rotatorio conectado a tres cuencas de baja densidad que forman un ángulo de 120° entre ellas y permiten rotar sobre el eje si existe viento.

Tiene dos pines de salida que son cortocircuitados por zonas cada 90° , por lo que cada vuelta se cortocircuita dos veces.

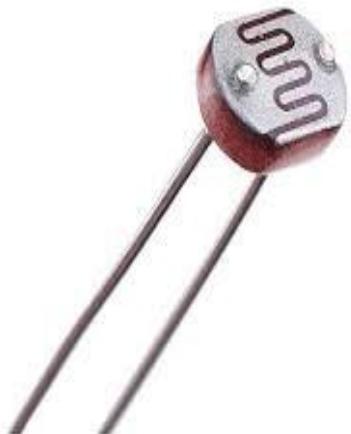
Su funcionamiento consiste en generar cortocircuitos entre los dos pines de salida a medida que el eje va rotando, por lo que podemos generar pulsos cuadrados a medida que el eje rota si lo conectamos adecuadamente.

En la siguiente figura, *Figura 3* podemos ver una imagen de este anemómetro del cuál no he encontrado documentación.



Figura 3.- Anemómetro.

LDR.



Este componente nos permitirá medir la cantidad de brillo que hay en la zona de manera que este varía la resistencia entre sus dos patillas en función de la luz que recibe. Esto sigue una relación lineal con una tolerancia entre los LUX en el ambiente y la resistencia entre ambas patillas. Conectándolo adecuadamente podemos traducir mediante un divisor resistivo y escogiendo un valor de pull-up adecuado esta resistencia nos dará los LUX que mida.

En la *Figura 4* incluyo una imagen del LDR y en la *Figura 5* una gráfica en la mejor calidad que he encontrado que representa la relación resistencia-LUX del componente.

Figura 4.- LDR.

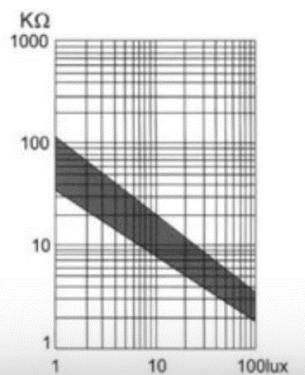


Figura 5.- Gráfica R-LUX

Sensor UV.

Este sensor nos permitirá medir el índice UV que reciba, este sensor traduce el índice UV que recibe a voltaje directamente sin necesidad de aplicar un circuito conversor resistivo como en los dos casos anteriores, en cambio, viene en forma de tarjeta pequeña con 3 pines de interés:

- Alimentación (+3.3V)
- Masa. (-0.0V)
- Salida de voltaje. [0, 3.3V]

Este sensor mide la índice UV y saca un índice del 0 al 10 en forma de la siguiente ecuación:

$$V_o = \frac{V_{cc} \cdot IndiceUV}{IndiceUVmax}$$

En la Figura 6 se muestra el sensor UV a utilizar (VMA30A)

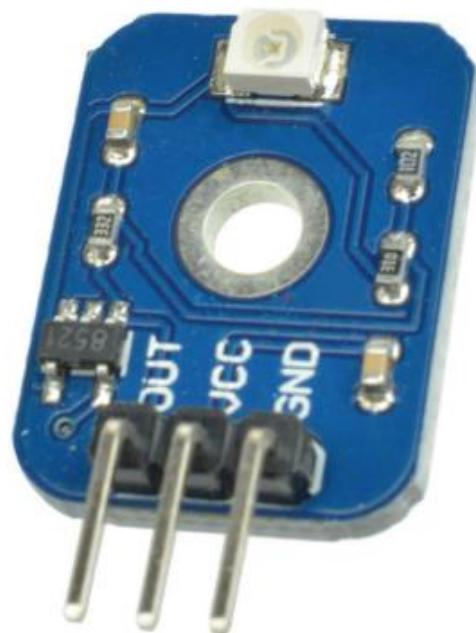


Figura 6.- Sensor UV30A.



Servo motor.

Este componente nos permitirá representar la temperatura que hay entre dos valores límite a elegir. El mínimo significará izquierda, mientras que el máximo de esa temperatura será a la derecha. Este servo controla el ángulo de posición en función de una señal modulada por pulso (PWM) y, en función de su ciclo de trabajo o tiempo a nivel alto, tendrá un ángulo u otro. En la Figura 7 aparece una imagen del servo motor a utilizar (SG90) y en la Figura 8 una gráfica del fabricante que sólo va a servir para identificar un problema que surge a raíz de utilizar la tarjeta Mini-DK2 con este servo.

Figura 7.- SG90, servo.

Como podemos observar en la Figura 8 la señal PWM generada ha de tener un valor a nivel alto de +5.0V. Es por ello que esta gráfica sólo nos va a dar una referencia del periodo de la señal a generar, pero no del tiempo en alta, dado que estos servos se guían por la potencia recibida que depende de la amplitud del pulso y el voltaje. Habrá que hacer una corrección en potencia del pulso.

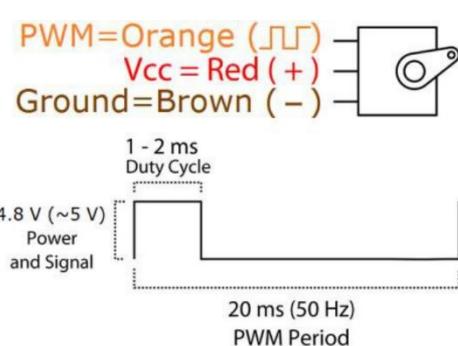


Figura 8.- Pulso a recibir.

Amplificador + altavoz.

Se utilizará un altavoz de 8Ω conectado a un amplificador de señal adaptado de 4 a 8Ω (PAM8320A) que contiene el siguiente patillaje:

- Vdc - +5.0V.
- GND - -0.0V.
- Shutdown - N/C.
- Audioin+ - Audio.
- Audioin- - GND.

Esta configuración permite pasar de una señal eléctrica de audio a audio en sí. En la Figura 9 muestro una imagen del amplificador soldado al altavoz de 8Ω .

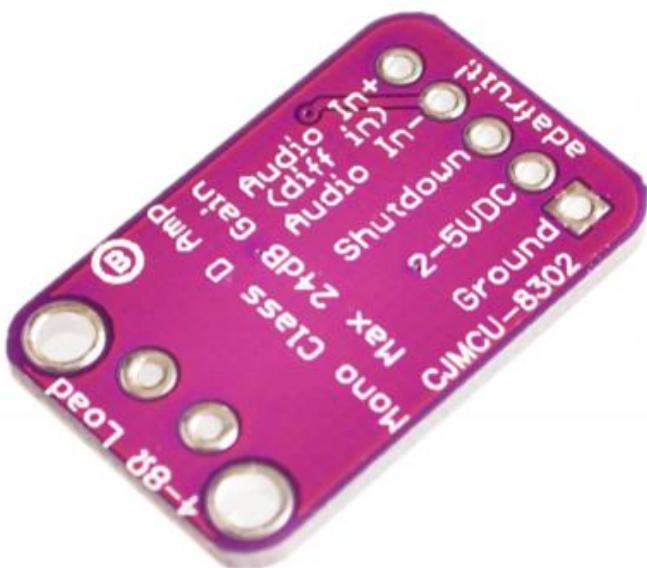


Figura 9.- PAM8320A + Altavoz

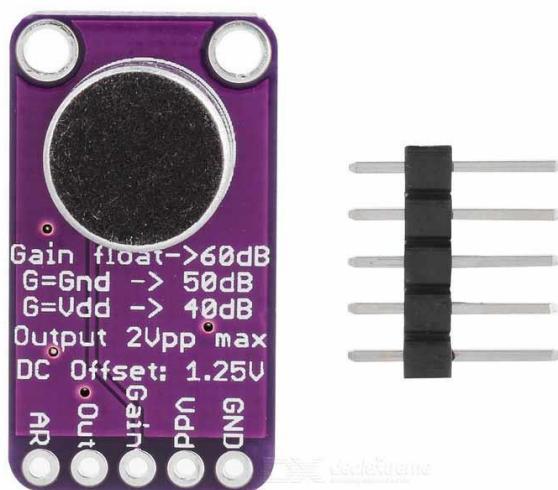


Figura 10.- MAX9814, micrófono.

Se podría configurar para que obtuviese ganancia personalizada, pero no conectar da una ganancia fija también que no tendremos en cuenta dado que nuestro objetivo no es mejorar la calidad de audio, sino simplemente tener una señal de audio. Lo mismo pasa con el pin AR, que fija el factor de compresión del audio; no conectarlo implica que tenga un valor fijo, dado que ambas entradas al aire no forman efecto de alta impedancia.

En la figura 10 tenemos una imagen de dicho micrófono (MAX9814) con parte de los valores más significativos de la datasheet impresa en la tarjeta.

Micrófono.

Este micrófono (MAX9814) es un micrófono de alta calidad y bajo coste que nos permitirá hacer vibrar la señal continua con un offset de +1.25V (amplitud máxima de +2Vpp) en función del audio recibido. Directamente podemos convertir la señal por el pin de salida con la siguiente configuración del patillaje:

- AR - N/C
- Gain - N/C
- Out - Salida.
- Vdd - +3.3V
- GND - -0.0V

Sensor de temperatura.

El sensor que medirá la humedad y la temperatura en interiores a utilizar será el DHT22. Este sensor pese a tener 4 patillas, una no hay que conectarla y la otra utiliza el protocolo OneWire. Las otras dos son de diferencia de potencial para alimentación. El conexionado es el siguiente, siendo ordenados los pines de izquierda a derecha:

- Pin 1 - +5V Vcc.
- Pin 2 - OneWire.
- Pin 3 - NC
- Pin 4 - 0V GND.

Cabe destacar que este sensor requiere de una resistencia de pull-up de $4k1\Omega$.



Figura 11.- Sensor DHT22.



Figura 12.- Módulo BMP180.

Sensor de presión.

El sensor a utilizar es un módulo que incorpora el sensor BMP180, que funciona por protocolo I2C.

Cabe destacar que el módulo contiene un regulador y las resistencias necesarias de pull-up para el funcionamiento de los pines SDA y SCL del protocolo. El conexionado del módulo es el siguiente:

- GND - -0.0V
- Vcc - +5.0V
- SDA - Datos I2C.
- SCL - Reloj I2C.

Este sensor utiliza un driver software que puede ser extraído del fabricante, pese a eso, se ha desarrollado un driver específico para estación meteorológica utilizando los mismos algoritmos que utiliza el driver del fabricante para leer los registros por I2C del sensor.

TFT.

Se utilizará un display táctil (TFT), para poder mostrar y recibir por pantalla datos relativos a la interfaz del usuario y variables ajustables del sistema. El panel se llama HY32-B y utiliza 8 bits para comunicarse. El conexionado de los 10 bits, los 8 anteriormente mencionados y dos de control se muestra a continuación:



Figura 13.- Panel TFT-LCD.

| Puerto / pin utilizado. | Función que desempeña. |
|-------------------------|-------------------------------|
| P0.6 | Touchpannel - CS |
| P0.7 | Touchpannel - SCK |
| P0.8 | Touchpannel - SDO |
| P0.9 | Touchpannel - SDI |
| P0.15 | LCD-SCK |
| P0.16 | LCD-CS |
| P0.17 | LCD-SDO |
| P0.18 | LCD-SDI |
| P1.26 | Control del brillo por PWM. |
| P2.13 | Interrupción del touchpannel. |

Tabla 3.- Conexionado del TFT.

Capítulo 3: Estructura del sistema.

Durante este capítulo nos dedicaremos a analizar cómo he estructurado de manera general el sistema y cómo pretendo hacer funcionar y gestionar los recursos de la placa para que no interfieran entre ellos.

Cabe mencionar que este es un análisis que hay que hacer desde un principio y en mi caso no lo he podido hacer debido a que he tenido que añadir las cosas por partes separadas sin conocer el funcionamiento de las posteriores. Por ello, he tenido que aplicar la estrategia de: *la utilización de los menores recursos posibles para que funcione y ya añadiré más recursos si me sobran y si quiero mejorar la calidad*, cosa que si pudiese haber evitado lo hubiese hecho.

0. Esquema general.

Mi esquema se basa en ramificar jerárquicamente en torno a la interfaz de usuario, es decir, el menú, los demás módulos de control del sistema según la Figura 1.

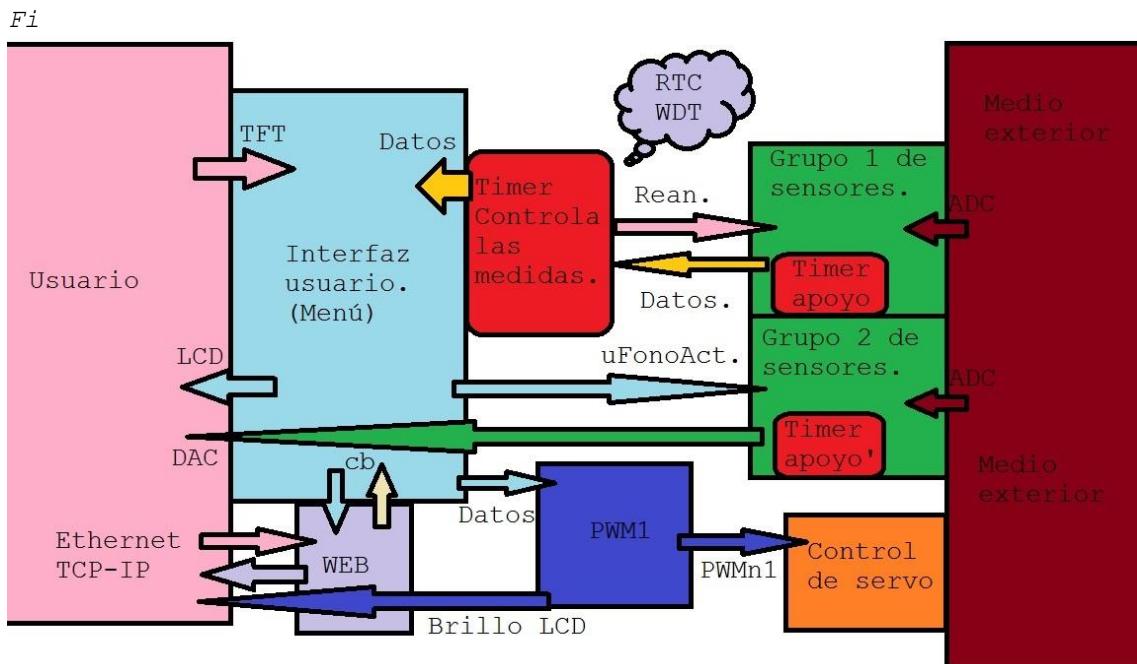


Figura 1.- Esquema de primera versión.

Cabe destacar que el esquema de la Figura 1 es la primera versión. Es decir, sin que exista conexión UART posible ni WiFi. Añadiré una versión así más adelante que únicamente añade esos dos módulos.

El esquema en general consta de módulos que se comunican todos con la interfaz de usuario indirecta o directamente. Por eso que es un requema ramificado a partir de la interfaz de usuario (que actúa como sistema de control). También llama la atención un timer que controla las medidas. Esto lo explicaré en la sección de este mismo capítulo *Timer controlador de medidas*.

A grandes rasgos, existe un timer que controla las medidas del grupo 1, es por eso que he separado en dos grupos los sensores: los controlados por timer y los controlados por la interfaz de usuario; grupo 2. Evidentemente, los del grupo 2 son los correspondientes a grabar y

reproducir audio y los demás corresponden al grupo 1. El tiempo de muestreo es, por tanto, marcado dentro del timer que controla esos sensores para el grupo 1 y constante para el grupo 2 de sensores. Todos ellos leen del medio exterior los datos con el único ADC de la placa que consta de varios canales multiplexados.

El usuario interactúa con el interfaz de usuario mediante el TFT y el LCD, además es posible comunicarse mediante UART y mediante Ethernet (TCP-IP).

1. PWM1 y control de servo.

El módulo PWM se utiliza para controlar un servomotor que se encargará de monitorizar las medidas de temperatura y presión del sistema. Los datos son almacenados en memoria y el sistema se encarga de, cada vez que se leen los sensores, actualizar el módulo PWM para enviar un pulso de manera adecuada para que el servo se posicione en función de unos valores máximos y mínimos que pueden ser modificados por el usuario en todas sus interfaces.

La estación meteorológica utiliza el servo expuesto en el capítulo 2 y lo actualiza el manejador de interrupción del Timer0 del LPC1768.

2. WEB.

La interfaz web utiliza una pila de protocolos y librerías TCP-IP para proporcionar los recursos que permiten montar un servidor WEB integrado en el LPC1768. Mediante un compilador de CGI compila un archivo con extensión de formato CGI que incorpora un lenguaje de marcado HTML, para configurar la interfaz de usuario WEB. Más adelante se explica el funcionamiento.

3. Usuario.

El usuario puede interactuar con las diferentes interfaces que se presentan para leer los datos de la estación meteorológica. Además, este usuario puede modificar mediante todas sus interfaces los valores modificables del sistema.

- Interfaz WEB: Interfaz mediante un servidor WEB sobre un cable ethernet, se puede acceder mediante un navegador y el usuario puede modificar las variables del sistema excepto la hora actual.
- Interfaz UART: Interfaz mediante el protocolo de comunicación UART sobre un cable USB, se puede acceder mediante una aplicación que monitorice la conexión serie asíncrona como Termite (por ejemplo).
- Interfaz TFT-LCD: Interfaz mediante el protocolo de comunicación SPI sobre los pines de la placa MiniDK-2. El usuario lee por la pantalla del LCD la interfaz creada en el programa mediante la iluminación del array de pixeles. Además, el usuario puede modificar las variables del sistema desde la pantalla.

Las variables que el usuario puede modificar son: máximos y mínimos valores de alarma y presión. Si se superan, suena una alarma. La hora puede ser modificada, también el umbral de brillo, que es un valor expresando en segundos de el tiempo que permanece encendida la pantalla en el modo ULP². Otra variable a modificar es la selección del servo, es decir, qué variable (si presión o temperatura) es representada.

² Ultra Low Power Mode. Ver manual de usuario.

4. Medio exterior.

El medio exterior se utiliza para obtener los dartos, los sensores utilizan el medio exterior para obtener valores que la estación meteorológica lee mediante los diferentes protocolos.

5. Timer controlador de medidas.

Se utilizan varios módulos temporizadores para apoyar a los sensores. A continuación, se exponen las funcionalidades de cada uno de los timers si se utilizasen:

- Timer 0: Controla las medidas, marca el tiempo de muestreo de cada uno de los sensores y se ocupa de actualizar el módulo PWM cada vez que interrumpe, en caso de haber cambiado un valor máximo o mínimo.
- Timer 1: Este Timer cumple **tres funciones**: la primera es utilizar el modo capture para medir los pulsos del **anemómetro**; la segunda es utilizar un MR (Match Register) para desactivar el **DAC** una vez haya acabado de reproducir audio; y la tercera es utilizada en caso de **grabar audio**, tener un MR que sirva para activar las cuentas.
- Timer 2: No usado.
- Timer 3: Sirve de apoyo al protocolo OneWire (actúa de contador).

6. Grupo de sensores 1.

Definimos como el primer grupo de sensores al grupo perteneciente al sensor BMP180 y DHT22, debido a que al tener incorporadas esperas activas es recomendable medirlo cuanto menos. Cabe recalcar que estos sensores miden variables que no cambian con gran velocidad, por lo que pueden ser medidas cada 5 segundos incluso más si se desease ajustar. Estas variables son la temperatura, la presión y la humedad. En caso de querer medirlas más rápido, cambiar el símbolo CsCAP definido en el archivo Systemsymbols.h y poner un valor de cuentas adecuado para la función que se deseé desempeñar.

7. Grupo de sensores 2.

Definimos como segundo grupo de sensores al grupo perteneciente a los sensores LDR y UVA. Debido a que necesitan un tiempo de muestreo más regular, la luz y el índice UV pueden cambiar rápidamente, por lo que es aconsejable medirlo con más frecuencia. Si se desease modificar el valor del tiempo de muestreo, referirse a CsADC definido en el archivo Systemsymbols.h y poner un valor de cuentas adecuado para la función que se deseé desempeñar.

8. WDT.

El WatchDogTimer es un contador regresivo que si se deja llegar a 0 provoca un reinicio del sistema. Así, si el sistema es bloqueado, el WDT lo desbloquea reiniciando el sistema. Cabe destacar que la hora debe de ser ajustada de nuevo y todas las variables que el usuario haya modificado.

9. RTC.

El RTC es un reloj en tiempo real que sirve para tener un reloj en la estación meteorológica. Es un contador que se utiliza para mostrar por la página WEB y por pantalla la hora actual de las medidas.

10. UART.

Es un módulo que se encarga de transmitir y recibir información por el protocolo de comunicación seria asíncrona universal. Sirve como interfaz para que el usuario pueda recibir información de la estación y mandar comandos a la misma.

11. ADC.

El módulo ADC, es un módulo integrado en la placa que tiene la capacidad de codificar señales analógicas y convertirlas a un formato digital, esto permite leer señales analógicas, en nuestro caso, el sensor LDR y el UVA.

12. DMA + DAC.

Estos módulos actúan unidos. El DMA se encarga de liberar carga computacional al procesador, transfiriendo el valor de los datos del audio al siguiente módulo, el DAC, mediante una señal de inicio de conversión. Mientras que el DAC es un módulo que genera una señal analógica en función de un valor digital recibido. El funcionamiento reside en activar el DMA cuando se deseé producir audio y desactivarlo cuando este termine, produciendo las transferencias de datos con el DMA y no con el procesador.

Se recuerda que el Timer encargado de desactivar la transferencia es el Timer 1.

Capítulo 4: Implementación del sistema.

En este capítulo veremos cómo se implementan tanto en software como en hardware los sistemas y módulos anteriormente mencionados en el *Capítulo 3: Estructura del sistema*. Intentando resolver tantos objetivos como nos sea posible a lo largo del capítulo. Además, se explica el fundamento de la solución adoptada.

En el Anexo I se incluye un esquemático de todo el **conexionado hardware del sistema**, cumpliendo las características del *Capítulo 2: Componentes del sistema*.

En el Anexo II se incluye el **código fuente**. Para leer las descripciones de las funciones y variables, leer en el código comentado de las secciones.

Carpeta MiniDK-2.

En esta carpeta se incluyen todas las carpetas del proyecto.

Carpeta README.

Contiene información del proyecto.

Carpeta Startup.

Contiene los ficheros de inicialización del sistema, así como librerías internas que utiliza el LPC1768 para iniciar. Incluyen las librerías:

- Startup LPC17XX.s: Inicializa el sistema. La única variable modificada en este archivo es Stack_Size y se le ha otorgado un valor de 0x200.
- Core cm3.c: Inicializa el core.
- System LPC17XX.c: Ajusta variables y define macros internos sobre todo de los relojes de la placa MiniDK-2.

Carpeta Principal.

Contiene el programa principal en su interior, dentro del archivo main.c. En el programa principal se crean variables globales y punteros a dichas variables globales, para utilizarlas de manera más sencilla.

Dentro del programa principal sólo es necesario llamar a la función de configuración del sistema. Luego es necesario crear un bucle infinito para llamar a la función mainLoop() que está definida en Statechart.c y a la función mantenerTCP() que se encuentra en el archivo HTTP_SOURCE.c. La función mainLoop() es la máquina de estados que rige la interfaz de usuario por el LCD y mantenerTCP() mantiene la conexión TCP mediante la librería de TCP haciendo un llamado a la función main_TcpNet().

Cabe destacar que no todas las variables importadas se encuentran en este archivo, dado que sólo residen las de mayor importancia como pueden ser los datos almacenados y señalizadores del sistema.

Carpeta Menu.

En esta carpeta se encuentra la librería TouchPanel.c y el archivo responsable de generar la máquina de estados, Satatechart.c. En la Figura 1. Podemos observar la máquina de estados implementada en el archivo. La función `_mainLoop()` se encarga de generar la máquina de estados, utilizando la variable global `ESATADO` como una variable de 8 bits que almacena el valor del estado que se quiere asignar. En este archivo se declara la variable `MODIFICABLES`, que contiene todos los valores modificables por el usuario en cualquiera de sus interfaces mencionados anteriormente.

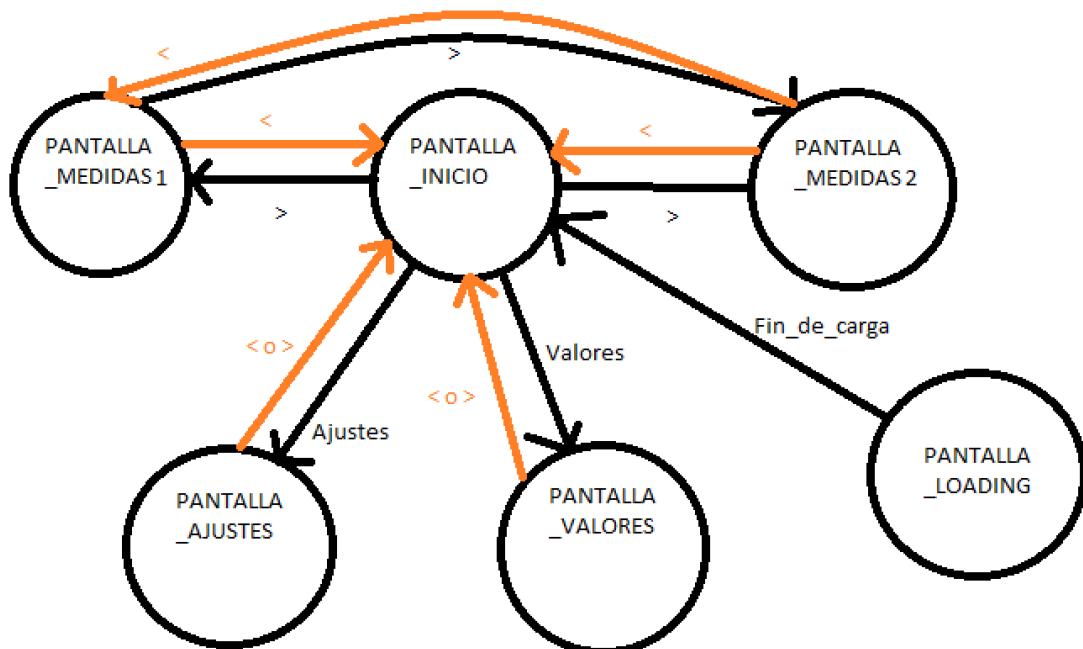


Figura 1.- Modelo de pantallas.

Además, se definen variables como `Modo_brillo`, `Modo_energetico` que guardan el estado del brillo, es decir, cualquiera de sus 4 niveles de brillo y el modo de ahorro de energía que se encuentra el sistema. La variable `pressedTouchPannel` refleja si se ha tocado la pantalla, para que se ilumine cuando esta, en su modo de ultra ahorro de energía; `ULPM`, se haya apagado.

Las variables internas `_brilloFade` y `_brilloAuto`, son variables que guardan si el sistema debe de apagar la pantalla o no (`_brilloFade`) y si deben de leer el LDR para proporcionar un control automático de brillo (`_brilloAuto`). El sistema cada iteración de la función principal, comprueba si `_brilloAuto` con ayuda del SysTickTimer cada 100ms si el estado es de brillo automático y actualiza el módulo PWM para proporcionar el pulso adecuado para generar un nivel de brillo que oscila entre el 20% y el 80% de luminosidad. Esto lo hace refiriéndose a la tabla en el archivo `LUT.c`, contretamente en el array `Brillo2ciclo_LDR`, que traduce el brillo al porcentaje del ciclo de trabajo del módulo PWM1.

Se definen variables de tipo `zona` que son variables que necesita la librería para generar la interfaz. Se le pasa como parámetro el `exe X e Y` y la longitud en `X` y en `Y` que ocupa el recuadro generado. Además, dichas variables admiten color y texto tanto como de la línea como del texto.

En el archivo se encuentran las siguientes funciones:

- _pintaX_(): pinta la pantalla X.
Dentro de estas funciones puede haber más estados que representen un tipo de color, por ejemplo, existen unas barras de colores que son interfaces gráficas para que el usuario sepa la temperatura que hace en función de los valores máximos y mínimos de la misma, cambiando el color en función del porcentaje.
Dentro de estas funciones se utilizan actualizadores, estos actualizadores sirven para actualizar el valor de los datos, es decir, mostrarlos por pantalla una vez sean medidos. Cuando un módulo termine de realizar una medida, este módulo debe de hacer un toogle al actualizador correspondiente para indicarle a la máquina de estados que se ha realizado un cambio en el valor de la medida y así poder mostrarla por pantalla.
- _configuraLCD_(): configura el LCD.
- squareButton(): Dibuja un botón cuadrado en función de la zona, los colores y un texto.
- squareBox(): Dibuja una caja en una zona de un color.
- checkTouchPanel(): Comprueba si se ha tocado un botón, además si se ha tocado cualquier zona de la pantalla, resetea el contador que apaga la pantalla en ULPM.
- ZoneNewPressed(): Comprueba si se ha tocado cierta zona de la pantalla, sirve para saber qué botón se ha pulsado. Si se toca la pantalla cuando esta ha sido apagada (esto sucede dentro del manejador del SysTickTimer) esta se recupera con un ciclo del 60% de PWM, es decir al 60% del brillo. Además, recupera las variables anteriormente mencionadas y resetea el contadorLuz, que es el contador encargado de controlar que no se supere el tiempo de apagar pantalla (este tiempo es modificable).

Dentro de cada pantalla, podemos visualizar o modificar variables.

- En la pantalla de medidas 1: Podemos ver los siguientes valores.
 - Velocidad del viento en m/s.
 - Humedad del aire en % relativo.
 - Claridad recibida en LUX.
 - Índice UV en UVs.
 - La altura respecto al nivel del mar en metros.
- En la pantalla de medidas 2:
 - Temperatura en °C.
 - Presión en mBar.
 - Gráfico de barras horizontales de colores para los valores máximos y mínimos de cada una.
- En la pantalla de ajustes: Podemos visualizar la IP del servidor WEB y además modificar las horas, minutos y segundos del reloj, además del día.
- En la pantalla de valores: Podemos modificar los valores de presión y temperatura máximos y mínimos que generan alarma, además de poder modificar si el servo representa la temperatura o la presión. El botón tendrá letras verdes para la presión y letras rojas para la temperatura. En dicho botón se representan los rangos máximos y mínimos de temperatura y presión a representar.

- En la pantalla de inicio: Podemos, además de acceder a otras pantallas, modificar el brillo, siendo 1 el mínimo y 4 el máximo. Además, está la selección de A que es brillo automático, lo que automáticamente le hace cambiar a LPM (Low power mode). También podemos grabar y reproducir audio con los botones de load y play, que lanzan las señales de grabar y reproducir audio. Desde esta pantalla podemos modificar el modo de ahorro de energía.

Importante:

Dentro de la librería TouchPanel.c, podemos encontrar una matriz denominada como matrix. Esta matriz contiene unos valores de calibración para calibrar la pantalla, cada una tiene sus valores y deberían de ser ajustados por el usuario. En caso de querer calibrar manualmente la pantalla cada vez que se inicia el sistema, descimentar la línea 68 del archivo configura.c, donde dice:
//TouchPanel_Calibrate();.

Carpeta Setup.

En esta carpeta se encuentra el archivo de configuración configura.c. Se encarga de mandar a pintar las diversas pantallas de carga, de iniciar las variables y de mandar a configurar todos los módulos. Además, inicia el módulo PWM con el 90% de ciclo para el servo y 50% de brillo. Inicia las variables a cero y las modificables con el valor de las macros.

Temperatura: [-10, 50] °C.

Presión: [500, 1500]mBar.

Tiempo de brillo: [10]seg.

Variable medida: Temperatura.

Carpeta I2C.

En esta carpeta hay una librería de I2C (I2Clib.c) que contiene todo lo necesario para realizar las llamadas a las funciones de comunicación de I2C. (Mandar un byte, recibir un byte, recibir una dirección). Luego está el archivo que se encarga de utilizar la librería de I2C para leer el sensor BMP180.

En el archivo I2C.c podemos encontrar funciones que se encargan de usar la librería para leer los registros de acuerdo con la datasheet del fabricante (ver la hoja de datos de BMP180). Utiliza la lectura de ciertos registros del sensor para calibrarlo y luego llama a medirBMP(), que utiliza un algoritmo que leyendo ciertos registros y trabajando con ellos podemos obtener la temperatura y la presión. Cuando se desee medir, solo hace falta llamar a medirBMP() para que automáticamente guarde en la variable DATOS la presión y temperatura. Además, se calcula la altura sobre el nivel del mar con una fórmula proporcionada por el fabricante.

Nota: La función procesarDato() admite parámetros que en versiones anteriores servían para elegir presión o temperatura, en la versión final se incluyen la altura, temperatura y presión. Se ha decidido tomar

la temperatura con este sensor, dado que es de mayor calidad que el DHT22.

Carpeta Anemómetro.

En esta carpeta se encuentra Anemometro.c que incluye dos funciones, una de configuración y otra de medición. El anemómetro funciona de la siguiente manera: necesita una resistencia de pull up (vale la configuración estándar de la placa), dado que cada cuarto de vuelta del ciclo de giro cortocircuita ambas patillas, por lo que la conexión debe de ser input-masa, la resistencia de pull up debe de estar en el lado de input. Cada vuelta genera 2 pulsos, dado que cortocircuita en dos cuartos de ciclo, por lo que el anemómetro genera 2 pulsos por vuelta.

Dicho esto, podemos utilizar el módulo CAPTURE del Timer 1 para calcular la diferencia de tiempos entre el inicio o fin de ambos flancos (subida o bajada a elegir). Podemos calcular la velocidad que recorre el viento de la siguiente manera:

$$Distancia_{recorrida} = Perímetro_{anemómetro} = \pi \cdot D_{anemómetro}$$

$$Tiempo_{medidoCapture} = \frac{Perímetro_{anemómetro}}{Pulsos_{vuelta} \cdot Velocidad_{viento}}$$

Podemos esperar a que se realicen dos pulsos y luego medir como si se hubiese realizado un pulso para reducir carga computacional:

$$Velocidad_{viento} = \frac{Perímetro_{anemómetro}}{Diferencia_{pulsos}} = \pi \cdot \frac{D}{T_{reloj} \cdot (CAPTURE_x - CAPTURE_{x-1})}$$

Medidas realizadas estiman el diámetro del anemómetro en **14 centímetros**.

Carpeta ADC.

En esta carpeta están los archivos necesarios para hacer funcionar todo el módulo del ADC y leer los sensores LDR y UVA30A. Además, se encuentra la lectura del micrófono. Los archivos inherentes son Ldr.c, uFono.c y UVA30A.c.

En el archivo LDR.c contiene toda la configuración del ADC, que se ejecuta en modo BURST. Dicha configuración realiza conversiones a todos los canales activados, que en este caso es el del LDR y el del UVA, podrían haberse metido más sensores sin ningún problema y sin aumentar la carga computacional, solo que se utilizan más pines del ADC en este modo BURST. El valor de CLKDIV es el máximo que se puede poner (0xFF), por dejar tiempo al ADC a que convierta con tranquilidad, podría reducirse dicho valor hasta cierto punto.

La interrupción que hace leer los registros del ADC es la penúltima, dado que hay un 'lag' de una conversión desde que se muestrea hasta que se produce la llamada al Handler del ADC. Cuando el Handler entra en acción, el último canal ya ha sido convertido. El penúltimo canal es del del LDR (1), por lo que la interrupción la provoca este fin de conversión.

En el archivo UVA30A.c sólo se configura el ADC para permitir un canal más en modo BURST, si no se ha configurado el LDR, el configurador del

UVA30A llama al del LDR primero, dado que hay que configurar todo el ADC primero para configurar que el UVA entre en modo BURST. El canal asociado al UVA es el 2.

Una vez teniendo las medidas en modo BURST, cuando se deseé grabar el micrófono, se lanzará una señal que lo indique. Esta señal es la función lanzaUFONO(), que configura todo el ADC y el Timer 1 para que se obtengan muestras cada MATCH0, pare la conversión de audio y vuelva a medir en modo BURST. Cabe destacar que las medidas se bloquean mientras el modo audio está activado. Este bloqueo lo representa la variable YaPuedesMedir, que además se utiliza en el Handler del ADC para saber en qué modo (si BURST o audio) se encuentra el sistema. Una vez alcanzado el número de muestras del audio, se lanza la desactivación del ADC en modo audio y se recupera el contexto del ADC para el modo BURST, además de activar los actualizadores y de marcar que ya se puede medir en modo BURST.

La configuración del micrófono configura el ADC para que cada 8000kHz se obtenga una muestra de audio y se guarde en la variable AUDIO, que ocupa un total de 8 bits por muestra a 16000 bytes. 16kB de audio.

Dentro del manejador de la interrupción del ADC podemos encontrar dos modos:

- YaPuedesMedir = 0: Mete las muestras recibidas por el canal 0 al array definido para el AUDIO, cuando se alcanza el número de muestras MUESTRAS_AUDIO, se reconfigura el ADC para medir en modo BURST.
- YaPuedesMedir = 1: Lee los canales 1 y 2 del ADC por cada interrupción.
- Para obtener el brillo: Se lee el canal 1 del ADC, se calcula el porcentaje relativo de voltios del canal referido a 3.3V; luego se calcula su resistencia y se pasa dicho valor a una look-up-table para traducir dicho valor a LUX.

$$R_{LDR} = R_{pull} \cdot \frac{\frac{ADC_1}{0xFFFF}}{1 - \frac{ADC_1}{0xFFFF}}$$

- Para obtener el índice UV: Es una función lineal, por lo que se obtiene el porcentaje relativo respecto a 3.3V del canal 2 de entrada y se multiplica por el máximo.

$$Indice_{UV} = Indice_{maximo} \cdot 3.3V \cdot \frac{ADC_2}{0xFFFF}$$

La resistencia de pull-up escogida son 70kOhm, debido a que linealiza mucho su respuesta. Es muy importante escoger una resistencia de pull-up adecuada, debido a que una pequeña variación en el error podría producir una gran variación de la resistencia. A continuación, gráficas que muestran dicho comportamiento.

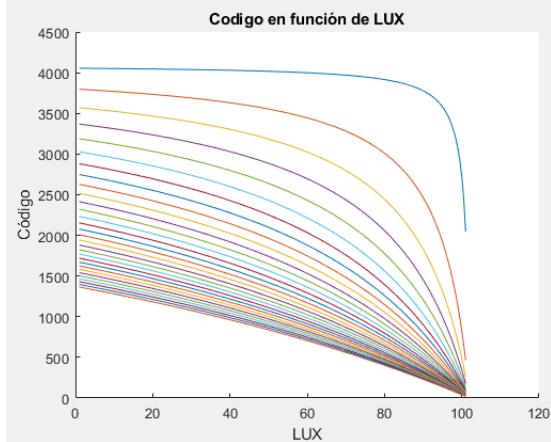


Figura 2.- Código en función de LUX.

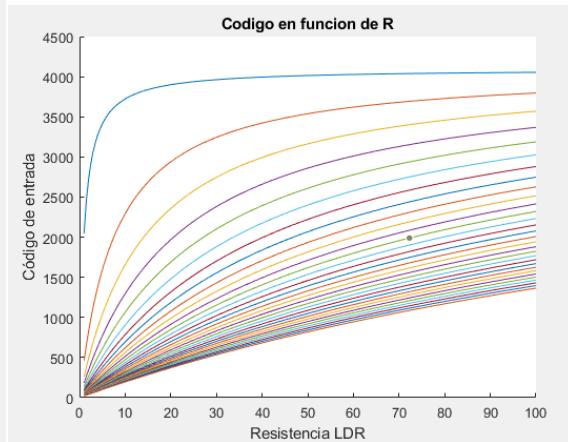


Figura 3.- 2 En función de los ohm.

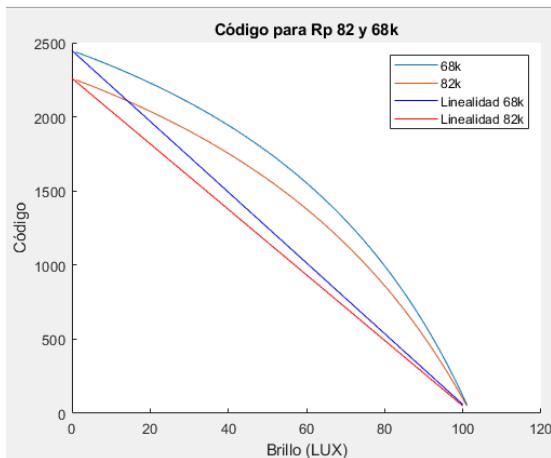


Figura 4.- Linealidad a 68 y 82k.

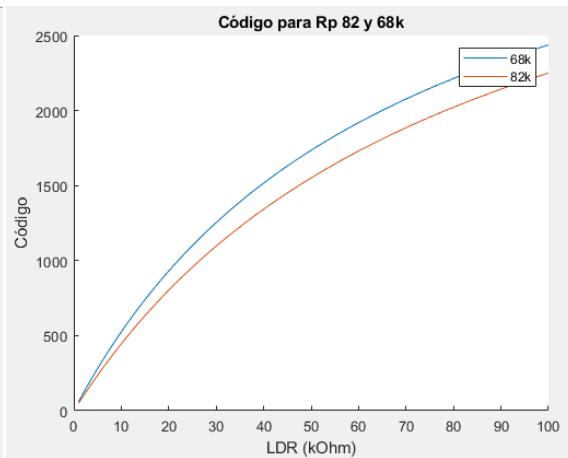


Figura 5.- 4 En función de los ohm.

Se han escogido valores de 68kohm debido a que son valores estándar y tienen una buena respuesta en la linealidad sin sacrificar mucho porcentaje del código que se ha recibido. Perdemos la mitad del rango, que podría arreglarse seleccionando una fuente de voltaje adecuada, pero ganamos linealidad y por ende, más margen de error.

Carpeta OneWire.

Pese a que el código de este apartado es muy claro y este sí está bien comentado, explico por encima las funciones que realiza.

En la datasheet del DTH22 especifican un protocolo de lectura de monofilo, lo que requiere que exista una configuración dinámica del pin de datos, para mandar una señal de inicio de conversión y configurar dicho GPIO como entrada. He utilizado GPIO como forma de lectura utilizando esperas activas del orden de los microsegundos ayudándome de el contador del Timer 3, dado que este no lo toca ningún otro módulo y estaba sin usar. Dado que realizando pruebas este sensor tiene una cantidad de ruido importante, utilizar el modo capture era realmente un desafío. Por lo que he optado utilizar esperas activas, que no dañan a penas la ejecutabilidad del programa dado que la relación ejecución/uso (0.076% del tiempo) es muy baja; se llama cada 5 segundos a la función en el Handler del Timer 0 y ocupa 3.8ms de media en realizarse. Algo parecido pasa con I2C que utiliza librerías de espera activa.

En este archivo se ejecuta el protocolo y se almacena en una variable local a `mideTemperatura()`, que es la señal de inicio de medida, que se llama Rx. Se divide en 4 etapas, marcadas en el código en el marcador `@state`:

- Inicio de medida: Se configura el pin como salida de datos y se crea un pull down, luego se configura como entrada.
- Señal de respuesta: Para esta señal, se espera recibir 50us de nivel bajo más otros 50 de nivel alto, con un margen de 10us. Esto se ejecuta sobre la función `compruebaRespuesta()`. No se tiene en cuenta el tiempo que tarda el sensor en volver a poner el tiempo en alto, por lo que hay que esperar a que lo haga, con un valor de timeout de 45us.
- Lectura de datos: Se leen los datos en función de la duración del flanko de subida, si se excede el tiempo de timeout, se aborta la medición.
- Comprobación de checksum: Se lee el último byte y se compara con el algoritmo de checksum del fabricante. Si coincide, se insertan los nuevos datos, si no coincide se aborta la medida y se espera a la siguiente llamada.

Existe la función `reinicia cuenta`, que devuelve el valor del contador TC, que está reducido por un valor de 25, lo que provoca cuentas cada 1 us, y acto seguido lo reinicia. Por eso podemos obtener el valor del tiempo fácilmente.

Cabe destacar que si no se resuelven los márgenes, se sale de la función con un código de salida de 1 en la función `compruebaRespuesta()` y un código de 0 para la función `LeerByte()`. Los valores de timeout de datos son de 100us, si no se ha recibido un cambio en ese periodo, se sale de la medición y queda abortada.

Carpeta PWM.

En esta carpeta se encuentra todo lo referente al módulo PWM, residente en el archivo PWM.c. Existen dos funciones: `_configuraPWM()` y `modificaPulso()`. El código está comentado y explica todos los input y output de las funciones.

- La función de configuración lo que hace es recibir una frecuencia para el pulso de PWM y los puertos y pines que se quieren activar. En nuestro caso sólo los pines 2_1 y 6_1, uno para el servomotor y otro para el brillo del LCD.
 - La función modifica pulso, lo que hace es modificar el ciclo del pulso PWM, para cambiar su frecuencia, hay que reconfigurarla. Las entradas son:
 - PWMn: El pin PWM seleccionado.
 - Modo: Indica si el siguiente argumento es para el servo o para el LCD.
 - Ciclo: Si es modo ciclo, se utiliza este ciclo de trabajo del módulo PWM.
 - Grados: Si es modo servo, se utiliza este argumento como los grados a los que debe inclinarse la aguja del servo.
- PARA EL MODO SERVO:
- Mínimo: Valor mínimo del pulso a nivel alto en segundos.
 - Máximo: Valor máximo del pulso a nivel alto en segundos.

IMPORTANTE: Cabe destacar que el servomotor utiliza ciclos de 5V, cuando la placa LPC1768 utiliza ciclos de 3.3V. Esto es un problema dado que el servo parece utilizar la potencia del pulso y no precisamente su duración, es por ello por lo que no funciona de 0 a 180° y funciona de 45° a 135°. Esto puede corregirse modificando los valores de duración mínimo y máximo del pulso.

Experimentalmente se ha decidido que los valores por los que hay que multiplicar dichas constantes sean las siguientes:

- KMX: Constante del máximo = 1.3
- KMN: Constante del mínimo = 0.6

Lo que implica proporcionar más potencia al servo con la señal de 3.3V, dado que el mínimo disminuye y el máximo aumenta.

Las fórmulas utilizadas son, en función del ciclo o los grados, las siguientes:

$$MR_x = MR_0 \cdot \frac{Ciclo}{100}$$

$$MR_x = \left(Maximo + (Maximo - Minimo) \cdot \frac{Grados}{180} \right) \cdot \frac{1}{T_{reloj}} - 1$$

Carpeta DAC.

En esta carpeta se encuentra todo lo referente a la señalización y configuración del DAC. Para configurarlo, se utiliza el prototipo de función que se ha utilizado hasta ahora: `_configuraDAC_()`, que incluye la puesta a nivel alto de los LED de la placa, cuando están encendidos significa que está disponible la escritura de audio. Lo mismo para la lectura de audio. Esta función únicamente configura esos pines dado que el DAC se utiliza con el DMA y por ende se configura junto a él.

Las funciones `activarDac()` y `desactivarDac()` realizan la función de activación o desactivación del DAC.

Al activar el DAC, se activa el canal correspondiente del DMA, es decir, se manda una señal de inicio. Además, se configura un Timer para llamar a `desactivarDac()` pasados los 2 segundos. Se apaga el LED de la placa que señala escritura de audio.

Para desactivar el DAC, se señala al sistema con el actualizador que se ha acabado el tiempo del DAC y desactiva el canal del DMA, es decir, se manda una señal al DMA de fin. Además, se enciende el led correspondiente a escritura de audio y se escribe un valor de 0 en el DAC, dado que no hay señal de salida una vez finalizada la conversión.

Carpeta Database.

En esta carpeta se encuentran todos los archivos referentes a look-up-tables (LUT.c) y transferencia de memoria (DMA.c). En el archivo LUT.c podemos ver que reside la función `goto_LUT` que admite los siguientes parámetros:

- Variable: Variable de entrada.
- LUTn: El tipo de tabla que vamos a usar.
- Ret_x: Son un grupo de parámetros por los que hay que señalar una dirección en la que se va a guardar el resultado. Los tipos admisibles son de enteros sin signo de 8 a 64 bits y flotantes.

Las tablas que existen son Brillo_LDR, que traduce el valor en ohmios medido por el ADC a brillo, y Brillo2Ciclo_LDR, que traduce la variable brillo o claridad expresada en LUX a un ciclo de trabajo para controlar el servomotor. Sólo el LDR utiliza estas tablas.

En el archivo de DMA.c existen tres funciones y las tres son de configuración.

La primera función es la función `_configuraDMA_()`, que genera un tono de 32 muestras y activa la salida analógica (P0.26). Luego hace una llamada a la función `_configuraTono_()`, que deja configurado el audio pregrabado, que es un tono de 400Hz. Este es activado si se supera el valor mínimo de la alarma. Si se superase el valor máximo, se llamaría a la función `_configuraAudio_()`, esta función configura el DMA para reproducir el audio grabado. En ambos casos las muestras son de 8 bits y se crea una estructura (LLI0) que contiene las direcciones de origen y destino de las transferencias, así como que el destino se incremente y el número de muestras a transferir. Además, en ambos se configura que la transferencia es de memoria a periférico y que se transfiera al DAC. Dentro del DAC, hay que configurar que las transferencias son realizadas vía DMA.

La frecuencia a la que se transfieren las muestras son las siguientes:

- Para el tono:

$$T_{dma} = \frac{Num_{muestras}}{400Hz}$$

- Para el audio:

$$T_{dma} = \frac{Duracion_{audio}}{F_s} = 4kHz$$

Las funciones `_configuraTono()` y `_configuraAudio()` son llamadas si se exceden los valores límite de alarma, para que el DMA lance la transferencia ya sea por audio o por alarma pregrabada. Tras configurar el DMA así, se lanza un Timer (Timer 1), que desactiva el canal pasados 2 segundos.

Carpeta WDT.

Esta carpeta contiene los archivos referentes al WatchDogTimer, en `WDT.c`. Existen dos funciones:

- `_configuraWDT()`: Se encarga de configurar el WDT, la configuración seleccionada es de un timeout de 10 segundos utilizando el reloj de $F_{clk}/4$. La acción a realizar tras vencer el temporizador es de reiniciar el sistema, por lo que es muy importante estar alimentando el WDT continuamente.
- `alimentaWDT()`: Es la función que reinicia el contador del WDT. Se escribe en su registro de alimentación el código 1 y acto seguido el código 2 para reiniciarlo y evitar el reinicio del sistema. El código 1 se define como `0xAA` y el código 2 se define como `0x55`.

Esto se hace cada ciclo de `_mainLoop()`. Si no se entra a dicha función en 10 segundos, este temporizador reinicia el sistema. Esto evita bloqueos indeseados.

Carpeta UART.

Esta carpeta utiliza una librería proporcionada por la signatura SEDA de la UAH. Y el archivo `UART0.c`.

Dentro del archivo `UART0.c` podemos observar dos funciones, una de configuración y una de procesado del comando.

Respecto a la configuración, la función hace referencia a la librería para configurar, con 9600 baudios, el UART0 que es la conexión por USB (controlador de UART de la placa). En la librería se tiene una función para el cálculo de los registros de configuración de los baudios con un método iterativo. También se encuentra el manejador de la interrupción de UART0, que se activa cada byte recibido y por cada vez que el buffer de transmisión esté lleno.

Cuando se envía una cadena, hay que usar la función de la librería `tx_cadena_UART0`.

Para procesar el comando recibido, se guarda cada byte en un buffer (bufferx) y al recibir el carácter 13 (\r) se considera como acabado y se llama a la función de procesar comando: procesarComando(char * string).

Esta función tiene una máquina de estados implementada como se muestra en la figura 6. Donde se espera a recibir un tipo de comando para ejecutar una determinada acción. Los comandos pueden observarse en el Capítulo 6: Manual de usuario, en el apartado de UART.

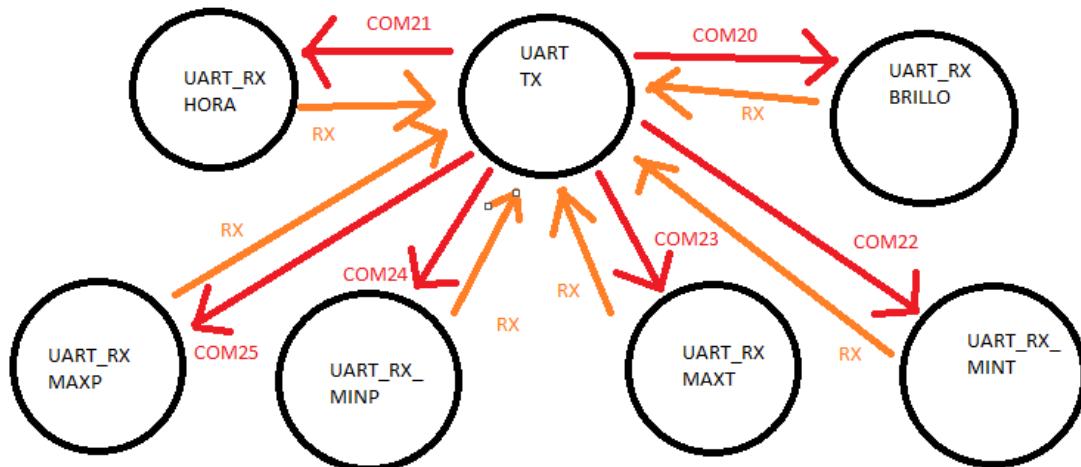


Figura 6.- Diagrama de estados de UART0.

Una vez obtenido el comando de modificación necesario, se procede a introducir el valor de la variable a modificar, el nombre del estado hace alusión a la variable a modificar. (Ver manual de usuario para más información sobre los comandos y el formato de salida).

La cadena a transmitir se guarda en un buffer y luego se transmite (UART0_BUFFER_TX[]) con la función de la librería (tx_cadena_UART0).

Carpeta TCP/IP.

En esta carpeta se encuentran los archivos referentes a la pila de protocolos TCP/IP que forman en servidor web. Se incluyen dos librerías: TCP_CM3.lib (la del fabricante) y EMAC_LPC17XX_LAN8720.c (la que se encarga del servicio sobre ethernet). Además, existe el archivo WEB.imp que guarda la orden de compilación:

```
index.cgi to WEB.C nopr root (.../TCPIP)
```

Esta orden manda compilar el archivo CGI a otro llamado WEB.C que contiene la página web, convirtiendo el archivo de marcado en un objeto de C.

Luego se encuentran los siguientes archivos:

- `HTTP_SOURCE.c`: Contiene la configuración de la página web (`_configuraWEB_()`) que llama a la función `init_TcpNet()` de la librería, que inicia la conexión TCP. Además, encapsula en `_mantenerTCP_()` a la función de la librería `main_TcpNet()` que es la que hay que llamar con cierta frecuencia (se encuentra en el programa `main.c`).
- `NET_Config.c`: Contiene la configuración de la conexión TCP, lo más relativo de este fichero se encuentra definido en el archivo `miGlobal.h`, que define la IP del servidor, el Gateway y la máscara subred. En este archivo se define con `DEFAULT` la ip 192.168.1.120 con un Gateway de 192.168.1.20 y una máscara subred de 255.255.255.0. Además, el usuario y la contraseña se definen como `user` y `Alver` respectivamente.
- `Index.cgi`: contiene el lenguaje de marcado en formato CGI y html de la página web a compilar. Esta pasa a convertirse en `WEB.C`.
- `HTTP_CGI.c`: Se encarga de la parte automática de la página web, mediante llamadas a callbacks y al método GET.
- **Callbacks**: se encuentra en la función `cgi_func`, función llamada cuando una línea de `index.cgi` empieza por la letra c. Se utiliza un espacio y una letra para determinar la acción de dicha callback:
 - t para temperatura.
 - v para velocidad del viento.
 - p para presión.
 - h para humedad.
 - i para índice UV.
 - b para brillo.
 - a para altitud.
 - X no usado.
 - Y no usado.
 - A año.
 - M mes.
 - D día.
 - H horas.
 - T minutos.
 - S segundos.Así podemos sacar por este callback y la función `sprintf` la variable en cuestión y poder representarla en el formato de la página WEB.
- **GET**: Se utiliza para enviar comandos a la estación, se define una tabla en la que se pueden escribir valores para enviar a la estación, todos los modificables comentados con anterioridad exceptuando la hora, que es modificable por pantalla y por UART. Primero se lee de la cadena si coincide la variable con las siguientes cadenas:
 - `Tmin=` para temperatura mínima.
 - `Tmax=` para temperatura máxima.
 - `Pmin=` para presión mínima.
 - `Pmax=` para presión máxima.
 - `Vart=` para seleccionar temperatura.
 - `Varp=` para seleccionar presión.Si coincide con dicha string, se pasa a procesar el número que viene a continuación y se guarda en la variable en el campo de la estructura correspondiente `MODIFICABLES.[campo]`.

Carpeta RTC.

Dentro de esta carpeta, se encuentra el archivo RTC.c, lo que contiene la configuración y el manejador de la interrupción del RTC. Se configura con `_configuraRTC_()` y se incicia el reloj a principios de año de 2020. Se configura para interrumpir cada segundo para actualizar una variable llamada `Clock`, que contiene una string con el valor de la fecha actual, además tiene un contador de segundos.

Carpeta Timers.

En esta carpeta se encuentra todo lo referido a los Timers, es la carpeta más importante dado que es donde se hacen todas las llamadas a las funciones de las demás carpetas. Dentro de esta carpeta se encuentran las configuraciones y los manejadores de interrupción de los temporizadores y del SysTickTimer.

- `_configuraSysTick_()`: Es la función que configura el SysTick para interrumpir cada 100ms.
- `SysTick_Handler()`: En esta función se hace una llamada a una función necesaria para la librería TCP/IP: `timer_Tick()`. Además, incrementa un contador (`contadorLUZ`) cada 100ms. Si este alcanza el tiempo límite sin tocar la pantalla (recuerdo que tocar la pantalla reinicia dicho contador) y este se encuentra en ULPM, esta desactiva el brillo automático y pasa a mandar una señal PWM de brillo del 1%. Por lo que se apaga, dejando un pequeño margen de luz del 1%.
- `_configuraTimer0_()`: Configura el Timer0 para interrumpir cada 0.5 segundos. En esta configuración se activan todos los timer a usar, aunque no el manejador de sus interrupciones.
- `Timer0_IRQHandler()`: En este manejador, tenemos tres funciones: medir el grupo 1 de sensores cada 0.5 segundos, medir el grupo 2 de sensores cada 5 segundos y actualizar el servo cada 0.5 segundos. Este temporizador controla el tiempo de muestreo.
- `Timer1_IRQHandler()`: Si interrumpe el MR1, se considera interrupción por fin de salida del DAC y se desactiva el mismo. Si interrumpe el modo capture del (CAP1.0), se considera que ha llegado un pulso del anemómetro y se llama a la función que lo mide. (Hay que tener en cuenta que quien activa el anemómetro es el Timer0 y se desactiva solo tras obtener una medida, si este no manda un pulso, la velocidad saldrá como 0 m/s indicando que no existe viento en ese momento).

Carpeta SDCARD.

Esta carpeta contiene los ficheros diskio.c, ff.c, SPI_MSD_Driver.c en caso de que en un futuro se quiera utilizar la tarjeta SD en la estación meteorológica. En esta versión no se incluye la tarjeta SD.

Carpeta CMSIS.

En esta carpeta se encuentra el software del fabricante CMSIS para utilizar en las librerías, varias de ellas utilizan los archivos de esta carpeta. Entre ellos se encuentran:

- lpc17xx_ssp.c
- lpc17xx_pinsel.c
- lpc17xx_gpio.c
- lpc17xx_clkpwr.c
- lpc17xx_libcfg_default.c

Carpeta GLCD.

En esta carpeta se encuentra la librería que se encarga de comunicar los pines de la placa con el TFT, utilizando funciones para mandar datos por dichos pines. (Ver esquema del Anexo I).

Capítulo 5: Pruebas, ejecutabilidad y conclusiones.

En este capítulo se explican los bugs que puede llegar a tener la estación meteorológica en esta versión con pruebas que se han realizado. Luego un estudio de ejecutabilidad y una recapitulación de objetivos logrados.

Pruebas.

Pruebas realizadas con la estación concluyen que puede haber los siguientes bugs:

- Aparición de números en las medidas no deseables: Cuando se realiza una medida de una variable que por un momento se ha excedido de un valor alto, puede que, si al no cambiar de pantalla el valor se reduce lo suficiente como para pasar de 3 a 2 dígitos, por ejemplo, el último dígito no sea borrado de la pantalla y parezca que sigue ahí. Esto es resuelto cambiando de pantalla. Puede observarse reduciendo valores de alarma de presión cómo la letra acaba por duplicado. El error viene de las funciones de la librería TouchPanel.c.
- Caída de la conexión TCP: Depende del tiempo de ejecución y de lo que se haya realizado hasta el momento, puede haber una caída de la conexión TCP y que la página WEB se caiga, esto se resuelve reiniciando el sistema. La media medida en la que se produce este fallo oscila entre los 9 o 10 minutos de ejecución del programa, habiendo veces que no sucede y habiendo veces que al minuto 2 está caída.
- Bloqueo del sistema por caída de la conexión TCP: Depende de si se ha caído la conexión TCP o no, puede bloquearse el sistema por un error que denomina la librería como ERROR_FREE_MEMORY, que es un error de liberación de memoria. Inmediatamente este error reinicia el sistema debido al WatchDogTimer. Si no se cae la conexión TCP, este error no sucede. La media medida de este error oscila las 2:30 horas de ejecución, aunque si no se produce la caída no se forma, y aun produciéndose puede tardar bastante más en caerse. Su record medido es de 6 horas después de caerse la conexión TCP a los 8 minutos de ejecución. Este error es resuelto automáticamente con el WatchDogTimer.

Las medidas de la estación son bastante precisas para la calidad y el precio de los sensores utilizados. Aunque las pruebas realizadas no han sido posibles con el sensor UVA30A, se ha probado a simular su comportamiento y el comportamiento es correcto.

Ejecutabilidad.

Con el simulador de Keil uVision4 podemos obtener la diferencia de tiempos entre dos breakpoints y medir las diferentes tareas que componen el sistema, en la Tabla 1 se muestran las tareas, el tiempo máximo que hay para realizarlas, el tiempo mínimo que tardan en ser requeridas y el tiempo máximo que tardan en ejecutarse.

| Tarea | Deadline (D) | T.mín (T) | T.ejec (R) | Unidades |
|----------------------------|--------------|-----------|------------|----------|
| RTC | 1000 | 1000 | 0.41 | ms |
| Timer0 | 500 | 500 | 1.21+0.69 | ms |
| Timer0* | 5000 | 5000 | 10.32 | ms |
| SysTick | 100 | 100 | 0.92 | ms |
| UART | 1.04 | 1.04 | 0.12 | ms |
| ADC (microfono) | 0.125 | 0.125 | 0.10 | ms |
| WDT | / | 10000 | 108.729 | ms |
| Statechart. | | | | |

Tabla 1.- Datos de ejecutabilidad del sistema.

El RTC al interrumpir cada segundo y actualizar el reloj, su D = T es de ese segundo.

El Timer0 interrumpe cada 0.5s, siendo cada 5s el mayor tiempo de ejecución que tiene, incluyendo el protocolo OneWire e I2C que son los que incluyen esperas activas, es por eso por lo que incluyo dos Timer0, el largo cada 5s y el corto cada 500ms.

El SysTick tiene 100ms para mantener abierta la pila de protocolos TCP.

El Timer1 no tiene tiempo mínimo de ejecución, dado que lee los pulsos del anemómetro y cuando necesite y se cierra, esto está incluido en el tiempo de ejecución del Timer0 cada 500ms.

UART tiene que mandar 9600 bits por segundo y lanza 11 bits, 8 de datos y 1 de start y stop y otro de paridad.

WDT reinicia al sistema cada 10s, por lo que hay que el tiempo máximo es de 10s.

El ADC tiene una frecuencia de muestreo de 8000kHz, por lo que su tiempo de muestreo es el máximo que tenemos para ejecutar las funciones dentro del manejador.

Nota: Los valores 0.001 son valores que no se han conseguido medir dado que el simulador los ha considerado despreciables, dado que rondan el orden de las 3 o 4 líneas de código.

Nada es bloqueante excepto el ADC, pero como se ha comentado con anterioridad, cuando se mide con el ADC la voz, se bloquean las medidas, por lo que el Timer0 no ejecuta código. La suma de la duración de todas las tareas no es mayor que los 100ms del SysTick. Luego está la tarea del Statechart, pero al no estar en una interrupción su prioridad es mínima, por lo que la máxima interrupción de Statechart sería de 12ms aproximadamente en caso de que todo interrumpiese a la vez. Cosa que el usuario no notaría para nada.

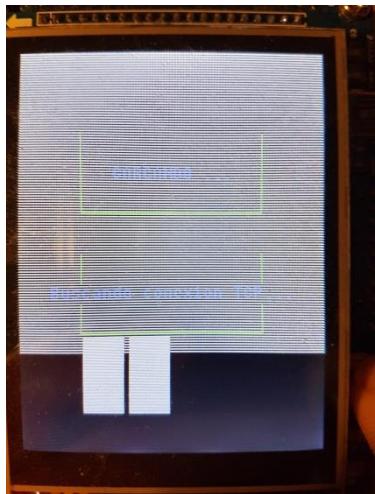
Conclusiones.

La práctica en su totalidad está completa, aunque no se han añadido muchos de los objetivos opcionales (sólo el RTC y el DMA y alguna mejora de la interfaz de pantalla de usuario), por lo general funciona muy bien en cuanto a lectura de sensores y funcionamiento de pantallas refiere. La UART también funciona muy bien, aunque personalmente me gustaría mejorar los bugs que tiene la pila de protocolos TCP/IP que bloquean el sistema tras 4 o 5 horas de ejecución, de hecho, si comento la línea de configuración de TCP/IP, el sistema no se bloquea nunca al no ser que se mande el comando KILL (ver manual de usuario).

La parte más costosa ha sido el ADC y el DMA junto con la página WEB, dado que he tenido muchos problemas al leer el audio y reproducirlo. La poca documentación sobre DMA que disponía me ha hecho tener que referirme a ejemplos colgados en internet y el largo tiempo que me ha llevado modificar pequeñas variables de la página web para que funcione ha sido determinante para centrarme en la calidad de esta. Además, no disponía ninguna experiencia en html ni en diseño web, por lo que la página es un poco simplista, aunque hace lo que se requiere de ella sin muchos problemas excepto ese bloqueo ocasional.

Capítulo 6: Manual de usuario.

En este capítulo se redacta lo necesario que debe conocer alguien que no sabe el comportamiento interno de la estación y solo desea utilizarla, transparentemente del comportamiento interno que la misma realice. A continuación, para las diferentes interfaces, se resume su manual.



TFT.

La pantalla al ser iniciada muestra por pantalla una interfaz que muestra el porcentaje de carga o pasos realizados para finalizar su configuración. Una vez realizada, se pasa a la pantalla de inicio. La primera barra indica que se ha iniciado el sistema, la segunda que se está iniciando la conexión TCP, la tercera que se están generando variables de alto costo computacional y la cuarta que se están iniciando los módulos del sistema.

Figura 1.- Carga.

Pantalla de inicio:

En esta pantalla se puede observar el reloj en la parte superior junto a dos flechas que permiten desplazarnos a las pantallas de medidas. Tras iniciar el sistema la fecha predeterminada son las 0:0:0 del día 1/1/2020. Más abajo podemos ver el botón de valores y el de ajustes que nos mandan a dichas pantallas.

Debajo de Nivel de brillo podemos seleccionar el nivel de brillo de la pantalla, del 1 al 4 en orden de brillo ascendente y automático a la derecha del todo. Presionar el botón de automático es sinónimo de pasar a modo de bajo consumo (LPM o LP). Si presionamos los botones HP automáticamente se selecciona el brillo 4, y seleccionando cualquier otro nivel de brillo se pasa al modo de alto consumo (HPM o HP). Si se presiona el botón de ULP se pasa al modo de muy bajo consumo, que, además de tener brillo automático, se apaga la pantalla si en un tiempo determinado y modificable no es utilizada por el usuario.

En esta pantalla se encuentra el botón de grabar y reproducir audio (Load en la esquina derecha y Play en la esquina izquierda respectivamente).

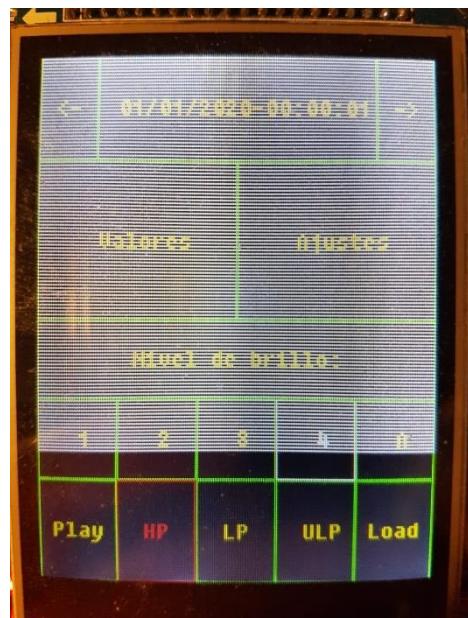


Figura 2.- Inicio.



Figura 3.- Medidas (1).

Pantalla de medidas 1:

En esta pantalla podemos observar las medidas actuales, es decir, las últimas tomadas por la estación. En esta pantalla podemos observar las variables de altitud en metros, velocidad del viento en metros por segundo, humedad en porcentaje relativo, claridad en LUX e índice UV en UVs.

Además, podemos observar el reloj en la misma posición que la pantalla de inicio con las dos flechas de mover la pantalla, por lo que la interfaz superior sigue siendo la misma para cualquiera de las pantallas en la que nos encontramos.



Figura 4.- Medidas (2)



Figura 5.- Ajustes.

Pantalla de ajustes:

En esta pantalla se puede modificar fecha y la hora pulsando los botones de incremento y decremento de la fila correspondiente (+ y - respectivamente). Además, podemos visualizar la IP que utiliza el servidor en el mismo momento que estamos visualizando el valor.

El valor por defecto de la dirección IP del servidor es 192.168.1.120, pero si es modificado, se mostrará el valor modificado para que el usuario sepa en todo momento la dirección a la que referirse.

Para retroceder a la pantalla de inicio basta ocn pulsar cualquier flecha de las situadas en las esquinas superiores.

Pantalla de valores:

En esta pantalla se pueden modificar los valores máximo y mínimo de la estación meteorológica. Tanto de alarma de presión como de alarma de temperatura, valores máximos y mínimos. Pulsando las interfaces de incremento y decremento (+ y - respectivamente) podemos modificar dichos valore. Al final de la pantalla podemos observar los límites mencionados.

Al lado de Pres (que indica presión) vemos dos números, el de la izquierda es el valor mínimo de alarma y el de la derecha el valor máximo.

Al lado de Temp (que indica temperatura) vemos dos números, el de la izquierda es el valor mínimo de alarma y el de la derecha el valor máximo.



Figura 6.- Valores.

Además, si se toca la última línea que muestra la información de los valores mínimos y máximos, vemos que cambia de color, rojo indicando que se representa temperatura y verde que se representa presión, esto puede verse en el servomotor, que indica a su izquierda en su valor mínimo y a la derecha el máximo.

WEB.

La interfaz web consta de 3 partes, una tabla que se actualiza cada 20 segundos con los datos recibidos de la estación meteorológica, una segunda con la fecha de la medida y una tercera con los valores a modificar.

En la figura 7 podemos ver la interfaz WEB, en la última tabla pueden modificarse los siguientes valores:

- Valor mínimo de la alarma de temperatura. (°C)
- Valor máximo de la alarma de temperatura. (°C)
- Valor mínimo de la alarma de presión. (mBar.)
- Valor máximo de la alarma de presión. (mBar.)
- Tiempo que tarda en apagarse la pantalla en ULPM (segundos).

Estacion meteorologica

| Datos medios actuales: | | | |
|------------------------|------------|-----------------------|----------|
| Temperatura: | 14.000000 | Velocidad del viento: | 0.000000 |
| Humedad: | 62.799999 | Indice UV: | 0.000000 |
| Presion: | 936.049988 | Brillo: | 0.000000 |
| Altitud: | 663.514160 | Longitud: | 0.000000 |
| Latitud: | 0.000000 | | |

Hora de la ultima muestra:

| | | | | | |
|-------|------|---------|---|-----------|----|
| Ano: | 2020 | Mes: | 1 | Dia: | 1 |
| Hora: | 0 | Minuto: | 0 | Segundos: | 24 |

Magnitudes modificables:

| | |
|--|------|
| Temperatura min. : | -10 |
| Temperatura max. : | 50 |
| Presion min. : | 500 |
| Presion max. : | 1500 |
| Segundos encendido : | 10 |
| <input checked="" type="radio"/> Temperatura | |
| <input type="radio"/> Presion | |
| <input type="button" value="Enviar"/> | |

Autor: Alberto Palomo Alonso. Sistemas Electronicos Digitales Avanzados. Universidad de Alcala - Escuela politecnica superior.

Figura 7.- Interfaz WEB.

Para configurar nuestro dispositivo que accede vía ethernet, debemos dirigirnos a nuestra configuración de IPv4. En Windows, eso es en la siguiente ruta:

- Configuración de Red e Internet/Cambiar opciones del Adaptador/Click derecho Ethernet/Propiedades (root)/Protocolo de internet versión 4/Propiedades/Usar la siguiente dirección IP.

Por defecto está configurado que se utilice cualquier otra IP que no sea la del servidor, por ejemplo: 192.168.1.10.

Hay que configurar la máscara subred tal que sea: 255.255.255.0

La dirección de la puerta de enlace predeterminada o Gateway debe de ser: 192.168.1.20.

Estos valores pueden ser modificados dentro del código fuente del proyecto.

Para acceder a la web basta con conectar el cable ethernet al dispositivo y acceder a la dirección IP del servidor: 192.168.1.120 por defecto. Una vez entrando nos pedirán usuario y contraseña:

- Usuario: user
- Clave: Alver

Una vez introducido, deberíamos ser capaces de visualizar la interfaz WEB del servidor.

UART.

La interfaz UART es la interfaz USB del dispositivo - placa, con conectar un puerto USB mediante un cable USB-Micro USB al puerto de la tarjeta MINI-DK2 sería suficiente para iniciar la instalación automática del controlador.

Al realizar la instalación, se precisa de un monitor serie para podernos comunicar con un terminal serie, como puede ser Termite. La configuración del puerto serie debe de ser la siguiente:

- 9600 baudios.
- Paridad impar (odd).
- 8 bits por dato.
- 1 bit de stop.
- Append CR. (Sólo CR, el programa está hecho para interpretar sólo el carácter 13.)

Una vez realizada la configuración podemos comunicarnos con los siguientes comandos:

- GIVE [X]: Muestra por pantalla del dato [X] de la lista expuesta a continuación; sin los corchetes:
 - IP
 - TEMPERATURA
 - PRESION
 - VIENTO
 - LUGAR
 - INDICEUV
 - HORA
 - HUMEDAD
 - BRILLOCada uno haciendo referencia a la variable que indica en su propio nombre.
- SET [Y]: Configura la variable [Y] de la lista expuesta a continuación; sin los corchetes:
 - BRILLO
 - HORA
 - MIN TEMP
 - MAX TEMP
 - MIN PRES
 - MAX PRES
 - TEMPERATURA
 - PRESIONDonde brillo es el tiempo que transcurre desde que no se toca la pantalla hasta que se apaga en modo ULPM, min representa mínimo y max representa máximo de los valores de alarma de TEMP, temperatura, y PRES, presión. TEMPERATURA activa la representación de la temperatura en el servomotor y PRESION la de presión en el servomotor.
- KILL: Cuelga el sistema en un while(1), activa el WatchDog.
- ABOUT: Muestra información del creador.
- HELP: Muestra información de la UART.
- HELP SET: Muestra la información del comando GIVE.
- HELP GIVE: Muestra la información del comando SET.

Referencias

<https://github.com/iTzAlver/EstacionMeteorologica.git>

<https://developer.arm.com/ip-products/processors/cortex-m/cortex-m3> [En línea] / aut. ARM.

https://www.amazon.es/Anem%C3%B3metro-Velocidad-Estaci%C3%B3n-Meteorol%C3%B3gica-Arduino/dp/B07BMVYBW9/ref=asc_df_B07BMVYBW9/?tag=googshopes-21&linkCode=df0&hvadid=300930138206&hvpos=1o1&hvnetw=g&hvrand=3634125468818365177&hvptwo=&hvqmt=&hvdev=c&h [En línea] / aut. anemómetro Imagen.

www.intel.com [En línea] / aut. Intel.

www.nxp.com [En línea] / aut. NXP.

www.powermcu.com [En línea] / aut. POWER MCU.

<https://cloud.smartdraw.com/>

www.blackboard.com

https://uah.blackboard.com/ultra/courses/_17817_1/cl/outline

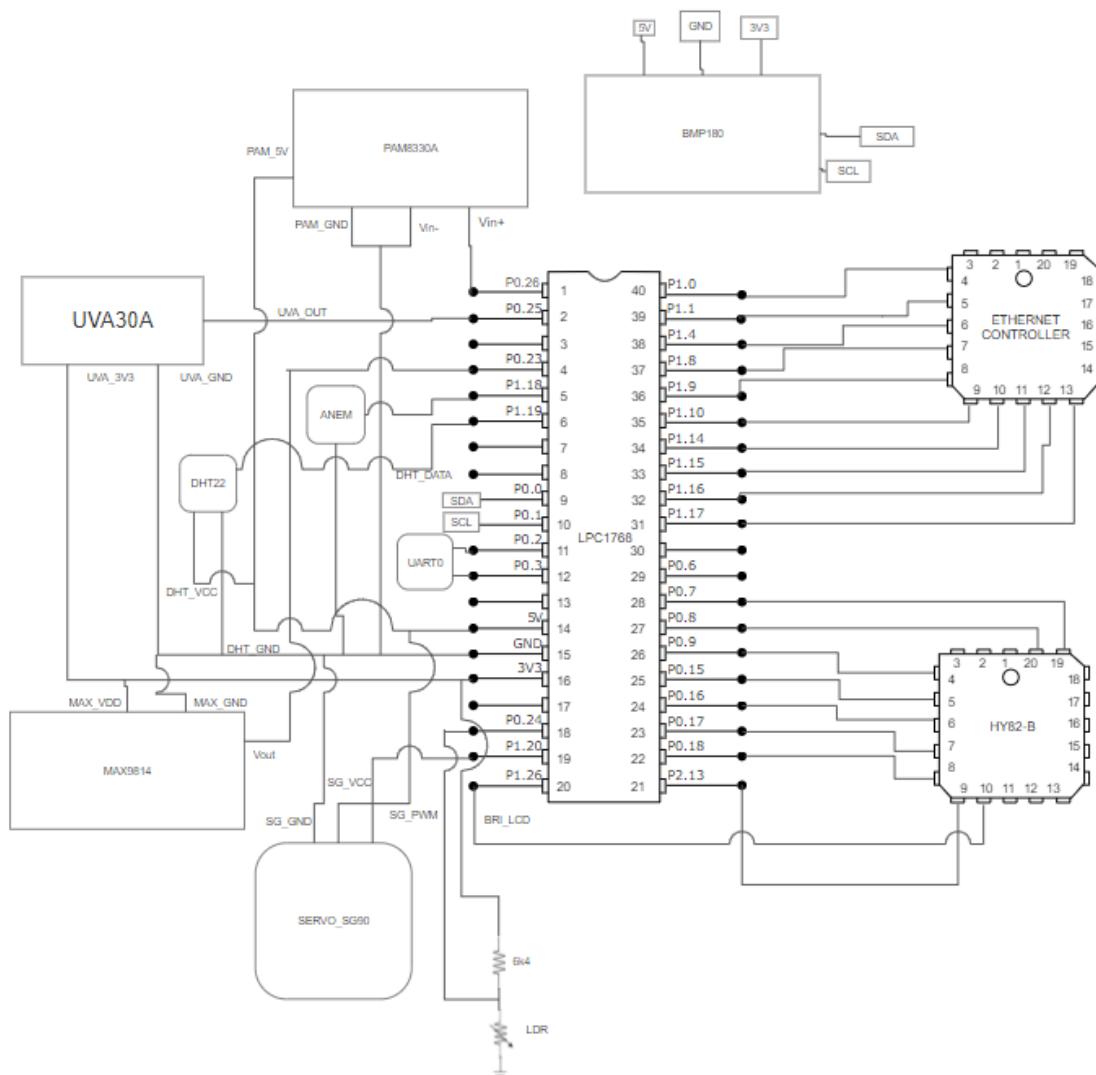
https://github.com/OCFreaks/LPC1768-Tutorial-Examples/blob/master/DHT11_Interfacing/main.c

Anexos.

A continuación, los anexos de la memoria de la estación meteorológica.

Anexo I – Esquemático.

En este anexo se adjunta un pequeño esquemático con la distribución de pines.



Anexo II – Código fuente.

Header.h

```
/**-----//  
//  @filename  header.h          //  
//  @version   0.00             //  
//  @author    Alberto Palomo Alonso //  
//  
//  @brief     Cabezera del código fuente, agrupa todos los archivos.           //  
//  
//  @category   Principal.          //  
//  
//  @map       @include           //  
//            @end                //  
//  
//-----//  
//  
//  @include   Incluye todos los archivos necesarios.           //  
//  
//-----*/  
// Librería de registros.  
#ifndef LPC17XX  
#define LPC17XX  
#include "LPC17XX.H"  
#endif  
// Símbolos del sistema.  
#ifndef SYSTEMSYMBOLS  
#define SYSTEMSYMBOLS  
#include "Systemsymbols.h"  
#endif  
// Pwm.  
#ifndef PWM  
#define PWM  
#include "PWM.h"  
#endif  
// Configura  
#ifndef CONFIGURA  
#define CONFIGURA  
#include "configura.h"  
#endif  
#ifndef STATECHART  
#define STATECHART  
#include "Statechart.h"  
#endif  
#ifndef GLCD  
#define GLCD  
#include "GLCD.h"  
#include "TouchPanel.h"  
#include "menu.h"  
#include "leds.h"  
#endif  
#ifndef STATECHART  
#define STATECHART  
#include "Statechart.h"  
#endif  
#ifndef HTTPSOURCE  
#define HTTPSOURCE  
#include "HTTP_SOURCE.h"  
#endif  
#ifdef DEBUG  
#include "DEBUG.h"
```

```
#endif
/**-----//  
//  
//-----//  
// @end ENDFILE.  
//  
//-----*/
```

Systemsymbols.h

```
/*-----//  
// @filename Systemsymbols.h //  
// @version 0.00 //  
// @author Alberto Palomo Alonso //  
//  
// @brief Este es el archivo donde son guardados los símbolos del sistema utilizados para los  
// diferentes archivos que necesiten una llamada a este tipo de definiciones. //  
//  
// @category Esencial. //  
//  
// @map @none //  
// @types //  
// @end //  
//-----*/  
#ifndef LPC17XX  
#define LPC17XX  
#include "LPC17XX.H"  
#endif  
// Acopladores de código.  
#ifndef null  
#define null 0  
#endif  
#ifndef NULL  
#define NULL 0  
#endif  
#define none 0  
#define NONE 0  
#define VOID void  
// Símbolos prácticos de valores instantáneos.  
#define TODO_1_8 0xFFFF  
#define TODO_1_16 0xFFFFFFFF  
#define TODO_1_32 0xFFFFFFFFFFFFFF  
// Símbolos correspondientes a definiciones propias para facilitar lectura de código.  
#define SUBIDA 1  
#define BAJADA 0  
#define ALTO 1  
#define BAJO 0  
#define FLANCO 1  
#define NIVEL 0  
#define INICIALMENTE_ACTIVO 1  
#define INICIALMENTE_INACTIVO 0  
#define FUNC0 0x0  
#define FUNC1 0x1  
#define FUNC2 0x2  
#define FUNC3 0x3  
#define NOPIN 0x00000000  
#define PIN_00 0x00000001  
#define PIN_01 0x00000002  
#define PIN_02 0x00000004  
#define PIN_03 0x00000008  
#define PIN_04 0x00000010  
#define PIN_05 0x00000020  
#define PIN_06 0x00000040
```

```

#define PIN_07      0x00000080
#define PIN_08      0x00000100
#define PIN_09      0x00000200
#define PIN_10      0x00000400
#define PIN_11      0x00000800
#define PIN_12      0x00001000
#define PIN_13      0x00002000
#define PIN_14      0x00004000
#define PIN_15      0x00008000
#define PIN_16      0x00010000
#define PIN_17      0x00020000
#define PIN_18      0x00040000
#define PIN_19      0x00080000
#define PIN_20      0x00100000
#define PIN_21      0x00200000
#define PIN_22      0x00400000
#define PIN_23      0x00800000
#define PIN_24      0x01000000
#define PIN_25      0x02000000
#define PIN_26      0x04000000
#define PIN_27      0x08000000
#define PIN_28      0x10000000
#define PIN_29      0x20000000
#define PIN_30      0x40000000
#define PIN_31      0x80000000

// Símbolos correspondientes a relojes del sistema.
#define Fcpu        100000000          // 100MHz velocidad de la cpu.
#define APBvalue    4                  // Valor del primer prescaler, 4 after-reset.
#define Prescaler   0                  // Valor del prescaler, 0 after-reset.
#define Fclk        (float)Fcpu/(float)APBvalue // Valor del reloj prescalado por APB.
#define Ftick       Fclk/(float)(Prescaler+1) // Valor del reloj asociado a los contadores.

#define Ts0         0.5                // Tiempo de muestreo en segundos sin prescaler. (Muestras)
#define Fs0         (float)1/(float)Ts0 // Frecuencia de muestreo en Hz. (Muestras)
#define CsADC       Fs0                // Frecuencia de muestreo del LDR.
#define CsCAP       10*Fs0              // Frecuencia de muestreo del UVA.

#define FsAudio     8000               // 3kHz de audio, Nyquist *= 2, Yo *= 8khz.
#define TsAudio     (float)1/(float)FsAudio // Periodo de muestreo del audio.
#define DURACION_AUDIO 2                // 2 segundos de audio.
#define MUESTRAS_AUDIO DURACION_AUDIO*FsAudio // Muestras en los 2 segundos de audio.
#define MUESTRAS_SENO 32
#define LECTURA_AUDIO 25               // Pin que señala lectura de audio.
#define ESCRITURA_AUDIO 26              // Pin que señala escritura de audio.

#define MAX_PRES   MODIFICABLES.Max_servo_p
#define MAX_TEMP   MODIFICABLES.Max_servo_t
#define MIN_PRES   MODIFICABLES.Min_servo_p
#define MIN_TEMP   MODIFICABLES.Min_servo_t

// Constantes universales.
#define PI          3.141592
/**-
//                                     //**
// @types      Tipos utilizados para el programa.                                //
//**/                                         //**/
typedef signed char      int8_t;
typedef signed short int int16_t;
typedef signed int        int32_t;
typedef signed long long int64_t;

typedef unsigned char     uint8_t;
typedef unsigned short int uint16_t;
typedef unsigned int       int  uint32_t;

```

```

typedef unsigned long long      uint64_t;

typedef struct {      // Contadores de 8, 16 y 32 bits.
    _IO uint8_t i;
    _IO uint8_t j;
    _IO uint16_t k;
    _IO uint32_t Audio;
    _IO uint32_t Segundos;
    _IO uint32_t RITicks;
}Counters_t;

typedef struct {
    uint8_tCHART;
}State_t;

typedef struct {
    float Longitud;
    float Latitud;
    float Altura;
}locat_t;

typedef struct {
    float Temperatura; // En grados celsius.
    float Presion;     // En pascales.
    float Humedad;    // En %.
    float IndiceUV;   // En UVs.
    locat_t Lugar;    // Sitio donde el GPS nos posiciona.
    float VelViento; // En m/s.
    float Brillo;    // En LUX.
}misDatos_t;

typedef struct {
    _IO uint8_t Anemometro:1;
    _IO uint8_t AnemometroRev:1;
    _IO uint8_t LDR:1;
    _IO uint8_tLDRrev:1;
    _IO uint8_tUVA:1;
    _IO uint8_tUVArrev:1;
    _IO uint8_tAudio:1;
    _IO uint8_tAudiorev:1;
    _IO uint8_t TempRev:1;
}actualizador_t;

typedef struct {
    uint32_t source;      // Start of source area
    uint32_t destination; // Start of destination area
    uint32_t next;        // Address of next strLLI in chain
    uint32_t control;     // DMACxControl register
} DMA_t;

typedef struct {
    float Max_servo_t;   // Done
    float Min_servo_t;   // Done
    float Max_servo_p;   // Done
    float Min_servo_p;   // Done

    uint8_tVar_medida;  // Done
    uint32_t TiempoBrillo; // Done
}modificables_t;

-----//
//
-----//
//      @end      ENDFILE.
//
-----*/

```

Main.c

```
/*
//  @filename  main.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Código fuente del programa principal.
//
//  @category  Principal.
//
//  @map      @include
//            @global
//            @main
//            @end
//
//  @include  Estos son los archivos utilizados con el código fuente.
//  //  */
//  #ifndef HEADER
//  #define  HEADER
//  #include "header.h"
//  #endif
//  /*
//  //  */
//  //  @global  Programa principal, variables globales.
//  //  //  */
//  misDatos_t    objDATOS;          // Objeto.
misDatos_t * DATOS = &objDATOS; // Mis datos almacenados en la variable objDATOS.
State_t        objESTADO;         // Objeto.
State_t * ESTADO = &objESTADO;    // Declarar como extern. (Hey, compilador, creeme que hay una variable por ahí que se llama ESTADO)
Counters_t     objCOUNTERS;       // Objeto.
Counters_t * COUNTERS = &objCOUNTERS; // Declarar como extern. (Hey, compilador, creeme que hay una variable por ahí que se llama COUNTERS)
actualizador_t objACTUALIZADOR;   // Objeto.
actualizador_t * ACTUALIZADOR = &objACTUALIZADOR; // Declarar como extern. (Hey, compilador, creeme que hay una variable por ahí que se llama ACTUALIZADOR)
//  /*
//  //  */
//  //  @main   Programa principal, inicio after-reset.
//  //  //  */
//  //  @ref    __configuraPrograma__ -> configura.h
//  //          __mainLoop__           -> statechart.h
//  //  //  */
//  int main ()
{
    __configuraPrograma__();
    while (1)
    {
        __mainLoop__();
        __mantenerTCP__();
    }
}
//  /*
//  //  */
//  //
```

```
//      @end      ENDFILE.  
//  
//-----*/  
//          //
```

Statechart.h

```
/**-----//  
//      @filename  Statechart.c           //  
//      @version   0.00                  //  
//      @author    Alberto Palomo Alonso //  
//  
//      @brief     Cabecera del código fuente de Statechart.c //  
//  
//      @category  Principal.          //  
//  
//      @map      @include            //  
//                  @private             //  
//                  @types              //  
//                  @funcdef            //  
//                  @end                //  
//  
//  
//-----//  
//  
//      @include   Estos son los archivos utilizados con el statechart. //  
//  
//-----*/  
// LCD  
#ifndef GLCD  
#define  GLCD  
#include "GLCD.h"  
#include "TouchPanel.h"  
#include "menu.h"  
#endif  
#ifndef SYSTEMSYMBOLS  
#define  SYSTEMSYMBOLS  
#include "Systemsymbols.h"  
#endif  
#ifndef STRING  
#define  STRING  
#include <string.h>  
#endif  
#ifndef WTD  
#define  WDT  
#include "WDT.h"  
#endif  
#ifndef PWM  
#define  PWM  
#include "PWM.h"  
#endif  
#ifndef STDIO  
#define  STDIO  
#include <stdio.h>  
#endif  
#ifndef MIGLOBAL  
#define  MIGLOBAL  
#include "miGlobal.h"  
#endif  
#ifndef TIMERS  
#define  TIMERS  
#include "Timers.h"
```

```
#endif
#ifndef RTL
#define RTL
#include "RTL.h"
#endif
#ifndef HTTPSOURCE
#define HTTPSOURCE
#include "HTTP_SOURCE.h"
#endif
#ifndef LUT
#define LUT
#include "LUT.h"
#endif
#ifndef DAC
#define DAC
#include "DAC.h"
#endif
#ifndef UFONO
#define UFONO
#include "uFono.h"
#endif
#ifndef ONEWIRE
#define ONEWIRE
#include "OneWire.h"
#endif
/**-
// @private Estos son los símbolos correspondientes al statechart.
// */
// @types Tipos utilizados en el statechart. COPIADOS DE menu.c
// -----
typedef struct {
    uint16_t x;
    uint16_t y;
    uint16_t size_x;
    uint16_t size_y;
    uint8_t pressed;
}screenZone_t;
/**-
// @funcdef Estas son las funciones correspondientes al statechart.
// */
// -----
void __mainLoop__ ( void );
void __configuraLCD__ ( void );
void __pintaInicio__ ( void );
void __pintaAjustes__ ( void );
void __pintaMedidas1__ ( void );
void __pintaMedidas2__ ( void );
```

```

void squareButton      ( screenZone_t * zone      , char *      text      , uint16_t textColor , uint16_t lineColor);
void checkTouchPanel   ( void );
int8_t zoneNewPressed (screenZone_t * zone );
void squareBox          (screenZone_t * zone      ,  uint16_t  color);
void __pintaCargandoSeno__ ( void );
void __pintaCargandoConexion__( void );
void __pintaCargandoDone__ ( void );
void __pintaCargandoInicio__ ( void );
void __pintaCargandoIniciando__ ( void );
void __pintaValores__ ( void );
/*-----*/
//                                     //           //
//                                     //           //
// @end    ENDFILE.                  //           //
//                                     //           //
//-----*/                         //

```

Statechart.c

```

/**-----*/
//  @filename  Statechart.c                      //
//  @version   0.00                                //
//  @author    Alberto Palomo Alonso              //
//                                                     //
//  @brief     Código fuente correspondiente a la máquina de estados que compone el menú.      //
//                                                     //
//  @category  Principal.                          //
//                                                     //
//  @map      @include                           //
//            @global                            //
//            @function                           //
//            @end                               //
//                                                     //
//                                                     //
//-----*/                                         //
//  @include   Estos son los archivos utilizados en el statechart.                         //
//                                                     //
//-----*/                                         //
#ifndef STATECHART
#define STATECHART
#include "Statechart.h"
#endif
/**-----*/
//                                     //
//                                     //
//  @global    Estas son las variables globales pertenecientes al statechart.                 //
//                                                     //
//-----*/                                         //
extern uint8_t      Clock[23];
extern State_t      * ESTADO;
extern misDatos_t   * DATOS;
extern actualizador_t * ACTUALIZADOR;
extern uint8_t      OWEjecutameExterno;
extern uint16_t     contadorLUZ;
modifiables_t       MODIFICABLES;
char buffer[23];

```

```

uint8_tModo_energetico=0;
uint8_tModo_brillo=3;
uint8_tpressedTouchPanel;
uint8_t__brilloFade = 0;
uint8_t__brilloAuto = 0;
uint8_tAux8;

// ZONA DE PANTALLA DE INICIO.

screenZone_t zona_0= { 0 , 0 , MAXIMOX , MAXIMOOY*0.2 , 0 }; // Marco del reloj y
botones de adelante y atrás.
screenZone_t zona_1= { MAXIMOX*0.15 , 0 , MAXIMOX*0.7 , MAXIMOOY*0.2 , 0 }; // Reloj.
screenZone_t zona_2= { MAXIMOX*0.85 , 0 , MAXIMOX*0.15 , MAXIMOOY*0.2 , 0 }; // Derecha.
screenZone_t zona_3= { MAXIMOX*0 , 0 , MAXIMOX*0.15 , MAXIMOOY*0.2 , 0 }; // Izquierda.
screenZone_t zona_4= { MAXIMOX*0 , MAXIMOOY*0.2 , MAXIMOX*0.5 , MAXIMOOY*0.3 , 0 }; //
Primer botón.
screenZone_t zona_5= { MAXIMOX*0.5 , MAXIMOOY*0.2 , MAXIMOX*0.5 , MAXIMOOY*0.3 , 0 }; //
Segundo botón.
screenZone_t zona_6= { MAXIMOX*0 , MAXIMOOY*0.5 , MAXIMOX , MAXIMOOY*0.15 , 0 }; // Brillo
info.
screenZone_t zona_7= { MAXIMOX*0 , MAXIMOOY*0.65 , MAXIMOX*0.2 , MAXIMOOY*0.15 , 0 }; //
Primer botón de brillo.
screenZone_t zona_8= { MAXIMOX*0.2 , MAXIMOOY*0.65 , MAXIMOX*0.2 , MAXIMOOY*0.15 , 0 }; //
Segundo botón de brillo.
screenZone_t zona_9= { MAXIMOX*0.4 , MAXIMOOY*0.65 , MAXIMOX*0.2 , MAXIMOOY*0.15 , 0 }; //
Tercer botón de brillo.
screenZone_t zona_10= { MAXIMOX*0.6 , MAXIMOOY*0.65 , MAXIMOX*0.2 , MAXIMOOY*0.15 , 0 }; //
Cuarto botón de brillo.
screenZone_t zona_11= { MAXIMOX*0.8 , MAXIMOOY*0.65 , MAXIMOX*0.2 , MAXIMOOY*0.15 , 0 }; //
Brillo automático.
screenZone_t zona_12= { MAXIMOX*0 , MAXIMOOY*0.8 , MAXIMOX*0.2 , MAXIMOOY*0.2 , 0 }; //
Botón de audio.
screenZone_t zona_13= { MAXIMOX*0.2 , MAXIMOOY*0.8 , MAXIMOX*0.2 , MAXIMOOY*0.2 , 0 }; //
Volumen = 1.
screenZone_t zona_14= { MAXIMOX*0.4 , MAXIMOOY*0.8 , MAXIMOX*0.2 , MAXIMOOY*0.2 , 0 }; //
Volumen = 2.
screenZone_t zona_15= { MAXIMOX*0.6 , MAXIMOOY*0.8 , MAXIMOX*0.2 , MAXIMOOY*0.2 , 0 }; //
Volumen = 3.
screenZone_t zona_16= { MAXIMOX*0.8 , MAXIMOOY*0.8 , MAXIMOX*0.2 , MAXIMOOY*0.2 , 0 }; //
Botón de load.
// ZONA DE MEDIDAS.
screenZone_t zona_17= { MAXIMOX*0 , MAXIMOOY*0.2 , MAXIMOX , MAXIMOOY*0.1 , 0 }; //
Información de página, medidas.
screenZone_t zona_18= { MAXIMOX*0 , MAXIMOOY*0.3 , MAXIMOX , MAXIMOOY*0.1 , 0 }; //
Localización.
screenZone_t zona_19= { MAXIMOX*0 , MAXIMOOY*0.4 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Temperatura.
screenZone_t zona_20= { MAXIMOX*0 , MAXIMOOY*0.55 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Humedad.
screenZone_t zona_21= { MAXIMOX*0 , MAXIMOOY*0.70 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Presión.
screenZone_t zona_22= { MAXIMOX*0 , MAXIMOOY*0.85 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
ÍndiceUV.
// ZONAS DE DATOS.
screenZone_t zona_23= { MAXIMOX*0.5 , MAXIMOOY*0.4 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Valor de temperatura.
screenZone_t zona_24= { MAXIMOX*0.5 , MAXIMOOY*0.55 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Valor de humedad.
screenZone_t zona_25= { MAXIMOX*0.5 , MAXIMOOY*0.70 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Valor de presión.
screenZone_t zona_26= { MAXIMOX*0.5 , MAXIMOOY*0.85 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Valor de ÍndiceUV.
// ZONAS DE AJUSTES.
screenZone_t zona_27= { MAXIMOX*0 , MAXIMOOY*0.2 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Horas.
screenZone_t zona_28= { MAXIMOX*0 , MAXIMOOY*0.35 , MAXIMOX*0.5 , MAXIMOOY*0.15 , 0 }; //
Minutos.

```

```

screenZone_t zona_29 = { MAXIMOX*0 , MAXIMOY*0.5 , MAXIMOX*0.5 , MAXIMOY*0.15 , 0 }; // Segundos.
screenZone_t zona_30 = { MAXIMOX*0 , MAXIMOY*0.65 , MAXIMOX*0.5 , MAXIMOY*0.15 , 0 }; // Dia.
screenZone_t zona_31 = { MAXIMOX*0 , MAXIMOY*0.8 , MAXIMOX , MAXIMOY*0.2 , 0 }; // Slot libre.
screenZone_t zona_27m = { MAXIMOX*0.5 , MAXIMOY*0.2 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Resta horas.
screenZone_t zona_28m = { MAXIMOX*0.5 , MAXIMOY*0.35 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Resta minutos.
screenZone_t zona_29m = { MAXIMOX*0.5 , MAXIMOY*0.5 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Resta segundos.
screenZone_t zona_30m = { MAXIMOX*0.5 , MAXIMOY*0.65 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Resta dia.
screenZone_t zona_27M = { MAXIMOX*0.75 , MAXIMOY*0.2 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Suma horas.
screenZone_t zona_28M = { MAXIMOX*0.75 , MAXIMOY*0.35 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Suma minutos.
screenZone_t zona_29M = { MAXIMOX*0.75 , MAXIMOY*0.5 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Suma segundos.
screenZone_t zona_30M = { MAXIMOX*0.75 , MAXIMOY*0.65 , MAXIMOX*0.25 , MAXIMOY*0.15 , 0 }; // Suma dia.
// ZONAS DE MEDIDAS 2 (VIENTO)
screenZone_t zona_32 = { MAXIMOX*0 , MAXIMOY*0.2 , MAXIMOX*0.5 , MAXIMOY*0.2 , 0 }; // Velocidad del viento.
screenZone_t zona_32n = { MAXIMOX*0.5 , MAXIMOY*0.2 , MAXIMOX*0.5 , MAXIMOY*0.2 , 0 }; // Velocidad del viento.
screenZone_t zona_33 = { MAXIMOX*0 , MAXIMOY*0.4 , MAXIMOX , MAXIMOY*0.2 , 0 }; // Velocidad del viento.
screenZone_t zona_34 = { MAXIMOX*0 , MAXIMOY*0.6 , MAXIMOX*0.5 , MAXIMOY*0.2 , 0 }; // Cantidad de brillo.
screenZone_t zona_34n = { MAXIMOX*0.5 , MAXIMOY*0.6 , MAXIMOX*0.5 , MAXIMOY*0.2 , 0 }; // Cantidad de brillo.
screenZone_t zona_35 = { MAXIMOX*0 , MAXIMOY*0.8 , MAXIMOX , MAXIMOY*0.2 , 0 }; // Cantidad de brillo.
// Display de barras.
screenZone_t zona_350 = { MAXIMOX*0 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_351 = { MAXIMOX*0.1 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_352 = { MAXIMOX*0.2 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_353 = { MAXIMOX*0.3 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_354 = { MAXIMOX*0.4 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_355 = { MAXIMOX*0.5 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_356 = { MAXIMOX*0.6 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_357 = { MAXIMOX*0.7 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_358 = { MAXIMOX*0.8 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_359 = { MAXIMOX*0.9 , MAXIMOY*0.8 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
// Display de barras.
screenZone_t zona_330 = { MAXIMOX*0.0 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_331 = { MAXIMOX*0.1 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_332 = { MAXIMOX*0.2 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_333 = { MAXIMOX*0.3 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_334 = { MAXIMOX*0.4 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_335 = { MAXIMOX*0.5 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_336 = { MAXIMOX*0.6 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_337 = { MAXIMOX*0.7 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_338 = { MAXIMOX*0.8 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
screenZone_t zona_339 = { MAXIMOX*0.9 , MAXIMOY*0.4 , MAXIMOX*0.1 , MAXIMOY*0.2 , 0 };
// Menú de carga.
screenZone_t zona_lo0 = { MAXIMOX*0.2 , MAXIMOY*0.2 , MAXIMOX*0.6 , MAXIMOY*0.2 , 0 };
screenZone_t zona_lo1 = { MAXIMOX*0.2 , MAXIMOY*0.5 , MAXIMOX*0.6 , MAXIMOY*0.2 , 0 };
screenZone_t zona_lo20 = { MAXIMOX*0.2 , MAXIMOY*0.7 , MAXIMOX*0.15 , MAXIMOY*0.2 , 0 };
screenZone_t zona_lo21 = { MAXIMOX*0.35 , MAXIMOY*0.7 , MAXIMOX*0.15 , MAXIMOY*0.2 , 0 };
screenZone_t zona_lo22 = { MAXIMOX*0.5 , MAXIMOY*0.7 , MAXIMOX*0.15 , MAXIMOY*0.2 , 0 };
screenZone_t zona_lo23 = { MAXIMOX*0.65 , MAXIMOY*0.7 , MAXIMOX*0.15 , MAXIMOY*0.2 , 0 };
/*-----*/

```

```

//                                     //
//      @function    __mainLoop__()
//                                     //
//      @brief      Esta función es la que se ejecuta en el bucle principal del main. Debe contener
//                  todo el código ejecutable por el loop principal.                                //
//                                     //
//-----**/                                         //
void  __mainLoop__( void   )
{
    /**@LOOP: Primera parte del programa. */
alimentaWDT();
checkTouchPanel();
if( __brilloAuto && (SysTick->CTRL & 0x10000) ) // Cada 100 ms si el brillo auto está activado.
{
    goto_LUT( DATOS->Brillo, BRILLO2CICLO_LDR , none , &Aux8 , none , none );
    modificaPulso(  PWM6,   MODO_CICLO , Aux8 ,  none ,  none ,  none );
}
/**@LOOP: Máquina de estados LCD. */
switch( ESTADO->CHART )
{
    case PANTALLA_INICIO:
        __pintaInicio__();
        if(zoneNewPressed( &zona_2))
        {
            ESTADO->CHART = PANTALLA_MEDIDAS1;
            LCD_Clear(Black);
        }
        if(zoneNewPressed( &zona_3))
        {
            ESTADO->CHART = PANTALLA_MEDIDAS2;
            LCD_Clear(Black);
        }
        if(zoneNewPressed( &zona_5))
        {
            ESTADO->CHART = PANTALLA_AJUSTES;
            LCD_Clear(Black);
        }
        if(zoneNewPressed( &zona_4))
        {
            ESTADO->CHART = PANTALLA_VALORES;
            LCD_Clear(Black);
        }
        if(zoneNewPressed( &zona_7))
        {
            __brilloAuto = 0;
            __brilloFade = 0;
            modificaPulso(  PWM6,   MODO_CICLO ,  1 ,  none ,  none ,  none );
            Modo_brillo = 0;
            Modo_energetico =  0; // HP.
        }
        if(zoneNewPressed( &zona_8))
        {
            __brilloAuto = 0;
            __brilloFade = 0;
            modificaPulso(  PWM6,   MODO_CICLO ,  20 ,  none ,  none ,  none );
            Modo_brillo = 1;
            Modo_energetico =  0; // HP.
        }
        if(zoneNewPressed( &zona_9))
        {
            __brilloAuto = 0;
            __brilloFade = 0;
            modificaPulso(  PWM6,   MODO_CICLO ,  40 ,  none ,  none ,  none );
            Modo_brillo = 2;
            Modo_energetico =  0; // HP.
        }
}

```

```

        }

if(zoneNewPressed( &zona_10))
{
    modificaPulso(  PWM6 ,  MODO_CICLO ,  60 ,  none ,  none ,  none );
    __brilloAuto = 0;
    __brilloFade = 0;
    Modo_brillo = 3;
    Modo_energetico =  0; // HP.

}

if(zoneNewPressed( &zona_11))
{
    __brilloAuto = 1;
    __brilloFade = 0;
    Modo_brillo = 4;
    Modo_energetico =  1; // LP.

}

if(zoneNewPressed( &zona_12))
{
    if (  ACTUALIZADOR->Audiorev  )
    {
        ACTUALIZADOR->Audiorev = 0;
        __configuraTono__();
        activarDac();
    }
}

if(zoneNewPressed( &zona_16))
{
    if (  ACTUALIZADOR->Audiorev  )
    {
        ACTUALIZADOR->Audiorev = 0;
        lanzaUFONO();
    }
}

if(zoneNewPressed( &zona_13))
{
    Modo_energetico =  0; // HP.
    __brilloAuto      =  0; // No hay brillo auto.
    Modo_brillo       =  3; // Brillo a tope.
    modificaPulso(  PWM6 ,  MODO_CICLO ,  60 ,  none ,  none ,  none );
    __brilloFade      =  0;
}

if(zoneNewPressed( &zona_14))
{
    Modo_energetico =  1; // LP.
    Modo_brillo      =  4; // Brillo auto.
    __brilloAuto      =  1; // Activo el brillo automático.
    __brilloFade      =  0; // No pueda apagarse la pantalla.
}

if(zoneNewPressed( &zona_15))
{
    Modo_energetico =  2; // ULP.
    Modo_brillo      =  4; // Brillo auto.
    __brilloAuto      =  1; // Activo el brillo automático.
    __brilloFade      =  1; // Que pueda apagarse la pantalla.
}

}

break;

case PANTALLA_MEDIDAS1:
    __pintaMedidas1__();
    if(zoneNewPressed( &zona_2))
    {
        ESTADO->CHART = PANTALLA_MEDIDAS2;
    }
}

```

```
        LCD_Clear(Black);
    }
    if(zoneNewPressed( &zona_3))
    {
        ESTADO->CHART = PANTALLA_INICIO;
        LCD_Clear(Black);
    }
    break;

case PANTALLA_MEDIDAS2:
    __pintaMedidas2__();
    if(zoneNewPressed( &zona_2))
    {
        ESTADO->CHART = PANTALLA_INICIO;
        LCD_Clear(Black);
    }
    if(zoneNewPressed( &zona_3))
    {
        ESTADO->CHART = PANTALLA_MEDIDAS1;
        LCD_Clear(Black);
    }
    break;

case PANTALLA_AJUSTES:
    __pintaAjustes__();
    if(zoneNewPressed( &zona_2))
    {
        ESTADO->CHART = PANTALLA_INICIO;
        LCD_Clear(Black);
    }
    if(zoneNewPressed( &zona_3))
    {
        ESTADO->CHART = PANTALLA_INICIO;
        LCD_Clear(Black);
    }
    if(zoneNewPressed( &zona_27m))
    {
        LPC_RTC->HOUR--;
    }
    if(zoneNewPressed( &zona_28m))
    {
        LPC_RTC->MIN--;
    }
    if(zoneNewPressed( &zona_29m))
    {
        LPC_RTC->SEC--;
    }
    if(zoneNewPressed( &zona_30m))
    {
        LPC_RTC->DOM--;
    }
    if(zoneNewPressed( &zona_27M))
    {
        LPC_RTC->HOUR++;
    }
    if(zoneNewPressed( &zona_28M))
    {
        LPC_RTC->MIN++;
    }
    if(zoneNewPressed( &zona_29M))
    {
        LPC_RTC->SEC++;
    }
    if(zoneNewPressed( &zona_30M))
    {
        LPC_RTC->DOM++;
    }
```

```

        }
        break;
    case PANTALLA_LOADING:
        break;
    case PANTALLA_VALORES:
        __pintaValores__();
        if(zoneNewPressed( &zona_2))
        {
            ESTADO->CHART = PANTALLA_INICIO;
            LCD_Clear(Black);
        }
        if(zoneNewPressed( &zona_3))
        {
            ESTADO->CHART = PANTALLA_INICIO;
            LCD_Clear(Black);
        }
        if(zoneNewPressed( &zona_27m))
        {
            MODIFICABLES.Min_servo_t--;
        }
        if(zoneNewPressed( &zona_28m))
        {
            MODIFICABLES.Max_servo_t--;
        }
        if(zoneNewPressed( &zona_29m))
        {
            MODIFICABLES.Min_servo_p -= 10;
        }
        if(zoneNewPressed( &zona_30m))
        {
            MODIFICABLES.Max_servo_p -= 10;
        }
        if(zoneNewPressed( &zona_27M))
        {
            MODIFICABLES.Min_servo_t++;
        }
        if(zoneNewPressed( &zona_28M))
        {
            MODIFICABLES.Max_servo_t++;
        }
        if(zoneNewPressed( &zona_29M))
        {
            MODIFICABLES.Min_servo_p += 10;
        }
        if(zoneNewPressed( &zona_30M))
        {
            MODIFICABLES.Max_servo_p += 10;
        }
        if(zoneNewPressed( &zona_31))
        {
            MODIFICABLES.Var_medida = 1 - MODIFICABLES.Var_medida;
        }
    default:
        break;
    }
}

// @function __configuraLCD__()
// @brief Esta función configura el TFT HY32B conectado al driver ILI9325C
// void __configuraLCD__( void )

```

```

{
    TP_Init();
    LCD_Initialization();
}
/*
//-----//-----//-----//
//      @function   __pintaInicio__()
//      @brief     Esta función pinta la pantalla de inicio.
//-----//-----//-----*/
void __pintaInicio__( void )
{
    squareButton(&zona_1 , (char *)Clock , Yellow , Green );
    squareButton(&zona_2 , ">" , Yellow , Green );
    squareButton(&zona_3 , "<" , Yellow , Green );
    squareButton(&zona_4 , "Valores" , Yellow , Green );
    squareButton(&zona_5 , "Ajustes" , Yellow , Green );
    squareButton(&zona_6 , "Nivel de brillo:" , Yellow , Green );
    switch ( Modo_brillo )
    {
        case 0:
            squareButton(&zona_7 , "1" , White , White );
            squareButton(&zona_8 , "2" , Yellow , Green );
            squareButton(&zona_9 , "3" , Yellow , Green );
            squareButton(&zona_10 , "4" , Yellow , Green );
            squareButton(&zona_11 , "A" , Yellow , Green );
            break;
        case 1:
            squareButton(&zona_7 , "1" , Yellow , Green );
            squareButton(&zona_8 , "2" , White , White );
            squareButton(&zona_9 , "3" , Yellow , Green );
            squareButton(&zona_10 , "4" , Yellow , Green );
            squareButton(&zona_11 , "A" , Yellow , Green );
            break;
        case 2:
            squareButton(&zona_7 , "1" , Yellow , Green );
            squareButton(&zona_8 , "2" , Yellow , Green );
            squareButton(&zona_9 , "3" , White , White );
            squareButton(&zona_10 , "4" , Yellow , Green );
            squareButton(&zona_11 , "A" , Yellow , Green );
            break;
        case 3:
            squareButton(&zona_7 , "1" , Yellow , Green );
            squareButton(&zona_8 , "2" , Yellow , Green );
            squareButton(&zona_9 , "3" , Yellow , Green );
            squareButton(&zona_10 , "4" , White , White );
            squareButton(&zona_11 , "A" , Yellow , Green );
            break;
        case 4:
            squareButton(&zona_7 , "1" , Yellow , Green );
            squareButton(&zona_8 , "2" , Yellow , Green );
            squareButton(&zona_9 , "3" , Yellow , Green );
            squareButton(&zona_10 , "4" , Yellow , Green );
            squareButton(&zona_11 , "A" , White , White );
            break;
    }
    squareButton(&zona_12 , "Play" , Yellow , Green );
    switch ( Modo_energetico )
    {
        case 0:
            squareButton(&zona_13 , "HP" , Red , Red );
            squareButton(&zona_14 , "LP" , Yellow , Green );
            squareButton(&zona_15 , "ULP" , Yellow , Green );
            break;
    }
}

```



```

        squareButton(&zona_31 , (char*)buffer , Red , Green );
    }
squareButton(&zona_27m , "_" , Yellow , Green );
squareButton(&zona_28m , "_" , Yellow , Green );
squareButton(&zona_29m , "_" , Yellow , Green );
squareButton(&zona_30m , "_" , Yellow , Green );
squareButton(&zona_27M , "+" , Yellow , Green );
squareButton(&zona_28M , "+" , Yellow , Green );
squareButton(&zona_29M , "+" , Yellow , Green );
squareButton(&zona_30M , "+" , Yellow , Green );
}
/*-----*/
//                                     //           //
//                                     //           //
// @function __pintaCargandoInicio__()
//                                     //           //
// @brief Esta función pinta la pantalla cargando incio.
//                                     //           //
//-----*/**
void __pintaCargandoInicio__( void )
{
    squareButton(&zona_lo0, "CARGANDO ..." , Blue , Green );
    squareButton(&zona_lo1, "Preparando el sistema..." , Blue , Green );
}
/*-----*/
//                                     //           //
//                                     //           //
// @function __pintaCargandoSeno__()
//                                     //           //
// @brief Esta función pinta la pantalla cargando seno.
//                                     //           //
//-----*/**
void __pintaCargandoSeno__(void )
{
    squareButton(&zona_lo0, "CARGANDO ..." , Blue , Green );
    squareButton(&zona_lo1, "Creando muestras..." , Blue , Green );
    squareBox( &zona_lo20, White );
}
/*-----*/
//                                     //           //
//                                     //           //
// @function __pintaCargandoConexion__()
//                                     //           //
// @brief Esta función pinta la pantalla cargando conexión.
//                                     //           //
//-----*/**
void __pintaCargandoConexion__( void )
{
    squareButton(&zona_lo0, "CARGANDO ..." , Blue , Green );
    squareButton(&zona_lo1, "Buscando conexion TCP..." , Blue , Green );
    squareBox( &zona_lo20, White );
    squareBox( &zona_lo21, White );
}
/*-----*/
//                                     //           //
//                                     //           //
// @function __pintaCargandoIniciando__()
//                                     //           //
// @brief Esta función pinta la pantalla cargando iniciando.
//                                     //           //
//-----*/**
void __pintaCargandoIniciando__( void )
{
    squareButton(&zona_lo0, "CARGANDO ..." , Blue , Green );
    squareButton(&zona_lo1, "Iniciando modulos..." , Blue , Green );
    squareBox( &zona_lo20, White );
}

```

```

        squareBox( &zona_lo21, White );
        squareBox( &zona_lo22, White );
    }

    /**
     * @function __pintaCargandoDone__()
     * @brief Esta función pinta la pantalla cargando hecho.
     */
    void __pintaCargandoDone__( void )
    {
        squareButton(&zona_lo0, "CARGADO" , Blue , Green );
        squareButton(&zona_lo1, "100%" , Blue , Green );
        squareBox( &zona_lo20, White );
        squareBox( &zona_lo21, White );
        squareBox( &zona_lo22, White );
        squareBox( &zona_lo22, White );
    }

    /**
     * @function __pintaMedidas1__()
     * @brief Esta función pinta la primera pantalla de medidas.
     */
    void __pintaMedidas1__( void )
    {
        squareButton(&zona_1 , (char *)Clock , Yellow , Green );
        squareButton(&zona_2 , ">" , Yellow , Green );
        squareButton(&zona_3 , "<" , Yellow , Green );
        squareButton(&zona_17 , "MEDIDAS ACTUALES" , Yellow , Green );
        if ( ACTUALIZADOR->TempRev )
        {
            sprintf((char*)buffer,"Altura: %0.02f m.", DATOS->Lugar.Altura);
            squareButton(&zona_18 , (char *)buffer , Yellow , Green );
        }
        squareButton(&zona_19 , "Vel. v.:" , Yellow , Green );
        squareButton(&zona_20 , "Humedad:" , Yellow , Green );
        squareButton(&zona_21 , "Claridad:" , Yellow , Green );
        squareButton(&zona_22 , "Indice UV:" , Yellow , Green );
        if ( ACTUALIZADOR->Anemometro )
        {
            sprintf((char*)buffer,"%0.02f mps", DATOS->VelViento);
            squareButton(&zona_23 , (char *)buffer , Yellow , Green );
            ACTUALIZADOR->Anemometro = 0;
        }
        if ( ACTUALIZADOR->TempRev )
        {
            sprintf((char*)buffer,"%0.02f %%" , DATOS->Humedad);
            squareButton(&zona_24 , (char *)buffer , Yellow , Green );
            ACTUALIZADOR->TempRev = 0;
        }
        sprintf((char*)buffer,"%0.02f LUX",DATOS->Brillo);
        squareButton(&zona_25 , (char *)buffer , Yellow , Green );
        sprintf((char*)buffer,"%0.02f UVs" , DATOS->IndiceUV);
        squareButton(&zona_26 , (char *)buffer , Yellow , Green );
    }

    /**
     * @function __pintaMedidas2__()
     */

```

```

//      @brief Esta función pinta la segunda pantalla de medidas.          //
//      //////////////////////////////////////////////////////////////////*****//
void  __pintaMedidas2__( void  )
{
    squareButton( &zona_1 , (char *)Clock , Yellow , Green );
    squareButton( &zona_2 , ">" , Yellow , Green );
    squareButton( &zona_3 , "<" , Yellow , Green );
    squareButton( &zona_32 , "Temperatura:" , Yellow , Green );
    if ( ACTUALIZADOR->TempRev )
    {
        sprintf((char*)buffer,"%0.02f dC", DATOS->Temperatura);
        squareButton( &zona_32n , (char *)buffer , Yellow , Green );
        sprintf((char*)buffer,"%0.02f mBar.", DATOS->Presion);
        squareButton( &zona_34n , (char *)buffer , Yellow , Green );
        ACTUALIZADOR->TempRev = 0;
    }
    // Digo que toca medir.
    switch ( (int)(10*(DATOS->Temperatura - MIN_TEMP)/(MAX_TEMP - MIN_TEMP)) )
    {
        case 0:
            squareBox( &zona_330 , Black);
            squareBox( &zona_331 , Black);
            squareBox( &zona_332 , Black);
            squareBox( &zona_333 , Black);
            squareBox( &zona_334 , Black);
            squareBox( &zona_335 , Black);
            squareBox( &zona_336 , Black);
            squareBox( &zona_337 , Black);
            squareBox( &zona_338 , Black);
            squareBox( &zona_339 , Black);
            break;
        case 1:
            squareBox( &zona_330 , White);
            squareBox( &zona_331 , Black);
            squareBox( &zona_332 , Black);
            squareBox( &zona_333 , Black);
            squareBox( &zona_334 , Black);
            squareBox( &zona_335 , Black);
            squareBox( &zona_336 , Black);
            squareBox( &zona_337 , Black);
            squareBox( &zona_338 , Black);
            squareBox( &zona_339 , Black);
            break;
        case 2:
            squareBox( &zona_330 , White);
            squareBox( &zona_331 , White);
            squareBox( &zona_332 , Black);
            squareBox( &zona_333 , Black);
            squareBox( &zona_334 , Black);
            squareBox( &zona_335 , Black);
            squareBox( &zona_336 , Black);
            squareBox( &zona_337 , Black);
            squareBox( &zona_338 , Black);
            squareBox( &zona_339 , Black);
            break;
        case 3:
            squareBox( &zona_330 , Yellow);
            squareBox( &zona_331 , Yellow);
            squareBox( &zona_332 , Yellow);
            squareBox( &zona_333 , Black);
            squareBox( &zona_334 , Black);
            squareBox( &zona_335 , Black);
            squareBox( &zona_336 , Black);
            squareBox( &zona_337 , Black);
            squareBox( &zona_338 , Black);
            squareBox( &zona_339 , Black);
    }
}

```

```
        break;
case 4:
    squareBox( &zona_330 , Yellow);
    squareBox( &zona_331 , Yellow);
    squareBox( &zona_332 , Yellow);
    squareBox( &zona_333 , Yellow);
    squareBox( &zona_334 , Black);
    squareBox( &zona_335 , Black);
    squareBox( &zona_336 , Black);
    squareBox( &zona_337 , Black);
    squareBox( &zona_338 , Black);
    squareBox( &zona_339 , Black);
    break;
case 5:
    squareBox( &zona_330 , Blue);
    squareBox( &zona_331 , Blue);
    squareBox( &zona_332 , Blue);
    squareBox( &zona_333 , Blue);
    squareBox( &zona_334 , Blue);
    squareBox( &zona_335 , Black);
    squareBox( &zona_336 , Black);
    squareBox( &zona_337 , Black);
    squareBox( &zona_338 , Black);
    squareBox( &zona_339 , Black);
    break;
case 6:
    squareBox( &zona_330 , Blue);
    squareBox( &zona_331 , Blue);
    squareBox( &zona_332 , Blue);
    squareBox( &zona_333 , Blue);
    squareBox( &zona_334 , Blue);
    squareBox( &zona_335 , Blue);
    squareBox( &zona_336 , Black);
    squareBox( &zona_337 , Black);
    squareBox( &zona_338 , Black);
    squareBox( &zona_339 , Black);
    break;
case 7:
    squareBox( &zona_330 , Green);
    squareBox( &zona_331 , Green);
    squareBox( &zona_332 , Green);
    squareBox( &zona_333 , Green);
    squareBox( &zona_334 , Green);
    squareBox( &zona_335 , Green);
    squareBox( &zona_336 , Green);
    squareBox( &zona_337 , Black);
    squareBox( &zona_338 , Black);
    squareBox( &zona_339 , Black);
    break;
case 8:
    squareBox( &zona_330 , Green);
    squareBox( &zona_331 , Green);
    squareBox( &zona_332 , Green);
    squareBox( &zona_333 , Green);
    squareBox( &zona_334 , Green);
    squareBox( &zona_335 , Green);
    squareBox( &zona_336 , Green);
    squareBox( &zona_337 , Green);
    squareBox( &zona_338 , Black);
    squareBox( &zona_339 , Black);
    break;
case 9:
    squareBox( &zona_330 , Red);
    squareBox( &zona_331 , Red);
    squareBox( &zona_332 , Red);
    squareBox( &zona_333 , Red);
```

```

squareBox( &zona_334 , Red);
squareBox( &zona_335 , Red);
squareBox( &zona_336 , Red);
squareBox( &zona_337 , Red);
squareBox( &zona_338 , Red);
squareBox( &zona_339 , Black);
break;
case 10:
    squareBox( &zona_330 , Red);
    squareBox( &zona_331 , Red);
    squareBox( &zona_332 , Red);
    squareBox( &zona_333 , Red);
    squareBox( &zona_334 , Red);
    squareBox( &zona_335 , Red);
    squareBox( &zona_336 , Red);
    squareBox( &zona_337 , Red);
    squareBox( &zona_338 , Red);
    squareBox( &zona_339 , Red);
break;
default:
    if ( DATOS->Temperatura > MIN_TEMP)
    {
        squareBox( &zona_330 , Red);
        squareBox( &zona_331 , Red);
        squareBox( &zona_332 , Red);
        squareBox( &zona_333 , Red);
        squareBox( &zona_334 , Red);
        squareBox( &zona_335 , Red);
        squareBox( &zona_336 , Red);
        squareBox( &zona_337 , Red);
        squareBox( &zona_338 , Red);
        squareBox( &zona_339 , Red);
    }
    if ( DATOS->Temperatura < MIN_TEMP)
    {
        squareBox( &zona_330 , Black);
        squareBox( &zona_331 , Black);
        squareBox( &zona_332 , Black);
        squareBox( &zona_333 , Black);
        squareBox( &zona_334 , Black);
        squareBox( &zona_335 , Black);
        squareBox( &zona_336 , Black);
        squareBox( &zona_337 , Black);
        squareBox( &zona_338 , Black);
        squareBox( &zona_339 , Black);
    }
};

squareButton(&zona_34 , "Presion:" , Yellow , Green );
switch ( (int)(10*(DATOS->Presion - MIN_PRES)/(MAX_PRES - MIN_PRES)) )
{
    case 0:
        squareBox( &zona_350 , Black);
        squareBox( &zona_351 , Black);
        squareBox( &zona_352 , Black);
        squareBox( &zona_353 , Black);
        squareBox( &zona_354 , Black);
        squareBox( &zona_355 , Black);
        squareBox( &zona_356 , Black);
        squareBox( &zona_357 , Black);
        squareBox( &zona_358 , Black);
        squareBox( &zona_359 , Black);
break;
    case 1:
        squareBox( &zona_350 , White);
        squareBox( &zona_351 , Black);

```

```
squareBox( &zona_352 , Black);
squareBox( &zona_353 , Black);
squareBox( &zona_354 , Black);
squareBox( &zona_355 , Black);
squareBox( &zona_356 , Black);
squareBox( &zona_357 , Black);
squareBox( &zona_358 , Black);
squareBox( &zona_359 , Black);
break;
case 2:
squareBox( &zona_350 , White);
squareBox( &zona_351 , White);
squareBox( &zona_352 , Black);
squareBox( &zona_353 , Black);
squareBox( &zona_354 , Black);
squareBox( &zona_355 , Black);
squareBox( &zona_356 , Black);
squareBox( &zona_357 , Black);
squareBox( &zona_358 , Black);
squareBox( &zona_359 , Black);
break;
case 3:
squareBox( &zona_350 , Yellow);
squareBox( &zona_351 , Yellow);
squareBox( &zona_352 , Yellow);
squareBox( &zona_353 , Black);
squareBox( &zona_354 , Black);
squareBox( &zona_355 , Black);
squareBox( &zona_356 , Black);
squareBox( &zona_357 , Black);
squareBox( &zona_358 , Black);
squareBox( &zona_359 , Black);
break;
case 4:
squareBox( &zona_350 , Yellow);
squareBox( &zona_351 , Yellow);
squareBox( &zona_352 , Yellow);
squareBox( &zona_353 , Yellow);
squareBox( &zona_354 , Black);
squareBox( &zona_355 , Black);
squareBox( &zona_356 , Black);
squareBox( &zona_357 , Black);
squareBox( &zona_358 , Black);
squareBox( &zona_359 , Black);
break;
case 5:
squareBox( &zona_350 , Blue);
squareBox( &zona_351 , Blue);
squareBox( &zona_352 , Blue);
squareBox( &zona_353 , Blue);
squareBox( &zona_354 , Blue);
squareBox( &zona_355 , Black);
squareBox( &zona_356 , Black);
squareBox( &zona_357 , Black);
squareBox( &zona_358 , Black);
squareBox( &zona_359 , Black);
break;
case 6:
squareBox( &zona_350 , Blue);
squareBox( &zona_351 , Blue);
squareBox( &zona_352 , Blue);
squareBox( &zona_353 , Blue);
squareBox( &zona_354 , Blue);
squareBox( &zona_355 , Blue);
squareBox( &zona_356 , Black);
squareBox( &zona_357 , Black);
```

```
    squareBox( &zona_358 , Black);
    squareBox( &zona_359 , Black);
    break;
case 7:
    squareBox( &zona_350 , Green);
    squareBox( &zona_351 , Green);
    squareBox( &zona_352 , Green);
    squareBox( &zona_353 , Green);
    squareBox( &zona_354 , Green);
    squareBox( &zona_355 , Green);
    squareBox( &zona_356 , Green);
    squareBox( &zona_357 , Black);
    squareBox( &zona_358 , Black);
    squareBox( &zona_359 , Black);
    break;
case 8:
    squareBox( &zona_350 , Green);
    squareBox( &zona_351 , Green);
    squareBox( &zona_352 , Green);
    squareBox( &zona_353 , Green);
    squareBox( &zona_354 , Green);
    squareBox( &zona_355 , Green);
    squareBox( &zona_356 , Green);
    squareBox( &zona_357 , Green);
    squareBox( &zona_358 , Black);
    squareBox( &zona_359 , Black);
    break;
case 9:
    squareBox( &zona_350 , Red);
    squareBox( &zona_351 , Red);
    squareBox( &zona_352 , Red);
    squareBox( &zona_353 , Red);
    squareBox( &zona_354 , Red);
    squareBox( &zona_355 , Red);
    squareBox( &zona_356 , Red);
    squareBox( &zona_357 , Red);
    squareBox( &zona_358 , Red);
    squareBox( &zona_359 , Black);
    break;
case 10:
    squareBox( &zona_350 , Red);
    squareBox( &zona_351 , Red);
    squareBox( &zona_352 , Red);
    squareBox( &zona_353 , Red);
    squareBox( &zona_354 , Red);
    squareBox( &zona_355 , Red);
    squareBox( &zona_356 , Red);
    squareBox( &zona_357 , Red);
    squareBox( &zona_358 , Red);
    squareBox( &zona_359 , Red);
    break;
default:
    if( DATOS->Presion > MAX_PRES)
    {
        squareBox( &zona_350 , Red);
        squareBox( &zona_351 , Red);
        squareBox( &zona_352 , Red);
        squareBox( &zona_353 , Red);
        squareBox( &zona_354 , Red);
        squareBox( &zona_355 , Red);
        squareBox( &zona_356 , Red);
        squareBox( &zona_357 , Red);
        squareBox( &zona_358 , Red);
        squareBox( &zona_359 , Red);
    }
    if( DATOS->Presion < MIN_PRES)
```

```

    {
        squareBox( &zona_350 , Black);
        squareBox( &zona_351 , Black);
        squareBox( &zona_352 , Black);
        squareBox( &zona_353 , Black);
        squareBox( &zona_354 , Black);
        squareBox( &zona_355 , Black);
        squareBox( &zona_356 , Black);
        squareBox( &zona_357 , Black);
        squareBox( &zona_358 , Black);
        squareBox( &zona_359 , Black);
    }
}

/*
// @function squareButton()
// @brief Dibuja un botón cuadrado, con texto y colores.
// */
void squareButton(screenZone_t* zone, char * text, uint16_t textColor, uint16_t lineColor)
{
    LCD_DrawLine( zone->x, zone->y, zone->x + zone->size_x - 1, zone->y, lineColor);
    LCD_DrawLine( zone->x, zone->y, zone->x, zone->y + zone->size_y - 1, lineColor);
    LCD_DrawLine( zone->x, zone->y + zone->size_y - 1, zone->x + zone->size_x - 1, zone->y + zone->size_y - 1, lineColor);
    LCD_DrawLine( zone->x + zone->size_x - 1, zone->y, zone->x + zone->size_x - 1, zone->y + zone->size_y - 1, lineColor);
    GUI_Text(zone->x + zone->size_x/2 - (strlen(text)/2)*8, zone->y + zone->size_y/2 - 8, (uint8_t*) text, textColor, Black);
}

/*
// @function squareBox()
// @brief Dibuja un cuadrado de un color.
// */
void squareBox(screenZone_t* zone, uint16_t color)
{
    int i;
    for (i = 0; i < (zone->size_x - 4) ; i++)
    {
        LCD_DrawLine( zone->x + i + 2, zone->y + 2, zone->x + i + 2, zone->y + zone->size_y - 2, color);
    }
}

/*
// @function checkTouchPanel()
// @brief Verifica se si ha tocado la pantalla.
// */
void checkTouchPanel(void)
{
    Coordinate* coord;

    coord = Read_Ads7846();

    if (coord > 0) {
        getDisplayPoint(&display, coord, &matrix );
        pressedTouchPanel = 1;
    }
    else
        pressedTouchPanel = 0;
}

```

```

}

/*
//-----// //-----//
//  @function  zoneNewPressed() // //-----//
//  @brief    Verifica si se ha presionado una cierta zona de la pantalla. // //
//  @zone   Zona a comprobar. // //
//  @return  0 - Si no se ha producido un toque. // //
//          1 - Si se ha producido un toque. // //
//-----*/ //-----//

int8_t zoneNewPressed(screenZone_t * zone)
{
    if(pressedTouchPanel == 1) {

        if ((display.x > zone->x) && (display.x < zone->x + zone->size_x) &&
            (display.y > zone->y) && (display.y < zone->y + zone->size_y))
        {
            if (zone->pressed == 0)
            {
                zone->pressed = 1;
                return 1;
            }
            return 0;
        }
        /* @MOD: Esto lo he añadido yo */
        if (contadorLUZ >= (FREQ_OVERFLOW_SYSTICK * MODIFICABLES.TiempoBrillo)) // Si se ha activado el apagar pantalla...
        {
            modificaPulso ( PWM6, MODO_CICLO, 60, none, none, none, none ); // La enciendo como
si hubiese habido un reset.
            Modo_brillo = 3;
            if ( Modo_energetico > 1 )
            {
                __brilloAuto = 1;
                Modo_brillo = 4;
                if ( Modo_energetico == 2 )
                {
                    __brilloFade = 1;
                }
            }
        }
        contadorLUZ = 0; // Reseteo el contador de apagar la pantalla.
    }

    zone->pressed = 0;
    return 0;
}
/*
//-----// //-----//
//  @end   ENDFILE. // //
//-----*/ //-----//

```

Configura.h

```
/**-----//  
// @filename configura.h //  
// @version 0.00 //  
// @author Alberto Palomo Alonso //  
// @brief Cabecera para el código de configuración.c //  
// @category Principal. //  
// @map @include //  
// @private //  
// @functdef //  
// @end //  
// //  
// //  
//-----//  
// //  
// @include Estos son los archivos utilizados con el código de configuración. //  
// //  
//-----*/  
// PWM  
#ifndef PWM  
#define PWM  
#include "PWM.h"  
#endif  
#ifndef GLCD  
#define GLDC  
#include "GLCD.h"  
#include "TouchPanel.h"  
#include "menu.h"  
#endif  
#ifndef STATECHART  
#define STATECHART  
#include "Statechart.h"  
#endif  
#ifndef RTC  
#define RTC  
#include "RTC.h"  
#endif  
#ifndef SYSTEMSYMBOLS  
#define SYSTEMSYMBOLS  
#include "Systemsymbols.h"  
#endif  
#ifndef WDT  
#define WDT  
#include "WDT.h"  
#endif  
#ifndef HTTP_SOURCE  
#define HTTP_SOURCE  
#include "HTTP_SOURCE.h"  
#endif  
#ifndef TIMERS  
#define TIMERS  
#include "Timers.h"  
#endif  
#ifndef ANEMOMETRO  
#define ANEMOMETRO
```

```

#include "Anemometro.h"
#endif
#ifndef DAC
#define DAC
#include "DAC.h"
#endif
#ifndef UVA30A
#define UVA30A
#include "UVA30A.h"
#endif
#ifndef UFONO
#define UFONO
#include "uFono.h"
#endif
#ifndef LUT
#define LUT
#include "LUT.h"
#endif
#ifndef DMA
#define DMA
#include "DMA.h"
#endif
#ifndef I2C
#define I2C
#include "I2C.h"
#endif
#ifndef UART0
#define UART0
#include "UART0.h"
#endif
#ifndef UART3
#define UART3
#include "UART3.h"
#endif
/**-
// @private Estos son los símbolos correspondientes a la configuración. //
// */ **
#define ACTIVOS_TODOS2 0x003F
#define ACTIVOS_1_2 0x0001
#define ACTIVOS_2_2 0x0002
#define ACTIVOS_1_1 0x0100
#define ACTIVOS_2_1 0x0200
#define ACTIVOS_6_2 0x0020
#define ACTIVOS_6_1 0x2000

#define MAXIMO_PRESION 1500
#define MINIMO_PRESION 500
#define MAXIMO_TEMPERATURA 50
#define MINIMO_TEMPERATURA -10
/**-
// */ **
// @funcdef Estas son las funciones correspondientes a la configuración. //
// */ **
void __configuraPrograma__( void );
void __iniciaVariables__( void );
/**-
// */ **
// @end ENDFILE. //
// */ **

```

Configura.c

```
/**-----//  
//  @filename  configura.c                                //  
//  @version   0.00                                     //  
//  @author    Alberto Palomo Alonso                   //  
//                                                       //  
//  @brief     Código que configura y llama a las funciones de configuración para hacer un Setup del programa.//  
//                                                       //  
//  @category  Principal.                            //  
//                                                       //  
//  @map      @include  
//            @funcion  
//            @end  
//                                                       //  
//                                                       //  
//-----//  
//  
//  @include  Estos son los archivos utilizados en el código de configuración.          //  
//                                                       //  
//-----**/  
#ifndef  CONFIGURA  
#define  CONFIGURA  
#include "configura.h"  
#endif  
extern State_t    * ESTADO;  
extern misDatos_t * DATOS;  
extern modificables_t MODIFICABLES;  
/**-----//  
//  
//  @funcion  Esta función configura el programa entero.          //  
//                                                       //  
//  @ref      __configuraPWM__      -> PWM.h                  //  
//            modificaPulso       -> PWM.h                  //  
//            __configuraLCD__     -> Statechart.h           //  
//                                                       //  
//-----**/  
void __configuraPrograma__( void )  
{  
    __configuraLCD__ ();  
    LCD_Clear(Black);  
    __pintaCargandoInicio__ ();  
    __iniciaVariables__ ();  
    LCD_Clear(Black);  
    __pintaCargandoSeno__ ();  
    // Añadir generación de variables de alto costo computacional.  
    LCD_Clear(Black);  
    __pintaCargandoConexion__();  
    __configuraWEB__ ();  
    LCD_Clear(Black);  
    __pintaCargandoIniciando__();  
    __configuraSysTick__();  
    __configuraTimer0__();  
    __configuraLDR__();  
    __configuraUVA30A__();
```

Alberto Palomo Alonso.

```

__configuraUFONO__    ();
__configuraRTC__      ();
__configuraPWM__      ( Fpwm , ACTIVOS_2_1 | ACTIVOS_6_1 );
modificaPulso ( PWM2, MODO_SERVO , none , 90 , MINIMO_SERVO , MAXIMO_SERVO );
modificaPulso ( PWM6, MODO_CICLO , 50 , none , none , none );
__configuraWDT__      ();
__configuraDAC__      ();
__configuraDMA__      ();
__configuraOW__        ();
__configuraAnemometro__();
__configuraUART0__     ();
// __configuraUART3__     ();
__configuraI2C__       ();
#ifndef DEBUG
// TouchPanel_Calibrate();
#endif
LCD_Clear(Black);
__pintaCargandoDone__();
LCD_Clear(Black);
ESTADO->CHART = PANTALLA_INICIO;
}
/**-----//  

//-----//  

//-----@funcion __iniciaVariables__() //  

//-----@brief Esta función inicia las variables del sistema para que tengan un momento inicial. //  

//-----**/  

void __iniciaVariables__()
{
ESTADO->CHART = PANTALLA_LOADING;
DATOS->Temperatura      = 0;
DATOS->Humedad          = 0;
DATOS->Presion           = 0;
DATOS->VelViento         = 0;
DATOS->IndiceUV          = 0;
DATOS->Lugar.Altura       = 0;
DATOS->Lugar.Longitud     = 0;
DATOS->Lugar.Latitud      = 0;

MODIFICABLES.Max_servo_t = (float)MAXIMO_TEMPERATURA;
MODIFICABLES.Min_servo_t = (float)MINIMO_TEMPERATURA;
MODIFICABLES.Max_servo_p= (float)MAXIMO_PRESION;
MODIFICABLES.Min_servo_p= (float)MINIMO_PRESION;
MODIFICABLES.TiempoBrillo = 10;
MODIFICABLES.Var_medida   = 0; // 0 la temperatura, 1 la presión.
}
/**-----//  

//-----//  

//-----@end ENDFILE. //-----**/  

//-----//
```

I2C.h

```
//  @filename  I2C.h                                //  //
//  @version   0.00                                 //  //
//  @author    Alberto Palomo Alonso               //  //
//  @brief     Esta es la cabecera que recoge la comunicación por I2C.      //
//  @category  Opcional.                         //  //
//  @map      @include                           //  //
//          @function                          //  //
//          @end                               //  //
//-----//-----//-----//-----//-----//-----//
//  @include   Includes pertenecientes a la comunicación por I2C.           //
//-----//-----**///
#ifndef LPC17XX
#define  LPC17XX
#include "LPC17XX.H"
#endif
#ifndef SYSTEMSYMBOLS
#define  SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
#ifndef MATH
#define  MATH
#include <math.h>
#endif

typedef struct
{
    short ac1;
    short ac2;
    short ac3;
    unsigned short ac4;
    unsigned short ac5;
    unsigned short ac6;
    short b1;
    short b2;
    short mb;
    short mc;
    short md;
}BMP_t;
/**-
//-----//-----//-----//-----//-----//-----//
//  @private   Estos son los símbolos correspondientes al protocolo I2C.           //
//-----//-----**///
#define  SDA          0    // Pin 0
#define  SCL          1    // Pin 1
#define  DELAYTOTAL   100  // Ajustar.

#define  BMP_ADD      119 // Cambia en función del sensor.
#define  REGISTRO_PRESION 0xF6 // Acceso al registro de presión.
#define  REGISTRO_TEMPERATURA 0x

#define  READ         1
#define  WRITE        0
#define  SACK         0
#define  NACK         1
#define  PRESION_BMP  10
#define  TEMPERATURA_BMP 11
```

```
#define AC1          0xAA
#define AC2          0xAC
#define AC3          0xAE
#define AC4          0xB0
#define AC5          0xB2
#define AC6          0xB4
#define B1           0xB6
#define B2           0xB8
#define MB           0xBA
#define MC           0xBC
#define MD           0xBE
/*-----*/
//                                     //           //
//      @funcdef   Estas son las funciones correspondientes al protocolo I2C.           //
//                                     //           //
//-----*/                           //
void __configuraI2C__ ( void )      ;
void irRegistro ( uint8_tREG)      ;
void pedirDato ( void )          ;
void pedirDatoReg ( uint8_tREG)   ;
void __calibraBMP ( void )        ;
int32_t calculaTemperatura( void ) ;
int32_t calculaPresion ( void )   ;
void medirBMP ( void )          ;
uint16_t obtenerDato( uint8_tREG );
void mandaDato ( uint8_tREG , uint8_tDATA);

void I2Cdelay ( void )          ;
void I2CSendAddr ( uint8_taddr , uint8_trw );
void I2CSendByte ( uint8_tbyte )  ;
uint8_t I2CGetByte ( uint8_tAACK ) ;
void I2CSendStop ( void )       ;
/*-----*/
//                                     //           //
//      @end      ENDFILE.           //
//                                     //           //
//-----*/                           //
```

I2C.c

```
**-
//      @filename  I2C.c           //
//      @version   0.00           //
//      @author    Alberto Palomo Alonso           //
//                                     //           //           //
```

```

//      @brief  Este es el programa que recoge la comunicación por I2C.          //
//      @category Opcional.                                //  //
//      @map      @include                                //  //
//                  @variables                            //  //
//                  @funcion                             //  //
//                  @end                                 //  //
//                                                     //  //
//-----//                                                     //  //
//      @include    Includes pertenecientes a la comunicación por I2C.           //  //
//-----//                                                     **/  //
#ifndef I2C
#define I2C
#include "I2C.h"
#endif
/*-----//  //
//      @variables    Variables del fichero.          //  //
//-----//                                                     **/  //
// Variables globales y externas.
BMP_t COEF;
extern misDatos_t * DATOS;
uint8_t TemperaturaConBmp = 0;
uint8_t LecturaBMPO ;
uint8_t LecturaBMP1 ;
uint16_t LecturaBMP ;
float temperatura;
float presion;
/*-----//  //
//      @function   __configural2C__()
//      @brief     Esta función es la que configura el protocolo I2C.           //  //
//-----//                                                     **/  //
void __configural2C__( void )
{
    __calibraBMP();
}
void __calibraBMP()
{
    I2CSendAddr( BMP_ADD , WRITE );
    I2CSendByte( AC1 );
    I2CSendAddr( BMP_ADD , READ );
    COEF.ac1 = I2CGetByte( SACK ) << 8;
    COEF.ac1 |= I2CGetByte( NACK );
    I2CSendStop();

    I2CSendAddr( BMP_ADD , WRITE );
    I2CSendByte( AC2 );
    I2CSendAddr( BMP_ADD , READ );
    COEF.ac2 = I2CGetByte( SACK ) << 8;
    COEF.ac2 |= I2CGetByte( NACK );
    I2CSendStop();

    I2CSendAddr( BMP_ADD , WRITE );
    I2CSendByte( AC3 );
    I2CSendAddr( BMP_ADD , READ );
    COEF.ac3 = I2CGetByte( SACK ) << 8;
}

```

```

COEF.ac3 |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( AC4 );
I2CSendAddr( BMP_ADD , READ );
COEF.ac4 = I2CGetByte( SACK ) << 8;
COEF.ac4 |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( AC5 );
I2CSendAddr( BMP_ADD , READ );
COEF.ac5 = I2CGetByte( SACK ) << 8;
COEF.ac5 |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( AC6 );
I2CSendAddr( BMP_ADD , READ );
COEF.ac6 = I2CGetByte( SACK ) << 8;
COEF.ac6 |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( B1 );
I2CSendAddr( BMP_ADD , READ );
COEF.b1 = I2CGetByte( SACK ) << 8;
COEF.b2 |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( B2 );
I2CSendAddr( BMP_ADD , READ );
COEF.b2 = I2CGetByte( SACK ) << 8;
COEF.b2 |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( MB);
I2CSendAddr( BMP_ADD , READ );
COEF.mb = I2CGetByte( SACK ) << 8;
COEF.mb |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( MC);
I2CSendAddr( BMP_ADD , READ );
COEF.mc = I2CGetByte( SACK ) << 8;
COEF.mc |= I2CGetByte( NACK );
I2CSendStop();

I2CSendAddr( BMP_ADD , WRITE );
I2CSendByte( MD);
I2CSendAddr( BMP_ADD , READ );
COEF.md = I2CGetByte( SACK ) << 8;
COEF.md |= I2CGetByte( NACK );
I2CSendStop();
}

/**-----//  

//  

//      @function    procesarDatos()  

//  

//      @brief      Procesa el dato guardado en la variable LecturaBMP.  

//

```

```

//-----*/
void procesarDatos( uint8_tTipo )
{
    DATOS->Presion = presion*0.01;
    DATOS->Temperatura = temperatura;
    DATOS->Lugar.Altura = (float)44330*( 1 - pow((presion / 101325 ),(1/5.255)));
}
//-----*/
//                                     //           //
// @function     pedirDatosReg()          //           //
// @brief       Pide un dato al sensor I2C de 16 bits especificando el registro.      //
// //                                     //           //
//-----*/
void pedirDatosReg( uint8_tREG )
{
    irRegistro(REG);
    pedirDatos();
    procesarDatos(PRESION_BMP);
}
//-----*/
//                                     //           //
// @function     pedirDatos()            //           //
// @brief       Pide un dato al sensor I2C de 16 bits.                                //
// //                                     //           //
//-----*/
void pedirDatos( void )
{
    I2CSendAddr( BMP_ADD, READ );           // Mandar la dirección en modo lectura.
    LecturaBMP0 = I2CGetByte( SACK );        // Leo el primer byte.
    LecturaBMP1 = I2CGetByte( NACK );        // Leo el segundo byte.
    I2CSendStop();                         // Mando el fin de la comunicación.
    LecturaBMP = (LecturaBMP0<<8) | LecturaBMP1; // Todo al buffer de 16 bits.
}
//-----*/
//                                     //           //
// @function     irRegistro()           //           //
// @brief       Accede al registro REG de la memoria EEPROM.                          //
// //                                     //           //
//-----*/
void irRegistro( uint8_tREG ) // Acceso al registro REG del sensor.
{
    I2CSendAddr( BMP_ADD, WRITE); // Selecciono BMP.
    I2CSendByte( REG );         // Selecciono el registro de presión.
}
//-----*/
//                                     //           //
// @function     bmp180_get_pressure()      //           //
// @brief       No hablo el mismo idioma que el creador del driver, pero parece que devuelve la presión.  //
// @ref        Extraido de https://github.com/BoschSensortec/BMP180_driver; referido por la datasheet.   //
//-----*/
uint16_t obtenerDatos( uint8_tREG )
{
    uint16_t RETVAL;
    I2CSendAddr( BMP_ADD, WRITE );
    I2CSendByte( REG );
    I2CSendAddr( BMP_ADD, READ );
}

```

```

    RETVAL = I2CGetByte( SACK ) << 8;
    RETVAL |= I2CGetByte( NACK );
    I2CSendStop();
    return RETVAL;
}
void mandaDato ( uint8_tREG , uint8_tDATA)
{
    I2CSendAddr( BMP_ADD , WRITE );
    I2CSendByte( REG );
    I2CSendByte( DATA );
    I2CSendStop();
}

void medirBMP()
{
    int i;
    long UT, UP, X1, X2, X3, B3, B5, B6, T, p;
    unsigned long B4, B7;
    mandaDato ( 0xF4 , 0x2E );
    //Espera 4.7ms.
    for ( i = 0; i < 1000; i++)
    {
        I2Cdelay();
    }
    //Espera activa corta!
    UT = obtenerData ( 0xF6 );
    mandaDato ( 0xF4 , 0x34 );
    //Esperar 4.7ms.
    for ( i = 0; i < 1000; i++)
    {
        I2Cdelay();
    }
    //Espera activa corta!
    UP = obtenerData ( 0xF6 );
    X1 = (UT - COEF.ac6) * COEF.ac5 / 32768;
    X2 = COEF.mc * 2048 / (X1 + COEF.md);
    B5 = X1 + X2;
    T = ((B5 + 8) >> 4);

    B6 = B5 - 4000;
    X1 = (COEF.b2 * ((B6 * B6) >> 12)) >> 11;
    X2 = (COEF.ac2 * B6) >> 11;
    X3 = X1 + X2;
    B3 = ((COEF.ac1 * 4 + X3) + 2) / 4;
    X1 = (COEF.ac3 * B6) >> 13;
    X2 = (COEF.b1 * ((B6 * B6) >> 12)) >> 16;
    X3 = (X1 + X2 + 2) >> 2;
    B4 = COEF.ac4 * (unsigned long)(X3 + 32768) >> 15;
    B7 = ((unsigned long)UP - B3)*(50000);

    if (B7 < 0x80000000)
    {
        p = (B7*2) / B4;
    }
    else
    {
        p = (B7 / B4) * 2;
    }

    X1 = (p >> 8)*(p >> 8);
    X1 = (X1 * 3038 >> 16);
    X2 = (-7357 * p) >> 16;
    p = p + ((X1 + X2 + 3791) >> 4);
    // temperatura = (float)(28.0/107.0)*((float)T)/10;
    // presion = (float)(936.0/1150.0)*(float)p;
    temperatura = ((float)T)/10;
}

```

```

    presion      = (float)p;
    procesarDato(0);
}
/**-----//  

//  

//  @end      ENDFILE.  

//-----*/
```

Anemometro.h

```

/**-----//  

//  @filename  Anemometro.h  

//  @version   0.00  

//  @author    Alberto Palomo Alonso  

//  

//  @brief     Esta es la cabecera donde se declara todo lo utilizado en el anemómetro.  

//  

//  @category  Opcional.  

//  

//  @map      @include  

//            @private  

//            @funcdef  

//            @end  

//  

//-----//  

//  

//  @include   Includes pertenecientes al módulo del PWM.  

//  

//-----*/  

#ifndef LPC17XX
#define  LPC17XX
#include "LPC17XX.H"
#endif
#ifndef SYSTEMSYMBOLS
#define  SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
#ifndef TIMERS
#define  TIMERS
#include "Timers.h"
#endif
/**-----//  

//  

//  @private   Estos son los símbolos correspondientes al anemómetro.  

//  

//-----*/  

#define PULSOS_VUELTA 2
#define DIAMETRO_ANEMOMETRO 14 //En centímetros.
#define PULSOS_CENTIMETRO (float)PULSOS_VUELTA/((float)PI*(float)DIAMETRO_ANEMOMETRO)
#define CTCR_MASCARA 0x0// Dejar al TC contando.
#define CCR_MASCARA_EN 0x5// Generar interrupción.
#define CCR_MASCARA_DIS 0x4// Desactivar interrupción.
#define WARMUP_CICLOS 4 // Ciclos de calentamiento.
```

```
define CAPTURE_FUNCION 0x3// Capture 1.0.
#define PULL_UP      0x0// El pull.
#define CAP10_IR     0x10 // El IR de CAP1.0
#define MR1_IR       0x00000002 // El IR del MR1.
#define MR2_IR       0x00000005 // El IR del MR2 QUE SE JUNTA CON LA DEL 0.
/**_
//                                     //_
// @funcdef    Estas son las funciones correspondientes al anemómetro.          //
// //_____*_**_*/_
void __configuraAnemometro__( void );
void mideAnemometro      ( void );
/**_
//                                     //_
// @end    ENDFILE.                //_
// //_____*_**_*/_
```

Anemometro.c

```

/*
//  @filename  Anemometro.c
//  @version   2.00
//  @author    Alberto Palomo Alonso
//
//  @brief    Este es el programa donde se encuentran las funciones correspondientes al
//            anemómetro de la estación.
//
//  @category Medida.
//
//  @map      @include
//            @variables
//            @function
//            @end
//
//  @include  Includes pertenecientes al módulo del anemómetro.
//
//  @variables Variables del fichero.
//
uint8_t CAPcont = 2*PULSOS_VUELTA;
uint8_t SLAYERcont = 0;
uint32_t CLKbuff[] = {0, 0};
extern misDatos_t * DATOS;
*/

```

```

extern actualizador_t* ACTUALIZADOR;
float aux_viento = 0;
<*/-----//  

//-----//  

//-----//  

//-----//  

//-----//  

void __configuraAnemometro()  

{  

    LPC_PINCON->PINMODE3     &= ~(PULL_UP << (2*2)); // PULL_UP  

    LPC_PINCON->PINMODE3     |= PULL_UP << (2*2); // PULL_UP  

    LPC_PINCON->PINSEL3      &= ~((CAPTURE_FUNCION) << (2*2)); // CAPTURE1.0  

    LPC_PINCON->PINSEL3      |= (CAPTURE_FUNCION) << (2*2); // CAPTURE1.0  

    LPC_TIM1->CTCR          |= CTCR_MASCARA; // Flanco de subida.  

    LPC_TIM1->CCR           |= CCR_MASCARA_EN; // Inicio con interrupción.  

    LPC_SC->PCONP          |= TIMER1_BIT; // Activo el módulo del timer 1.  

    LPC_TIM1->PR             = 0; // Sin prescaler.  

    LPC_TIM1->TCR           |= RESET_TIMER_TCR; // Reseteo el contador.  

    LPC_TIM1->TCR           &= ~RESET_TIMER_TCR; // Reseteo el contador.  

    LPC_TIM1->TCR           |= 0x1; // Que cuente.  

    LPC_PINCON->PINSEL3     |= 0x3 << 4; // Entrada como CAP1.0.  

    NVIC_EnableIRQ( TIMER1 IRQn );  

}  

<*/-----//  

//-----//  

//-----//  

//-----//  

//-----//  

void mideAnemometro()  

{  

    /** @WARNING: Esto me parece un poco sucio, pero es la única manera de que no se generen  

    interrupciones espúrias utilizando únicamente recursos software. Idealmente  

    no debería usar delays y mucho menos en interrupciones. Si no uso esto se generan  

    varias interrupciones por flanco debido al ruido. Esto se arregla con un condensador  

    a masa en la entrada, pero es perder recursos hardware y he decidido perderlos  

    mediante software. */  

    int i;  

    for(i = 0; i < 30000; i++) {}  

    /** @TODO Poner condensadores entre Vin y masa del capture 1.0 para evitar doble int en flancos ascendentes. */  

    LPC_TIM1->IR = 1 << 4; // Desactivo la interrupción.  

    CAPcont++; // Incremento el contador de pulsos.  

    if ( CAPcont >= PULSOS_VUELTA ) /** @WARNING: Se generan UN interrupciones por pulso y 2*PULSOS_VUELTA por  

    vuelta. */  

    {  

        CLKbuff[1] = CLKbuff[0]; // Almaceno el valor anterior.  

        CLKbuff[0] = LPC_TIM1->CRO; // Cargo el valor actual.  

        CAPcont = 0; // Reseteo el contador de pulsos.  

        aux_viento = Ftick*(float)PI*(float)DIAMETRO_ANEMOMETRO/((float)100*(float)((uint32_t)CLKbuff[0] -  

        (uint32_t)CLKbuff[1])); // Metros / segundo.  

        SLAYERcont++; // Hay warmup, aumento el slayer.  

        if ( SLAYERcont == WARMUP_CICLOS )  

        {  

            LPC_TIM1->CCR |= ~CCR_MASCARA_DIS; // Slay capture. OJO: QUE HAY QUE REVIVIRLO.  

            SLAYERcont = 0; // Reseteo el slayer.  

            DATOS->VelViento = aux_viento; // Guardo el valor calentado.  

            ACTUALIZADOR->AnemometroRev = 0; // Digo que he medido al timer.  

            ACTUALIZADOR->Anemometro = 1; // Digo que he medido al statechart.  

        }  

        else  

        {  

    }
}

```

Alberto Palomo Alonso.

```
LPC_TIM1->CCR |= CCR_MASCARA_EN; // Si no, está activado.  
}  
}  
}  
} //  
//  
// @end ENDFILE.  
// //  
// ***//
```

LDR.h

```

//                                            //
//-----*/                                     */
#define PCONP_ADC_ON (1<<12)                //
#define PINSEL_ADC01 (1<<16)                 //
#define PINMODE_ADC01(3<<16)                  //
#define BRUST_PIN (1<<16)                     //
#define SEL_CANAL1 (1<<1)                      //
#define SEL_CANAL_GLOBAL (1<<8)                //
#define ADC_POWER (1<<21)                      //
#define ADC_START (0x6<<24)                   //
#define CLK_DIV_MAX(0xFF<<8)                  //

#define RESISTENCIA_PULL 70.00                 //
#define LDRRESISTENCIA_MAX 100                 //
#define LDRRESISTENCIA_MIN 1                    //
#define BRILLO_MAX 100                         //
#define BRILLO_MIN 1                           //

#define VREF 1.2                                //
#define VINDICE 10                             //

/*-----*/                                     //
//                                            //
//      @funcdef   Estas son las funciones correspondientes a la configuración.    //
//-----*/                                     //
void __configuraLDR__ ( void );               //
void ponAudioDMA ( void );                  //

/*-----*/                                     //
//                                            //
//      @end      ENDFILE.                  //
//-----*/                                     //

```

LDR.c

```

/*-----*/                                     //
//      @filename  LDR.c                      //
//      @version   0.00                        //
//      @author    Alberto Palomo Alonso     //
//-----*/                                     //
//      @brief     Código fuente que contiene las funciones para LDR (ADC). //

```

```

//          // @category Periférico.          //
//          @map      @include           //
//                  @VARIABLES          //
//                  @funcion            //
//                  @end                //
//          //                                //
//          //-----@include      Estos son los archivos utilizados en el código de LDR.          //
//          //                                //          //
//          //-----**/                      //
#ifndef LDR
#define LDR
#include "LDR.h"
#endif
/*-----*/
//          //                                //
//          //-----@variables     Variables del fichero.          //
//          //                                //          //
//          //-----**/                      //
extern misDatos_t * DATOS;
float BUFFER_BRILLO = 0;
float BUFFER_UVA = 0;
extern actualizador_t * ACTUALIZADOR;
extern uint8_t YaPuedesMedir;
uint32_t contador;
uint8_t AUDIO[MUESTRAS_AUDIO];
/*-----*/
//          //                                //
//          //-----@funcion    __configuraLDR__()
//          //                                //          //
//          //-----@brief Esta función configura el ADC y el LDR          //
//          //                                //          //
//          //-----**/                      //
void __configuraLDR_()
{
    LPC_SC->PCONP |= PCONP_ADC_ON; // Enciendo el ADClock.
    LPC_PINCON->PINSEL1 |= PINSEL_ADC01; // ADO.1
    LPC_PINCON->PINMODE1 &= ~PINMODE_ADC01; // ADO.1
    LPC_ADC->ADCR |= BRUST_PIN // Modo ráfaga.
    | SEL_CANAL1 // ADO.1 activado.
    | ADC_POWER // Empiezo ENCENDIENDO el ADC.
    | CLK_DIV_MAX; // Clkdiv hace que Fadc = Fclk/256, inferior al umbral de 13MHz. (Ojo: clkdiv = 0 implica
que no funcione en placa).
    LPC_ADC->ADINTEN |= SEL_CANAL1; // Genera interrupción el canal 1. (Debería ser el penúltimo)
    ACTUALIZADOR->LDRev = 1; // Inicia para activar.
    LPC_ADC->ADINTEN &= ~(SEL_CANAL_GLOBAL); // Apago la interrupción global.
    NVIC_EnableIRQ(ADC IRQn);
}
/*-----*/
//          //-----@HANDLER    ADC_IRQHandler()          //
//          //-----@brief Esta función gestiona la interrupción del ADC.          //
//          //-----**/                      //
void ADC_IRQHandler()
{
    switch( YaPuedesMedir )

```

```

{
    case 1:
        LPC_ADC->ADCR      &= ~BRUST_PIN;                                // Mata el BURST.
        BUFFER_BRILLO       = (float)((LPC_ADC->ADDR1&(0xFFFF)) >> 4); // Empieza a partir del
        bit 4.                                          
        BUFFER_BRILLO       /= (float)0xFF; // *rel                                         // Relación de código.
        (Código/Código máximo)
        BUFFER_BRILLO       = RESISTENCIA_PULL*(BUFFER_BRILLO)/(1.00 - BUFFER_BRILLO); // Leo el ADC.
        (Resistencia del LDR en kOhms)
        goto_LUT( BUFFER_BRILLO , BRILLO_LDR_NOLUT, (float *)&DATOS->Brillo , none , none , none ); // Traduzco resistencias a LUX.

    Traduzco resistencias a LUX.
    BUFFER_UVA           = (float)((LPC_ADC->ADDR2&(0xFFFF)) >> 4); // Empieza a partir del bit 4.
    DATOS->IndiceUV     = (float)VINDICE*VREF*BUFFER_UVA/(float)(0xFF); // Traducción del código

    al índice.
    ACTUALIZADOR->LDRrev = 1;                                         // Digo que el LDR ha sido leido.
    //ACTUALIZADOR->LDR      = 1;                                         // Señal al LCD para que
muestre por pantalla.
    break;
}

case 0:
    AUDIO[contador] = (uint8_t)((0xFF) & LPC_ADC->ADDR0 >> (4+4)); // El ADC es de 12 bits y las muestras de 8 bits, por
lo que hay que reducir los 4 LSB.
    if(contador++ >= MUESTRAS_AUDIO - 1)
    {
        contador      = 0; // Reseteo el contador.
        LPC_TIM1->MCR   = 0; // No interrumpe el MRO.
        ACTUALIZADOR->Audiorev = 1; // Señalo el fin del audio.
        recuperaContexto(); // Recupero el contexto del ADC.
    }
    break;
}
/**-----// // -----
// // -----
// @end ENDFILE. // // -----
// -----**/ // // -----

```

uFono.h

```
/*-----//  
// @filename uFono.h //  
// @version 0.00 //  
// @author Alberto Palomo Alonso //  
// -----//
```

```

//      @brief   Cabecera para configurar el audio del micrófono.          //
//      @category Opcional.          //
//      @map      @include          //
//                  @funcdef          //
//                  @end              //
//          //
//          //-----//
//          //
//      @include   Estos son los archivos utilizados para el audio del micrófono.  //
//          //
//-----*/ //
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
#ifndef LPC17XX
#define LPC17XX
#include "LPC17XX.H"
#endif
#ifndef TIMERS
#define TIMERS
#include "Timers.h"
#endif
/**-----//
//          //
//      @private   Estos son los símbolos correspondientes al audio del micrófono.  //
//          //
//-----*/ //
#define FUNC_ADC      0x1
#define PIN_UFONO     (23-16)
#define CANAL_ADC_UF  0x1
/**-----//
//          //
//      @funcdef   Estas son las funciones correspondientes al audio del micrófono.  //
//          //
//-----*/ //
void __configuraUFONO__ ( void ); // Configuramos el micrófono.
void lanzaUFONO   ( void ); // Lanzamos la carga de audio.
void recuperaContexto ( void ); // Recupera el ADC para lanzar las muestras.
/**-----//
//          //
//      @end     ENDFILE.          //
//          //
//-----*/ //

```

uFono.c

```

/**-----//
//      @filename  uFono.c          //
//      @version    1.00            //
//          //

```

```

// @author      Alberto Palomo Alonso
// @brief      Código que contiene todo lo relacionado con el micrófono.
// @category    Opcional.
// @map        @include
//             @variables
//             @funcion
//             @end
// -----
// @include     Estos son los archivos utilizados para el audio del micrófono.
// -----
#ifndef UFONO
#define UFONO
#include "uFono.h"
#endif
/**-
// -----
// @variables   Variables del fichero.
// -----
// */
extern uint8_t Timer2_MODO;
uint8_t tYaPuedesMedir = 1;
uint32_t ADC_ConfigBuffer;
uint32_t ADC_IntentBuffer;
/**-
// -----
// @funcion __configuraUFONO__()
// @ref      Configura todo lo necesario para la lectura de audio.
// @WARNING Utiliza variables de bloqueo.
// -----
// */
void __configuraUFONO__()
{
// LPC_PINCON->PINSEL1 |= ~(0x3 << (2*PIN_UFONO));
LPC_PINCON->PINSEL1 |= (FUNC_ADC << (2*PIN_UFONO));

// NEW:
LPC_TIM1->MRO  =  Fclk*TsAudio - 1; // Cada MRO se genera una interrupción de leer el audio.
LPC_TIM1->TCR  =  0x2;           // Reset al contador.
LPC_TIM1->TCR  =  0x1;           // Activo contador.
LPC_TIM1->MCR  =  0x0;           // MRO que NO genera la interrupción.
NVIC_EnableIRQ( TIMER1_IRQn ); // Activo interrupción.
}

/**-
// -----
// @funcion lanzaUFONO()
// @ref      Lanza la lectura de audio.
// -----
// */
void lanzaUFONO()
{
// Prepara el contexto.
YaPuedesMedir = 0;           // Bloqueo el ADC para el audio.
}

```

```

LPC_GPIO3->FIOSET = ( 1 << LECTURA_AUDIO); // Señalizo lectura de audio.
Timer2_MODO = MODO_ENTRADA; // Indico que el audio está siendo grabado.
ADC_ConfigBuffer = LPC_ADC->ADCR; // Guardo el contexto de la configuración.
ADC_IntenBuffer = LPC_ADC->ADINTEN; // Guardo el contexto de la configuración de interrupciones.

// Configurar ADC.
LPC_ADC->ADINTEN = 1; // No quiero interrupciones por conversión excepto en AD0.0.
LPC_ADC->ADCR &= ~0xFF; // Borro el sel entero, sólo voy a usar un canal.
LPC_ADC->ADCR |= CANAL_ADC_UF; // Canal para el audio.
//
LPC_ADC->ADCR &= ~(0xFF << 8); // Borro el clkdiv.
LPC_ADC->ADCR |= (0x1 << 8); // CLKDiv a 1.

// Empiezo con la conversión.
LPC_ADC->ADCR &= ~BRUST_PIN; // QUITO EL MODO BURST. // Reanimo el timer.
LPC_ADC->ADCR &= ~(0x7 << 24); // Configuro el start.
LPC_ADC->ADCR |= ADC_START; // Configuro el start.

// Activo el timer.
LPC_TIM1->MCR = 0x2; // Reset on match.
LPC_TIM1->TCR = 0x2; // Que resetee.
LPC_TIM1->TCR = 0x1; // Que cuente.
LPC_TIM1->EMR = 0x31; // Activo el Match0 en modo toogle.

}

/*
 */
// @funcion recuperaContexto() // 
// @ref Desbloquea los recursos utilizados para la lectura de audio. //
// */

void recuperaContexto()
{
    // Recupero el contexto.
    if( Timer2_MODO == MODO_ENTRADA) // Si toca recuperar...
    {
        LPC_ADC->ADCR = ADC_ConfigBuffer; // Cargo el contexto de la configuración.
        LPC_ADC->ADINTEN = ADC_IntenBuffer; // Cargo el contexto de la configuración de interrupciones.
        LPC_ADC->ADCR &= ~(0x7 << 24); // Borro el START del ADC.
        LPC_ADC->ADCR |= (0xFF << 8); // CLKDIV max.
        YaPuedesMedir = 1; // Desbloqueo el ADC.
        LPC_GPIO3->FIOCLR = ( 1 << LECTURA_AUDIO); // Señalizo fin de lectura.
        Timer2_MODO = MODO_SALIDA; // Default modo salida.
    }
}

/*
*/
// @end ENDFILE. //
// */
// */

```

```

//  @filename  UVA30A.h                                //
//  @version   0.00                                     //
//  @author    Alberto Palomo Alonso                  //
//                                                       //
//  @brief     Cabecera para el código de UVA30A.c   //
//                                                       //
//  @category  Periférico.                            //
//                                                       //
//  @map      @include                                //
//            @private                                 //
//            @functype                                //
//            @end                                    //
//                                                       //
//                                                       //
//-----//                                           //
//                                                       //
//-----//                                           //
//  @include   Estos son los archivos utilizados con el código de configuración.  //
//-----//                                           //
//-----*/                                         //
#ifndef LPC17XX
#define  LPC17XX
#include "LPC17XX.H"
#endif
#ifndef LDR
#define  LDR
#include "LDR.h"
#endif
/**-----//                                           //
//                                                       //
//  @private   Estos son los símbolos correspondientes a la configuración.  //
//-----//                                           //
//-----*/                                         //
#define  LDR_primer 1
#define  PINSEL_ADC02  (1 << 18)
#define  PINMODE_ADC02 (3 << 18)
#define  SEL_CANAL2  (1 << 2)
/**-----//                                           //
//                                                       //
//  @funcdef   Estas son las funciones correspondientes a la configuración.  //
//-----//                                           //
//-----*/                                         //
void  __configuraUVA30A__( void );
/**-----//                                           //
//                                                       //
//  @end      ENDFILE.                               //
//-----//                                           //
//-----*/                                         //

```

UVA30A.c

```

/**-----//                                           //
//  @filename  UVA30A.c                                //
//  @version   0.00                                     //
//  @author    Alberto Palomo Alonso                  //
//                                                       //

```

```
//          @brief Código fuente que contiene las funciones para UVA30A (ADC).
//          @category Periférico.
//          @map      @include
//                      @funcion
//                      @end
//
//          @include   Estos son los archivos utilizados en el código de configuración del UVA.
//          //----- */
#ifndef UVA30A
#define UVA30A
#include "UVA30A.h"
#endif
/**-----
//          @funcion __configuraUVA30A__()
//          @ref      Configura todo lo necesario para que el UVA30A lea el índice UV.
//          //----- */
void __configuraUVA30A__()
{
    if( !LDR_primer )
    {
        __configuraLDR__();
    }
    LPC_PINCON->PINSEL1 |= PINSEL_ADC02;      // AD0.2

    LPC_PINCON->PINMODE1  &= ~PINMODE_ADC02; // AD0.2
    LPC_ADC->ADCR      |= SEL_CANAL2;      // AD0.2
}
/**-----
//          @end      ENDFILE.
//          //----- */
//----- */
```

OneWire.h

```
/**-----//
```

```

//  @filename  OneWire.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Cabecera para configurar el protocolo OneWire.
//
//  @category  Opcional.
//
//  @map      @include
//            @funcdef
//            @end
//
//
//-----//
//
//  @include  Estos son los archivos utilizados para el protocolo OneWire.
//
//-----**/
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
/**-
//-----*/
//
//  @private   Estos son los símbolos correspondientes al protocolo OneWire.
//
//-----*/
#define SIZEOF_TRAMA     40 // Tamaño del array como buffer de 8 bits.
#define SIZEOF_SLOT       8  // Los últimos 8 bits son el checksum.

#define OW_RESET_BAJO    0.018 // Tiempo a nivel bajo de la señal de start. (MCU -> Sensor)
                           /** @CHANGE: 0.04 */
#define OW_RESET_ALTO    0.02  // Tiempo a nivel alto de la señal de start. (MCU -> Sensor)
#define OW_RESPUESTA_BAJO 0.08 // Señal respuesta del sensor a nivel bajo.
#define OW_RESPUESTA_ALTO 0.08 // Señal respuesta del sensor a nivel alto.
                           /** @CHANGE: 0.078 */
#define OW_MANDADO0      0.076 // Duración del tiempo de un bit = 0.
#define OW_MANDADO1      0.120 // Duración del tiempo de un bit = 1.

#define PERMISO_DIFERENCIAL -0.005 // Error que le permito tener al sensor.
#define PERMISO_PROPORCIONAL  0.005 // Error que le permito tener al sensor.

#define OW_PULL_UP        0x0 // Función de pull up.
#define OW_PULL_DOWN       0x3 // Función de pull down.
#define OW_CAPTURE_FUNC    0x3 // Función capture.
#define OW_CTCR_MASCARA   0x0 // Dejar TC contando.
#define OW_CCR_MASCARA_EN 0x30 // Activo por flanco de bajada.
#define OW_CCR_MASCARA_DIS 0x20 // Desactivo por flanco.
#define CAP11_IR          0x00000020//El IR de CAP1.1
#define MRO_IR             0x00000001//El IR de MRO.

#define OWDEEPSLEEP       0
#define OWINICIO          1
#define OWESPERANDO       2
#define OWESPERANDO_BIT   3
#define OWCHECKSUM         4
#define OWGENERA           5
#define OWESPERANDO_SEQ   6

#define LIMITE_FALLOS     5
#define BITOW              19

#define US_AHORA          (LPC_TIM3->TC)
#define PIN_OWP            19

```

```
#define PIN_OW      (1 << 19)
#define CONFIG_OUT   (LPC_GPIO1->FIODIR |= PIN_OW)
#define CONFIG_IN    (LPC_GPIO1->FIODIR &= ~(PIN_OW))
#define CLEAR_PIN    (LPC_GPIO1->FIOCLR = PIN_OW)
#define SET_PIN      (LPC_GPIO1->FIOSET = PIN_OW)
#define ENTRADA      (((LPC_GPIO1->FIOPIN >> PIN_OWp) & 1)
/*-----*/
//                                     //           //
//      @funcdef  Estas son las funciones correspondientes al protocolo OneWire.          //
//                                     //           //           //
//-----*/ **
void __configuraOW_( void ); // Configuración del protocolo OneWire.
void mideTemperatura( void ); // Código de medición de temperatura.
void activaMedidaOW ( void ); // Lanza el activador del one wire.

void StateChartOneWire ( uint32_t DeltaCap );
void OWSetPin      ( uint8_t tNivel );
void OWConfiguraEntrada ( void )     ;
void OWConfiguraSalida ( void )     ;
void ErrorRx       ( void )     ;
void ErrorTx       ( void )     ;
void InvalidChecksum( void )     ;
/*-----*/
//                                     //           //
//      @end    ENDFILE.               //           //           //
//-----*/ **/
```

OneWirev2.c

```
/*-----*/
//      @filename  OneWire.c
//-----*/ //
```

```

// @version 4.01
// @author Alberto Palomo Alonso
// @brief Código que configura el protocolo monohilo del sensor de temperatura y humedad.
// @category Opcional.
// @map @include
// @variables
// @funcion
// @end
// -----
// @include Estos son los archivos utilizados para el protocolo OneWire.
// -----
#ifndef ONEWIRE
#define ONEWIRE
#include "OneWire.h"
#endif
/**-
// -----
// @variables Variables del fichero.
// -----
*/
uint32_t reiniciaCuenta    ( void );
void   inicializaT3         ( void );
void   _delayUS             ( uint16_t us );
uint8_t  comprouebaRespuesta( void );
uint8_t _tleerByte          ( void );
// Externo:
extern misDatos_t      * DATOS;
uint32_t  TRAZA [100];
uint32_t  HOLD   [100];
int p;
uint8_t  Checksum;
/**-
// -----
// @function __configuraOW_()
// @brief Configura los pines y los recursos utilizados para el protocolo OneWire.
// -----
*/
void __configuraOW_()
{
    inicializaT3();
    reiniciaCuenta();
}
/**-
// -----
// @function mideTemperatura()
// @brief Configura los pines y los recursos utilizados para el protocolo OneWire.
// -----
*/
void mideTemperatura( void )
{
    int i;
    uint8_t Check[4] = {0,0,0,0};
    uint32_t Rx        = 0;
}

```

```

uint8_t Checksum_Recibido = 0;
p = 0;
Checksum = 0;
/** @state: Estado en el que mandamos la señal de petición. */
CONFIG_OUT;
CLEAR_PIN;
_delayUS(18000);
CONFIG_IN;
/** @state: Esperamos la respuesta. */
if( compruebaRespuesta() )
{
    // ERROR A AL ESPERAR LA RESPUESTA DEL SENSOR...
    return;
}
/** @state: Leemos los 5 bytes... */
for(i = 0; i < 4; i++)
{
    Rx |= (leerByte() << (3-i)*8);
}
Checksum_Recibido = leerByte();
/** @state: Procesamos Rx. */
Check[0] = ((Rx & (0xFF000000)) >> 6*4);
Check[1] = ((Rx & (0x00FF0000)) >> 4*4);
Check[2] = ((Rx & (0x0000FF00)) >> 2*4);
Check[3] = ((Rx & (0x000000FF)) >> 0*4);
Checksum = Check[0] + Check[1] + Check[2] + Check[3];

if( Checksum == Checksum_Recibido )
{
    DATOS->Humedad = (float)((Rx >> 16) & 0xFFFF)/10.0;
    DATOS->Temperatura = (float)((Rx >> 00) & 0xFFFF)/10.0;
}
/** -----
//                                     //                                     //
//      @function   reiniciaCuenta()                                //      //
//      @brief     Reinicia el contador del timer 3.                //      //
//      @ret       Retorna el valor de la cuenta antes de reiniciarla.  //
//----- */                                                       //

uint32_t reiniciaCuenta()
{
    uint32_t retval = US_AHORA;
    LPC_TIM3->TCR = 2; // Reinicia timer.
    LPC_TIM3->TCR = 1; // Activa cuentas.
    return retval;
}
/** -----
//                                     //                                     //
//      @function   inicializaT3()                                //      //
//      @brief     Configura el timer 3 para utilizarlo para el OneWire.  //
//----- */                                                       //

void inicializaT3()
{
    LPC_SC->PCONP |= (1 << 23); // Activo el timer 3.
    LPC_TIM3->CTCR = 0;           // Contar por prescaler.
    LPC_TIM3->PR= 24;            // Cuentas cada 1us.
}
/** -----
//----- */

```

```

//                                     //
//      @function    _delayUS()          //           //
//      @brief      Espera activa de [usegundos] microsegundos.      //
//      @param      us segundos           //           //
//-----*/                                //
void  _delayUS( uint16_t  us segundos)
{
    reiniciaCuenta();
    while (US_AHORA < us segundos) {}
}
/**-----*/
//                                     //
//      @function    compruebaRespuesta()        //
//      @brief      Comprueba si el sensor ha respondido apropiadamente.  //
//      @ret       Retorna 0 si todo ha ido bien, 1 si no.                //
//-----*/                                //
uint8_tcompruebaRespuesta()
{
    uint32_t  Tiempo = 0;
    /**@state:   Esperamos que el sensor responda con un pull down... */
    reiniciaCuenta();
    while ( ENTRADA && US_AHORA < 45)           // Si la entrada está a nivel alto y no han pasado 45 us...
    {
        // Mantente en espera.
    }
    Tiempo = reiniciaCuenta();                   // Me quedo con cuentos us han pasado.
    TRAZA[p++] = Tiempo; //!!!!!
    if ( Tiempo < 5 || Tiempo > 45)             // Si el margen de pull down del sensor no es el adecuado.
    {
        return 1;                                // Exit error code.
    }
    /**@state:   Esperamos la respuesta del sensor... */
    reiniciaCuenta();
    while ( ENTRADA == 0 && US_AHORA < 100)// Tiempo nivel bajo...
    {
        // 
    }
    Tiempo = reiniciaCuenta();
    TRAZA[p++] = Tiempo; //!!!!!
    if ( Tiempo < 70 || Tiempo > 90)           // Si el tiempo de pull down no es adecuado...
    {
        return 1;                                // Exit error code.
    }
    reiniciaCuenta();
    while ( ENTRADA     && US_AHORA < 100)// Tiempo nivel alto...
    {
        // 
    }
    Tiempo = reiniciaCuenta();
    TRAZA[p++] = Tiempo; //!!!!!
    if ( Tiempo < 70 || Tiempo > 90)           // Si el tiempo de pull down no es adecuado...
    {
        return 1;                                // Exit error code.
    }
    return 0;                                  // Respuesta correcta.
}
/**-----*/
//                                     //
//      @function    leerByte()           //

```

```

//          //          //
//      @brief   Lee 8 bits en ráfaga del sensor.          //          //
//          //          //          **/
//-----**//-----**//-----**//-----**//-----**//
uint8_t leerByte( void )          {
{
    int i;
    uint8_t Rx=0;
    uint32_t Tiempo = 0;
    for (i = 0; i < 8; i++)
    {
        reiniciaCuenta();
        while ( ENTRADA == 0 && US_AHORA < 100) // Mientras la entrada valga 0...
        {
            // Mantenernos esperando.
        }
        Tiempo = reiniciaCuenta();
        TRAZA[p++] = Tiempo; //!!!!!
        if (Tiempo < 40 || Tiempo > 67)           // Si el tiempo está fuera del margen.
        {
            return 0;                            // Error al leer el bit, retorna 0.
        }
        Tiempo = 0;
        reiniciaCuenta();
        while ( ENTRADA && US_AHORA < 100) // Mientras la entrada valga 1...
        {
            // Mantenernos esperando.
        }
        Tiempo = reiniciaCuenta();
        TRAZA[p++] = Tiempo; //!!!!!
        if ( Tiempo > 60 && Tiempo < 80)           // Si entra en el margen del 1...
        {
            Rx |= 1 << (7-i);                  // Añadimos un 1.
        }
        else
        {
            if ( Tiempo < 10 || Tiempo > 100)     // Si se ha salido del margen...
            {
                return 0;                          // Error al leer el bit, retorna 0.
            }
        }
    }
    return Rx;
}
//-----**//-----**//-----**//-----**//-----**//
//      @end      ENDFILE.          //          //
//-----**//-----**//-----**//-----**//-----**//

```

```
/*
//  @filename  PWM.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief    Este es el programa donde están definidas las funciones a utilizar en el módulo
//            PWM dedicado al proyecto de Sistemas electrónicos digitales avanzados (UAH - EPS).
//
//  @category Opcional.
//
//  @map      @include
//            @private
//            @funcdef
//            @end
//
//  |---| | | | \ / / |
//  | | | | | | | \ / / |
//  |---| | | | | | | V | |
//  | | | | | | | | | |
//  | | | | | | | | |
//
//  @include  Estos son los archivos utilizados con el código PWM.
//
//-----*/
#ifndef LPC17XX
#define LPC17XX
#include "LPC17XX.H"
#endif

#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif

/*
//  @private  Estos son los símbolos correspondientes al módulo PWM.
//
//-----*/
#define F pwm 50 // Velocidad de 20Hz.
#define T pwm 0.02 // Período de 20ms.

#define TCR_MASK 0x9 // Activo el contador y el PWM.
#define MCR_MASK 0x4 // Para el set del PWM.
#define MCR_MASK_RESET 0x5 // Para resetear el MCR.
#define PCONP_MASK 0x1 << 6 // Para encender el PWM.

#define MODO_CICLO 1 // Para la función modificaPulso().
#define MODO_SERVO 0 // Para la función modificaPulso().

#define PWM1 1
#define PWM2 2
#define PWM3 3
#define PWM4 4
#define PWM5 5
#define PWM6 6

#define OPEN_DRAIN 0x2
#define PULL_UP 0x0
#define PULL_DOWN 0x3
#define UNUSED 0x1
```

```
#define KMX          1.3      // Constante de corrección máxima.
#define KMN          0.6      // Constante de corrección mínima.
#define MINIMO_SERVO 0.001
#define MAXIMO_SERVO 0.002

#define TEMP_MAX       32      // Visionado de temperatura.
#define TEMP_MIN       17      // Visionado de temperatura.

/**-----*/
//                                         //
// @funcdef   Estas son las funciones correspondientes al módulo PWM.           //
//                                         //
//-----*/
void __configuraPWM__ ( float FrecuenciaPWM , uint16_t CualesPWM ); // Default = 0x3F3F -> Todo activado.
void modificaPulso ( uint32_t PWMn , uint8_t Modo , uint8_t Ciclo , uint8_t Grados , float
Minimo , float Maximo );
void softMod ( uint8_t GradosObjetivo , uint8_t GradosActuales, float Minimo , float Maximo , uint32_t
PWMn );
/**-----*/
//                                         //
// @end     ENDFILE.                   //
//                                         //
//-----*/
```

PWM.c

```
/**-
//  @filename  PWM.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief    Este es el programa donde están definidas las funciones a utilizar en el módulo
//            PWM dedicado al proyecto de Sistemas electrónicos digitales avanzados (UAH - EPS).
//
//  @category Opcional.
//
//  @map      @include
//  @variables
//  @function
//  @end
//
//  |---| | | | \ / |
//  | | | | | | \ / |
//  |---| | | | | | V | |
//  | | | | | | | | |
//  | | | | | | | |
//
//  @include   Includes pertenecientes al módulo del PWM.
//
//-----*/
#ifndef PWM
#define PWM
#include "PWM.h"
#endif
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
/**-
//  @variables   Variables del fichero.
//
//-----*/
extern Counters_t * COUNTERS;
/**-
//  @function   __configuraPWM_()
//
//  @brief     Configura el PWM en función de la frecuencia a utilizar y que PWM se quieren usar.
//             El primer byte activa las del puerto 2 y las del segundo las del otro puerto en orden
//             ascendente.
//
//  @FrecuenciaPWM   Frecuencia a la que se desea ajustar el ciclo PWM. En Hz.
//  @CualesPWM     Máscara que define los PWM a configurar, los 6 primeros bits corresponden al
//                 puerto 2. Del 9º bit al 14º bit corresponden al puerto 1. En orden ascendente
//                 desde del PWM1.1 al PWM1.6
//
//-----*/
void __configuraPWM_(float FrecuenciaPWM , uint16_t CualesPWM )
{
    LPC_SC->PCONP |= PCONP_MASK; // Enciendo el PWM.
}
```

```

LPC_PWM1->MRO      = ((uint32_t)(Ftick/FrecuenciaPWM) - 1);           // Configuro la frecuencia.
LPC_PWM1->TCR      |= TCR_MASK;                                         // Activo el PWM.
LPC_PWM1->MCR      &= ~TODO_1_32;                                         // Reset al MCR.
LPC_PWM1->MCR      |= 0x2;                                              // Ahora el MRO resetea el contador.
LPC_PINCON->PINSEL4 &= ~0xFF;                                            // Reset en pines PWM puerto 2.
LPC_PINCON->PINSEL3 &= ~0x33CF30;                                         // Reset en pines PWM puerto 1.

for (COUNTERS->i = 0; COUNTERS->i < 6; (COUNTERS->i)++)           // Para el puerto 2: seleccionados.
{
    if ( (CualesPWM >> COUNTERS->i) & ~0xFFFF)                         // Miro si está seleccionado el iésimo.
    {
        LPC_PINCON->PINSEL4 |= (FUNC1    << (2*COUNTERS->i)); // Pongo la función 1 en los pines PWM.
        LPC_PWM1->PCR |= (0x1 << (COUNTERS->i + 0x9));          // Pongo la función de enable output en el PWM.
    }
}
/** @REMARK: Esto se configuraría como open drain sobre todo por no perder potencia, pero prefiero asegurarme con pull.*/
for (COUNTERS->i = 6; COUNTERS->i < 12; (COUNTERS->i)++)           // Para el otro puerto: seleccionados.
{
    if ( (CualesPWM >> (COUNTERS->i + 2)) & ~0xFFFF)                         // Miro si está seleccionado el iésimo.
    {
        LPC_PWM1->PCR |= (0x1 << (COUNTERS->i - 0x6 + 0x9)); // Pongo la función de enable output en el PWM.
        switch (COUNTERS->i)                                         // Pongo la función 2 en los pines PWM.
        {
            case 6:
                LPC_PINCON->PINSEL3 |= FUNC2 << 2*2;
                LPC_PINCON->PINMODE3 |= OPEN_DRAIN << 2*2;
                break;
            case 7:
                LPC_PINCON->PINSEL3 |= FUNC2 << 2*4;
                LPC_PINCON->PINMODE3 |= OPEN_DRAIN << 2*4;
                break;
            case 8:
                LPC_PINCON->PINSEL3 |= FUNC2 << 2*5;
                LPC_PINCON->PINMODE3 |= OPEN_DRAIN << 2*5;
                break;
            case 9:
                LPC_PINCON->PINSEL3 |= FUNC2 << 2*7;
                LPC_PINCON->PINMODE3 |= OPEN_DRAIN << 2*7;
                break;
            case 10:
                LPC_PINCON->PINSEL3 |= FUNC2 << 2*8;
                LPC_PINCON->PINMODE3 |= OPEN_DRAIN << 2*8;
                break;
            case 11:
                LPC_PINCON->PINSEL3 |= FUNC2 << 2*10;
                LPC_PINCON->PINMODE3 |= OPEN_DRAIN << 2*8;
                break;
            default:
                break;
        }
    }
}
COUNTERS->i = 0;                                                       // Dejo el contador a 0.
}

// -----
// @function modificaPulso()                                         //
// @brief Configura el pulso de PWM en función del ciclo de trabajo o de valores de oscilación ////
// si se encuentra en modo servo.                                         //
// @PWMn Selecciona el PWM1.n a modificar.                           //
// @Modo Selecciona si modo servo (valores) o modo ciclo (en porcentaje). ////
// @Ciclo Selecciona el ciclo de trabajo para el modo ciclo.          //
// @Grados Selecciona los grados a rotar el servo en modo servo.       //

```

```

//          @Minimo Selecciona el mínimo valor de Ton del ciclo PWM. En segundos.          //
//          @Maximo Selecciona el máximo valor de Ton del ciclo PWM. En segundos.          //
//          //          //          //          //          //          //          //
//          //          //          //          //          //          //          //
//          //          //          //          //          //          //          //
//          //----- */
void  modificaPulso(  uint32_t  PWMn,   uint8_t Modo ,   uint8_t Ciclo ,   uint8_t Grados,   float Minimo ,   float
                      Maximo  )
{
    if (PWMn > 3)
    {
        PWMn += 6; // Debido a la asimétrica distribución de los registros.
    }

    Minimo *= KMN;      // Definitivamente había algo mal.
    Maximo *= KMX;     // Estos servos utilizan la potencia del pulso y no precisamente su duración.

/**@REMARK: La potencia entregada no es la debida. En la datasheet especifica pulsos del rango de 5V, se ofrece uno
de 3.3V, se ha podido usar un transistor, pero deduzco que estos servos utilizan la potencia de la señal
PWM para obtener el ángulo y modificando los tiempos podemos entregar más potencia de señal.*/

switch( Modo )
{
    case MODO_CICLO: //Escribo en LPC_PWM1->MRn el valor correspondiente al porcentaje de MRO; < 1.
        *(_IO uint32_t *)((uint32_t)&(LPC_PWM1->MRO) + (uint32_t)(0x4*PWMn)) =
        (uint32_t)((float)(LPC_PWM1->MRO)*((float)Ciclo/(float)100));
        break;
    case MODO_SERVO: //Escribo en LPC_PWM1->MRn el valor correspondiente al tiempo Ton en función del grado.
        *(_IO uint32_t *)((uint32_t)&(LPC_PWM1->MRO) + (uint32_t)(0x4*PWMn)) = (uint32_t)((Ftick*(Minimo +
        Maximo)*(float)(Grados/(float)(180)))-(float)1);
        break;
    default:
        break;
}
    if (PWMn > 3)
    {
        PWMn -= 6; // Devolvemos PWMn a su estado orígen.
    }
    LPC_PWM1->LER |= 0x1 << PWMn | 0x1; //Activo el load de los MRO y MRn.
}
/**-----*/
//          //          //          //          //          //
//      @end      ENDFILE.          //          //
//----- */

```

DAC.h

DAC.c

```
/*
//  @filename  DAC.c
//  @version   6.01
//  @author    Alberto Palomo Alonso
//
//  @brief    Código fuente que contiene las funciones para audio (DAC).
//
//  @category Periférico.
//
//  @map      @include
//            @variables
//            @funcion
//            @end
//
//  @include  Estos son los archivos utilizados en el código de LDR.
//
//-----*/
#ifndef DAC
#define  DAC
#include "DAC.h"
#endif
/*
//  @variables  Variables del fichero.
//
//-----*/
extern actualizador_t * ACTUALIZADOR;
/*
//  @funcion   __configuraDAC_()
//  @brief     Función de configuración del DAC y su timer (Timer2).
//  @REMARK:   Para activar un periodo del DAC (2 segundos)
//
//-----*/
void __configuraDAC_()
{
    LPC_GPIO3->FIODIR |= (1 << LECTURA_AUDIO ) | (1 << ESCRITURA_AUDIO); //Leds de lectura / escritura de audio.
    LPC_GPIO3->FIOCLR = (1 << LECTURA_AUDIO); //Turn ON LED1
    LPC_GPIO3->FIOCLR = (1 << ESCRITURA_AUDIO); //Turn ON LED2
}
/*
//  @funcion   activarDac()
//  @brief     Señal de activar el timer del DAC.
//  @REMARK:   Utiliza DMA.
//
//-----*/
void activarDac()
{
    /*@TODO: DMA*/
    LPC_GPDMA0->DMACConfig |= 1; // Activo el DMA.
}
```

```

LPC_TIM1->MCR      = (1 << 3) | (1 << 4); // Activo la interrupción por MR1 y reset por MR1.
LPC_TIM1->MR1      = (Fcclk*DURACION_AUDIO) - 1; // Valor de MR1.
LPC_TIM1->TCR      = 0x2; // Reset del timer.
LPC_TIM1->TCR      = 0x1; // El timer cuenta.
ACTUALIZADOR->Audiorev = 0; // Señalizo el bloqueo de audio.
LPC_GPIO3->FIOSET   = (1 << ESCRITURA_AUDIO); // Señalo escritura de audio.
}

/**-----// // -----
// @funcion    activarDac() // // -----
// @brief      Señal de activar el timer del DAC. // // -----
// @REMARK:    Activador del DAC. // // -----
// -----**/ // // -----
void desactivarDAC()
{
    LPC_GPDMA0->DMACConfig &= ~0x1; // Desactivo el DMA.
    ACTUALIZADOR->Audiorev = 1; // Señalizo el fin del DAC.
    LPC_GPIO3->FIOCLR = (1 << ESCRITURA_AUDIO); // Señalo escritura de audio.
    LPC_TIM1->MCR     &= ~(7 << 3); // Desactivo la interrupción por MR1 y reset tras MR1.
    LPC_DAC->DACR     = 0; // No hay señal de salida.
}
/**-----// // -----
// @end      ENDFILE. // // -----
// -----**/ // // -----

```

LUT.h

```
/*
//  @filename  LUT.g
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Este es el programa donde se encuentra la cabecera de LUT.c
//
//  @category  Opcional.
//
//  @map      @include
//            @private
//            @funcdef
//            @end
//
//  @include   Includes pertenecientes al módulo del anemómetro.
//
//  @endfile
*/
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
#ifndef LDR
#define LDR
#include "LDR.h"
#endif
#ifndef DMA
#define DMA
#include "DMA.h"
#endif
#ifndef MATH
#define MATH
#include <math.h>
#endif

#define MUESTRAS_SENO 32
#define BRILLO_LDR 0
#define BRILLO2CICLO_LDR 1
#define INDICE_UVA 2
#define BRILLO_LDR_NOLUT 3

void goto_LUT( float dato , uint8_t LUTn , float * ret_f , uint8_t * ret_8 , uint16_t * ret_16 , uint32_t * ret_32 );
void crearSeno( void );
void ponTonoDMA( void );
/*
//
//  @end  ENDFILE.
//
*/
#ifndef
//  @filename  LUT.g
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Este es el programa donde se encuentra la cabecera de LUT.c
//
//  @category  Opcional.
//
//  @map      @include
//            @private
//            @funcdef
//            @end
//  @include   Includes pertenecientes al módulo del anemómetro.
//
//  @endfile
*/
```

```
//          @end          //  
//          //  
//-----  
//          //          //  
//      @include    Includes pertenecientes al módulo del anemómetro.  
//          //          //  
//-----*/  
#ifndef SYSTEMSYMBOLS  
#define SYSTEMSYMBOLS  
#include "Systemsymbols.h"  
#endif  
#ifndef LDR  
#define LDR  
#include "LDR.h"  
#endif  
#ifndef DMA  
#define DMA  
#include "DMA.h"  
#endif  
#ifndef MATH  
#define MATH  
#include <math.h>  
#endif  
  
#define MUESTRAS_SENO 32  
#define BRILLO_LDR 0  
#define BRILLO2CICLO_LDR 1  
#define INDICE_UVA 2  
#define BRILLO_LDR_NOLUT 3  
  
void goto_LUT( float dato , uint8_t LUTn , float * ret_f , uint8_t * ret_8 , uint16_t * ret_16 , uint32_t * ret_32 );  
void crearSeno( void );  
void ponTonoDMA( void );  
/*-----//  
//          //          //  
//      @end    ENDFILE.  
//          //          //  
//-----*/
```

LUT.c

```
/*
//  @filename  LUT.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Este es el programa donde se encuentran las look up tables que optimizan
//             el uso de cpu sacrificando memoria, pero como uso una SD la memoria no es
//             un problema grande.
//
//  @category  Opcional.
//
//  @map      @include
//            @variables
//            @LUT
//            @function
//            @end
//
//  @include   Includes pertenecientes al módulo del anemómetro.
//
//-----*/
//  @include   Includes pertenecientes al módulo del anemómetro.
//-----*/
//  @variables Variables del fichero.
//-----*/
//  @LUT      LookUpTables
//  @brief    Bases de datos.
//-----*/
uint8_t PREGRABADA[MUESTRAS_SENO];
extern uint8_t* AUDIO;
//-----*/
//  @function goto_LUT()
//  @brief   Esta función es la que mira las LUT y obtiene el dato que queremos de la base
//          de datos.
//-----*/
void goto_LUT( float variable , uint8_t LUTn , float * ret_flotante , uint8_t * ret_int8 , uint16_t * ret_int16 , uint32_t * ret_int32)
```

```
{  
    switch(LUTn)  
    {  
        case BRILLO_LDR:  
            *ret_flotante = Brillo_LDR[      (uint8_t)((variable - LDRRESISTENCIA_MIN)/(LDRRESISTENCIA_MAX -  
LDRRESISTENCIA_MIN)) *Brillo_LDR[0]] + 1;  
            break;  
        case BRILLO2CICLO_LDR:  
            *ret_int8 = Brillo2ciclo_LDR[      (uint8_t)((variable - BRILLO_MIN)      /(BRILLO_MAX - BRILLO_MIN))  
*Brillo2ciclo_LDR[0]] + 1;  
            break;  
        case INDICE_UVA:  
            *ret_flotante = variable; // El output DC corresponde al índice, es muy sencillo traducirlo, se recomienda no llamar a esta  
función en este modo.  
            break;  
        case BRILLO_LDR_NOLUT:  
            *ret_flotante = -(1.0102)*variable + 102.0204;  
            if(*ret_flotante < 0)  
            {  
                *ret_flotante = 0;  
            }  
        default:  
            break;  
    }  
}  
/*-----//  
//-----//  
//-----//  
// @end ENDFILE.  
//-----//  
//-----*/
```

DMA.h

```

/***
//  @filename  DMA.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Cabecera del configurado del DMA.
//
//  @category  Opcional.
//
//  @map      @include
//            @private
//            @funcdef
//            @end
//
//
//-----*/
//
//  @include   Estos son los archivos utilizados con el código del DMA.
//  @private   Estos son los símbolos correspondientes al DMA.
//  @funcdef  Estas son las funciones que se usan en el DMA.

#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif

#ifndef LDR
#define LDR
#include "LDR.h"
#endif

#ifndef LUT
#define LUT
#include "LUT.h"
#endif

#include <LPC17xx.H>
#include <math.h>

/*
//  @private   Estos son los símbolos correspondientes al DMA.
//  @funcdef  Estas son las funciones que se usan en el DMA.

#define N_samples_wave    32 // Nº de muestras por ciclo
#define Ftono        400

/*
//  @funcdef  Estas son las funciones que se usan en el DMA.

void __configuraDMA__ ( void ); // Configurador del DMA.
void __configuraTono__ ( void ); // Configurador del tono.
void __configuraAudio__ ( void ); // Configurador del audio.

/*
//  @end     ENDFILE.
//  @private
//-----*/

```

DMA.c

```
/*
//  @filename  DMA.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Código fuente que contiene las funciones para audio (DMA).
//
//  @category  Periférico.
//
//  @map      @include
//            @variables
//            @funcion
//            @end
//
//  @include  Estos son los archivos utilizados en el código de LDR.
//
//  @variables Variables del fichero.
//
//  @funcion  __configuraDMA_()
//  @brief    Función de configuración del DMA.
//  @REMARK: Para activar un periodo del DAC (2 segundos)
//
void __configuraDMA__(void)
{
    int i;
    for(i=0; i < N_samples_wave; i++)
    {
        sinusode[i] = (int)(127 + 127*sin(2*PI*i/N_samples_wave)); // DACR bit 6-15 VALUE (valor ya desplazado!!!)
    }
    LPC_PINCON->PINSEL1 |= (2<<20); // enable AOUT (P0.26) pin
    __configuraTono_();
}
//
//  @funcion  __configuraTono_()
//  @brief    Función de configuración del tono.
//

```

```

void __configuraTono_()
{
    // Linked CH0
    LLIO.source      = (uint32_t) &sinusoide[0];
    LLIO.destination = (uint32_t) &(LPC_DAC->DACR) + 1;
    LLIO.next        = (uint32_t) &LLIO;
    LLIO.control     = 1<<26 | 0<<21 | 0<<18 | N_samples_wave; //Transfersize= N_samples_wave, SWidth=8bits, DWidth=8bits,
    Source Increment

LPC_SC->PCONP |= (1<<29);                                // Power DMA
LPC_GPDMA->DMACConfig = 1;                                // enable the GPDMA controller
LPC_GPDMA->DMACSync   = (1<<6);                          // enable synchro logic for all reqs

LPC_GPDMA->DMACCSrcAddr = (uint32_t) &sinusoide[0];          // Power DMA
LPC_GPDMA->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR) + 1; // enable the GPDMA controller
LPC_GPDMA->DMACCLLI   = (uint32_t) &LLIO; // linked lists for ch0
LPC_GPDMA->DMACControl = N_samples_wave // transfer size (0 - 11) = 32 muestras / ciclo
    | (0 << 12)           // source burst size (12 - 14) = 1
    | (0 << 15)           // destination burst size (15 - 17) = 1
    | (0 << 18)           // source width (18 - 20) = 32 bit CAMBIADO
    | (0 << 21)           // destination width (21 - 23) = 32 bit NO CAMBIADO
    | (0 << 24)           // source AHB select (24) = AHB 0
    | (0 << 25)           // destination AHB select (25) = AHB 0
    | (1 << 26)           // source increment (26) = increment
    | (0 << 27)           // destination increment (27) = no increment
    | (0 << 28)           // mode select (28) = access in user mode
    | (0 << 29)           // (29) = access not bufferable
    | (0 << 30)           // (30) = access not cacheable
    | (0 << 31);          // terminal count interrupt disabled

LPC_GPDMA->DMACConfig = 0 // channel enabled (0)
    | (0 << 1)           // source peripheral (1 - 5) = none
    | (7 << 6)           // destination peripheral (6 - 10) = DAC
    | (1 << 11)           // flow control (11 - 13) = MEM to PERF
    | (0 << 14)           // (14) = mask out error interrupt
    | (0 << 15)           // (15) = mask out terminal count interrupt
    | (0 << 16)           // (16) = no locked transfers
    | (0 << 18);          // (27) = no HALT

//F_out (salida del DAC)
LPC_DAC->DACCNTVAL = Fclk/N_samples_wave/Ftono -1; // (Ts DAC = F_out/N_samples < Tsetup DAC = 1useg. !!!)

/* DMA, timer running, dbuff */
LPC_DAC->DACCTRL = 1<<3 | 1<<2 | 1<<1;
ACTUALIZADOR->Audiorev = 1;
}

// -----
//                                         //                                         //
// @funcion      __configuraAudio_()
//                                         //                                         //
// @brief        Función de configuración del audio.                         //
//                                         //                                         //
//                                         //                                         **/


void __configuraAudio_()
{
    // Linked CH0
    LLIO.source      = (uint32_t) AUDIO;
    LLIO.destination = (uint32_t) &(LPC_DAC->DACR) + 1;
    LLIO.next        = (uint32_t) &LLIO;
    LLIO.control     = 1<<26 | 0<<21 | 0<<18 | MUESTRAS_AUDIO; // Incremento origen, MUESTRAS_AUDIO muestras, 8 bits
    todo.

LPC_SC->PCONP |= (1<<29); // Enciendo el DMA.
LPC_GPDMA->DMACConfig = 1; // Activo el controlador del DMA.

```

```

LPC_GPDMA->DMACSync    = (1<<6);           // Sincronización de registros.

LPC_GPDMDMACH0->DMACCSrcAddr = (uint32_t) AUDIO;      // Empieza en AUDIO.
LPC_GPDMDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR) + 1; // Ve a DACR + 2.
LPC_GPDMDMACH0->DMACLLI     = (uint32_t) &LLI0; // linked lists for ch0
LPC_GPDMDMACH0->DMACCCControl = MUESTRAS_AUDIO // transfer size (0 - 11) = 32 muestras /ciclo
| (0 << 12)          // source burst size (12 - 14) = 1
| (0 << 15)          // destination burst size (15 - 17) = 1
| (0 << 18)          // source width (18 - 20) = 32 bit
| (2 << 21)          // destination width (21 - 23) = 32 bit
| (0 << 24)          // source AHB select (24) = AHB 0
| (0 << 25)          // destination AHB select (25) = AHB 0
| (1 << 26)          // source increment (26) = increment
| (0 << 27)          // destination increment (27) = no increment
| (0 << 28)          // mode select (28) = access in user mode
| (0 << 29)          // (29) = access not bufferable
| (0 << 30)          // (30) = access not cacheable
| (0 << 31);         // terminal count interrupt disabled

LPC_GPDMDMACH0->DMACCCConfig = 0           // channel enabled (0)
| (0 << 1)          // source peripheral (1 - 5) = none
| (7 << 6)          // destination peripheral (6 - 10) = DAC
| (1 << 11)          // flow control (11 - 13) = MEM to PERF
| (0 << 14)          // (14) = mask out error interrupt
| (0 << 15)          // (15) = mask out terminal count interrupt
| (0 << 16)          // (16) = no locked transfers
| (0 << 18);         // (27) = no HALT

//F_out (salida del DAC)
LPC_DAC->DACCNTVAL = (Fcclk/4000) - 1; // (Ts DAC = TsAudio < Tsetup DAC = 1useg. !!!)

/* DMA, timer running, dbuff */
LPC_DAC->DACCTRL = 1<<3 | 1<<2 | 1<<1;
ACTUALIZADOR->Audiorev = 1;
}

/**-----//  

//-----//  

//-----//  

//-----@end ENDFILE.  

//-----//  

//-----*/

```

WDT.h

WDT.c

```
/*
//  @filename  WDT.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief    Código fuente del configurado del WDT.
//
//  @category Interno.
//
//  @map      @include
//            @function
//            @end
//
//
//  @include  Estos son los archivos utilizados con el código del WDT.
//
//-----*/
#ifndef WDT
#define WDT
#include "WDT.h"
#endif
/*
//
//  @function __configuraWDT_()
//
//  @brief Función que configura el WDT como un contador que observa si se ha bloqueado el programa.
//
//-----*/
void __configuraWDT_()
{
    LPC_WDT->WDTC    = Fwdt*WATCHDOG_TIMEOUT; // Timeout de WATCHDOG_TIMEOUT segundos.
    LPC_WDT->WDCLKSEL = WDCLKSEL_MASK;        // Se selecciona el reloj que se desea para el WDT.
    LPC_WDT->WDMOD   = WDMOD_MASK;           // Se selecciona la acción a realizar si WDT llega a cero.
    alimentaWDT();
}
/*
//
//  @function alimentaWDT()
//
//  @brief Función que evita que el contador del WatchDogTimer llegue a 0.
//
//-----*/
void alimentaWDT()
{
    LPC_WDT->WDFEED = WDT_CODIGO1;
    LPC_WDT->WDFEED = WDT_CODIGO2;
}
/*
//
//  @end    ENDFILE.
//
//-----*/

```

UART0.h

```
/*
//  @filename  UART0.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Este es el programa que recoge la transmisión por UART0.
//
//  @category  Opcional.
//
//  @map      @include
//            @function
//            @end
//
//  @include   Includes pertenecientes a e la transmisión asíncrona.
//
//----- */
#ifndef LPC17XX
#define  LPC17XX
#include "LPC17XX.H"
#endif
#ifndef SYSTEMSYMBOLS
#define  SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
#ifndef STDIO
#define  STDIO
#include <stdio.h>
#endif
#ifndef STRING
#define  STRING
#include <string.h>
#endif
#ifndef PWM
#define  PWM
#include "PWM.h"
#endif
#ifndef UART
#define  UART
#include "uart.h"
#endif
#ifndef MIGLOBAL
#define  MIGLOBAL
#include "miGlobal.h"
#endif
/*
//  @private   Estos son los símbolos correspondientes al protocolo UART0.
//
//----- */
#define DLP      7
#define UART0_MTX  (1 << 1)
#define UART0_MRX  (1 << 2)
#define RetornoDeCarro 13
#define CADMAX   120

#define COM10    "GIVE SUGAR\r"
#define COM11    "GIVE IP\r"
#define COM12    "GIVE TEMPERATURA\r"
#define COM13    "GIVE PRESION\r"
#define COM14    "GIVE VIENTO\r"
```

```
#define COM15      "GIVE LUGAR\r"
#define COM16      "GIVE INDICEUV\r"
#define COM17      "GIVE HORA\r"
#define COM18      "GIVE HUMEDAD\r"
#define COM19      "GIVE BRILLO\r"

#define COM20      "SET BRILLO\r"
#define COM21      "SET HORA\r"
#define COM22      "SET MIN TEMP\r"
#define COM23      "SET MAX TEMP\r"
#define COM24      "SET MIN PRES\r"
#define COM25      "SET MAX PRES\r"
#define COM26      "SET TEMPERATURA\r"
#define COM27      "SET PRESION\r"

#define COM3       "KILL\r"

#define COM0       "ABOUT\r"
#define COM4       "HELP\r"
#define COM41      "HELP GIVE\r"
#define COM42      "HELP SET\r"

#define UART_TX     0
#define UART_RX_BRILLO 1
#define UART_RX_MINT 2
#define UART_RX_MAXT 3
#define UART_RX_MINP 4
#define UART_RX_MAXP 5
#define UART_RX_HORA 6
#define UART_RX_VARM 7

/*
//                                     //           //
//      @funcdef   Estas son las funciones correspondientes al protocolo UART0.          //
//                                     //           //           //
//-----*/                                //           //
void __configuraUART0__( void );
uint8_t procesarComando( char * );
/*
//                                     //           //
//      @end    ENDFILE.                      //           //
//                                     //           //
//-----*/                                //           //
```

UART0.c

```
/*
//  @filename  UART0.c
//  @version   2.01
//  @author    Alberto Palomo Alonso
//
//  @brief     Este es el programa que recoge la transmisión por UART0.
//
//  @category  Opcional.
//
//  @map      @include
//            @variables
//            @function
//            @end
//
//  @include   Includes pertenecientes a la transmisión asíncrona.
//
//  @variables Variables del fichero.
//
//  Variables globales y externas.
char UARTO_BUFFER_RX[CADMAX + 1];
extern char bufferx[30];
extern misDatos_t * DATOS;
extern uint8_t Clock[23];
uint8_t EstadoUARTO = UART_RX;
extern modificables_t MODIFICABLES;
uint8_t Inmortal = 0;
//
//  @function  __configuraUARTO_()
//  @brief     Esta función es la que configura el UART0 para transmisión y recepción.
//
void __configuraUARTO_(void) // Configurado a 9600 baudios.
{
    uart0_init(9600);
    tx_cadena_UART0( "Hola.\n" );
}
//
//  @function  procesarComando()
//  @brief     Esta función manda el UARTO_BUFFER_RX a la salida TX del UART0.
//  @input     char * Buff: El buffer donde está contenido el comando.
//  @ret       Devuelve 1 si ha sido exitoso y 0 si no se ha obtenido el comando.
//
uint8_t procesarComando( char * Buff )
```

```

{
    uint8_t    retval = 0;
    switch( EstadoUART0 )
    {
        case UART_TX:
            /**SECCIÓN PARA LOS COMANDOS DE TIPO 0: ABOUT*/
            if ( !strcmp( Buff ,  COM0 ) )
            {
                retval = 1;
                strcpy( UART0_BUFFER_TX ,  "\n Autor: \t Alberto Palomo Alonso \n Version: \t 2.1.0 \n Sistemas Electronicos
Digitales Avanzados \t UAH \n" );
            }
            /**SECCIÓN PARA LOS COMANDOS DE TIPO 1: GIVE*/
            if ( !strcmp( Buff ,  COM10) )
            {
                if( !Inmortal )
                {
                    strcpy ( UART0_BUFFER_TX ,  "\nSUGAR n.n\n");
                    Inmortal = 1;
                    retval = 1;
                }
            }
            if ( !strcmp( Buff ,  COM11) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nIP: %d.%d.%d.%d \n", __IP1B, __IP2B, __IP3B, __IP4B);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM12) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nTEMPERATURA: %f ºC\n", DATOS->Temperatura);

                retval = 1;
            }
            if ( !strcmp( Buff ,  COM13) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nPRESION: %f mBar.\n", DATOS->Presion);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM14) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nVELOCIDAD DEL VIENTO: %f m.s.\n", DATOS->VelViento);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM15) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nx: NA \nY: NA \nZ: %f m.\n", DATOS->Lugar.Altura);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM16) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nINDICE UV: %f\n", DATOS->IndiceUV);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM17) )
            {
                strcpy ( UART0_BUFFER_TX ,  (const char *)Clock);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM18) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nHUMEDAD: %f \n", 0.01*DATOS->Humedad);
                retval = 1;
            }
            if ( !strcmp( Buff ,  COM19) )
            {
                sprintf ( UART0_BUFFER_TX ,  "\nBRILLO: %f LUX.\n", DATOS->Brillo);
            }
    }
}

```

```
    retval = 1;
}
/**SECCIÓN PARA LOS COMANDOS DE TIPO 3: KILL*/
if ( !strcmp( Buff , COM3 ) )
{
    while ( !Inmortal );
    strcpy( UARTO_BUFFER_TX , "Demasiado dulce como para matarlo, mejor para otra ocasion...\n");
    retval = 1;
}
/**SECCIÓN PARA LOS COMANDOS DE TIPO 4: HELP */
if ( !strcmp( Buff , COM4 ) )
{
    strcpy( UARTO_BUFFER_TX , "Informacion:\n\n ABOUT: Muestra info. del sistema.\n GIVE: Proporciona el dato deseado.\n KILL: Cuelga el programa.\n SET: Modifica variables.\n" );
    retval = 1;
}
if ( !strcmp( Buff , COM41 ) )
{
    strcpy( UARTO_BUFFER_TX , "\nGIVE + [IP, TEMPERATURA, PRESION, BRILLO, LUGAR, VIENTO, INDICEUV, HORA, HUMEDAD]\n" );
    retval = 1;
}
if ( !strcmp( Buff , COM42 ) )
{
    strcpy( UARTO_BUFFER_TX , "\nSET + [BRILLO, HORA, MIN TEMP, MAX TEMP, MIN PRES, MAX PRES, TEMPERATURA, PRESION]\n" );
    retval = 1;
}
/**CONTROL DE ERROR: */
if( !retval )
{
    strcpy( UARTO_BUFFER_TX, "Error: comando no definido, escriba 'HELP' para ver la lista.\n");
}
/** SECCIÓN PARA LOS COMANDOS DE TIPO 2: SET/
if ( !strcmp( Buff , COM20 ) )
{
    strcpy( UARTO_BUFFER_TX , "Introduzca los segundos de brillo: \n" );
    retval = 1;
    EstadoUARTO = UART_RX_BRILLO;
}
if ( !strcmp( Buff , COM21 ) )
{
    strcpy( UARTO_BUFFER_TX , "Introduzca la fecha separada por espacios: \n" );
    retval = 1;
    EstadoUARTO = UART_RX_HORA;
}
if ( !strcmp( Buff , COM22 ) )
{
    strcpy( UARTO_BUFFER_TX , "Introduzca el valor minimo de temperatura: \n" );
    retval = 1;
    EstadoUARTO = UART_RX_MINT;
}
if ( !strcmp( Buff , COM23 ) )
{
    strcpy( UARTO_BUFFER_TX , "Introduzca el valor maximo de temperatura: \n" );
    retval = 1;
    EstadoUARTO = UART_RX_MAXT;
}
if ( !strcmp( Buff , COM24 ) )
{
    strcpy( UARTO_BUFFER_TX , "Introduzca el valor minimo de presion: \n" );
    retval = 1;
    EstadoUARTO = UART_RX_MINP;
}
if ( !strcmp( Buff , COM25 ) )
{
```

```

strcpy( UART0_BUFFER_TX , "Introduzca el valor maximo de presion: \n");
retval = 1;
EstadoUART0 = UART_RX_MAXP;
}
if ( !strcmp( Buff , COM27 ) )
{
strcpy( UART0_BUFFER_TX , "Ahora medimos presion... \n" );
retval = 1;
MODIFICABLES.Var_medida = 1;
}
if ( !strcmp( Buff , COM26 ) )
{
strcpy( UART0_BUFFER_TX , "Ahora medimos temperatura... \n" );
retval = 1;
MODIFICABLES.Var_medida = 0;
}
break;

case UART_RX_BRILLO:
sscanf(bufferx, "%d" , &MODIFICABLES.TiempoBrillo);
strcpy( UART0_BUFFER_TX , "Tiempo de hold cambiado.\n" );
EstadoUART0 = UART_TX;
break;
case UART_RX_MINT:
sscanf(bufferx, "%f" , &MODIFICABLES.Min_servo_t);
strcpy( UART0_BUFFER_TX , "Cota minima de temperatura cambiada.\n" );
EstadoUART0 = UART_TX;
break;
case UART_RX_MAXT:
sscanf(bufferx, "%f" , &MODIFICABLES.Max_servo_t);
strcpy( UART0_BUFFER_TX , "Cota maxima de temperatura cambiada.\n" );
EstadoUART0 = UART_TX;
break;
case UART_RX_MINP:
sscanf(bufferx, "%f" , &MODIFICABLES.Min_servo_p);
strcpy( UART0_BUFFER_TX , "Cota minima de presion cambiada.\n" );
EstadoUART0 = UART_TX;
break;
case UART_RX_MAXP:
sscanf(bufferx, "%f" , &MODIFICABLES.Max_servo_p);
strcpy( UART0_BUFFER_TX , "Cota maxima de presion cambiada.\n" );
EstadoUART0 = UART_TX;
break;
case UART_RX_HORA:
sscanf(bufferx, "%d %d %d %d %d" , (int *)&LPC_RTC->DOM , (int *)&LPC_RTC->MONTH, (int *)&LPC_RTC->YEAR,
(int *)&LPC_RTC->HOUR, (int *)&LPC_RTC->MIN, (int *)&LPC_RTC->SEC);
strcpy( UART0_BUFFER_TX , "Hora cambiada...\n" );
EstadoUART0 = UART_TX;
break;
}

tx_cadena_UART0(UART0_BUFFER_TX);
/**ZONA RETURN*/
return retval;
}
//-----// //-----//
// @end ENDFILE. // //-----*/

```

HTTP_SOURCE.h

```
/*
//  @filename  HTTP_SOURCE.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Cabecera que configura la página WEB.
//
//  @category  Opcional.
//
//  @map      @include
//            @funcdef
//            @end
//
//  @include   Estos son los archivos utilizados en el código de configuración.
//
//----- */
#ifndef NETCONFIG
#define NETCONFIG
#include <Net_Config.h>
#endif
#ifndef STDIO
#define STDIO
#include <stdio.h>
#endif
#ifndef LPC17XX
#define LPC17XX
#include <LPC17XX.H>
#endif
#ifndef RTL
#define RTL
#include <RTL.h>
#endif

void __configuraWEB__ ( void );
void __mantenerTCP__ ( void );

/*
//----- */
//  @end    ENDFILE.
//----- */
```

HTTP_SOURCE.c

```
/*
//  @filename  HTTP_SOURCE.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Código que configura la página WEB.
//
//  @category  Opcional.
//
//  @map      @include
//            @funcion
//            @end
//
//  @include  Estos son los archivos utilizados en el código de configuración.
//  //----- */
#ifndef HTTPSOURCE
#define HTTPSOURCE
#include "HTTP_SOURCE.h"
#endif

void __configuraWEB_()
{
    init_TcpNet(); // Inicializamos TcpNet(RTL.h).
}

void __mantenerTCP_()
{
    main_TcpNet();
}
//----- */
//  @end    ENDFILE.
//  //----- */
//----- */
```

NET_CONFIG.h

```
/*
 *      RL-ARM - TCPnet
 *
 *-----*
 *      Name:    NET_CONFIG.C
 *      Purpose: Configuration of RL TCPnet by user
 *      Rev.:    V4.72
 *-----*
 *      This code is part of the RealView Run-Time Library.
 *      Copyright (c) 2004-2013 KEIL - An ARM Company. All rights reserved.
 *-----*/

```

```
#include <Net_Config.h>
#include "miGlobal.h"

----- <<< Use Configuration Wizard in Context Menu >>> -----
//  

//<h>System Definitions  

//=====  

//<i> Global TCPnet System definitions  

//  <s.15>Local Host Name  

//  <i> This is the name under which embedded host  

//  can be accessed on a local area network.  

//  <i> Default: "mcb2300"  

#define LHOST_NAME      "MiniDK2"

//  <o>Memory Pool size <1536-262144;4></4>  

//  <i> This is the size of a memory pool in bytes. Buffers for  

//  TCPnet packets are allocated from this memory pool.  

//  <i> Default: 8000 bytes  

#define MEM_SIZE        3000

//  <o>Tick Timer interval <10=> 10 ms <20=> 20 ms <25=> 25 ms  

//  <40=> 40 ms <50=> 50 ms <100=> 100 ms  

//  <200=> 200 ms  

//  <i> System Tick Timer interval for software timers  

//  <i> Default: 100 ms  

#define TICK_INTERVAL   100

//</h>
//<e>Ethernet Network Interface  

//=====  

//<i> Enable or disable Ethernet Network Interface  

#define ETH_ENABLE      1

//  <h>MAC Address  

//  =====  

//  <i> Local Ethernet MAC Address  

//  <i> Value FF:FF:FF:FF:FF is not allowed.  

//  <i> It is an ethernet Broadcast MAC address.  

//  <o>Address byte 1 <0x00-0xff:2>  

//  <i> LSB is an ethernet Multicast bit.  

//  <i> Must be 0 for local MAC address.  

//  <i> Default: 0x00  

#define _MAC1           0xE0

//  <o>Address byte 2 <0x00-0xff>  

//  <i> Default: 0x30  

#define _MAC2           0xF8

//  <o>Address byte 3 <0x00-0xff>  

//  <i> Default: 0x6C  

#define _MAC3           0x46

//  <o>Address byte 4 <0x00-0xff>
```

```
//      <i> Default: 0x00
#define _MAC4          0x35

//      <o>Address byte 5 <0x00-0xff>
//      <i> Default: 0x00
#define _MAC5          0x45

//      <o>Address byte 6 <0x00-0xff>
//      <i> Default: 0x01
#define _MAC6          0xBC

//      </h>
//      <h>IP Address
//      =====
//      <i> Local Static IP Address
//      <i> Value 255.255.255.255 is not allowed.
//      <i> It is a Broadcast IP address.
//      <o>Address byte 1 <0-255>
//      <i> Default: 192
#define _IP1           __IP1B

//      <o>Address byte 2 <0-255>
//      <i> Default: 168
#define _IP2           __IP2B

//      <o>Address byte 3 <0-255>
//      <i> Default: 0
#define _IP3           __IP3B

//      <o>Address byte 4 <0-255>
//      <i> Default: 100
#define _IP4           __IP4B

//      </h>
//      <h>Subnet mask
//      =====
//      <i> Local Subnet mask
//      <o>Mask byte 1 <0-255>
//      <i> Default: 255
#define _MSK1          __MASK1B

//      <o>Mask byte 2 <0-255>
//      <i> Default: 255
#define _MSK2          __MASK2B

//      <o>Mask byte 3 <0-255>
//      <i> Default: 255
#define _MSK3          __MASK3B

//      <o>Mask byte 4 <0-255>
//      <i> Default: 0
#define _MSK4          __MASK4B

//      </h>
//      <h>Default Gateway
//      =====
//      <i> Default Gateway IP Address
//      <o>Address byte 1 <0-255>
//      <i> Default: 192
#define _GW1           __GW1B

//      <o>Address byte 2 <0-255>
//      <i> Default: 168
#define _GW2           __GW2B

//      <o>Address byte 3 <0-255>
```

```
//      <i> Default: 0
#define _GW3          __GW3B

//      <o>Address byte 4 <0-255>
//      <i> Default: 254
#define _GW4          __GW4B

//      </h>
//      <h>Primary DNS Server
//      =====
//      <i> Primary DNS Server IP Address
//      <o>Address byte 1 <0-255>
//      <i> Default: 194
#define _pDNS1        192

//      <o>Address byte 2 <0-255>
//      <i> Default: 25
#define _pDNS2        168

//      <o>Address byte 3 <0-255>
//      <i> Default: 2
#define _pDNS3        5

//      <o>Address byte 4 <0-255>
//      <i> Default: 129
#define _pDNS4        1

//      </h>
//      <h>Secondary DNS Server
//      =====
//      <i> Secondary DNS Server IP Address
//      <o>Address byte 1 <0-255>
//      <i> Default: 194
#define _sDNS1        194

//      <o>Address byte 2 <0-255>
//      <i> Default: 25
#define _sDNS2        25

//      <o>Address byte 3 <0-255>
//      <i> Default: 2
#define _sDNS3        2

//      <o>Address byte 4 <0-255>
//      <i> Default: 130
#define _sDNS4        130

//      </h>
//      <h>ARP Definitions
//      =====
//      <i> Address Resolution Protocol Definitions
//      <o>Cache Table size <5-100>
//      <i> Number of cached hardware/IP addresses
//      <i> Default: 10
#define ARP_TABSIZE   10

//      <o>Cache Timeout in seconds <5-255>
//      <i> A timeout for a cached hardware/IP addresses
//      <i> Default: 150
#define ARP_TIMEOUT   150

//      <o>Number of Retries <0-20>
//      <i> Number of Retries to resolve an IP address
//      <i> before ARP module gives up
//      <i> Default: 4
#define ARP_MAXRETRY  4
```

```
//      <o>Resend Timeout in seconds <1-10>
//      <i> A timeout to resend the ARP Request
//      <i> Default: 2
#define ARP_RESEND      2

//      <q>Send Notification on Address changes
//      <i> When this option is enabled, the embedded host
//      <i> will send a Gratuitous ARP notification at startup,
//      <i> or when the device IP address has changed.
//      <i> Default: Disabled
#define ARP_NOTIFY      0

//      </h>
//      <e>IGMP Group Management
//      =====
//      <i> Enable or disable Internet Group Management Protocol
#define IGMP_ENABLE      0

//      <o>Membership Table size <2-50>
//      <i> Number of Groups this host can join
//      <i> Default: 5
#define IGMP_TABSIZE     5

//      </e>
//      <q>NetBIOS Name Service
//      =====
//      <i> When this option is enabled, the embedded host can be
//      <i> accessed by his name on the local LAN using NBNS protocol.
#define NBNS_ENABLE      0

//      <e>Dynamic Host Configuration
//      =====
//      <i> When this option is enabled, local IP address, Net Mask
//      <i> and Default Gateway are obtained automatically from
//      <i> the DHCP Server on local LAN.
#define DHCP_ENABLE      0

//      <s.40>Vendor Class Identifier
//      <i> This value is optional. If specified, it is added
//      <i> to DHCP request message, identifying vendor type.
//      <i> Default: ""
#define DHCP_VCID        ""

//      <q>Bootfile Name
//      <i> This value is optional. If enabled, the Bootfile Name
//      <i> (option 67) is also requested from DHCP server.
//      <i> Default: disabled
#define DHCP_BOOTF       0

//      <q>NTP Servers
//      <i> This value is optional. If enabled, a list of NTP Servers
//      <i> (option 42) is also requested from DHCP server.
//      <i> Default: disabled
#define DHCP_NTPSRV     0

//      </e>
//</e>

//<e>PPP Network Interface
//=====
//<i> Enable or disable PPP Network Interface
#define PPP_ENABLE       0

//      <h>IP Address
//      =====
```

```
//  <i> Local Static IP Address
//    <o>Address byte 1 <0-255>
//    <i> Default: 192
#define _IP1P      192

//  <o>Address byte 2 <0-255>
//  <i> Default: 168
#define _IP2P      168

//  <o>Address byte 3 <0-255>
//  <i> Default: 125
#define _IP3P      125

//  <o>Address byte 4 <0-255>
//  <i> Default: 1
#define _IP4P      1

//  </h>
//  <h>Subnet mask
//  =====
//  <i> Local Subnet mask
//    <o>Mask byte 1 <0-255>
//    <i> Default: 255
#define _MSK1P      255

//  <o>Mask byte 2 <0-255>
//  <i> Default: 255
#define _MSK2P      255

//  <o>Mask byte 3 <0-255>
//  <i> Default: 255
#define _MSK3P      255

//  <o>Mask byte 4 <0-255>
//  <i> Default: 0
#define _MSK4P      0

//  </h>
//  <h>Primary DNS Server
//  =====
//  <i> Primary DNS Server IP Address
//    <o>Address byte 1 <0-255>
//    <i> Default: 194
#define _pDNS1P      194

//  <o>Address byte 2 <0-255>
//  <i> Default: 25
#define _pDNS2P      25

//  <o>Address byte 3 <0-255>
//  <i> Default: 2
#define _pDNS3P      2

//  <o>Address byte 4 <0-255>
//  <i> Default: 129
#define _pDNS4P      129

//  </h>
//  <h>Secondary DNS Server
//  =====
//  <i> Secondary DNS Server IP Address
//    <o>Address byte 1 <0-255>
//    <i> Default: 194
#define _sDNS1P      194

//  <o>Address byte 2 <0-255>
```

```
//      <i>Default: 25
#define _sDNS2P      25

//      <o>Address byte 3 <0-255>
//      <i>Default: 2
#define _sDNS3P      2

//      <o>Address byte 4 <0-255>
//      <i>Default: 130
#define _sDNS4P      130

//      </h>
//      <e>Logon Authentication
//      =====
//      <i>Enable or disable user authentication
#define PPP_AUTHEN    1

//      <q>Unsecured password (PAP)
//      <i>Allow or use Password Authentication Protocol.
#define PPP_PAPEN    1

//      <q>Secured password (CHAP-MD5)
//      <i>Request or use Challenge Handshake Authentication
//      <i>Protocol with MD5 digest algorithm.
#define PPP_CHAPEN    1

//      </e>
//      <q>Obtain Client IP address automatically
//      =====
//      <i>This option only applies when PPP Dial-up is used to dial
//      <i>to remote PPP Server. If checked, network connection
//      <i>dynamically obtains an IP address from remote PPP Server.
#define PPP_GETIP    1

//      <q>Use Default Gateway on remote Network
//      =====
//      <i>This option only applies when both Ethernet and PPP Dial-up
//      <i>are used. If checked, data that cannot be sent to local LAN
//      <i>is forwarded to Dial-up network instead.
#define PPP_DEFGW    1

//      <o>Async Control Character Map <0x0-0xffffffff>
//      <i>A bit-map of control characters 0-31, which are
//      <i>transmitted escaped as a 2 byte sequence.
//      <i>For XON/XOFF set this value to: 0x000A 0000
//      <i>Default: 0x00000000
#define PPP_ACCM      0x00000000

//      <o>LCP Echo Interval in seconds <0-3600>
//      <i>If no frames are received within this interval, PPP sends an
//      <i>Echo Request and expects an Echo Response from the peer.
//      <i>If the response is not received, the link is terminated.
//      <i>A value of 0 disables the LCP Echo test.
//      <i>Default: 30
#define PPP_ECHOTOUT  30

//      <o>Number of Retries <0-20>
//      <i>How many times PPP will try to retransmit data
//      <i>before giving up. Increase this value for links
//      <i>with low baud rates or high latency.
//      <i>Default: 3
#define PPP_MAXRETRY  3

//      <o>Retry Timeout in seconds <1-10>
//      <i>If no response received within this time frame,
//      <i>PPP module will try to resend the data again.
```

```
// <i> Default: 2
#define PPP_RETRYOUT 2

//</e>
//<e>SLIP Network Interface
//=====
//<i> Enable or disable SLIP Network Interface
#define SLIP_ENABLE 0

// <h>IP Address
// =====
// <i> Local Static IP Address
// <o>Address byte 1 <0-255>
// <i> Default: 192
#define _IP1S 192

// <o>Address byte 2 <0-255>
// <i> Default: 168
#define _IP2S 168

// <o>Address byte 3 <0-255>
// <i> Default: 225
#define _IP3S 225

// <o>Address byte 4 <0-255>
// <i> Default: 1
#define _IP4S 1

// </h>
// <h>Subnet mask
// =====
// <i> Local Subnet mask
// <o>Mask byte 1 <0-255>
// <i> Default: 255
#define _MSK1S 255

// <o>Mask byte 2 <0-255>
// <i> Default: 255
#define _MSK2S 255

// <o>Mask byte 3 <0-255>
// <i> Default: 255
#define _MSK3S 255

// <o>Mask byte 4 <0-255>
// <i> Default: 0
#define _MSK4S 0

// </h>
// <h>Primary DNS Server
// =====
// <i> Primary DNS Server IP Address
// <o>Address byte 1 <0-255>
// <i> Default: 194
#define _pDNS1S 194

// <o>Address byte 2 <0-255>
// <i> Default: 25
#define _pDNS2S 25

// <o>Address byte 3 <0-255>
// <i> Default: 2
#define _pDNS3S 2

// <o>Address byte 4 <0-255>
// <i> Default: 129
```

```
#define _pDNS4S      129

//  </h>
//  <h>Secondary DNS Server
//  =====
//  <i> Secondary DNS Server IP Address
//  <o>Address byte 1 <0-255>
//  <i> Default: 194
#define _sDNS1S      194

//  <o>Address byte 2 <0-255>
//  <i> Default: 25
#define _sDNS2S      25

//  <o>Address byte 3 <0-255>
//  <i> Default: 2
#define _sDNS3S      2

//  <o>Address byte 4 <0-255>
//  <i> Default: 130
#define _sDNS4S      130

//  </h>
//  <q>Use Default Gateway on remote Network
//  =====
//  <i> This option only applies when both Ethernet and SLIP Dial-up
//  <i> are used. If checked, data that cannot be sent to local LAN
//  <i> is forwarded to Dial-up network instead.
#define SLIP_DEFGW    1

//</e>
//<e>UDP Sockets
//=====
//<i> Enable or disable UDP Sockets
#define UDP_ENABLE    1

//  <o>Number of UDP Sockets <1-20>
//  <i> Number of available UDP sockets
//  <i> Default: 5
#define UDP_NUMSOCKS  2

//</e>
//<e>TCP Sockets
//=====
//<i> Enable or disable TCP Sockets
#define TCP_ENABLE    1

//  <o>Number of TCP Sockets <1-20>
//  <i> Number of available TCP sockets
//  <i> Default: 5
#define TCP_NUMSOCKS  12

//  <o>Number of Retries <0-20>
//  <i> How many times TCP module will try to retransmit data
//  <i> before giving up. Increase this value for high-latency
//  <i> and low_throughput networks.
//  <i> Default: 5
#define TCP_MAXRETRY   5

//  <o>Retry Timeout in seconds <1-10>
//  <i> If data frame not acknowledged within this time frame,
//  <i> TCP module will try to resend the data again.
//  <i> Default: 4
#define TCP_RETRYOUT   4

//  <o>Default Connect Timeout in seconds <1-600>
```

```
//  <i>Default TCP Socket Keep Alive timeout. When it expires
//  <i>with no TCP data frame send, TCP Connection is closed.
//  <i>Default: 120
#define TCP_DEFOUT    120

//  <o>Maximum Segment Size <536-1460>
//  <i>The Maximum Segment Size specifies the maximum
//  <i>number of bytes in the TCP segment's Data field.
//  <i>Default: 1460
#define TCP_MAXSEGSZ  1460

//  <o>Receive Window Size <536-65535>
//  <i>Receive Window Size specifies the size of data,
//  <i>that the socket is able to buffer in flow-control mode.
//  <i>Default: 4380
#define TCP_RECWSZ    4380

/* TCP fixed timeouts */
#define TCP_INIT_RETRY_TOUT 1      /* TCP initial Retransmit period in sec. */
#define TCP_SYN_RETRY_TOUT  2      /* TCP SYN frame retransmit period in sec. */
#define TCP_CONTRY        7      /* Number of retries to establish a conn. */

//</e>
//<e>HTTP Server
//=====
//<i>Enable or disable HTTP Server
#define HTTP_ENABLE     1

//  <o>Number of HTTP Sessions <1-10>
//  <i>Number of simultaneously active HTTP Sessions.
//  <i>Default: 3
#define HTTP_NUMSESS   6

//  <o>Port Number <1-65535>
//  <i>Listening port number.
//  <i>Default: 80
#define HTTP_PORTNUM   80

//  <s.50>Server-Id header
//  <i>This value is optional. If specified, it overrides
//  <i>the default HTTP Server header from the library.
//  <i>Default: ""
#define HTTP_SRVID     ""

//  <e>Enable User Authentication
//  <i>When enabled, the user will have to authenticate
//  <i>himself by username and password before accessing
//  <i>any page on this Embedded WEB server.
#define HTTP_ENAUTH    1

//  <s.20>Authentication Realm
//  <i>Default: "Embedded WEB Server"
#define HTTP_AUTHREALM "Embedded WEB Server"

//  <s.15>Authentication Username
//  <i>Default: "admin"
#define HTTP_AUTHUSER   "user"

//  <s.15>Authentication Password
//  <i>Default: ""
#define HTTP_AUTHPASSW "Alver"

//</e>
//</e>
//<e>Telnet Server
//=====
```

```
//<i> Enable or disable Telnet Server
#define TNET_ENABLE 0

//  <o>Number of Telnet Connections <1-10>
//  <i> Number of simultaneously active Telnet Connections.
//  <i> Default: 1
#define TNET_NUMSESS 2

//  <o>Port Number <1-65535>
//  <i> Listening port number.
//  <i> Default: 23
#define TNET_PORTNUM 23

//  <o>Idle Connection Timeout in seconds <0-3600>
//  <i> When timeout expires, the connection is closed.
//  <i> A value of 0 disables disconnection on timeout.
//  <i> Default: 120
#define TNET_IDLETOUT 120

//  <q>Disable Echo
//  <i> When disabled, the server will not echo
//  <i> characters it receives.
//  <i> Default: Not disabled
#define TNET_NOECHO 0

//  <e>Enable User Authentication
//  <i> When enabled, the user will have to authenticate
//  <i> himself by username and password before access
//  <i> to the system is allowed.
#define TNET_ENAUTH 1

//  <s.15>Authentication Username
//  <i> Default: "admin"
#define TNET_AUTHUSER "admin"

//  <s.15>Authentication Password
//  <i> Default: ""
#define TNET_AUTHPASSW ""

//  </e>
//</e>
//<e>TFTP Server
//=====
//<i> Enable or disable TFTP Server
#define TFTP_ENABLE 0

//  <o>Number of TFTP Sessions <1-10>
//  <i> Number of simultaneously active TFTP Sessions
//  <i> Default: 1
#define TFTP_NUMSESS 1

//  <o>Port Number <1-65535>
//  <i> Listening port number.
//  <i> Default: 69
#define TFTP_PORTNUM 69

//  <q>Enable Firewall Support
//  <i> Use the same Port Number to receive
//  <i> requests and send answers to clients.
//  <i> Default: Not Enabled
#define TFTP_ENFWALL 0

//  <o>Inactive Session Timeout in seconds <5-120>
//  <i> When timeout expires TFTP Session is closed.
//  <i> Default: 15
#define TFTP_DEFTOUT 15
```

```
//  <o>Number of Retries <1-10>
//  <i> How many times TFTP Server will try to
//  <i> retransmit the data before giving up.
//  <i> Default: 4
#define TFTP_MAXRETRY 4

//</e>
//<e>TFTP Client
//=====
//<i> Enable or disable TFTP Client
#define TFTPC_ENABLE 0

//  <o>Block Size <128=>128  <256=>256  <512=>512
//  <i>          <1024=>1024 <1428=>1428
//  <i> Size of transfer block in bytes.
//  <i> Default: 512
#define TFTPC_BLOCKSZ 512

//  <o>Number of Retries <1-10>
//  <i> How many times TFTP Client will try to
//  <i> retransmit the data before giving up.
//  <i> Default: 4
#define TFTPC_MAXRETRY 4

//  <o>Retry Timeout <2=>200 ms <5=>500 ms <10=>1 sec
//  <i>          <20=>2 sec <50=>5 sec <100=>10 sec
//  <i> If data frame not acknowledged within this time frame,
//  <i> TFTP Client will try to resend the data again.
//  <i> Default: 500 ms
#define TFTPC_RETRYTO 5

//</e>
//<e>FTP Server
//=====
//<i> Enable or disable FTP Server
#define FTP_ENABLE 0

//  <o>Number of FTP Sessions <1-10>
//  <i> Number of simultaneously active FTP Sessions
//  <i> Default: 1
#define FTP_NUMSESS 3

//  <o>Port Number <1-65535>
//  <i> Listening port number.
//  <i> Default: 21
#define FTP_PORTNUM 21

//  <s.50>Welcome Message
//  <i> This value is optional. If specified,
//  <i> it overrides the default welcome message.
//  <i> Default: ""
#define FTP_WELMSG ""

//  <o>Idle Session Timeout in seconds <0-3600>
//  <i> When timeout expires, the connection is closed.
//  <i> A value of 0 disables disconnection on timeout.
//  <i> Default: 120
#define FTP_IDLEOUT 120

//  <e>Enable User Authentication
//  <i> When enabled, the user will have to authenticate
//  <i> himself by username and password before access
//  <i> to the system is allowed.
#define FTP_ENAUTH 1
```

```
//      <s.15>Authentication Username
//      <i> Default: "admin"
#define FTP_AUTHUSER    "admin"

//      <s.15>Authentication Password
//      <i> Default: ""
#define FTP_AUTHPASSW   ""

//  </e>
//</e>
//<e>FTP Client
//=====
//<i> Enable or disable FTP Client
#define FTPC_ENABLE     0

//      <o>Response Timeout in seconds <1-120>
//      <i> This is a time for FTP Client to wait for a response from
//      <i> the Server. If timeout expires, Client aborts operation.
//      <i> Default: 10
#define FTPC_DEFTOUT    10

//      <q>Passive mode (PASV)
//      <i> The client initiates a data connection to the server.
//      <i> Default: Not passive (Active)
#define FTPC_PASVMODE   0

//</e>
//<e>DNS Client
//=====
//<i> Enable or disable DNS Client
#define DNS_ENABLE      0

//      <o>Cache Table size <5-100>
//      <i> Number of cached DNS host names/IP addresses
//      <i> Default: 20
#define DNS_TABSIZE     20

//</e>
//<e>SMTP Client
//=====
//<i> Enable or disable SMTP Client
#define SMTP_ENABLE     0

//      <o>Response Timeout in seconds <5-120>
//      <i> This is a time for SMTP Client to wait for a response from
//      <i> the SMTP Server. If timeout expires, Client aborts operation.
//      <i> Default: 20
#define SMTP_DEFTOUT    20

//</e>
//<e>SNMP Agent
//=====
//<i> Enable or disable SNMP Agent
#define SNMP_ENABLE     0

//      <s.15>Community Name
//      <i> Defines where an SNMP message is destined for.
//      <i> Default: "public"
#define SNMP_COMMUNITY  "public"

//      <o>Port Number <1-65535>
//      <i> Listening port number.
//      <i> Default: 161
#define SNMP_PORTNUM    161

//      <o>Trap Port Number <1-65535>
```

```
//  <i> Port number for Trap operations.  
//  <i> Default: 162  
#define SNMP_TRAPPORT 162  
  
//  <h>Trap Server  
//  =====  
//  <i> Trap Server IP Address  
//  <o>Address byte 1 <0-255>  
//  <i> Default: 192  
#define SNMP_TRAPIP1 192  
  
//  <o>Address byte 2 <0-255>  
//  <i> Default: 168  
#define SNMP_TRAPIP2 168  
  
//  <o>Address byte 3 <0-255>  
//  <i> Default: 0  
#define SNMP_TRAPIP3 0  
  
//  <o>Address byte 4 <0-255>  
//  <i> Default: 100  
#define SNMP_TRAPIP4 1  
  
//  </h>  
//</e>  
//<e>SNTP Client  
//=====  
//<i> Enable or disable SNTP Client  
#define SNTP_ENABLE 0  
  
//  <q>Broadcast Mode  
//  =====  
//  <i> Enable this option, if you have NTP/SNTP server  
//  <i> on LAN, which is broadcasting NTP time messages.  
//  <i> Disable this option to access public NTP server.  
//  <i> Default: disabled  
#define SNTP_BCMODE 0  
  
//  <h>NTP Server  
//  =====  
//  <i> Server IP Address  
//  <o>Address byte 1 <0-255>  
//  <i> Default: 217  
#define SNTP_SRVIP1 217  
  
//  <o>Address byte 2 <0-255>  
//  <i> Default: 79  
#define SNTP_SRVIP2 79  
  
//  <o>Address byte 3 <0-255>  
//  <i> Default: 179  
#define SNTP_SRVIP3 179  
  
//  <o>Address byte 4 <0-255>  
//  <i> Default: 106  
#define SNTP_SRVIP4 106  
  
//  </h>  
//</e>  
//<e>BSD Socket Interface  
//=====  
//<i> Enable or disable Berkeley Socket Programming Interface  
#define BSD_ENABLE 0  
  
//  <o>Number of BSD Sockets <1-20>  
//  <i> Number of available Berkeley Sockets
```

```
//  <i>Default: 2
#define BSD_NUMSOCKS  2

//  <o>Number of Streaming Server Sockets <0-20>
//  <i>Defines a number of Streaming (TCP) Server sockets,
//  <i>that listen for an incoming connection from the client.
//  <i>Default: 1
#define BSD_SRVSOCKS  1

//  <o>Receive Timeout in seconds <0-600>
//  <i>A timeout for socket receive in blocking mode.
//  <i>Timeout value of 0 means indefinite timeout.
//  <i>Default: 20
#define BSD_RCVTOUT  20

//  <q>Hostname Resolver
//  <i>Enable or disable Berkeley style hostname resolver.
#define BSD_GETHOSTEN 0

//</e>
//----- <<< end of configuration section >>> -----


/*
 *      Fatal Error Handler
 */
void sys_error (ERROR_CODE code) {
    /* This function is called when a fatal error is encountered. The normal */
    /* program execution is not possible anymore. Add your critical error     .*/
    /* handler code here. */                                         */

    switch (code) {
        case ERR_MEM_ALLOC:
            /* Out of memory. */
            break;

        case ERR_MEM_FREE:
            /* Trying to release non existing memory block. */
            break;

        case ERR_MEM_CORRUPT:
            /* Memory Link pointer is Corrupted. */
            /* More data written than the size of allocated mem block. */
            break;

        case ERR_MEM_LOCK:
            /* Locked Memory management function (alloc/free) re-entered. */
            /* RTX multithread protection malfunctioning, not implemented */
            /* or interrupt disable is not functioning correctly. */
            break;

        case ERR_UDP_ALLOC:
            /* Out of UDP Sockets. */
            break;

        case ERR_TCP_ALLOC:
            /* Out of TCP Sockets. */
            break;

        case ERR_TCP_STATE:
            /* TCP State machine in undefined state. */
            break;
    }
    /* End-less loop */
    while (1);
}
```

```
/*
 *-----*
 *      TCPnet Config Functions
 *-----*/
#define __NET_CONFIG__
#include <Net_lib.c>

/*
 *-----*
 * end of file
 *-----*/

```

miGlobal.h

```
#define DEFAULT

#ifndef DEFAULT
#define __IP1B 192
#define __IP2B 168
#define __IP3B 1
#define __IP4B 120

#define __GW1B 192
#define __GW2B 168
#define __GW3B 1
#define __GW4B 20

#define __MASK1B255
#define __MASK2B255
#define __MASK3B255
#define __MASK4B0
#endif
```

HTTP_CGI.h

```
/*
//  @filename  HTTP_CGI.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Cabecera que configura los callback la página WEB.
//
//  @category  Opcional.
//
//  @map      @include
//            @private
//            @funcdef
//            @end
//
//  @include   Estos son los archivos utilizados en el código de configuración.
//
//  @private   Estos son los símbolos correspondientes al cgi.
//
//-----*/
#ifndef NETCONFIG
#define NETCONFIG
#include <Net_Config.h>
#endif
#ifndef STDIO
#define STDIO
#include <stdio.h>
#endif
#ifndef STDLIB
#define STDLIB
#include <stdlib.h>
#endif
#ifndef STDINT
#define STDINT
#include <stdint.h>
#endif
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include <Systemsymbols.h>
#endif
#ifndef LPC17XX
#define LPC17XX
#include "LPC17XX.H"
#endif
/**-
//-----*/
//  @private   Estos son los símbolos correspondientes al cgi.
//
//-----*/
#define TEMPERATURA 't'
#define VELOCIDAD 'v'
#define PRESION 'p'
#define HUMEDAD 'h'
#define INDICEUV 'i'
#define BRILLO 'b'
#define ALTITUD 'a'
#define LONGITUD 'x'
#define LATITUD 'y'
#define ANYO 'A'
#define MES 'M'
#define DIA 'D'
#define HORAS 'H'
```

```
#define MINUTOS 'T'  
#define SEGUNDOS 'S'  
/**-----//  
// //-----//  
// @funcdef Estas son las funciones correspondientes al cgi. //  
// //-----*/  
void cgi_process_var ( U8* qs); // NO SE USA.  
void cgi_process_data ( U8 tipo , U8 * qs , U16 longitud); // NO SE USA.  
U16 cgi_func ( U8 * env, U8 * buff , U16 bufflen, U32 * pcgi);  
/*-----//  
// //-----//  
// @end ENDFILE. //  
// //-----*/
```

HTTP_CGI.c

```
/*
//  @filename  HTTP_CGI.c
//  @version   3.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Código que contiene las llamadas a las funciones de CGI.
//
//  @category  Opcional.
//
//  @map      @include
//            @extern
//            @funcion
//            @end
//
//  @include  Estos son los archivos utilizados en el código de configuración.
//
//  @include  misDatos_t * DATOS -> main.c
//
//  @function cgi_process_var
//  @brief   Utilizado para el método GET.
//
void cgi_process_var ( U8* qs)
{
    U8 * var;
    var = (U8 *)alloc_mem(40);

    do
    {
        qs = http_get_var(qs , var, 40);
        if( var[0] )
        {
            if(str_scomp( var , (U8 *)"tmin="))
            {
                sscanf( (const char *)&var[5] , "%f" , &MODIFICABLES.Min_servo_t);
            }
            if(str_scomp( var , (U8 *)"tmax="))
            {
                sscanf( (const char *)&var[5] , "%f" , &MODIFICABLES.Max_servo_t);
            }
            if(str_scomp( var , (U8 *)"pmin="))
            {
                sscanf( (const char *)&var[5] , "%f" , &MODIFICABLES.Min_servo_p);
            }
            if(str_scomp( var , (U8 *)"pmax="))
            {
                sscanf( (const char *)&var[5] , "%f" , &MODIFICABLES.Max_servo_p);
            }
        }
    }
}
```

```

{
    sscanf( (const char *)&var[5] , "%f" , &MODIFICABLES.Max_servo_p);
}
if(str_scomp( var , (U8 *)"bsec="))
{
    sscanf( (const char *)&var[5] , "%d" , &MODIFICABLES.TiempoBrillo);
}
if(str_scomp( var , (U8 *)"vart="))
{
    MODIFICABLES.Var_medida = 0;
}
if(str_scomp( var , (U8 *)"varp="))
{
    MODIFICABLES.Var_medida = 1;
}
}
}while(qs);

free_mem( (OS_FRAME *)var );
}

/*
// @function cgi_process_var
// @brief NO UTILIZADO.
// */
void cgi_process_data ( U8 tipo , U8 * qs , U16 longitud)
{
    // NO UTILIZADO, NO HAY PETICIONES EN ESTA VERSIÓN.
}

/*
// @function cgi_func()
// @brief Función que es llamada por el CGI cada vez que se solicita una callback.
//        Obtiene una cadena de caracteres como parámetro de CGI y actúa en consecuencia.
//        En nuestro caso sólo llama a los datos y los exporta a html.
// @env Cadena de caracteres de entrada.
// @buff Salida de datos.
// @bufflen (No utilizado)Tamaño del buffer.
// @pcgui (No utilizado)
// @return Tamaño de la cadena de salida en bytes.
//
// */
U16 cgi_func ( U8 * env, U8 * buff , U16 bufflen, U32 * pcgi)
{
    U32 longitud;

    switch( env[0] )
    {
        case TEMPERATURA:
            longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Temperatura );
            break;
        case PRESION:
            longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Presion );
            break;
        case HUMEDAD:
            longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Humedad );
            break;
    }
}

```

```

case BRILLO:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Brillo );
    break;
case ALTITUD:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Lugar.Altura );
    break;
case LATITUD:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Lugar.Latitud);
    break;
case LONGITUD:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->Lugar.Longitud);
    break;
case INDICEUV:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->IndiceUV );
    break;
case VELOCIDAD:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , DATOS->VelViento );
    break;
case ANYO:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , LPC_RTC->YEAR );
    break;
case MES:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , LPC_RTC->MONTH );
    break;
case DIA:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , LPC_RTC->DOM );
    break;
case HORAS:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , LPC_RTC->HOUR );
    break;
case MINUTOS:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , LPC_RTC->MIN );
    break;
case SEGUNDOS:
    longitud = sprintf ( (char*)buff , (const char *)&env[4] , LPC_RTC->SEC );
    break;
default:
    longitud = sprintf ( (char*)buff , "<p>Y... que quieres que ponga aqui? :v</p>" );
    break;
}
return ( (U16)longitud );
}

/**-----//**
//                                     //                                     //
//      @end      ENDFILE.                                         //      */
//-----*/

```

Index.cgi

```
t <!DOCTYPE html>
t <html>
t <head>
t   <meta content="text/html; charset=UTF-8" http-equiv="content-type">
#   -----> OJO QUE HAY QUE PONER LA URL CORRECTA:
t   <meta http-equiv="refresh" content="20; url=http://192.168.1.120/index.cgi">
t   <title>ESTACION METEOROLOGICA</title>
t </head>
t   <body style="background-color:rgb(200,200,200);">
t     <h1 align="center">Estacion meteorologica</h1>
t     <br />
t     <table align="center" border="1">
t       <caption>Datos medios actuales:</caption>
t       <tr>
t         <td>Temperatura:</td>
t         <td>
t           " %f
t         </td>
t         <td>Velocidad del viento:</td>
t         <td>
t           " %f
t         </td>
t       </tr>
t       <tr>
t         <td>Humedad:</td>
t         <td>
t           " %f
t         </td>
t         <td>Indice UV:</td>
t         <td>
t           " %f
t         </td>
t       </tr>
t       <tr>
t         <td>Presion:</td>
t         <td>
t           " %f
t         </td>
t         <td>Brillo:</td>
t         <td>
t           " %f
t         </td>
t       </tr>
t     </table>
t     <table align="center" border="1">
t       <tr>
t         <td>Altitud:</td>
t         <td>
t           " %f
t         </td>
t         <td>Longitud:</td>
t         <td>
t           " %f
t         </td>
t         <td>Latitud:</td>
t         <td>
t           " %f
t         </td>
t       </tr>
t     </table>
t     <br><br>
t     <table align="center" border="1">
t       <caption>Hora de la ultima muestra:</caption>
t       <tr>
```

```
t      <td>Anyo:</td>
t      <td>
c A    " %d
t      </td>
t      <td>Mes:</td>
t      <td>
c M    " %d
t      </td>
t      <td>Dia:</td>
t      <td>
c D    " %d
t      </td>
t      <tr>
t      <td>Hora:</td>
t      <td>
c H    " %d
t      </td>
t      <td>Minuto:</td>
t      <td>
c T    " %d
t      </td>
t      <td>Segundos:</td>
t      <td>
c S    " %d
t      </td>
t      </tr>
t      </table>
t      <br><br>
t      <br><br>
t      <table style="width: 100%" border="1" align="center">
t      <tbody>
t      <tr>
t      <td>
t          <h1 style="text-align: center;"> Magnitudes modificables: </h1>
t          <form method="GET" action="index.cgi">
t              <br> Temperatura min. :
t              <input size="10" value="-10" name="tmin" type="text">
t              <br> Temperatura max. :
t              <input size="10" value="50" name="tmax" type="text">
t              <br> Presion min. :
t              <input size="10" value="500" name="pmin" type="text">
t              <br> Presion max. :
t              <input size="10" value="1500" name="pmax" type="text">
t              <br> Segundos encendido :
t              <input size="10" value="10" name="bsec" type="text">
t              <br> <input value="si" type="radio" name="vart"> Temperatura<br>
t              <input value="si" type="radio" name="varp"> Presion<br>
t              <br> <input value="Enviar" type="submit">
t          </form>
t      </td>
t      </tr>
t      </tbody>
t      </table>
t      <br><br>
t      <br><br>
t      <br><br>
t      <br><br>
t      <table align="center">
t      <tr>
t          <td>Autor: Alberto Palomo Alonso.</td>
t          <td>Sistemas Electronicos Digitales Avanzados.</td>
t          <td>Universidad de Alcala - Escuela politecnica superior.</td>
t      </tr>
t      </table>
t      </body>
t  </html>
```

RTC.h

```
/*
//  @filename  RTC.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Cabecera del código RTC.
//
//  @category  Opcional.
//
//  @map      @include
//            @private
//            @funcdef
//            @end
//
//  @include  Estos son los archivos utilizados con el código fuente.
//  @private  Estos son los símbolos privados a RTC.
//  @funcdef  Funciones a definir en el RTC.
//  @end     ENDFILE.
*/
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif

#define MITIEMPO(time_t *)(LPC_RTC_BASE)
#define RTCMASK      1<<9
#define CALIBRACION_RTC (1<<4) | (0x1<<1)
#define CALIBRATION_VALUE 0x00
#define INT_SEGUNDOS 1<<0
```

RTC.c

```
/*
//  @filename  RTC.c
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Código fuente del configurado y manejador del RTC.
//
//  @category  Opcional.
//
//  @map      @include
//            @variables
//            @function
//            @HANDLER
//            @end
//
//  @include   Estos son los archivos utilizados con el código del RTC.
//
//  @variables Variables del fichero.
//
//  uint8_t Clock[23];
extern Counters_t * COUNTERS;
//
//  @function __configuraRTC_()
//  @brief Función que configura el RTC como un contador que interrumpe cada segundo.
//  @Tiempo Puntero a variable donde se almacena el tiempo actual.
//
void __configuraRTC_( void )
{
    LPC_SC->PCONP |= RTCMASK;           // Activo el RTC.
    LPC_RTC->CCR = CALIBRACION_RTC;    // Calibro el RTC.
    LPC_RTC->CALIBRATION = CALIBRATION_VALUE;
    LPC_RTC->CCR = 0x1;
    LPC_RTC->CIIR |= INT_SEGUNDOS;     // Interrupción del RTC cada segundo.
    LPC_RTC->YEAR = 2020;              // Configuro el registro
}

//-----*/
```

```
LPC_RTC->MONTH    = 1;           // que tiene en cuenta
LPC_RTC->DOM      = 1;           // los días, meses,
LPC_RTC->HOUR     = 0;           // minutos y segundos
LPC_RTC->MIN      = 0;           // del RTC.
LPC_RTC->SEC      = 0;           //
NVIC_EnableIRQ( RTC IRQn);      // Habilito la interrupción del RTC.
NVIC_SetPriority( RTC IRQn , 0 ); // Se le asigna prioridad alta.
}
/**-----// */
//                                     //                                     //
// @HANDLER   RTC_IRQHandler()          //          //          //
// @brief Manejador de la interrupción RTC.          //          //
//                                     //          //          //
//-----*/ */

void RTC_IRQHandler( void )
{
    COUNTERS->Segundos    += 0x1; // Incrementa el contador.
    LPC_RTC->ILR        |= 1; // Borra el flag de interrupción.
    sprintf((char*)Clock,"%02d/%02d/%04d-%02d:%02d:%02d",LPC_RTC->DOM,LPC_RTC->MONTH,LPC_RTC->YEAR,LPC_RTC->HOUR,LPC_RTC->MIN,LPC_RTC->SEC);
// if(COUNTERS->Segundos == 60)
// {
//     __sumaMinReloj_();
//     COUNTERS->Segundos = 0;
// }
// */
//                                     //                                     //
//                                     //                                     //
// @end      ENDFILE.          //          //          //
//-----*/
```

Timers.h

```
/*
//  @filename  Timers.h
//  @version   0.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Cabecera para configurar los timers.
//
//  @category  Principal.
//
//  @map      @include
//            @funcdef
//            @end
//
//
//  @include   Estos son los archivos utilizados para los timers.
//
//----- */
#ifndef RTL
#define RTL
#include "RTL.h"
#endif
#ifndef LPC17XX
#define LPC17XX
#include "LPC17XX.H"
#endif
#ifndef SYSTEMSYMBOLS
#define SYSTEMSYMBOLS
#include "Systemsymbols.h"
#endif
#ifndef ANEMOMETRO
#define ANEMOMETRO
#include "Anemometro.h"
#endif
#ifndef LDR
#define LDR
#include "LDR.h"
#endif
#ifndef DAC
#define DAC
#include "DAC.h"
#endif
#ifndef PWM
#define PWM
#include "PWM.h"
#endif
#ifndef UFONO
#define UFONO
#include "uFono.h"
#endif
#ifndef ONEWIRE
#define ONEWIRE
#include "OneWire.h"
#endif
#ifndef I2C
#define I2C
#include "I2C.h"
#endif
/*
//  @private   Estos son los símbolos correspondientes a los timers.
//
//----- //
```

```

//                                            //
//-----*/                                     //
#define FREQ_OVERFLOW_SYSTICK 10          // Frecuencia en Hz de overflow o fin de cuenta del SysTick. (Tsyst =
100ms)                                         //
#define ENABLEBIT_SYST      0x1
#define FCPUBIT_SYST       0x4
#define ENABLEINTBIT_SYST   0x2
#define MASCARA_CTRL_SYSTICK FCPUBIT_SYST | ENABLEBIT_SYST | ENABLEINTBIT_SYST
#define SYSTICK_COUNTFLAG   0x1 << 16
// Para cada timer.
#define ACTIVAR_TIMER        0x1
#define RESET_TIMER_TCR      0x2
#define TIMERO_BIT            (0x1 << 1)
#define TIMER1_BIT            (0x1 << 2)
#define TIMER2_BIT            (0x1 << 22)
#define TIMER3_BIT            (0x1 << 23)
#define TIMERO_MCR_MASK      0x3 << (0*3)           // Activo la interrupción y reseteo el contador.
#define TIMER1_MCR_MASK      0x3 << (0*3)           // No usado.
#define TIMER2_MCR_MASK      0x3 << (0*3)
#define TIMER3_MCR_MASK      0x1 << (0*3)

#define MODO_ENTRADA         1
#define MODO_SALIDA          0
/*-----//                                         //
//                                            //                                         //
//      @funcdef    Estas son las funciones correspondientes a los timers.           //
//-----*/                                     //
void __configuraSysTick__  ( void ); // TCP.
void __configuraTimer0__   ( void ); // Muestreo.
void __configuraTimer1__   ( void ); // ...
void __configuraTimer2__   ( void ); // Audio.
void __configuraTimer3__   ( void ); // ...
/*-----//                                         //
//                                            //                                         //
//      @end      ENDFILE.               //
//-----*/                                     //

```

Timers.c

```
/*
//  @filename  Timers.c
//  @version   7.00
//  @author    Alberto Palomo Alonso
//
//  @brief     Código que configura y programa los manejadores de los timers.
//
//  @category  Principal.
//
//  @map      @include
//            @variables
//            @funcion
//            @end
//
//  @include  Estos son los archivos utilizados para los timers.
//
//  @variables Variables del fichero.
//
//  uint8_t      TIM0_ticks = 0;
//  uint8_t      Timer2_MODO = MODO_SALIDA;
//  uint32_t     CAP11_BUFF = 0;
//  Contador.
//  uint16_t     contadorLUZ = 0;
//  Externos.
extern  uint8_t      __brilloAuto;
extern  uint8_t      __brilloFade;
extern  uint8_t      YaPuedesMedir;
extern  Counters_t   * COUNTERS;
extern  misDatos_t   * DATOS;
extern  actualizador_t * ACTUALIZADOR;
extern  uint8_t      * AUDIO;
extern  uint8_t      * CAPcont;
extern  modificables_t MODIFICABLES;
//
//  @function   __configuraSysTick_()
//
//  @brief     Configura el systick para desbordar cada 100 ms.
//
void __configuraSysTick_()
{
    SysTick->LOAD = (SystemCoreClock / FREQ_OVERFLOW_SYSTICK) - 1; // Systick configurado a desbordar cada 100 ms para TcpNet.
    SysTick->CTRL = MASCARA_CTRL_SYSTICK;                         // Fcpu como clock y no activo la interrupción del SysTickTimer.
    SysTick_Config(SystemCoreClock / FREQ_OVERFLOW_SYSTICK);
}
//
//  @function   __configuraTimer0_()
//
```

```

//                                         //          //
// @brief   Configura el Timer0 para interrumpir cada Ts0 segundos.      //
//                                         //          //
//-----*/                           //          //
void __configuraTimer0_()
{
    LPC_SC->PCONP |= 0x1 << 22 | 0x1 << 23 | 1 << 16; // Todos los timer.
    LPC_SC->PCONP |= TIMERO_BIT;                         // Activo el módulo del timer 0.
    LPC_TIM0->MCR = TIMERO_MCR_MASK;                     // Activo el ISR y reseteo TC.
    LPC_TIM0->PR= 0;                                     // Sin prescaler.
    LPC_TIM0->TCR |= ACTIVAR_TIMER;                      // Activo el timer.
    LPC_TIM0->MRO = Ftick * Ts0 - 1;                     // Cargo para que interrumpa cada 0.5s.
    NVIC_SetPriority( TIMERO IRQn , 1 );                  // Para que el ADC interrumpa bien.
    NVIC_EnableIRQ( TIMERO IRQn );
}
//-----//                                         //          //
//                                         //          //
// @function  __configuraTimer1_()
// @GOTO     iDEFINIDO EN EL ANEMOMETRO! (Anemometro.c)           //
//                                         //          //
//-----*/                           //          //
//                                         //          //
// @function  __configuraTimer3_()
// @GOTO     iDEFINIDO EN EL ONEWIRE! (OneWire.c)           //
//                                         //          //
//-----*/                           //          //
//                                         //          //
// @HANDLER  SysTick_Handler()           //
// @brief    Manejador de la interrupción del SysTick. Cada 100 ms se realizan acciones. //
//-----*/                           //          //
void SysTick_Handler()
{
    timer_tick();
    if(contadorLUZ < FREQ_OVERFLOW_SYSTICK * (MODIFICABLES.TiempoBrillo))
    {
        contadorLUZ++;
    }
    else
    {
        if (__brilloFade)                                // Si pasan 60s y el brillo automático está desactivado...
        {
            __brilloAuto = 0;
            __brilloFade = 0;
            modificaPulso(  PWM6 ,  MODO_CICLO ,  1 ,  none ,  none ,  none ); // Apago la pantalla.
        }
    }
}
//-----//                                         //          //
// @HANDLER  TIMERO IRQHandler()           //
// @brief    Manejador de la interrupción del Timer0. Reanima el muestreo de los sensores. //
//-----*/                           //          //
// Bloque 1: Apoyo del timer 1:

```

```

//          Temperatura + Humedad + Vel.Viento.
static void _subAnemoTempe()
{
    LPC_TIM0->IR = LPC_TIM0->IR;           // Borro interrupción.
    if( !(TIM0_ticks % (uint8_t)CsCAP) )      // Si toca muestrear captures...
    {
        LPC_TIM1->CCR |= CCR_MASCARA_EN;     // Genera interrupción el CAP1.0, ojo que se mata así en el timer 1.
        LPC_TIM1->CCR |= OW_CCR_MASCARA_EN;   // Genera interrupción el CAP1.1, ojo que se mata así en el timer 1.
        mideTemperatura();                   // Le digo a la placa que lance la señal de request.
        medirBMP();                         // Leo el sensor de presión atmosférica.
        if( !ACTUALIZADOR->AnemometroRev && YaPuedesMedir) // Si el actualizador está a 0 (Es decir, no hay datos capturados).
        {
            DATOS->VelViento = 0;           // No hay viento.
            ACTUALIZADOR->Anemometro = 1;    // Ya está medido, es 0 m/s.
        }
        ACTUALIZADOR->AnemometroRev = 0;     // Digo que ya he medido.
        ACTUALIZADOR->TempRev = 1;
    }
}
// Bloque 2: ADC burst:
//          UVA + LDR.
static void _subBurst()
{
    if( !(TIM0_ticks % (uint8_t)CsADC) )      // LDR + UVA van el BURST.
    {
        if( ACTUALIZADOR->LDRrev && YaPuedesMedir ) // Es bloqueable por el audio.
        {
            LPC_SC->PCOMP |= PCOMP_ADC_ON; // Enciendo el ADC.
            ACTUALIZADOR->LDRrev = 0;       // Aviso que no he medido aún.
            LPC_ADC->ADCR |= ~ADC_START; // Ojito que es modo ráfaga, no hay start.
            LPC_ADC->ADCR |= BRUST_PIN;   // Ráfaga.
        }
    }
}
// Actualizo el servo.
void _subServo( void )
{
    if( !MODIFICABLES.Var_medida )
    {
        if( DATOS->Temperatura >= MODIFICABLES.Max_servo_t)
        {
            modificaPulso ( PWM2, MODO_SERVO , none , 180 , MINIMO_SERVO , MAXIMO_SERVO );
            if(ACTUALIZADOR->Audiorev)
            {
                ACTUALIZADOR->Audiorev = 0;
                __configuraTono__();
                activarDac();
            }
        }
        if( DATOS->Temperatura <= MODIFICABLES.Min_servo_t)
        {
            modificaPulso ( PWM2, MODO_SERVO , none , 0 , MINIMO_SERVO , MAXIMO_SERVO );
            if(ACTUALIZADOR->Audiorev)
            {
                ACTUALIZADOR->Audiorev = 0;
                __configuraAudio__();
                activarDac();
            }
        }
        modificaPulso ( PWM2, MODO_SERVO , none , (180*(DATOS->Temperatura - MIN_TEMP)/(MAX_TEMP - MIN_TEMP)) , MINIMO_SERVO , MAXIMO_SERVO );
    }
    else
    {
        if( DATOS->Presion >= MODIFICABLES.Max_servo_p)
    }
}

```



```
//          //          //
//      @HANDLER    TIMER3_IRQHandler()          //          //
//      @brief     Timer de apoyo para el monohilo.          //
//-----*/          //          //
// USADO POR EL MONOHILO.          //
void  TIMER3_IRQHandler()          //
{          //
    // NO USADO.          //
}          //-----*/          //
/*-----*/          //          //
//          //          //
//      @end    ENDFILE.          //          //
//-----*/          //
```

ARD 2

Arduino Compatibles

Controllers, Shields, Modules & Sensors

UV Detection Sensor

ARD2-2062

- Detect the intensity of UV radiation**
- Suitable for UV Index Monitoring, DIY projects, UV-A Lamp Monitoring, Gardening, Environmental monitoring**

Description

This UV Sensor is used for detecting the intensity of incident ultraviolet (UV) radiation. This form of electromagnetic radiation has shorter wavelengths than visible radiation. This module is based on the sensor UVM-30A, which has a wide spectral range of 200nm–370nm. The module outputs electrical signal which varies with the UV intensity.



Specifications

| | |
|------------------------------|-------------------------------|
| Operating Voltage | 3.0–5.0VDC |
| Current | 0.06mA (Standard)/0.1mA (Max) |
| Response Wavelength | 200~370nm |
| Operating Temperature | -20~+85°C |
| Accuracy | ±1UV Index |
| Colour (Board) | Black |
| Material | PCB |

Pinout

| Module | Arduino | Function |
|---------------|----------------|-------------------|
| + | 5V | Power |
| - | GND | Ground Connection |
| Out | A0 | Analog Output |



Test Code

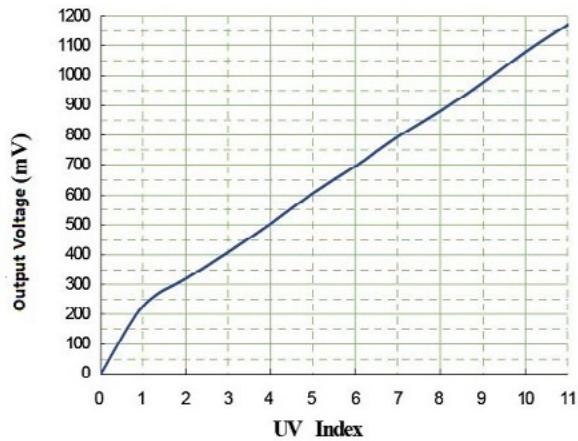
```

void setup()
{
  Serial.begin(9600); // open serial port, set the baud rate to
9600 bps
}
void loop()
{
  int sensorValue;
  sensorValue = analogRead(0); //connect UV sensors to Analog 0
  Serial.println(sensorValue); //print the value to serial
  delay(200);
}

```



ARD 2

Arduino Compatibles
Controllers, Shields, Modules & Sensors

BMP180

Digital pressure sensor

Bosch Sensortec



BOSCH
Invented for life

BMP180 Data sheet

| | |
|-----------------------------|---|
| Document revision | 2.5 |
| Document release date | 5 April 2013 |
| Document number | BST-BMP180-DS000-09 |
| Technical reference code(s) | 0 273 300 244 |
| Notes | Data in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance. |



BMP180

DIGITAL PRESSURE SENSOR

Key features

| | |
|---|---|
| Pressure range: | 300 ... 1100hPa (+9000m ... -500m relating to sea level) |
| Supply voltage: | 1.8 ... 3.6V (V_{DD}) 1.62V ... 3.6V (V_{DDIO}) |
| Package: | LGA package with metal lid Small footprint: 3.6mm x 3.8mm Super-flat: 0.93mm height |
| Low power: | 5µA at 1 sample / sec. in standard mode |
| Low noise: | 0.06hPa (0.5m) in ultra low power mode 0.02hPa (0.17m) advanced resolution mode |
| - Temperature measurement included | |
| - I ² C interface | |
| - Fully calibrated | |
| - Pb-free, halogen-free and RoHS compliant, | |
| - MSL 1 | |

Typical applications

- Enhancement of GPS navigation (dead-reckoning, slope detection, etc.)
- In- and out-door navigation
- Leisure and sports
- Weather forecast
- Vertical velocity indication (rise/sink speed)



BMP180 general description

The BMP180 is the function compatible successor of the BMP085, a new generation of high precision digital pressure sensors for consumer applications.

The ultra-low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment. With a low altitude noise of merely 0.25m at fast conversion time, the BMP180 offers superior performance. The I²C interface allows for easy system integration with a microcontroller.

The BMP180 is based on piezo-resistive technology for EMC robustness, high accuracy and linearity as well as long term stability.

Robert Bosch is the world market leader for pressure sensors in automotive applications. Based on the experience of over 400 million pressure sensors in the field, the BMP180 continues a new generation of micro-machined pressure sensors.

**Index of Contents**

| | |
|--|-----------|
| 1. ELECTRICAL CHARACTERISTICS | 6 |
| 2. ABSOLUTE MAXIMUM RATINGS..... | 8 |
| 3. OPERATION | 9 |
| 3.1 GENERAL DESCRIPTION | 9 |
| 3.2 GENERAL FUNCTION AND APPLICATION SCHEMATICS..... | 9 |
| 3.3 MEASUREMENT OF PRESSURE AND TEMPERATURE | 11 |
| 3.3.1 Hardware pressure sampling accuracy modes | 12 |
| 3.3.2 Software pressure sampling accuracy modes | 13 |
| 3.4 CALIBRATION COEFFICIENTS..... | 13 |
| 3.5 CALCULATING PRESSURE AND TEMPERATURE | 14 |
| 3.6 CALCULATING ABSOLUTE ALTITUDE..... | 16 |
| 3.7 CALCULATING PRESSURE AT SEA LEVEL | 17 |
| 4. GLOBAL MEMORY MAP | 18 |
| 5. I²C INTERFACE | 19 |
| 5.1 I ² C SPECIFICATION..... | 19 |
| 5.2 DEVICE AND REGISTER ADDRESS..... | 20 |
| 5.3 I ² C PROTOCOL | 20 |
| 5.4 START TEMPERATURE AND PRESSURE MEASUREMENT | 21 |
| 5.5 READ A/D CONVERSION RESULT OR E ² PROM DATA | 22 |
| 6. PACKAGE..... | 23 |
| 6.1 PIN CONFIGURATION | 23 |
| 6.2 OUTLINE DIMENSIONS | 24 |
| 6.2.1 Bottom view | 24 |
| 6.2.2 Top view | 25 |
| 6.2.3 Side view | 25 |
| 6.3 MOISTURE SENSITIVITY LEVEL AND SOLDERING..... | 26 |
| 6.4 RoHS COMPLIANCY..... | 26 |
| 6.5 MOUNTING AND ASSEMBLY RECOMMENDATIONS | 26 |
| 7. LEGAL DISCLAIMER..... | 27 |

**BOSCH**Data sheet
BMP180

Page 5

| | |
|--|-----------|
| 7.1 ENGINEERING SAMPLES | 27 |
| 7.2 PRODUCT USE..... | 27 |
| 7.3 APPLICATION EXAMPLES AND HINTS | 27 |
| 8. DOCUMENT HISTORY AND MODIFICATION..... | 28 |



1. Electrical characteristics

If not stated otherwise, the given values are ± 3 -Sigma values over temperature/voltage range in the given operation mode. All values represent the new parts specification; additional solder drift is shown separately.

Table 1: Operating conditions, output signal and mechanical characteristics

| Parameter | Symbol | Condition | Min | Typ | Max | Units |
|---|-------------|----------------------------------|------|-------|----------------|-------|
| Operating temperature | T_A | operational | -40 | | +85 | °C |
| | | full accuracy | 0 | | +65 | |
| Supply voltage | V_{DD} | ripple max. 50mVpp | 1.8 | 2.5 | 3.6 | V |
| | | | 1.62 | 2.5 | 3.6 | |
| Supply current @ 1 sample / sec. 25°C | I_{DDLOW} | ultra low power mode | | 3 | | µA |
| | I_{DDSTD} | standard mode | | 5 | | µA |
| | I_{DDHR} | high resolution mode | | 7 | | µA |
| | I_{DDUHR} | Ultra high res. mode | | 12 | | µA |
| | I_{DDAR} | Advanced res. mode | | 32 | | µA |
| Peak current | I_{peak} | during conversion | | 650 | 1000 | µA |
| Standby current | I_{DDSBM} | @ 25°C | | 0.1 | 4 ¹ | µA |
| Relative accuracy pressure $V_{DD} = 3.3V$ | | 950 ... 1050 hPa @ 25 °C | | ±0.12 | | hPa |
| | | | | ±1.0 | | m |
| | | 700 ... 900hPa 25 ... 40 °C | | ±0.12 | | hPa |
| | | | | ±1.0 | | m |
| Absolute accuracy pressure $V_{DD} = 3.3V$ | | 300 ... 1100 hPa 0 ... +65 °C | -4.0 | -1.0* | +2.0 | hPa |
| | | 300 ... 1100 hPa -20 ... 0 °C | -6.0 | -1.0* | +4.5 | hPa |
| Resolution of output data | | pressure | | 0.01 | | hPa |
| | | temperature | | 0.1 | | °C |
| Noise in pressure | | see table on page 12-13 | | | | |
| Absolute accuracy temperature $V_{DD} = 3.3V$ | | @ 25 °C | -1.5 | ±0.5 | +1.5 | °C |
| | | 0 ... +65 °C | -2.0 | ±1.0 | +2.0 | °C |

¹ at 85°C



| | | | | | | |
|-----------------------------|-----------------------|----------------------------|------|------|------|-----|
| Conversion time pressure | t _{c_p_low} | ultra low power mode | | 3 | 4.5 | ms |
| | t _{c_p_std} | standard mode | | 5 | 7.5 | ms |
| | t _{c_p_hr} | high resolution mode | | 9 | 13.5 | ms |
| | t _{c_p_luhr} | ultra high res. mode | | 17 | 25.5 | ms |
| | t _{c_p_ar} | Advanced res. mode | | 51 | 76.5 | ms |
| Conversion time temperature | t _{C_temp} | standard mode | | 3 | 4.5 | ms |
| Serial data clock | f _{SCL} | | | | 3.4 | MHz |
| Solder drifts | | Minimum solder height 50µm | -0.5 | | +2 | hPa |
| Long term stability** | | 12 months | | ±1.0 | | hPa |

* The typical value is: -1±1

** Long term stability is specified in the full accuracy operating pressure range 0 ... 65°C



2. Absolute maximum ratings

Table 2: Absolute maximum ratings

| Parameter | Condition | Min | Max | Units |
|---------------------|------------------------------|------|--------|-------|
| Storage temperature | | -40 | +85 | °C |
| Supply voltage | all pins | -0.3 | +4.25 | V |
| ESD rating | HBM, R = 1.5kΩ, C = 100pF | | ±2 | kV |
| Overpressure | | | 10,000 | hPa |

The BMP180 has to be handled as

Electrostatic Sensitive Device (ESD).



Figure 1: ESD



3. Operation

3.1 General description

The BMP180 is designed to be connected directly to a microcontroller of a mobile device via the I²C bus. The pressure and temperature data has to be compensated by the calibration data of the E²PROM of the BMP180.

3.2 General function and application schematics

The BMP180 consists of a piezo-resistive sensor, an analog to digital converter and a control unit with E²PROM and a serial I²C interface. The BMP180 delivers the uncompensated value of pressure and temperature. The E²PROM has stored 176 bit of individual calibration data. This is used to compensate offset, temperature dependence and other parameters of the sensor.

- UP = pressure data (16 to 19 bit)
- UT = temperature data (16 bit)

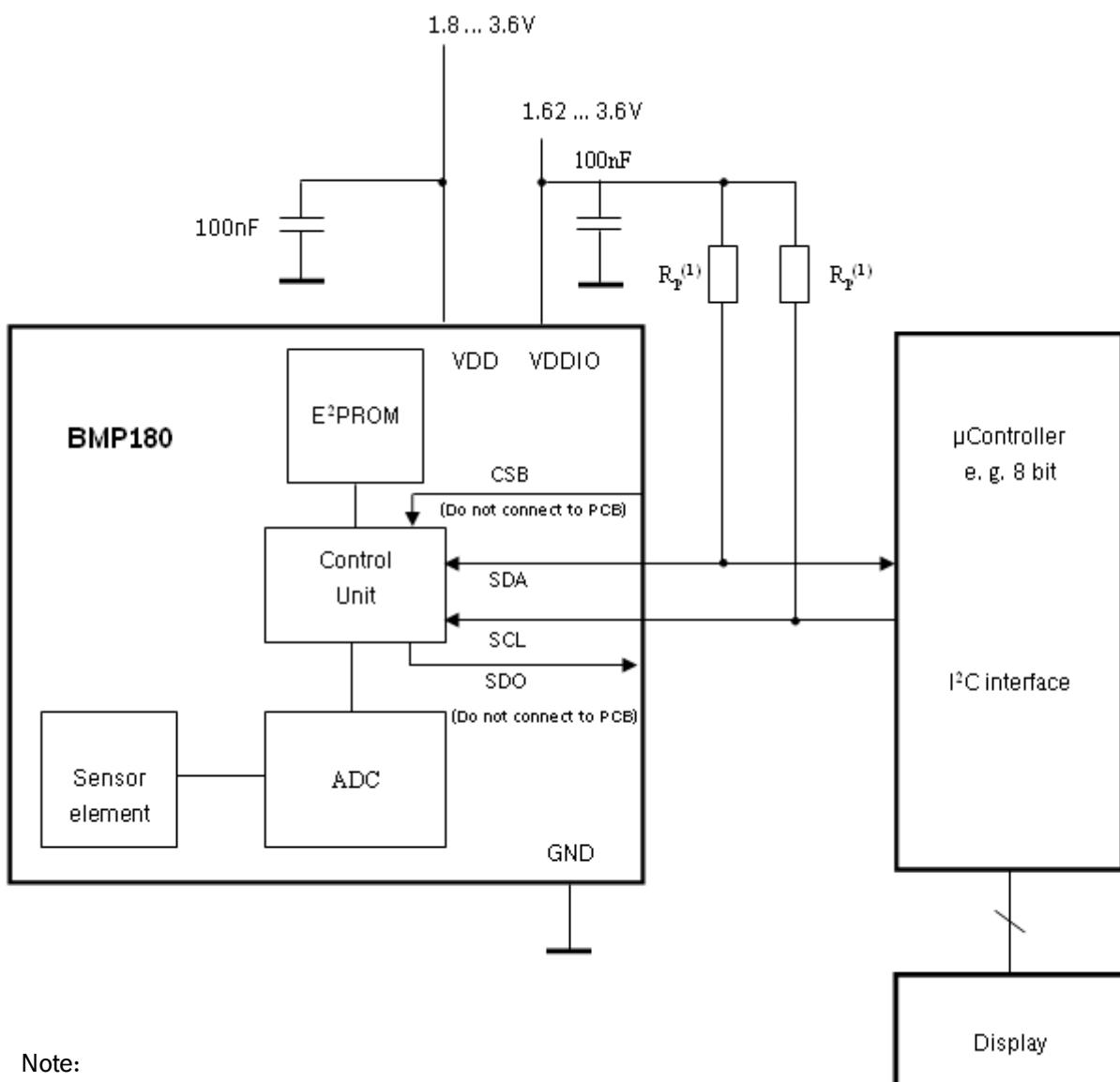


Figure 2: Typical application circuit

3.3 Measurement of pressure and temperature

For all calculations presented here an ANSI C code is available from Bosch Sensortec ("BMP180_AP").

The microcontroller sends a start sequence to start a pressure or temperature measurement. After converting time, the result value (UT or UP, respectively) can be read via the I²C interface. For calculating temperature in °C and pressure in hPa, the calibration data has to be used. These constants can be read out from the BMP180 E²PROM via the I²C interface at software initialization.

The sampling rate can be increased up to 128 samples per second (standard mode) for dynamic measurement. In this case, it is sufficient to measure the temperature only once per second and to use this value for all pressure measurements during the same period.

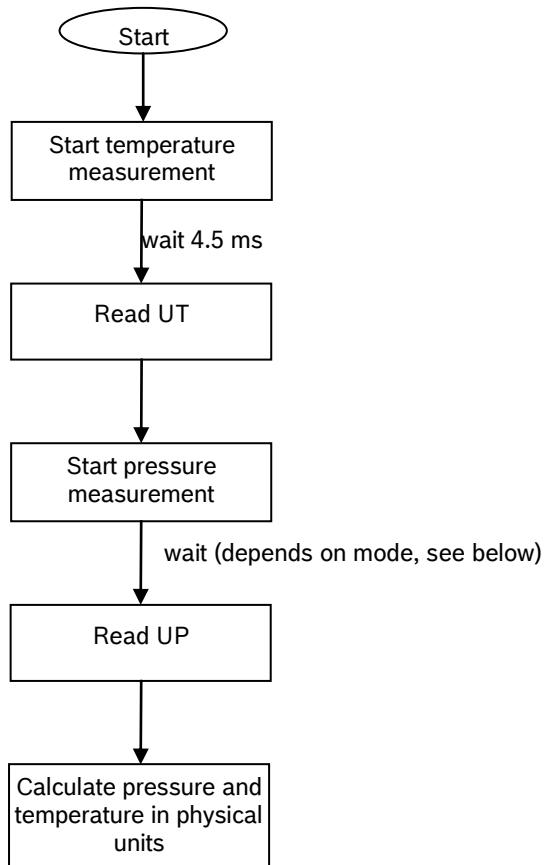


Figure 3: Measurement flow BMP180



3.3.1 Hardware pressure sampling accuracy modes

By using different modes the optimum compromise between power consumption, speed and resolution can be selected, see below table.

Table 3: Overview of BMP180 hardware accuracy modes, selected by driver software via the variable *oversampling_setting*

| Mode | Parameter <i>oversampling_setting</i> | Internal number of samples | Conversion time pressure max. [ms] | Avg. current @ 1 sample/s typ. [µA] | RMS noise typ. [hPa] | RMS noise typ. [m] |
|-----------------------|--|----------------------------------|--|---|-------------------------------|-----------------------------|
| ultra low power | 0 | 1 | 4.5 | 3 | 0.06 | 0.5 |
| standard | 1 | 2 | 7.5 | 5 | 0.05 | 0.4 |
| high resolution | 2 | 4 | 13.5 | 7 | 0.04 | 0.3 |
| ultra high resolution | 3 | 8 | 25.5 | 12 | 0.03 | 0.25 |

For further information on noise characteristics see the relevant application note “Noise in pressure sensor applications”.

All modes can be performed at higher speeds, e.g. up to 128 times per second for standard mode, with the current consumption increasing proportionally to the sample rate.



3.3.2 Software pressure sampling accuracy modes

For applications where a low noise level is critical, averaging is recommended if the lower bandwidth is acceptable. Oversampling can be enabled using the software API driver (with OSR = 3).

Table 4: Overview of BMP180 software accuracy mode, selected by driver software via the variable *software_oversampling_setting*

| Mode | Parameter <i>oversampling_setting</i> | software_oversampling_setting | Conversion time pressure max. [ms] | Avg. current @ 1 sample/s typ. [μ A] | RMS noise typ. [hPa] | RMS noise typ. [m] |
|---------------------|--|-------------------------------|------------------------------------|---|----------------------|--------------------|
| Advanced resolution | 3 | 1 | 76.5 | 32 | 0.02 | 0.17 |

3.4 Calibration coefficients

The 176 bit E²PROM is partitioned in 11 words of 16 bit each. These contain 11 calibration coefficients. Every sensor module has individual coefficients. Before the first calculation of temperature and pressure, the master reads out the E²PROM data.

The data communication can be checked by checking that none of the words has the value 0 or 0xFFFF.

Table 5: Calibration coefficients

| Parameter | BMP180 reg adr | |
|-----------|----------------|------|
| | MSB | LSB |
| AC1 | 0xAA | 0xAB |
| AC2 | 0xAC | 0xAD |
| AC3 | 0xAE | 0xAF |
| AC4 | 0xB0 | 0xB1 |
| AC5 | 0xB2 | 0xB3 |
| AC6 | 0xB4 | 0xB5 |
| B1 | 0xB6 | 0xB7 |
| B2 | 0xB8 | 0xB9 |
| MB | 0xBA | 0xBB |
| MC | 0xBC | 0xBD |
| MD | 0xBE | 0xBF |



3.5 Calculating pressure and temperature

The mode (ultra low power, standard, high, ultra high resolution) can be selected by the variable *oversampling_setting* (0, 1, 2, 3) in the C code.

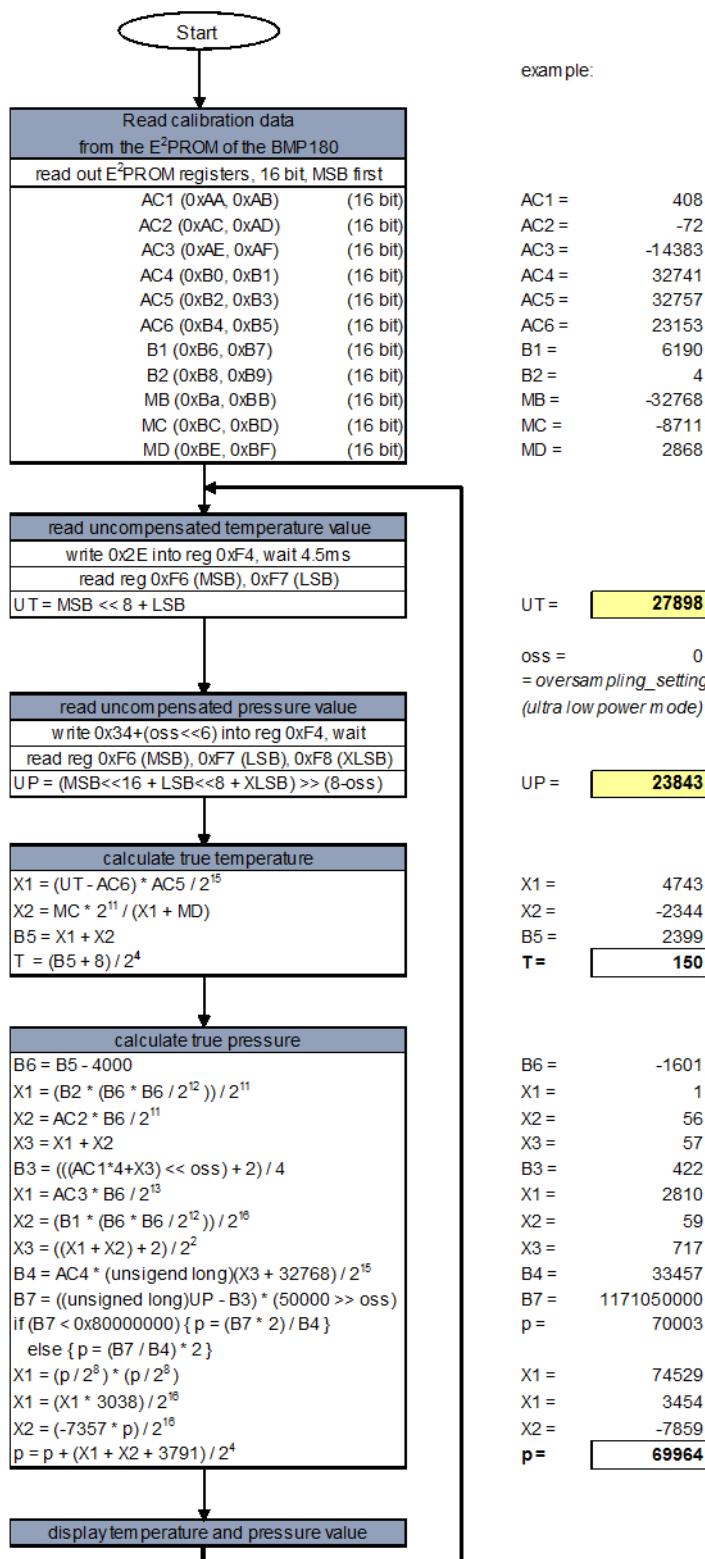
Calculation of true temperature and pressure in steps of 1Pa (= 0.01hPa = 0.01mbar) and temperature in steps of 0.1°C.

The following figure shows the detailed algorithm for pressure and temperature measurement.

This algorithm is available to customers as reference C source code ("BMP180_API") from Bosch Sensortec and via its sales and distribution partners. **Please contact your Bosch Sensortec representative for details.**



Calculation of pressure and temperature for BMP180



example:

C code function: type:

bmp180_get_cal_param

| | | |
|-------|--------|----------------|
| AC1 = | 408 | short |
| AC2 = | -72 | short |
| AC3 = | -14383 | short |
| AC4 = | 32741 | unsigned short |
| AC5 = | 32757 | unsigned short |
| AC6 = | 23153 | unsigned short |
| B1 = | 6190 | short |
| B2 = | 4 | short |
| MB = | -32768 | short |
| MC = | -8711 | short |
| MD = | 2868 | short |

bmp180_get_ut

UT = 27898 long

OSS = 0 short (0 .. 3)
= oversampling_setting (ultra low power mode)

bmp180_get_up

UP = 23843 long

X1 = 4743 long
X2 = -2344 long
B5 = 2399 long
T = 150 temp in 0.1°C long

bmp180_get_temperature

B6 = -1601 long
X1 = 1 long
X2 = 56 long
X3 = 57 long
B3 = 422 long
X1 = 2810 long
X2 = 59 long
X3 = 717 long
B4 = 33457 unsigned long
B7 = 1171050000 unsigned long
P = 70003 long

X1 = 74529 long
X1 = 3454 long
X2 = -7859 long
P = 69964 press. in Pa long

Figure 4: Algorithm for pressure and temperature measurement

3.6 Calculating absolute altitude

With the measured pressure p and the pressure at sea level p_0 e.g. 1013.25hPa, the altitude in meters can be calculated with the international barometric formula:

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5.255}} \right)$$

Thus, a pressure change of $\Delta p = 1\text{hPa}$ corresponds to 8.43m at sea level.

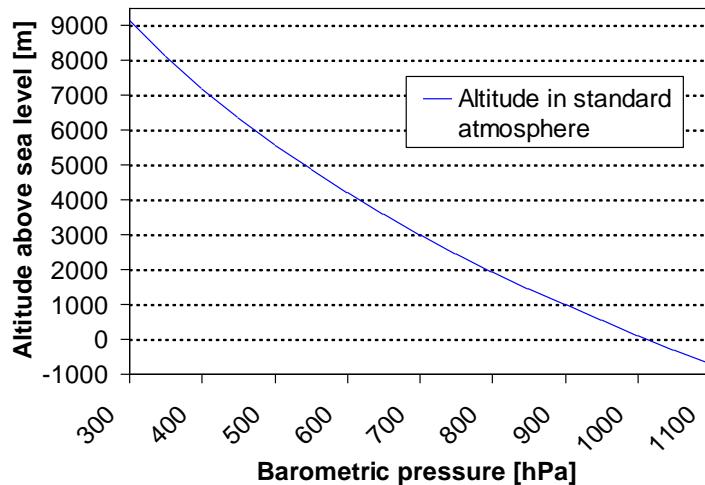


Figure 5: Transfer function: Altitude over sea level – Barometric pressure



3.7 Calculating pressure at sea level

With the measured pressure p and the absolute altitude the pressure at sea level can be calculated:

$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

Thus, a difference in altitude of $\Delta\text{altitude} = 10\text{m}$ corresponds to 1.2hPa pressure change at sea level.



4. Global Memory Map

The memory map below shows all externally accessible data registers which are needed to operate BMP180. The left columns show the memory addresses. The columns in the middle depict the content of each register bit. The colors of the bits indicate whether they are read-only, write-only or read- and writable. The memory is volatile so that the writable content has to be re-written after each power-on.

Not all register addresses are shown. These registers are reserved for further Bosch factory testing and trimming.

| Register Name | Register Adress | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Reset state |
|-----------------------|-----------------|---------------------------------|------|------|---------------------|------|------|------|------|-------------|
| out_xlsb | F8h | adc_out_xlsb<7:3> | | | | | 0 | 0 | 0 | 00h |
| out_lsb | F7h | adc_out_lsb<7:0> | | | | | | | | |
| out_msb | F6h | adc_out_msb<7:0> | | | | | | | | |
| ctrl_meas | F4h | oss<1:0> | | sco | measurement control | | | | | |
| soft reset | E0h | reset | | | | | | | | |
| id | D0h | id<7:0> | | | | | | | | |
| calib21 downto calib0 | BFh downto AAh | calib21<7:0> downto calib0<7:0> | | | | | | | | |

Registers: Control registers | Calibration registers | Data registers | Fixed
Type: read / write | read only | read only | read only

Figure 6: Memory map

Measurement control (register F4h <4:0>): Controls measurements. Refer to Figure 6 for usage details.

Sco (register F4h <5>): Start of conversion. The value of this bit stays “1” during conversion and is reset to “0” after conversion is complete (data registers are filled).

Oss (register F4h <7:6>): controls the oversampling ratio of the pressure measurement (00b: single, 01b: 2 times, 10b: 4 times, 11b: 8 times).

Soft reset (register E0h): Write only register. If set to 0xB6, will perform the same sequence as power on reset.

Chip-id (register D0h): This value is fixed to 0x55 and can be used to check whether communication is functioning.

After conversion, data registers can be read out in any sequence (i.e. MSB first or LSB first). Using a burst read is not mandatory.



5. I²C Interface

- I²C is a digital two wire interface
- Clock frequencies up to 3.4Mbit/sec. (I²C standard, fast and high-speed mode supported)
- SCL and SDA needs a pull-up resistor, typ. 4.7kOhm to V_{DDIO}
(one resistor each for all the I²C bus)

The I²C bus is used to control the sensor, to read calibration data from the E²PROM and to read the measurement data when A/D conversion is finished. SDA (serial data) and SCL (serial clock) have open-drain outputs.

For detailed I²C-bus specification please refer to:
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

5.1 I²C specification

Table 6: Electrical parameters for the I²C interface

| Parameter | Symbol | Min. | Typ | Max. | Units |
|--|-----------------------|-------------------------|-----|-------------------------|-------|
| Clock input frequency | f _{SCL} | | | 3.4 | MHz |
| Input-low level | V _{IL} | 0 | | 0.2 * V _{DDIO} | V |
| Input-high level | V _{IH} | 0.8 * V _{DDIO} | | V _{DDIO} | V |
| Voltage output low level @ V _{DDIO} = 1.62V, I _{OL} = 3mA | V _{OL} | | | 0.3 | V |
| SDA and SCL pull-up resistor | R _{pull-up} | 2.2 | | 10 | kOhm |
| SDA sink current @ V _{DDIO} = 1.62V, V _{OL} = 0.3V | I _{SDA_sink} | | 9 | | mA |
| Start-up time after power-up, before first communication | t _{Start} | 10 | | | Ms |



5.2 Device and register address

The BMP180 module address is shown below. The LSB of the device address distinguishes between read (1) and write (0) operation, corresponding to address 0xEF (read) and 0xEE (write).

Table 7: BMP180 addresses

| A7 | A6 | A5 | A4 | A3 | A2 | A1 | W/R |
|----|----|----|----|----|----|----|-----|
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0/1 |

5.3 I²C protocol

The I²C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.

At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.

Even though V_{DDIO} can be powered on before V_{DD} , there is a chance of excessive power consumption (a few mA) if this sequence is used, and the state of the output pins is undefined so that the bus can be locked. Therefore, V_{DD} *must* be powered before V_{DDIO} unless the limitations above are understood and not critical.

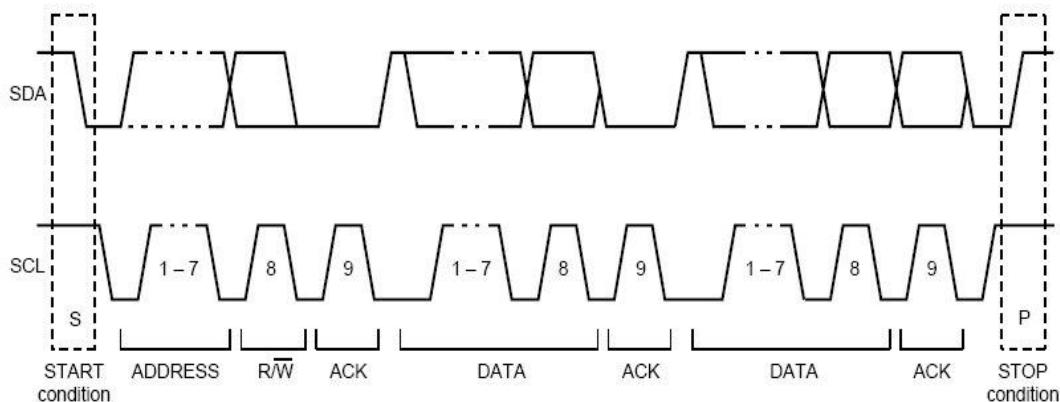


Figure 7: I²C protocol



5.4 Start temperature and pressure measurement

The timing diagrams to start the measurement of the temperature value UT and pressure value UP are shown below. After start condition the master sends the device address write, the register address and the control register data. The BMP180 sends an acknowledgement (ACKS) every 8 data bits when data is received. The master sends a stop condition after the last ACKS.

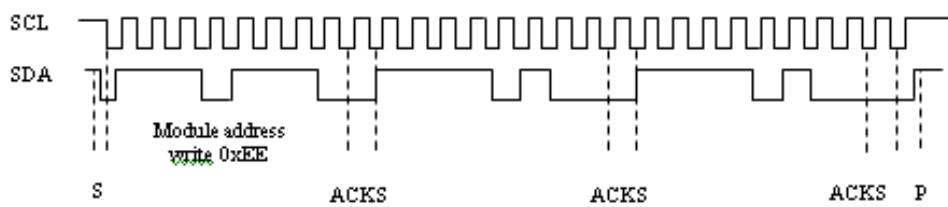


Figure 8: Timing diagram for starting pressure measurement

Abbreviations:

| | |
|-------|---------------------------|
| S | Start |
| P | Stop |
| ACKS | Acknowledge by Slave |
| ACKM | Acknowledge by Master |
| NACKM | Not Acknowledge by Master |

Table 8: Control registers values for different internal oversampling_setting (oss)

| Measurement | Control register value (register address 0xF4) | Max. conversion time [ms] |
|-----------------------|---|------------------------------|
| Temperature | 0x2E | 4.5 |
| Pressure (oss = 0) | 0x34 | 4.5 |
| Pressure (oss = 1) | 0x74 | 7.5 |
| Pressure (oss = 2) | 0xB4 | 13.5 |
| Pressure (oss = 3) | 0xF4 | 25.5 |



5.5 Read A/D conversion result or E²PROM data

To read out the temperature data word UT (16 bit), the pressure data word UP (16 to 19 bit) and the E²PROM data proceed as follows:

After the start condition the master sends the module address write command and register address. The register address selects the read register:

E²PROM data registers 0xAA to 0xBF

Temperature or pressure value UT or UP 0xF6 (MSB), 0xF7 (LSB), optionally 0xF8 (XLSB)

Then the master sends a restart condition followed by the module address read that will be acknowledged by the BMP180 (ACKS). The BMP180 sends first the 8 MSB, acknowledged by the master (ACKM), then the 8 LSB. The master sends a "not acknowledge" (NACKM) and finally a stop condition.

Optionally for ultra high resolution, the XLSB register with address 0xF8 can be read to extend the 16 bit word to up to 19 bits; refer to the application programming interface (API) software rev. 1.1 ("BMP180_API", available from Bosch Sensortec).

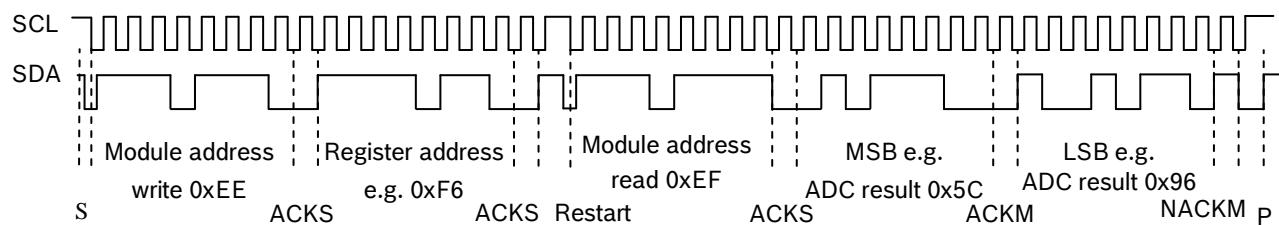


Figure 9: Timing diagram read 16 bit A/D conversion result



6. Package

6.1 Pin configuration

Picture shows the device in top view. Device pins are shown here transparently only for orientation purposes.

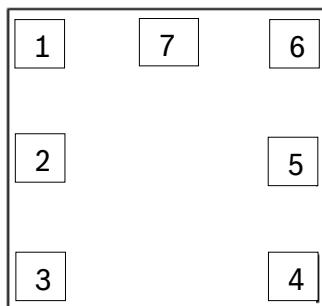


Figure 10: Layout pin configuration BMP180

Table 9: Pin configuration BMP180

| in No | Name | Function |
|-------|-------|------------------------------------|
| 1 | CSB* | Chip select |
| 2 | VDD | Power supply |
| 3 | VDDIO | Digital power supply |
| 4 | SDO* | SPI output |
| 5 | SCL | I2C serial bus clock input |
| 6 | SDA | I2C serial bus data (or SPI input) |
| 7 | GND | Ground |

* A pin compatible product variant with SPI interface is possible upon customer's request. For I²C (standard case) CSB and SDO are not used, they have to be left open.

All pins have to be soldered to the PCB for symmetrical stress input even though they are not connected internally.

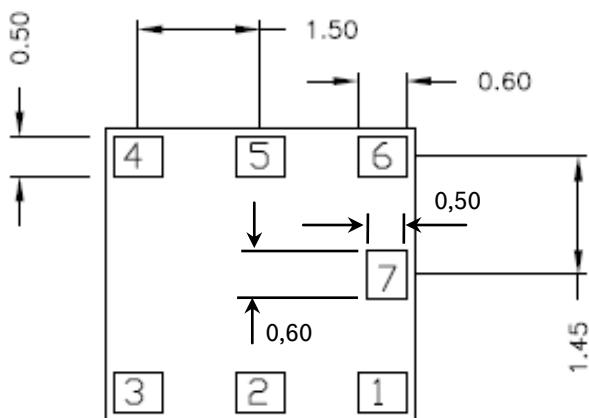


6.2 Outline dimensions

The sensor housing is a 7Pin LGA package with metal lid. Its dimensions are 3.60mm (± 0.1 mm) x 3.80mm (± 0.1 mm) x 0.93mm (± 0.07 mm).

Note: All dimensions are in mm.

6.2.1 Bottom view



BOTTOM VIEW

Figure 11: Bottom view BMP180



6.2.2 Top view

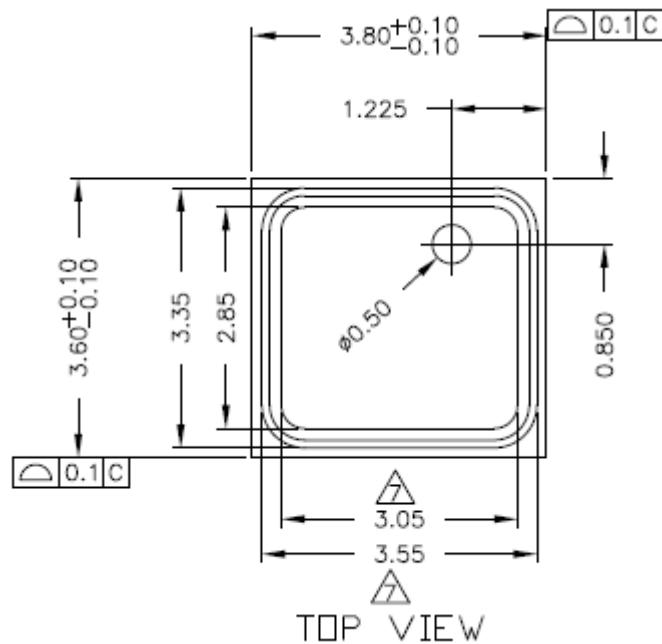


Figure 12: Top view BMP180

6.2.3 Side view

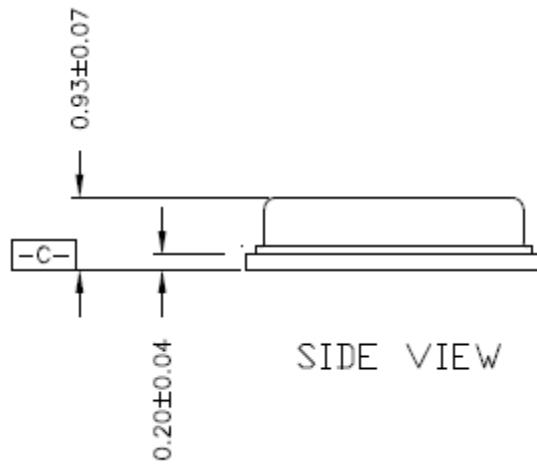


Figure 13: Side view BMP180



6.3 Moisture sensitivity level and soldering

The BMP180 is classified MSL 1 (moisture sensitivity level) according to IPC/JEDEC standards J-STD-020D and J-STD-033A.

The device can be soldered Pb-free with a peak temperature of 260°C for 20 to 40 sec. The minimum height of the solder after reflow shall be at least 50µm. This is required for good mechanical decoupling between the sensor device and the printed circuit board (PCB).

To ensure good solder-ability, the devices shall be stored at room temperature (20°C).

The soldering process can lead to an offset shift.

6.4 RoHS compliancy

The BMP180 sensor meets the requirements of the EC directive "Restriction of hazardous substances (RoHS)", please refer also to:

"Directive 2002/95/EC of the European Parliament and of the Council of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment".

The BMP180 sensor is also halogen-free.

6.5 Mounting and assembly recommendations

In order to achieve the specified performance for your design, the following recommendations and the "Handling, soldering & mounting instructions BMP180" should be taken into consideration when mounting a pressure sensor on a printed-circuit board (PCB):

- The clearance above the metal lid shall be 0.1mm at minimum.
- For the device housing appropriate venting needs to be provided in case the ambient pressure shall be measured.
- Liquids shall not come into direct contact with the device.
- During operation the sensor is sensitive to light, which can influence the accuracy of the measurement (photo-current of silicon).
- The BMP180 shall not be placed close to the fast heating parts. In case of gradients > 3°C/sec. it is recommended to follow Bosch Sensortec application note ANP015, "Correction of errors induced by fast temperature changes". Please contact your Bosch Sensortec representative for details.



7. Legal disclaimer

7.1 Engineering samples

Engineering Samples are marked with an asterisk (*) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.

7.2 Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or security sensitive systems. Security sensitive systems are those for which a malfunction is expected to lead to bodily harm or significant property damage. In addition, they are not fit for use in products which interact with motor vehicle systems.

The resale and/or use of products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the Purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.

The purchaser must monitor the market for the purchased products, particularly with regard to product safety, and inform Bosch Sensortec without delay of all security relevant incidents.

7.3 Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.



8. Document history and modification

| Rev. No | Chapter | Description of modifications/changes | Date |
|---------|---------------------------------|--|-------------------|
| 1.0 | | First edition for description of serial production material – Preliminary version | |
| 1.1 | 5.1 | New nomenclature of pin configuration | 27 July 2010 |
| 1.2 | 5 | Design change in package – hole in Lid and without slit | 13 September 2010 |
| 1.3 | 3.2 5.1 | - Standardizing pin naming over Bosch Sensortec products – typical application circuit - Optimizing pin description, SPI description | 15 December 2010 |
| 2.0 | 1 | - Non-preliminary version - Verifying parameter through characterization | 28 January 2011 |
| 2.1 | 3.2 4 5.3 6.1 6.2.1 | - Declaration of SDO and CSB pins in the typical application circuit - Adding global memory map and bits description - Power-up sequence - Description of used interfaces - Dimension pin7 | 1 April 2011 |
| 2.2 | 6.1 | Correction of the pin configuration (editorial change) | 14 April 2011 |
| 2.3 | 3.3 | Optimizing noise performance | 25 May 2011 |
| 2.4 | 6.3 | Removed shelf-life constraints | 26 January 2012 |
| | page 2 | Comparison removed | |
| | 5.1 | Voltage output low level added | |
| | 5.3 | Power on sequence of V_{DD} and V_{DDIO} defined | |
| 2.5 | 1 | Added max values for supply current for restricted version | 15 Feb 2013 |
| | 1 | Added max value for standby current for restricted version | 5 Apr 2013 |
| | Figure 4 | Update of calculation of algorithm for pressure and temperature measurement | |
| | Page 2 | Changed wording from “ultra high resolution mode” to “advanced resolution mode” | |

Bosch Sensortec GmbH
Gerhard-Kindler-Strasse 8
72770 Reutlingen / Germany

contact@bosch-sensortec.com
www.bosch-sensortec.com

Modifications reserved | Printed in Germany
Specifications are subject to change without notice
Revision_2.5_042013
Document number: BST-BMP180-DS000-09



FOTORRESISTENCIA LDR 4,3mm x Ø 5,1mm

C-2795

?? Los nombres registrados y marcas que se citan son propiedad de sus respectivos titulares.

DESCRIPCION GENERAL.

Fotorresistencia o resistencia dependiente de la luz, consistente en una célula de Sulfuro de Cadmio, altamente estable, encapsulada con una resina epoxi transparente, resistente a la humedad. La respuesta espectral es similar a la del ojo humano. Su nivel de resistencia aumenta cuando el nivel de luz disminuye.

Aplicaciones: Control de contraste en televisores y monitores, control automático de la iluminación en habitaciones, juguetes y juegos electrónicos, controles industriales, interruptores crepusculares, boyas y balizas de encendido automático, auto-flash, etc...

CARACTERÍSTICAS TÉCNICAS.

| Modelo | Valores máximos | | | Características a 25°C (nota E) | | | | | |
|---------------|-------------------------|----------------------------|------------------------------|---------------------------------|---------------------------|--------------------------------|---|---------------------------------|-----------|
| | Tensión a 25°C (Vdc) | Potencia disipable (mW) | Temperatura ambiente (°C) | Resistencia (notaA) | | ? (notaC) Min.(k?) Max.(k?) | Tiempos de respuesta a 10 lx (notaD) t. subida (ms) t. bajada (ms) | Respuesta espectral (pico) (nm) | |
| | | | | 10 lux (2856K) Min.(k?) | 0 lux (notaB) Max.(M?) | | | | |
| C-2795 | 150 | 90 | -25 a 75 | 50 | 140 | 20 | 0.9 | 60 | 25 |

Notas: A) Medido con una fuente luminosa formada por una lámpara de tungsteno, trabajando a una temperatura de color de 2856K.

B)Medición efectuada 10 segundos después de retirar una iluminación incidente de 10 lux.

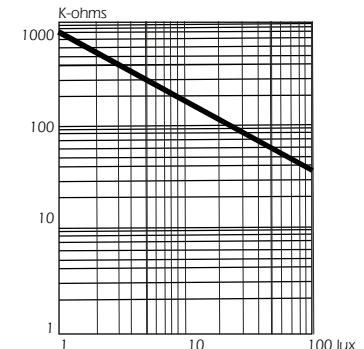
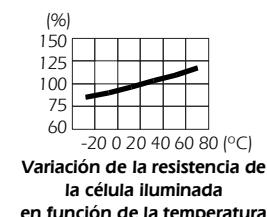
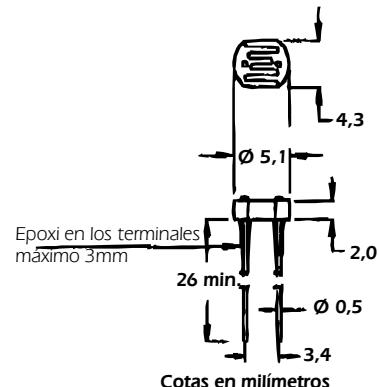
C) Sensibilidad entre 10 y 100 lux, dada por:

$$\begin{aligned} ? &= \log(R100) - \log(R10) \\ &= \log(E100) - \log(E10) \end{aligned}$$

donde R100, R10 son las resistencias a 100 y 10 lux respectivamente, y E100, E10 las iluminancias de 100 y 10 lx respectivamente.

D) Tiempo de subida es el tiempo necesario para alcanzar el 63% del nivel de saturación. Tiempo de bajada es el necesario para que la célula alcance el 37% desde el nivel saturación.

E) Todas las características están medidas con la célula LDR expuesta a la luz (100-500 lux) durante 1 o 2 horas.



CONSIDERACIONES.

Este componente está destinado para su uso por parte de profesionales, o usuarios con un nivel técnico o conocimientos suficientes, que les permita desarrollar por sí mismos los proyectos o aplicaciones deseados. Por este motivo no se facilitará asistencia técnica sobre problemas de implementación del citado componente en las aplicaciones en las que sea empleado.

Para cualquier problema relativo al funcionamiento del producto (excluidos los problemas de aplicación), póngase en contacto con nuestro **departamento técnico**. **Fax 93 432 29 95**.

Correo electrónico: sat@fadisel.com. La documentación técnica de este producto responde a una transcripción de la proporcionada por el fabricante.

Los productos de la familia "Componentes" de Cebek disponen de **1 año de garantía** a partir de la fecha de compra. Quedan excluidos el trato o manipulación incorrectos.

Disponemos de más productos que pueden interesarle, visítenos en: www.fadisel.com ó **SOLICITE GRATUITAMENTE nuestro catálogo**.

Microphone Amplifier with AGC and Low-Noise Microphone Bias

MAX9814

General Description

The MAX9814 is a low-cost, high-quality microphone amplifier with automatic gain control (AGC) and low-noise microphone bias. The device features a low-noise preamplifier, variable gain amplifier (VGA), output amplifier, microphone-bias-voltage generator and AGC control circuitry.

The low-noise preamplifier has a fixed 12dB gain, while the VGA gain automatically adjusts from 20dB to 0dB, depending on the output voltage and the AGC threshold. The output amplifier offers selectable gains of 8dB, 18dB, and 28dB. With no compression, the cascade of the amplifiers results in an overall gain of 40dB, 50dB, or 60dB. A trilevel digital input programs the output amplifier gain. An external resistive divider controls the AGC threshold and a single capacitor programs the attack/release times. A trilevel digital input programs the ratio of attack-to-release time. The hold time of the AGC is fixed at 30ms. The low-noise microphone-bias-voltage generator can bias most electret microphones.

The MAX9814 is available in the space-saving 12-bump UCSP™ (1.5mm x 2mm) and 14-pin TDFN packages. This device is specified over the -40°C to +85°C extended temperature range.

Applications

| | |
|---------------------------------------|----------------------------------|
| Digital Still Cameras | Two-Way Communicators |
| Digital Video Cameras | High-Quality Portable Recorders |
| PDAs | IP Phones/Telephone Conferencing |
| Bluetooth Headsets | |
| Entertainment Systems (e.g., Karaoke) | |

Features

- ◆ Automatic Gain Control (AGC)
- ◆ Three Gain Settings (40dB, 50dB, 60dB)
- ◆ Programmable Attack Time
- ◆ Programmable Attack and Release Ratio
- ◆ 2.7V to 5.5V Supply Voltage Range
- ◆ Low Input-Referred Noise Density of 30nV/ $\sqrt{\text{Hz}}$
- ◆ Low THD: 0.04% (typ)
- ◆ Low-Power Shutdown Mode
- ◆ Internal Low-Noise Microphone Bias, 2V
- ◆ Available in the Space-Saving 12-Bump UCSP (1.5mm x 2mm) and 14-Pin TDFN (3mm x 3mm) Packages
- ◆ -40°C to +85°C Extended Temperature Range

Ordering Information

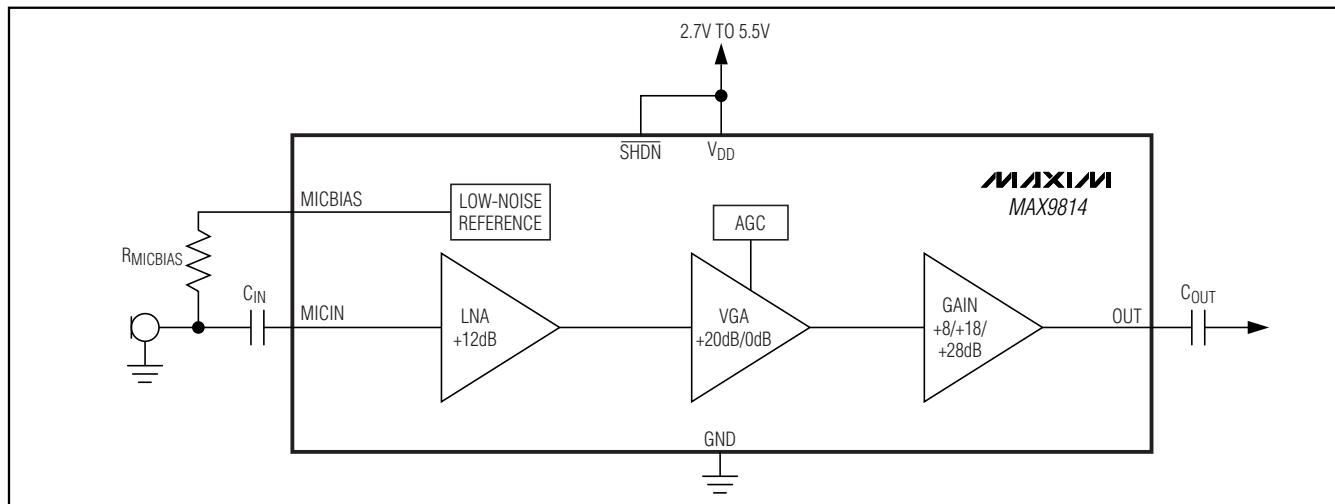
| PART | TEMP RANGE | PIN-PACKAGE | PKG CODE |
|--------------|----------------|-------------|----------|
| MAX9814EBC+T | -40°C to +85°C | 12 UCSP-12 | B12-3 |
| MAX9814ETD+T | -40°C to +85°C | 14 TDFN-14 | T1433-2 |

+Denotes a lead-free package.

Pin Configurations appear at end of data sheet.

UCSP is a trademark of Maxim Integrated Products, Inc.

Simplified Block Diagram



Microphone Amplifier with AGC and Low-Noise Microphone Bias

ABSOLUTE MAXIMUM RATINGS

| | |
|-----------------------------------|-----------------------------------|
| V _{DD} to GND | -0.3V to +6V |
| All Other Pins to GND | -0.3V to (V _{DD} + 0.3V) |
| Output Short-Circuit Duration | Continuous |
| Continuous Current (OUT, MICBIAS) | ±100mA |
| All Other Pins | ±20mA |

| | |
|---|----------------|
| Continuous Power Dissipation (T _A = +70°C) | |
| 12-Bump UCSP (derate 6.5mW/°C above +70°C) | 518mW |
| 14-Pin TDFN (derate 16.7mW/°C above +70°C) | 1481.5mW |
| Operating Temperature Range | -40°C to +85°C |
| Junction Temperature | +150°C |
| Lead Temperature (soldering, 10s) | +300°C |
| Bump Temperature (soldering) Reflow | +235°C |

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V_{DD} = 3.3V, $\overline{SHDN} = V_{DD}$, C_{CCT} = 470nF, C_{CG} = 2μF, GAIN = V_{DD}, T_A = T_{MIN} to T_{MAX}, unless otherwise specified. Typical values are at T_A = +25°C.) (Note 1)

| PARAMETER | SYMBOL | CONDITIONS | MIN | TYP | MAX | UNITS |
|--------------------------------------|----------------------|---|--------|------|------|-------------------|
| GENERAL | | | | | | |
| Operating Voltage | V _{DD} | Guaranteed by PSRR test | 2.7 | 5.5 | | V |
| Supply Current | I _{DD} | | 3.1 | 6 | | mA |
| Shutdown Supply Current | I _{SHDN} | | 0.01 | 1 | | μA |
| Input-Referred Noise Density | e _n | BW = 20kHz, all gain settings | 30 | | | nV/√Hz |
| Output Noise | | BW = 20kHz | 430 | | | μVRMS |
| Signal-to-Noise Ratio | SNR | BW = 22Hz to 22kHz (500mVRMS output signal) | 61 | | | dB |
| | | A-weighted | 64 | | | |
| Dynamic Range | DR | (Note 2) | 60 | | | dB |
| Total Harmonic Distortion Plus Noise | THD+N | f _{IN} = 1kHz, BW = 20Hz to 20kHz, R _L = 10kΩ, V _{TH} = 1V (threshold = 2V _{P-P}), V _{IN} = 0.5mVRMS, V _{CT} = 0V | 0.04 | | | % |
| | | f _{IN} = 1kHz, BW = 20Hz to 20kHz, R _L = 10kΩ, V _{TH} = 0.1V (threshold = 200mV _{P-P}), V _{IN} = 30mVRMS, V _{CT} = 2V | 0.2 | | | |
| Amplifier Input BIAS | V _{IN} | | 1.14 | 1.23 | 1.32 | V |
| Maximum Input Voltage | V _{IN_MAX} | 1% THD | 100 | | | mV _{P-P} |
| Input Impedance | Z _{IN} | | 100 | | | kΩ |
| Maximum Gain | A | GAIN = V _{DD} | 39.5 | 40 | 40.5 | dB |
| | | GAIN = GND | 49.5 | 50 | 50.6 | |
| | | GAIN = unconnected | 59.5 | 60 | 60.5 | |
| Minimum Gain | | GAIN = V _{DD} | 18.7 | 20 | 20.5 | dB |
| | | GAIN = GND | 29.0 | 30 | 30.8 | |
| | | GAIN = unconnected | 38.7 | 40 | 40.5 | |
| Maximum Output Level | V _{OUT_RMS} | 1% THD+N, V _{TH} = MICBIAS | 0.707 | | | VRMS |
| Regulated Output Level | | AGC enabled, V _{TH} = 0.7V | 1.26 | 1.40 | 1.54 | V _{P-P} |
| AGC Attack Time | t _{ATTACK} | C _T = 470nF (Note 3) | 1.1 | | | ms |
| Attack/Release Ratio | A/R | A/R = GND | 1:500 | | | ms/ms |
| | | A/R = V _{DD} | 1:2000 | | | |
| | | A/R = unconnected | 1:4000 | | | |

Microphone Amplifier with AGC and Low-Noise Microphone Bias

ELECTRICAL CHARACTERISTICS (continued)

($V_{DD} = 3.3V$, $\bar{SHDN} = V_{DD}$, $C_{CT} = 470nF$, $C_{CG} = 2\mu F$, GAIN = V_{DD} , $T_A = T_{MIN}$ to T_{MAX} , unless otherwise specified. Typical values are at $T_A = +25^\circ C$.) (Note 1)

| PARAMETER | SYMBOL | CONDITIONS | MIN | TYP | MAX | UNITS |
|------------------------------------|----------------------|--|---------------------------------|---------------------------------|--------------------------------|---------------|
| MICOUT High Output Voltage | V_{OH} | I_{OUT} sourcing 1mA | | 2.45 | | V |
| MICOUT Low Output Voltage | V_{OL} | I_{OUT} sinking 1mA | | 3 | | mV |
| MICOUT Bias | | MICOUT unconnected | 1.14 | 1.23 | 1.32 | V |
| Output Impedance | Z_{OUT} | | | 50 | | Ω |
| Minimum Resistive Load | R_{LOAD_MIN} | | | 5 | | $k\Omega$ |
| Maximum Capacitive Drive | C_{LOAD_MAX} | | | 200 | | pF |
| Maximum Output Current | I_{OUT_MAX} | 1% THD, $R_L = 500\Omega$ | 1 | 2 | | mA |
| Output Short-Circuit Current | I_{SC} | | 3 | 8 | | mA |
| Power-Supply Rejection Ratio | PSRR | AGC mode; $V_{DD} = 2.7V$ to $5.5V$ (Note 4) | 35 | 50 | | dB |
| | | $f = 217Hz$, $V_{RIPPLE} = 100mV_{P-P}$ (Note 5) | | 55 | | |
| | | $f = 1kHz$, $V_{RIPPLE} = 100mV_{P-P}$ (Note 5) | | 52.5 | | |
| | | $f = 10kHz$, $V_{RIPPLE} = 100mV_{P-P}$ (Note 5) | | 43 | | |
| MICROPHONE BIAS | | | | | | |
| Microphone Bias Voltage | $V_{MICBIAS}$ | $I_{MICBIAS} = 0.5mA$ | 1.84 | 2.0 | 2.18 | V |
| Output Resistance | $R_{MICBIAS}$ | $I_{MICBIAS} = 1mA$ | | 1 | | Ω |
| Output Noise Voltage | $V_{MICBIAS_NOISE}$ | $I_{MICBIAS} = 0.5mA$, BW = 22Hz to 22kHz | | 5.5 | | μV_{RMS} |
| Power-Supply Rejection Ratio | PSRR | DC, $V_{DD} = 2.7V$ to $5.5V$ | 70 | 80 | | dB |
| | | $I_{MICBIAS} = 0.5mA$, $V_{RIPPLE} = 100mV_{P-P}$, $f_{IN} = 1kHz$ | | 71 | | |
| TRILEVEL INPUTS (A/R, GAIN) | | | | | | |
| Tri-Level Input Leakage Current | | A/R or GAIN = V_{DD} | 0.5 V_{DD} / 180k Ω | 0.5 V_{DD} / 100k Ω | 0.5 V_{DD} / 50k Ω | mA |
| | | A/R or GAIN = GND | 0.5 V_{DD} / 180k Ω | 0.5 V_{DD} / 100k Ω | 0.5 V_{DD} / 50k Ω | |
| Input High Voltage | V_{IH} | | $V_{DD} \times 0.7$ | | | V |
| Input Low Voltage | V_{IL} | | $V_{DD} \times 0.3$ | | | V |
| Shutdown Enable Time | t_{ON} | | | 60 | | ms |
| Shutdown Disable Time | t_{OFF} | | | 40 | | ms |
| DIGITAL INPUT (SHDN) | | | | | | |
| SHDN Input Leakage Current | | | -1 | | +1 | μA |
| Input High Voltage | V_{IH} | | 1.3 | | | V |
| Input Low Voltage | V_{IL} | | | 0.5 | | V |
| AGC THRESHOLD INPUT (TH) | | | | | | |
| TH Input Leakage Current | | | -1 | | +1 | μA |

Note 1: Devices are production tested at $T_A = +25^\circ C$. Limits over temperature are guaranteed by design.

Note 2: Dynamic range is calculated using the EIAJ method. The input is applied at -60dBFS (0.707 μV_{RMS}), $f_{IN} = 1kHz$.

Note 3: Attack time measured as time from AGC trigger to gain reaching 90% of its final value.

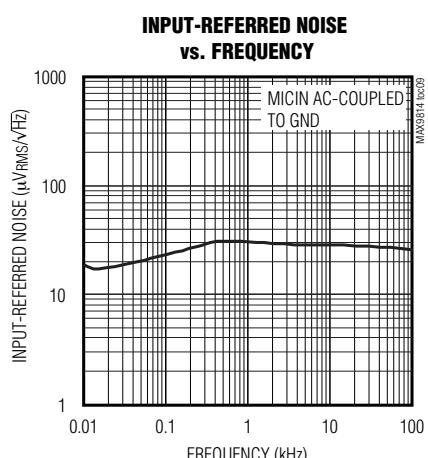
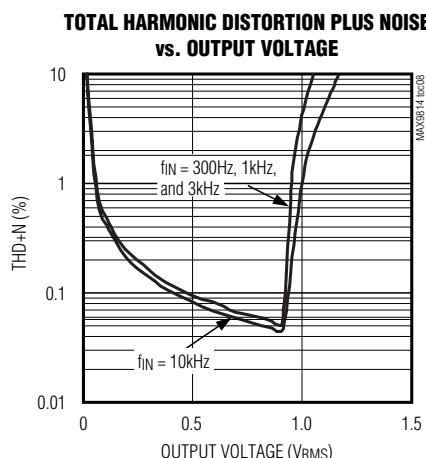
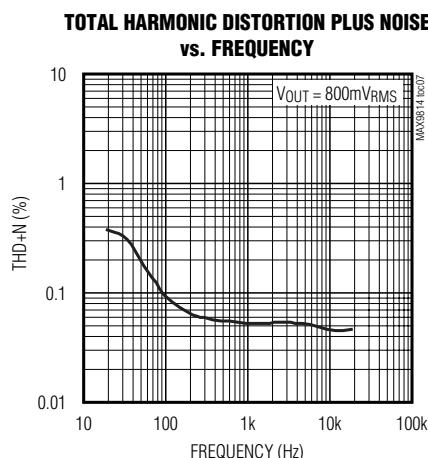
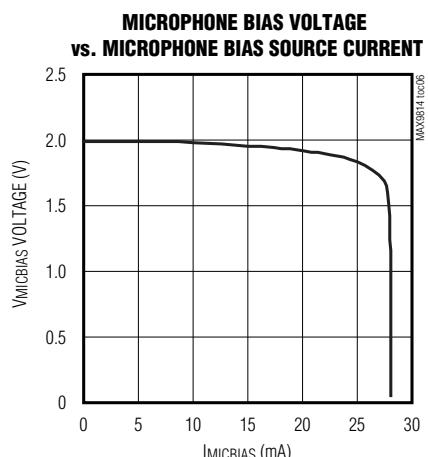
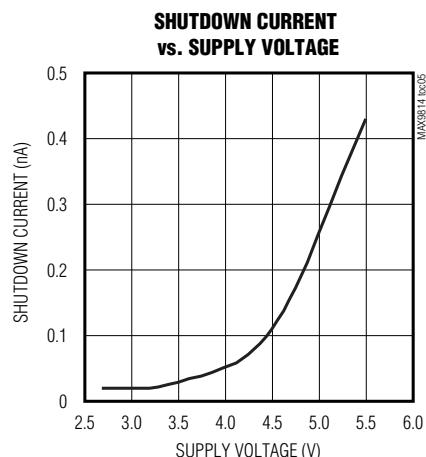
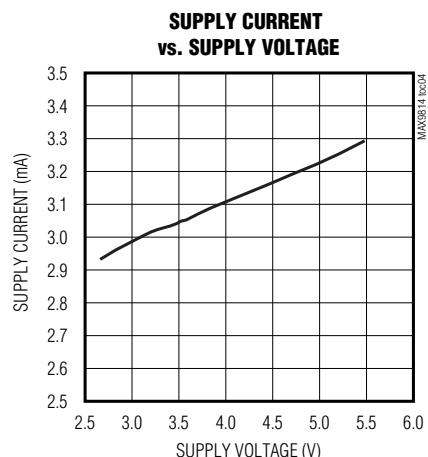
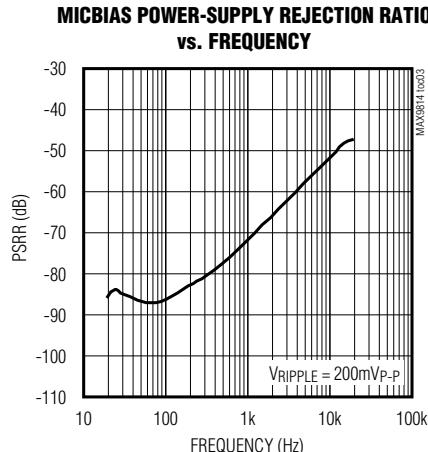
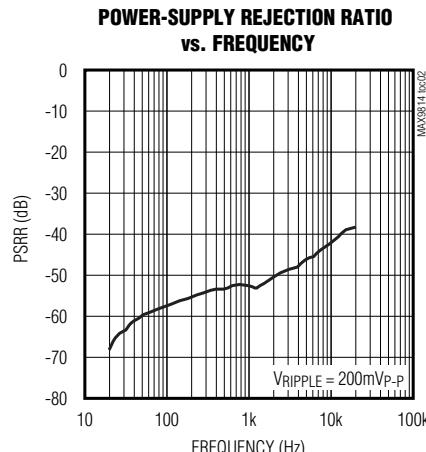
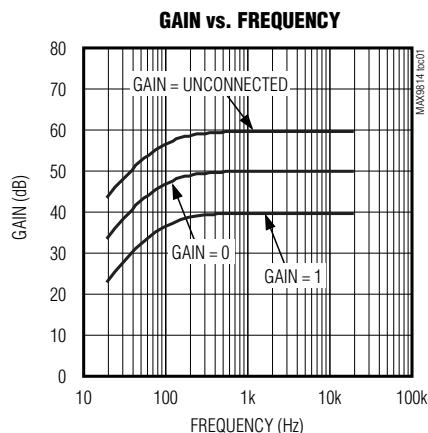
Note 4: CG is connected to an external DC voltage source, and adjusted until $V_{MICOUT} = 1.23V$.

Note 5: CG connected to GND with $2.2\mu F$.

Microphone Amplifier with AGC and Low-Noise Microphone Bias

Typical Operating Characteristics

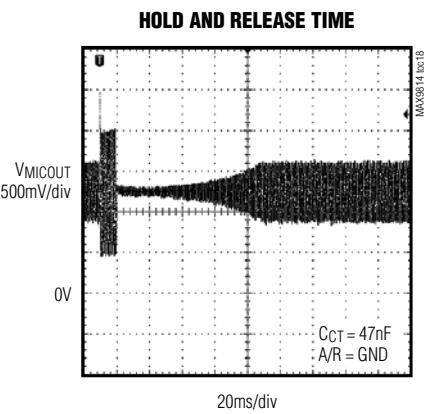
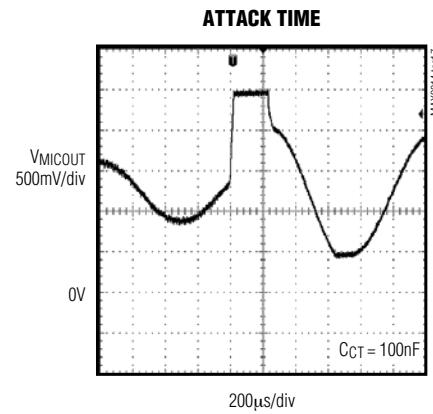
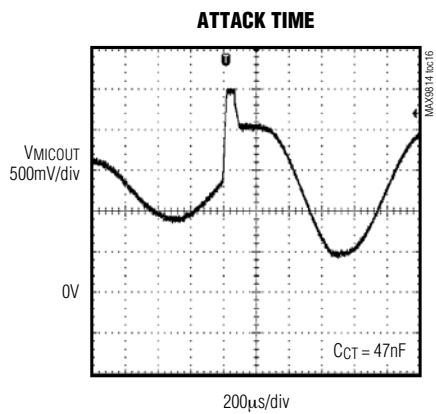
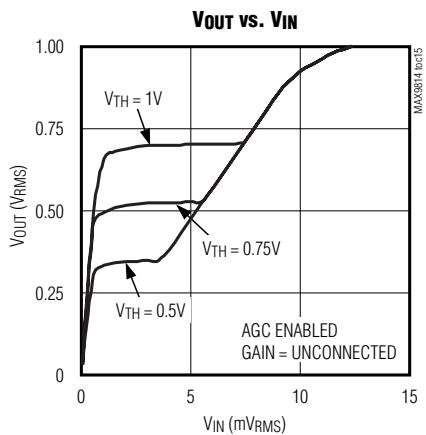
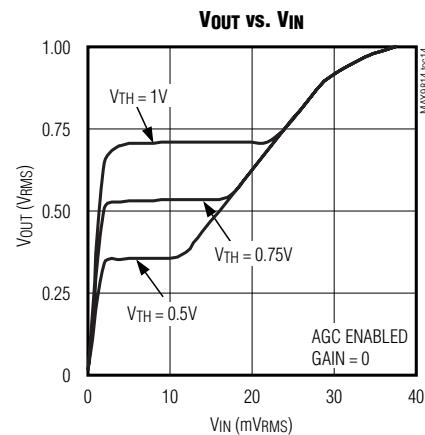
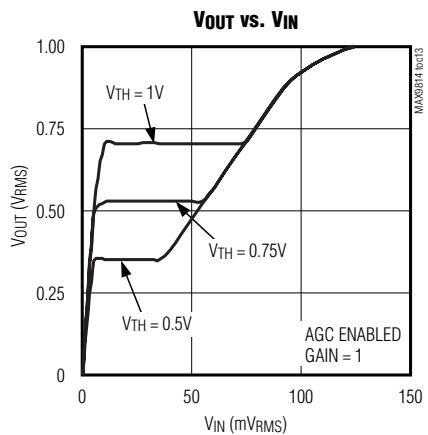
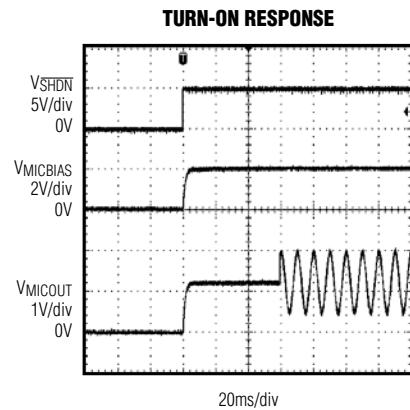
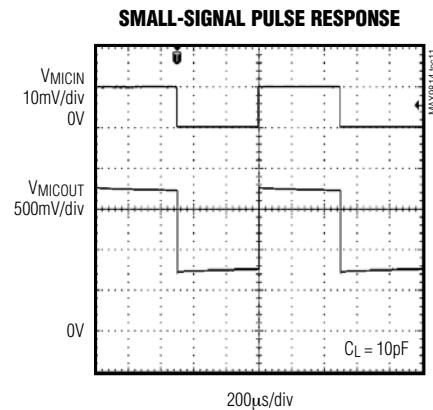
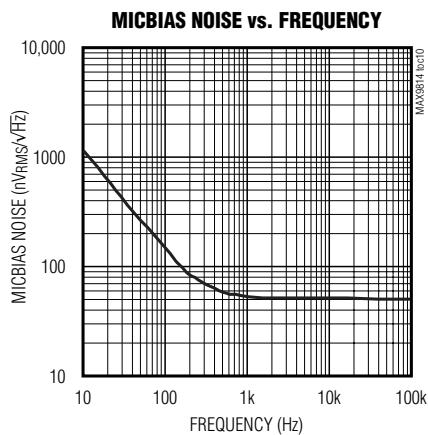
($V_{DD} = 5V$, $C_{CT} = 470nF$, $C_{CG} = 2.2\mu F$, $V_{TH} = V_{MICBIAS} \times 0.4$, GAIN = V_{DD} (40dB), AGC disabled, no load, $R_L = 10k\Omega$, $C_{OUT} = 1\mu F$, $T_A = +25^\circ C$, unless otherwise noted.)



Microphone Amplifier with AGC and Low-Noise Microphone Bias

Typical Operating Characteristics (continued)

($V_{DD} = 5V$, $C_{CT} = 470nF$, $C_{CG} = 2.2\mu F$, $V_{TH} = V_{MICBIAS} \times 0.4$, GAIN = V_{DD} (40dB), AGC disabled, no load, $R_L = 10k\Omega$, $C_{OUT} = 1\mu F$, $T_A = +25^\circ C$, unless otherwise noted.)

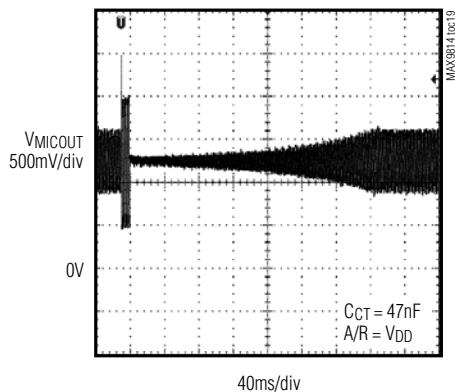


Microphone Amplifier with AGC and Low-Noise Microphone Bias

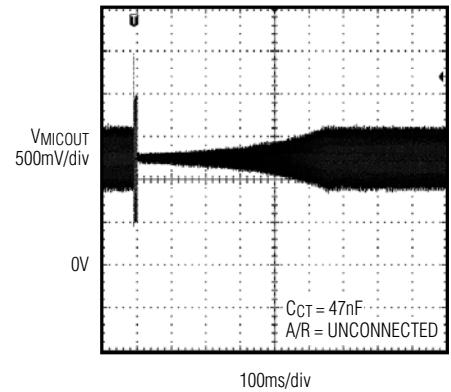
Typical Operating Characteristics (continued)

($V_{DD} = 5V$, $C_{CT} = 470nF$, $C_{CG} = 2.2\mu F$, $V_{TH} = V_{MICBIAS} \times 0.4$, GAIN = V_{DD} (40dB), AGC disabled, no load, $R_L = 10k\Omega$, $C_{OUT} = 1\mu F$, $T_A = +25^\circ C$, unless otherwise noted.)

HOLD AND RELEASE TIME



HOLD AND RELEASE TIME



Pin Description

| PIN | | NAME | FUNCTION |
|-------|------|----------|--|
| TDFN | UCSP | | |
| 1 | A1 | CT | Timing Capacitor Connection. Connect a capacitor to CT to control the Attack and Release times of the AGC. |
| 2 | B2 | SHDN | Active-Low Shutdown Control |
| 3 | A2 | CG | Amplifier DC Offset Adjust. Connect a $2.2\mu F$ capacitor to GND to ensure zero offset at the output. |
| 4, 11 | — | N.C. | No Connection. Connect to GND. |
| 5 | A3 | V_{DD} | Power Supply. Bypass to GND with a $1\mu F$ capacitor. |
| 6 | A4 | MICOUT | Amplifier Output |
| 7 | B4 | GND | Ground |
| 8 | C4 | MICIN | Microphone Noninverting Input |
| 9 | B3 | A/R | Tri-Level Attack and Release Ratio Select. Controls the ratio of attack time to release time for the AGC circuit. A/R = GND: Attack/Release Ratio is 1:500 A/R = V_{DD} : Attack/Release Ratio is 1:2000 A/R = BIAS: Attack/Release Ratio is 1:4000 |
| 10 | C3 | GAIN | Tri-Level Amplifier Gain Control. GAIN = V_{DD} , gain set to 40dB. GAIN = GND, gain set to 50dB. GAIN = Unconnected, uncompressed gain set to 60dB. |
| 12 | C2 | BIAS | Amplifier Bias. Bypass to GND with a $0.47\mu F$ capacitor. |
| 13 | C1 | MICBIAS | Microphone Bias Output |
| 14 | B1 | TH | AGC Threshold Control. TH voltage sets gain control threshold. Connect TH to MICBIAS to disable the AGC. |

Microphone Amplifier with AGC and Low-Noise Microphone Bias

Detailed Description

The MAX9814 is a low-cost, high-quality microphone amplifier with automatic gain control (AGC) and a low-noise microphone bias. The MAX9814 consists of several distinct circuits: a low-noise preamplifier, a variable gain amplifier (VGA), an output amplifier, a microphone-bias-voltage generator, and AGC control circuitry.

An internal microphone bias voltage generator provides a 2V bias that is suitable for most electret condenser microphones. The MAX9814 amplifies the input in three distinct stages. In the first stage, the input is buffered and amplified through the low-noise preamplifier with a gain of 12dB. The second stage consists of the VGA controlled by the AGC. The VGA/AGC combination is capable of varying the gain from 20dB to 0dB. The output amplifier is the final stage in which a fixed gain of 8dB, 18dB, 20dB is programmed through a single tri-level logic input. With no compression from the AGC, the MAX9814 is capable of providing 40dB, 50dB, or 60dB gain.

Automatic Gain Control (AGC)

A device without AGC experiences clipping at the output when too much gain is applied to the input. AGC prevents clipping at the output when too much gain is applied to the input, eliminating output clipping. Figure 1 shows a comparison of an over-gained microphone input with and without AGC.

The MAX9814's AGC controls the gain by first detecting that the output voltage has exceeded a preset limit. The microphone amplifier gain is then reduced with a selectable time constant to correct for the excessive output-voltage amplitude. This process is known as the attack time. When the output signal subsequently lowers in amplitude, the gain is held at the reduced state for a short period before slowly increasing to the normal value. This process is known as the hold and release time. The speed at which the amplifiers adjust to changing input signals is set by the external timing capacitor C_{CT} and the voltage applied to A/R. The AGC threshold can be set by adjusting V_{TH} . Gain reduction is a function of input signal amplitude with a maximum AGC attenuation of 20dB. Figure 2 shows the effect of an input burst exceeding the preset limit, output attack, hold and release times.

If the attack and release times are configured to respond too fast, audible artifacts often described as "pumping" or "breathing" can occur as the gain is rapidly adjusted to follow the dynamics of the signal. For best results, adjust the time constant of the AGC to accommodate the source material. For applications in which music CDs are the main audio source, a 160 μ s attack time with an 80ms release time is recommended. Music applications typically require a shorter release time than voice or movie content.

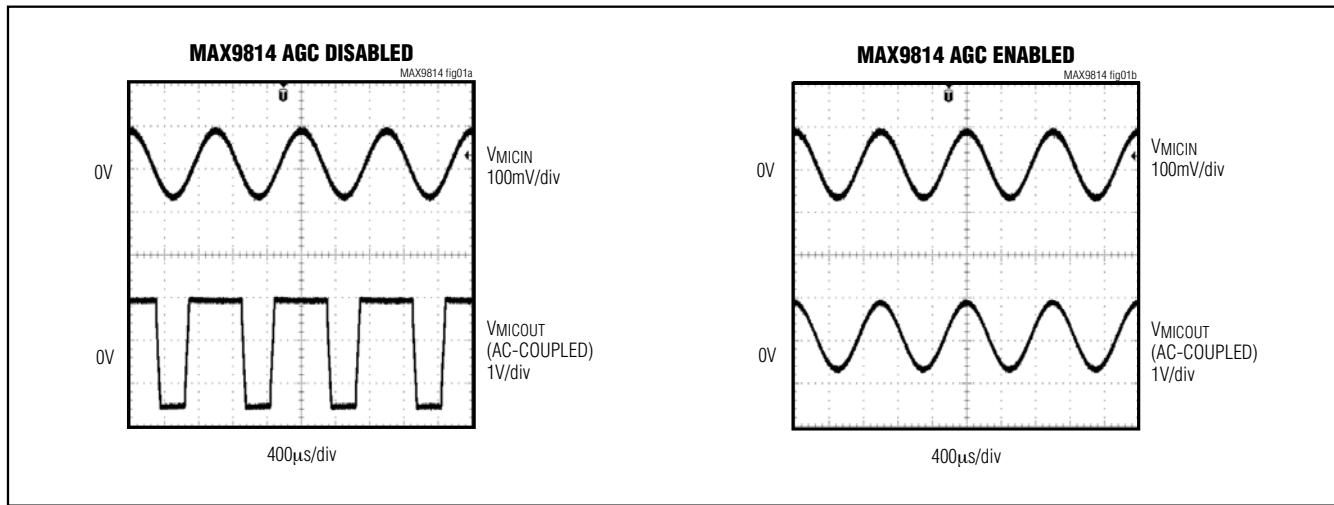


Figure 1. Microphone Input with and Without AGC

Microphone Amplifier with AGC and Low-Noise Microphone Bias

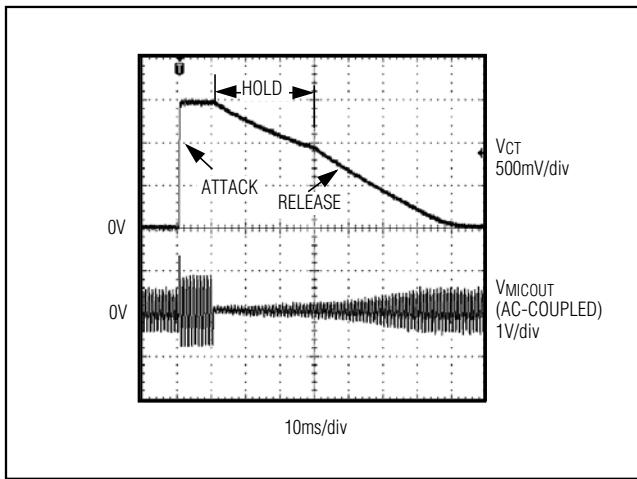


Figure 2. Input Burst Exceeding AGC Limit

Attack Time

The attack time is the time it takes for the AGC to reduce the gain after the input signal has exceeded the threshold level. The gain attenuation during the attack time is exponential, and defined as one-time constant. The time constant of the attack is given by $2400 \times C_{CT}$ seconds (where C_{CT} is the external timing capacitor):

- Use a short attack time for the AGC to react quickly to transient signals, such as snare drum beats (music) or gun shots (DVD).
- Use a longer attack time to allow the AGC to ignore short-duration peaks and only reduce the gain when a noticeable increase in loudness occurs. Short-duration peaks are not reduced, but louder passages are. This allows the louder passages to be reduced in volume, thereby maximizing output dynamic range.

Hold Time

Hold time is the delay after the signal falls below the threshold level before the release phase is initiated. Hold time is internally set to 30ms and nonadjustable. The hold time is cancelled by any signal exceeding the set threshold level, and the attack time is reinitiated.

Release Time

The release time is how long it takes for the gain to return to its normal level after the output signal has fallen below the threshold level and 30ms hold time has expired. Release time is defined as release from a 20dB gain compression to 10% of the nominal gain setting after the input signal has fallen below the TH threshold and the 30ms hold time has expired. Release time is adjustable and has a minimum of 25s. The release time is set by picking an attack time using C_{CT}

and setting the attack-to-release time ratio by configuring A/R as shown in Table 1:

- Use a small ratio to maximize the speed of the AGC.
- Use a large ratio to maximize the sound quality and prevent repeated excursions above the threshold from being independently adjusted by the AGC.

AGC Output Threshold

The output threshold that activates AGC is adjustable through the use of an external resistive divider. Once the divider is set, AGC reduces the gain to match the output voltage to the voltage set at the TH input.

Microphone Bias

The MAX9814 features an internal low-noise microphone bias voltage capable of driving most electret condenser microphones. The microphone bias is regulated at 2V to provide that the input signal to the low-noise preamplifier does not clip to ground.

Applications Information

Programming Attack and Release Times

Attack and release times are set by selecting the capacitance value between CT and GND, and by setting the logic state of A/R (Table 1). A/R is a tri-level logic input that sets the attack-to-release time ratio.

Table 1. Attack-and-Release Ratios

| A/R | ATTACK/RELEASE RATIO |
|-----------------|----------------------|
| GND | 1:500 |
| V _{DD} | 1:2000 |
| Unconnected | 1:4000 |

The attack and release times can be selected by utilizing the corresponding capacitances listed in Table 2.

Table 2. Attack-and-Release Time

| C _{CT} | t _{ATTACK} (ms) | t _{RELEASE} (ms) | | |
|-----------------|--------------------------|---------------------------|-----------------------|-------------------|
| | | A/R = GND | A/R = V _{DD} | A/R = UNCONNECTED |
| 22nF | 0.05 | 25 | 100 | 200 |
| 47nF | 0.11 | 55 | 220 | 440 |
| 68nF | 0.16 | 80 | 320 | 640 |
| 100nF | 0.24 | 120 | 480 | 960 |
| 220nF | 0.53 | 265 | 1060 | 2120 |
| 470nF | 1.1 | 550 | 2200 | 4400 |
| 680nF | 1.63 | 815 | 3260 | 6520 |
| 1μF | 2.4 | 1200 | 4800 | 9600 |

Microphone Amplifier with AGC and Low-Noise Microphone Bias

Setting the AGC Threshold

To set the output-voltage threshold at which the microphone output is clamped, an external resistor-divider must be connected from MICBIAS to ground with the output of the resistor-divider applied to TH. The voltage V_{TH} determines the peak output-voltage threshold at which the output becomes clamped. The maximum signal swing at the output is then limited to two times V_{TH} and remains at that level until the amplitude of the input signal is reduced. To disable AGC, connect TH to MICBIAS.

Microphone Bias Resistor

MICBIAS is capable of sourcing 20mA. Select a value for $R_{MICBIAS}$ that provides the desired bias current for the electret microphone. A value of $2.2\text{k}\Omega$ is usually sufficient for a microphone of typical sensitivity. Consult the microphone data sheet for the recommended bias resistor.

Bias Capacitor

The BIAS output of the MAX9814 is internally buffered and provides a low-noise bias. Bypass BIAS with a 470nF capacitor to ground.

Input Capacitor

The input AC-coupling capacitor (C_{IN}) and the input resistance (R_{IN}) to the microphone amplifier form a highpass filter that removes any DC bias from an input signal (see the *Typical Application Circuit/Functional Diagram*). C_{IN} prevents any DC components from the input-signal source from appearing at the amplifier outputs. The -3dB point of the highpass filter, assuming zero source impedance due to the input signal source, is given by:

$$f_{-3\text{dB_IN}} = \frac{1}{2\pi \times R_{IN} \times C_{IN}}$$

Choose C_{IN} such that $f_{-3\text{dB_IN}}$ is well below the lowest frequency of interest. Setting $f_{-3\text{dB_IN}}$ too high affects the amplifier's low-frequency response. Use capacitors with low-voltage coefficient dielectrics. Aluminum electrolytic, tantalum, or film dielectric capacitors are good choices for AC-coupling capacitors. Capacitors with high-voltage coefficients, such as ceramics (non-C0G dielectrics), can result in increased distortion at low frequencies.

Output Capacitor

The output of the MAX9814 is biased at 1.23V. To eliminate the DC offset, an AC-coupling capacitor (C_{OUT}) must be used. Depending on the input resistance (R_L) of the following stage, C_{OUT} and R_L effectively form a highpass filter. The -3dB point of the highpass filter, assuming zero output impedance, is given by:

$$f_{-3\text{dB_OUT}} = \frac{1}{2\pi \times R_L \times C_{OUT}}$$

Shutdown

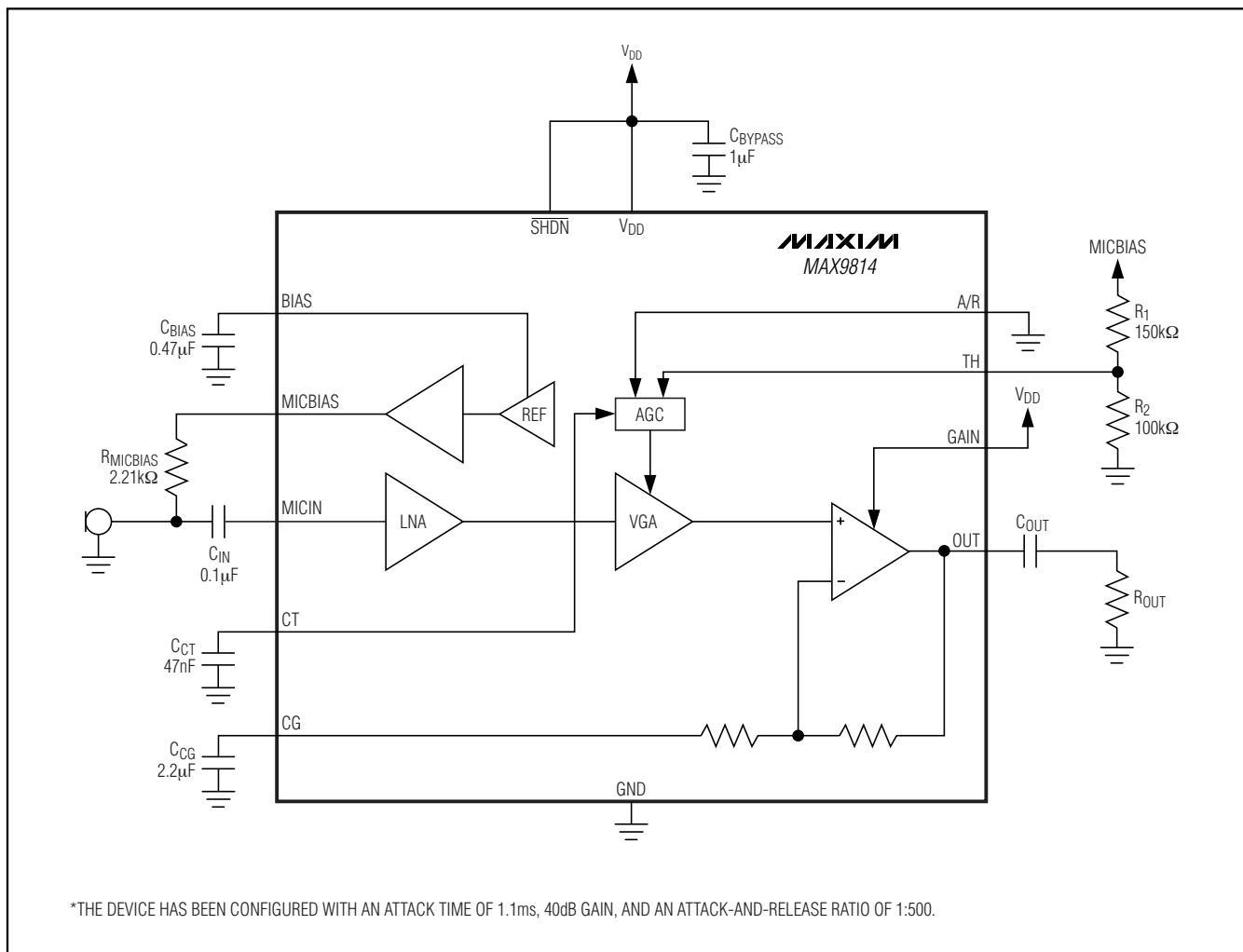
The MAX9814 features a low-power shutdown mode. When \overline{SHDN} goes low, the supply current drops to $0.01\mu\text{A}$, the output enters a high-impedance state, and the bias current to the microphone is switched off. Driving \overline{SHDN} high enables the amplifier. Do not leave \overline{SHDN} unconnected.

Power-Supply Bypassing and PCB Layout

Bypass the power supply with a $0.1\mu\text{F}$ capacitor to ground. Reduce stray capacitance by minimizing trace lengths and place external components as close to the device as possible. Surface-mount components are recommended. In systems where analog and digital grounds are available, connect the MAX9814 to analog ground.

Microphone Amplifier with AGC and Low-Noise Microphone Bias

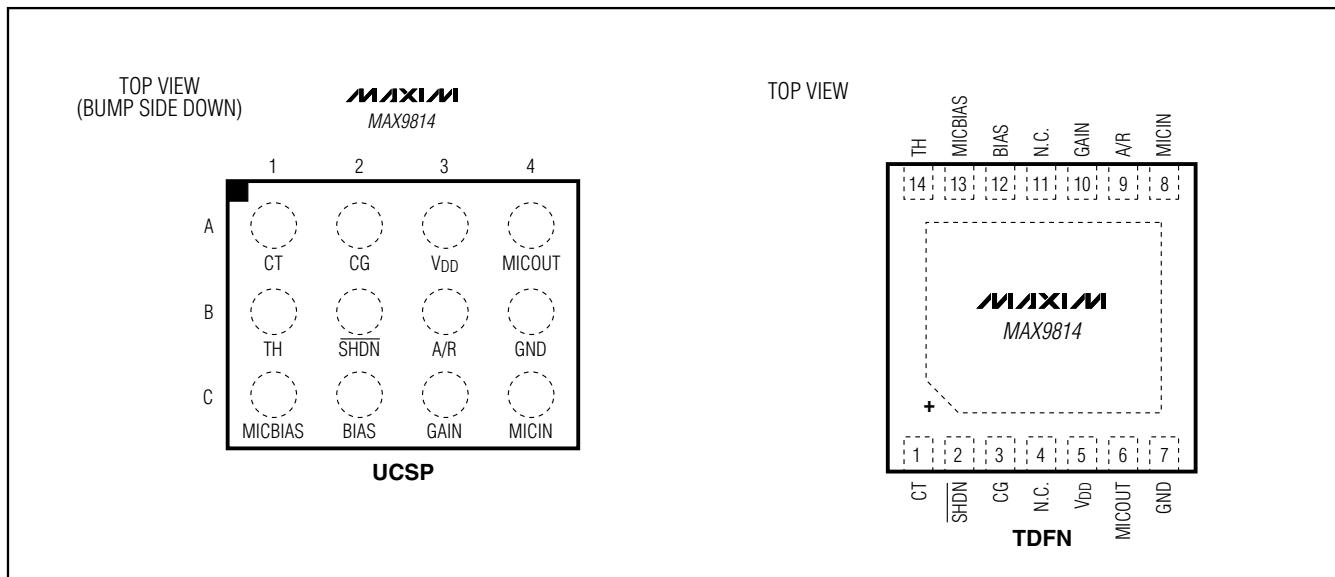
Typical Application Circuit/Functional Diagram



*THE DEVICE HAS BEEN CONFIGURED WITH AN ATTACK TIME OF 1.1ms, 40dB GAIN, AND AN ATTACK-AND-RELEASE RATIO OF 1:500.

Microphone Amplifier with AGC and Low-Noise Microphone Bias

Pin Configurations



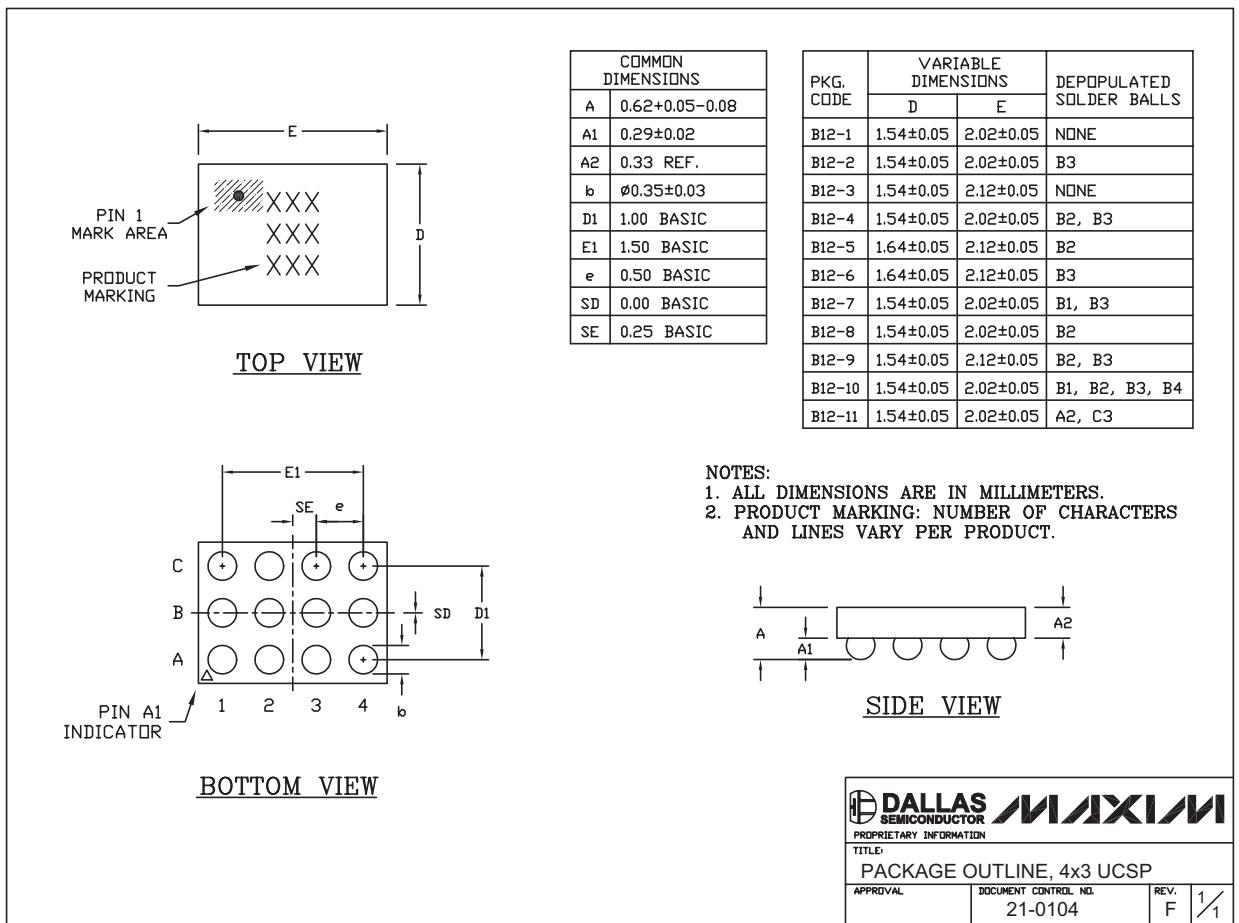
Chip Information

PROCESS: BiCMOS

Microphone Amplifier with AGC and Low-Noise Microphone Bias

Package Information

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information go to www.maxim-ic.com/packages.)



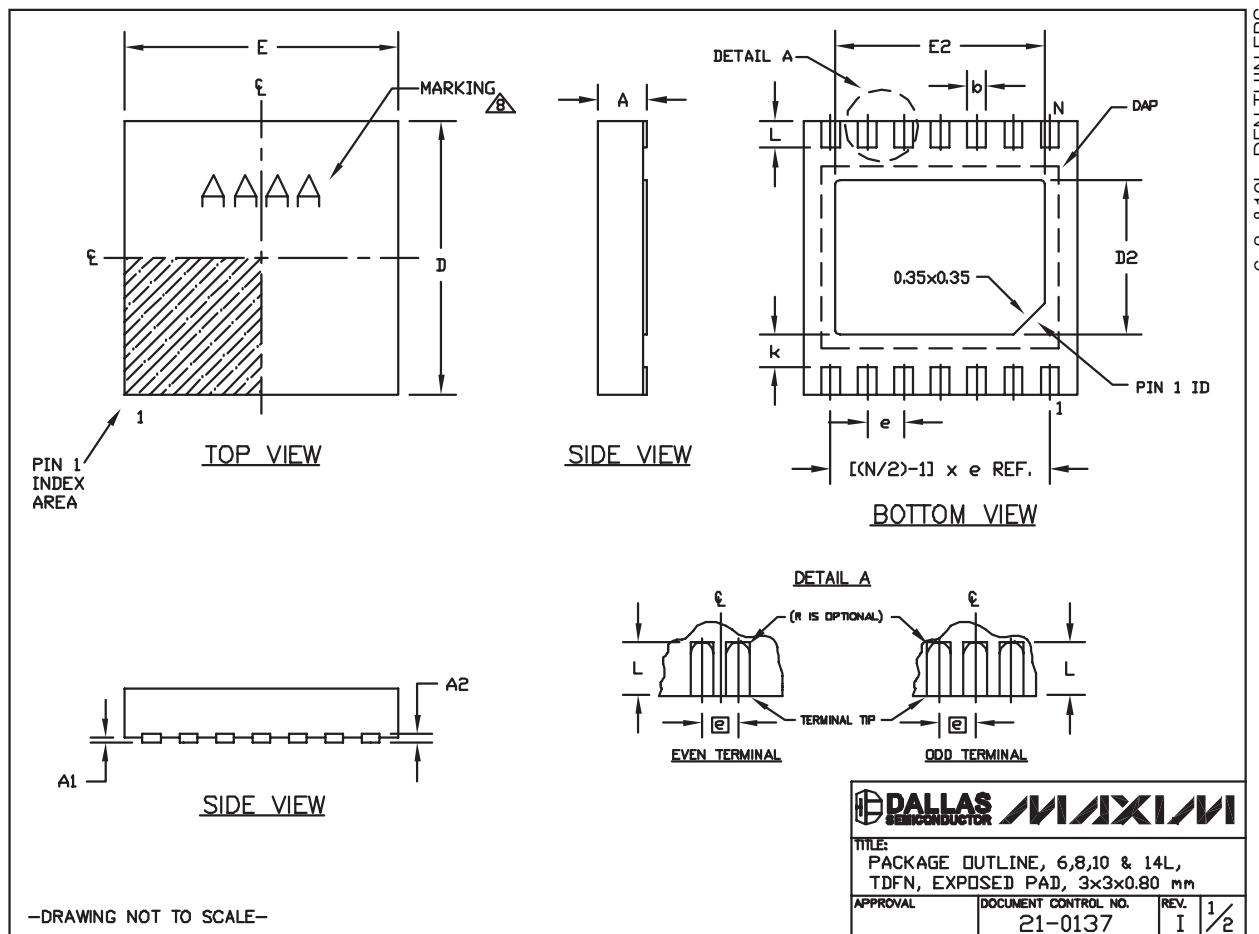
| | | |
|----------------------------------|---------------------------------|------------------|
| DALLAS SEMICONDUCTOR | | MAXIM |
| PROPRIETARY INFORMATION | | |
| TITLE: PACKAGE OUTLINE, 4x3 UCSP | | |
| APPROVAL | DOCUMENT CONTROL NO. 21-0104 | REV. F 1/1 |

Microphone Amplifier with AGC and Low-Noise Microphone Bias

Package Information (continued)

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information go to www.maxim-ic.com/packages.)

MAX9814



Microphone Amplifier with AGC and Low-Noise Microphone Bias

Package Information (continued)

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information go to www.maxim-ic.com/packages.)

| COMMON DIMENSIONS | | PACKAGE VARIATIONS | | | | | | | | |
|-------------------|-----------|--------------------|-----------|----|-----------|-----------|----------|----------------|-----------|---------------|
| SYMBOL | MIN. | MAX. | PKG. CODE | N | D2 | E2 | e | JEDEC SPEC | b | [(N/2)-1] x e |
| A | 0.70 | 0.80 | T633-2 | 6 | 1.50±0.10 | 2.30±0.10 | 0.95 BSC | MO229 / WEEA | 0.40±0.05 | 1.90 REF |
| D | 2.90 | 3.10 | T833-2 | 8 | 1.50±0.10 | 2.30±0.10 | 0.65 BSC | MO229 / WEEC | 0.30±0.05 | 1.95 REF |
| E | 2.90 | 3.10 | T833-3 | 8 | 1.50±0.10 | 2.30±0.10 | 0.65 BSC | MO229 / WEEC | 0.30±0.05 | 1.95 REF |
| A1 | 0.00 | 0.05 | T1033-1 | 10 | 1.50±0.10 | 2.30±0.10 | 0.50 BSC | MO229 / WEED-3 | 0.25±0.05 | 2.00 REF |
| L | 0.20 | 0.40 | T1033-2 | 10 | 1.50±0.10 | 2.30±0.10 | 0.50 BSC | MO229 / WEED-3 | 0.25±0.05 | 2.00 REF |
| k | 0.25 MIN. | | T1433-1 | 14 | 1.70±0.10 | 2.30±0.10 | 0.40 BSC | ----- | 0.20±0.05 | 2.40 REF |
| A2 | 0.20 REF. | | T1433-2 | 14 | 1.70±0.10 | 2.30±0.10 | 0.40 BSC | ----- | 0.20±0.05 | 2.40 REF |

NOTES:

1. ALL DIMENSIONS ARE IN mm. ANGLES IN DEGREES.
2. COPLANARITY SHALL NOT EXCEED 0.08 mm.
3. WARPAGE SHALL NOT EXCEED 0.10 mm.
4. PACKAGE LENGTH/PACKAGE WIDTH ARE CONSIDERED AS SPECIAL CHARACTERISTIC(S).
5. DRAWING CONFORMS TO JEDEC MO229, EXCEPT DIMENSIONS "D2" AND "E2", AND T1433-1 & T1433-2.
6. "N" IS THE TOTAL NUMBER OF LEADS.
7. NUMBER OF LEADS SHOWN ARE FOR REFERENCE ONLY.

 MARKING IS FOR PACKAGE ORIENTATION REFERENCE ONLY.

-DRAWING NOT TO SCALE-

| | | | |
|--|----------------------|---|---------------|
|  DALLAS SEMICONDUCTOR | |  | |
| TITLE: PACKAGE OUTLINE, 6,8,10 & 14L, TDFN, EXPOSED PAD, 3x3x0.80 mm | | | |
| APPROVAL | DOCUMENT CONTROL NO. | REV. | 21-0137 I 2/2 |

Maxim cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim product. No circuit patent licenses are implied. Maxim reserves the right to change the circuitry and specifications without notice at any time.

14 **Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600**

© 2007 Maxim Integrated Products

MAXIM is a registered trademark of Maxim Integrated Products, Inc.

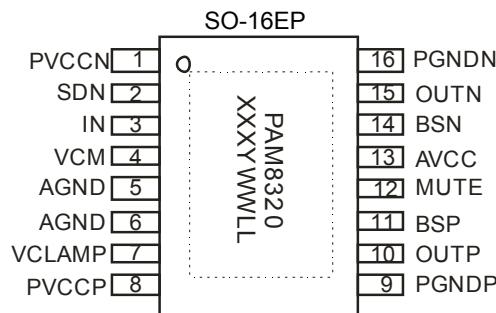
Description

The PAM8320 is an efficient 20W mono Class-D audio power amplifier, designed to drive speakers as low as 4Ω in a bridge-tied-load configuration. Due to the low power dissipation and high efficiency of up to 95%, the device can be used without any external heat sink whilst playing music.

The PAM8320 features short circuit protection, thermal shutdown, over voltage protection and under voltage lock-out.

The PAM8320 is available in a SO-16EP package.

Pin Assignments



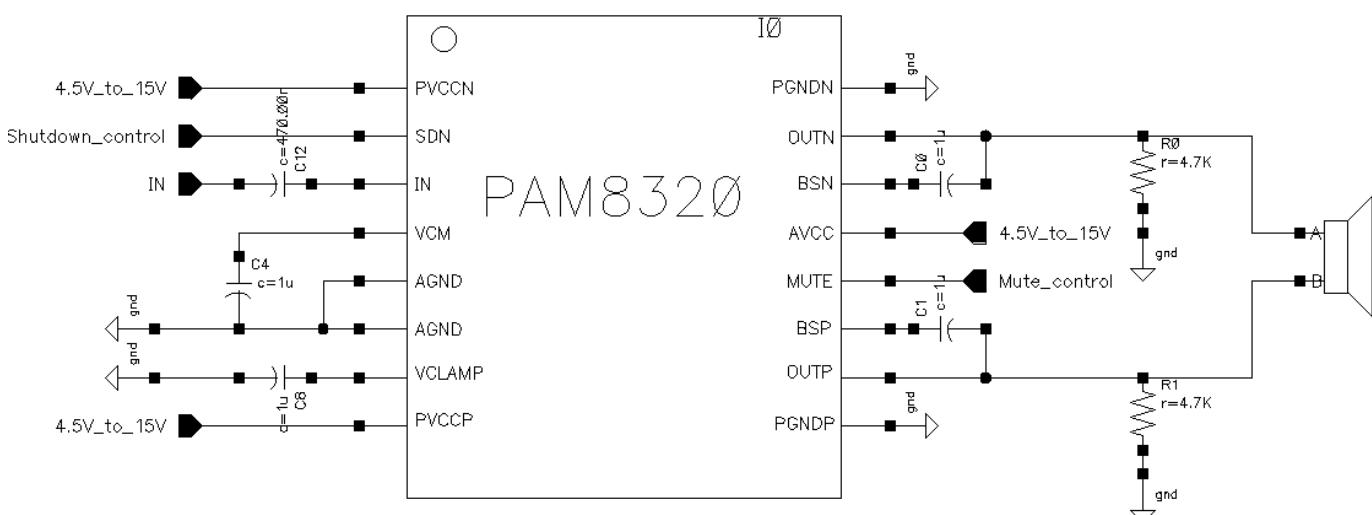
Features

- Operates from 4.5V to 15V
- 20W into 4Ω BTL Load from 12V Supply
- Single-Ended Analog Input
- No Pop Noise for Start-up and Shut-down Sequences
- Internal Oscillator (No External Components Required)
- High Efficient Class-D Operation Eliminates Need for Heat Sinks
- Thermal and Short-Circuit Protection with Auto Recovery
- Over Voltage Protection and Under Voltage Lock-out
- Space-Saving Surface-Mount SO-16EP Package
- Pb-Free Package

Applications

- PC Speaker
- Blue Tooth Speaker
- Home Sound Systems
- Active Speakers
- Docking stations

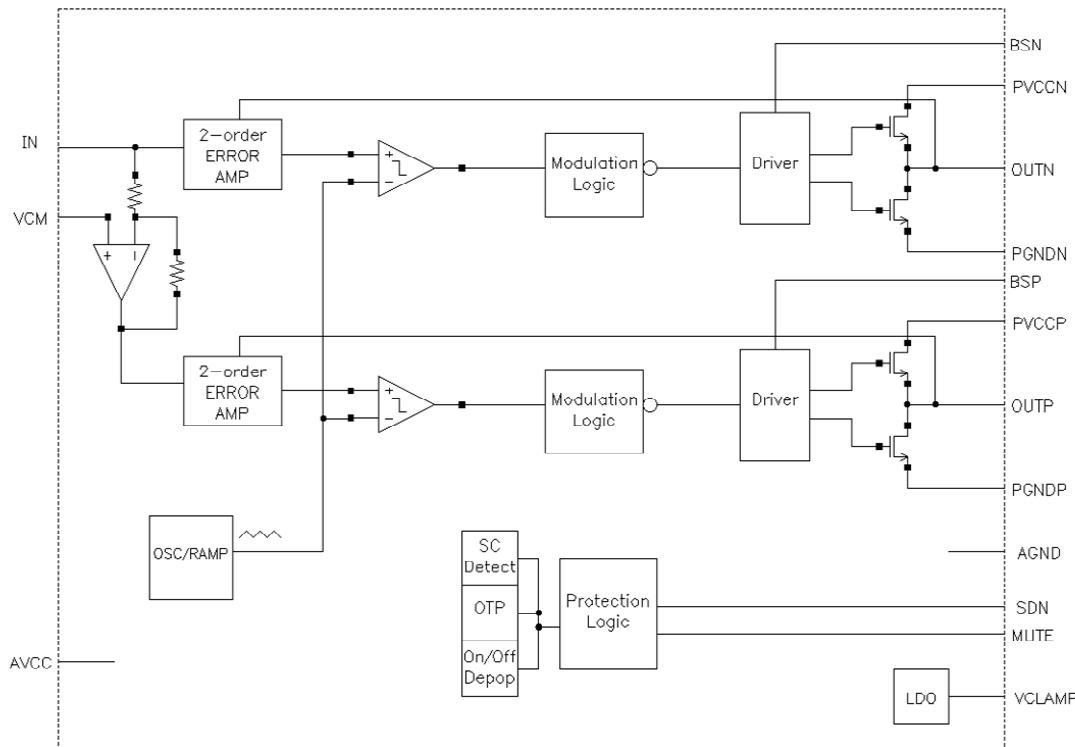
Typical Applications Circuit



Pin Descriptions

| Pin | Name | I/O/P | Description |
|-----|--------|-------|--|
| 1 | PVCCN | P | Power supply for negative H-bridge, not connected to PVCCP or AVCC |
| 2 | SDN | I | Shutdown signal for IC (low=shutdown, high=operational). TTL logic levels with compliance to AVCC |
| 3 | IN | I | Audio input |
| 4 | VCM | O | Reference for analog cells |
| 5,6 | AGND | P | Analog ground for digital/analog cells in core |
| 7 | VCLAMP | P | Internally generated voltage supply for bootstrap. Not to be used as a supply or connected to any component other than the decoupling capacitor. |
| 8 | PVCCP | P | Power supply for positive H-bridge, not connected to PVCCN or AVCC |
| 9 | PGNDP | P | Power ground for positive H-bridge |
| 10 | OUTP | O | Positive BTL output |
| 11 | BSP | I/O | Bootstrap terminal for high-side drive of positive BTL output |
| 12 | MUTE | I | A logic high on this pin disables the outputs. A low on this pin enables the outputs. TTL logic levels with compliance to AVCC |
| 13 | AVCC | P | High-voltage analog power supply |
| 14 | BSN | I/O | Bootstrap terminal for high-side drive of negative BTL output |
| 15 | OUTN | O | Negative BTL output |
| 16 | PGNDN | P | Power ground for negative H-bridge |

Functional Block Diagram



Absolute Maximum Ratings (@ $T_A = +25^\circ\text{C}$, unless otherwise specified.)

| Parameter | Rating | Unit |
|--|----------------------|------|
| Supply Voltage (V_{CC}) | 18 | V |
| Logic Input Voltage (SDN, MUTE) | -0.3 to $V_{CC}+0.3$ | V |
| Analog Input Voltage (I_N) | -0.3 to 5.5 | V |
| Storage Temperature | -65 to +150 | °C |
| Maximum Junction Temperature | +150 | °C |
| Junction to ambient thermal resistance | 40 | °C/W |

Recommended Operating Conditions (@ $T_A = +25^\circ\text{C}$, unless otherwise specified.)

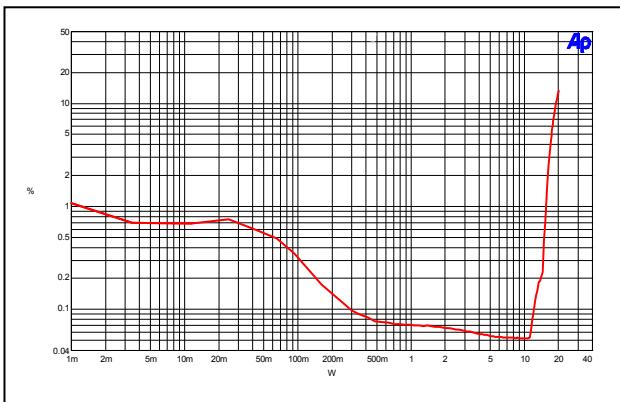
| Symbol | Parameter | Min | Max | Unit |
|----------|-------------------------------------|-----|------|------|
| V_{CC} | Supply Voltage | 4.5 | 15 | V |
| T_A | Operating Ambient Temperature Range | -40 | +85 | °C |
| T_J | Junction Temperature Range | -40 | +125 | °C |

Electrical Characteristics (@ $T_A = +25^\circ\text{C}$, $V_{CC} = 12\text{V}$, Gain = 20dB, $R_L = L(33\mu\text{H}) + R + L(33\mu\text{H})$, unless otherwise noted.)

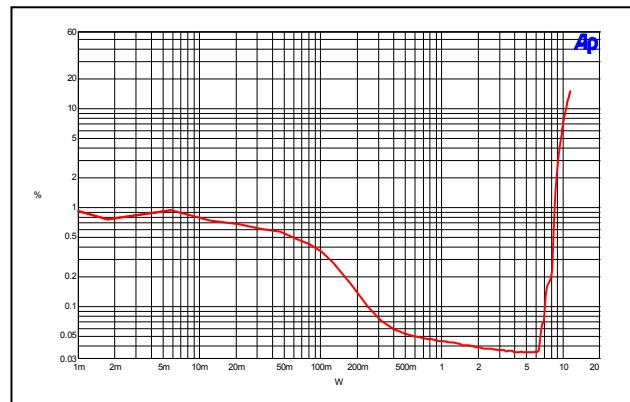
| Symbol | PARAMETER | Test Conditions | MIN | TYP | MAX | Units |
|-----------------------|---|--|-----|------|-----|------------------|
| $ V_{OS} $ | Class-D output offset voltage(measured differently) | $V_i=0\text{V}$, $A_v=20\text{dB}$ | — | 20 | 100 | mV |
| $I_{CC(q)}$ | Quiescent supply current | $\text{SDN}=3.0\text{V}$, $\text{MUTE}=0\text{V}$, No Load | — | 15 | 30 | mA |
| $I_{CC(\text{MUTE})}$ | Quiescent supply current in mute mode | $\text{MUTE}=2.0\text{V}$, No load | — | 8 | 20 | mA |
| $I_{CC(\text{SDN})}$ | Quiescent current in shutdown mode | $\text{SDN}=0.5\text{V}$, No load | — | 20 | 40 | uA |
| $R_{ds(on)}$ | Drain-source on-state resistance | $I_o=0.5\text{A}$ | — | 150 | — | $\text{m}\Omega$ |
| P_{SRR} | Power Supply Rejection Ratio | $V_{\text{ripple}}=200\text{mVpp}$, $f=1\text{kHz}$, gain=20dB | — | -60 | — | dB |
| P_o | Output Power at 1% THD+N | $f=1\text{kHz}$ | — | 15 | — | W |
| | Output Power at 10% THD+N | $f=1\text{kHz}$ | — | 20 | — | |
| THD+N | Total harmonic distortion + noise | $f=1\text{kHz}$, $P_o=7\text{W}$ | — | 0.05 | — | % |
| V_n | Output integrated noise floor | 20Hz to 22kHz, A-weighted, Gain=20dB | — | 300 | — | uV |
| SNR | Signal-to-noise ratio | Max output at THD+N<1%, $f=1\text{kHz}$, Gain=20dB | — | 95 | — | dB |
| OTP | Thermal trip point | — | — | +160 | — | °C |
| OTH | Thermal hysteresis | — | — | +40 | — | °C |
| f_{osc} | Oscillator frequency | — | — | 300 | — | kHz |
| V_{IH_SDN} | SDN Input High | — | 3 | — | — | — |
| V_{IL_SDN} | SDN Input Low | — | — | — | 0.5 | — |
| V_{IH_MUTE} | MUTE Input High | — | 2 | — | — | — |
| V_{IL_MUTE} | MUTE Input Low | — | — | — | 0.5 | — |

Performance Characteristics (@ $T_A = +25^\circ\text{C}$, $V_{DD} = 12\text{V}$, Gain = 20dB, $R_L = L(33\mu\text{H}) + R + L(33\mu\text{H})$, unless otherwise noted.)

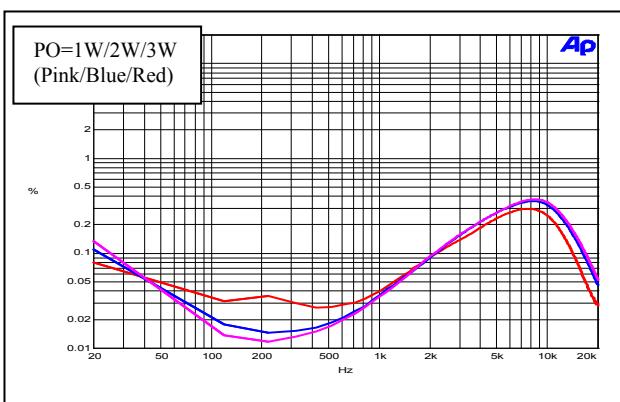
THD+N Vs. Output Power ($RL=4\Omega$)



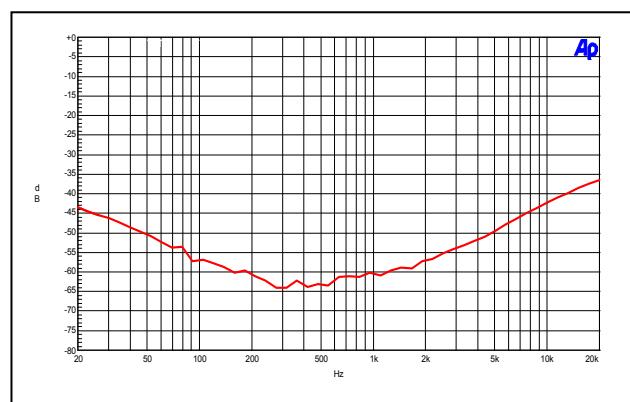
THD+N Vs. Output Power ($RL=8\Omega$)



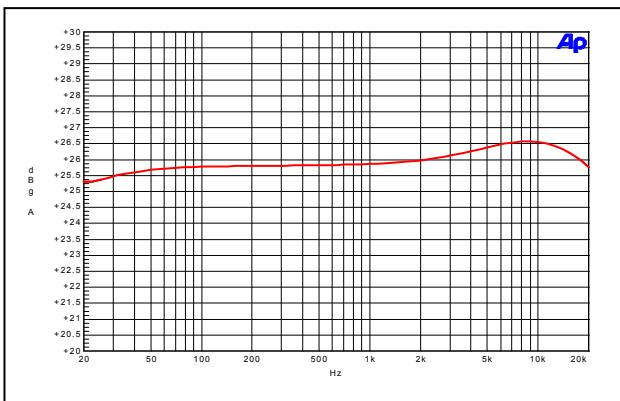
THD+N Vs. Frequency



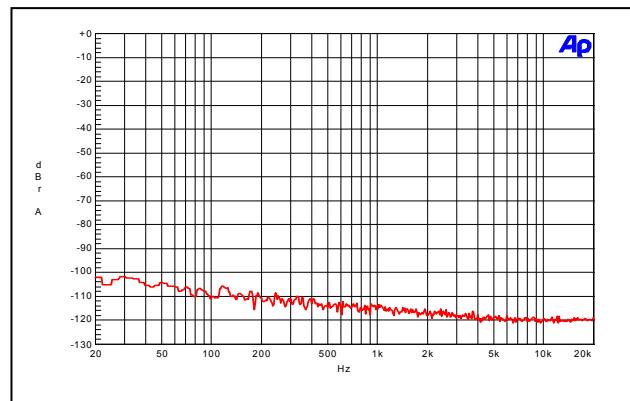
PSRR Vs. Frequency



Frequency Response

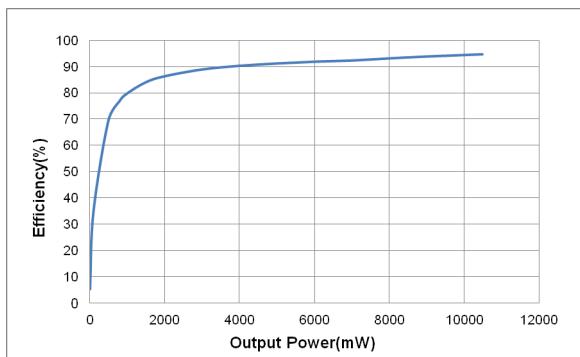


Noise Floor

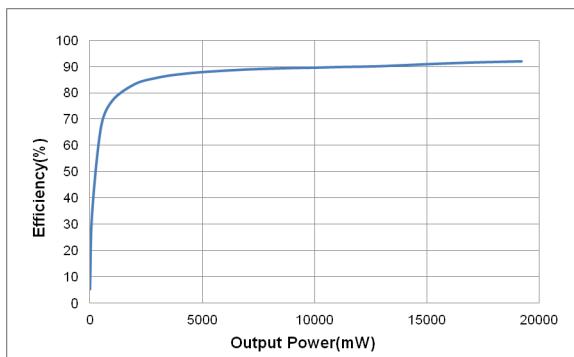


Performance Characteristics (@ $T_A=25^\circ\text{C}$, $V_{DD}=12\text{V}$, Gain=20dB, $R_L=L(33\mu\text{H})+R+L(33\mu\text{H})$, unless otherwise noted.)

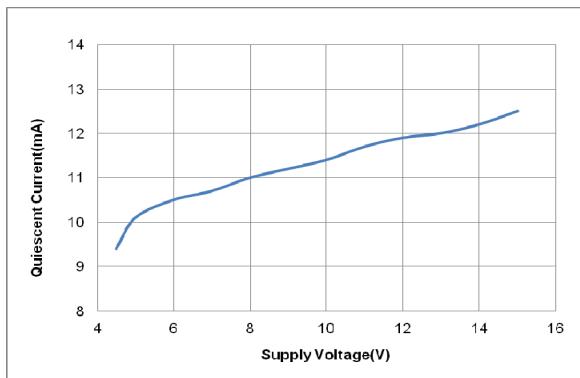
Efficiency Vs. Output Power ($RL=8\Omega$)



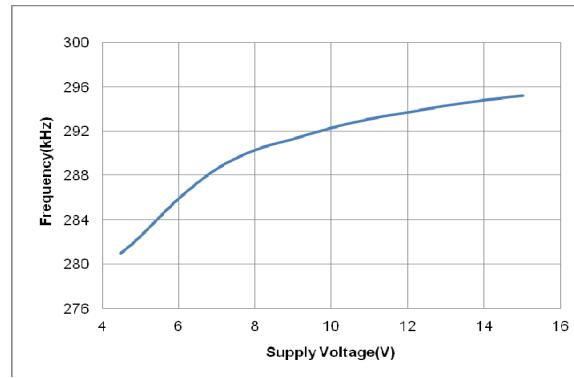
Efficiency Vs. Output Power ($RL=4\Omega$)



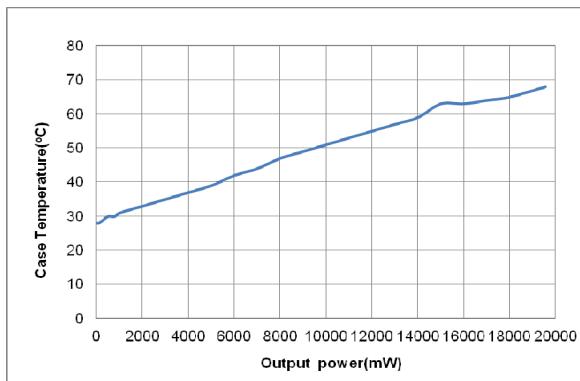
Quiescent Current Vs. Supply Voltage



OSC Frequency Vs. Supply Voltage



Case Temperature Vs. Output Power ($RL=4\Omega$)



Turn-on Response



Application Information

Input Capacitors (Ci)

In the typical application, an input capacitor Ci, is required to allow the amplifier to bias the input signal to the proper DC level for optimum operation. In this case, Ci and the minimum input impedance Ri form a high-pass filter with the corner frequency determined in the follow equation:

$$f_c = \frac{1}{(2\pi R_i C_i)}$$

It is important to consider the value of Ci as it directly affects the low frequency performance of the circuit. For example, when Ri is 40kΩ and the specification calls for a flat bass response are down to 20Hz. The equation is reconfigured as followed to determine the value of Ci:

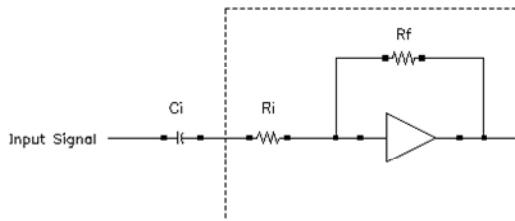
$$C_i = \frac{1}{(2\pi R_i f_c)}$$

When input resistance variation is considered Ci is 200nF, so one would likely choose a value of 220nF. A further consideration for this capacitor is the leakage path from the input source through the input network (Ci, Ri and Rf) to the load. This leakage current creates a DC offset voltage at the input to the amplifier that reduces useful headroom, especially in high gain applications. For this reason, a low-leakage tantalum or ceramic capacitor is the best choice. When polarized capacitors are used, the positive side of the capacitor should face the amplifier input in most applications as the DC level is held at VDD/2, which is likely higher than the source DC level. Please note that it is important to confirm the capacitor polarity in the application.

Input Resistance

The value of the input resistance (Ri) of the amplifier is 40kΩ ±20%. If a single capacitor is added to the input of the high-pass filter the –3dB cutoff frequency can be calculated using equation:

$$f_c = \frac{1}{(2\pi R_i C_i)}$$



Gain Formula with External Input Resistor

The default gain of PAM8320 is 26dB. The gain can be reduced by adding one external resistor between input decoupling capacitor and IN PIN. The gain formula is as below:

$$A_v = \frac{20}{1 + 14 \times \frac{R_x}{400k}}$$

Note: Rx is external input resistor

Power and Heat Dissipation

Speakers must be chosen to withstand the large output power from the PAM8320, otherwise speaker damage may occur.

Heat dissipation is very important when the device works in full power operation. Two factors affect the heat dissipation, the efficiency of the device that determines the dissipation power and the thermal resistance of the package that determines the heat dissipation capability.

The PAM8320 class-D amplifier is highly efficiency and should not need heat sink. Operating at higher powers a heat sink still may not be necessary if the PCB is carefully designed to achieve good thermal dissipation.

Dual-Side PCB

To achieve good heat dissipation the PCB's copper plate should be thicker than 35um and the copper plate on both sides of the PCB should be utilized for heat sink.

The thermal pad on the bottom of the device should be soldered to the plate of the PCB and via holes (usually 9 to 16) should be drilled in the PCB area under the device. Deposited copper on the vias should be thick enough so that the heat can be dissipated to the other side of the plate. There should be no insulation mask on the other side of the copper plate. More vias can and should be added to the PCB around the device for further thermal optimization.

How to Reduce EMI

Most applications require a ferrite bead filter for EMI elimination shown at Figure 1. The ferrite filter reduces EMI around 1MHz and higher. When selecting a ferrite bead it should be chosen with high impedance at high frequencies but low impedance at low frequencies.

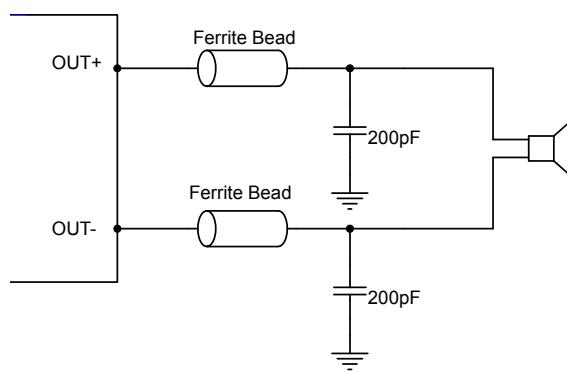


Figure 1: Ferrite Bead Filter to Reduce EMI

Shutdown Operation

The PAM8320 employs a shutdown operation mode to reduce supply current to the absolute minimum level during periods of non-use to save power. The SDN input terminal should be pull high during normal operation. Pulling SDN low causes the outputs to be muted and the amplifier enters a low-current state. SDN should never be left unconnected.

Anti-POP and Anti-Click Circuitry

The PAM8320 contains circuitry to minimize turn-on and turn-off transients or "click and pops", where turn-on refers to either power supply turn-on or device recover from shutdown mode. When the device is turned on, the amplifiers are internally muted. An internal current source ramps up the internal reference voltage. The device will remain in mute mode until the reference voltage reaches half supply voltage. As soon as the reference voltage is stable, the device will begin full operation. For the best power-off pop performance, the amplifier should be set in shutdown mode prior to removing the power supply voltage.

Internal Bias Generator Capacitor Selection

The internal bias generator (VCM) provides the internal bias for the preamplifier stage. The external input capacitors and this internal reference allow the inputs to be biased within the optimal common-mode range of the input preamplifiers.

The selection of the capacitor value on the VCM terminal is critical for achieving the best device performance. During startup or recovery from shutdown state the VCM capacitor determines the rate at which the amplifier starts up. The startup time is not critical for the best de-pop performance since any heard pop sound is the result of the class-D output switching-on other than that of the startup time. However, at least a 0.47 μ F capacitor is recommended for the VCM capacitor.

Another function of the VCM capacitor is to bypass high frequency noise on the internal bias generator.

Power Supply Decoupling, CS

The PAM8320 is a high-performance CMOS audio amplifier that requires adequate power supply decoupling to ensure the output total harmonic distortion (THD) as low as possible. Power supply decoupling also prevents the oscillations causing by long lead length between the amplifier and the speaker.

Optimum decoupling is achieved by using two different types of capacitors that target different types of noise on the power supply leads. Higher frequency transients, spikes or digital hash should be filtered with a good low equivalent-series-resistance (ESR) ceramic capacitor with a value of typically $0.1\mu F$. This capacitor should be placed as close as possible to the PVCC pin of the device. Lower frequency noise signals should be filtered with a large ceramic capacitor of $470\mu F$ or greater. It's recommended to place this capacitor near the audio power amplifier. The $10\mu F$ capacitor also serves as a local storage capacitor for supplying current during large signal transients on the amplifier outputs.

BSN and BSP Capacitors

The half H-bridge output stages use NMOS transistors therefore requiring bootstrap capacitors for the high side of each output to turn on correctly. A ceramic capacitor $220nF$ or more rated for over 25V must be connected from each output to its corresponding bootstrap input. Specifically, one $220nF$ capacitor must be connected from OUTN to BSN and another $220nF$ capacitor from OUTP to BSP. It is recommended to use $1\mu F$ BST capacitor to replace $220nF$ for lower than 100Hz applications.

VCLAMP Capacitors

To ensure that the maximum gate-to-source voltage for the NMOS output transistors is not exceeded, an internal regulator is used to clamp the gate voltage. A $1\mu F$ capacitor must be connected from VCLAMP to ground and must be rated for at least 25V. The voltages at the VCLAMP terminals vary with VCC and may not be used to power any other circuitry.

Using low-ESR Capacitors

Low-ESR capacitors are recommended throughout this application section. A real (with respect to ideal) capacitor can be modeled simply as a resistor in series with an ideal capacitor. The voltage drop across this resistor minimizes the beneficial effects of the capacitor in the circuit. The lower the equivalent value of this resistance the more the real capacitor behaves as an ideal capacitor.

Short-circuit Protection

The PAM8320 has short circuit protection circuitry on the outputs to prevent damage to the device when output-to-output shorts (BTL mode), output-to-GND shorts, or output-to-VCC shorts occur. Once a short-circuit is detected on the outputs, the output drive is immediately disabled. This is not a latched fault, if the short is removed the normal operation is restored.

Thermal Protection

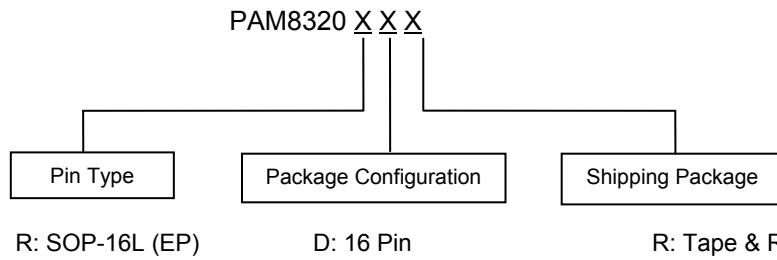
Thermal protection prevents the device from damage. When the internal die temperature exceeds a typical of $160^{\circ}C$ the device will enter a shutdown state and the outputs are disabled. This is not a latched fault, once the thermal fault is cleared and the temperature of the die decreased by $40^{\circ}C$ the device will restart with no external system interaction.

Over Voltage Protection and Under Voltage Lock-out (OVP and UVLO)

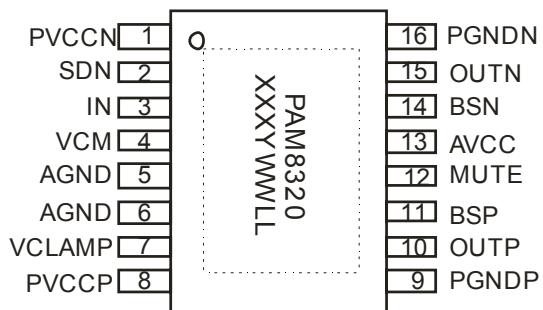
An over voltage protection (OVP) circuit is integrated in PAM8320, when the supply voltage is over 18V the OVP is active and then the output stage is disabled. The PAM8320 will auto recovery when the supply voltage is lower than the OVP threshold.

The PAM8320 incorporates circuitry designed to detect low supply voltage. When the supply voltage drops to 4.4V or below, the PAM8320 goes into a state of shutdown. When the supply voltage is higher than 4.5V normal operation is resumed.

PAM8320

Ordering Information

| Part Number | Package | Standard Package |
|-------------|---------|----------------------|
| PAM8320RDR | SO-16EP | 2,500Units/Tape&Real |

Marking Information

PAM8320: Product Code

X: Internal Code

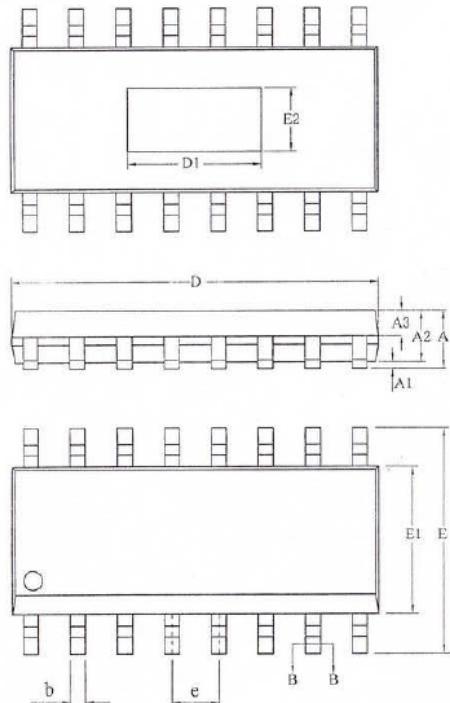
Y: Year

W: Week

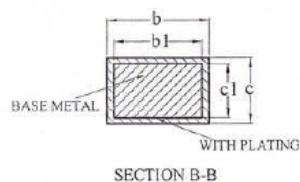
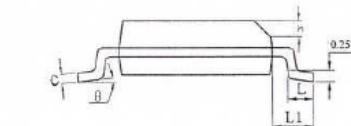
LL: Internal Code

Package Outline Dimensions (All dimensions in mm.)

Package: SO-16EP



| SYMBOL | MILLIMETER | | |
|-----------------|------------|------|-------|
| | MIN | NOM | MAX |
| A | — | — | 1.75 |
| A1 | 0.05 | — | 0.225 |
| A2 | 1.30 | 1.40 | 1.50 |
| A3 | 0.60 | 0.65 | 0.70 |
| b | 0.39 | — | 0.48 |
| b1 | 0.38 | 0.41 | 0.43 |
| c | 0.21 | — | 0.26 |
| c1 | 0.19 | 0.20 | 0.21 |
| D | 9.70 | 9.90 | 10.10 |
| E | 5.80 | 6.00 | 6.20 |
| E1 | 3.70 | 3.90 | 4.10 |
| e | 1.27BSC | | |
| h | 0.25 | — | 0.50 |
| L | 0.50 | — | 0.80 |
| L1 | 1.05BSC | | |
| Ø | Ø | — | 8° |
| D1 | 3.86REF | | |
| E2 | 1.67REF | | |
| LT载体尺寸 (mil) | 95*180 | | |



IMPORTANT NOTICE

DIODES INCORPORATED MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARDS TO THIS DOCUMENT, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION).

Diodes Incorporated and its subsidiaries reserve the right to make modifications, enhancements, improvements, corrections or other changes without further notice to this document and any product described herein. Diodes Incorporated does not assume any liability arising out of the application or use of this document or any product described herein; neither does Diodes Incorporated convey any license under its patent or trademark rights, nor the rights of others. Any Customer or user of this document or products described herein in such applications shall assume all risks of such use and will agree to hold Diodes Incorporated and all the companies whose products are represented on Diodes Incorporated website, harmless against all damages.

Diodes Incorporated does not warrant or accept any liability whatsoever in respect of any products purchased through unauthorized sales channel. Should Customers purchase or use Diodes Incorporated products for any unintended or unauthorized application, Customers shall indemnify and hold Diodes Incorporated and its representatives harmless against all claims, damages, expenses, and attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized application.

Products described herein may be covered by one or more United States, international or foreign patents pending. Product names and markings noted herein may also be covered by one or more United States, international or foreign trademarks.

This document is written in English but may be translated into multiple languages for reference. Only the English version of this document is the final and determinative format released by Diodes Incorporated.

LIFE SUPPORT

Diodes Incorporated products are specifically not authorized for use as critical components in life support devices or systems without the express written approval of the Chief Executive Officer of Diodes Incorporated. As used herein:

A. Life support devices or systems are devices or systems which:

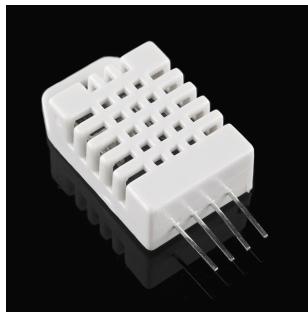
1. are intended to implant into the body, or
2. support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in significant injury to the user.

B. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or to affect its safety or effectiveness.

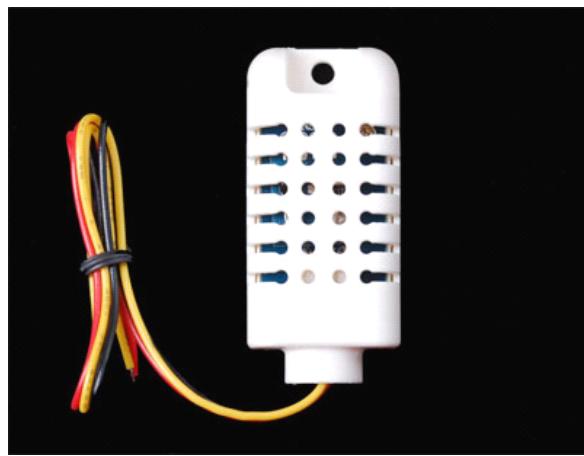
Customers represent that they have all necessary expertise in the safety and regulatory ramifications of their life support devices or systems, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of Diodes Incorporated products in such safety-critical, life support devices or systems, notwithstanding any devices- or systems-related information or support that may be provided by Diodes Incorporated. Further, Customers must fully indemnify Diodes Incorporated and its representatives against any damages arising out of the use of Diodes Incorporated products in such safety-critical, life support devices or systems.

Copyright © 2013, Diodes Incorporated

www.diodes.com



Standard AM2302/DHT22



AM2302/DHT22 with big case and wires

Digital relative humidity & temperature sensor AM2302/DHT22

1. Feature & Application:

- *High precision
- *Capacitive type
- *Full range temperature compensated
- *Relative humidity and temperature measurement
- *Calibrated digital signal
- *Outstanding long-term stability
- *Extra components not needed
- *Long transmission distance, up to 100 meters
- *Low power consumption
- *4 pins packaged and fully interchangeable

2. Description:

AM2302 output calibrated digital signal. It applies exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability. Its sensing elements are connected with 8-bit single-chip computer.

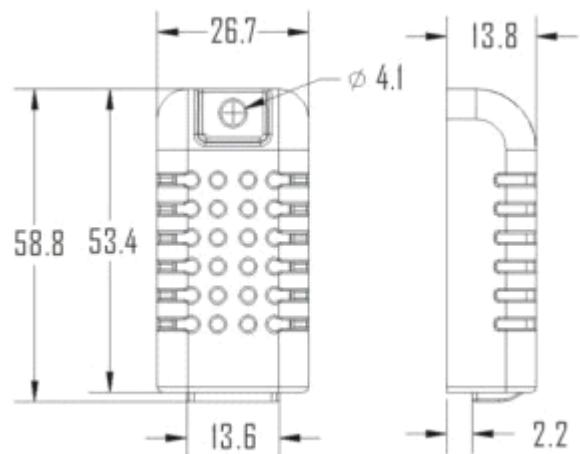
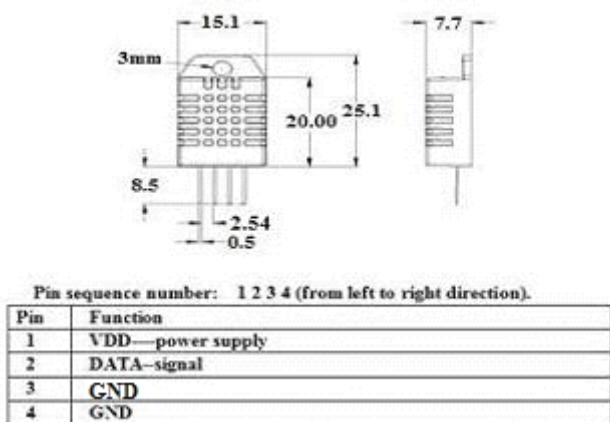
Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory.

Small size & low consumption & long transmission distance(100m) enable AM2302 to be suited in all kinds of harsh application occasions. Single-row packaged with four pins, making the connection very convenient.

3. Technical Specification:

| | | |
|---------------------------|-------------------------------|---------------------------|
| Model | AM2302 | |
| Power supply | 3.3-5.5V DC | |
| Output signal | digital signal via 1-wire bus | |
| Sensing element | Polymer humidity capacitor | |
| Operating range | humidity 0-100%RH; | temperature -40~80Celsius |
| Accuracy | humidity +2%RH(Max +5%RH); | temperature +-0.5Celsius |
| Resolution or sensitivity | humidity 0.1%RH; | temperature 0.1Celsius |
| Repeatability | humidity +-1%RH; | temperature +-0.2Celsius |
| Humidity hysteresis | +0.3%RH | |
| Long-term Stability | +0.5%RH/year | |
| Interchangeability | fully interchangeable | |

4. Dimensions: (unit---mm)



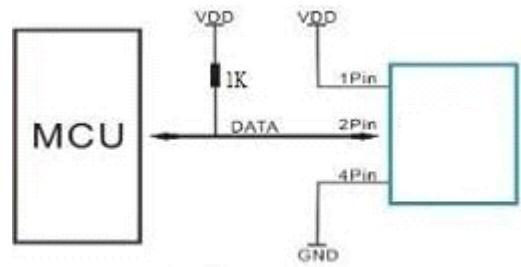
Standard AM2302's dimensions as above

Big case's dimensions as above

Red wire—power supply, Black wire—GND

Yellow wire—Data output

5. Electrical connection diagram:



6. Operating specifications:

(1) Power and Pins

Power's voltage should be 3.3-5.5V DC. When power is supplied to sensor, don't send any instruction to the sensor within one second to pass unstable status. One capacitor valued 100nF can be added between VDD and GND for wave filtering.

(2) Communication and signal

1-wire bus is used for communication between MCU and AM2302. (Our 1-wire bus is specially designed, it's different from Maxim/Dallas 1-wire bus, so it's incompatible with Dallas 1-wire bus.)

Illustration of our 1-wire bus:

DATA=16 bits RH data+16 bits Temperature data+8 bits check-sum

Example: MCU has received 40 bits data from AM2302 as

0000 0010 1000 1100 0000 0001 0101 1111 1110 1110
 16 bits RH data 16 bits T data check sum

Here we convert 16 bits RH data from binary system to decimal system,

0000 0010 1000 1100 → 652
 Binary system Decimal system

RH=652/10=65.2%RH

Here we convert 16 bits T data from binary system to decimal system,

0000 0001 0101 1111 → 351
 Binary system Decimal system

T=351/10=35.1°C

When highest bit of temperature is 1, it means the temperature is below 0 degree Celsius.

Example: 1000 0000 0110 0101, T= minus 10.1°C

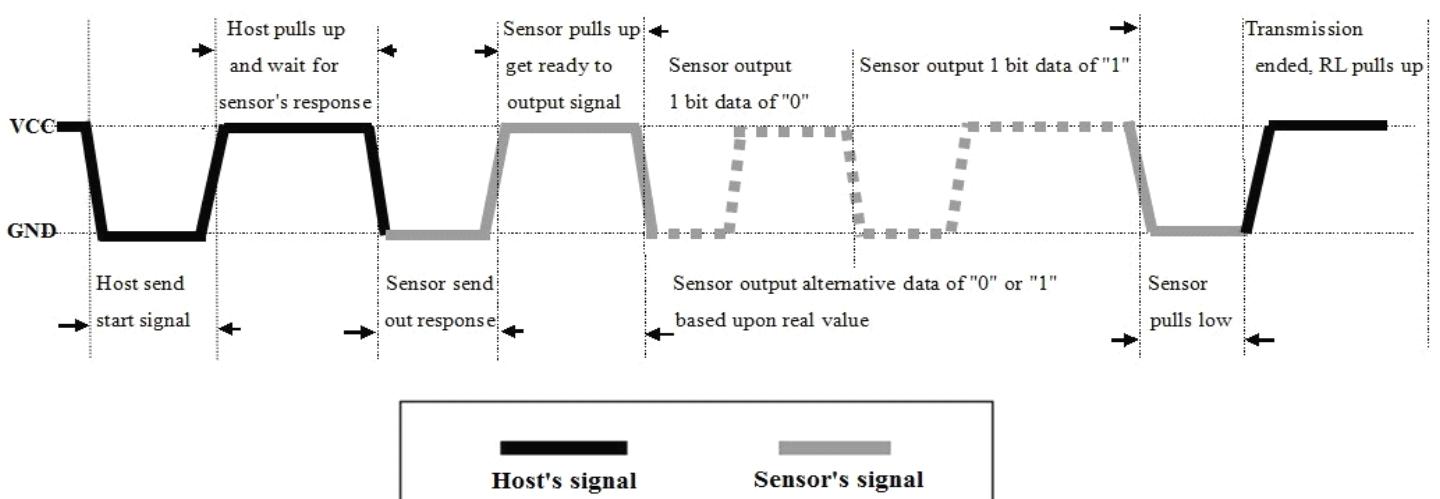
16 bits T data

Sum=0000 0010+1000 1100+0000 0001+0101 1111=1110 1110

Check-sum=the last 8 bits of Sum=1110 1110

When MCU send start signal, AM2302 change from standby-status to running-status. When MCU finishes sending the start signal, AM2302 will send response signal of 40-bit data that reflect the relative humidity and temperature to MCU. Without start signal from MCU, AM2302 will not give response signal to MCU. One start signal for one response data from AM2302 that reflect the relative humidity and temperature. AM2302 will change to standby status when data collecting finished if it don't receive start signal from MCU again.

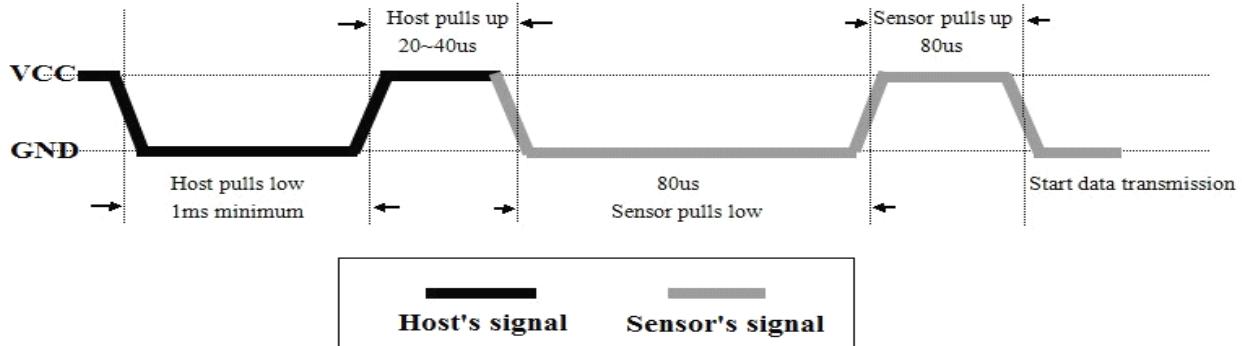
See below figure for overall communication process, **the interval of whole process must beyond 2 seconds**.



- 1) Step 1: MCU send out start signal to AM2302 and AM2302 send response signal to MCU

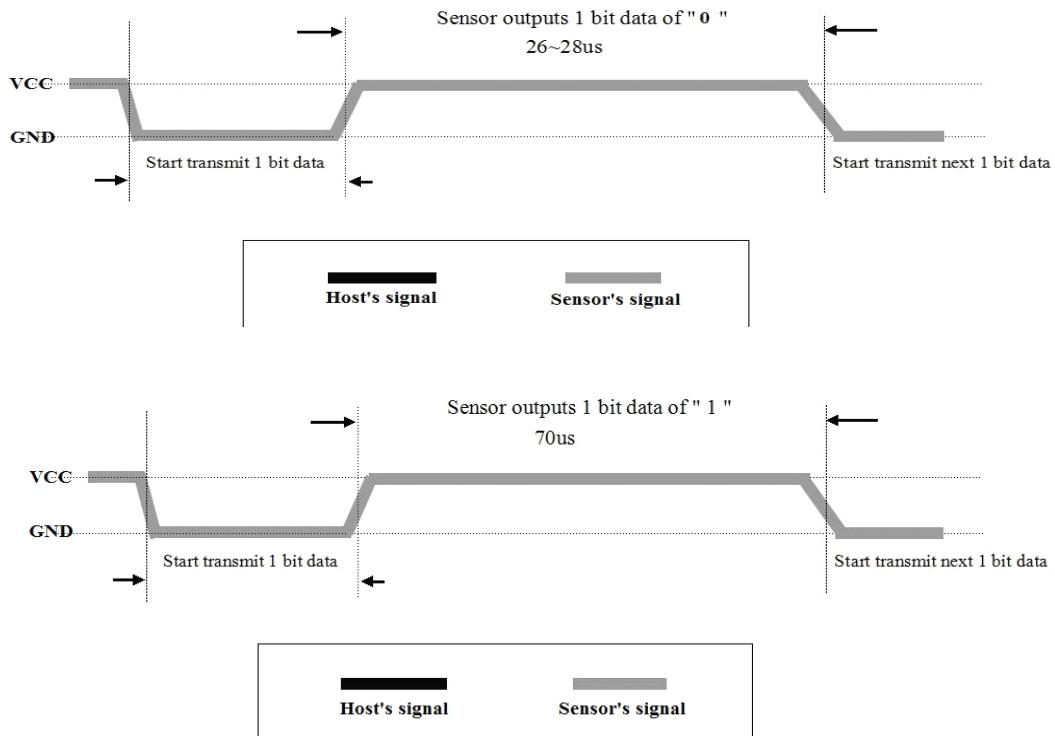
Data-bus's free status is high voltage level. When communication between MCU and AM2302 begins, MCU will pull low data-bus and this process must beyond at least 1~10ms to ensure AM2302 could detect MCU's signal, then MCU will pulls up and wait 20-40us for AM2302's response.

When AM2302 detect the start signal, AM2302 will pull low the bus 80us as response signal, then AM2302 pulls up 80us for preparation to send data. See below figure:



2). Step 2: AM2302 send data to MCU

When AM2302 is sending data to MCU, every bit's transmission begin with low-voltage-level that last 50us, the following high-voltage-level signal's length decide the bit is "1" or "0". See below figures:



Attention:

If signal from AM2302 is always high-voltage-level, it means AM2302 is not working properly, please check the electrical connection status.

7. Electrical Characteristics:

| Items | Condition | Min | Typical | Max | Unit |
|-------------------|-----------|-----|---------|-----|--------|
| Power supply | DC | 3.3 | 5 | 6 | V |
| Current supply | Measuring | 1 | | 1.5 | mA |
| | Stand-by | 40 | Null | 50 | uA |
| Collecting period | Second | | 2 | | Second |

8. Attentions of application:

(1) Operating and storage conditions

We don't recommend the applying RH-range beyond the range stated in this specification. The AM2302 sensor can recover after working in abnormal operating condition to calibrated status, but will accelerate sensors' aging.

(2) Attentions to chemical materials

Vapor from chemical materials may interfere AM2302's sensitive-elements and debase AM2302's sensitivity.

(3) Disposal when (1) & (2) happens

Step one: Keep the AM2302 sensor at condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;

Step two: After step one, keep the AM2302 sensor at condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Attention to temperature's affection

Relative humidity strongly depend on temperature, that is why we use temperature compensation technology to ensure accurate measurement of RH. But it's still be much better to keep the sensor at same temperature when sensing.

AM2302 should be mounted at the place as far as possible from parts that may cause change to temperature.

(5) Attentions to light

Long time exposure to strong light and ultraviolet may debase AM2302's performance.

(6) Attentions to connection wires

The connection wires' quality will effect communication's quality and distance, high quality shielding-wire is recommended.

(7) Other attentions

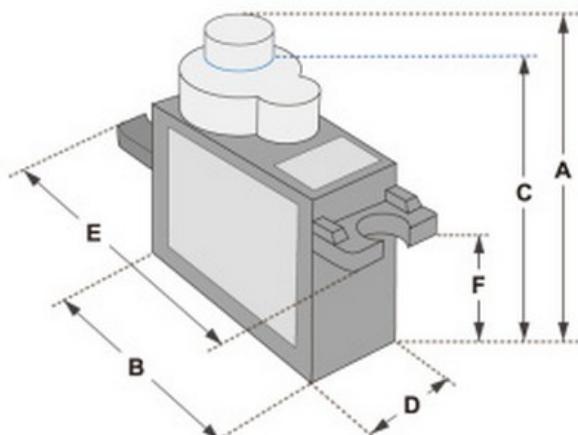
* Welding temperature should be bellow 260Celsius.

* Avoid using the sensor under dew condition.

* Don't use this product in safety or emergency stop devices or any other occasion that failure of AM2302 may cause personal injury.



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.



Dimensions & Specifications

| | |
|------------------|---------|
| A (mm) : | 32 |
| B (mm) : | 23 |
| C (mm) : | 28.5 |
| D (mm) : | 12 |
| E (mm) : | 32 |
| F (mm) : | 19.5 |
| Speed (sec) : | 0.1 |
| Torque (kg-cm) : | 2.5 |
| Weight (g) : | 14.7 |
| Voltage : | 4.8 - 6 |

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

