Diseño de un sónar ultrasónico

Sistemas Electrónicos Digitales

Miguel Ángel de la Iglesia Salido y Violeta Untaru G.I.E.A.I

Contenido

1.Introducción.

En este documento se abordará el diseño y construcción de un sónar ultrasónico. Como sabemos, la palabra sonar procede de la palabra inglesa 'sonar', la cual es un acrónimo de Sound Navigation And Ranging, cuya traducción español sería 'navegación por sonido'.

Actualmente, el sonar es uno de los métodos más utilizados para la localización, si bien su utilización se remonta a la Primera Guerra Mundial, debido a la necesidad fundamentalmente de los británicos, para la detección de submarinos en el campo de batalla.

A diferencia del radar que utiliza la propagación de ondas electromagnéticas, el funcionamiento del sonar se basa en la utilización de ondas sonoras para la localización, aspecto que le permite ser de gran utilidad tanto en medios terrestres como acuáticos.

Mayoritariamente, las frecuencias utilizadas en los sistemas sonar, oscilan desde las intrasónicas a las extrasónicas, de 20Hz a 20.000Hz, que justo abarcan la capacidad auditiva del oído humano. También se pueden utilizar frecuencias superiores, que dotan al sonar de mayor precisión pero menos alcance.

En el caso que nos atañe, el sonar se diseñará teniendo como objetivo desarrollar un sistema empotrado basado en el microcontrolador LPC1768 (Cortex M3), proporcionado por la empresa ARM. Además, el sistema debe cumplir con otra serie de funcionalidades las cuales, se explicarán más extensamente a lo largo del documento.

1.1.Funcionalidades.

A modo de resumen previo indicar que nuestro objetivo es que el sónar sea capaz de medir en un entorno de 180º, con la posibilidad de que este pueda ser controlado o bien de forma manual con los pulsadores KEY 1, KEY 2 o ISP, ya incluidos en la tarjeta, o bien mediante la utilización de programa con comunicación serie. Opcionalmente se intentará configurar una aplicación bluetooth que permita el control del sistema de forma inalámbrica.

El sistema además tendrá un modo de funcionamiento como detector de obstáculos en el que se podrá configurar un umbral de detección de manera que ante la presencia emitirá, a través de un pequeño altavoz, un tono cuya frecuencia será proporcional a la distancia de detección.

En la siguiente imagen podemos apreciar la interconexión de todos los dispositivos con el microcontrolador, todas estas conexiones son necesarias para obtener las funcionalidades pedidas.

Diagrama de bloques del sónar ultrasónico:

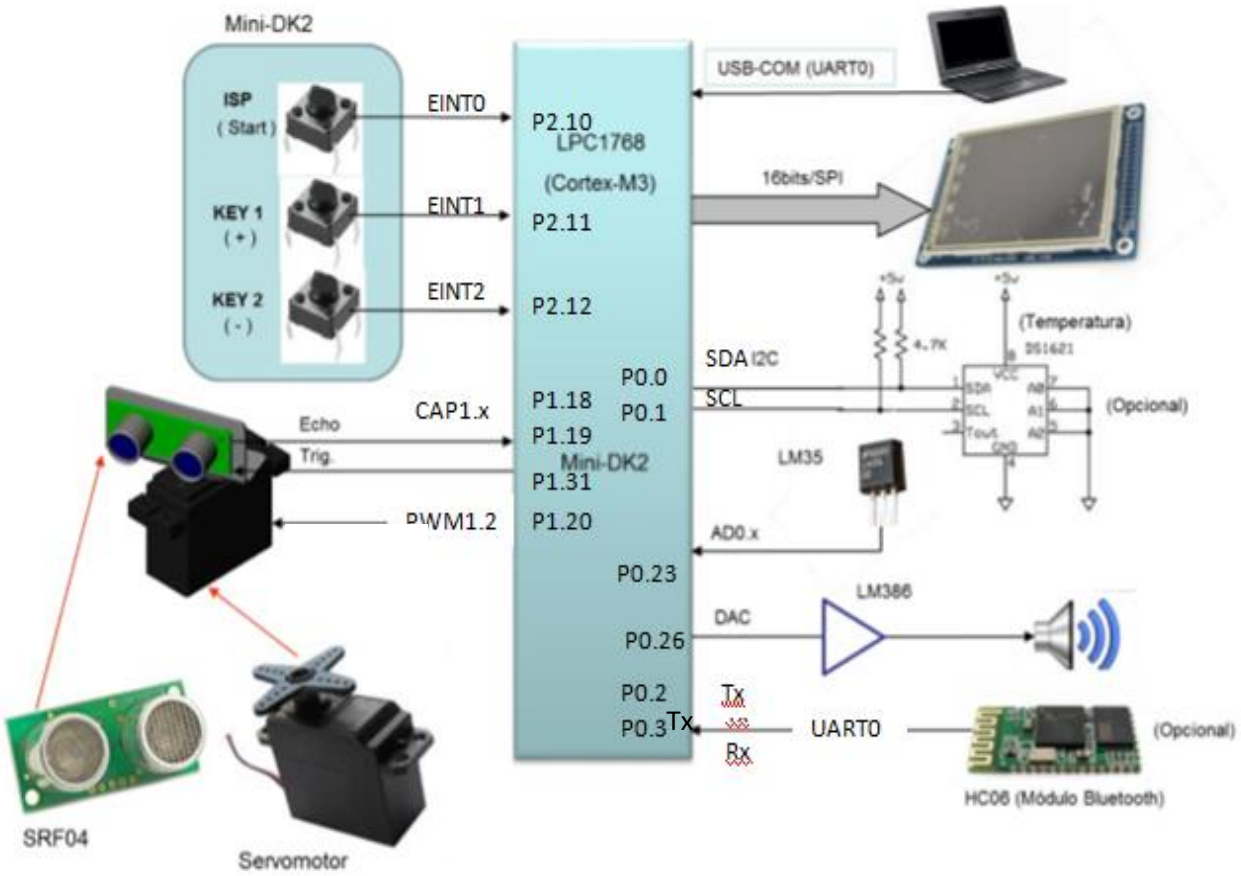


Figura 1. Diagrama de bloques del sónar ultrasónico

Las funcionalidades específicas requeridas en cada uno de los modos son:

1. **Modo manual**:
   Se utilizarán los pulsadores ya incluidos en la tarjeta (Key1 y Key2). Mediante la pulsación del pulsador Key1 se sumarán 10º a la posición actual en la que se encuentra el servo. De lo contrario, al pulsar el pulsador Key2, se restarán 10º. Una vez se alcance la orientación deseada, la pulsación del pulsador ISP permitirá obtener una medida. Si se habilita el modo en que se realicen medidas continuas, la pulsación de ISP iniciará el proceso. En este caso, tanto las medidas obtenidas, así como la posición del servo de mostrarán en el display cada 0.5 segundos.

2. **Modo barrido automático:**
   En este modo, el servo irá incrementando su posición 10º, cada 0.5 segundos. Esta posición oscilará entre 0º y 180º. En cada una de estas posiciones se mostrará por el display la medida efectuada.

3. **Modo detección de obstáculos:**
   A través de la comunicación serie, el sistema recibirá un umbral que oscilará entre 30cm. y 300cm, por defecto, el umbral inicial será de 100cm. Si se detecta un obstáculo por debajo de este umbral, el sistema emitirá un pitido desde el altavoz con una frecuencia proporcional al umbral. Esta frecuencia queda determinada por la siguiente ecuación:

   Frecuencia[Hz]= 5000-Umbral[cm] x 10

4. **Interfaz serie.**
   A través de la UART0, el usuario podrá comunicar al sistema mediante un menú y por tanto configurar, cada uno de los modos anteriormente citados, introducir el umbral deseado y visualizar las medidas obtenidas en formato ASCII.

2. Descripción hardware.

A continuación se detallarán cada uno de los dispositivos utilizados en la realización del sonar:

- **Servomotor de 9 gr.**

  Los servomotores son dispositivos cuyo funcionamiento se puede asociar al de un motor de corriente continua, con la característica de que estos son capaces de mantener una posición estable, siempre que esta se halle dentro de su rango de operación. Es posible, convertir un servomotor en un motor de corriente continua, si bien, el control quedaría ampliamente reducido.

  Para poder controlar la posición del servomotor, se utiliza la modulación por ancho de pulsos (PWM). La mayoría trabaja con una frecuencia de 50Hz, motivo por el cual, tendrán un periodo de 20 ms. En esta práctica, el servomotor utilizado tendrá un periodo de 20 ms.

  La posición del servo responderá al ancho de señal modulada proporcionada. En el datasheet de nuestro servomotor se especifica que ha de ser de entre 1ms y 2ms. En el caso de recibir un pulso de 1.5 ms, el servo se encontrará en la posición 0, en el caso de un pulso de 2 ms, se posicionará en 90º, y en el caso de recibir un pulso de 1 ms, se situará en una posición de -90º.

  A continuación podemos ver una imagen del servomotor utilizado:

Como podemos observar, posee tres cables conexionados, cada uno los cuales tendrá que ser conectado según la hoja de características:

- Cable rojo: A VCC.
- Cable marrón: A GND.
- Cable naranja: Será utilizado para el PWM.

En nuestro caso, la señal PWM utilizada es la **PWM1**, que estará conectada al **pin 1.20** de la tarjeta.

- **Ultrasonidos SRF04.**

Será el sensor utilizado para la medición de las distancias. Este sensor es capaz de detectar objetos en un rango de distancias comprendidas entre 3cm y 300cm. El funcionamiento de este sensor es sencillo, es capaz de determinar la distancia a partir del tiempo que tarda la onda sonora en retornar desde que fue emitida. Si partimos de la expresión de v=e/t, y sabemos que la velocidad del sonido son 340m/s, y sabemos el tiempo transcurrido (el tiempo transcurrido será el obtenido entre dos, debido a que el tiempo calculado es el que tarda en recibir el eco), la distancia puede ser fácilmente calculada.

A continuación podemos ver una imagen del ultrasonidos utilizado:

- El terminal 1 va a VCC.
- El termina 2 será el Trigger.
- El terminal 3 será el Echo.
- El terminal 4 va a GND.

Para su funcionamiento en la hoja de características se especifica que el ultrasonidos tiene que recibir una excitación a través del trigger. Esta excitación será creada con un

pulso de 10us según las especificaciones. En nuestro proyecto, el terminal **Trigger** estará conectado al **pin 1.31** de la tarjeta.

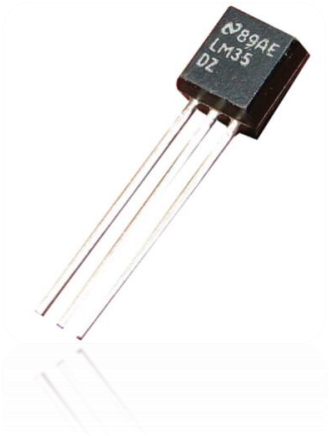
Una vez se ha emitido este pulso, el sensor estará a la espera de recibir el Echo, que determinará el tiempo que ha tardado en recibirse la señal sonara. En nuestro proyecto el terminal **Echo** estará conectado a los **pines 1.18 y 1.19.**

En apartados sucesivos, se explicará más en detalle el funcionamiento del ultrasonidos.

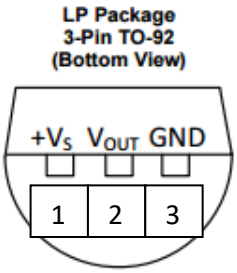
- **Sensor de temperatura analógico, LM35.**

Este sensor analógico modifica su tensión de salida en función de la temperatura medida, la cual puede encontrarse entre los -55º y 150º, si bien en nuestro proyecto no se contemplarán las medidas negativas, ya que para ello se necesitarían una resistencia externa y una alimentación negativa. La variación de su tensión responderá a 10mV por grado centígrado.

A continuación podemos ver una imagen del sensor utilizado:



Como podemos ver, el sensor tiene tres terminales:

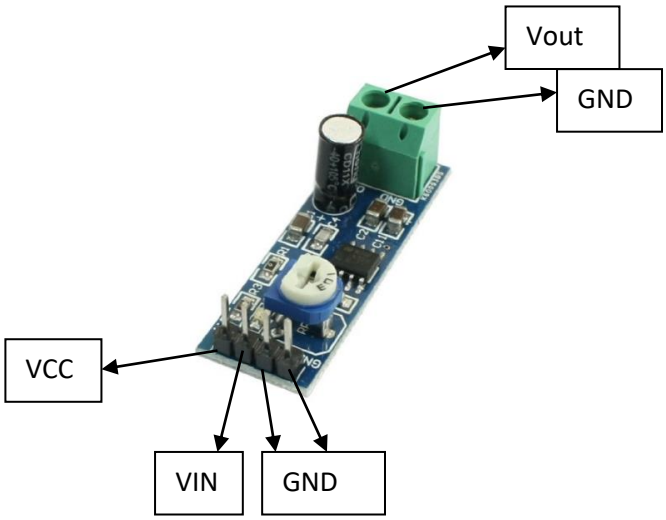


El terminal 1 estará conectado a VCC, el **terminal 2** al **pin 0.23** de la tarjeta pin y el terminal 3 a GND.

- **Amplificador, LM386.**

Este dispositivo será necesario para amplificar la señal senoidal procedente del DAC. Para que el altavoz sea capaz de emitir sonido, este tiene que recibir una potencia mínima, que la tarjeta por sí sola, es incapaz de dar.

A continuación podemos ver una imagen del sensor utilizado, con las conexiones necesarias para su funcionamiento:



La señal **Vin** que recibe el amplificador será la procedende del **DAC**, que en nuestro proyecto se emite a través del **pin 0.26.**

- **Altavoz.**

El altavoz utilizado para ampliar la señal Vout dada por el amplificador es el siguiente:

El altavoz posee dos terminales: Terminal +, conectado a Vout y Terminal -, conectado

- **Display, HY28A.**

Este módulo de visualización está formado por un display TFT de color, de 2.8", táctil y con una resolución de 320x240 pixeles basado en el controlador ILI9320/5.

Para su control, se ha utilizado una interfaz SPI, que nos permite una utilización menor de pines de la tarjeta, pero la comunicación es más lenta.

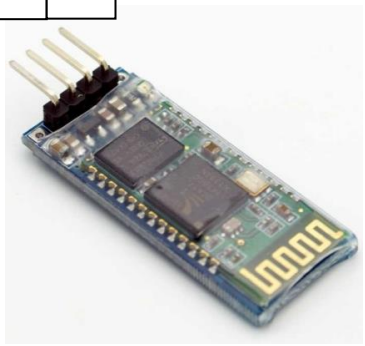
- **Módulo Bluetooth.**

El módulo que se ha utilizado es el HC06. Este módulo permite conectar mediante conectividad bluetooth la tarjeta mini DK2 con un dispositivo móvil, en nuestro caso con sistema operativo Android.

El proyecto podrá ser controlado desde el mismo terminal móvil, mediante un menú similar al utilizado para controlar el proyecto desde la UART.

A continuación se muestra una imagen del módulo HC06:

1	2	3	4



Como se puede observar el módulo presenta 4 terminales, atendiendo al orden establecido, la conexión de cada uno de estos terminales es la siguiente:

Terminal 1: Se conectará a +5V.

Terminal 2: Se conectará a GND.

Terminal 3: Correspondiente al terminal Tx (transmisión) el cual irá conectado al Rx (recepción) de la placa, **pin P0.3**.

Terminal 4: Correspondiente al terminal Rx (recepción) el cual irá conectado al Tx (transmisión) de la placa, **pin P0.2**.

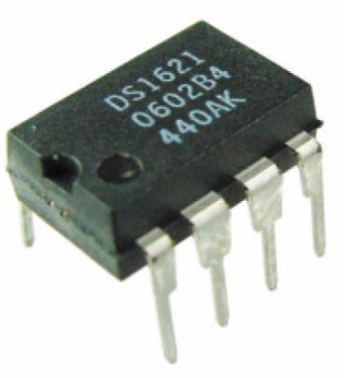
Para finalizar la conexión entre la placa y el dispositivo móvil, es necesario un aplicación que soporte este tipo de comunicación. La aplicación elegida es Bluetooth Terminal, desarrollada por Qwerty, la cual puede obtenerse por todos los dispositivos android desde Google Play Store.

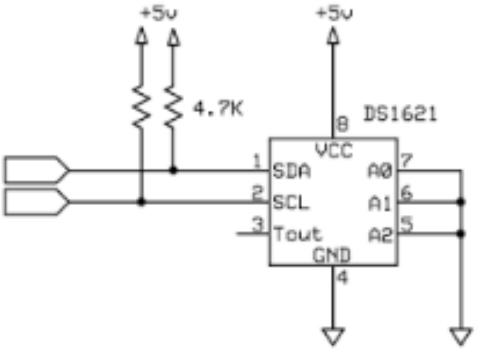
- **Sensor de temperatura digital DS1621.**

En este proyecto, además de la utilización de un sensor de temperatura analógico, el LM35, también se ha utilizado un sensor digital, el cual nos proporcionará la temperatura en un rango de valores que pueden oscilar desde -55º a 125º. Este sensor utiliza lecturas de temperaturas de 9 bits en C2 y se mostrarán con una resolución de

0.5º. Al ser un chip integrado, el número de patillas que han de conexionarse aumenta considerablemente con respecto al resto de dispositivos utilizados.

A continuación, se muestran una imagen del sensor utilizado y a su derecha un esquemático de su interconexionado.





Como podemos observar el encapsulado presenta 8 patillas las cuales siguen la siguiente conexión:

Patilla 0: Conectado a +5V.

Patilla 1: Conectado al **P0.0** de la placa.

Patilla 2: Conectado al **P0.1** de la placa.

Patilla 3: Esta patilla no se utilizará en el proyecto. Tiene una funcionalidad como termostato que no se requiere en la práctica.

Patilla 4: Conectado con GND.

Patilla 5: A0, estará conectada a GND.

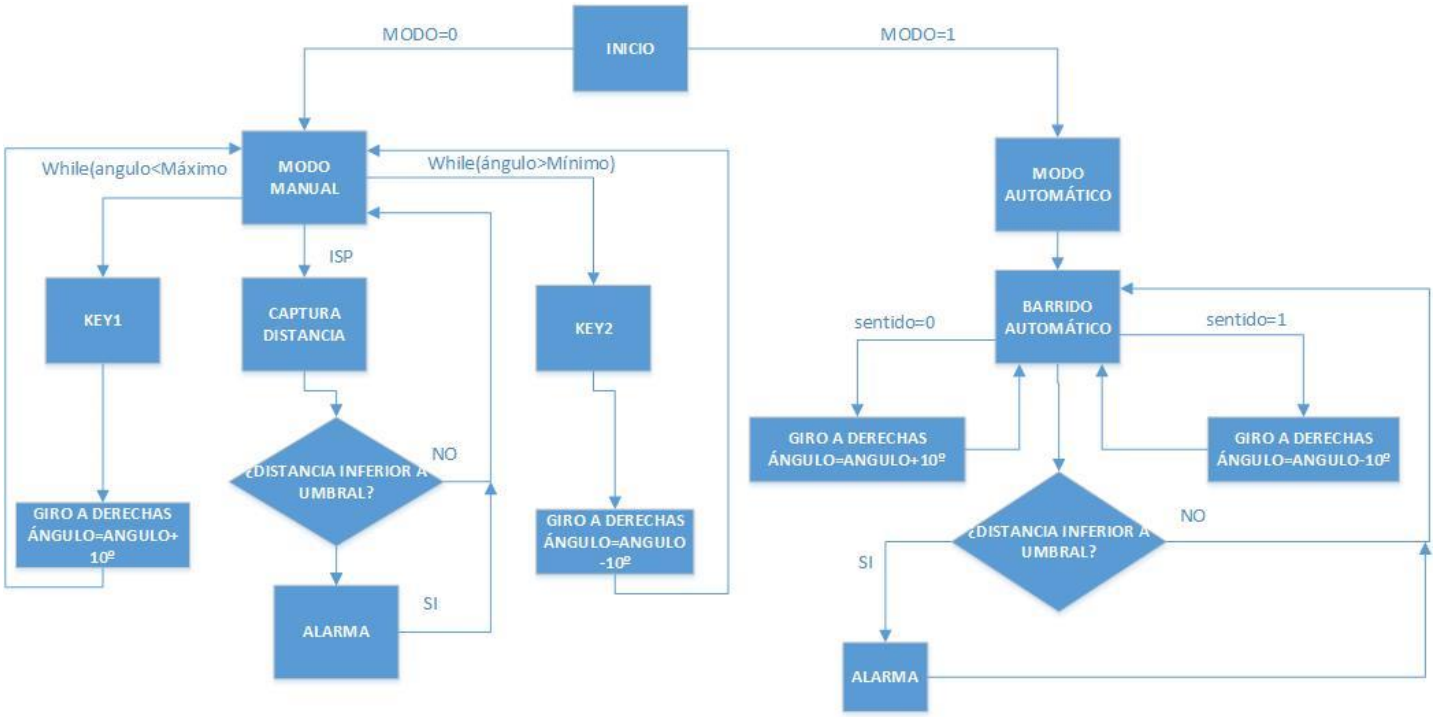
Patilla 6: A1, estará conectada a GND.

Patilla 7: A2, estará conectada a GND.

Las tres últimas patillas conectadas a masa, son las que permiten la conexión y la selección del dispositivo con la placa mediante la comunicación I2C que explicaremos en apartados posteriores. La existencia de estas tres terminales nos daría la posibilidad de conectar hasta 8 dispositivos distintos, aspecto que no se requiere en esta práctica.

3. Descripción software.

3.1. Modelado del comportamiento.



3.2. Descripción de cada uno de los módulos .c del proyecto.

Nuestros programas en KEIL trabajan la programación en c, debido a esto nuestros ficheros que se ejecutaran tendrán una extensión .c.

En la práctica hemos añadido o bien creado distintos ficheros .c. Uno de ellos es el fichero que contiene el código del programa que es cargado en la placa creado por nosotros mismos en el podemos encontrar todo el código que aparece en el punto 4, declaración de librerías, referencias a otro fichero como por ejemplo uart.c, lcddriver.c o i2c_lpc17xx.c.

Dentro del fichero uart.c viene definido la velocidad de transmisión de la UART, el modo en el que se trasmitirá con la selección de los pines, junto con todo el proceso que realiza la UART que vendrá explicado en los pasos siguientes. Mediante la uart no comunicaremos con el nuestros dispositivo Bluethood.

En el fichero lcddriver.c están definidas las librerías para poder visualizar en la pantalla de nuestra placa las medidas realizadas, también contiene información de cómo debemos colocar los datos para visualizarlos según las características, como los pines que transfieren dicha información.

El módulo i2c_lpc17xx.c contiene toda la información para la trasferencia de datos a través del puesto serie. El pin del pulso de reloj con su retardo, la configuración de los SDA tanto de entrada como de salida, el bit de START y el byte que envía al slave.

La igual que todos los dispoditivos que conectemos tendremos que seguir un protocolo que hará el correcto funcionamiento del dispositivo.

3.3. Descripción de las funciones del proyecto.

Funciones para la configuración del servomotor.

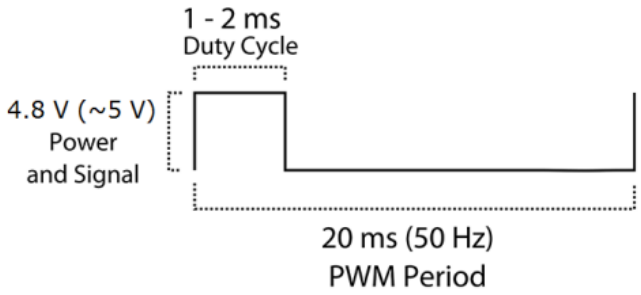
PWM1, SET SERVO, CONFIG TIMER 2 Y HANDLER TIMER2

Según las especificaciones de la práctica, se pide que el servomotor funciones en dos modalidades:

- **Modo manual**: El servomotor permanecerá parado, hasta que se pulsen los pulsadores Key2 y Key1, que restarán o sumarán respectivamente a la posición actual del servo (inicialmente se ha configurado 90º) 10º.
- **Modo automático:** Se realizará un barrido desde 0º a 180º y de 180º a 0º, con giros de 10º, los cuales se ejecutarán cada 0.5 segundos.

En primer lugar configuraremos la Pwm, que será la responsable del movimiento del servo mediante la generación de pulsos. En función de la anchura de estos pulsos, el servo se desplazará a una determinada posición en 0º y 180º.

En la hoja de características del fabricante podemos observar el funcionamiento del servo en función del ancho del pulso.



En el caso de recibir un pulso de 1.5 ms, el servo se encontrará en la posición 0, en el caso de un pulso de 2 ms, se posicionará en 90º, y en el caso de recibir un pulso de 1 ms, se situará en una posición de -90º.

Para conseguir estás especificaciones configuraremos la pwm1 con un ciclo de trabajo de 20ms.

```
void config_pwm1(void){

        LPC_PINCON->PINSEL3|=(1<<9);       // Pin P1.20 salida de PWM (PWM1.2).
        LPC_SC->PCONP|=(1<<6);           //Power on.
        LPC_SC -> PCLKSEL0 |= 1<<3;      //FCCLK/2
        LPC_PWM1 -> PR = 49;             //Prescaler 49+1=50;
        LPC_PWM1 -> MR0 =20000;          //Ts=20ms, 1pulso-->0.000001s, 0.02s-->20000pulsos.
        LPC_PWM1->PCR|=(1<<10);            //Configurado el ENA2 (1.2)
        LPC_PWM1->MCR|=(1<<1);             //Reset en MR0.
```

```
        LPC_PWM1->TCR|=(1<<0)|(1<<3);    //Contador habilitado y el modoPMW está
                                           habilitado.


        }
```

Como vemos la salida pwm está configurada con el pin de salida 1.20 como se indicó en la descripción hardware.

```
void set_servo(float grados){  //Esta función permitirá situar el
                  servo en un ánguo determinado.

        LPC_PWM1->MR2=((grados*50)/9)+300;   //Se traduce el valor del
                                               ángulo a pulsos.
        LPC_PWM1->LER|=(1<<2)|(1<<0);        //Se resetea el valor de MR0
                                               y MR2.
}
```

Con el MR2 estamos indicando al servo la posición deseada. El valor que se ha establecido en el MR2, se debe a un regla de tres calculada experimentalmente para poder fijar correctamente el valor de 0º. En un primer momento este valor se calculó con una interpolación entre 0º cuyo valor era 1ms, y 180º cuyo valor era de 2ms, según la hoja de características. Sin embargo, este cálculo no se ajustó con la realidad.

Una vez se ha configurado el movimiento del servo, ahora continuaremos para establecer que el movimiento del servo se realice cada 0.5 segundos, que es lo que se requiere en el modo automático.

Para ello, configuraremos el timer 2, que se muestra a continuación:

```
void init_TIMER2(void){

        LPC_SC->PCONP|=1<<22;           //Power on, para alimentar.
        LPC_SC->PCLKSEL1|=1<<12;        //Lo que configuramos aqui es el divisor de la
                                          señal. Pclk/2.
        LPC_TIM2->PR=49;               //Configuramos el preescaler 49+1=50.
        LPC_TIM2->MR1=500000;          //1pulso-->0.000001, para obtener 0.5 segundos-->
                                         500000pulsos.
        LPC_TIM2->MCR|=(1<<3)|(1<<4);        //Interrupción en MR1 y Reset en MR1.
        NVIC_SetPriority(TIMER2_IRQn,2);     //Configuramos la prioridad del timer.
        NVIC_EnableIRQ(TIMER2_IRQn);         //Habilitamos la prioridad.
        LPC_TIM2->TCR|=(1<<0);                //Habilitamos el timer.

}
```

Como deseamos que el timer ejecute una serie de sentencias cada 0.5 segundos, hemos configurado el MR1 como interrupción cada 500000 pulsos. (1pulso equivale a 1us).

A continuación, mostraremos el código que se ejecuta cada vez que salta la interrupción del timer 2:

```
void TIMER2_IRQHandler(void){

        if((LPC_TIM2->IR & (1<<1))==(1<<1))    //MR1
        {
                LPC_TIM2->IR|=(1<<1);          //Flanco de interrupción para MR1.
                LPC_ADC->ADCR|=(1<<24);        //Se inicia la conversión del ADC.
                leerTemperatura();             //Se lee la temperatura del DS1621.
                printString();                 //Se imprimen los valores por el LCD.


                if(modo==AUTOMATICO && pulsaModo==1){      //Si el sistema está en modo
                                                             barrido y se pulsa la Eint0:
                LPC_GPIO1->FIOPIN |= 0x80000000;       //Escribimos un 1 en el pin 1.31
```

```
                LPC_TIM0->TCR = 0x01;                   //Activamos el timer 0.

                //Esto permitirá realizar medidas constantes.

                angulo+=(INCREMENTO_SERVO)*sentido;  //Incrementamos o decrementamos 10°
                                                      en función del giro a izquierdas o
                                                      derechas del servo.
                set_servo(angulo);                   //Mandamos esa posición al servo.

                        if(angulo==ANGULO_MAXIMO)     //Si el servo se encuentra en 180°
                                sentido=DESCENDENTE;  //Decrementará su valor.
                        else if(angulo==ANGULO_MINIMO)//Si el servo se encuentra en 0°
                                sentido=ASCENDENTE;   //Aumentará su valor.
                }

                else{   //Si el sistema está en modo manual y se pulsa Eint0:

                        if(pulsadorkey2) //Si se pulsa el pulsador controlado por la Eint2
                                {
                        if (angulo>ANGULO_MINIMO){//Mientras el angulo sea mayor al mínimo
                        angulo-=INCREMENTO_SERVO;     //Decrementa
                        set_servo(angulo);}           //Envía posición al servo.

                        pulsadorkey2=0;  //Inicializamos a cero por si se vuelve a pulsar
                                           la EINT2.

                                }
                else if(pulsadorkey1){//Si se pulsa el pulsador controlado por la Eint1.
                        if (angulo<ANGULO_MAXIMO){   //Mientras el angulo sea inferior al
                                                     máximo
                        angulo+=INCREMENTO_SERVO;     //Aumenta
                        set_servo(angulo);            //Envía posición al servo.
                        }
                        pulsadorkey1=0;//Inicializamos a cero por si se vuelve a pulsar la
                                         EINT1.
                                              }
                        }
        }

}
```

Como aspectos importantes en el código anterior, indicar algunas cosas:

En primer lugar, para cambiar de modo manual a modo automático, la variable modo, se modificará, o bien por la pestaña Watch1, o mediante los medios de comunicación serie, ya sea con la UART o mediante el bluetooth. Únicamente si está variable se ha modificado, se puede pasar de modo barrido a manual.

En segundo lugar, este timer controla principalmente el movimiento del servo cada 0.5 segundos. Como queremos que cada vez que se ejecute el movimiento se imprima una medida, inicializamos el pin 1.31 a uno para enviar el estímulo al ultrasonidos (inicialmente este pin siempre se encuentra a cero), y activamos el timer. De esta manera la captura de la distancia es continua.

Por último, comentar que en esta función, también se manda la sentencia que nos permite iniciar la conversión del ADC. Creemos adecuado introducir esta línea aquí, ya que queremos que la temperatura se actualice y se visualice (como el resto de variables: distancia, ángulo y temperatura del DS16121) cada 0,5 segundos.

Funciones para la configuración de la UART.

La UART (Universal Asynchronous receiver/transmitter) es un dispositivo electrónico hardware para la comunicación serie, con el cual, la forma de los datos y la velocidad pueden ser configurables.

En nuestro caso, existen cuatro módulos, que van desde la UART0 hasta la UART3. La transmisión (y la recepción) de los datos se puede realizar de manera directa a la memoria y con una velocidad máxima de 6Mbps.

El formato de comunicación está compuesto por: un bit de start que inicia la transmisión o recepción, datos que pueden ser de 5 a 8 bits, un bit de paridad y un bit de stop.

La transmisión está basada en un registro de desplazamiento, en el cual se introduce un dato y es el propio hardware el que añade un bit de inicio, el bit de paridad y el bit de stop.

Por otra parte, la recepción se realiza siguiendo el mismo modelo que la transmisión. Al detectar el bit de start, se inicia un muestro de la línea de recepción del dato, con una frecuencia de 16 veces a la que llegan los datos. El dato quedará fijado a partir del valor central que se haya tomado.

Todo sale al ritmo establecido por la configuración de la UART. La velocidad de la UART se mide en baudios y se puede configurar de la siguiente manera:

$$UART = \frac{Fpclk}{16 * UnDl_{16} * Fr}$$

A continuación, incluimos el código utilizado para la configuración y utilización de la UART.

En primer lugar incluimos el documento **uart.c**, gracias a este documento la interrupción de la uart ya está configurada (con un prioridad de 0), lo único que se tendrá que añadir en nuestro main.c será aquellas sentencias que configuren la velocidad de transmisión, inicializar el buffer donde se almacenarán nuestros datos, y programar aquellas cadenas de texto que permitan crear el menú pedido por la práctica. Todos estos puntos son de gran importancia, puesto que la configuración de la UART permitirá el uso del bluetooth.

```
// Variables UART
char buffer[200];      // Buffer de recepción de 200 caracteres
char *ptr_rx;          // Puntero de recepción
char rx_completa;      // Flag de recepción de cadena que se activa a "1" al recibir la
                          tecla return CR(ASCII=13)
char *ptr_tx;          // puntero de transmisión
char tx_completa;      //Flag de transmisión de cadena que se activa al transmitir el
                          caracter null (fin de cadena).
```

Se ha configurado el tamaño del buffer con una dimensión de 200, ya que con una dimensión de 30, se imprimían valores que pertenecían a otros sectores de la memoria.

```
//Configuración de la Uart.
int op;              //Se crea una variable que nos permitirá guardar los valores
                       introducidos por la UART.
ptr_rx=buffer;       //Inicializa el puntero de recepción al comienzo del buffer.
uart0_init(9600);  //Se configura la velocidad de la UART a 9600 baudios.
```

Hemos configurado la UART con una velocidad de 9600 baudios, ya que con la preestablecida de 19200 baudios, el bluetooth no funcionaba correctamente.

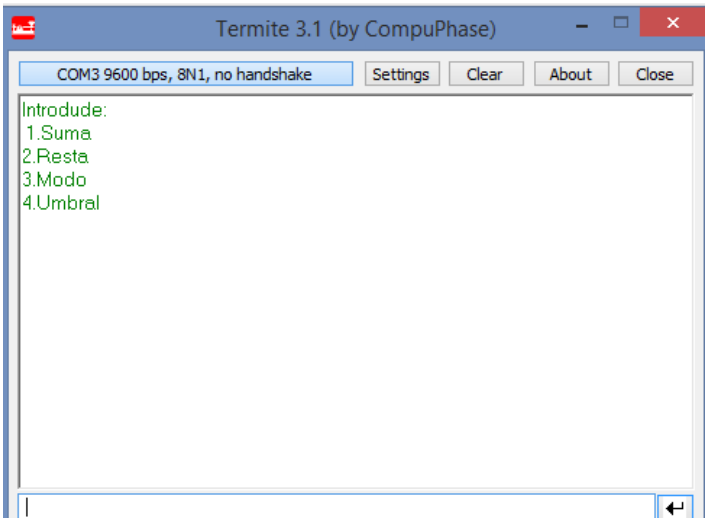
El menú que nos permitirá controlar el proyecto es el siguiente:

```
//Se transmite la cadena de caracteres.
tx_cadena_UART0("Introdude:\n 1.Suma\n2.Resta\n3.Modo\n4.Umbral\n\r");
	while(1){

		/*
//Estas líneas de código permitirán conectarse a MATLAB.

		sprintf(buffer,"%d %3.2f\n", angulo, distancia);
		tx cadena UART0(buffer);
		while(tx completa==0);
		tx_completa=0;
		*/

		if(rx_completa){
					//Si se ha finalizado la transmisión.
		rx_completa=0;
					//Se inicializa rx=0, para nuevas transmisiones.
		opcion= atoi(buffer); //Se guarda en el buffer la opción señalada.

			switch(opcion){
				case 1:
				//Se incrementa en 10° el angulo del servo.
					if (angulo < ANGULO_MAXIMO ){
						angulo += INCREMENTO_SERVO;
						set_servo(angulo);
					}
					break;

				case 2:
				//Se decrementa en 10° el angulo del servo.
					if (angulo > ANGULO_MINIMO ){
						angulo -= INCREMENTO_SERVO;
						set_servo(angulo);
					}
					break;

				case 3:
				//Cambio de modo manual automático.
					modo = !modo;
					break;

				default :
				//Se establece el nuevo umbral.
					if(opcion>=30 && opcion<=300){
	//Si el umbral no se haya entre esos valores no se actulizará.
					umbral=opcion;
					}
				break;

			}

//En el caso de que la distancia sea menor o igual a 300:
			if(distancia<=300){
//Se mostrarán los valores pedidos.
	sprintf(buffer,"Grados: %d\nDistancia: %f\nUmbral: %d\n\r", angulo, distancia,
umbral);	} //Si no
			else{	//Se mostrará esta cadena.

	sprintf(buffer,"Grados: %d\nDistancia: Fuera de rango\nUmbral: %d\n\r", angulo,
umbral);			}

tx_cadena_UART0(buffer);while(tx_completa==0); tx_completa=0;	//Se transmite la
                                                                cadena anterior.
tx_cadena_UART0("Introdude:\n 1.Suma\n2.Resta\n3.Modo\n4.Umbral\n\r"); //Se vuelve a
transmitir la cadena inicial.

}
```

Cabe destacar que para que se pueda reproducir en MATLAB el documento de *radar.m* es necesario que la UART se comente y el código siguiente se descomente.

```
sprintf(buffer,"%d %3.2f\n", angulo, distancia);
tx cadena UART0(buffer);
while(tx completa==0);
tx_completa=0;
```

Por último, señalar que, el programa que nos permitirá utilizar la comunicación serie es el termite, propuesto por la asignatura. A continuación se muestra una captura del menú configurado:



Funciones para la configuración del LCD.

Esta función permitirá ir visualizando los valores que deseemos en la pantalla LCD de nuestra tarjeta mini-DK2. Para configurar correctamente las sentencias que se utilizan en esta función es necesario incorporar a nuestro proyecto el archivo **lcddriver.c**, ya que en él se encuentran definidas dos de las sentencias que se utilizarán, que son las siguientes:

fillScreen: Permite pintar la pantalla de algún color.

drawString: Permite escribir en pantalla aquellos caracteres que se han almacenado previamente en el array con sprintf. Además nos permitirá situar estos caracteres allí donde deseemos y configurar su color y tamaño.

Por otra parte, explicaremos brevemente la función sprintf, utilizada también en el código adjunto.

Esta función nos permitirá guardar en un array la cadena de caracteres que se desee, además de variables utilizadas en el programa. En nuestro caso, todas las variables utilizadas (distancia, angulo, temperatura, temperature), son variables globales.

El código de la función utilizada es el siguiente:

```
void printString(){
```

```
                fillScreen(BLACK); //Pinta en un primer momento la pantalla de negro.

                //Mostrar distancia, esta línea mostrará por la pantalla la distancia.
                if(distancia<=300){ //Si la distancia es inferior a 300cm, se mostrará la
distancia con su valor.
                sprintf(linea,"Distancia: %3.2f cm",distancia); //Escribe.
                drawString(5, 20, linea, YELLOW, BLACK, MEDIUM);    //Configura posición
del texto, color y tamaño.
                }
                else{ //En caso contrario, si la distancia supera los 300cm. se imprimirá
el mensaje fuera de rango.
                sprintf(linea,"Distancia fuera de rango"); //Escribe.
                drawString(5, 20, linea, YELLOW, BLACK, MEDIUM);    //Configura posición
del texto, color y tamaño.
                }

                //Mostrar ángulo, esta línea mostrará por la pantalla el ángulo actual.
                sprintf(linea,"Angulo servo: %d grados",angulo); //Escribe.
                drawString(5, 50, linea, YELLOW, BLACK, MEDIUM); //Configura posición del
texto, color y tamaño.

                //Mostrar temperatura, esta línea mostrará por la pantalla la temperatura
medida por el LM35.
                sprintf(linea,"Temperatura %3.2f",temperatura);  //Escribe.
                drawString(5, 80, linea, YELLOW, BLACK, MEDIUM); //Configura posición del
texto, color y tamaño.

                //Mostrar temperatura I2C, esta línea mostrará por la pantalla la
temperatura medida por el DS1621.
                sprintf(linea,"Temp.I2C %3.2f",temperature); //Escribe.
                drawString(5, 110, linea, YELLOW, BLACK, MEDIUM); //Configura posición
del texto, color y tamaño.

}
```

Como se observa, en el momento en el que se llame a la función printString, la pantalla representará los valores de la distancia, el ángulo actual en el que se encuentra el servo, y las temperaturas obtenidas analógicamente por el sensor LM35 y digitalmente por el sensor DS1621.

Para inicializar el controlador del LCD se llamará a la función lcdInitDisplay en la función main del programa.

Funciones para la configuración del BLUETOOTH.

Para la comunicación entre la tarjeta mini-Dk2 y el dispositivo móvil mediante conectividad bluetooth, se ha utilizado la aplicación Bluetooth Terminal.

Para facilitar la interfaz de comunicación se ha optado por utilizar el menú previamente configurado en la UART0. El menú que se mostrará por pantalla tendrá exactamente las misma funcionalidad y el mismo modo de uso que el establecido para la UART0.

La única ampliación que hemos hecho respecto al hardware previo, es la conexión de los terminales Tx y Rx (transmisión y recepción) del módulo HC06 con los terminales Rx y Tx de la tarjeta respectivamente. Los pines utilizados son los establecidos en el manual de Cortex-M3 para la UART0y son: para Tx=P0.2 y para Rx=P0.3

Funciones para la configuración del I2C.

El objetivo de la configuración de este tipo de comunicación, será obtener a partir del sensor DS1621 la temperatura ambiente actual del sitio en el que nos localicemos.

Como se explicó anteriormente, este sensor mide la temperatura digitalmente mediante el uso de 9 bits, por lo tanto, la utilización del tipo de comunicación I2C, se presenta de manera adecuada ya que su funcionamiento se basa en la transmisión de bits en serie.

El funcionamiento del I2C se basa en un bus direccional de dos cables: SDA (es el responsable de la transmisión serie de los datos) y el SCL (es el reloj que controla la transmisión).

La configuración del I2C permite atribuir a cada dispositivo conectado un rol: estos pueden ser master, slave o master/slave. En nuestro caso el controlador funcionará como maestro y el sensor con esclavo. Existe una dirección para cada dispositivo que se conecta; los primeros bits vienen establecidos de fabrica, y los únicos que podremos cambiar son A2,A1 y A0. En nuestro caso, estos tres pines tendrán un valor de 0.

El I2C sigue un protocolo de comunicación que arranca con la condición de START (controlado por el master). Los dos cables de los que hablábamos anteriormente, SDA y SCL, estarán inicialmente a 1, gracias a la conexión pull-up realizado en el circuito.

A continuación del bit de START, le sigue la dirección del esclavo, y después le sigue el bit de READ o NOT-WRITE, que nos permitirá elegir si la comunicación va a ser de lectura o de escritura.

En este momento, se lanza los datos en paquetes de 8 bits. Una vez, el maestro o el esclavo hayan recibido los datos, se debe escribir en el bit ACK. En caso de que este bit tenga un valor de 0, significará que la transmisión ha sido correcta. La transmisión finaliza con la condición de STOP.

Por último, antes de comenzar con las funciones del proyecto, hablaremos del registro de control y operación, el cual viene definido por:

MSb	Bit 6	Bit5	Bit 4	Bit 3	Bit 2	Bit 1	LSb
DONE	THF	TLF	NVB	X	X	POL	1SHOT

- El bit DONE adquirirá un valor de 1 en el momento en el que la conversión este completa, mientras se este procesando el valor será de 0.
- El bit THF, adquiere un valor igual a 1 cuando la temperatura es igual o mayor a TH.
- El bit TLF, adquiere un valor igual a 1 cuando la temperatura es igual o menor a TL.
- NVB, que indica el flag de memoria no volatil ocupada.
- POL, es un bit no volátil, que indica la polaridad de la salida.
- 1SHOT, es un bit no volátil, cuyo valor (si es 1) iniciará la conversión de temperatura por el DS1621.

A continuación expondremos las funciones utilizadas para la configuración del I2C. En primer lugar, indicar que para su funcionamiento, es necesario la incorporación del archivo **i2c_lpc17xx.c.**

En este documento, se definen algunas de las funciones que utilizaremos a posteriori, son las siguientes:

- **void I2Cdelay(void):** Retardo mínimo de 4,7 us.
- **void pulso_SCL(void):** Genera un pulso de reloj.
- **void I2CSendByte(unsigned char byte):** Manda por SDA un bit controlado por el SCL. Cuando termia espera al bit ACK.
- **void I2CSendAddr(unsigned char addr, unsigned char rw):** Función que envía START más la dirección dirección del Slave (con bit LSB inicando R/W).
- **unsigned char I2CGetByte(unsigned char ACK):** Función para leer un Byte del Slave. El Master envía al final de la lectura el bit ACK que se pasa como argumento de la función.
- **void I2CSendStop(void):** Fin de la escritura o lectura con la condición de STOP.

El código de las funciones utilizadas es el siguiente:

```
void DS1621_config(void)  //Configuración del sensor DS1621.
{
I2CSendAddr(0x48,0);     //Dirección establecida para el DS1621
                          (A2,A1,A0 a GND)
I2CSendByte(0xAC);       //Se accede al registro de configuración.
I2CSendByte(0x02);       //Modo de conversión continua(1SHOT=0)
I2CSendStop();
I2CSendAddr(0x48,0);      //Dirección establecida para el DS1621
                          (A2,A1,A0 a GND)
I2CSendByte(0xEE);        //Inicio de la conversión.
I2CSendStop();
}
```

La configuración del bit oneshot a cero nos permitirá una conversión continua, sin necesidad de enviar continuamente el bit de START. Si bien, la cantidad de energía utilizada será mayor.

A continuación, incluimos la función que se ha utilizado para leer la temperatura.

```
void leerTemperatura(){

 char temp;                    //Declaración de variable local.
I2CSendAddr(0x48,0);           //Escritura (r/w=0).
I2CSendByte(0xAA);             //Comando de lectura de Tª.
I2CSendAddr(0x48,1);           //Re-Start, para lectura (r/w=1).
temp=I2CGetByte(0);            //Lectura de byte MSB.

//Con las sentencias siguientes podremos establecer la precisión.
if(I2CGetByte(1)==0) temperatura=(float)temp; //Si el  bit7=0 del byte LSB la variable
                                                permanece constante..
else temperature=(float)(temp)+0.5;           // Si el  bit7=1 del byte LSB se añade a
                                                la variable +0.5°.
I2CSendStop();

}
```

En primer lugar se inicia la comunicación, con carácter de escritura, indicando la dirección establecida para el sensor. A continuación, se accede a la lectura de la temperatura. Se realiza un re-start, para poder acceder a la lectura del dato. Y por último, se obtiene el dato.

Para entender la comunicación que existe entre placa y sensor a continuación se incluye a modo de resumen el esquema del sistema de comunicación.

1.Inicio de la conversión:

START	1001 (Fábrica)	0 0 0 (A2A1A0)	0 (r/w)	A		A	P

2.Configuración de DS1621_config():

START	1001 (Fábrica)	0 0 0 (A2A1A0)	0 (r/w)	0 (r/w)	A	0xAC	A	DATO	A	P

3.Configuración de leerTemperatura():

Escritura:

START	1001 (Fábrica)	0 0 0 (A2A1A0)	0 (r/w)	A	Re-START	1001 (Fábrica	0 0 0 (A2A1A0)

Lectura:

1 (r/w)	MSB	0	LSB	1	P

Funciones para la configuración del ADC.

La funcionalidad del ADC es convertir una señal analógica en una digital. Nuestro microprocesador LPC1768 tiene un convertidor de 12 bits, con 8 entradas multiplexadas y un canal global, dependiendo de número de canales que queramos convertir seccionamos el modo Global (para un solo canal) o el modo Burst (para más de un canal), el ADC tiene un tiempo mínimo de conversión con 65 ciclos de reloj, frecuencia de conversión a de ser menor que 12.5MHz. El inicio de la conversión se puede llevar a cabo de varios modos, por flanco, por una señal interna del programa, o una señal externa.

Nosotros emplearemos el ADC para leer la temperatura exterior desde un sensor LM35. Este sensor dispone de 3 patillas uno para la alimentación, masa o GND y finalmente otra para AD0.0 que ira conectada a la patilla P0.23. El ADC a parte de su función de configuración necesita un Handler y un Timer que se comentarán más tarde.

Primero tendremos que alimentar el ADC, esta alimentación se hará con la el registro **PCONP** correspondiendo el pin 12 a nuestro ADC. El segundo paso será la habilitación como entrada de los pines, en nuestro caso solo hay uno AD0.0. Al configurar el ADC tenemos que tener cuidado para no olvidarnos de deshabilitar las resistencias de PULL UP Y PULL DOWN mediante el registro **PINMODE**. El registro ADCR será empleado para el Canal 0, la frecuencia de 12.5

MHz y que empiece cuando se produce la interrupción del TIMER 2. En las siguientes líneas habilitaremos las interrupciones.

```
void configADC(){

LPC_SC->PCONP|=(1<<12); //  LO UTILIZAMOS PARA ACTIVAR EL ADC. POWER ON.
LPC_PINCON->PINSEL1|=(1<<14); // ENTRADA DEL ADC SERA P0.23 (ADO.0).
LPC_PINCON->PINMODE1|= (2<<14);  // DESHABILITA PULL DOWN.
LPC_ADC->ADCR|= (1<<0)|(1<<8)|(1<<21);//       Canal 0, CLKDIV=1, PDN=1,
EMPIEZA CON EL MATCH 1 TIMER 2.
LPC_ADC->ADINTEN=1; //      Habilitamos interrupción fin de conversión canal 0
NVIC_EnableIRQ(ADC_IRQn); //Habilitar interrupcion de ADC
NVIC_SetPriority(ADC_IRQn,2); // Establecer la prioridad

}
```

Para leer el dato que el ADC nos proporciona debemos acceder al registro ADDR0 esta operación de hará dentro de la función Handler con una frecuencia marcada por el TIMER2.

```
void ADC_IRQHandler(void)
{
       float voltios; //Variable local
       voltios= ((LPC_ADC->ADDR0>>4)&0xFFF)*3.3/4095;        //se borra
automat. el flag DONE al leer ADCGDR.
       temperatura=voltios*100; //Valor de la temperatura
}
```

El TIMER2 para su adecuado funcionamiento con el ADC tendremos primero que alimentarlo con el registro **PCONP,** para cuando se produce el TIMER tiene que hacer un Reset, una interrupción y un stop todas ellas incluidas en los primeros 3 bits de registro. El MR0 fijaremos la frecuencia, todos estos registro ya han sido declarados en el apartado de funciones para el configuración del servo.

Dentro del Timer Handler borramos el flag de la interrupción para que se pueda ejecutar la próxima vez que esté solicitado, y habilitamos el registro de control del ADC de la siguiente forma.

```
void TIMER2_IRQHandler(void){

       if((LPC_TIM2->IR & (1<<1))==(1<<1))    //MR1
       {
              LPC_TIM2->IR|=(1<<1);       //Flanco de interrupción para MR1.
              LPC_ADC->ADCR|=(1<<24);     //Se inicia la conversión del ADC.
```

Funciones para la configuración del DAC

El DAC es un convertidor digital analógico nosotros lo emplearemos para producir el pitido de más o menos frecuencia dependiendo de la distancia umbral a la que se encuentra. Las principales características de un DAC son 10 bit que puede convertir, con una arquitectura de cadena de resistencias, un buffer de salida, seleccionar la velocidad y la amplitud de salida y la frecuencias máxima a la que puede trabajar un DAC es de 1MHz.

Como dispositivo físico del DAC emplearemos el amplificador LM386 mencionado anteriormente este dispositivo ira conectado a un altavoz y con la ayuda de un potenciómetro que viene instalado sobre el variaremos el volumen del altavoz.

Para poder trabajar con el DAC tenemos que generar unas muestras pertenecientes a la función seno, que variaran con la amplitud y la frecuencia. Dichas muestras serán empleadas cuando la función de DAC sea ejecutada.

```
//Funcción que genera muestras en el DAC
int muestras[N_muestras];
int i;
void genera_muestras(uint8_t muestras_ciclo)
{
	uint8_t i;
	for(i=0;i<muestras_ciclo;i++)
		muestras[i]=1023*(0.5 + 0.5*sin(2*pi*i/muestras_ciclo)); //
Funcion que hace el seno,como son 10 bits de multiplica por 1023.
}
```

El DAC al igual que todo los dispositivos necesita una función de configuración, dentro de ella habilitamos el pin 0.26 perteneciente a AOUT, con ella lo alimentamos, mediante el registro **PINSEL**, a diferencia de los otros que no tenían modo DMA lo deshabilitamos.

```
void init_DAC(void)
{
	LPC_PINCON->PINSEL1|= (2<<20); // Pin de salida P0.26 (AOUT) para el DAC
	LPC_PINCON->PINMODE1|= (2<<20);  // Deshabilita pullup o pulldown
	LPC_SC->PCLKSEL0|= (0x00<<22);   // CCLK/4 (Fpclk después del reset)
	LPC_DAC->DACCTRL=0; //Modo DMA deshabilitado
}
```

Como modo de activación del DAC emplearemos el TIMER3, su función es conseguir el dato de la función 'genera_muestras' y pasarlas al amplificador.

```
void init_TIMER3(void)
{
	 LPC_SC->PCONP|=(1<<23); // Alimentamos TIMER3
    LPC_TIM3->PR = 0x00;
    LPC_TIM3->MCR = 0x03; // Reset TC on Match, and Interrupt MR0
    LPC_TIM3->MR0 =F_out; //Frecuencia dependiente del Umbral
    LPC_TIM3->EMR = (1<<1);  // Nivel alto ELBIRDE salida
    LPC_TIM3->TCR = 0x02;
    LPC_TIM3->TCR = 0x01; // Habilitar TIMER
    NVIC_SetPriority(TIMER3_IRQn,1);  // Habilitar prioridad
    NVIC_EnableIRQ(TIMER3_IRQn); // Habilitar interupcion

}
```

En la función TIMER3_Handler activamos el registro DACR y con el cogemos la muestra y la pasamos a nuestro amplificador. Debemos tener cuidado al final borrar el flag de la interrupción para que a la próxima vez pueda ejecutarse.

```
void TIMER3_IRQHandler(void)
{
	static uint8_t indice_muestra;
	LPC_TIM3->IR|= (1<<0); // Limpiar interrupcion de TIMER 3
	LPC_DAC->DACR= muestras[indice_muestra++] << 6; // El DAC se encuentra
entre el bit 6 y el bit 15
	indice_muestra&= 0x1F;     // Limpiamos para futuros recibos de
informacion
}
```

Funciones para la configuración del sensor ultrasonido

El dispositivo físico empleado como ya se ha mencionado en los apartados anteriores es SRF04, cuya funcionalidad es de tomar la distancia a la que se encuentra del obstáculo mediante el envió de un pulso. Esta distancia se medirá mediante el tiempo que tarda en ir y volver el pulso, dicha diferencia al multiplicarla por la velocidad del sonido obtendremos la distancia a la que nos encontramos. Para el correcto funcionamiento del sensor no debemos encontrarnos a una distancia superior a 3 metros.

El programa se ha realizado de la siguiente forma, empleamos el TIMER0 para mandar el pulso de 10µs a través del pin 1.31, para ello primero tendremos que declara como salida dicho pin.

```
void configurarPines(){
	// Declarar como salida el pin del pulso
	LPC_GPIO1->FIODIR |= 0x80000000; //pin 1.31
	LPC_GPIO1->FIOPIN &= ~0x80000000;
}
```

El TIMER0 nos marcará el flanco de subida y el de bajada y la duración de este 10µs, dicho TIMER se ejecutará siempre que esté en modo automático o cuando se accione el pulsado ISP estando en modo manual. Para la configuración del TIMER0 es el mismo modo que hemos comentado en el apartado de ADC.

```
void init_TIMER0(void){
	// Tiempo de interrupción en SG * 25e6 - 1
	LPC_SC->PCONP|=(1<<1); // Alimenta TIMER0
	LPC_TIM0->MCR = 0x07; // con 7 activo los 3 bits: STOP, Interrupción,
Reset
	LPC_TIM0->MR0 = (Fpclk * 1e-5)-1; //10micro segundos
	LPC_TIM0->TCR = 0; // Apagar TIMER
	NVIC_SetPriority(TIMER0_IRQn,2);  //Establecer prioridades
	NVIC_EnableIRQ(TIMER0_IRQn); //Habilitar interrupción
}

void TIMER0_IRQHandler(void) {
	LPC_TIM0->IR|= (1<<1);     //Borrar el flag de interrupción del TIMER0
  LPC_GPIO1->FIOPIN &= ~0x80000000;     //Limpiar el pin
	LPC_TIM0->TCR = 0; // Apagar timer0
}
```

Para medir la distancia a la que nos encontramos emplearemos el TIMER1 y con dos captures CAP1.0 y CAP1.0. El primero de ellos medirá cuando se encuentre con un flanco de subida y el segundo medirá cuando se encuentre con un flanco de bajada. Para la configuración del TIMER1 tendremos primero que alimentar mediante el registro **PCONP** poniendo a 1 el bit 2, establecemos con el registro **PINSEL3** la función de pin 1.18 y 1.19, establecemos cuando haya un flanco de subida y uno de bajada mediante interrupción, estas últimas se harán mediante el registro **CCR** todo ello se realizara poniendo a 1 los bits 0, 2, 4, 5.

```
void init_Timer1(){
	//Programar Capture
	LPC_SC->PCONP|=(1<<2); //Alimentamos el TIMER1
	LPC_PINCON->PINSEL3|=(3<<4)|(3<<6); //Establecemos la función de dicho
pin
	LPC_TIM1->PR=24;
	LPC_TIM1->CCR = (1<<0)|(1<<2)|(1<<4)|(1<<5);  //CAP1.0 flanco de subida|
interrupción| CAP1.1 flanco bajada| interrupción
	LPC_TIM1->TCR =(1<<0);    //Contador RESET
	NVIC_SetPriority(TIMER1_IRQn,2); //Establecer prioridad
	NVIC_EnableIRQ(TIMER1_IRQn);     //Habilitar interrupción
}
```

En la función TIMER1_IRQHandler desactivaremos la interrupción para que cuando vuelva a producirse pueda calcular dicha diferencia y con ello la distancia, la desactivación de la interrupción se hará mediante el registro IR.

La distancia mediante la función String se imprimirá por pantalla.

```
void TIMER1_IRQHandler(){
```

```
float dif_pulsos; //variable local

	LPC_TIM1->IR|=(1<<4);		//Reset captura de CR0
	LPC_TIM1->IR|=(1<<5);		//Reset captura de CR1
	dif_pulsos=((LPC_TIM1->CR1)-(LPC_TIM1->CR0)); //Calcular diferencia
	distancia=dif_pulsos*0.017;		// Distancia final

 }
```

La función init_TIMER0 e init_TIMER1 tienes que estar puestas dentro de la función main.

Funciones para la configuración de IRQS

En nuestro proyecto hemos empleado también las interrupciones EINT0, EINT1 y EINT2, cada una de ellas tiene una función distinta cuando se activen.

Nuestras interrupciones han de ser activas por flanco, ya que cuando se produce un cambio en los pulsadores correspondientes a los pines P2.10, P2.11 y P2.12. Cada interrupción pertenece a un pulsador de la placa ISP a la EINT2, KEY 1->EINT0, KEY2->EINT2. Otra utilidad que le hemos dado al pulsador IPS es aquella que cuando se encuentra en modo manual toma la medida mediante el sensor de ultrasonido.

```
void Config_IRQs(){

	LPC_PINCON->PINSEL4 |=(1<<(2*13))| (1<<(12*2))|(1<<(11*2)|(1<<(2*10)));
//Habiblitar pines para que funciones como EINT

	LPC_SC->EXTMODE |= (1<<3)|(1<<2)|(1<<1)|(1<<0); // Interupciones activas
por flanco.
	LPC_SC->EXTPOLAR |= 0;
	//Establecer la prioridad de las interupciones
	NVIC_SetPriority(EINT2_IRQn,1);
	NVIC_SetPriority(EINT1_IRQn,1);
	NVIC_SetPriority(EINT0_IRQn,1);
	NVIC_SetPriority(EINT3_IRQn,PRIORITY_EINT);
	//Habilitar cada una de las interrupciones
	NVIC_EnableIRQ(EINT3_IRQn);
	NVIC_EnableIRQ(EINT2_IRQn);
	NVIC_EnableIRQ(EINT1_IRQn);
	NVIC_EnableIRQ(EINT0_IRQn);
}
```

Dentro de cada Handler de la interrupción asignamos su funcionalidad, para la EINT0 correspondiente al pulsador ISP, este nos cambia el modo en el que nos encontremos en el momento de pulsar, otra funcionalidad es tomar las medidas si se encuentra en modo manual. Activando un flag que se activara al producirse la interrupción, en nuestro caso 'pulsaModo'.

```
void EINT0_IRQHandler(void){
	// Rutina de tratamiento de interrupción: EINT0
	//Borrar el flag de la EINT0, escribiendo un 1.
	LPC_SC->EXTINT |= (1<<0);
	LPC_GPIO1->FIOPIN |= 0x80000000;
	LPC_TIM0->TCR = 0x01;

	if(modo==1){
		pulsaModo=1;
	}
	else
		pulsaModo=0;
}
```

Eint1 tiene la funcionalidad de mientras este pulsado incrementar el ángulo del servo en 10º, dicha funcionalidad solamente estará activa si el dispositivo se encuentra en modo manual. Al activarse dicha interrupción cambiara el valor de una variable, que estará detectada en la función del servo y de allí incrementará el valor. La variable flag activada hasta ahora es 'pulsadorkey1=1' que entrará en el Handler del Timer2.

```
void EINT1_IRQHandler(void){
       // Rutina de tratamiento de interrupción: EINT1
       //Borrar el flag de la EINT1, escribiendo un 1.
       LPC_SC->EXTINT |= (1<<1);
       pulsadorkey1=1;
 }
```

Funcionalidad de EINT2 perteneciente al pulsador KEY 2 es el de decrementar el ángulo en el que se encuentra el servo en 10º. Al igual que la otra interrupción dicha funcionalidad estará activa si el dispositivo se encuentra en modo manual, activaremos una variable, 'pulsadorkey2' que será detectada en la función del servo.

```
  void EINT2_IRQHandler(void){
       // Rutina de tratamiento de interrupción: EINT2

       //Borrar el flag de la EINT2, escribiendo un 1.
       LPC_SC->EXTINT |= (1<<2);
       pulsadorkey2=1; //Cambiar de modo si está en automático pasa a manual y
viceversa.
 }
```

Finalmente la función Config_IRQs tiene que venir declarada en la función main de nuestro código, en caso contrario no podrá detectar la interrupción nunca.

5. Conclusiones: Código del programa.

Como conclusión podríamos decir que la realización del proyecto no ha ayudado a poner en práctica los conocimientos adquiridos durante las sesiones de teoría, saber como enfrentarnos a ciertos problemas prácticos, entender el funcionamiento de cada módulo que hemos añadido a nuestra placa, así como el microprocesador LPC1768.

Los errores como en todas la practicas han existido y no hemos encontrado con ciertos problemas o bien del código como de la placa. Haciendo una lista podemos citar algunos, el primero seria "calibrar" el servo atendiendo al código que teníamos, otro problema que podemos citar fue al juntar las distintas partes que funcionaban inicialmente por separado, dejaban de funcionar este problema se solvento con la variación de distintas prioridades.

Nuestros dispositivos vienen ya fabricados con pequeños errores, lo que implica que su comportamiento es no es al 100% el que nosotros deseamos, una ejemplo claro es el medidor de distancias o también llamado sensor ultrasonidos, hay ocasiones en el que la distancia no es medida con mucha precisión. Otros casos que podemos citar es el sensor de temperatura Lm35, sensor DS1621, etc.

```
//PROGRAMA PARA ULTRASONIDOS MIGUEL ÁNGEL DE LA INGLESIA SALIDO - VIOLETA
UNTARU

#include <lpc17xx.h>
#include "i2c_lpc17xx.h"
#include "lcddriver.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "uart.h"

 #undef  atoi


 /* Convert a string to an int.  */
 int
 atoi (const char *nptr)
 {
   return (int) strtol (nptr, (char **) NULL, 10);
 }


#define F_cpu 100e6 // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4 // Defecto despues del reset
#define Fpclk 25e6  // Fcpu/4 (defecto después del reset)
#define Tpwm 15e-3  // Perido de la señal PWM (15ms)

// PRIORIDADES DE LOS PERIFERICOS:
#define PRIORITY_SYSTICK 8
#define PRIORITY_EINT 4
#define PRIORITY_ISP 2
#define PRIORITY_TIM0 2

#define AUTOMATICO 1
#define MANUAL 0

#define INCREMENTO_SERVO 10 //incremento 10°
#define ANGULO_MAXIMO 180 //maximo 180°
#define ANGULO_MINIMO 0   //minimo 0°

#define ASCENDENTE 1      //-10
#define DESCENDENTE -1    //+10

#define N_muestras 32
#define pi 3.141516

// Variables UART
char buffer[200];         // Buffer de recepción de 200 caracteres
char *ptr_rx;               // Puntero de recepción
char rx_completa;         // Flag de recepción de cadena que se activa a "1"
al recibir la tecla return CR(ASCII=13)
char *ptr_tx;               // puntero de transmisión
char tx_completa;         // Flag de transmisión de cadena que se activa al
transmitir el caracter null (fin de cadena)
//Variables globales
int sentido=1;
int disparo=0;
int barrido=1;

int pulsadorkey1=0;             //pulsador activo
int pulsadorkey2=0;             //pulsador activo

int modo=MANUAL;          //inicialmente modo manual
int pulsaModo=0;
int angulo=90; //Mirando al frente
float temperatura;
float temperature;
```

```
int F_out=0;        //frecuencia de salida con el ultrasonido para DAC
float distancia;
int umbral=100;           //umbral que podemos elegir
uint32_t temp;
int contadorSysTick = 0;
char linea[100];

//Funcción que genera muestras en el DAC
int muestras[N_muestras];
int i;


void genera_muestras(uint8_t muestras_ciclo)
{
      uint8_t i;
      for(i=0;i<muestras_ciclo;i++)
            muestras[i]=1023*(0.5 + 0.5*sin(2*pi*i/muestras_ciclo)); //
Funcion que hace el seno,como son 10 bits de multiplica por 1023.
}


void init_DAC(void)
{
      LPC_PINCON->PINSEL1|= (2<<20);          // DAC output = P0.26 (AOUT)
      LPC_PINCON->PINMODE1|= (2<<20);  // Deshabilita pullup o pulldown
      LPC_SC->PCLKSEL0|= (0x00<<22);          // CCLK/4 (Fpclk después del
reset)
      LPC_DAC->DACCTRL=0;                                                //
Modo DMA deshabilitado
}


void printString(){


            fillScreen(BLACK); //Pinta en un primer momento la pantalla de
negro.

            //Mostrar distancia, esta línea mostrará por la pantalla la
distancia.
            if(distancia<=300){ //Si la distancia es inferior a 300cm, se
mostrará la distancia con su valor.
            sprintf(linea,"Distancia: %3.2f cm",distancia); //Escribe.
            drawString(5, 20, linea, YELLOW, BLACK, MEDIUM);     //Configura
posición del texto, color y tamaño.
            }
            else{ //En caso contrario, si la distancia supera los 300cm. se
imprimirá el mensaje fuera de rango.
            sprintf(linea,"Distancia fuera de rango"); //Escribe.
            drawString(5, 20, linea, YELLOW, BLACK, MEDIUM);     //Configura
posición del texto, color y tamaño.
            }

            //Mostrar angulo, esta línea mostrará por la pantalla el ángulo
actual.
            sprintf(linea,"Angulo servo: %d grados",angulo); //Escribe.
            drawString(5, 50, linea, YELLOW, BLACK, MEDIUM); //Configura
posición del texto, color y tamaño.

            //Mostrar temperatura I2C, esta línea mostrará por la pantalla la
temperatura medida por el LM35.
            sprintf(linea,"Temperatura %3.2f",temperatura);  //Escribe.
            drawString(5, 80, linea, YELLOW, BLACK, MEDIUM); //Configura
posición del texto, color y tamaño.

            //Mostrar temperatura I2C, esta línea mostrará por la pantalla la
temperatura medida por el DS1621.
            sprintf(linea,"Temp.I2C %3.2f",temperature); //Escribe.
```

```
            drawString(5, 110, linea, YELLOW, BLACK, MEDIUM); //Configura
posición del texto, color y tamaño.

}


void config_pwm1(void){
            LPC_PINCON->PINSEL3|=(1<<9); // P1.20 salida PWM (PWM1.2)
            LPC_SC->PCONP|=(1<<6);     //Alimentamos el PWM
            LPC_SC -> PCLKSEL0 |= 1<<3; //FCCLK/2
            LPC_PWM1 -> PR = 49; //1us
            LPC_PWM1 -> MR0 =20000;  //Ts = 20 ms
            //LPC_PWM1->MR0=Fpclk*Tpwm-1;
            LPC_PWM1->PCR|=(1<<10); //configurado el ENA2 (1.2)
            LPC_PWM1->MCR|=(1<<1); //reset en mro.
            LPC_PWM1->TCR|=(1<<0)|(1<<3);
      }


void configADC(){

LPC_SC->PCONP|=(1<<12); //  LO UTILIZAMOS PARA ACTIVAR EL ADC. POWER ON.
LPC_PINCON->PINSEL1|=(1<<14); // ENTRADA DEL ADC SERA P0.23 (ADO.0).
LPC_PINCON->PINMODE1|= (2<<14);  // DESHABILITA PULL DOWN.
LPC_ADC->ADCR|= (1<<0)|(1<<8)|(1<<21);//Canal 0, CLKDIV=1, PDN=1, EMPIEZA CON
EL MATCH 1 TIMER 2.
LPC_ADC->ADINTEN=1; // Habilitamos interrupción fin de conversión canal 0
NVIC_EnableIRQ(ADC_IRQn); //Habilitar interrupcion de ADC
NVIC_SetPriority(ADC_IRQn,2); // Establecer la prioridad
}


void set_servo(float grados){
            LPC_PWM1->MR2=((grados*50)/9)+300;
            LPC_PWM1->LER|=(1<<2)|(1<<0);
}


void DS1621_config(void)  //Configuración del sensor DS1621.
{
I2CSendAddr(0x48,0);             //Dirección establecida para el DS1621
(A2,A1,A0 a GND)
I2CSendByte(0xAC);               //Se accede al registro de configuración.
I2CSendByte(0x02);               //Modo de conversión continua(1SHOT=0)
I2CSendStop();
I2CSendAddr(0x48,0);             //Dirección establecida para el DS1621
(A2,A1,A0 a GND)
I2CSendByte(0xEE);               //Inicio de la conversión.
I2CSendStop();
}


void leerTemperatura(){
 char temp;                                  //Declaración de variable local.
I2CSendAddr(0x48,0);  //Escritura. (r/w=0).
I2CSendByte(0xAA);        //Comando de lectura de Tª.
I2CSendAddr(0x48,1);  //Re-Start para lectura(r/w=1).
temp=I2CGetByte(0);   //Lectura de byte MSB.

//Con las sentencias siguientes podremos establecer la precisión.
if(I2CGetByte(1)==0) temperatura=(float)temp; //Si el  bit7=0 del byte LSB la
variable permanece constante..
else temperature=(float)(temp)+0.5; // Si el  bit7=1 del byte LSB se añade a
la variable +0.5°.
I2CSendStop();

}
```

```
void Config_IRQs(){
	LPC_PINCON->PINSEL4 |=(1<<(2*13))| (1<<(12*2))|(1<<(11*2)|(1<<(2*10)));
//Habiblitar pines para que funciones como EINT

	LPC_SC->EXTMODE |= (1<<3)|(1<<2)|(1<<1)|(1<<0); // Interupciones activas
por flanco.
	LPC_SC->EXTPOLAR |= 0;
	//Establecer la prioridad de las interupciones
	NVIC_SetPriority(EINT2_IRQn,1);
	NVIC_SetPriority(EINT1_IRQn,1);
	NVIC_SetPriority(EINT0_IRQn,1);
	NVIC_SetPriority(EINT3_IRQn,PRIORITY_EINT);
	//Habilitar cada una de las interrupciones
	NVIC_EnableIRQ(EINT3_IRQn);
	NVIC_EnableIRQ(EINT2_IRQn);
	NVIC_EnableIRQ(EINT1_IRQn);
	NVIC_EnableIRQ(EINT0_IRQn);
}


void EINT0_IRQHandler(void){
	// Rutina de tratamiento de interrupción: EINT0
	//Borrar el flag de la EINT0, escribiendo un 1.
	LPC_SC->EXTINT |= (1<<0);
	LPC_GPIO1->FIOPIN |= 0x80000000;
	LPC_TIM0->TCR = 0x01;
	//Función para cambiar de modo
	if(modo==1){
		pulsaModo=1;
	}
	else
		pulsaModo=0;
}


void ADC_IRQHandler(void)
{
	float voltios; //Variable local
	voltios= ((LPC_ADC->ADDR0>>4)&0xFFF)*3.3/4095;		//se borra
automat. el flag DONE al leer ADCGDR.
	temperatura=voltios*100; //Valor de la temperatura
}


void init_TIMER2(void){
	LPC_SC->PCONP|=1<<22; //Power on, para alimentar
	LPC_SC->PCLKSEL1|=1<<12; //Lo que configuramos aqui es el divisor de la
señal. Pclk/2
	LPC_TIM2->PR=49; //Configuramos el preescaler 49+1=50
	LPC_TIM2->MR1=500000; //1pulso-->0.000001, para obtener 0.5 segundos--
>500000pulsos.
	LPC_TIM2->MCR|=(1<<3)|(1<<4); //Interrupción en MR1 y Reset en MR1.
	NVIC_SetPriority(TIMER2_IRQn,1); //Configuramos la prioridad del timer.
	NVIC_EnableIRQ(TIMER2_IRQn); //Habilitamos la prioridad.
	LPC_TIM2->TCR|=(1<<0); //Habilitamos el timer.
}


void TIMER2_IRQHandler(void){

	if((LPC_TIM2->IR & (1<<1))==(1<<1))	//MR1
	{
		LPC_TIM2->IR|=(1<<1);		//Flanco de interrupción para MR1.
		LPC_ADC->ADCR|=(1<<24);		//Se inicia la conversión del ADC.
		leerTemperatura();		//Se lee la temperatura del DS1621.
		printString();			//Se imprimen los valores por el LCD.
```

```
            if(modo==AUTOMATICO && pulsaModo==1){  //Si el sistema está en
modo barrido y se pulsa la Eint0:
            LPC_GPIO1->FIOPIN |= 0x80000000; //Escribimos un 1 en el pin 1.31
            LPC_TIM0->TCR = 0x01;  //Activamos el timer 0.
//Esto permitirá realizar medidas constantes.

            angulo+=(INCREMENTO_SERVO)*sentido; //Incrementamos o
decrementamos 10° en función del giro a izquierdas o
derechas  del servo.
            set_servo(angulo); //Mandamos esa posición al servo.

                  if(angulo==ANGULO_MAXIMO)//Si el servo se encuentra en
180°
                        sentido=DESCENDENTE; //Decrementará su valor.
                  else if(angulo==ANGULO_MINIMO)//Si el servo se encuentra
en 0°
                        sentido=ASCENDENTE; //Aumentará su valor.
            }

            else{  //Si el sistema está en modo manual y se pulsa Eint0:

                  if(pulsadorkey2) //Si se pulsa el pulsador controlado por
la Eint2
                        {
                  if (angulo>ANGULO_MINIMO){//Mientras el angulo sea mayor
al mínimo
                  angulo-=INCREMENTO_SERVO;  //Decrementa
                  set_servo(angulo);}        //Envía posición al servo.

                  pulsadorkey2=0;  //Inicializamos a cero por si se vuelve a
pulsar la EINT2.

                        }
            else if(pulsadorkey1){     //Si se pulsa el pulsador controlado
por la Eint1.
                  if (angulo<ANGULO_MAXIMO){   //Mientras el angulo sea
inferior al                                          máximo
                  angulo+=INCREMENTO_SERVO;  //Aumenta
                  set_servo(angulo);         //Envía posición al servo.
                  }
                  pulsadorkey1=0;//Inicializamos a cero por si se vuelve a
pulsar la EINT1.
                                    }
                  }
      }

}

void init_TIMER0(void){
      // Tiempo de interrupción en SG * 25e6 - 1
      LPC_SC->PCONP|=(1<<1); // Alimenta TIMER0
      LPC_TIM0->MCR = 0x07; // con 7 activo los 3 bits: STOP, Interrupcion,
Reset
      LPC_TIM0->MR0 = (Fpclk * 1e-5)-1; // 10micro segundos
      LPC_TIM0->TCR = 0; // Apagar TIMER
      NVIC_SetPriority(TIMER0_IRQn,2);  //Establecer prioridades
      NVIC_EnableIRQ(TIMER0_IRQn);     //Habilitar interrupcion
}


void TIMER0_IRQHandler(void) {
      LPC_TIM0->IR|= (1<<1);     //Borrar el flag de interrupción del TIMER0
      LPC_GPIO1->FIOPIN &= ~0x80000000;      //Limpiar el pin
      LPC_TIM0->TCR = 0; // Apagar timer0
}
```

```
void configurarPines(){
	// Declarar como salida el pin del pulso
	LPC_GPIO1->FIODIR |= 0x80000000; //pin 1.31
	LPC_GPIO1->FIOPIN &= ~0x80000000;
}


void init_Timer1(){
	//Programar Capture
	LPC_SC->PCONP|=(1<<2); //Alimentamos el TIMER1
	LPC_PINCON->PINSEL3|=(3<<4)|(3<<6); //Establecemos la función de dicho
pin
	LPC_TIM1->PR=24;
	LPC_TIM1->CCR = (1<<0)|(1<<2)|(1<<4)|(1<<5);  //CAP1.0 flanco de subida|
interrupción| CAP1.1 flanco bajada| interrupción
	LPC_TIM1->TCR =(1<<0);    //Contador RESET
	NVIC_SetPriority(TIMER1_IRQn,2); //Establecer prioridad
	NVIC_EnableIRQ(TIMER1_IRQn);     //Habilitar interrupción
}


void TIMER1_IRQHandler(){

	float dif_pulsos; //variable local

		LPC_TIM1->IR|=(1<<4); //Reset captura de CR0
		LPC_TIM1->IR|=(1<<5);     //Reset captura de CR1
			dif_pulsos=((LPC_TIM1->CR1)-(LPC_TIM1->CR0)); //Calcular
diferencia
			distancia=dif_pulsos*0.017;            // Distancia final

 }


 void EINT1_IRQHandler(void){
	// Rutina de tratamiento de interrupción: EINT1
	//Borrar el flag de la EINT1, escribiendo un 1.
	LPC_SC->EXTINT |= (1<<1);
	pulsadorkey1=1;

 }

 void EINT2_IRQHandler(void){
	// Rutina de tratamiento de interrupción: EINT2
	//Borrar el flag de la EINT2, escribiendo un 1.
	LPC_SC->EXTINT |= (1<<2);
	pulsadorkey2=1; //Cambiar de modo si está en automático pasa a manual y
viceversa.
 }


void init_TIMER3(void)
{
	  LPC_SC->PCONP|=(1<<23); // Alimentamos TIMER3
    LPC_TIM3->PR = 0x00;
    LPC_TIM3->MCR = 0x03; // Reset TC on Match, and Interrupt MR0
    LPC_TIM3->MR0 =F_out; //Frecuencia dependiente del Umbral
    LPC_TIM3->EMR = (1<<1); // Nivel alto ELBIRDE salida
    LPC_TIM3->TCR = 0x02;
	LPC_TIM3->TCR = 0x01;      // Habilitar TIMER
	NVIC_SetPriority(TIMER3_IRQn,1);  // Habilitar prioridad
	NVIC_EnableIRQ(TIMER3_IRQn);     // Habilitar interupcion

}

void TIMER3_IRQHandler(void)
{
	static uint8_t indice_muestra;
```

```
	LPC_TIM3->IR|= (1<<0); // Limpiar interrupcion de TIMER 3
	LPC_DAC->DACR= muestras[indice_muestra++] << 6;// El DAC se encuentra
entre el bit 6 y el bit 15
	indice_muestra&= 0x1F;// Limpiamos para futuros recibos de informacion
}

int main(void)
{
//Configuración de la Uart.
int opcion; //Se crea una variable que nos permitirá guardar los valores
introducidos por la UART.
ptr_rx=buffer;//Inicializa el puntero de recepción al comienzo del buffer.
uart0_init(9600); //Se configura la velocidad de la UART a 9600 baudios.


	//Funciones a utilizar
	lcdInitDisplay();
	configurarPines();
	init_TIMER0();
	init_Timer1();
	DS1621_config();
	leerTemperatura();
	init_TIMER2();
	config_pwm1();
	Config_IRQs();
	genera_muestras(N_muestras);
	configADC();
	init_DAC();
	init_TIMER3();


	//Se transmite dentro de la cadena de caracteres.
	tx_cadena_UART0("Introdude:\n 1.Suma\n2.Resta\n3.Modo\n4.Umbral\n\r");
	//Se transmite la cadena de caracteres.

	while(1){

		/*
		//Estas lineas de codigo permitiran conectarse a MATLAB
		sprintf(buffer,"%d %3.2f\n", angulo, distancia);     //Estas
líneas de código permitirán conectarse a MATLAB.
		tx_cadena_UART0(buffer);
		while(tx_completa==0);
		tx_completa=0;
		*/

		if(rx_completa){
			//Si se ha finalizado la transmisión.
		rx_completa=0;
			//Se inicializa rx=0, para nuevas transmisiones.
		opcion= atoi(buffer);
			//Se guarda en el buffer la opcion señalada.

			switch(opcion){
				case 1:
				//Se incrementa en 10° el angulo del servo.
					if (angulo < ANGULO_MAXIMO ){
						angulo += INCREMENTO_SERVO;
						set_servo(angulo);
					}
					break;

				case 2:
				//Se decrementa en 10° el angulo del servo.
					if (angulo > ANGULO_MINIMO ){
						angulo -= INCREMENTO_SERVO;
						set_servo(angulo);
					}
```

```
                        break;

                    case 3:
                    //Cambio de modo manual automático.
                        modo = !modo;
                        break;

                    default :
                    //Se establece el nuevo umbral.
                        if(opcion>=30 && opcion<=300){
      //Si el umbral no se haya entre esos valores no se actulizará.
                        umbral=opcion;
                        }
                    break;

                }


                if(distancia<=300){ //En el caso de que la distancia sea
menor o igual a 300:
                sprintf(buffer,"Grados: %d\nDistancia: %f\nUmbral: %d\n\r",
angulo, distancia, umbral);   //Se mostrarán los valores pedidos.
                }
                else{
            //Si no
                sprintf(buffer,"Grados: %d\nDistancia: Fuera de
rango\nUmbral: %d\n\r", angulo, umbral); //Se mostrará esta cadena.
                }

                tx_cadena_UART0(buffer);while(tx_completa==0);
                tx_completa=0;    //Se transmite la cadena anterior.
                tx_cadena_UART0("Introdude:\n
                1.Suma\n2.Resta\n3.Modo\n4.Umbral\n\r"); //Se vuelve a
                transmitir la cadena inicial.

}

      if(distancia<=umbral){
      F_out= 5000 - umbral*10;
      LPC_TIM3->MR0 =(F_pclk/F_out/N_muestras)-1;
      LPC_TIM3->TCR = 0x01;
      }
      else
            LPC_TIM3->TCR = 0x02;


      }
}
```