

 <b>UNIVERSIDAD DE ALCALÁ</b> <b>ESCUELA POLITÉCNICA SUPERIOR</b> <b>DEPARTAMENTO DE ELECTRÓNICA</b>		 <b>GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES</b>	
<b>ASIGNATURA</b>	<b>SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS</b>	<b>FECHA</b>	<b>MAYO 2019</b>
<b>APELLIDOS, NOMBRE</b>	<b>SOLUCIÓN</b>		<b>GRUPO</b>

### PRUEBA DE EVALUACIÓN FINAL

#### CUESTIÓN 1.

Se propone diseñar un SE basado en microcontrolador LPC1768 (Cortex-M3) para implementar un medidor de temperatura y humedad con dos grandes marcadores de representación analógicos, construidos mediante dos servomotores. Mediante un PC y con ayuda de un programa terminal de comunicaciones se podrán configurar determinados parámetros del sistema. El sistema permitirá generar varios tipos de señales acústicas de alarma seleccionables desde un menú. Un potenciómetro permite ajustar el volumen del audio. Un display LCD gráfico mostrará los valores de temperatura y humedad cada minuto.

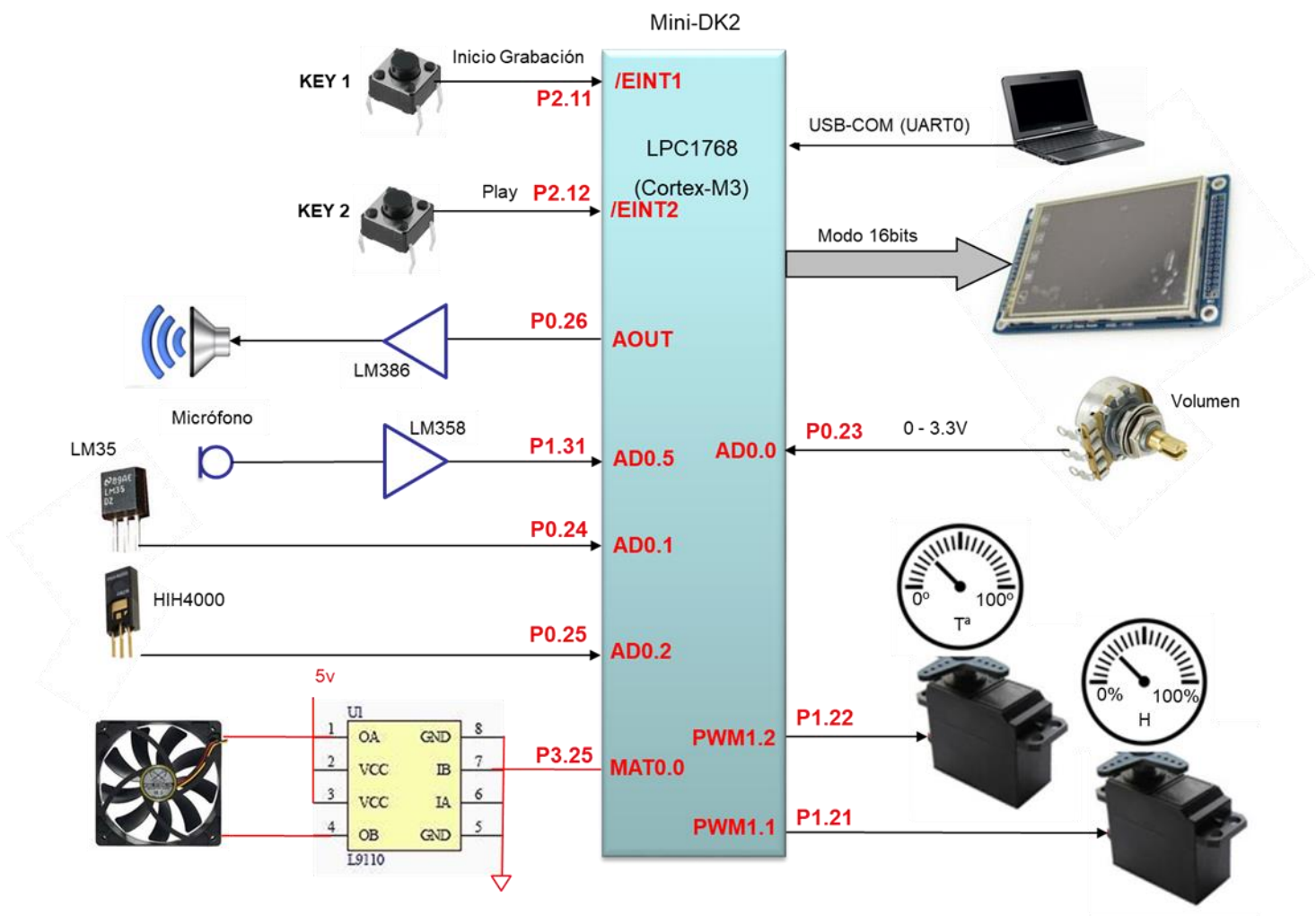


Figura 1. Diagrama de bloques sistema

#### Descripción del funcionamiento del sistema:

El movimiento de cada servo ( $0^{\circ}$ - $180^{\circ}$ ) está asociado a los valores de la temperatura y humedad y provocará un desplazamiento angular de la aguja de cada marcador, en un rango de 0 a 100 de cada magnitud a medir.

Un ventilador permite extraer el calor interior en caso de que esté expuesto al sol, si se supera determinado umbral de temperatura.

Considere que existe la variable global *tipo\_alarma* que permite seleccionar si la alarma es un tono o una señal de audio. En caso de que esté seleccionado el tono, la frecuencia de reproducción será de 1Khz para la temperatura y 2Khz para la humedad.

Considere el SysTick habilitado y que interrumpa periódicamente cada 50ms. Utilice esta función de interrupción para hacer medidas de tiempos grandes.

- a) Complete el interconexión sobre el diagrama de la figura 1, el pin **Pn.x** (sobre la línea de conexión) y el nombre del recurso utilizado dentro del bloque que representa al LPC1768 (ej. **MAT1.0**).
- b) Complete la función de configuración del ADC (incluyendo los comentarios) para las entradas del sensor de **temperatura** (LM35), **humedad** (HIH4000) y del **potenciómetro** de volumen. **Calcule la frecuencia de muestreo de cada canal y el tiempo de conversión.**

```
void init_ADC_sensores(void)
{
LPC_ADC->ADCR= ( 7 <<0) |           // Canal 0,1 y 2
               ( 24<<8) |         // CLKDIV=24   FclkADC= 25MHz/25= 1 MHz
               (1<<21) |         // PDN=1
               (1<<16);         // Modo Burst
NVIC_DisableIRQ(ADC_IRQn);      // ADC no interrumpe
}
```

$$t_{conv} = 65 * 1\mu s = 65 \mu s$$

$$F_{muestreo} = \frac{1 \text{ MHz}}{65 * 3} = 5,1 \text{ kHz}$$

NOTA: Considere ya escrita la siguiente función y **añada sólo los comentarios:**

```
void init_ADC_pines(void)
{
LPC_SC->PCONP|= (1<<12);           // Power ON
LPC_PINCON->PINSEL |= (           ) ; // P0[23...25] es AD0[0...2]
LPC_PINCON->PINMODE |= (         ) ; // Deshab. pullup/pulldown P0[23...25]
}
```

- c) Complete la función de configuración de las señales PWMs de los servos, y de actualización de su posición en función de los argumentos de entrada. Añada los comentarios. Considere que el periodo sea de **15ms**, y el tiempo a nivel alto varíe entre **0.4-2.6ms**, para un movimiento de su posición entre 0° y 180°.

```
void config_pwm_servos(void)
{
//pseudocódigo
Configurar P1.21 y P1.22 como PWM1.1 y PWM1.2
Power módulo PWM
MR0=F_pclk*15e-3 - 1; // Periodo PWM
Reset on Match 0, NO interrumpe
ENA2=1, ENA1=1      // Hab. PWM1.1 y PWM1.2 (modo single edge)
Start Timer
}
```

```
void set_servo(uint8_t temperatura, uint8_t humedad)
{
LPC_PWM1->MR1 = F_pclk*2.6e-3 - F_pclk*2.2e-3*temperatura/100;
LPC_PWM1->MR2 = F_pclk*2.6e-3 - F_pclk*2.2e-3*humedad/100;
LPC_PWM1->LER|= (1<<2) | (1<<1) | (1<<0);
}
```

- d) Explique el recurso que utilizaría y como lo configuraría, para generar el **señal PWM** que actuaría sobre el **ventilador**. Considere una frecuencia de la señal PWM de 1 KHz y que el ciclo de trabajo varía entre el 10% y 90%, en función de una variable global **ciclo\_PWM** ya declarada que toma valores entre 0 y 1. Complete la conexión del L9110 con el ventilador y el LPC1768, según **Anexo III**.

### Cómo la frecuencia de la PWM es distinta, empleamos un Timer en modo MATCH (Timer0)

#### P3.25 salida MAT0.0

$MR0 = F_{clk}/1e3 - 1$  para fijar el periodo.

Reset on Match Timer 0 y modo **SET on MATCH 0** (P3.25 a nivel alto en cada Match de MR0) **sin interrupción**.

Hab. Interrupt on Match (MR1) para fijar el ciclo de trabajo, y en la interrupción del Timer ponemos manualmente a nivel bajo MAT0.0 (ojo en reg. EMR0)

**MR1 = MR0 \* ciclo\_PWM;**

- e) Complete la función de **configuración del ADC** (incluyendo los comentarios) para la entrada a la que se conecta el micrófono e indique el **tiempo de conversión** y la configuración del Timer (**sin escribir el código**) que proporciona el muestreo ( $F_s = 8\text{kHz}$  a **12bits**). Considere que funciona por DMA para grabar **1** segundo de audio para cada alarma (temperatura y humedad). **Justifique los recursos de memoria necesarios**.

```
void init_ADC_microfono(void)
{
LPC_ADC->ADCR= (    1<<5 |    // AD0.5 -> canal 5 (P1.31)
                (    1<<8 |    // Fclk= 25e6/2= 12,5 MHz
                (1<<21 |    // PDN=1
                (6<<24);    // MAT1.0 -> frecuencia de muestreo
LPC_ADC->ADINTEN=(1<<5);    // Hab. Int. canal 5 para DMA
NVIC_DisableIRQ(ADC_IRQn); // Ojo! ADC no interrumpe!!
}
```

**$T_{conv} = 65 * 1 / 12,5\text{MHz} = 5,2 \mu\text{s}$**

**$T_s = 1 / F_s = 125 \mu\text{s}$**

#### Configuración TIMER 1:

Power ON Timer1

Reset on Match 0

**MR0 = 25e6 / 8000 / 2 - 1;** // Cada 2 Match inicio de conversión ( $F_s = 8\text{kHz}$ )

Sin interrupción

Stop Timer

#### Memoria RAM:

$F_s = 8000 \text{ KHz}$  (8bits) Duración = 1 seg. por mensaje

Tamaño =  $8000 \text{ Ks/s} * 1\text{s} = 8000 \text{ muestras} * 2 \text{ bytes/muestra} = 16 \text{ KB por mensaje}$

**int16\_t buffer\_alarm\_temp[8000]; int16\_t buffer\_alarm\_humedad[8000];**

- f) Escriba la función de interrupción del Timer que saca las muestras hacia el DAC para generar la señal de alarma (tono o voz), y calcule el periodo **mínimo** de interrupción que soportaría en el peor de los casos.

**NOTA1:** Considere ya inicializado el array con los valores discretos de un ciclo para la señal de alarma basada en un tono de frecuencia F.

**NOTA2:** Tenga en cuenta que existe la variable global *tipo\_alarma* y los arrays *buffer\_alarm\_temp[]* y *buffer\_alarm\_hum[]* con las muestras de los respectivos audios ya grabadas. **No considere la posibilidad que se den las dos alarmas de forma simultánea.**

```
void genera_muestras()
{
    uint16_t i;
    for(i=0;i<muestras_ciclo;i++) muestras[i]=(int) (511 + 511*sin(2*pi*i/32))<<6;
}

void TIMER2_IRQHandler(void)
{
    LPC_TIM2->IR|= (1<<0); // borrar flag
    volumen=(float) ((LPC_ADC->ADDR0>>8) &0xFF) /255; // Basta resolución de 8 bits
    if (tipo_alarma==tono) { //muestras ya desplazadas
        muestra_volumen= (uint32_t)((muestras[indice++]>>6)*volumen);
        LPC_DAC->DACR= muestra_volumen << 6;
        indice&=31; // 32 muestras por ciclo
    }
    else {
        if (alarma_temperatura) {
            muestra_volumen= (uint32_t)(buffer_alarm_temp[indice++]*volumen);
            LPC_DAC->DACR= muestra_volumen << 4; // Alinear con bit de mayor peso
        }
        else {
            muestra_volumen= (uint32_t)(buffer_alarm_hum[indice++]*volumen);
            LPC_DAC->DACR= muestra_volumen << 4; // Alinear con bit de mayor peso
        }
        if (indice==8000){
            indice=0;
            LPC_TIM2->TC=0x02; // Stop Timer 2
        }
    }
}
```

**NOTA:** Para activar la alarma será necesario arrancar el Timer e inicializar T2MR0 en función del tipo de alarma seleccionado.

En caso de ser un tono, la frecuencia más alta ( $F = 2\text{kHz}$ ) de este es la que hace que el periodo de interrupción sea mínimo, pues la frecuencia de interrupción del timer es proporcional a la frecuencia de salida del tono y al  $N^\circ$  de muestras por ciclo, esto es:  $F_{out} * 32 \text{ KHz}$ .

**$T2MR0 = F_{pclk} / 8000 - 1$ ;  $T_{int} = 1 / 8\text{kHz} = 125 \text{ us}$  (audio)**

**$T2MR0 = F_{pclk} / (F_{out} * 32) - 1$ ;  $T_{int} = 1 / (F_{out} * 32) \text{ Hz}$  (tono)**

**$T_{int}$  mínimo será para  $F_{out} = 2 \text{ KHz}$**

g) Explique qué recurso utilizaría y como lo configuraría para reducir la carga de CPU durante la **reproducción de la señal de alarma**.

Evitaríamos la ejecución de la interrupción del Timer3 empleando el DMA con 2 canales para cada tipo de alarma. Tono → Modo *Linked* con el propio canal (ej. Canal 0), para que al finalizar la transferencia se inicie de nuevo sin que sea necesaria la interrupción del DMA.

- `LPC_GPDMA0->DMACCSrcAddr = (uint32_t) &muestras[0]; // Fuente Memoria`
- `LPC_GPDMA0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR); //Destino Perif. (ojo 16 bits)`
- `Transfer Size=32; // Tamaño del bloque`
- Ancho transferencia= **16 bits**; las muestras del array ya están desplazadas al bit 6.
- Incrementa Fuente/No incremento Destino
- Sin interrupción (ojo, modo Linked)

**NOTA:** Para modificar la Fout en función de la alarma (ej. en la rutina del SysTick):

`LPC_DAC->DACCNTVAL = (F_pclk/32/Fout) -1; // Frecuencia transf. DMA`

Audio → **2 transferencias de 4000 muestras** (8000 muestras) con interrupción (ej. Canal 1) iniciando el DMA con distinta dirección origen en función de la alarma o mensaje.

- `LPC_GPDMA1->DMACCSrcAddr = (uint32_t) & buffer_alarm_temp[0]; // Si alarma por tempera`
- `LPC_GPDMA1->DMACCSrcAddr = (uint32_t) & buffer_alarm_hum[0]; // Si alarma por Humedad`
- `LPC_GPDMA1->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR); //Destino Perif.`
- `Transfer Size=4000; // Tamaño del bloque`
- Ancho transferencia= **16 bits**; Ojo ADC → 12 bits
- Incrementa Fuente/No incremento Destino
- Habilita interrupción por transferencia completa.

**NOTA:** `LPC_DAC->DACCNTVAL = (F_pclk/8000) -1; // Frecuencia transf. DMA`

h) Se desea controlar el sistema mediante una conexión WIFI, para lo cual se conecta con la **UART3** el módulo ESP8266-01 de la figura.

- Complete el diagrama de conexión del LPC1768 con el módulo WIFI.
- Complete la función de configuración del puerto serie y **realice los cálculos** para configurar la velocidad a 115200 baudios, 8 bits por dato y **paridad par**.
- Calcule la velocidad real obtenida y el tiempo que tarda en transmitirse el comando enviado al ejecutarse la sentencia, `tx_cadena_UART3("Ejecutando...\n\r")`. Ver **Anexo IV**.
- Qué parámetro cambiaría (que no sea DivAddval ni Mulval) para mejorar la exactitud de la velocidad real.

```
void uart3_init(void)
```

```
{
    LPC_PINCON->PINSEL9|=3<<26; // RXD3 (P4.29);
    LPC_PINCON->PINSEL9|=3<<24; // TXD3 (P4.28);
    LPC_UART3->LCR=0x93;
    LPC_UART3->DLL= 14;
    LPC_UART3->DLM= 0;
    LPC_UART3->LCR= 0x13;
}
```

Consideramos FR=1.

$DL_{16} = 25e6 / (16 * 115200) * FR = 13,56 \rightarrow 14$

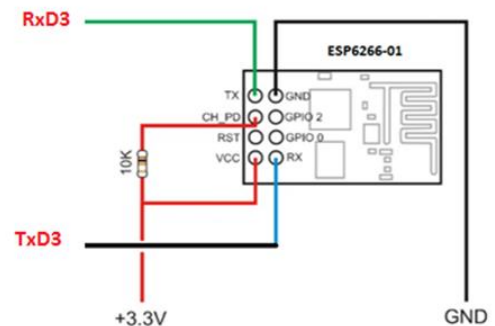
$Vt = 25e6 / (16 * 14) = 111.607$  baudios

$T_{mensaje} = (1/Vt) * N^{\circ} \text{ caracteres} * 11 \text{ bits/carácter} = 8.96e-6 * 15 * 10 = 1,47ms$

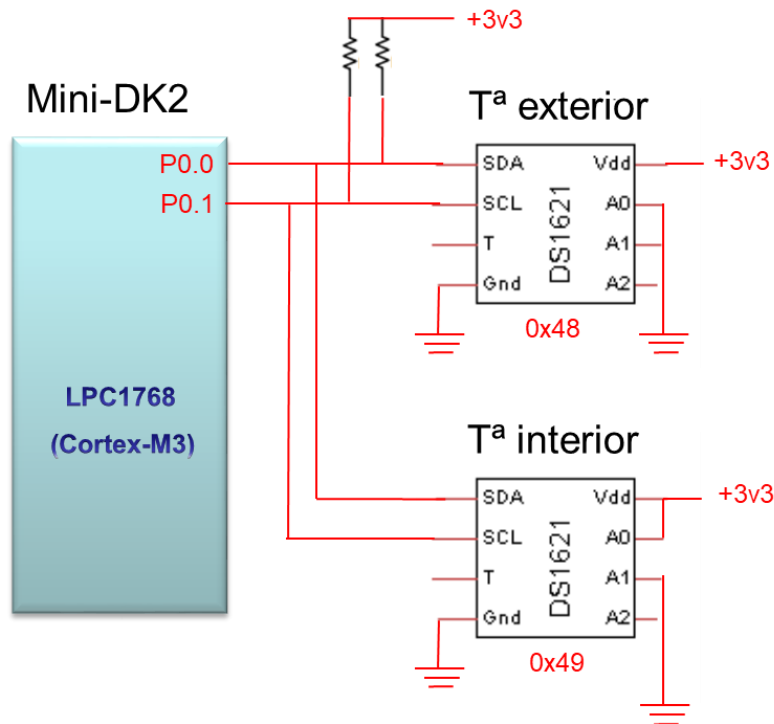
Modificando PCLK de la UART a 50 Mhz (configurar el prescaler ÷2) resulta:

$DL_{16} = 50e6 / (16 * 115200) * FR = 27,12 \rightarrow 27$

$Vt = 50e6 / (16 * 27) = 115.740,7$  baudios



- i) Se desea conectar al sistema dos sensores digitales de temperatura con bus I2C (DS1621). Complete sobre la figura el diagrama de conexión, y escriba la función de configuración de la **temperatura máxima y mínima** del termostato interior y exterior. Considere estos valores como enteros (**T<sup>a</sup> con precisión de 1 grado**).  
**NOTA:** Ver Anexos V y VI.



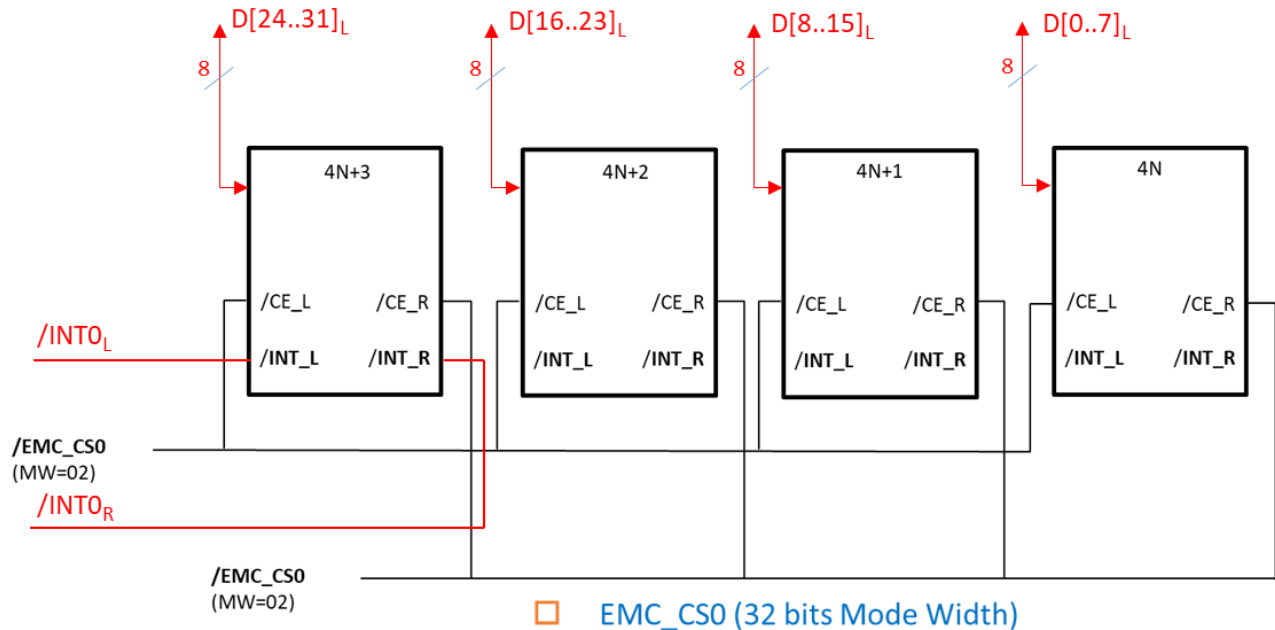
```
void DS1621_set_termostato(uint8_t termostato, uint8_t TH, uint8_t TL)
{
    if (termostato==exterior) {
        I2CSendAddr(0x48,0); // Dir. DS1621 ( Tª exterior)
        I2CSendByte(0xA1); // Acceso a TH
        I2CSendByte(TH); // Sólo se escribe 1 byte (resolución 1 grado)
        I2CSendStop();
        I2CSendAddr(0x48,0); // Dir DS1621 ( Tª exterior)
        I2CSendByte(0xA2); // Acceso a TL
        I2CSendByte(TL);
        I2CSendStop();
    }
    else {
        I2CSendAddr(0x49,0); // Dir. DS1621 ( Tª interior)
        I2CSendByte(0xA1);
        I2CSendByte(TH);
        I2CSendStop();
        I2CSendAddr(0x49,0);
        I2CSendByte(0xA2);
        I2CSendByte(TL);
        I2CSendStop();
    }
    I2Cdelay_50ms(); // tiempo escritura EEPROM
}
```

**CUESTIÓN 2.**

Complete las conexiones de los buses de datos (CPU izq. y de las líneas de interrupción (CPU izq. y der.) considerando que se utiliza el método de arbitración por interrupción empleando las **dos últimas direcciones más altas del mapa**.

Justifique el funcionamiento del método de arbitración a partir de mapa resultante.

**NOTA:** Considere que el bloque de memoria está mapeado en la dirección **0x8000.0000** y que cada chip tiene una capacidad de **64Kbytes**.



El espacio de direccionamiento del bloque de memoria de la Dual-Port de la figura, para la CPU de la izquierda y derecha es de **256 kBytes** (**0x8000.0000 – 0x8003.FFFF**).

Para que la arbitración por interrupción del bloque de memoria de la DUAL-PORT emplee **las dos últimas posiciones de memoria del mapa** es necesario utilizar las últimas posiciones del chip **4N+3** con sus respectivas líneas de interrupción para cada lado. Esto hace que se utilicen las posiciones **0x8003.FFFF** para la **CPU<sub>L</sub>** y la **0x8003.FFFB** para la **CPU<sub>R</sub>**.

La **CPU<sub>L</sub>** escribe en **0x8003.FFFF** e interrumpe a la CPU<sub>R</sub> activando /INT<sub>R</sub>. La CPU<sub>R</sub> lee dicha posición y desactiva o deja en reposo (nivel alto) la señal de interrupción (/INT<sub>R</sub>).

La **CPU<sub>R</sub>** escribe en **0x8003.FFFB** e interrumpe a la CPU<sub>L</sub> activando /INT<sub>L</sub>. La CPU<sub>L</sub> lee dicha posición y desactiva o deja en reposo (nivel alto) la señal de interrupción (/INT<sub>L</sub>).

**CUESTIÓN 3.**

Se desea realizar el State Chart del funcionamiento de un medidor láser de distancia<sup>1</sup>. El medidor tiene 8 botones pero sólo es necesario modelar la funcionalidad de 4 de ellos (CLR, MEAS, UNIT y MODE):



- Botón **CLR**
  - Pulsación corta – borra los números de la pantalla y reinicia la medida que está haciendo. Si no está en el modo distancia simple se pone en él.
  - Pulsación larga – apaga el dispositivo entrando en modo de bajo consumo.
- Botón **MEAS** (Measure)
  - Pulsación larga - Con el dispositivo apagado (en modo de bajo consumo) lo enciende.
  - Pulsación corta – Estando el sistema activo enciende el láser en la primera pulsación y en la segunda pulsación toma medida de distancia, genera 3 parpadeos en el láser de 100ms, y visualiza la medida en el display.
- Botón **UNIT**
  - Pulsación larga – Cambia la visualización de metros a pies y viceversa.
- Botón **MODE** - Cambia el modo de medida y la visualización.
  - Por defecto está en “Modo Distancia Simple” y con una pulsación pasa al “Modo Área”.
  - “Modo Área” – En este modo al pulsar MEAS toma una medida y la visualiza. En una segunda pulsación a MEAS visualiza también la medida y calcula el área. Una pulsación MODO y pasa a “Modo Volumen”.
  - “Modo Volumen” – De similar forma al “Modo Área” en este modo se toman tres medidas y calcula el volumen del espacio. Una nueva pulsación a MODO y pasa a “Modo Área”.
  - Para pasar el “Modo de Distancia Simple” es necesario una pulsación de CRL.
- Si el dispositivo está encendido y no se toca ningún botón:
  - en 10 seg el backlight de la pantalla se apaga.
  - en 30 seg se apaga el láser en caso de estar encendido.
  - en 3 minutos se apaga dispositivo yendo a bajo consumo.

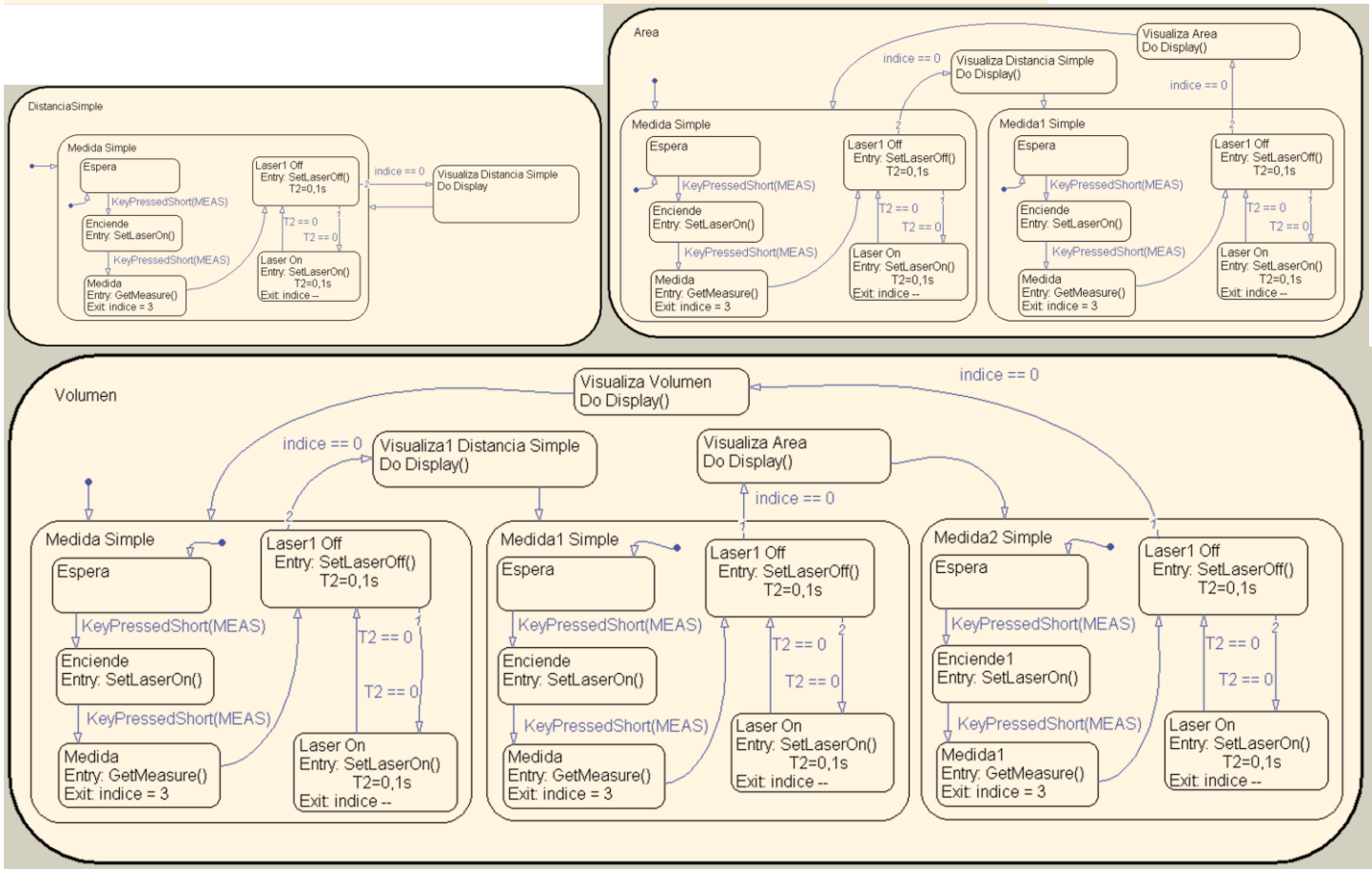
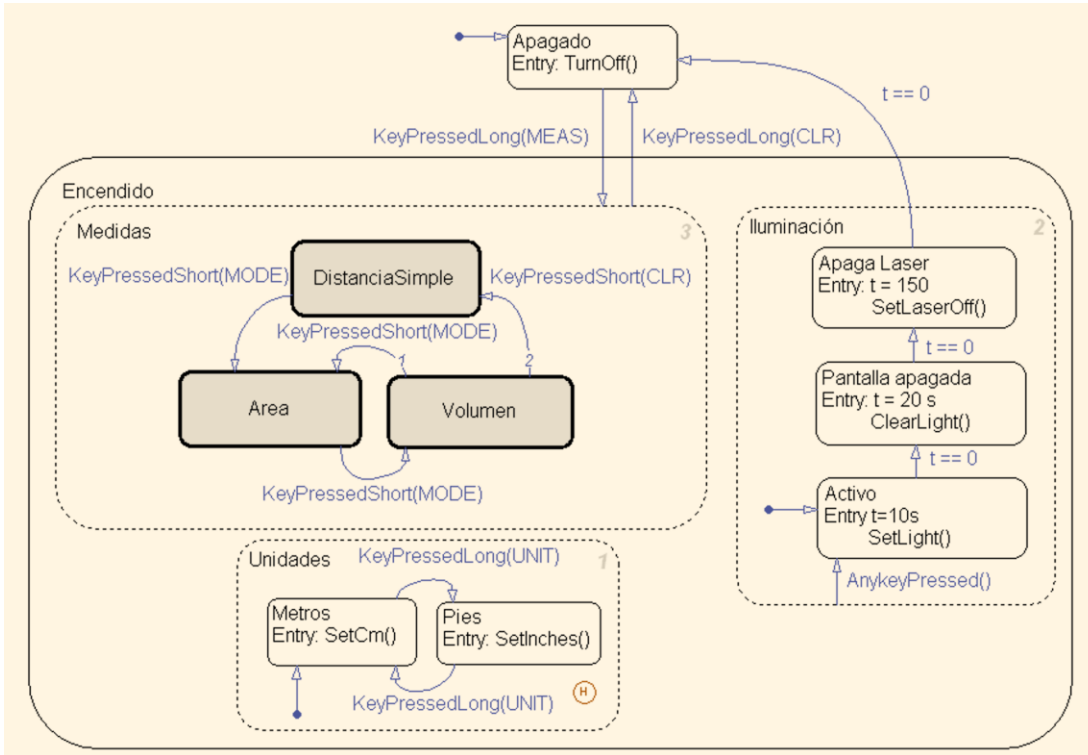
Realice el StateChart que modela el funcionamiento del sistema suponiendo que están implementadas las siguientes funciones:

- void SetLaserOn(void) – Enciende el láser
- void SetLaserOff(void) – Apaga el láser.
- float GetMeasure(void) – Toma una medida de distancia. Si negativo la medida es inválida.
- void SetCm(void) – Selecciona medida en centímetros
- void SetInches(void) – Selecciona medida en pulgadas
- void Display(void) – Refresca la información de la pantalla en función del estado correspondiente.
- void TurnOff(void) – Envía el sistema a modo de bajo consumo
- void SetLight(void) – Activa la iluminación del display.
- void ClearLight(void) – Desactiva la iluminación del display.
- bool AnyKeyPressed(void) – Devuelve un nivel alto si se detecta cualquier pulsación.
- bool KeyPressedShort(TKey key) – Devuelve un nivel alto cuando se detecta una pulsación corta en el botón indicado definido por los nombres: CLR, MEAS, UNIT y MODE ( ejemplo de uso: if (KeyPressedShort(MEAS)) ...)
- bool KeyPressedLong(TKey key) – Devuelve un nivel alto cuando se detecta una pulsación larga en el botón indicado

<sup>1</sup> Imagen obtenida de: <https://www.amazon.es/Suaoki-Tel metro-individual-triangular-sustracci n/dp/B019MPVTDK/>



Se supone que las temporizaciones se realizan con temporizadores software implementados en una interrupción periódica donde se van decrementando hasta llegar a cero.



**CUESTIÓN 4**

Explique las ventajas e inconvenientes de utilizar un Sistema Operativo de Tiempo Real en el proyecto de la asignatura (práctica de laboratorio).

**CUESTIÓN 5.**

Suponga que el sistema de control explicado en la cuestión 3 se implementa con un LPC1768 y proporciona la funcionalidad de poderse monitorizar y controlar mediante un servidor HTTP empotrado.

La información monitorizada corresponde a la de la imagen que se muestra a continuación y contiene los siguientes elementos:

- Información del modo activo
- Selección de las unidades de medida de distancia
- Orden de inicio de medida
- Visualización de la última medida de distancia en texto. En caso de que la última medida haya sido inválida se visualizará “\_” en vez de la medida. Deberán indicarse las unidades correctas (“cm” o “pulgadas”) según corresponda.
- Visualización de la última medida de distancia con una barra a modo de vómetro donde el ancho de la barra es de anchura de tantos pixeles como centímetros se miden (no aparece si la medida es errónea).

Escriba el contenido del fichero **measure.cgi** y de las funciones **cgi\_func(...)**, **cgi\_process\_var(...)** y/o **cgi\_process\_data(...)** que sea necesario. (Nota: complete el código HTML impreso para formar el cgi dejando claro lo que se añade o se modifica).

A continuación se muestra la apariencia de la página y el código HTML que la genera.

**Control remoto del medidor de distancia****Modo de Distancia Simple**

Unidades:

- Centímetros  
 Pulgadas

Distancia: \_\_\_\_\_ cm

**Control remoto del medidor de distancia****Modo de Distancia Simple**

Unidades:

- Centímetros  
 Pulgadas

Distancia: 54 cm



```

t 2 style="color: #2e6c80;">Control remoto del medidor de distancia</h2>
t
t <table>
t <tbody>
t <tr>
t <td>
t <p></p>
t </td>
t <td>
t <table style="width: 300px; height: 10px;">
t <tbody>
t <tr>
t <td><p style="text-align: center;"><strong>Modo de Distancia Simple</strong></p></td>
t </tr>
t <tr>
t <td>
t <p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Unidades:</p>
t <form action="measure.cgi" method="get" >
ca 1 <input type="radio" name="unit" value="cm" />
t Centímetros<br />
t </input>
ca 2 <input type="radio" name="unit" value="inch" /> Pulgadas</input>
t <button type="submit">Actualizar unidades</button>
t </form>
t </td>
t </tr>
t <tr>
t <td style="text-align: center;">
t <form action="measure.cgi" method="get">
t <button type="submit" name="measure" value="OK" >Nueva medida</button>
t </form>
t </td>
t </tr>
t <tr>
ca 3 <td>Distancia: %s %s </td>
t </tr>
t <tr>
t <td>
ca 4 <table style="width: %dpx; height: 10px; background-color: red;">
t <tbody><tr><td></td></tr></tbody>
t </table>
t </td>
t </tr>
t </tbody>
t </table>
t </tbody>
t </table>

```

```

#include <Net_Config.h>
#include <stdio.h>
#include <string.h>
#include <externas.h>

float ultima_medida = -1.0;

U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
    U32 len = 0;
    U32 indice = 0;
    char cadena [19];

    switch (env[0]) {
        case 'a' :
            switch (env[2]) {
                case '1':
                    len = sprintf((char *)buf, (const char *)&env[4], (GetUnit()==UNIT_CM) ? "checked='checked'" : "");
                    break;
                case '2':
                    len = sprintf((char *)buf, (const char *)&env[4], (GetUnit()==UNIT_INCH) ? "checked='checked'" : "");
                    break;
                case '3':
                    sprintf(cadena, "%f", ultima_medida);
                    len = sprintf((char *)buf, (const char *)&env[4], (ultima_medida >= 0) ? cadena : "____",
                                (GetUnit()==UNIT_CM) ? "cm" : "inches");
                    break;
                case '4':
                    len = sprintf((char *)buf, (const char *)&env[4], (ultima_medida >= 0) ? (int) ultima_medida : 0);
                    break;
                default:
                    break;
            }
        default:
            break;
    }
    return ((U16)len);
}

void cgi_process_var (U8 *qs) {
    U8 *var;

    var = (U8 *)alloc_mem (40);
    do {
        qs = http_get_var (qs, var, 40);
        if (var[0] != 0) {
            if (str_scomp (var, "unit=cm") == __TRUE) {
                SetCm();
            }
            if (str_scomp (var, "unit=inch") == __TRUE) {
                SetInches();
            }
            if (str_scomp (var, "measure=OK") == __TRUE) {
                ultima_medida = GetMeasure();
            }
        }
    } while (qs);
    free_mem ((OS_FRAME *)var);
}

```

**CUESTIÓN 6.**

Explique qué es un DSP, sus características fundamentales y ponga algún ejemplo de aplicación.

**CUESTIÓN 7**

Se desea utilizar un LPC1768 para realizar las acciones que se indican a continuación. Para cada una de las acciones especifique **la tarea o tareas** que influyen en la ejecutabilidad así como el “Periodo Mínimo de Ejecución” (**T**) y el “Plazo Límite de Finalización” (**D**) de cada tarea justificando la respuesta.

- Comunicación serie asíncrona con 8 bits de datos y paridad par (transmisión y recepción) a 115200 baudios.
- Lectura de tres sensores analógicos cada segundo. El ADC configurado en modo “burst” con un tiempo de conversión de 1ms de manera que cada segundo tome una muestra de cada sensor y se desactive el modo “burst” hasta la siguiente orden de inicio de conversión. El modo “burst” se activa cada segundo con ayuda del Timer 0 y se desactiva con el fin de conversión del último canal.
- Se desea medir la frecuencia de una señal que puede oscilar entre 10kHz y 100kHz para lo que se utiliza un temporizador configurado como contador de pulsos y cada 1ms se calcula la frecuencia en función del número de pulsos medidos de la señal externa.
- Generar dos señales cuadradas: una de 1kHz y otra de 3kHz utilizando el Timer 2.
- Generar una señal acústica de alarma mediante una señal de 1kHz de frecuencia (20 muestras por ciclo) que suene cada 10 segundos durante 1 segundo. Utilizar el DMA para reducir la carga computacional.

- Comunicación serie asíncrona con 8 bits de datos y paridad par (transmisión y recepción) a 115200 baudios.

El tiempo entre eventos se calcula multiplicando el número de bits por el tiempo de bit.

Número de bits: 8 bits de datos + paridad + start + stop = 11 bits

Tiempo de bit: 1/115200 seg.

$T = D = 11/115200 = 95,5 \text{ us}$

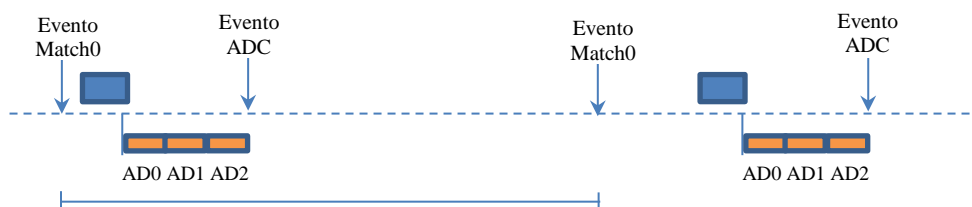
Pueden producirse evento por la transmisión o por la recepción por lo que habría que considerar **dos tareas** diferentes:  $T_{RX}$  y  $T_{TX}$  con los mismos tiempos.

- Lectura de tres sensores analógicos cada segundo (con un error admisible de 10ms). El ADC configurado en modo “burst” con un tiempo de conversión de 1ms de manera que cada segundo tome una muestra de cada sensor y se desactive el modo “burst” hasta la siguiente orden de inicio de conversión. El modo “burst” se activa cada segundo con ayuda del Timer 0.

Habría que configurar el Timer 0 para que interrumpa cada 1 segundo. En esta interrupción se configuraría el ADC en modo “burst” y se iniciaría la conversión.

Esta interrupción podría realizarse sin parar el temporizador utilizando por ejemplo el “Match0” y en cada interrupción avanzando en 1 segundo el valor de temporización.

Habría también que configurar la interrupción del último canal deshabilitando el modo “Burst”.



Por tanto, habría **dos tareas**, una asociada al Timer 0 con un periodo de 1 segundo (**T**). EN el enunciado no hay información suficiente para calcular el Plazo Límite de Finalización (**D**) que sería como máximo de 997ms (habría que asegurar que la conversión finaliza antes de iniciar la siguiente conversión). De todas formas, lo lógico sería que fuera mucho menor, por ejemplo, un  $D = 10\text{ms}$  implicaría un error máximo en la frecuencia de muestreo de un 1%.

En el caso de la interrupción asociada al ADC, el periodo sería también de 1 segundo (T) pero el Plazo Límite de Finalización (D) debería ser  $1\text{seg} - D(\text{Match}0)$ . En el caso de que  $D(\text{Match}0) = 10\text{ms}$   $D(\text{ADC})$  debería ser como máximo 990ms ya que debe leerse antes del siguiente inicio de conversión.

- c) Se desea medir la frecuencia de una señal que puede oscilar entre 10kHz y 100kHz para lo que se utiliza el Timer 1 configurado como contador de pulsos y cada 1ms se calcula la frecuencia en función del número de pulsos medidos de la señal externa. El error máximo debería ser de 10%.

Opción A

El Timer 1 en modo contador no genera interrupciones. Se configura un Match del Timer 0 por ejemplo para que cada 1ms interrumpa, lea el valor del contador y reste el valor de la medida anterior. Con esto sabría cuantos pulsos se han leído cada 1ms y calcula la frecuencia. En este caso sólo estaría la interrupción del Match del Timer 0 cada 1ms (T) y el Plazo Límite de Finalización (D) sería de 100us (un 10%)

Opción B

Se podría configurar el Match asociado al Timer 0 para que conmute cada 1 seg un pin de salida que se conecta a la entrada de captura del Timer 1 configurado para que capture con cualquier flanco. En este caso no habría error en la medida porque la captura sería en el momento adecuado pero habría dos tareas: Timer 0 ( $D=T=1\text{ms}$ ) y captura del Timer 1 ( $D=T=1\text{ms}$ ) que calcularía la frecuencia.

- d) Generar dos señales cuadradas: una de 1kHz y otra de 3kHz utilizando el Timer 2.

Habría que utilizar dos salidas asociadas a dos Match del Timer 2 por lo que habría **dos tareas**, una  $T1\text{kHz}=D1\text{kHz} = 500\text{us}$  y  $T3\text{kHz}=D3\text{kHz} = 166\text{us}$ .

- e) Generar una señal acústica de alarma mediante una señal de 1kHz de frecuencia que suene cada 10 segundos durante 1 segundo. Utilizar el DMA para reducir la carga computacional.

Se pueden guardar muestras de 1 ciclo de la señal de salida y configurar un canal de DMA asociado al DAC que esté enlazado consigo mismo para que lo repita constantemente. Esto no generaría interrupción.

Se podría configurar un Match de un timer para que interrumpa al cabo de 1segundo (donde desactivaría el reloj asociado al DAC) y luego 9 segundos después (donde activaría el reloj del DAC) repitiéndolo de forma periódica.

Se utilizaría una tarea de  $T=D=1\text{s}$  ya que es el caso más restrictivo.

## ANEXO I (Mapa de memoria LPC1768 y Registros ADC)

**Table 3. LPC176x/5x memory usage and details**

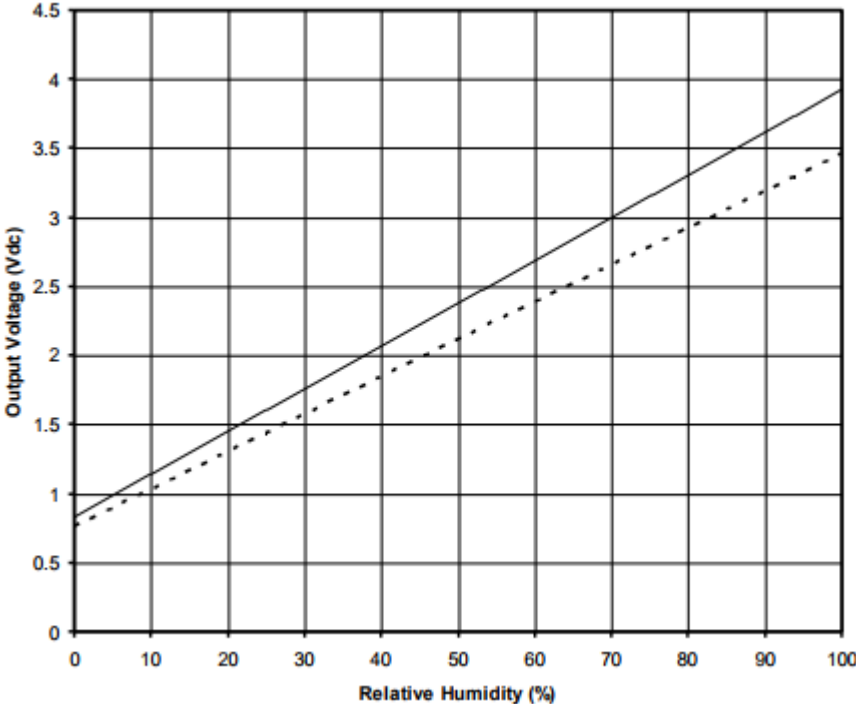
Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
		0x0000 0000 - 0x0003 FFFF	For devices with 256 kB of flash memory.
		0x0000 0000 - 0x0001 FFFF	For devices with 128 kB of flash memory.
		0x0000 0000 - 0x0000 FFFF	For devices with 64 kB of flash memory.
		0x0000 0000 - 0x0000 7FFF	For devices with 32 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
		0x1000 0000 - 0x1000 3FFF	For devices with 16 kB of local SRAM.
0x1000 0000 - 0x1000 1FFF		For devices with 8 kB of local SRAM.	
Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.	
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2007 C000 - 0x2007 FFFF	AHB SRAM - bank 0 (16 kB), present on devices with 32 kB or 64 kB of total SRAM.
		0x2008 0000 - 0x2008 3FFF	AHB SRAM - bank 1 (16 kB), present on devices with 64 kB of total SRAM.
	GPIO	0x2009 C000 - 0x2009 FFFF	GPIO.
0x4000 0000 to 0x5FFF FFFF	APB Peripherals	0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks, 16 kB each.
		0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks, 16 kB each.
	AHB peripherals	0x5000 0000 - 0x501F FFFF	DMA Controller, Ethernet interface, and USB interface.
0xE000 0000 to 0xE00F FFFF	Cortex-M3 Private Peripheral Bus	0xE000 0000 - 0xE00F FFFF	Cortex-M3 related functions, includes the NVIC and System Tick Timer.

**Table 530. ADC registers**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	AD0 Name & Address
ADCR	A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.	R/W	1	AD0CR - 0x4003 4000
ADGDR	A/D Global Data Register. This register contains the ADC's DONE bit and the result of the most recent A/D conversion.	R/W	NA	AD0GDR - 0x4003 4004
ADINTEN	A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt.	R/W	0x100	AD0INTEN - 0x4003 400C
ADDR0	A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0.	RO	NA	AD0DR0 - 0x4003 4010
ADDR1	A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.	RO	NA	AD0DR1 - 0x4003 4014
ADDR2	A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.	RO	NA	AD0DR2 - 0x4003 4018
ADDR3	A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.	RO	NA	AD0DR3 - 0x4003 401C
ADDR4	A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.	RO	NA	AD0DR4 - 0x4003 4020
ADDR5	A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.	RO	NA	AD0DR5 - 0x4003 4024
ADDR6	A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.	RO	NA	AD0DR6 - 0x4003 4028
ADDR7	A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.	RO	NA	AD0DR7 - 0x4003 402C
ADSTAT	A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt/DMA flag.	RO	0	AD0STAT - 0x4003 4030
ADTRM	ADC trim register.	R/W	0x0000 0F00	AD0TRM - 0x4003 4034



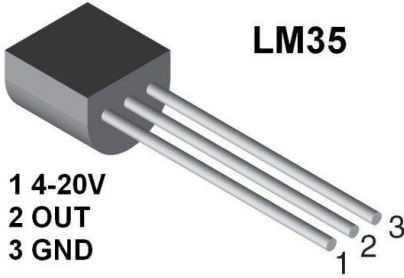
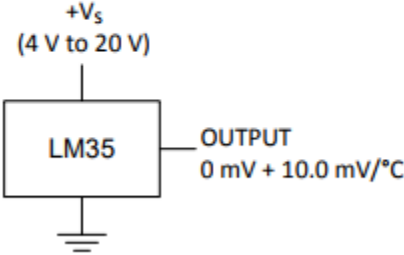
ANEXO II (Características sensores HIH4000 y LM35, Vcc=+5V)



— 0 C  
- - - 70 C



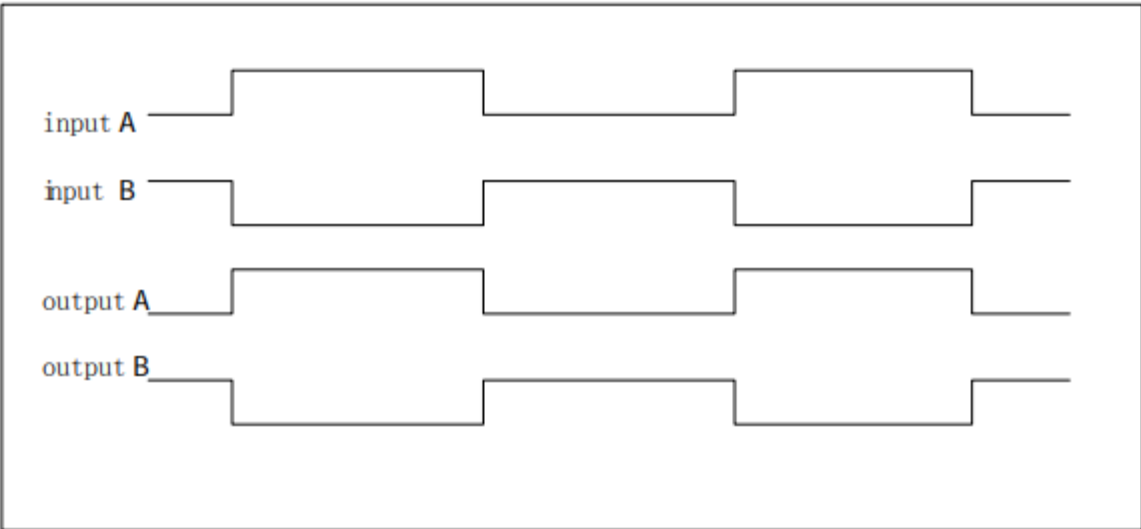
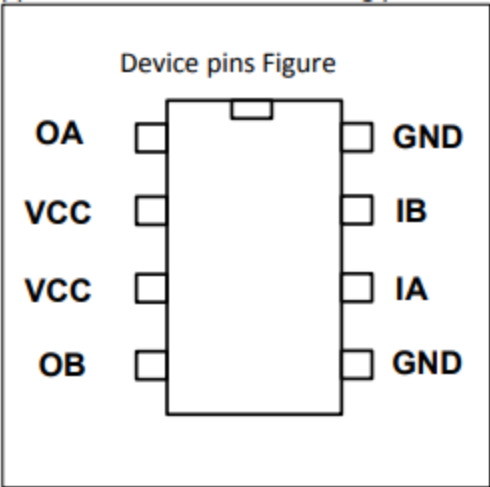
Basic Centigrade Temperature Sensor (2°C to 150°C)



### ANEXO III. Driver ventilador. L9110

Pin definitions:

No.	Symbol	Function
1	OA	A road output pin
2	VCC	Supply Voltage
3	VCC	Supply Voltage
4	OB	B output pin
5	GND	Ground
6	IA	A road input pin
7	IB	B input pin
8	GND	Ground



## ANEXO IV. Funciones de control del Puerto Serie (UART0)

```

void UART0_IRQHandler(void) {
    switch(LPC_UART0->IIR&0x0E) {
    case 0x04: /* RBR, Receiver Buffer Ready */
        *ptr_rx=LPC_UART0->RBR; /* lee el dato recibido y lo almacena */
        if(*ptr_rx++ ==13){ /* Caracter return --> Cadena completa */
            *ptr_rx=0; /* Añadimos el caracter null para tratar los datos recibidos como una cadena*/
            rx_completa = 1; /* rx completa */
            ptr_rx=buffer; /* puntero al inicio del buffer para nueva recepción */
        }
        break;
    case 0x02: /* THRE, Transmit Holding Register empty */
        if(*ptr_tx!=0)
            LPC_UART0->THR = *ptr_tx++; /* carga un nuevo dato para ser transmitido */
        else
            tx_completa=1;
        break;
    }
}

```

```

void tx_cadena_UART0(char *cadena)
{
    ptr_tx=cadena;
    tx_completa=0;
    LPC_UART0->THR = *ptr_tx; // IMPORTANTE: Introducir un carácter al comienzo para iniciar TX o
} // activar flag interrupción por registro transmisor vacío

```

**Table 285: UARTn Fractional Divider Register (U0FDR - address 0x4000 C028, U2FDR - 0x4009 8028, U3FDR - 0x4009 C028) bit description**

Bit	Function	Value	Description	Reset value
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UARTn baudrate.	0
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UARTn to operate properly, regardless of whether the fractional baud-rate generator is used or not.	1
31:8	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

$$\text{UARTn}_{\text{baudrate}} = \frac{PCLK}{16 \times (256 \times \text{UnDLM} + \text{UnDLL}) \times \left(1 + \frac{\text{DivAddVal}}{\text{MulVal}}\right)}$$

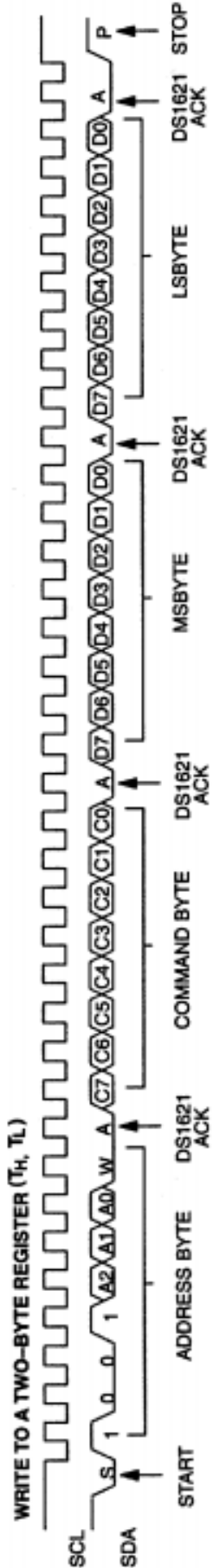
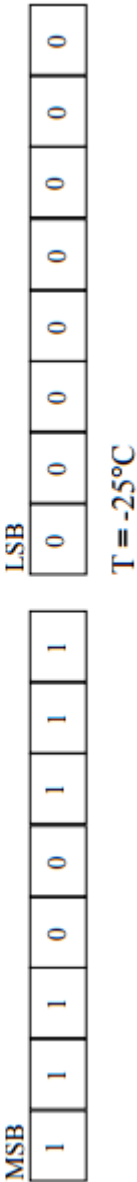
**Table 280: UARTn Line Control Register (U0LCR - address 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C) bit description**

Bit	Symbol	Value	Description
1:0	Word Length Select	00	5-bit character length
		01	6-bit character length
		10	7-bit character length
		11	8-bit character length
2	Stop Bit Select	0	1 stop bit.
		1	2 stop bits (1.5 if UnLCR[1:0]=00).
3	Parity Enable	0	Disable parity generation and checking.
		1	Enable parity generation and checking.
5:4	Parity Select	00	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.
		01	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.
		10	Forced "1" stick parity.
		11	Forced "0" stick parity.
6	Break Control	0	Disable break transmission.
		1	Enable break transmission. Output pin UARTn TXD is forced to logic 0 when UnLCR[6] is active high.
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.
		1	Enable access to Divisor Latches.

**ANEXO V. Protocolo acceso sensor de temperatura DS1621**

THERMOSTAT COMMANDS			
Access TH	Reads or writes high temperature limit value into TH register.	A1h	<write data>
Access TL	Reads or writes low temperature limit value into TL register.	A2h	<write data>
Access Config	Reads or writes configuration data to configuration register.	ACh	<write data>

**Figure 2. TEMPERATURE, TH, and TL FORMAT**



## ANEXO VI. (Funciones de control del Bus I2C)

```

#include <LPC17xx.h>
#define SDA 0 //pin 0
#define SCL 1 //pin 1
void I2Cdelay(void) //retardo minimo de 4.7 us
{
    unsigned char i;
    for(i=0;i<100;i++); //Modificar límite para garantizar los tiempos (Bus standar -->F_max=100kHz)
}
//Genera un pulso de reloj (1 ciclo)
void pulso_SCL(void){
    LPC_GPIO0->FIOSET=(1<<SCL); // Genera pulso de reloj (nivel alto)
    I2Cdelay();
    LPC_GPIO0->FIOCLR=(1<<SCL); // Nivel bajo
    I2Cdelay();}

// Función para escribir un Byte en el Slave. El Master envía al final pulso de reloj para leer ACK
void I2CSendByte(unsigned char byte){
    unsigned char i;
    for(i=0;i<8;i++){
        if (byte &0x80) LPC_GPIO0->FIOSET=(1<<SDA); // envia cada bit, comenzando por el MSB
        else LPC_GPIO0->FIOCLR=(1<<SDA);
        byte = byte <<1; // siguiente bit
        pulso_SCL();}

//Leer ACK que envía el Slave (el Master ha de enviar un pulso de reloj)
// CONFIGURAR PIN SDA COMO ENTRADA; //espera ACK(config. pin como entrada)
LPC_GPIO0->FIODIR&=~(1<<SDA);
pulso_SCL();
// CONFIGURA PIN SDA COMO SALIDA;
LPC_GPIO0->FIODIR|=(1<<SDA); // Dejamos SDA de nuevo como salida
}

//Función que envia START + Byte de dirección del Slave (con bit LSB inicando R/W)
void I2CSendAddr(unsigned char addr, unsigned char rw){
    //CONFIGURAR PINS SDA, SCL COMO SALIDAS; // Por si se nos olvidada en la conf. general.
    LPC_GPIO0->FIODIR|=(1<<SDA)|(1<<SCL);
    LPC_GPIO0->FIOSET|=(1<<SDA)|(1<<SCL);// SDA y SCL a nivel alto para garantizar el
    // nivel de reposo del bus + tiempo.

    I2Cdelay();
    SDA=0; //condicion START: Bajar SDA y luego SCL
    I2Cdelay();
    SCL=0;
    I2Cdelay();
    I2CSendByte((addr=addr<<1) + rw);//envia byte de direccion ->addr, direccion(7bits); rw=1, lect.; rw=0, escrit.
}

// Función para leer un Byte del Slave. El Master envía al final de la lectura
// el bit ACK o NACK (si es último byte leído) que se pasa como argumento de la función.
unsigned char I2CGetByte(unsigned char ACK){
    // ACK = 0, para cualquier byte que no sea el ultimo.
    // ACK = 1 (NACK), despues de leer el ultimo byte
    unsigned char i, byte;
    //CONFIGURAR PIN SDA COMO ENTRADA; //configura pin SDA como entrada
    LPC_GPIO0->FIODIR&=~(1<<SDA);
    for(i=0;i<8;i++){ //lee un bit comenzando por el MSB
        LPC_GPIO0->FIOSET=(1<<SCL);//mientras SCL=1
        I2Cdelay();
        byte=byte<<1;
        if(LPC_GPIO0->FIOPIN&(1<<SDA)) byte++;//Si leemos "1" sumamos para introducir el "1"
        LPC_GPIO0->FIOCLR=(1<<SCL); //Si leemos "0" solo desplazamos (se introduce un "0")
        I2Cdelay();}

//CONFIGURAR PIN SDA COMO SALIDA; // Master envía un ACK por cada byte leído.
LPC_GPIO0->FIODIR|=(1<<SDA);
    if(ACK)LPC_GPIO0->FIOSET=(1<<SDA); // ACK o (NACK) es funcion del último byte leído
    else LPC_GPIO0->FIOCLR=(1<<SDA);
    pulso_SCL(); // Pulso de reloj para su envío
    return (byte);
}

void I2CSendStop(void)
{
    LPC_GPIO0->FIOCLR=(1<<SDA);
    I2Cdelay();
    LPC_GPIO0->FIOSET=(1<<SCL); // Subir SCL, y después SDA!! para dejar el bus en reposo
    I2Cdelay();
    LPC_GPIO0->FIOSET=(1<<SDA);
    I2Cdelay();
}

```