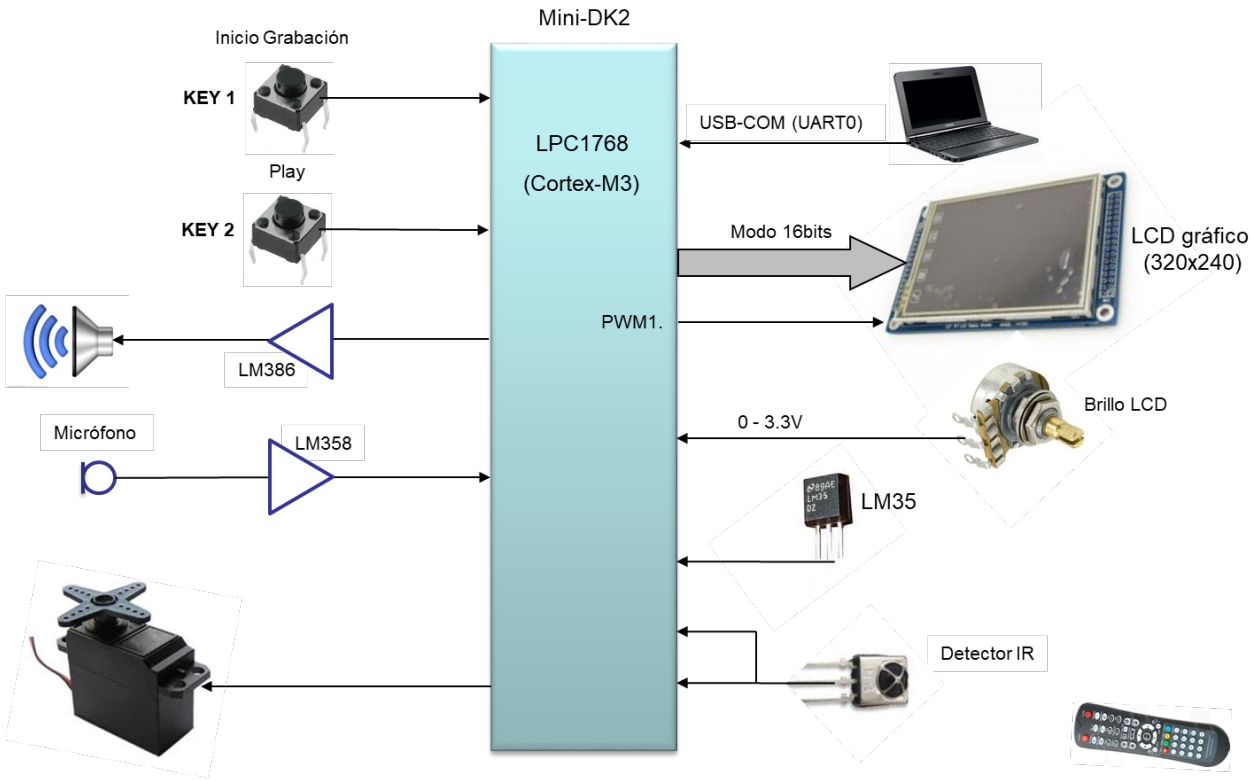
UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES	
ASIGNATURA	SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS	FECHA	MAYO 2018
APELLIDOS, NOMBRE		GRUPO	

PRUEBA DE EVALUACIÓN FINAL

CUESTIÓN 1

El diagrama de la figura muestra un sistema empotrado basado en el LPC1768 (Fcpu=100Mhz) para posicionar un servomotor (0-180º) mediante tonos DTMF que proporciona un teclado numérico. Igualmente podrá controlarse mediante un mando a distancia por infrarrojos, para lo cual se añade un detector de infrarrojos que se conecta a **dos entradas** para detectar a nivel bajo los pulsos que codifican cada tecla del mando modulando una portadora de 38kHz.

El sistema también es capaz de medir la temperatura exterior a partir de un sensor analógico, controlar el brillo del LCD de forma manual mediante un potenciómetro, y generar un mensaje audible de alarma previamente almacenado en memoria. El LCD muestra información y permite monitorizar las variables.



Condiciones de diseño:

El valor de la tensión analógica que proporciona el potenciómetro y el sensor de temperatura se ha de tomar **periódicamente cada segundo**.
Un menú de configuración permitirá seleccionar una opción para grabar en RAM el mensaje de alarma utilizando un micrófono con el correspondiente circuito acondicionador. La pulsación de KEY 1 iniciará el proceso de grabación a una frecuencia de muestreo de 8 kHz (12 bits) durante 2 segundos.
Considere que el SysTick está habilitado y que interrumpe periódicamente cada 50ms, entre otras cosas, para para hacer medidas de tiempos grandes.

a) Complete sobre el diagrama de la figura1, el nombre del pin (**Pn.x**) sobre la línea de conexión y el nombre del recurso utilizado dentro del bloque que representa la Mini-DK2 (ej. **MAT1.0**) excepto el LCD.

b) 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 (Fs=8kHz a 12bits). Considere que funciona por DMA. Calcule la frecuencia de muestreo máxima que admite el sistema para la configuración del ADC mostrada.

```
void init_ADC_microfono(void)
{
LPC_ADC->ADCR= (    1<< )|          //
               (    2<<8)|          //
                  (1<<21)|          // PDN=1
                   (6<<24);         //



}
```

c) Complete la función de configuración del ADC (incluyendo los comentarios) para las entradas del sensor de temperatura y del potenciómetro de brillo del LCD. Considere que frecuencia de muestreo de cada canal sea aproximadamente de 1 kHz.

```
void init_ADC_sensores(void)
{
LPC_ADC->ADCR= (    1<< )|          //
               (     <<8)|          //
                  (1<<21)|          // PDN=1
                   (1<<16);         //
NVIC_DisableIRQ(ADC_IRQn);          //
}
```

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

```
void init_ADC_pines(void)
{
LPC_SC->PCONP|= (1<<12);                          //
LPC_PINCON->PINSEL   |=(       )              ;    //
LPC_PINCON->PINMODE  |=(       )              ;    //
}
```

d) Indique la configuración del DMA para el modo de **procesado de la señal de audio** que ha de detectar los tonos DTMF considerando que se han de almacenar en memoria las muestras del ADC de **12 bits** (1000 muestras)

e) Escriba la función de interrupción del Timer 3 que saca las muestras hacia el DAC para generar la señal de alarma, y el valor del registro **MRx** correspondiente para obtener la señal de salida correctamente.

```
void TIMER3_IRQHandler(void)
{




}
```

f) Explique qué recurso utilizaría y como lo configuraría para reducir la carga de CPU durante la **generación de la señal de alarma**. Indique claramente la **dirección de memoria** a partir de la cual han de estar almacenadas las muestras (ver Anexo 1).

g) Escriba la función de interrupción asociada al Timer que detecta la tecla o código del mando IR considerando que se utiliza una codificación SONY como la mostrada en el Anexo 2.

h) Para notificar que se han recibido datos por DTMF y por el mando a distancia IR, se utilizan las siguientes variables:
- DTMF_Flag: Se pone a 1 cuando se detecta un código DTMF nuevo.
- DTMF_Data: Almacena el código ASCII de la tecla DTMF detectada.
- IR_Flag: Se pone a 1 cuando se detecta un código de mando a distancia nuevo.
- IR_Data: Almacena el código ASCII de la tecla del mando detectado.

Realice el diagrama del StateChart que iría en la función EvaluaStateChart()que decodifique la secuencia introducida y que realice las acciones que se indican a continuación:

#0* → Pone el servo en posición 0 (valor válido de 0 a 180) → Llamaría a SetServo(0)
#23* → Pone el servo en posición 23 grados (valor válido de 0 a 180) → Llamaría a SetServo(23)
#123* → Pone el servo en posición 123 grados (valor válido de 0 a 180) → Llamaría a SetServo(123)
#++* → Aumenta la posición del servo en 10 grados. Llamaría a IncServo()
--* → Disminuye la posición del servo en 10 grados. Llamaría a DecServo()

NOTA1: En caso de que se detecte una secuencia errónea se ignorará el comando.

NOTA2: Suponga escritas las funciones SetServo(), IncServo(), DecServo() y suponga que el programa principal tiene la siguiente estructura:

```
…
while (1)
{
  …
  if (DTMF_Flag || IR_Flag) {
       if (DTMF_Flag) {
        Dat = DTMF_Data;
        DTMF_Flag = 0;
       } else {
        Dat = IR_Data;
        IR_Flag = 0;
       }
      EvaluaStateChart(Dat);
   }
   …
}
```

i) Se desea también poder controlar remotamente el sistema de posición del servomotor implementando un servidor HTTP empotrado en el microcontrolador. La página que el usuario debe visualizar corresponde con la representada en la figura donde se muestra la temperatura, los grados de giro y un dibujo del servomotor que se inclina en función de la posición configurada.

También permite introducir numéricamente la posición a la que se desea que vaya el servo, y moverlo de 10 en 10 grados en un sentido o en otro (aumentando o disminuyendo) al pulsar en los respectivos botones.

Se indica también el código HTML que recibirá el navegador. Esta página incluye código JavaScript para gestionar la recarga automática de la página y para pintar el dibujo del servomotor en la posición deseada.

Escriba el contenido del fichero **posicion.cgi**, y de las funciones **cgi_func(...)**, **cgi_process_var(...)** y/o **cgi_process_data(...)** que sea necesario.

```
<!DOCTYPE html>
<html>
<body>
<h1 style="text-align:center;">Sistema de posicionamiento</h1>

 <table style="width:100%">
  <tr>
    <th>
      <canvas id="myCanvas" width="200" height="200"
              style="border:1px solid #d3d3d3;">
              Your browser does not support the HTML5 canvas tag.
      </canvas>
    </th>
    <th>
       <p><b>Temperatura:</b> 27 &deg;C</p>
       <form action="http://www.seda.es/posicion.cgi" method="get">
          Posición del servomotor:<br>
          <input type="number" name="posicion" min="0" max="180" value=""><br>
          <input type="submit" value="Envia Posición"><br>
       </form>
       <form action="http://www.seda.es/posicion.cgi" method="get">
         <button type="submit" name="Aumenta" value="ON">Aumenta</button>
         <button type="submit" name="Disminuye" value="ON">Disminuye</button>
       </form>
    </th>

  </tr>
 </table>

 <script>

   var grados = 35;

   var c = document.getElementById("myCanvas");
   var ctx = c.getContext("2d");
   ctx.translate(100,100);
   ctx.font="20px Georgia";
   ctx.strokeText(grados+" grados",-50,80);
   ctx.rotate((grados-180) * Math.PI / 180);
   ctx.fillRect(-50, -25, 100, 50);
   ctx.fillStyle="#FF0000";
   ctx.fillRect(0, -10,70, 20);
 </script>
 <script type="text/javascript">
    window.onload = setupRefresh;

    function setupRefresh() {
      setTimeout("refreshPage();", 5000); // milliseconds
    }
    function refreshPage() {
       window.location = "http://www.seda.es/posicion.cgi";
    }
  </script>

 </body>
</html>
```

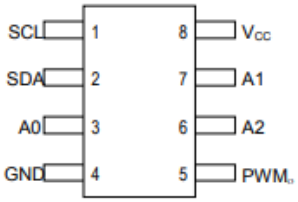
Suponga que existen dos variables globales llamadas “temperatura” y “posición” que contienen la temperatura en ºC y la posición del servomotor configurada en grados.

Suponga también escritas las funciones SetServo(), IncServo() y DecServo()

CUESTIÓN 2

Se desea modificar el control del brillo del LCD utilizando el chip DS1050Z**-001** que proporciona una señal PWM de **1KHz** cuyo ciclo de trabajo es proporcional a un código binario de 5 bits. (ver **Anexo 4 y 5**).

a) Modifique el conexionado del LCD del diagrama de bloques de la página 1 y complete la conexiones del chip con la Mini-DK2.



b) Escriba la función que modifica el brillo a partir del tanto por ciento (0-100) que se pasa como argumento a la función:

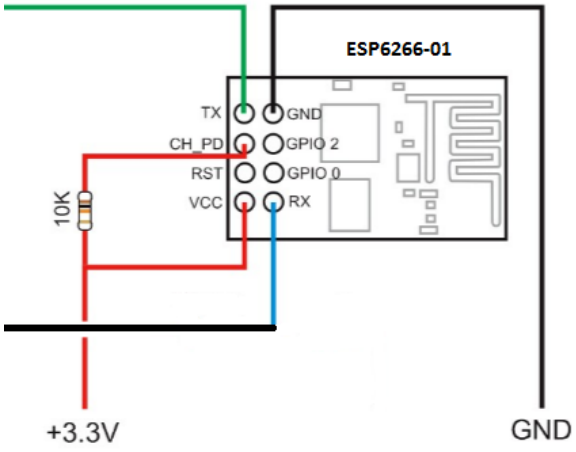
```
void set_brillo_LCD(char brillo)
{




}
```

CUESTIÓN 3

Se desea controlar el sistema propuesto mediante una conexión WIFI, para lo cual se conecta con la **UART3** el módulo ESP8266.

- Complete el diagrama de conexión del LPC1768 con el módulo WIFI. **Señale los pines físicos de conexión.**
- Complete la función de configuración del puerto serie y **realice los cálculos** para configurar la velocidad a 38400 baudios, 8 bits por dato y sin paridad.
- Deduzca **DivAddval** y **Mulval** para mejorar la exactitud de la velocidad real. Ver **Anexo 3.**
- Calcule la velocidad real obtenida y el tiempo que tarda en transmitirse el comando enviado al ejecutarse la sentencia, ***tx_cadena_UART0_3("En espera...\n\r")***. **NOTA: considere ya escrita esta función que saca la cadena de texto por la UART0 y por la UART3.**

```
void uart3_init(void)
{
      LPC_PCONP->SC
      LPC_PINCON->PINSEL  |=          ;
      LPC_UART3->LCR=0x83;
      LPC_UART3->DLL=       ;
      LPC_UART3->DLM=       ;
      LPC_UART3->LCR= 0x03;
}
```



CUESTIÓN 4.

Suponiendo que un sistema basado en el LPC1768 tiene tres tareas con los parámetros que se indican en la tabla, y teniendo en cuenta que existe una **región crítica** en la **Tarea B**, **Tarea C** y en el programa principal de **5ms**,

Tarea	Prioridad	Subprioridad	C	T	D
Tarea A	0	0	1	10	10
Tarea B	1	0	10	50	30
Tarea C	1	1	10	100	100

Nota: Unidades en ms

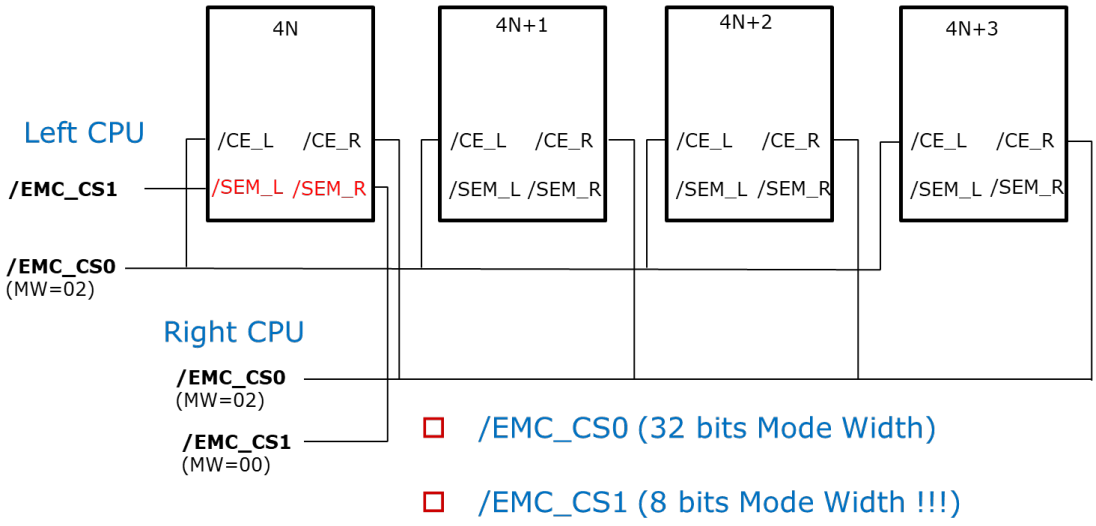
Se pide:

a) Analice si el sistema es ejecutable.
b) Si el sistema sale ejecutable:
   - Calcular la duración máxima de la región crítica en el programa princial que haría que el sistema dejara de ser ejecutable.

   Si el sistema no es ejecutable:
   - Calcular la duración máxima de la región crítica en el programa princial que haría que el sistema fuera ejecutable.

CUESTIÓN 5.

Explique en qué consiste el método de arbitración que se ha empleado para el banco de memoria DUAL_PORT de un sistema basado en el LPC1788, a partir del esquema de la figura. Utilice el mapa de memoria para indicar dónde está mapeada la memoria y los semáforos.
NOTA: Cada chip tiene una capacidad de 64Kx8bits.



Four static memory chip selects:	
0x8000 0000 - 0x83FF FFFF	Static memory chip select 0 (up to 64 MB)
0x9000 0000 - 0x93FF FFFF	Static memory chip select 1 (up to 64 MB)
0x9800 0000 - 0x9BFF FFFF	Static memory chip select 2 (up to 64 MB)
0x9C00 0000 - 0x9FFF FFFF	Static memory chip select 3 (up to 64 MB)

CUESTIÓN 6.
Explique qué es un DSP y sus características fundamentales.

CUESTIÓN 7.
Responda a una de estas dos preguntas:

- Explique qué se entiene por System on Chip comentando algún ejemplo.
- Explique en qué consiste y la ventaja al utilizar la plataforma **Mbed** de ARM

Anexo I (Mapa de memoria y Registros del ADC)

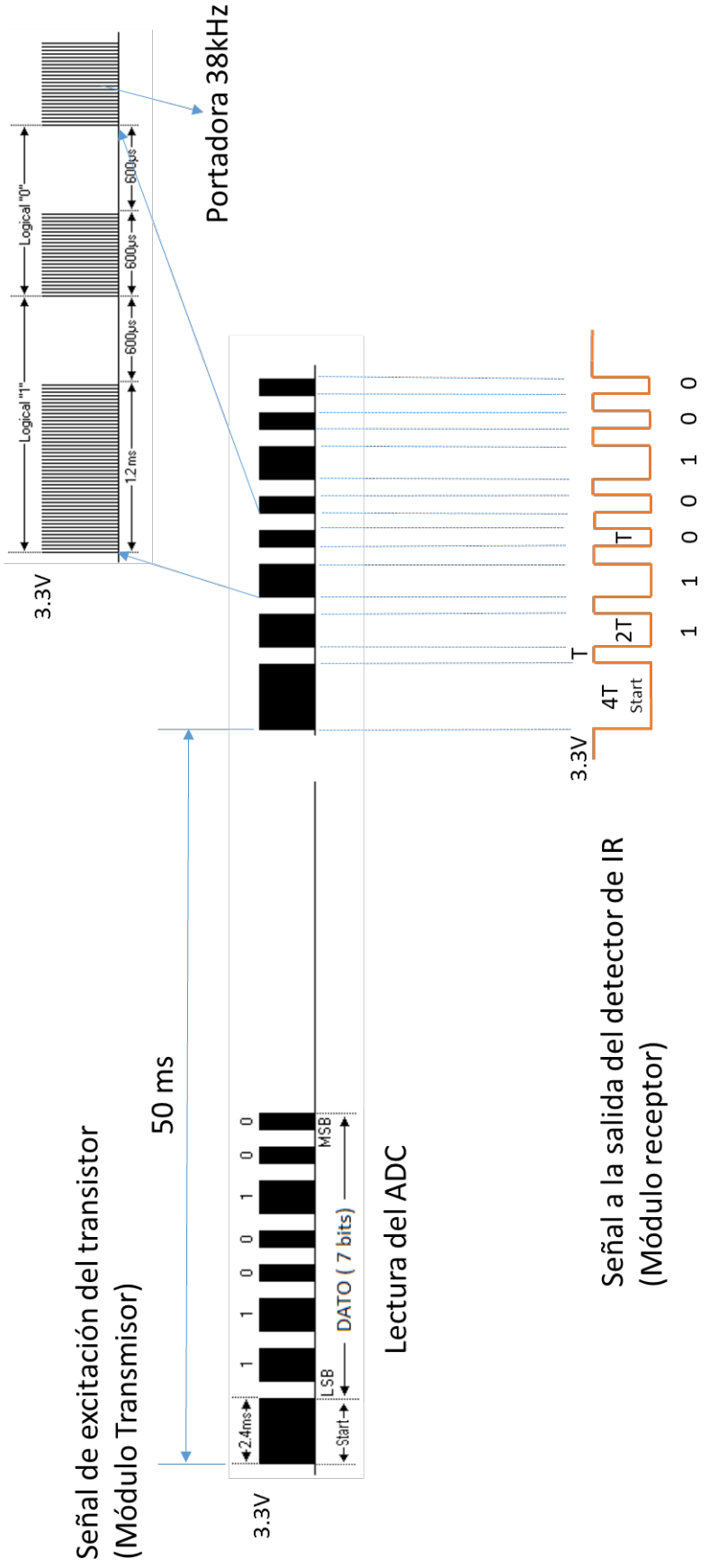
Table 3. LPC176x/5x memory usage and details

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[1]	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 2 (Codificación – Decodificación IR)



Anexo 3. 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 uart0_init(int baudrate) {

    LPC_PINCON->PINSEL0 = (1 << 4) | (1 << 6);     // Change P0.2 and P0.3 mode to TXD0 and RXD0

    // Set 8N1 mode (8 bits/dato, sin paridad, y 1 bit de stop)
    LPC_UART0->LCR |= CHAR_8_BIT | STOP_1_BIT | PARITY_NONE;

    uart0_set_baudrate(baudrate);                 // Set the baud rate

    LPC_UART0->IER = THRE_IRQ_ENABLE | RBR_IRQ_ENABLE; // Enable UART TX and RX interrupt (for LPC17xx UART)
    NVIC_EnableIRQ(UART0_IRQn);                        // Enable the UART interrupt (for Cortex-CM3 NVIC)

}
```

```
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 vacio
```

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

Table 287. Fractional Divider setting look-up table

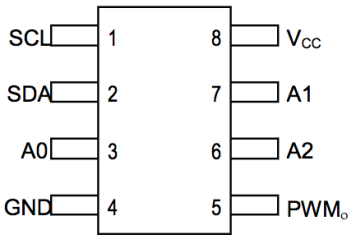
FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal
1.000	0/1	1.250	1/4	1.500	1/2	1.750	3/4
1.067	1/15	1.267	4/15	1.533	8/15	1.769	10/13
1.071	1/14	1.273	3/11	1.538	7/13	1.778	7/9
1.077	1/13	1.286	2/7	1.545	6/11	1.786	11/14
1.083	1/12	1.300	3/10	1.556	5/9	1.800	4/5
1.091	1/11	1.308	4/13	1.571	4/7	1.818	9/11
1.100	1/10	1.333	1/3	1.583	7/12	1.833	5/6
1.111	1/9	1.357	5/14	1.600	3/5	1.846	11/13
1.125	1/8	1.364	4/11	1.615	8/13	1.857	6/7
1.133	2/15	1.375	3/8	1.625	5/8	1.867	13/15
1.143	1/7	1.385	5/13	1.636	7/11	1.875	7/8
1.154	2/13	1.400	2/5	1.643	9/14	1.889	8/9
1.167	1/6	1.417	5/12	1.667	2/3	1.900	9/10
1.182	2/11	1.429	3/7	1.692	9/13	1.909	10/11
1.200	1/5	1.444	4/9	1.700	7/10	1.917	11/12
1.214	3/14	1.455	5/11	1.714	5/7	1.923	12/13
1.222	2/9	1.462	6/13	1.727	8/11	1.929	13/14
1.231	3/13	1.467	7/15	1.733	11/15	1.933	14/15

Anexo 4. DS1050. (5-Bit, Programmable Pulse Width Modulator)

FEATURES

- Single 5-bit, programmable, pulse-width modulator (PWM)
- Adjustable Duty Cycle: 0% to 100%
- 2.7V to 5.5V Operation
- Standard Frequency Values: 1kHz, 5kHz, 10kHz, and 25kHz
- 2-Wire Addressable Interface
- Packages: 8-Pin (150-mil) SOIC and 8-Pin (118-mil) µSOP
- Operating Temperature: -40°C to +85°C

PIN ASSIGNMENT



8-Pin 150-mil SOIC
8-Pin 118-mil µSOP

ORDERING INFORMATION

DS1050Z-001	1kHz	8-Pin 150-mil SOIC
DS1050Z-005	5kHz	8-Pin 150-mil SOIC
DS1050Z-010	10kHz	8-Pin 150-mil SOIC
DS1050Z-025	25kHz	8-Pin 150 mil SOIC
DS1050U-001	1kHz	8-Pin 118-mil µSOP
DS1050U-005	5kHz	8-Pin 118-mil µSOP
DS1050U-010	10kHz	8-Pin 118-mil µSOP
DS1050U-025	25kHz	8-Pin 118-mil µSOP

PIN DESCRIPTION

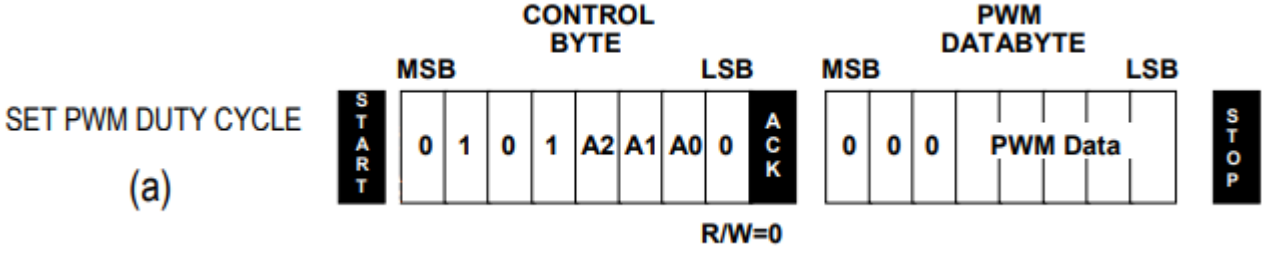
V_{CC}	- 2.7V to 5.5V Power Supply
PWM_O	- PWM Ouput
A0, A1, A2	- Device Address
SDA	- Serial Data I/O
SCL	- Serial Clock Input
GND	- Ground

DESCRIPTION

The DS1050 is a programmable, 5-bit, pulse-width modulator featuring a 2-wire addressable controlled interface. The DS1050 operates from power supplies ranging from 2.7V up to 5.5V. The PWM output provides a signal that swings from 0V to V_{CC}. The DS1050 requires a typical operating current of 50µA and a programmable shutdown supply current of 1µA.

Four standard PWM output frequencies are offered and include 1kHz, 5kHz, 10kHz, and 25kHz. The 2-wire addressable interface allows operation of multiple devices on a single 2-wire bus and provides compatibility with other Dallas Semiconductor 2-wire devices such as real-time clocks (RTCs), digital thermometers, and digital potentiometers.

The device is ideal for low-cost LCD contrast and/or brightness control, power supply voltage adjustment, and battery charging or current adjustment. The DS1050 is offered in standard integrated circuit packaging including the 8-pin (150-mil) SOIC and space-saving 8-pin (118-mil) µSOP.



Anexo 5. FUNCIONES DEL BUS i2c

```
#include <LPC17xx.h>
#define SDA 0
#define SCL 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();
}

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 envía 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 de START: Bajar SDA y luego SCL
 I2Cdelay();
 SCL=0;
 I2Cdelay();
 I2CSendByte((addr=addr<<1) + rw);   //envia byte de direccion //addr, direccion (7bits)
                                     //rw=1, lectura.  //rw=0, escritura
}

// 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();
}
```