UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES	
ASIGNATURA	SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS	**FECHA**	**MARZO 2014**
APELLIDOS, NOMBRE		**GRUPO**	

PRUEBA DE EVALUACIÓN INTERMEDIA

Ejercicio 1 (20 puntos)

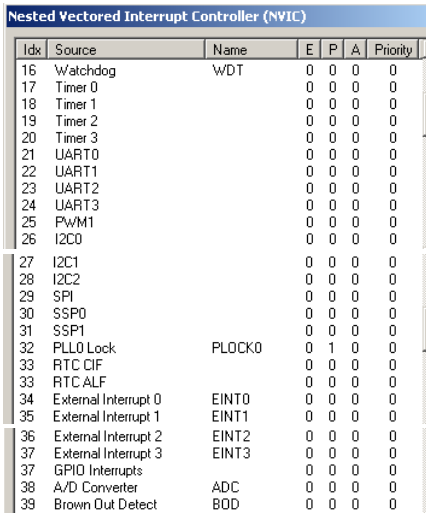
Escribe la función necesaria para medir la frecuencia de una señal periódica empleando la tarjeta Mini-DK2, actualizando en el LCD (modo 16 bits) el valor de la medida cada segundo desde el programa principal. Indica el pin que utilizas como entrada y su configuración. **NOTA:** No se permite emplear funciones de retardo.
*void GUI_Text(uint16_t Xpos, uint16_t Ypos, uint8_t *str,uint16_t Color, uint16_t bkColor)*

Ejercicio 2 (50 puntos)

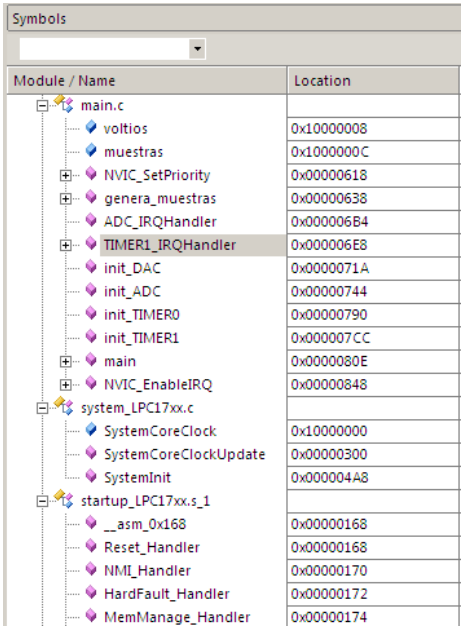
El proyecto cuyo código se muestra en **ANEXO I**, trata de demostrar el funcionamiento del ADC y DAC del LPC1768.

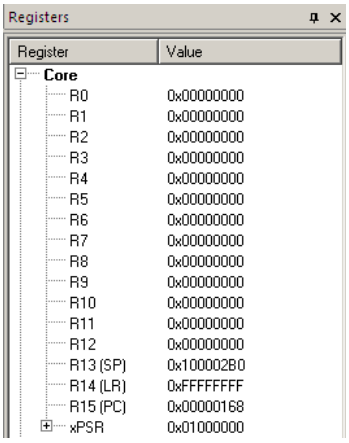
El ADC se configura para convertir el canal 0 de forma que el inicio de conversión se lleve a cabo periódicamente por medio del Timer 0 en Modo Match (MAT0.1 -->Comparación). La frecuencia de muestreo se lleva a cabo en el flanco de subida del pin asociado (P1.29) configurando el modo *Toggle* del Timer 0. Se ha habilitado dicha salida con objeto de trazar la señal cuadrada (F. muestreo) en el analizador lógico o en un osciloscopio. Así, cada dos Match (en flanco de subida) se muestrea el canal 0. El ADC trabaja por interrupción. El DAC se utiliza para generar una señal senoidal de 1 KHz a través del pin P0.23 (AOUT). Para ello se almacenan en un array 32 muestras o valores discretos (10 bits) de un periodo de una señal senoidal (0-3.3V) con un offset de 1.65V (Vrefp/2).

a) A partir de las ventanas capturadas en el simulador (después de un reset), conteste a las siguientes cuestiones:



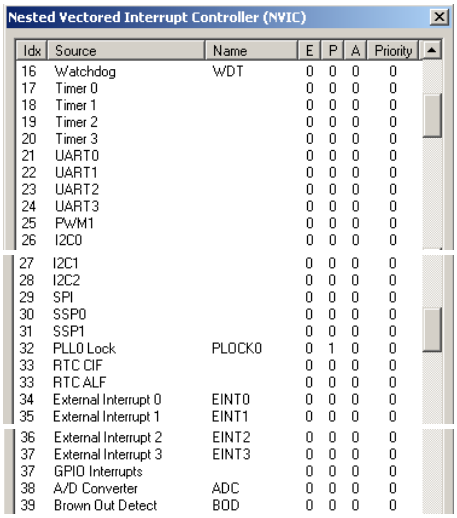
Idx	Source	Name	E	P	A	Priority
16	Watchdog	WDT	0	0	0	0
17	Timer 0		0	0	0	0
18	Timer 1		0	0	0	0
19	Timer 2		0	0	0	0
20	Timer 3		0	0	0	0
21	UART0		0	0	0	0
22	UART1		0	0	0	0
23	UART2		0	0	0	0
24	UART3		0	0	0	0
25	PWM1		0	0	0	0
26	I2C0		0	0	0	0
27	I2C1		0	0	0	0
28	I2C2		0	0	0	0
29	SPI		0	0	0	0
30	SSP0		0	0	0	0
31	SSP1		0	0	0	0
32	PLL0 Lock	PLOCK0	0	1	0	0
33	RTC CIF		0	0	0	0
33	RTC ALF		0	0	0	0
34	External Interrupt 0	EINT0	0	0	0	0
35	External Interrupt 1	EINT1	0	0	0	0
36	External Interrupt 2	EINT2	0	0	0	0
37	External Interrupt 3	EINT3	0	0	0	0
37	GPIO Interrupts		0	0	0	0
38	A/D Converter	ADC	0	0	0	0
39	Brown Out Detect	BOD	0	0	0	0





a.1) Muestre detalladamente (DIRECCIÓN, CONTENIDO) de la tabla de vectores, lo referente al vector de reset y a las fuentes de interrupción del programa (VTOR=0x00). (5 ptos.)

a.2) Modifique sobre la ventana del NVIC los cambios que ocurren tras la ejecución de las funciones de configuración. (2 pto.)



b) Represente gráficamente la evolución temporal del *timer* que genera la frecuencia de muestreo del ADC indicando el valor exacto del Match, la señal que obtendríamos en P1.29, y una señal triangular aplicada por AIN0 (f=20Hz) indicando los instantes de muestreo. (8 ptos.)

c) Completa la instrucción que configura la frecuencia de interrupción del Timer 1 e indica su valor. (5 ptos.)

```
void init_TIMER1(void)
{
    LPC_SC->PCONP|=(1<<2);              //
    LPC_TIM1->PR = 0x00;                //
    LPC_TIM1->MCR = 0x03;               // Reset TC on Match, and Interrupt!
    LPC_TIM1->MR0 =                  ;  //
    LPC_TIM1->EMR = 0x02;               //
    LPC_TIM1->TCR = 0x01;               //
    NVIC_EnableIRQ(TIMER1_IRQn);        //
    NVIC_SetPriority(TIMER1_IRQn,1);    //
}
```

d) Completa la instrucción que configura la frecuencia de muestreo del ADC e indica su valor. (5 ptos.)

```
void init_TIMER0(void)
{
    LPC_SC->PCONP|=(1<<1);              //
    LPC_PINCON->PINSEL3|= 0x0C000000;   //
    LPC_TIM0->PR = 0x00;                //
    LPC_TIM0->MCR = 0x10;               //
    LPC_TIM0->MR1 =                  ; // Se han de producir DOS Match para iniciar la conversión!!!!
    LPC_TIM0->EMR = 0x00C2;             //
    LPC_TIM0->TCR = 0x01;               //
}
```

e) ¿Calcula la frecuencia de interrupción del ADC y su tiempo de conversión? (5 ptos.)

f) ¿Qué modificación es necesario introducir (si crees que es necesario) en la función de interrupción del ADC si se duplica el número de muestras por ciclo? (5 ptos.)

g) ¿Qué recurso utilizarías para reducir la carga de CPU a la hora de generar la señal senoidal? Explica sin escribir el código como lo configurarías (qué parámetros) y explica las funciones asociadas que sería necesario programar, reescribiendo el código de la función *genera_muestras()*. (15 ptos.)

Ejercicio 3 (20 puntos)

Realiza un programa para controlar 2 servomotores de forma que su posición sea proporcional a la posición de 2 potenciómetros, actualizando dicha posición cada 50 ms. Dibuja el diagrama de conexión y configura adecuadamente los recursos que creas necesarios. El valor de la posición de cada servo se ha de enviar por el puerto serie (UART0) periódicamente cada segundo a 115200 baudios, siguiendo la siguiente secuencia:

servo 1: xxx grados, servo 2: yyy grados \n\r

NOTA: El puerto serie ha de funcionar por interrupción (**ver Anexo II**).

PWM servos: (1ms<TH<2ms), Periodo 15ms.

Completa la siguiente función de configuración de la velocidad del puerto serie (considere FR=1)

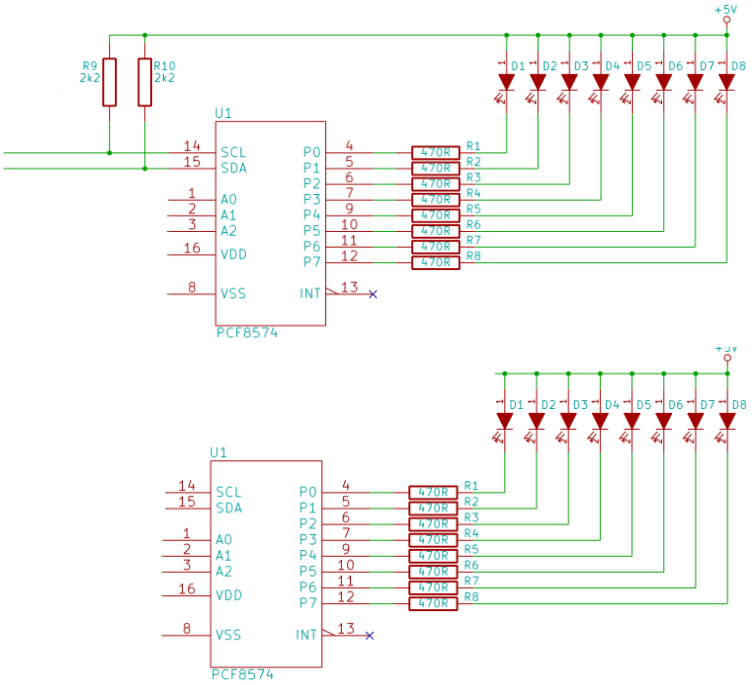
```
void uart0_set_baudrate(int baudrate)
{
LPC_UART0->LCR |= DLAB_ENABLE;
LPC_UART0->DLM =
LPC_UART0->DLL =
LPC_UART0->LCR &= ~DLAB_ENABLE;
}
```

```
#define CHAR_8_BIT
#define STOP_1_BIT
#define PARITY_NONE
#define DLAB_ENABLE
#define RBR_IRQ_ENABLE
#define THRE_IRQ_ENABLE
```

Continuación (Ejercicio 3)

Ejercicio 4 (10 puntos)

Completa las conexiones de los dos chips PCF8574 con el LPC1768 y escribe la función ***I2C_write_16bits(int dato)*** que escribe un dato sobre los 16 LEDs. Considere ya escritas las funciones de control del bus I2C.




```
/*Funciones de control del bus I2C*/
void I2CSendByte(unsigned char byte)
{
 unsigned char i;
 for(i=0;i<8;i++)
       {if (byte &0x80) SDA=1;
        else SDA=0;
        SCL=1;
        I2Cdelay();
        SCL=0;
        I2Cdelay();
        byte = byte <<1;
       }
 SDA=1;          //espera ACK(config.lectura)
 SCL=1;
 I2Cdelay();
 SCL=0;
 I2Cdelay();
}

void I2CSendAddr(unsigned char addr, unsigned char rw)
{
 SCL=1;
 I2Cdelay();
 SDA=0;                                 //condicion de STAR
 I2Cdelay();
 SCL=0;
 I2Cdelay();
 I2CSendByte((addr=addr<<1) + rw);  //envia byte de direccion
}                                       //addr, direccion (7bits)
                                        //rw=1, lectura
                                        //rw=0, escritura
void I2CSendStop(void)
{
 SDA=0;
 I2Cdelay();
 SCL=1;
 I2Cdelay();
 SDA=1;
 I2Cdelay();
}
```

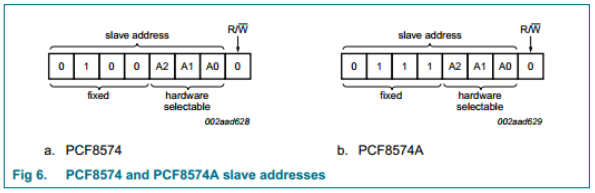


Fig 6. PCF8574 and PCF8574A slave addresses

ANEXO I (Código del programa Ejercicio 2)

```
#include <LPC17xx.H>
#include <Math.h>
#define F_cpu 100e6   // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4 // Defecto despues del reset
#define F_muestreo 100 // Fs=100Hz (Cada 10ms se toma una muestra del canal 0)
#define pi 3.14159
#define F_out 1000
#define N_muestras 32
#define V_refp 3.3

uint16_t muestras[N_muestras];      // Array para guardar las muestras de un ciclo de un seno
float voltios;

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)); // Ojo! el DAC es de 10bits
}

void ADC_IRQHandler(void)
{
  voltios= ((LPC_ADC->ADGDR >>4)&0xFFF)*3.3/4095; // se borra automat. el flag DONE al leer ADCGDR
}

// Timer 1 interrumpe periódicamente a F = F_out x N_muestras !!!!
// La muestra correpondiente del array,  se saca al DAC en cada interrupción
void TIMER1_IRQHandler(void)
{
  static uint8_t indice_muestra;
  LPC_TIM1->IR|= (1<<0);            //
  LPC_DAC->DACR= muestras[indice_muestra++] << 6; // ?
  indice_muestra&= 0x1F;            //
}

void init_DAC(void)
{
  LPC_PINCON->PINSEL1|= (2<<20);    // DAC output = P0.26 (AOUT)
  LPC_PINCON->PINMODE1|= (2<<20);   // Deshabilita pullup/pulldown
  LPC_SC->PCLKSEL0|= (0x00<<22);    // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
  LPC_DAC->DACCTRL=0;               //
}
```

```
void init_ADC(void)
{
  LPC_SC->PCONP|= (1<<12);           // POwer ON
  LPC_PINCON->PINSEL1|= (1<<14);     // ADC input= P0.23 (AD0.0)
  LPC_PINCON->PINMODE1|= (2<<14);    // Deshabilita pullup/pulldown
  LPC_SC->PCLKSEL0|= (0x00<<8);      // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
  LPC_ADC->ADCR= (0x01<<0)|          // Canal 0
                 (0x01<<8)|          // CLKDIV=1   (Fclk_ADC=25Mhz /(1+1)= 12.5Mhz)
                 (0x01<<21)|         // PDN=1
                 (4<<24);            // Inicio de conversión con el Match 1 del Timer 0

  LPC_ADC->ADINTEN= (1<<0)|(1<<8);   // Hab. interrupción fin de conversión canal 0
  NVIC_EnableIRQ(ADC_IRQn);          //
  NVIC_SetPriority(ADC_IRQn,2);      //
}

/*  Timer 0 en modo Output Compare (reset TOTC on Match 1)
  Counter clk: 25 MHz   MAT0.1 : On match, Toggle pin/output (P1.29)
  Cada 2 Match se provoca el INICIO DE CONVERSIÓN DEL ADC
  Habilitamos la salida (MAT0.1) para observar la frecuencia de muestreo del ADC */

void init_TIMER0(void)
{
    LPC_SC->PCONP|=(1<<1);             //
    LPC_PINCON->PINSEL3|= 0x0C000000; //
    LPC_TIM0->PR = 0x00;               //
    LPC_TIM0->MCR = 0x10;              //
    LPC_TIM0->MR1 =                  ; // Se han de producir DOS Match para iniciar la conversión!!!!
    LPC_TIM0->EMR = 0x00C2;            //
    LPC_TIM0->TCR = 0x01;              //
}


/*  Timer 1 en modo Output Compare (reset TOTC on Match 0)
  Counter clk: 25 MHz   MAT1.0 : On match, salida de una muestra hacia el DAC */
void init_TIMER1(void)
{
    LPC_SC->PCONP|=(1<<2);             //
    LPC_TIM1->PR = 0x00;               //
    LPC_TIM1->MCR = 0x03;              // Reset TC on Match, and Interrupt!
    LPC_TIM1->MR0 =                ;  //
    LPC_TIM1->EMR = 0x02;              //
    LPC_TIM1->TCR = 0x01;              //
    NVIC_EnableIRQ(TIMER1_IRQn);       //
    NVIC_SetPriority(TIMER1_IRQn,1);  //
}

int main(void)
{
  NVIC_SetPriorityGrouping(2);
  genera_muestras(N_muestras);
  init_ADC();
  init_DAC();
  init_TIMER0();
  init_TIMER1();
  while(1);
}
```

ANEXO II (Código ejemplo 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
```