
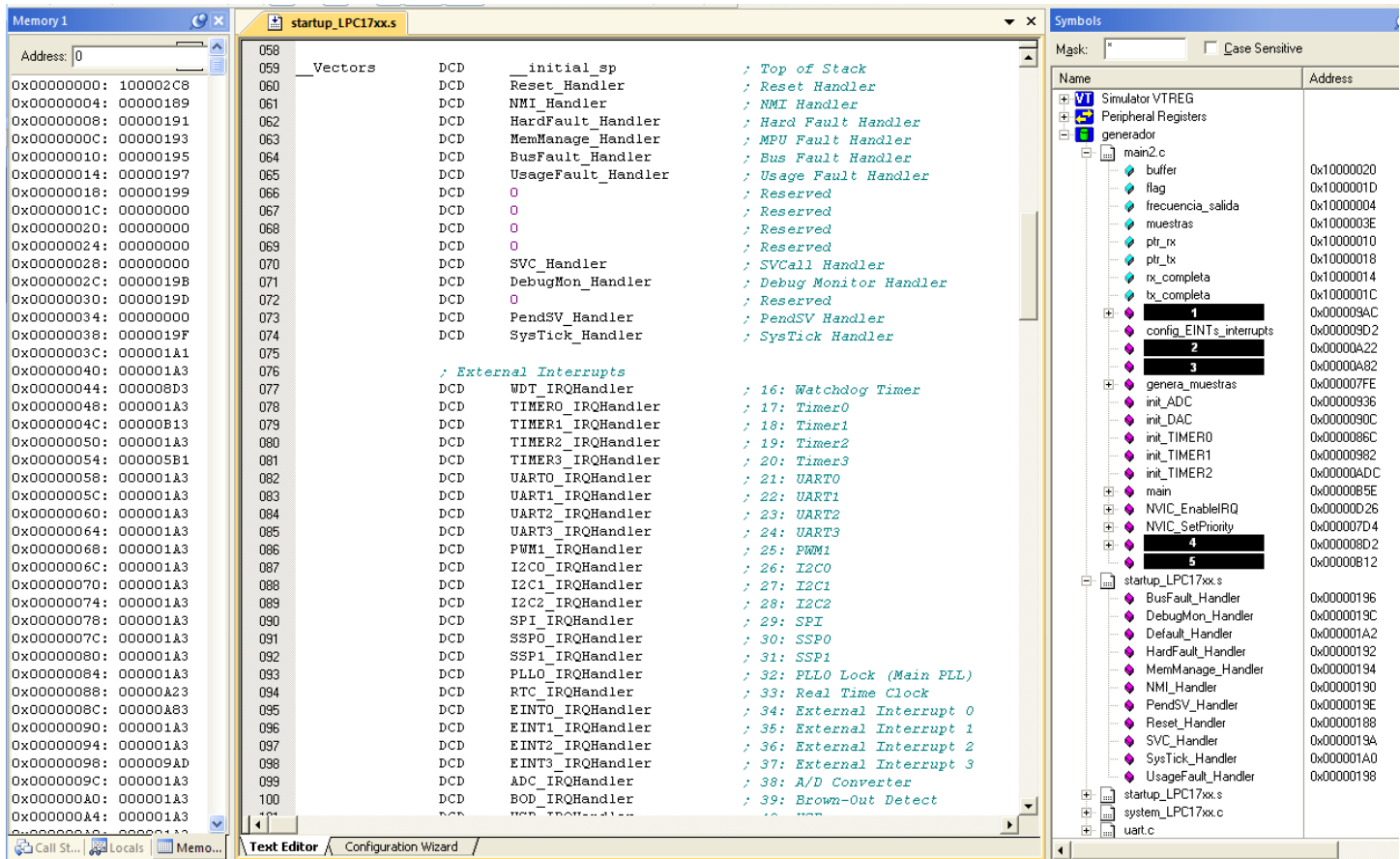
 UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		 GRADO EN INGENIERÍA EN TECNOLOGÍAS TELECOMUNICACIÓN	
ASIGNATURA	SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS	FECHA	6-10- 2014
APELLIDOS, NOMBRE		GRUPO	

PRUEBA DE EVALUACIÓN INTERMEDIA

Ejercicio 1 (10 puntos)

Atendiendo al volcado de memoria y al contenido del fichero *Startup_LPC17xx.s* que se muestran en la siguiente figura, indicar los nombres de los manejadores de interrupción que están ocultos en la ventana *Symbols*:



The screenshot displays three windows from a development environment:

- Memory 1:** Shows a memory dump starting at address 0x00000000, with values like 100002C8, 00000189, etc.
- startup_LPC17xx.s:** Shows assembly code for vectors and interrupt handlers. Key entries include:
 - 058: `_Vectors` (DCD) `_initial_sp` (comment: `; Top of Stack`)
 - 059: `Reset_Handler` (comment: `; Reset Handler`)
 - 061: `NMI_Handler` (comment: `; NMI Handler`)
 - 062: `HardFault_Handler` (comment: `; Hard Fault Handler`)
 - 063: `MemManage_Handler` (comment: `; MPU Fault Handler`)
 - 064: `BusFault_Handler` (comment: `; Bus Fault Handler`)
 - 065: `UsageFault_Handler` (comment: `; Usage Fault Handler`)
 - 066: `Reserved` (comment: `; Reserved`)
 - 067: `Reserved` (comment: `; Reserved`)
 - 068: `Reserved` (comment: `; Reserved`)
 - 069: `Reserved` (comment: `; Reserved`)
 - 070: `SVC_Handler` (comment: `; SVCcall Handler`)
 - 071: `DebugMon_Handler` (comment: `; Debug Monitor Handler`)
 - 072: `Reserved` (comment: `; Reserved`)
 - 073: `PendSV_Handler` (comment: `; PendSV Handler`)
 - 074: `SysTick_Handler` (comment: `; SysTick Handler`)
 - 075: `External Interrupts` (comment: `; External Interrupts`)
 - 077: `WDT_IRQHandler` (comment: `; 16: Watchdog Timer`)
 - 078: `TIMER0_IRQHandler` (comment: `; 17: Timer0`)
 - 079: `TIMER1_IRQHandler` (comment: `; 18: Timer1`)
 - 080: `TIMER2_IRQHandler` (comment: `; 19: Timer2`)
 - 081: `TIMER3_IRQHandler` (comment: `; 20: Timer3`)
 - 082: `UART0_IRQHandler` (comment: `; 21: UART0`)
 - 083: `UART1_IRQHandler` (comment: `; 22: UART1`)
 - 084: `UART2_IRQHandler` (comment: `; 23: UART2`)
 - 085: `UART3_IRQHandler` (comment: `; 24: UART3`)
 - 086: `PWM1_IRQHandler` (comment: `; 25: PWM1`)
 - 087: `I2C0_IRQHandler` (comment: `; 26: I2C0`)
 - 088: `I2C1_IRQHandler` (comment: `; 27: I2C1`)
 - 089: `I2C2_IRQHandler` (comment: `; 28: I2C2`)
 - 090: `SPI_IRQHandler` (comment: `; 29: SPI`)
 - 091: `SSP0_IRQHandler` (comment: `; 30: SSP0`)
 - 092: `SSP1_IRQHandler` (comment: `; 31: SSP1`)
 - 093: `PLL0_IRQHandler` (comment: `; 32: PLL0 Lock (Main PLL)`)
 - 094: `RTC_IRQHandler` (comment: `; 33: Real Time Clock`)
 - 095: `EINT0_IRQHandler` (comment: `; 34: External Interrupt 0`)
 - 096: `EINT1_IRQHandler` (comment: `; 35: External Interrupt 1`)
 - 097: `EINT2_IRQHandler` (comment: `; 36: External Interrupt 2`)
 - 098: `EINT3_IRQHandler` (comment: `; 37: External Interrupt 3`)
 - 099: `ADC_IRQHandler` (comment: `; 38: A/D Converter`)
 - 100: `BOD_IRQHandler` (comment: `; 39: Brown-Out Detect`)
- Symbols:** Shows a list of symbols with their addresses. Key entries include:
 - `Simulator VREG` (Address: 0x1000001D)
 - `Peripheral Registers` (Address: 0x10000004)
 - `generador` (Address: 0x10000018)
 - `main2.c` (Address: 0x10000010)
 - `buffer` (Address: 0x10000020)
 - `flag` (Address: 0x1000001D)
 - `frecuencia_salida` (Address: 0x10000004)
 - `muestras` (Address: 0x1000003E)
 - `plt_rx` (Address: 0x10000010)
 - `plt_tx` (Address: 0x10000018)
 - `rx_completa` (Address: 0x10000014)
 - `tx_completa` (Address: 0x1000001C)
 - `1` (Address: 0x000009AC)
 - `config_EINTs_interrupts` (Address: 0x000009D2)
 - `2` (Address: 0x00000A22)
 - `3` (Address: 0x00000A82)
 - `genera_muestras` (Address: 0x000007FE)
 - `init_ADC` (Address: 0x00000936)
 - `init_DAC` (Address: 0x0000090C)
 - `init_TIMER0` (Address: 0x0000086C)
 - `init_TIMER1` (Address: 0x00000982)
 - `init_TIMER2` (Address: 0x00000ADC)
 - `main` (Address: 0x0000085E)
 - `NVIC_EnableIRQ` (Address: 0x00000D26)
 - `NVIC_SetPriority` (Address: 0x00000D44)
 - `4` (Address: 0x000008D2)
 - `5` (Address: 0x00000812)
 - `startup_LPC17xx.s` (Address: 0x00000196)
 - `BusFault_Handler` (Address: 0x0000019C)
 - `DebugMon_Handler` (Address: 0x000001A2)
 - `Default_Handler` (Address: 0x00000192)
 - `HardFault_Handler` (Address: 0x00000194)
 - `MemManage_Handler` (Address: 0x00000190)
 - `NMI_Handler` (Address: 0x00000198)
 - `PendSV_Handler` (Address: 0x0000019E)
 - `Reset_Handler` (Address: 0x00000188)
 - `SVC_Handler` (Address: 0x0000019A)
 - `SysTick_Handler` (Address: 0x000001A0)
 - `UsageFault_Handler` (Address: 0x00000198)
 - `startup_LPC17xx.s` (Address: 0x00000196)
 - `system_LPC17xx.c` (Address: 0x00000198)
 - `uart.c` (Address: 0x00000198)

(1)

(2)

(3)

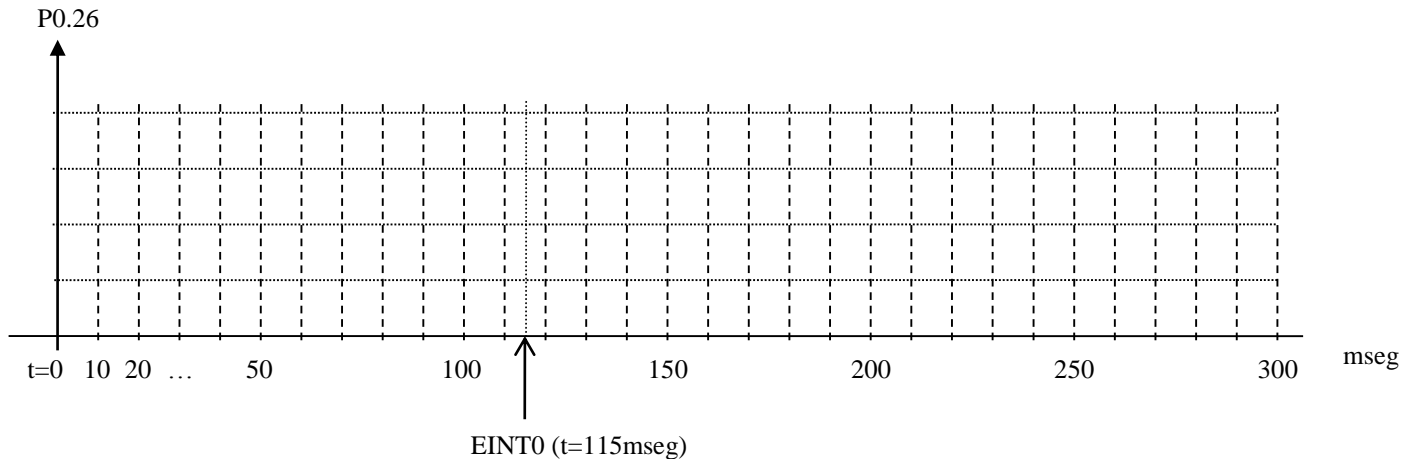
(4)

(5)

Ejercicio 2 (30 puntos)

Se ha diseñado una aplicación para la generación de una señal triangular periódica con el LPC1768. Para ello se ha elegido una V_{REF_P} igual a 3'3v y una V_{REF_N} igual a 0v. La frecuencia de reloj del núcleo del microcontrolador es igual a 100MHz. El código del programa se muestra en el Anexo 1.

a) Dibuje en el siguiente cronograma el valor de la tensión de salida del pin P0.26, considerando que en el instante señalado se produce una interrupción externa EINT0. (10 ptos.)



b) A la vista del resultado, ¿qué solución propone para que la forma de onda sea periódica? (10 ptos.)

c) Si la llamada a la función *pinta_LCD()* se realizase dentro del manejador del TIMER0, ¿cuál podría ser la duración máxima de dicha función para que la forma de onda generada con el DAC no se viese afectada? (10 ptos.)

Ejercicio 3 (60 puntos)

Se desea implementar el control de un servomotor mediante una señal PWM cuyo periodo sea de **15ms**, y el tiempo a nivel alto entre **0.5-2.3ms**, para un movimiento de su posición entre 0° y 180° .

La posición en grados se modificará en pasos de **20°** a través de los pulsadores **KEY1** (para aumentar) y **KEY2** (para disminuir) conectados a dos entradas de interrupción (Nivel 3 de prioridad), o mediante un potenciómetro conectado a **AD0.3** del ADC.

Un interruptor (**Select_Servo**) conectado a **P0.0** que permitirá llevar al servo a su posición central, cuando esté cerrado.

El sistema ha de permitir, en tiempo real, medir el ancho del pulso de la señal PWM con una resolución de **$1\mu s$** , a partir de una interconexión entre la salida PWM y una entrada al microcontrolador, guardando el resultado en una variable entera (**TH_PWM**).

Por el puerto serie (UART0) se ha de enviar a 115200 baudios (8bits/dato, sin paridad, 1 bit de Stop) la posición en grados del servo cada **0.5** segundos **si se produce un cambio en su posición**. Considere ya escritas las funciones de la UART (**Anexo 3**).

- a)** A partir del esquema eléctrico de la Mini-DK2 (**Anexo 2**), proponga una interconexión de los distintos dispositivos externos necesarios, considerando que el LCD (modo 16 bits) ha de estar operativo.

NOTA: Señale claramente sobre el Pin, la funcionalidad utilizada.

(5 ptos.)



LPC1768
(Mini-DK2)

Complete la función de configuración de la señal PWM, y de actualización de la posición del servo ($F_{cpu} = 100\text{MHz}$). (10 ptos.)

```
void config_pwm(void)
{
    LPC_PINCON->PINSEL |=
    LPC_SC->PCONP|=
    LPC_PWM1->MR0=
    LPC_PWM1->PCR|=
    LPC_PWM1->MCR|=(1<<1);
    LPC_PWM1->TCR|=(1<<0)|(1<<3);
}
```

```
void set_servo(uint8_t grados)
{
    LPC_PWM1->MR
    LPC_PWM1->LER
}
```

- b)** Escriba la función de interrupción que permite realizar la medida del ancho del pulso (TH) en **microsegundos**, de la señal PWM, y explique la configuración del recurso utilizado sin escribir el código. (10 ptos.)

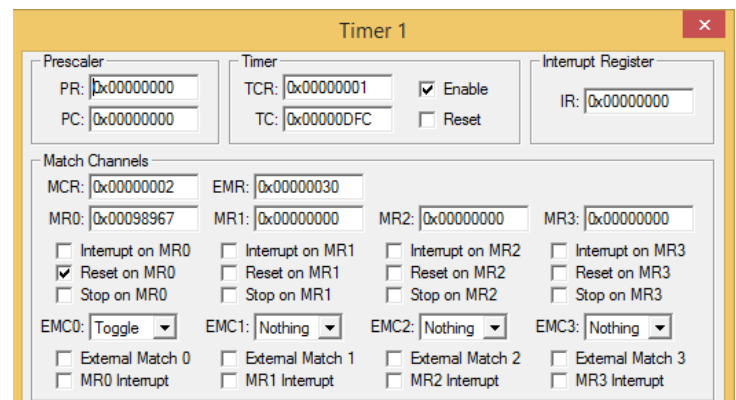
c) Complete la función de configuración del ADC y comente el código. (10 pts.)

```
void init_ADC(void)
{
    LPC_SC->PCONP |= (1<<12);           // Power ON
    LPC_PINCON->PINSEL |=
    LPC_PINCON->PINMODE |=
    LPC_ADC->ADCR = (0x    <<0) |
                  (4<<8) |
                  (1<<21) |           // PDN=1
                  (6<<24);

    LPC_ADC->ADINTEN=
    NVIC_SetPriority(ADC_IRQn, );
    NVIC_EnableIRQ(ADC_IRQn);
}
```

d) A la vista de la configuración del ADC y de la ventana de simulación de *Keil*, responda a las siguientes cuestiones. (5 pts.)

- Frecuencia de reloj del ADC.
- Frecuencia de muestreo de la entrada AD0.3.
- Tiempo de conversión.



e) Escriba la función de interrupción del ADC. (10 pts.)

```
void ADC_IRQHandler(void)
{
```

```
}
```

- f) Complete la parte del programa principal, que lee el estado del interruptor, modifica la PWM en función de la posición del potenciómetro, y envía por la UART0 la posición del servo en grados (ej. **Servo: 120**) **cada 0.5 seg. sólo si se produce un cambio en la posición del potenciómetro.**

NOTA: Considere configurado el Timer 2 (modo Match) para generar interrupciones periódicas cada 0.5 seg. y que existe la función de interrupción asociada que modifica un flag (medio_segundo=1) señalando dicho evento.

(10 ptos.)

```
while(1){  
    if(                )=  
    {  
  
    }  
    else  
    {  
  
    }  
}
```

Anexo 1. (Código fuente ejercicio 2)

```
#include    <LPC17xx.H>
#define     N_muestras      8

uint32_t   senal[] = {0,1,2,3,4,3,2,1};
int indice = 0;

void init_EINT0(void)
{
    LPC_PINCON->PINSEL4 |= (0x01<<20);    // P2.10 es entrada interrup. EXT 0
    LPC_SC->EXTMODE |= (1<<1) | (1<<0);    // Por flanco...
    LPC_SC->EXTPOLAR=0;                    // de bajada

    NVIC_SetPriority(EINT0_IRQn, 0);
    NVIC_EnableIRQ(EINT0_IRQn);
}

void init_TIMER0(void)
{
    LPC_SC->PCONP |= (1<<1);                // Power Timer 0
    LPC_SC->PCLKSEL0 |= (0x2<<2);           // PCLK = CCLK/2
    LPC_TIM0->MCR = 0x18;                   // Reset TC on Match_1, and Interrupt!
    LPC_TIM0->MR1 = 500000;
    LPC_TIM0->TCR = 0x01;                   // Enable TC

    NVIC_SetPriority(TIMER0_IRQn,1);
    NVIC_EnableIRQ(TIMER0_IRQn);
}

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);        // PCLK = CCLK/4
    LPC_DAC->DACCTRL=0;                    // Configuración del DAC
}

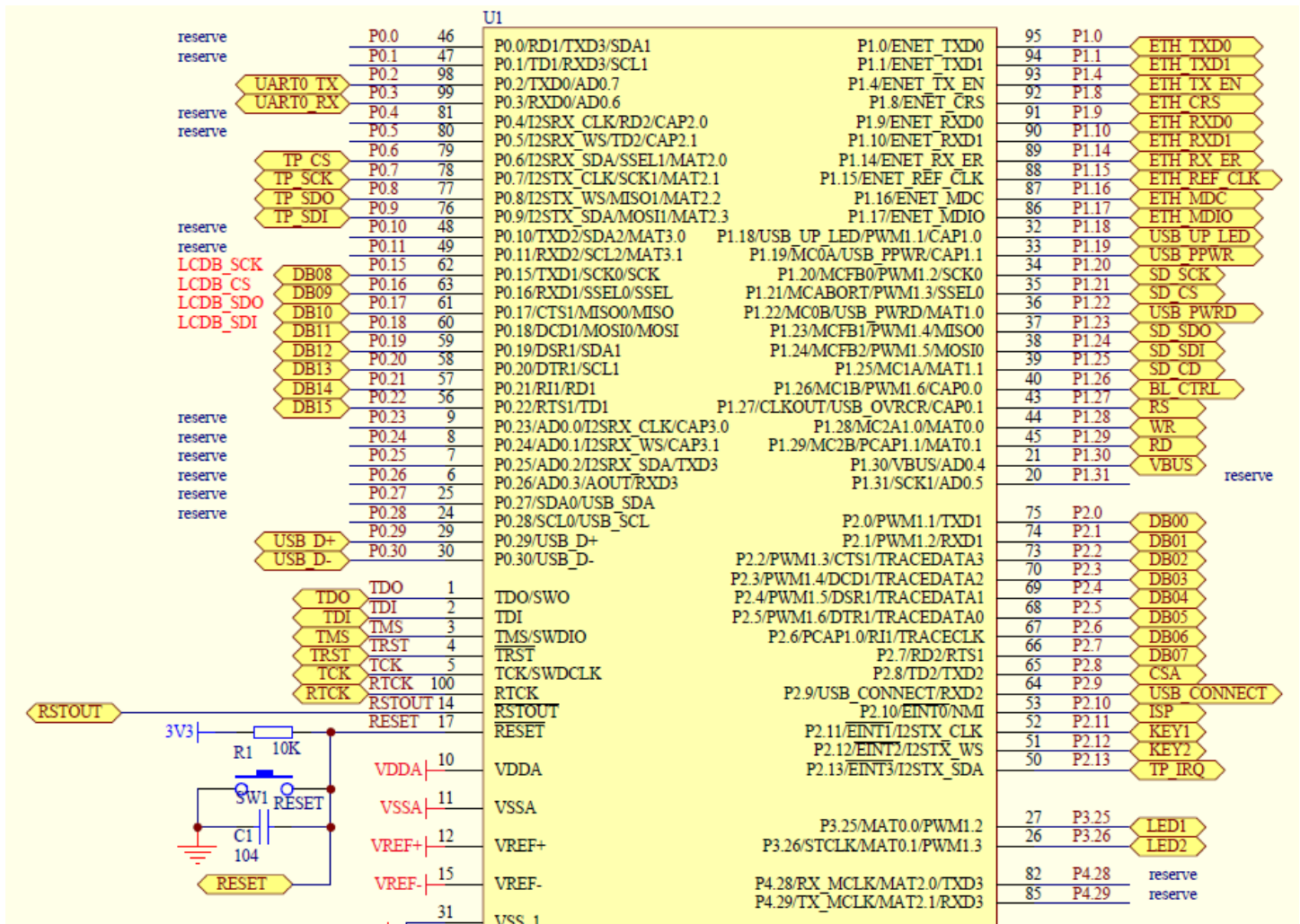
void EINT0_IRQHandler()
{
    LPC_SC->EXTINT=(1<<0);                  // Borrar flag Externa 0
    pinta_LCD();                           // Esta función tarda en ejecutarse 100ms
}

void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR |= (1<<1);                 // Borrar flag interrup.
    LPC_DAC->DACR = (senal[indice++] * 1023/4) << 6; // Escribir la muestra en el DAC (Bits 6...15)
    if(indice==N_muestras) indice =0;
}

int main(void)
{
    NVIC_SetPriorityGrouping(2);            // 32 niveles de prioridad preemptive (Sin sub-prioridad)
    init_EINT0();
    init_TIMER0();
    init_DAC();

    while(1){};
}
```

Anexo 2. (Esquema Mini-DK2)



Anexo 3 (Funciones UART)

```

/* uart.c
 * contiene las funciones:
 1 UART0_IRQHandler(void)
 2 tx_cadena_UART0(char *ptr)
 3 uart0_set_baudrate(unsigned int baudrate)
 4 uart0_init(int baudrate)
 */
#include <LPC17xx.h>
#include "uart.h"
/*
 * UART0 interrupt handler
 */
void UART0_IRQHandler(void) {

    switch(LPC_UART0->IIR&0x0E) {

        case 0x04:
            /* RBR, Receiver Buffer Ready */
            /* lee el dato recibido y lo almacena */
            /* Caracter return --> Cadena completa */
            *ptr_rx=LPC_UART0->RBR;
            if (*ptr_rx++ == '\r')
            {
                /* Añadimos el caracter null para tratar los datos recibidos como una cadena */
                /* 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;

    }
}

// Función para enviar una cadena de texto
// El argumento de entrada es la dirección de la cadena, o
// directamente la cadena de texto entre comillas
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
}

static int uart0_set_baudrate(unsigned int baudrate) {
    int errorStatus = -1; //< Failure

    // UART clock (FCCO / PCLK_UART0)
    // unsigned int uClk = SystemCoreClock / 4;
    unsigned int uClk =SystemCoreClock/4;
    unsigned int calcBaudrate = 0;
    unsigned int temp = 0;

    unsigned int mulFracDiv, dividerAddFracDiv;
    unsigned int divider = 0;
    unsigned int mulFracDivOptimal = 1;
    unsigned int dividerAddOptimal = 0;
    unsigned int dividerOptimal = 0;

    unsigned int relativeError = 0;
    unsigned int relativeOptimalError = 100000;

    uClk = uClk >> 4; /* div by 16 */

```

```

/*
 * The formula is :
 * BaudRate= uCk * (mulFracDiv/(mulFracDiv+dividerAddFracDiv) / (16 * DLL)
 *
 * The value of mulFracDiv and dividerAddFracDiv should comply to the following expressions:
 * 0 < mulFracDiv <= 15, 0 <= dividerAddFracDiv <= 15
 */
for (mulFracDiv = 1; mulFracDiv <= 15; mulFracDiv++) {
    for (dividerAddFracDiv = 0; dividerAddFracDiv <= 15; dividerAddFracDiv++) {
        temp = (mulFracDiv * uCk) / (mulFracDiv + dividerAddFracDiv);

        divider = temp / baudrate;
        if ((temp % baudrate) > (baudrate / 2))
            divider++;

        if (divider > 2 && divider < 65536) {
            calcBaudrate = temp / divider;

            if (calcBaudrate <= baudrate) {
                relativeError = baudrate - calcBaudrate;
            } else {
                relativeError = calcBaudrate - baudrate;
            }

            if (relativeError < relativeOptimalError) {
                mulFracDivOptimal = mulFracDiv;
                dividerAddOptimal = dividerAddFracDiv;
                dividerOptimal = divider;
                relativeOptimalError = relativeError;
                if (relativeError == 0)
                    break;
            }
        }
    }
}

if (relativeError == 0)
    break;
}

if (relativeOptimalError < ((baudrate * UART_ACCEPTED_BAUDRATE_ERROR) / 100)) {
    LPC_UART0->LCR |= DLAB_ENABLE; // importante poner a 1
    LPC_UART0->DLM = (unsigned char) ((dividerOptimal >> 8) & 0xFF);
    LPC_UART0->DLL = (unsigned char) dividerOptimal;
    LPC_UART0->LCR &= ~DLAB_ENABLE; // importante poner a 0

    LPC_UART0->FDR = ((mulFracDivOptimal << 4) & 0xF0) | (dividerAddOptimal & 0x0F);

    errorStatus = 0; //< Success
}

return errorStatus;
}

void uart0_init(int baudrate) {
    LPC_PINCON->PINSEL0|=(1<<4)|(1<<6); // Change P0.2 and P0.3 mode to TXD0 and RXD0

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

    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)
}

```