Documentacion de funciones de librerias de CMSIS utilizadas para EDIII

GPIO: para manipular puertos utilizaremos las funciones de la libreria

```
#include "lpc17xx_gpio.h"
```

```
// Define si el pin va a ser entrada o salida.
// Si dir = 0, el pin sera entrada.
// Si dir = 1 el pin sera salida.
// En pin podemos cargar un valor en hexa para elegir mas de uno.
void GPIO_SetDir (uint8_t portNum, uint32_t pin, uint8_t dir)

// Pone en 1 el pin elegido (Siempre y cuando haya sido seteado como salida).
// Tambien podemos cargar un valor en hexa para trabajar con mas de un pin
void GPIO_SetValue (uint8_t portNum, uint32_t pin)

// Pone en 0 el pin elegido (Siempre y cuando haya sido seteado como salida).
// Tambien podemos cargar un valor en hexa para trabajar con mas de un pin
void GPIO_ClearValue (uint8_t portNum, uint32_t pin)

// Nos devuelve el estado de cada pin del puerto
uint32_t GPIO_ReadValue (uint8_t portNum)

// Habilita interrupciones para P0[0-30] y P2[0-13].
// portNum puede ser 0 o 2 y pin es el pin que queremos usar para la interrupiones
// edgeSate = 0 para interrupcion por flanco de subida
// edgeSate = 1 para interrupcion por flanco de bajada
void GPIO_IntCmd (uint8_t portNum, uint32_t pin, uint8_t edgeState)

// Nos devuelve ENABLE si hubo una interrupcion en el pin o DISABLE en caso contario
// Valido para P0[0-30] y P2[0-13]
// edgeSate = 0 para interrupcion por flanco de subida
// edgeSate = 1 para interrupcion por flanco de bajada
FunctionalState GPIO_GetIntStatus (uint8_t portNum, uint32_t pinD, uint8_t edgeState)

// Limpia la bandera de interrupcion del pin
// Valido para P0[0-30] y P2[0-13]
void GPIO_ClearInt (uint8_t portNum, uint32_t pin)
```

PinSel: para elegir la funcion de los pines utilizaremos la libreria

```
#include "lpc17xx_pinsel.h"
```

```
// Creamos estructura
PINSEL_CFG_Type pin_configuration;

pin_configuration.Portnum = PINSEL_PORT_X; // [0-4]
pin_configuration.Pinnum = PINSEL_PIN_ X; // [0-31]
pin_configuration.Funcnum = PINSEL_FUNC_X;; // [0-3]
pin_configuration.Pinmode = PINSEL_PINMODE_X; // [PULLUP-TRISTATE]
pin_configuration.OpenDrain = PINSEL_PINMODE_X; // [NORMAL-OPENDRAIN]

PINSEL_ConfigPin (&pin_configuration); // Direccion de la estructura como parametro
```

SysTick: utilizaremos las funciones incuidas en la libreria

```
#include "lpc17xx_systick.h"
```

```
// Configuracion de System Tick con clock interno
// time debe ser expresada en ms
void SYSTICK_InternalInit (uint32_t time)

// Habilita o deshabilita las interrupciones por SysTick
// state debe ser ENABLE o DISABLE
void SYSTICK_IntCmd (FuncionalState state)

// Habilita o deshabilita el SysTick
// state debe ser ENABLE o DISABLE
void SYSTICK_Cmd (FuncionalState state)

// Nos devuelve el valor del contador de SysTick
uint32_t SYSTICK_GetCurrentValue (void)

// Limpia la bandera del contador
void SYSTICK_ClearCounterFlag(void)
```

Timer: utilizaremos las funciones incluidas en la libreria

```
#include "lpc17xx_timer.h"
```

```
// Creamos la estructura
TIM_TIMERCFG_Type timer_0_configuration;

timer_0_configuration.PrescaleOption = TIM_PRESCALE_X; // [TICKVAL-USVAL] Usamos USVAL para us
timer_0_configuration.PrescaleValue = uint32_t; // Valor del prescaler

TIM_Init (LPC_TIM0, TIM_TIMER_MODE, &timer_0_configuration);

// Creamos la estructura
TIM_MATCHCFG_Type channel_configuration;

channel_configuration.MatchChannel = X; // [0-3]
channel_configuration.IntOnMatch = X; // [ENABLE-DISABLE]
channel_configuration.StopOnMatch = X; // [ENABLE-DISABLE]
channel_configuration.ResetOnMatch = X; // [ENABLE-DISABLE]
channel_configuration.ExtMatchOutputType = TIM_EXTMATCH_X; // [NOTHING-LOW-HIGH-TOGGLE]
channel_configuration.MatchValue = uint32_t; // #define SECOND (uint32_t) 10000

TIM_ConfigMatch (LPC_TIM0, &channel_configuration); // Direccion de la estructura como parametro

// Enciendo Timer0
Tim_CMD (LPC_TIM0, X); // [ENABLE-DISABLE]
```

ADC: utilizaremos las funciones incluidas en la libreria

```
#include "lpc17xx_adc.h"
```

```
// Configurar el clock del ADC; rate_conversion debe ser <=200KHz
void ADC_Init (LPC_ADC, uint32_t rate_conversion)

// Habilita o deshabilita cualquiera de los 8 canales que posee la LPC,
// por lo que la variable channel puede obtener valores de 0 a 7,
// mientras que state puede ser ENABLE o DISABLE
void ADC_ChannelCmd (LPC_ADC, uint8_t channel, state)

// Habilita o deshabilita las interrupciones en cualquiera de los 8 canales que posee la LPC,
// por lo que la variable channel puede obtener valores de 0 a 7,
// mientras que state puede ser ENABLE o DISABLE
void ADC_IntConfig (LPC_ADC, uint8_t channel, state)

// Nos devuelve SET o RESET dependiendo de si el ADC termino de convertir o no.
// Recordamos que el ADC posee 2 tipos de estados, Burst (0) o Done (1).
// El tipo de estado Done es el que usaremos. En los parametros de la funcion,
// channel corresponde al canal [0-7] mientras que status al tipo de estado por
// el que queremos consultar. Veamos un ejemplo
FlagStatus ADC_ChannelGetStatus (LPC_ADC, uint8_t channel, uint32_t status)

// Aqui estariamos estariamos realizando algo siempre y cuando el canal [0]
// NO haya terminado de convertir (por eso se iguala a RESET).
// Cuando termine de convertir, devolvera un SET y se procedera a realizar otra cosa
while (ADC_ChannelGetStatus (LPC_ADC, 0, 1) == RESET)

// Nos devuelve el resultado de la conversion  del registro ADDR0.
// En el parametro channel ponemos el canal del que queremos leer los datos [7-0]
uint32_t ADC_GetData (uint32_t channel)

// Seteamos el modo para operar con el ADC.
// Podemos utilizar esta funcion por ejemplo luego de terminar de convertir,
// para que el ADC comience otra vez a tomar datos. En start_mode podemos elegir entre:
// ADC_START_CONTINUOUS, ADC_START_NOW, ADC_START_ON_EINT0, etc. Generalmente usamos ADC_START_NOW.
void ADC_StartCmd (LPC_ADC, uint8_t start_mode)

// Close ADC
void ADC_DeInit (LPC_ADC)
```
