```c
/**
 * @file
 * @brief Funciones de configuracion basadas en macros establecidos en
 * config.h
 */

#include "config.h"
#include "LPC17xx.h"
#include "lpc17xx_adc.h"
#include "lpc17xx_gpdma.h"
#include "lpc17xx_gpio.h"
#include "lpc17xx_pinsel.h"
#include "lpc17xx_timer.h"
#include "lpc17xx_uart.h"
#include "lpc_types.h"
#include <stdint.h>

//Estructuras del DMA globales
static GPDMA_LLI_Type lliUART = {0};
static GPDMA_Channel_CFG_Type dmaUART = {0};

/**
 * @brief Configura pines
 */
void configPCB(void) {
  PINSEL_CFG_Type cfgPinLedG = {0}; // Seria el buzzer
  PINSEL_CFG_Type cfgPinLed = {0};
  PINSEL_CFG_Type cfgADC = {0};
  PINSEL_CFG_Type cfg_TX_UART = {0};
  PINSEL_CFG_Type cfg_RX_UART = {0};

  cfgPinLedG.portNum = PORT_LED_GREEN;
  cfgPinLedG.pinNum = PIN_LED_GREEN;
  cfgPinLedG.funcNum = FUNC_GPIO;

  cfgPinLed.portNum = PORT_LED_RED;
  cfgPinLed.pinNum = PIN_LED_RED;
  cfgPinLed.funcNum = FUNC_GPIO;

  cfgADC.portNum = PORT_ADC;
  cfgADC.pinNum = PIN_ADC;
  cfgADC.funcNum = FUNC_ADC;
  cfgADC.pinMode = PINSEL_TRISTATE;

  cfg_TX_UART.portNum = PORT_TX_UART2;
  cfg_TX_UART.pinNum = PIN_TX_UART2;
  cfg_TX_UART.funcNum = FUNC_TX_UART2;

  cfg_RX_UART.portNum = PORT_RX_UART2;
  cfg_RX_UART.pinNum = PIN_RX_UART2;
  cfg_RX_UART.funcNum = FUNC_RX_UART2;
  cfg_RX_UART.pinMode = PINSEL_TRISTATE;

  PINSEL_ConfigPin(&cfgPinLedG);
  PINSEL_ConfigPin(&cfgPinLed);
  PINSEL_ConfigPin(&cfgADC);
  PINSEL_ConfigPin(&cfg_TX_UART);
  PINSEL_ConfigPin(&cfg_RX_UART);
}

/**
 * @brief Configura los GPIO y los limpia.
```

```
 63 */
 64 void configGPIO(void) {
 65
 66   GPIO_SetDir(PORT_LED_RED, BIT_VALUE(PIN_LED_RED), OUTPUT);
 67   GPIO_SetDir(PORT_LED_GREEN, BIT_VALUE(PIN_LED_GREEN), OUTPUT);
 68
 69   GPIO_SetPins(PORT_LED_RED, BIT_VALUE(PIN_LED_RED));
 70   GPIO_SetPins(PORT_LED_GREEN, BIT_VALUE(PIN_LED_GREEN));
 71 }
 72
 73 /**
 74  * @brief Configura el timer destinado a interrumpir cada 1 minuto para
 75  * calcular la cantidad de ppm (pulsos por minuto)
 76  */
 77 void configTimerPPM(void) {
 78
 79   TIM_TIMERCFG_Type tim_min = {0};
 80   TIM_MATCHCFG_Type match_min = {0};
 81
 82   tim_min.prescaleOption = TIM_USVAL;
 83   tim_min.prescaleValue = TIMER_PS_1MS;
 84
 85   match_min.matchChannel = TIMER_CHANNEL_0;
 86   match_min.matchValue = TIMER_60S; // 60000 ms = 60 s
 87   match_min.intOnMatch = ENABLE;
 88   match_min.resetOnMatch = ENABLE;
 89   match_min.stopOnMatch = DISABLE;
 90
 91   TIM_Init(LPC_TIM2, TIM_TIMER_MODE, &tim_min);
 92   TIM_ConfigMatch(LPC_TIM2, &match_min);
 93 }
 94
 95 /**
 96  * @brief Configura el timer del ADC para realizar el muestreo a 1 kHz
 97  *
 98  */
 99 void configTimerADC(void) {
100
101   TIM_TIMERCFG_Type tim_adc = {0};
102   TIM_MATCHCFG_Type mat_adc = {0};
103
104   tim_adc.prescaleOption = TIM_TICKVAL;
105   tim_adc.prescaleValue = 99;
106
107   mat_adc.matchChannel = 1;
108   mat_adc.intOnMatch = DISABLE;
109   mat_adc.resetOnMatch = ENABLE;
110   mat_adc.stopOnMatch = DISABLE;
111   mat_adc.extMatchOutputType = TIM_TOGGLE;
112   mat_adc.matchValue = 124;
113
114   TIM_Init(LPC_TIM0, TIM_TIMER_MODE, &tim_adc);
115   TIM_ConfigMatch(LPC_TIM0, &mat_adc);
116
117   TIM_Cmd(LPC_TIM0, ENABLE);
118 }
119 /**
120  * @brief Configura el modulo ADC por trigger de MAT0 TMR1
121  */
122 void configADC(void) {
123
124   ADC_Init(2000);
```

```
125   ADC_BurstCmd(DISABLE);
126   ADC_StartCmd(ADC_START_ON_MAT01);
127   ADC_ChannelCmd(ADC_CHANNEL_0, ENABLE);
128   ADC_EdgeStartConfig(ADC_START_ON_RISING);
129   ADC_IntConfig(ADC_CHANNEL_0, ENABLE);
130 }
131
132 /**
133  * @brief Configura el modulo UART para funcionar con el modulo de GPDMA
134  */
135 void configUART(void) {
136
137     UART_CFG_Type      cfgUART2;
138     UART_FIFO_CFG_Type cfgFIFO;
139
140     UART_ConfigStructInit(&cfgUART2);
141     cfgUART2.Baud_rate = BAUD_RATE;
142     cfgUART2.Parity    = UART_PARITY_NONE;
143     cfgUART2.Databits  = UART_DATABIT_8;
144     cfgUART2.Stopbits  = UART_STOPBIT_1;
145
146     UART_Init(LPC_UART2, &cfgUART2);
147
148     UART_FIFOConfigStructInit(&cfgFIFO);
149     cfgFIFO.FIFO_DMAMode    = ENABLE;
150     cfgFIFO.FIFO_Level      = UART_FIFO_TRGLEV2;
151     cfgFIFO.FIFO_ResetRxBuf = ENABLE;
152     cfgFIFO.FIFO_ResetTxBuf = ENABLE;
153     UART_FIFOConfig(LPC_UART2, &cfgFIFO);
154
155     UART_TxCmd(LPC_UART2, ENABLE);
156 }
157 /**
158  * @brief Configura el modulo de GPDMA par funcionar con el modulo UART
159  *
160  */
161 void configGPDMA_UART(volatile uint8_t *txBuffer) {
162
163   GPDMA_Init();
164
165   lliUART.nextLLI = 0;
166
167   dmaUART.channelNum = GPDMA_CHANNEL_UART;
168   dmaUART.srcMemAddr = (uint32_t)(uintptr_t)txBuffer;
169   dmaUART.dstConn = GPDMA_UART2_Tx;
170   dmaUART.transferType = GPDMA_M2P;
171   dmaUART.transferSize = (uint32_t)TX_BUFFER_SIZE;
172   dmaUART.linkedList = (uint32_t)(uintptr_t)&lliUART;
173
174   NVIC_EnableIRQ(DMA_IRQn);
175 }
176
177 void startUART_DMA(uint8_t *buffer, volatile uint8_t *dmaUartBusy) {
178
179
180   lliUART.srcAddr = (uint32_t)(uintptr_t)buffer;
181   lliUART.dstAddr = (uint32_t)(uintptr_t)&LPC_UART2->THR;
182   lliUART.nextLLI = 0;
183
184   lliUART.control = (TX_BUFFER_SIZE << 0) |
185                     (GPDMA_BSIZE_1 << 12) |
186                     (GPDMA_BSIZE_1 << 15) |
```

```
187                     (GPDMA_BYTE << 18) |
188                     (GPDMA_BYTE << 21) |
189                     (1 << 26) |
190                     (0 << 27) |
191                     (1 << 31);
192
193   dmaUART.srcMemAddr = (uint32_t)(uintptr_t)buffer;
194   dmaUART.transferSize = (uint32_t)TX_BUFFER_SIZE;
195   dmaUART.linkedList = (uint32_t)(uintptr_t)&lliUART;
196
197   GPDMA_Setup(&dmaUART);
198   GPDMA_ChannelCmd(GPDMA_CHANNEL_UART, ENABLE);
199
200   *dmaUartBusy = 1;
201 }
202
```