```c
#include "arm_math.h"
#include "config.h"
#include "lpc17xx_adc.h"
#include "lpc17xx_gpdma.h"
#include "lpc17xx_gpio.h"
#include "lpc17xx_timer.h"
#include "lpc17xx_uart.h"
#include <stdint.h>

// Filtro IIR Notch 50Hz Direct Form I Bicascada
arm_biquad_casd_df1_inst_f32 iir_instance;
static float32_t pState_iir[4 * NUM_STAGES];
const float32_t b0 = B0;
const float32_t b1 = B1;
const float32_t b2 = B2;
const float32_t a1 = A1;
const float32_t a2 = A2;

// Arreglo de coeficientes
float32_t iir_coeffs[5] = {b0, b1, b2, -a1, -a2};

// Buffers auxiliares
volatile float32_t fifoInputIIR[TX_BUFFER_SIZE];
volatile float32_t fifoOutputIIR[TX_BUFFER_SIZE];

// Buffer para DMA UART
volatile uint8_t txBufferA[TX_BUFFER_SIZE];
volatile uint8_t txBufferB[TX_BUFFER_SIZE];
// puntero al buffer que se está llenando
volatile uint8_t *txFillBuffer = txBufferA;
// índice dentro del buffer que se está llenando
volatile uint16_t fillIndex = 0;
// flag: DMA ocupada transmitiendo algún buffer
volatile uint8_t dmaUartBusy = 0;

// ADC
volatile uint16_t adcIndex = 0;

// Variables para conteo de pulsos acumulativo
volatile uint16_t pulsos_acumulados = 0; // Contador acumulativo durante 60s
volatile uint8_t ultimo_valor = 0;       // Para detectar flancos
volatile uint8_t r_flag = 0;             // Flag de detección de pico

// Variable de ppm
volatile uint8_t ppm = 0;

int main() {
  SystemInit();

  arm_biquad_cascade_df1_init_f32(&iir_instance, NUM_STAGES, iir_coeffs,
                                  pState_iir); // Inicializa filtro

  // Configura los módulos y periféricos
  configPCB();
  configGPIO();
  configUART();
  configADC();
  configTimerPPM();
  configTimerADC();
  configGPDMA_UART(txBufferA);

  // Habilita interrupciones
```

```
 63   NVIC_EnableIRQ(ADC_IRQn);
 64   NVIC_EnableIRQ(TIMER2_IRQn);
 65
 66   TIM_Cmd(LPC_TIM2, ENABLE);
 67
 68   while (1) {
 69   }
 70
 71   return 0;
 72 }
 73
 74 void ADC_IRQHandler(void) {
 75
 76   if (ADC_ChannelGetStatus(ADC_CHANNEL_0, ADC_DATA_DONE)) {
 77
 78     uint16_t adcRawData = ADC_ChannelGetData(ADC_CHANNEL_0);
 79
 80     // Procesamiento de la señal mediante filtro IIR
 81     fifoInputIIR[adcIndex] = (float32_t)adcRawData;
 82
 83     arm_biquad_cascade_df1_f32(&iir_instance, &fifoInputIIR[0] + adcIndex,
 84                                &fifoOutputIIR[0] + adcIndex, 1);
 85
 86     // Escalado a 0–255
 87     float32_t scaled = fifoOutputIIR[adcIndex] * 255.0f / 4095.0f;
 88     if (scaled < 0.0f)
 89       scaled = 0.0f;
 90     else if (scaled > 255.0f)
 91       scaled = 255.0f;
 92
 93     // Redondeo
 94     uint8_t data_u8 = (uint8_t)(scaled + 0.5f);
 95
 96     // --- Conteo de pulsos en tiempo real ---
 97     // Detectar flanco de subida
 98     if (data_u8 >= VAL_UMBRAL && !r_flag) {
 99       GPIO_ClearPins(PORT_LED_GREEN, BIT_VALUE(PIN_LED_GREEN));
100       pulsos_acumulados++;
101       r_flag = 1;
102     }
103
104     // Detectar flanco de bajada para resetear detección
105     if (data_u8 < VAL_UMBRAL && r_flag) {
106       GPIO_SetPins(PORT_LED_GREEN, BIT_VALUE(PIN_LED_GREEN));
107       r_flag = 0;
108     }
109
110     adcIndex++;
111     if (adcIndex >= TX_BUFFER_SIZE) {
112       adcIndex = 0;
113     }
114
115     // --- Ping Pong UART ---
116     if (fillIndex < TX_BUFFER_SIZE) {
117       txFillBuffer[fillIndex] = data_u8;
118       fillIndex++;
119     } else {
120       if (!dmaUartBusy) {
121         // Buffer lleno y DMA libre
122         uint8_t *txDMABuffer = (uint8_t *)txFillBuffer;
123         startUART_DMA(txDMABuffer, &dmaUartBusy);
124
```

```
125         // Cambiar buffer de llenado
126         txFillBuffer = (txFillBuffer == txBufferA) ? txBufferB : txBufferA;
127         fillIndex = 0;
128       } else {
129         // Buffer lleno y DMA ocupada
130         fillIndex = TX_BUFFER_SIZE;
131       }
132     }
133   }
134 }
135
136 void DMA_IRQHandler(void) {
137
138   if (GPDMA_IntGetStatus(GPDMA_INT, GPDMA_CHANNEL_UART)) {
139
140     if (GPDMA_IntGetStatus(GPDMA_INTTC, GPDMA_CHANNEL_UART)) {
141       GPDMA_ClearIntPending(GPDMA_CLR_INTTC, GPDMA_CHANNEL_UART);
142       dmaUartBusy = 0; // Reinicia transferencia
143     }
144
145     if (GPDMA_IntGetStatus(GPDMA_INTERR, GPDMA_CHANNEL_UART)) {
146       GPDMA_ClearIntPending(GPDMA_CLR_INTERR, GPDMA_CHANNEL_UART);
147       dmaUartBusy = 0; // Reinicia transferencia
148     }
149   }
150 }
151
152 /**
153  * @brief Handler cada 60 segundos para calcular ppm y controlar LED RED
154  */
155 void TIMER2_IRQHandler(void) {
156
157   if (TIM_GetIntStatus(LPC_TIM2, TIM_MR0_INT)) {
158
159     uint16_t pulsos = pulsos_acumulados;
160     pulsos_acumulados = 0;
161
162     // Limita ppm a 255
163     ppm = (pulsos > 255) ? 255 : (uint8_t)pulsos;
164
165     // Verifica si el PPM está fuera del rango normal
166     if (ppm > PPM_UMBRAL_MAX || ppm < PPM_UMBRAL_MIN) {
167       GPIO_ClearPins(PORT_LED_RED, BIT_VALUE(PIN_LED_RED));
168     } else {
169       GPIO_SetPins(PORT_LED_RED, BIT_VALUE(PIN_LED_RED));
170     }
171
172     TIM_ClearIntPending(LPC_TIM2, TIM_MR0_INT);
173   }
174 }
175
```