#include "stm32f4xx.h"

// Define pins

#define TRIG\_PIN (1 << 8) // PA8 - Trigger Pin

#define ECHO\_PIN (1 << 9) // PA9 - Echo Pin

#define LED\_PIN (1 << 5) // PA5 (PWM for LED)

#define DISTANCE\_THRESHOLD 5 // Distance threshold in cm

#define LDR\_CHANNEL 0 // ADC Channel 0 (LDR connected to PA0)

#define TIMEOUT\_LIMIT 10000 // Timeout limit for echo signal in cycles

// Function prototypes

void GPIO\_Init(void);

void Timer\_Init(void);

void ADC\_Init(void);

uint16\_t read\_LDR\_value(void);

float measure\_distance(void);

void delay\_us(uint32\_t delay);

int main(void) {

GPIO\_Init();

Timer\_Init();

ADC\_Init();

while (1) {

// Measure the distance

float distance = measure\_distance();

// Check if the distance is below the threshold

if (distance > 0 && distance <= DISTANCE\_THRESHOLD) {

// Read LDR value

uint16\_t ldrValue = read\_LDR\_value();

if (ldrValue <= 300){

// Map the LDR value (0-4095) to a PWM duty cycle (0-100%)

uint16\_t pwmValue = 100 - (ldrValue \* 100) / 4095;

// Set PWM duty cycle

TIM2->CCR1 = (pwmValue \* TIM2->ARR) / 100; // Scale to timer period

}

} else {

// Distance exceeds the threshold, turn off PWM and LED

TIM2->CCR1 = 0; // Set PWM duty cycle to 0

}

}

}

void GPIO\_Init(void) {

// Enable GPIOA clock

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN;

// Configure PA8 (TRIG\_PIN) as output

GPIOA->MODER |= (1 << 16); // Set PA8 as output

GPIOA->OTYPER &= ~TRIG\_PIN; // Set PA8 as push-pull

GPIOA->OSPEEDR |= (3 << 16); // Set PA8 as high speed

// Configure PA9 (ECHO\_PIN) as input

GPIOA->MODER &= ~(3 << 18); // Set PA9 as input

// Configure PA5 (LED\_PIN) as alternate function (PWM output)

GPIOA->MODER |= GPIO\_MODER\_MODER5\_1; // Alternate function mode

GPIOA->AFR[0] |= (1 << 20); // AF1 for TIM2\_CH1

// Configure PA0 (LDR input) as analog input

GPIOA->MODER |= GPIO\_MODER\_MODER0; // Analog mode

}

void Timer\_Init(void) {

// Enable TIM2 clock

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN;

// Configure TIM2 for PWM

TIM2->PSC = 84 - 1; // Prescaler: 1 MHz (1 µs per tick)

TIM2->ARR = 1000 - 1; // Auto-reload: 1000 steps (1 kHz PWM frequency)

TIM2->CCMR1 |= (6 << 4); // PWM mode 1 on channel 1

TIM2->CCER |= TIM\_CCER\_CC1E; // Enable channel 1 output

TIM2->CR1 |= TIM\_CR1\_CEN; // Enable the timer

}

void ADC\_Init(void) {

// Enable ADC1 clock

RCC->APB2ENR |= RCC\_APB2ENR\_ADC1EN;

// Configure ADC1 for single-channel continuous conversion

ADC1->SQR3 = LDR\_CHANNEL; // Select channel 0 (PA0)

ADC1->CR2 |= ADC\_CR2\_ADON; // Enable ADC1

ADC1->CR2 |= ADC\_CR2\_CONT; // Continuous conversion mode

ADC1->CR2 |= ADC\_CR2\_SWSTART; // Start conversion

}

uint16\_t read\_LDR\_value(void) {

while (!(ADC1->SR & ADC\_SR\_EOC)); // Wait for conversion to complete

return ADC1->DR; // Return ADC value

}

// Delay function in microseconds

void delay\_us(uint32\_t delay) {

TIM2->CNT = 0; // Reset the timer count

while (TIM2->CNT < delay); // Wait until the timer reaches the delay

}

// Function to measure distance using the ultrasonic sensor

float measure\_distance(void) {

uint32\_t start\_time, end\_time;

// Send a 10 µs pulse on the trigger pin

GPIOA->ODR &= ~TRIG\_PIN; // Trigger low

delay\_us(2);

GPIOA->ODR |= TRIG\_PIN; // Trigger high

delay\_us(10);

GPIOA->ODR &= ~TRIG\_PIN; // Trigger low

// Wait for echo pin to go high (start of the pulse)

uint32\_t timeout = TIMEOUT\_LIMIT;

while (!(GPIOA->IDR & ECHO\_PIN) && timeout--) {

// Timeout protection

}

if (timeout == 0) return -1; // Timeout, return error code

start\_time = TIM2->CNT; // Record the start time

// Wait for echo pin to go low (end of the pulse)

timeout = TIMEOUT\_LIMIT;

while (GPIOA->IDR & ECHO\_PIN && timeout--) {

// Timeout protection

}

if (timeout == 0) return -1; // Timeout, return error code

end\_time = TIM2->CNT; // Record the end time

// Calculate the duration and convert to distance

uint32\_t duration = end\_time - start\_time;

if (duration == 0) return -1; // Avoid divide by zero

// Calculate the distance based on the duration

float distance = (duration \* 0.034) / 2.0; // Speed of sound = 0.034 cm/µs

return distance;

}