**DYNAMIC APPALANATION TONOMETRY**

#include "stm32f10x.h"

#include <stdio.h>

// Constants

#define APPLANATION\_THRESHOLD 1000 // Threshold for applanation detection

#define P\_INITIAL 10 // Initial air pressure (e.g., 10 mmHg)

#define P\_STEP 1 // Pressure increment step

#define I\_BASELINE 500 // Baseline intensity level

#define C1 0.5 // Calibration constant for pressure

#define C2 0.3 // Calibration constant for applanation time

#define C3 0.2 // Calibration constant for recovery time

// Global variables

uint16\_t airPressure = 0; // Air pressure sensor value

uint16\_t lightIntensity = 0; // Light reflection sensor value

uint32\_t applanationTime = 0; // Time taken for applanation (ms)

uint32\_t recoveryTime = 0; // Time taken for recovery (ms)

// Function prototypes

void init\_ADC(void);

void init\_TIM2(void);

void start\_air\_puff(void);

void stop\_air\_puff(void);

uint16\_t read\_ADC(uint8\_t channel);

uint32\_t measure\_time(void);

float calculate\_IOP(uint16\_t P\_ap, uint32\_t T\_ap, uint32\_t T\_rec);

int main(void)

{

// Initialize peripherals

init\_ADC();

init\_TIM2();

uint16\_t P\_ap = 0;

uint32\_t T\_ap = 0;

uint32\_t T\_rec = 0;

float IOP = 0.0;

// Start air puff and increase pressure gradually

start\_air\_puff();

for (airPressure = P\_INITIAL; airPressure < 60; airPressure += P\_STEP)

{

// Read light reflection intensity

lightIntensity = read\_ADC(2);

// Detect applanation

if (lightIntensity >= APPLANATION\_THRESHOLD)

{

P\_ap = airPressure; // Record applanation pressure

T\_ap = measure\_time(); // Record applanation time

stop\_air\_puff(); // Stop air puff after applanation detected

break;

}

}

// Measure recovery time

while (1)

{

lightIntensity = read\_ADC(2);

if (lightIntensity <= I\_BASELINE)

{

T\_rec = measure\_time(); // Record recovery time

break;

}

}

// Calculate Intraocular Pressure (IOP)

IOP = calculate\_IOP(P\_ap, T\_ap, T\_rec);

// Print or display the IOP result

printf("Intraocular Pressure (IOP): %.2f mmHg\n", IOP);

while (1);

}

// ADC initialization

void init\_ADC(void)

{

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

ADC\_InitTypeDef ADC\_InitStructure;

ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE;

ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE;

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;

ADC\_InitStructure.ADC\_NbrOfChannel = 1;

ADC\_Init(ADC1, &ADC\_InitStructure);

ADC\_Cmd(ADC1, ENABLE);

}

// Timer initialization for measuring time (e.g., TIM2)

void init\_TIM2(void)

{

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2, ENABLE);

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_TimeBaseStructure.TIM\_Period = 0xFFFF;

TIM\_TimeBaseStructure.TIM\_Prescaler = 72 - 1; // 1 MHz clock (assuming 72 MHz system clock)

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0;

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM2, &TIM\_TimeBaseStructure);

TIM\_Cmd(TIM2, ENABLE);

}

// Function to start air puff using PWM control

void start\_air\_puff(void)

{

// Assuming PWM control of air puff via GPIO

// GPIO\_SetBits(GPIOA, GPIO\_Pin\_X); (Where X is connected to puff actuator)

}

// Function to stop air puff

void stop\_air\_puff(void)

{

// GPIO\_ResetBits(GPIOA, GPIO\_Pin\_X); (Stop air puff)

}

// Read ADC value from a specific channel

uint16\_t read\_ADC(uint8\_t channel)

{

ADC\_RegularChannelConfig(ADC1, channel, 1, ADC\_SampleTime\_55Cycles5);

ADC\_SoftwareStartConvCmd(ADC1, ENABLE);

while (ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_EOC) == RESET);

return ADC\_GetConversionValue(ADC1);

}

// Measure time using TIM2 (reset before measuring)

uint32\_t measure\_time(void)

{

return TIM\_GetCounter(TIM2);

}

// Calculate IOP based on applanation pressure, time, and recovery time

float calculate\_IOP(uint16\_t P\_ap, uint32\_t T\_ap, uint32\_t T\_rec)

{

float IOP\_base = C1 \* P\_ap + C2 \* T\_ap + C3 \* T\_rec;

return IOP\_base;

}