```c
#include "stm32f4xx.h"
#include <stdlib.h>

/* ================= CONSTANTS ================= */
#define SLAVE_ADDR       0x27

// TIME SETTINGS
#define TIMEOUT_LIMIT    10000 // 10 seconds
#define MIN_DELAY        3000  // 3 seconds
#define MAX_DELAY        7000  // 7 seconds

// Hardware Pins
#define GREEN_LED_PIN    (1U << 0)   // PA0
#define RED_LED_PIN      (1U << 1)   // PA1
#define PATIENT_BTN_PIN  (1U << 0)   // PB0
#define RESET_BTN_PIN    (1U << 1)   // PB1

/* ================= GLOBALS ================= */
volatile uint32_t reaction_time_ms = 0;
volatile uint8_t button_pressed_flag = 0;
volatile uint8_t test_in_progress = 0;
volatile uint8_t false_start_flag = 0;

/* ================= PROTOTYPES ================= */
void SystemClock_Config(void);
void GPIO_Init(void);
void TIM2_Init(void);
void EXTI_Init(void);
void I2C1_Init(void);

void LCD_Init(void);
void LCD_SendCmd(uint8_t cmd);
void LCD_SendData(uint8_t data);
void LCD_SendString(char *str);
void LCD_SetCursor(uint8_t row, uint8_t col);
void LCD_Clear(void);
void LCD_Print_Number(uint32_t num);
void LCD_Write_Nibble(uint8_t nibble, uint8_t rs);
void LCD_SendByte(uint8_t byte, uint8_t rs);

void I2C_Start(void);
void I2C_Stop(void);
void I2C_Write(uint8_t data);

void Delay_ms(volatile int ms);
void LED_Control(uint8_t green_on, uint8_t red_on);
uint32_t GenerateRandomDelay(void);

void DisplayWelcomeScreen(void);
void DisplayResults(void);
void RunReactionTest(void);
void DisplayError(void);

/* ================= MAIN ================= */
int main(void) {
    SystemClock_Config();
    TIM2_Init();
    GPIO_Init();
    I2C1_Init();
    EXTI_Init();

    // Alive Check
    LED_Control(1, 1);
    Delay_ms(200);
    LED_Control(0, 0);

    LCD_Init();
    srand(TIM2->CNT);

    DisplayWelcomeScreen();

    while (1) {
```

```
73          if (!(GPIOB->IDR & RESET_BTN_PIN)) {
74              Delay_ms(50); // Debounce
75              if (!(GPIOB->IDR & RESET_BTN_PIN)) {
76
77                  RunReactionTest();
78
79                  while ((GPIOB->IDR & RESET_BTN_PIN));
80                  Delay_ms(50);
81                  while (!(GPIOB->IDR & RESET_BTN_PIN));
82                  Delay_ms(200);
83
84                  DisplayWelcomeScreen();
85              }
86          }
87      }
88  }
89
90  /* ================= LOGIC ================= */
91  void RunReactionTest(void) {
92      button_pressed_flag = 0;
93      false_start_flag = 0;
94      test_in_progress = 1;
95
96      LCD_Clear();
97      LCD_SendString("WAIT...");
98      LED_Control(0, 1); // Red On
99
100     uint32_t wait_time = GenerateRandomDelay();
101
102     // NON-BLOCKING WAIT & FALSE START CHECK
103     TIM2->CNT = 0;
104     while (TIM2->CNT < wait_time) {
105         if (!(GPIOB->IDR & PATIENT_BTN_PIN) || button_pressed_flag) {
106             false_start_flag = 1;
107             break;
108         }
109     }
110
111     if (false_start_flag) {
112         DisplayError();
113         LED_Control(1, 1);
114         test_in_progress = 0;
115         return;
116     }
117
118     // REAL TEST
119     TIM2->CNT = 0; // Reset Timer
120     LCD_Clear();
121     LCD_SendString("GO!");
122     LED_Control(1, 0); // Green On
123
124     while (!button_pressed_flag) {
125         if (TIM2->CNT > TIMEOUT_LIMIT) break;
126         if (!(GPIOB->IDR & PATIENT_BTN_PIN)) button_pressed_flag = 1;
127     }
128
129     if (button_pressed_flag) {
130         reaction_time_ms = TIM2->CNT;
131     } else {
132         reaction_time_ms = TIMEOUT_LIMIT;
133     }
134
135     LED_Control(0, 0);
136     DisplayResults();
137     test_in_progress = 0;
138 }
139
140 void DisplayWelcomeScreen(void) {
141     LCD_Clear();
142     LCD_SendString("Neuro React Test");
143     LCD_SetCursor(1, 0);
144     LCD_SendString("Press PB1(Start)");
```

```
145         LED_Control(0, 1);
146     }
147
148     void DisplayResults(void) {
149         LCD_Clear();
150         if (reaction_time_ms >= TIMEOUT_LIMIT) {
151             LCD_SendString("No Response!");
152         } else {
153             LCD_SendString("Time:");
154             LCD_Print_Number(reaction_time_ms);
155             LCD_SendString("ms");
156         }
157         LCD_SetCursor(1, 0);
158         LCD_SendString("Press PB1 Reset");
159     }
160
161     void DisplayError(void) {
162         LCD_Clear();
163         LCD_SendString("TOO EARLY!");
164         LCD_SetCursor(1, 0);
165         LCD_SendString("False Start :(");
166     }
167
168     /* ================= INTERRUPT ================= */
169     void EXTI0_IRQHandler(void) {
170         if (EXTI->PR & EXTI_PR_PR0) {
171             if (test_in_progress) button_pressed_flag = 1;
172             EXTI->PR = EXTI_PR_PR0;
173         }
174     }
175
176     /* ================= HELPERS & DRIVERS ================= */
177     void Delay_ms(volatile int ms) {
178         for (int i = 0; i < ms; i++) {
179             for (int j = 0; j < 3200; j++) __NOP();
180         }
181     }
182
183     void LED_Control(uint8_t green_on, uint8_t red_on) {
184         if (green_on) GPIOA->BSRR = GREEN_LED_PIN;
185         else          GPIOA->BSRR = GREEN_LED_PIN << 16;
186         if (red_on)   GPIOA->BSRR = RED_LED_PIN;
187         else          GPIOA->BSRR = RED_LED_PIN << 16;
188     }
189
190     uint32_t GenerateRandomDelay(void) {
191         return (rand() % (MAX_DELAY - MIN_DELAY + 1)) + MIN_DELAY;
192     }
193
194     /* ================= HARDWARE CONFIG (16MHz) ================= */
195     void SystemClock_Config(void) {
196         // 1. Enable HSI (16MHz Internal)
197         RCC->CR |= RCC_CR_HSION;
198         while (!(RCC->CR & RCC_CR_HSIRDY));
199
200         // 2. Select HSI as System Clock (No PLL needed)
201         RCC->CFGR &= ~RCC_CFGR_SW;      // Clear SW bits
202         RCC->CFGR |= RCC_CFGR_SW_HSI;   // Select HSI
203         while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI);
204
205         // 3. Configure Buses (AHB=1, APB1=1, APB2=1)
206         RCC->CFGR &= ~(RCC_CFGR_HPRE | RCC_CFGR_PPRE1 | RCC_CFGR_PPRE2);
207     }
208
209     void TIM2_Init(void) {
210         RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
211
212         // Calculation: 16,000,000 / 1000Hz = 16,000.
213         // 16,000 fits inside the 16-bit register (Max 65,535).
214         TIM2->PSC = 16000 - 1;
215
216         TIM2->ARR = 0xFFFFFFFF;
```

```
217         TIM2->EGR |= TIM_EGR_UG;
218         TIM2->SR &= ~TIM_SR_UIF;
219         TIM2->CR1 |= TIM_CR1_CEN;
220     }
221
222     void I2C1_Init(void) {
223         RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;
224         I2C1->CR1 |= I2C_CR1_SWRST;
225         I2C1->CR1 &= ~I2C_CR1_SWRST;
226
227         I2C1->CR2 = 16;  // APB1 = 16MHz
228
229         // CCR = T_high / T_pclk
230         // 5000ns / (1/16MHz = 62.5ns) = 80
231         I2C1->CCR = 80;
232
233         // TRISE = 1000ns / 62.5ns + 1 = 17
234         I2C1->TRISE = 17;
235
236         I2C1->CR1 |= I2C_CR1_PE;
237     }
238
239     void GPIO_Init(void) {
240         RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOBEN;
241         GPIOA->MODER |= (1<<0) | (1<<2); // LEDs
242         GPIOB->MODER &= ~(3U | (3U<<2)); // Buttons
243         GPIOB->PUPDR |= (1U<<0) | (1U<<2);
244
245         // I2C Open Drain
246         GPIOB->MODER |= (2<<12) | (2<<14);
247         GPIOB->OTYPER |= (1<<6) | (1<<7);
248         GPIOB->OSPEEDR |= (3<<12) | (3<<14);
249         GPIOB->PUPDR |= (1<<12) | (1<<14);
250         GPIOB->AFR[0] |= (4<<24) | (4<<28);
251     }
252
253     void EXTI_Init(void) {
254         RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
255         SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PB;
256         EXTI->IMR |= EXTI_IMR_MR0;
257         EXTI->FTSR |= EXTI_FTSR_TR0;
258         NVIC_EnableIRQ(EXTI0_IRQn);
259         NVIC_SetPriority(EXTI0_IRQn, 0);
260     }
261
262     void I2C_Start(void) {
263         I2C1->CR1 |= I2C_CR1_START;
264         while (!(I2C1->SR1 & I2C_SR1_SB));
265         I2C1->DR = SLAVE_ADDR << 1;
266         while (!(I2C1->SR1 & I2C_SR1_ADDR));
267         (void)I2C1->SR2;
268     }
269     void I2C_Stop(void) { I2C1->CR1 |= I2C_CR1_STOP; }
270     void I2C_Write(uint8_t data) {
271         I2C1->DR = data;
272         while (!(I2C1->SR1 & I2C_SR1_BTF));
273     }
274     void LCD_Write_Nibble(uint8_t nibble, uint8_t rs) {
275         uint8_t data = (nibble & 0xF0) | 0x08 | (rs ? 1:0);
276         I2C_Start(); I2C_Write(data | 0x04); I2C_Write(data | 0x00); I2C_Stop();
277         for(int i=0; i<2000; i++);
278     }
279     void LCD_SendByte(uint8_t byte, uint8_t rs) { LCD_Write_Nibble(byte & 0xF0, rs); LCD_Write_Nibble(byte
        << 4, rs); }
280     void LCD_SendCmd(uint8_t cmd) { LCD_SendByte(cmd, 0); }
281     void LCD_SendData(uint8_t data) { LCD_SendByte(data, 1); }
282     void LCD_SendString(char *str) { while (*str) LCD_SendData(*str++); }
283     void LCD_SetCursor(uint8_t row, uint8_t col) { LCD_SendCmd(row ? 0xC0 + col : 0x80 + col); }
284     void LCD_Clear(void) { LCD_SendCmd(0x01); Delay_ms(5); }
285     void LCD_Init(void) {
286         Delay_ms(50);
287         LCD_Write_Nibble(0x30, 0); Delay_ms(5);
```

```
288         LCD_Write_Nibble(0x30, 0); Delay_ms(5);
289         LCD_Write_Nibble(0x30, 0); Delay_ms(1);
290         LCD_Write_Nibble(0x20, 0); Delay_ms(5);
291         LCD_SendCmd(0x28); LCD_SendCmd(0x08); LCD_SendCmd(0x01); Delay_ms(5); LCD_SendCmd(0x06);
    LCD_SendCmd(0x0C);
292     }
293     void LCD_Print_Number(uint32_t num) {
294         char buf[12]; int i = 0;
295         if (num == 0) { LCD_SendData('0'); return; }
296         while (num) { buf[i++] = (num % 10) + '0'; num /= 10; }
297         while (i--) LCD_SendData(buf[i]);
298     }
```