```
#include "stm32f446xx.h" // Main device header for register definitions
#include <stdio.h>
#include <stdlib.h>

// --- Game Logic Typedefs and Defines ---
typedef enum {
   STATE_IDLE, STATE_COUNTDOWN, STATE_WAIT, STATE_MEASURE, STATE_DISPLAY
} GameState;

#define COUNTDOWN_STAGE_DELAY_MS 1000
#define RESET_HOLD_TIME_MS      2000
#define SS_DISPLAY_TIME_MS      500
#define MIN_WAIT_TIME_MS        500
#define REACTION_TIMEOUT_MS     3000

#define DEBOUNCE_TIME_MS        200
#define ADC_MAX_VALUE           4095
#define ADC_LEVEL_WIDTH         512
#define SEG_CHAR_DASH           10
#define SEG_CHAR_BLANK          11
#define SEG_CHAR_S              5


// --- Pin Definitions for Readability ---
#define D1_PIN 12
#define D2_PIN 13
#define D3_PIN 14
#define D4_PIN 15
#define D5_PIN 8
#define D6_PIN 9
#define D7_PIN 10
#define D8_PIN 11
#define BUZZER_PIN 2

// --- Game State Variables ---
GameState game_state = STATE_IDLE;
uint8_t current_difficulty_level = 1;
uint32_t reaction_start_time = 0;
uint32_t reaction_time_ms = 0;
uint32_t random_wait_time = 0;
uint32_t wait_start_time = 0;

// --- Interrupt Flags ---
volatile uint8_t reset_button_flag = 0;
volatile uint8_t start_button_flag = 0;
volatile uint8_t reaction_button_flag = 0;
```

```
// --- Systick Delay Variable ---
volatile uint32_t ms_ticks = 0;


// --- 7-Segment Patterns ---
const uint8_t seven_seg_patterns[] = {
   0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90, 0xBF, 0xFF
};


// --- Function Prototypes ---
void SystemClock_Init_Reg(void);
void Systick_Init_Reg(void);
void GPIO_Init_Reg(void);
void ADC_Init_Reg(void);
void Interrupt_Init_Reg(void);
void Delay_ms(uint32_t ms);
uint16_t ADC_Read_Reg(void);
void Perform_Game_Reset(void);
void Update_Difficulty_State(void);
void Generate_Random_Wait(void);
void Traffic_Light_Sequence(void);
void Display_Digit(uint8_t display_num, uint8_t pattern_index);

// --- ADDED MISSING PROTOTYPES TO FIX LINKER ERRORS ---
void Display_Number(uint8_t number);
void Display_Special(uint8_t pattern_idx1, uint8_t pattern_idx2);
// --- END OF FIX ---

void All_LEDs_On(void);
void All_LEDs_Off(void);
void Quick_Beep(void);
void Long_Beep(void);



/*=========================    INITIALIZATION FUNCTIONS ========================== */

void SystemClock_Init_Reg(void) {
   RCC->CR |= RCC_CR_HSION;//Turn ON High-Speed Internal Oscillator;set HSIO bit  Control Register
   while (!(RCC->CR & RCC_CR_HSIRDY));//Is HSI oscillator stable?
   RCC->CFGR &= ~RCC_CFGR_SW;//Clear SW[1:0] bits in CFGR register
   RCC->CFGR |= RCC_CFGR_SW_HSI;// Set SW[1:0] = 01 for HSI selection
   while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI);//Check Clock Switch Status
}


void Systick_Init_Reg(void) {      //Configure Timer Reload Value
   SysTick->LOAD = 16000 - 1;     //Timer counts from 15999 down to 0
```

```
  SysTick->VAL = 0;        //Clear Current Countvalue
  SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk |
SysTick_CTRL_ENABLE_Msk;      //Control Register Bit Analysis
}


void GPIO_Init_Reg(void) {
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOBEN|RCC_AHB1ENR_GPIOCEN
| RCC_AHB1ENR_GPIODEN; //Enable clock for GPIO ports A,B,C & D
  GPIOA->MODER |= (0b11 << (0 * 2)); //Set PA0 to analog mode
  GPIOA->MODER &= ~((0b11 << (2*2)) | (0b11 << (3*2))); // Clear bits for PA1
  GPIOA->MODER |= (0b01 << (8*2)) | (0b01 << (9*2)) | (0b01 << (10*2)) | (0b01 << (11*2));
  GPIOA->PUPDR |= (0b10 << (2*2)) | (0b10 << (3*2));
  GPIOB->MODER &= ~((0b11 << (0*2)) | (0b11 << (1*2)));
  GPIOB->MODER |= (0b01 << (12*2)) | (0b01 << (13*2)) | (0b01 << (14*2)) | (0b01 << (15*2));
  GPIOB->PUPDR |= (0b01 << (0*2)) | (0b01 << (1*2));// Sets Pull-Up resistors for PB0&PB1
  GPIOC->MODER = 0x55555555;// Sets ALL 16pins of GPIOC to output modes
  GPIOD->MODER |= (0b01 << (2*2));
// Sets PD2 to output mode; Calculation: (2*2)=4, configure bits [5:4] for PD2
}


void ADC_Init_Reg(void) {
  RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;//Enable ADC1 peripheral clock
  ADC->CCR |= (0b01 << 16);//Set ADCPRE = 01(PCLK2/4)
  ADC1->CR1 &= ~(ADC_CR1_RES | ADC_CR1_SCAN);//RES=00:12-bit resolution, SCAN=0
  ADC1->CR2 &= ~(ADC_CR2_CONT | ADC_CR2_ALIGN); //CONT=0: Single conversion; ALIGN=0
  ADC1->SQR1 &= ~ADC_SQR1_L;//L=0000:Convert 1 channel
  ADC1->SQR3 &= ~ADC_SQR3_SQ1;// SQ1=0000: Channel 0 (PA0)
  ADC1->CR2 |= ADC_CR2_ADON;//Turn on ADC1 peripheral
}


void Interrupt_Init_Reg(void) {
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN; //Enable SYSCFG peripheral clock
  SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI0; //Clear EXTI0 configuration bits
  SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PB; //Map EXTI0 to Port B Pin 0
  SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI1; // Clear  EXTI1 configuration bits
  SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PB; //Map EXTI1 to Port B Pin 1
  SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI2; //Clear EXTI2 configuration bits
  SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3; //Clear EXTI3 configuration bits
  EXTI->IMR |= EXTI_IMR_IM0 | EXTI_IMR_IM1 | EXTI_IMR_IM2 | EXTI_IMR_IM3;//Enable interrupt
generation for EXTI0-3
  EXTI->RTSR |= EXTI_RTSR_TR2 | EXTI_RTSR_TR3; //Enable rising edge trigger for EXTI2&EXTI3
  EXTI->FTSR &= ~(EXTI_FTSR_TR2 | EXTI_FTSR_TR3);// Disable falling edge trigger for EXTI2 &EXTI3
  EXTI->FTSR |= EXTI_FTSR_TR0 | EXTI_FTSR_TR1;   // Enable falling edge trigger for EXTI0 &EXTI1
  EXTI->RTSR &= ~(EXTI_RTSR_TR0 | EXTI_RTSR_TR1);    //Disable rising edge trigger for EXTI0&EXTI1
  NVIC_EnableIRQ(EXTI0_IRQn); //Enable EXTI0 interrupt in NVIC
```

```
    NVIC_EnableIRQ(EXTI1_IRQn); // Enable EXTI1 interrupt in NVIC
    NVIC_EnableIRQ(EXTI2_IRQn); //Enable EXTI2_interrupt in NVIC
    NVIC_EnableIRQ(EXTI3_IRQn); //Enable EXTI3_interrupt in NVIC
}


/*========================== INTERRUPT SERVICE ROUTINES ========================== */


void SysTick_Handler(void) {
    ms_ticks++;
}

void Button_ISR_Handler(volatile uint8_t* flag_ptr) {
    static uint32_t last_interrupt_time = 0;
    uint32_t current_time = ms_ticks;
    if (current_time - last_interrupt_time > DEBOUNCE_TIME_MS) {
        *flag_ptr = 1;
        last_interrupt_time = current_time;
    }
}



void EXTI0_IRQHandler(void) {
    if (EXTI->PR & EXTI_PR_PR0) {
        if (game_state == STATE_IDLE) Button_ISR_Handler(&start_button_flag);
        EXTI->PR |= EXTI_PR_PR0;
    }
}


void EXTI1_IRQHandler(void) {
    if (EXTI->PR & EXTI_PR_PR1) {
        if (game_state == STATE_WAIT || game_state == STATE_MEASURE)
Button_ISR_Handler(&reaction_button_flag);
        EXTI->PR |= EXTI_PR_PR1;
    }
}


void EXTI2_IRQHandler(void) {
    if (EXTI->PR & EXTI_PR_PR2) {
        Button_ISR_Handler(&reset_button_flag);
        EXTI->PR |= EXTI_PR_PR2;
    }
}
```

```
void EXTI3_IRQHandler(void) {
  if (EXTI->PR & EXTI_PR_PR3) {
    if (game_state == STATE_WAIT || game_state == STATE_MEASURE)
Button_ISR_Handler(&reaction_button_flag);
    EXTI->PR |= EXTI_PR_PR3;
  }
}

/*=========================MAIN FUNCTION=========================*/

int main(void) {
  SystemClock_Init_Reg();
  Systick_Init_Reg();
  GPIO_Init_Reg();
  ADC_Init_Reg();
  Interrupt_Init_Reg();


GPIOC->ODR |= (1 << 7) | (1 << 15);
  Quick_Beep();
  game_state = STATE_IDLE;


while (1) {
    if (reset_button_flag) {
      reset_button_flag = 0;
      Perform_Game_Reset();
    }

    switch (game_state) {
      case STATE_IDLE:
        Update_Difficulty_State();
        if (start_button_flag) {
          start_button_flag = 0;
          game_state = STATE_COUNTDOWN;
        }
        break;
case STATE_COUNTDOWN:
        All_LEDs_Off();
        Display_Special(SEG_CHAR_BLANK, SEG_CHAR_BLANK);
        Traffic_Light_Sequence();
        Display_Special(SEG_CHAR_S, SEG_CHAR_S);
        Delay_ms(SS_DISPLAY_TIME_MS);
        Display_Special(SEG_CHAR_BLANK, SEG_CHAR_BLANK);
        Generate_Random_Wait();
        wait_start_time = ms_ticks;
        game_state = STATE_WAIT;
        break;
case STATE_WAIT:
```

```
            if (reaction_button_flag) {
                reaction_button_flag = 0;
                All_LEDs_Off();
                Display_Number(88);
                Quick_Beep();
                Delay_ms(1000);
                game_state = STATE_IDLE;
                break;
            }
            if (ms_ticks - wait_start_time >= random_wait_time) {
                All_LEDs_On();
                reaction_start_time = ms_ticks;
                Long_Beep();
                game_state = STATE_MEASURE;
            }
            break;
case STATE_MEASURE:
            if (reaction_button_flag) {
                reaction_button_flag = 0;
                reaction_time_ms = ms_ticks –
reaction_start_time;
            game_state = STATE_DISPLAY;
            Quick_Beep();
            } else if (ms_ticks - reaction_start_time > REACTION_TIMEOUT_MS) {
                reaction_time_ms = REACTION_TIMEOUT_MS;
                game_state = STATE_DISPLAY;
            }
            if (game_state == STATE_DISPLAY) {
                uint8_t display_value = reaction_time_ms / 30;
                if (display_value > 99) display_value = 99;
                Display_Number(display_value);
                All_LEDs_On();
            }
            break;
        case STATE_DISPLAY:
            break;
    }
    Delay_ms(10);
  }
}

/*================GAME LOGIC & HARDWARE CONTROL FUNCTIONS====================*/

void Delay_ms(uint32_t ms) {
  uint32_t start_tick = ms_ticks;
  while ((ms_ticks - start_tick) < ms);
}


uint16_t ADC_Read_Reg(void) {
  ADC1->CR2 |= ADC_CR2_SWSTART;
```

```
  while (!(ADC1->SR & ADC_SR_EOC));
  return (uint16_t)(ADC1->DR);
}

void All_LEDs_On(void) {
  GPIOB->BSRR = (1 << D1_PIN) | (1 << D2_PIN) | (1 << D3_PIN) | (1 << D4_PIN);
  GPIOA->BSRR = (1 << (D5_PIN + 16)) | (1 << (D6_PIN + 16)) | (1 << (D7_PIN + 16)) | (1 << (D8_PIN +
16));
}

void All_LEDs_Off(void) {
  GPIOB->BSRR = (1 << (D1_PIN + 16)) | (1 << (D2_PIN + 16)) | (1 << (D3_PIN + 16)) | (1 << (D4_PIN +
16));
  GPIOA->BSRR = (1 << D5_PIN) | (1 << D6_PIN) | (1 << D7_PIN) | (1 << D8_PIN);
}

void Display_Digit(uint8_t display_num, uint8_t pattern_index) {
  if (pattern_index >= sizeof(seven_seg_patterns)) return;
  uint8_t pattern = seven_seg_patterns[pattern_index];
  uint32_t port_c_output = GPIOC->ODR;
  if (display_num == 0) {
    port_c_output &= 0xFFFFFF00;
    port_c_output |= (uint32_t)pattern;
  } else {
    port_c_output &= 0xFFFF00FF;
    port_c_output |= ((uint32_t)pattern << 8);
  }
  GPIOC->ODR = port_c_output;
}

void Display_Number(uint8_t number) {
  if (number > 99) number = 99;
  Display_Digit(0, number / 10);
  Display_Digit(1, number % 10);
}

void Display_Special(uint8_t pattern_idx1, uint8_t pattern_idx2) {
  Display_Digit(0, pattern_idx1);
  Display_Digit(1, pattern_idx2);
}

void Update_Difficulty_State(void) {
  uint32_t adc_value = ADC_Read_Reg();
  current_difficulty_level = (adc_value / ADC_LEVEL_WIDTH) + 1;
  if (current_difficulty_level > 8) current_difficulty_level = 8;
  All_LEDs_Off();
  if (current_difficulty_level >= 1) GPIOB->BSRR = (1 << D1_PIN);
```

```
    if (current_difficulty_level >= 2) GPIOB->BSRR = (1 << D2_PIN);
    if (current_difficulty_level >= 3) GPIOB->BSRR = (1 << D3_PIN);
    if (current_difficulty_level >= 4) GPIOB->BSRR = (1 << D4_PIN);
    if (current_difficulty_level >= 5) GPIOA->BSRR = (1 << (D5_PIN + 16));
    if (current_difficulty_level >= 6) GPIOA->BSRR = (1 << (D6_PIN + 16));
    if (current_difficulty_level >= 7) GPIOA->BSRR = (1 << (D7_PIN + 16));
    if (current_difficulty_level >= 8) GPIOA->BSRR = (1 << (D8_PIN + 16));
    Display_Digit(0, SEG_CHAR_BLANK);
    Display_Digit(1, current_difficulty_level);
}

void Traffic_Light_Sequence(void) {
    All_LEDs_Off();
    GPIOB->BSRR = (1 << D1_PIN); GPIOA->BSRR = (1 << (D5_PIN + 16));
Delay_ms(COUNTDOWN_STAGE_DELAY_MS);
    GPIOB->BSRR = (1 << (D1_PIN+16)); GPIOA->BSRR = (1 << D5_PIN);
    GPIOB->BSRR = (1 << D2_PIN); GPIOA->BSRR = (1 << (D6_PIN + 16));
Delay_ms(COUNTDOWN_STAGE_DELAY_MS);
    GPIOB->BSRR = (1 << (D2_PIN+16)); GPIOA->BSRR = (1 << D6_PIN);
    GPIOB->BSRR = (1 << D3_PIN); GPIOA->BSRR = (1 << (D7_PIN + 16));
Delay_ms(COUNTDOWN_STAGE_DELAY_MS);
    GPIOB->BSRR = (1 << (D3_PIN+16)); GPIOA->BSRR = (1 << D7_PIN);
    GPIOB->BSRR = (1 << D4_PIN); GPIOA->BSRR = (1 << (D8_PIN + 16));
Delay_ms(COUNTDOWN_STAGE_DELAY_MS);
    All_LEDs_Off();
}

void Generate_Random_Wait(void) {
    srand(ms_ticks);
    uint32_t max_additional_wait = (current_difficulty_level * 1500) - MIN_WAIT_TIME_MS;
    if (max_additional_wait <= 0) {
        random_wait_time = MIN_WAIT_TIME_MS;
    } else {
        random_wait_time = MIN_WAIT_TIME_MS + (rand() % (max_additional_wait + 1));
    }
}

void Perform_Game_Reset(void) {
    All_LEDs_On();
    Display_Special(SEG_CHAR_DASH, SEG_CHAR_DASH);
    GPIOD->BSRR = (1 << BUZZER_PIN);
    Delay_ms(RESET_HOLD_TIME_MS);
    GPIOD->BSRR = (1 << (BUZZER_PIN + 16));
    All_LEDs_Off();
    Display_Special(SEG_CHAR_BLANK, SEG_CHAR_BLANK);
    Delay_ms(100);
    Display_Number(88);
    Delay_ms(500);
```

```
    game_state = STATE_IDLE;
}

void Quick_Beep(void) {
    GPIOD->BSRR = (1 << BUZZER_PIN);
    Delay_ms(50);
    GPIOD->BSRR = (1 << (BUZZER_PIN + 16));
}


void Long_Beep(void) {
    GPIOD->BSRR = (1 << BUZZER_PIN);
    Delay_ms(500);
    GPIOD->BSRR = (1 << (BUZZER_PIN + 16));
}
```