

班級：資訊三乙 學號：D1210799 姓名：王建葦

一、【實驗目的】：

What was your design? What were the concepts you have used for your design?

本次實驗分為兩部分，重點在於中斷控制 (Interrupt)、ADC 類比轉數位輸入、以及 LCD 動態圖形顯示的整合應用。

1. Lab8.1 — 數字會跑步 (Running Numbers with External Interrupt)

- 隨機產生四個「非零、且互不重複」的數字，並分別顯示於 LCD 的 Line0~Line3 (位置 (0,0)、(0,16)、(0,32)、(0,48)) 。
- 按下 PB15 產生外部中斷後，四個數字開始往右移動，移動速度依數值大小不同：

數值越大移動越快，分別對應速度 2、4、6、8 pixels / 0.2 秒。

- 當最大的數字到達右邊界，對應的 LED (PC12~PC15) 亮起。
- 當所有數字都到達右邊界後，按下任意鍵即可重新開始。

實作重點包括：

1. EINT1 外部中斷設定 (PB15 Falling Edge Trigger)
 2. 亂數生成與速度對應關係 (Bubble Sort 排序 + 映射)
 3. 多物件同步位移與 LCD 即時更新。
2. Lab8.2 — Ping Pong 遊戲 (PingPong with ADC & Buzzer)

- 在 LCD 畫面下方設置一個 16x8 的「球拍」，上半部有一個固定的 16x8 方形障礙物。
- 按下任意鍵啟動遊戲，球 (8x8) 隨機往四個方向（左上、右上、左下、右下）移動。
- 轉動可變電阻 VR1 (ADC Channel 7) 控制球拍左右移動。
- 當球碰到球拍時，蜂鳴器響一聲並反射回彈；若沒接到球，LCD 顯示 "GAME OVER" 並結束遊戲。

本部分練習整合：

- 1. ADC 類比輸入 → 球拍座標轉換 (0~4095 → 0~112)
- 2. 碰撞偵測 (AABB Bounding Box)
- 3. 蜂鳴器 GPIO 控制 (Active Low)
- 4. 遊戲狀態機 (Init / Playing / GameOver)。

二、【遭遇的問題】：

What problems you faced during design and implementation?

- Lab8.1 中外部中斷偶爾誤觸發，導致數字提前開始移動。
- LCD 顯示更新速度過快時畫面閃爍明顯。
- 四個數字移動速度同步時，LED 點亮邏輯出現重複觸發。
- Lab8.2 中球拍移動不平滑，ADC 取樣值變化太劇烈。
- 球與障礙物同時碰撞時會產生錯誤反彈方向。

- 若球在邊界與球拍之間重疊太多畫面會卡死或「Game Over」未正常顯示。

三、【解決方法】：

How did you solve the problems?

- 於 PB15 外部中斷設定中加入去彈跳機制 (GPIO_SET_DEBOUNCE_TIME) · 確保只在按鍵明確下降緣時觸發。
- 將 LCD 更新封裝於 draw_all() 函式，確保一次性清除與重繪，減少閃爍。
- 設置 led_shown 旗標，確保 LED 僅在第一個到達邊界的數字亮起。
- 在 Lab8.2 中對 ADC 取樣進行 8 次平均 ($adc_val = adc_val / 8;$) · 穩定球拍移動速度。
- 改進碰撞函式 Check_Collision() · 增加邊界判斷，防止多重接觸誤反彈。
- 在遊戲主迴圈中使用 GameState 狀態機區分階段 (Init / Playing / GameOver) · 避免邏輯交錯。
- 蜂鳴器以 Beep() 函式封裝，延遲 50ms，確保聲音可辨識且不阻塞畫面刷新。

在找尋這些問題的解決方法與問題點時，我有使用 ChatGPT 協助我找尋與解決問題。包含 實驗結報的內容修改與潤飾都有使用 ChatGPT 協助。

四、【未能解決的問題】：

Was there any problem that you were unable to solve? Why was it unsolvable?

- 當 Lab8.1 四個數字同時到達邊界時，LED 僅能顯示最後更新的結果，無法同時亮起四顆。
- 亂數初始化以固定種子 `srand(1234)` 執行，每次重啟結果相同，尚未導入系統時間為隨機來源。
- PingPong 遊戲中球速固定，未能根據遊玩時間或次數動態增加難度。
- 由於 LCD 僅採單緩衝更新，畫面仍存在極輕微閃爍現象。
- 球體與障礙物碰撞時，未實作蜂鳴器提示，僅在球拍碰撞時發聲。

五、【程式碼】：

Lab8.1:

```

#include <stdio.h>
#include <stdlib.h>
#include "NUC100Series.h"
#include "MCU_init.h"
#include "SYS_init.h"
#include "LCD.h"
#include "Scankey.h"

// =====
// 常數定義
// =====
// 終點位置：LCD寬度128 - 字元寬度6 = 122
// 使用5x7字元顯示，考慮邊界留白
#define RIGHT_BOUND 122

// =====
// 資料結構定義
// =====
/***
 * @brief 移動物件結構體
 * @param num 數字值 (1-9)
 * @param x 當前X座標位置
 * @param speed 移動速度 (像素/次)
 * @param reached 是否已到達終點 (1=已到達, 0=未到達)
 */
typedef struct {
    int num;          // 數字值 (1-9)
    int x;            // 當前X座標位置 (0-122)
    int speed;        // 移動速度 (像素/次, 值為2, 4, 6, 8)
    int reached;      // 是否已到達終點 (1=已到達, 0=未到達)
} MOVING;

// =====
// 全域變數
// =====
MOVING obj[4];           // 4個移動物件的陣列
volatile int start_flag = 0; // 外部中斷啟動旗標 (volatile確保編譯器不優化)

// =====
// 延遲函數
// =====
/***
 * @brief 毫秒級延遲函數
 * @param ms 延遲的毫秒數
 * @note 使用系統時鐘延遲，每次迴圈延遲1ms
 */
void Delay_ms(int ms)
{
    int i;
    // 迴圈ms次，每次延遲1ms (1000微秒)
    for(i = 0; i < ms; i++)
        CLK_SysTickDelay(1000); // 1000微秒 = 1毫秒
}

```

```

// =====
//          外部中斷處理
// =====
/***
 * @brief 外部中斷1中斷服務程式 (EINT1 IRQ Handler)
 * @note 當PB15腳位發生下降緣時觸發此中斷
 * @note 此函數由硬體自動呼叫，執行時間應盡量短
 */
void EINT1_IRQHandler(void)
{
    // 設定啟動旗標，通知主程式開始競賽
    start_flag = 1;

    // 清除PB15的中斷來源旗標 (ISRC: Interrupt Source Register)
    PB->ISRC = (1 << 15);

    // 清除NVIC中的待處理中斷，確保中斷正確處理
    NVIC_ClearPendingIRQ(EINT1_IRQn);
}

/***
 * @brief 初始化外部中斷1 (PB15腳位)
 * @note 設定PB15為輸入模式，啟用下降緣觸發中斷
 * @note 啟用硬體防彈跳功能，避免按鈕震動造成多次觸發
 */
void init_EINT1(void)
{
    // 設定PB15為輸入模式
    GPIO_SetMode(PB, BIT15, GPIO_MODE_INPUT);

    // 啟用PB15的外部中斷，設定為下降緣觸發 (按鈕按下時觸發)
    GPIO_EnableInt(PB, 15, GPIO_INT_FALLING);

    // 在NVIC中啟用EINT1中斷
    NVIC_EnableIRQ(EINT1_IRQn);

    // 啟用PB15的防彈跳功能 (DBEN: Debounce Enable)
    PB->DBEN |= (1 << 15);

    // 設定防彈跳時鐘源為LIRC (低頻內部振盪器)，防彈跳時間為64個時鐘週期
    // 這可以過濾掉按鈕按下時的機械震動
    GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_64);
}

```

```

// =====
//          LED控制函數
// =====
/***
 * @brief 初始化LED腳位 (PC12~PC15)
 * @note LED為共陽極連接，輸出1 (高電位) 時LED熄滅，輸出0 (低電位) 時LED點亮
 */
void init_LED(void)
{
    // 設定PC12~PC15為輸出模式
    GPIO_SetMode(PC, BIT12, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PC, BIT13, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PC, BIT14, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PC, BIT15, GPIO_MODE_OUTPUT);

    // 初始化所有LED為熄滅狀態 (高電位)
    PC12 = PC13 = PC14 = PC15 = 1;
}

/***
 * @brief 點亮指定索引的LED
 * @param idx LED索引 (0-3) ，對應PC12-15
 * @note 只有對應索引的LED點亮 (低電位) ，其他LED熄滅 (高電位)
 */
void LED_On(int idx)
{
    // 根據索引點亮對應的LED (輸出0) ，其他LED熄滅 (輸出1)
    PC12 = (idx == 0 ? 0 : 1); // idx=0時點亮PC12
    PC13 = (idx == 1 ? 0 : 1); // idx=1時點亮PC13
    PC14 = (idx == 2 ? 0 : 1); // idx=2時點亮PC14
    PC15 = (idx == 3 ? 0 : 1); // idx=3時點亮PC15
}

/***
 * @brief 關閉所有LED
 * @note 將所有LED腳位設為高電位 (熄滅狀態)
 */
void LED_OffAll(void)
{
    // 所有LED輸出高電位 (熄滅)
    PC12 = PC13 = PC14 = PC15 = 1;
}

```

```
// =====
//          LCD繪圖函數
// =====
/***
 * @brief 繪製所有移動物件到LCD
 * @note 清除整個LCD畫面，然後在對應位置顯示4個數字
 * @note 每個數字垂直間距16像素，使用5x7字元顯示
 */
void draw_all(void)
{
    int i;

    // 清除整個LCD畫面
    clear_LCD();

    // 繪製4個移動物件
    for(i = 0; i < 4; i++)
        // 在座標(obj[i].x, i*16)位置顯示數字
        // obj[i].num + '0' 將數字轉換為ASCII字元
        // i*16 確保每個數字垂直間距16像素 (0, 16, 32, 48)
        printC_5x7(obj[i].x, i * 16, obj[i].num + '0');
}

/**
 * @brief 檢查所有物件是否都已到達終點
 * @return 1表示所有物件都已到達，0表示還有物件未到達
 */
int all_done(void)
{
    int i;
    // 檢查所有物件的reached旗標
    for(i = 0; i < 4; i++)
        if(!obj[i].reached) return 0; // 如果有任何物件未到達，返回0
    return 1; // 所有物件都已到達
}
```

```
// =====
//          數字產生與速度分配
// =====
/***
 * @brief 產生4個不重複的隨機數字並分配移動速度
 * @note 步驟1：產生4個1–9的不重複隨機數字
 * @note 步驟2：複製數字陣列用於排序
 * @note 步驟3：將數字由大到小排序
 * @note 步驟4：根據數字大小分配速度（大數字速度快）
 *
 * 速度分配規則：
 * - 最大數字：速度 = 8 像素/次
 * - 第二大數字：速度 = 6 像素/次
 * - 第三大數字：速度 = 4 像素/次
 * - 最小數字：速度 = 2 像素/次
 */
void generate_numbers(void)
{
    int used[10] = {0}; // 標記陣列，記錄已使用的數字（索引1–9）
    int numbers[4]; // 儲存產生的4個數字
    int sorted[4]; // 用於排序的數字陣列
    int i, j, temp;

    // ===== 步驟1：產生4個不重複的隨機數字 =====
    for(i = 0; i < 4; i++)
    {
        int r;
        // 持續產生隨機數直到得到未使用的數字
        do {
            r = rand() % 9 + 1; // 產生1–9的隨機數
        } while(used[r]); // 如果數字已使用，重新產生

        // 標記數字為已使用
        used[r] = 1;
        numbers[i] = r;

        // 初始化物件資料
        obj[i].num = r; // 設定數字值
        obj[i].x = 0; // 重置X座標為起始位置（最左側）
        obj[i].reached = 0; // 重置到達旗標
    }

    // ===== 步驟2：複製數字陣列用於排序 =====
    for(i = 0; i < 4; i++)
    {
        for(j = i + 1; j < 4; j++)
        {
            if(numbers[i] < numbers[j])
            {
                temp = numbers[i];
                numbers[i] = numbers[j];
                numbers[j] = temp;
            }
        }
    }
}
```

```

// =====
//      數字產生與速度分配
// =====
/***
 * @brief 產生4個不重複的隨機數字並分配移動速度
 * @note 步驟1：產生4個1–9的不重複隨機數字
 * @note 步驟2：複製數字陣列用於排序
 * @note 步驟3：將數字由大到小排序
 * @note 步驟4：根據數字大小分配速度（大數字速度快）
 *
 * 速度分配規則：
 * - 最大數字：速度 = 8 像素/次
 * - 第二大數字：速度 = 6 像素/次
 * - 第三大數字：速度 = 4 像素/次
 * - 最小數字：速度 = 2 像素/次
 */
void generate_numbers(void)
{
    int used[10] = {0}; // 標記陣列，記錄已使用的數字（索引1–9）
    int numbers[4]; // 儲存產生的4個數字
    int sorted[4]; // 用於排序的數字陣列
    int i, j, temp;

    // ===== 步驟1：產生4個不重複的隨機數字 =====
    for(i = 0; i < 4; i++)
    {
        int r;
        // 持續產生隨機數直到得到未使用的數字
        do {
            r = rand() % 9 + 1; // 產生1–9的隨機數
        } while(used[r]); // 如果數字已使用，重新產生

        // 標記數字為已使用
        used[r] = 1;
        numbers[i] = r;

        // 初始化物件資料
        obj[i].num = r; // 設定數字值
        obj[i].x = 0; // 重置X座標為起始位置（最左側）
        obj[i].reached = 0; // 重置到達旗標
    }

    // ===== 步驟2：複製數字陣列用於排序 =====
    for(i = 0; i < 4; i++)
        sorted[i] = numbers[i];

    // ===== 步驟3：氣泡排序（由大到小） =====
    // 外層迴圈：進行3輪比較（4個數字需要3輪）
    for(i = 0; i < 3; i++)
    {
        // 內層迴圈：比較剩餘未排序的數字
        for(j = i + 1; j < 4; j++)
        {
            // 如果後面的數字比前面的數字大，則交換
            if(sorted[j] > sorted[i])
            {
                temp = sorted[j];
                sorted[j] = sorted[i];
                sorted[i] = temp;
            }
        }
    }
}

// 排序完成後，sorted[0]為最大數字，sorted[3]為最小數字

```



```

// =====
//          主程式
// =====
/***
 * @brief 主程式：數字競賽遊戲
 * @note 遊戲流程：
 *      1. 產生4個不重複的隨機數字
 *      2. 根據數字大小分配移動速度
 *      3. 等待外部中斷按鈕啟動
 *      4. 數字開始移動，第一個到達終點的觸發LED
 *      5. 所有數字到達後等待按鍵繼續下一輪
 */
int main(void)
{
    int i;
    int led_shown = 0; // LED顯示旗標，確保只有第一個到達者觸發LED

    // ===== 系統初始化 =====
    SYS_Init();           // 系統時鐘和基本設定初始化
    init_LCD();           // LCD顯示器初始化
    clear_LCD();          // 清除LCD畫面
    OpenKeyPad();          // 按鍵矩陣初始化
    init_LED();           // LED腳位初始化
    init_EINT1();          // 外部中斷1 (PB15) 初始化

    // ===== 初始狀態設定 =====
    LED_OffAll();         // 關閉所有LED
    srand(1234);          // 設定隨機數種子（固定種子可重現結果）

    // ===== 主遊戲迴圈 =====
    while(1)
    {
        // 重置LED顯示旗標
        led_shown = 0;

        // 產生4個不重複的隨機數字並分配速度
        generate_numbers();

        // 繪製初始位置（所有數字在X=0位置）
        draw_all();

        // 重置啟動旗標，等待外部中斷
        start_flag = 0;

        // ===== 等待外部中斷按鈕啟動 =====
        // 當PB15按鈕按下時，EINT1_IRQHandler會設定start_flag=1
        while(!start_flag);

        // ===== 競賽移動迴圈 =====
        while(1)
        {
            // 更新每個物件的位置
            for(i = 0; i < 4; i++)
            {
                // 只處理尚未到達終點的物件
                if(!obj[i].reached)
                {
                    // 根據速度更新X座標
                    obj[i].x += obj[i].speed;

                    // 檢查是否到達終點
                    if(obj[i].x >= RIGHT_BOUND)
                    {

```

```

        // 限制X座標不超過終點
        obj[i].x = RIGHT_BOUND;
        // 標記為已到達
        obj[i].reached = 1;

        // 只有第一個到達終點的物件會觸發LED
        // 這確保即使多個物件同時到達，也只有第一個會顯示LED
        if(!led_shown)
        {
            LED_On(i);      // 點亮對應索引的LED
            led_shown = 1;  // 設定旗標，防止後續到達者觸發LED
        }
    }

    // 更新LCD顯示 (繪製所有物件的新位置)
    draw_all();

    // 延遲200毫秒，控制移動速度
    Delay_ms(200);

    // 檢查是否所有物件都已到達終點
    if(all_done())
        break; // 跳出移動迴圈
}

// ===== 等待按鍵繼續下一輪 =====
// 所有物件到達後，等待使用者按下任意按鍵
while(ScanKey() == 0);

// 關閉所有LED，準備下一輪
LED_OffAll();
}
}

```

Lab8.2:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "NUC100Series.h"
#include "MCU_init.h"
#include "SYS_init.h"
#include "LCD.h"
#include "Scankey.h"
// 2D繪圖函數庫，包含矩形繪製等功能
#include "Draw2D.h"

// =====
// 常數定義
// =====
// LCD顯示器尺寸
#define LCD_W 128          // LCD寬度 (像素)
#define LCD_H 64           // LCD高度 (像素)

// 硬體腳位定義
#define BUZZER_PIN 11      // 蜂鳴器腳位 (PB11)
#define ADC_VR_CHANNEL 7   // ADC可變電阻通道 (PA7)

// =====
// 遊戲物件尺寸定義
// =====
#define BALL_SIZE 8         // 球體尺寸 (8x8像素)
#define PADDLE_W 16          // 擋板寬度 (16像素)
#define PADDLE_H 8           // 擋板高度 (8像素)
#define OBSTACLE_W 16        // 障礙物寬度 (16像素)
#define OBSTACLE_H 8          // 障礙物高度 (8像素)

// =====
// 遊戲狀態列舉
// =====
/***
 * @brief 遊戲狀態列舉
 * @note STATE_INIT: 初始狀態，等待按鍵開始
 * @note STATE_PLAYING: 遊戲進行中
 * @note STATE_GAMEOVER: 遊戲結束，球體掉落底部
 */
typedef enum {
    STATE_INIT,           // 初始狀態
    STATE_PLAYING,        // 遊戲進行中
    STATE_GAMEOVER        // 遊戲結束
} GameState;

// =====
// 資料結構定義
// =====
/***
 * @brief 矩形結構體 (用於擋板和障礙物)
 * @param x 左上角X座標
 * @param y 左上角Y座標
 * @param w 寬度
 * @param h 高度
 */
typedef struct {
    int x, y; // 左上角座標
    int w, h; // 寬度和高度
} Rect;

```

```
/**  
 * @brief 球體物件結構體  
 * @param x 當前X座標  
 * @param y 當前Y座標  
 * @param dx X方向速度 (像素/次, 可為正負)  
 * @param dy Y方向速度 (像素/次, 可為正負)  
 * @param w 寬度  
 * @param h 高度  
 */  
typedef struct {  
    int x, y;      // 當前座標  
    int dx, dy;   // X和Y方向的速度  
    int w, h;     // 寬度和高度  
} BallObj;  
  
// ======  
// 全域變數  
// ======  
volatile GameState g_state = STATE_INIT; // 當前遊戲狀態 (volatile確保編譯器不優化)  
Rect g_paddle;                      // 檻板物件  
Rect g_obstacle;                     // 障礙物物件  
BallObj g_ball;                      // 球體物件  
  
// ======  
// 函數宣告  
// ======  
void Init_Hardware(void);           // 硬體初始化  
void Init_Game_Data(void);          // 遊戲資料初始化  
void Update_Paddle_Pos(void);       // 更新檻板位置 (根據ADC)  
void Beep(void);                   // 蜂鳴器響聲  
void Draw_Game(void);              // 繪製遊戲畫面
```

```

// =====
// 硬體初始化
// =====
/***
 * @brief 初始化所有硬體設備
 * @note 包含系統初始化、LCD、按鍵、ADC和蜂鳴器
 */
void Init_Hardware(void)
{
    // 基本系統初始化
    SYS_Init();          // 系統時鐘和基本設定
    init_LCD();          // LCD顯示器初始化
    clear_LCD();         // 清除LCD畫面
    OpenKeyPad();         // 按鍵矩陣初始化

    // ===== ADC初始化 (可變電阻VR1連接至PA7) =====
    // 解鎖暫存器寫入保護 (需要特定序列)
    SYS->REGWRPROT = 0x59;
    SYS->REGWRPROT = 0x16;
    SYS->REGWRPROT = 0x88;

    // 設定PA7為ADC功能 (多功能腳位設定)
    SYS->GPA_MFP |= (1UL << ADC_VR_CHANNEL);

    // 啟用ADC時鐘 (APBCLK bit 28)
    CLK->APBCLK |= (1UL << 28);

    // 選擇ADC時鐘源 (清除bit 2-3，使用預設時鐘)
    CLK->CLKSEL1 &= ~(0x3UL << 2);

    // 清除ADC時鐘分頻器 (使用預設分頻)
    CLK->CLKDIV &= ~(0xFFUL << 16);

    // 重新鎖定暫存器寫入保護
    SYS->REGWRPROT = 0x00;

    // 設定PA7為類比輸入模式 (清除PMD設定)
    PA->PMD &= ~(0x3UL << (ADC_VR_CHANNEL * 2));

    // 關閉數位輸入緩衝器 (類比輸入需要)
    PA->OFFD |= (1UL << ADC_VR_CHANNEL);

    // 啟用ADC (ADCR bit 0)
    ADC->ADCR |= (1UL << 0);

    // 啟用ADC通道7 (ADCHER: ADC Channel Enable Register)
    ADC->ADCHER |= (1UL << ADC_VR_CHANNEL);

    // ===== 蜂鳴器初始化 (PB11) =====
    // 清除PB11的模式設定
    PB->PMD &= ~(0x3UL << (BUZZER_PIN * 2));

    // 設定PB11為輸出模式 (PMD = 01)
    PB->PMD |= (0x1UL << (BUZZER_PIN * 2));

    // 初始化蜂鳴器為高電位 (不響)
    // 注意：蜂鳴器為Active-Low，低電位時響
    PB->DOUT |= (1UL << BUZZER_PIN);
}

```

```
// =====遊戲邏輯函數=====
// =====

/** 
 * @brief 初始化遊戲資料
 * @note 設定球體、擋板和障礙物的初始位置和速度
 */
void Init_Game_Data(void)
{
    int speed = 4; // 球體初始速度 (像素/次)

    // ===== 障礙物初始位置 =====
    // 障礙物位於螢幕上方中央
    g_obstacle.x = 56; // X座標 : (128-16)/2 = 56 (水平置中)
    g_obstacle.y = 8; // Y座標 : 距離頂部8像素
    g_obstacle.w = OBSTACLE_W;
    g_obstacle.h = OBSTACLE_H;

    // ===== 擋板初始位置 =====
    // 擋板位於螢幕底部中央
    g_paddle.x = 56; // X座標 : (128-16)/2 = 56 (水平置中)
    g_paddle.y = LCD_H - PADDLE_H; // Y座標 : 64-8 = 56 (緊貼底部)
    g_paddle.w = PADDLE_W;
    g_paddle.h = PADDLE_H;

    // ===== 球體初始位置和速度 =====
    // 球體位於螢幕中央
    g_ball.x = (LCD_W - BALL_SIZE) / 2; // X座標 : (128-8)/2 = 60 (水平置中)
    g_ball.y = (LCD_H - BALL_SIZE) / 2; // Y座標 : (64-8)/2 = 28 (垂直置中)
    g_ball.w = BALL_SIZE;
    g_ball.h = BALL_SIZE;

    // 隨機設定球體的初始移動方向
    // dx和dy可以是+speed或-speed，形成四個方向之一
    g_ball.dx = (rand() % 2 == 0) ? speed : -speed; // X方向：隨機左右
    g_ball.dy = (rand() % 2 == 0) ? speed : -speed; // Y方向：隨機上下
}
```

```

/***
 * @brief 更新擋板位置 (根據ADC可變電阻值)
 * @note 讀取ADC值並映射到擋板的X座標範圍
 * @note 使用8次取樣平均，提高穩定性
 */
void Update_Paddle_Pos(void)
{
    uint32_t adc_val = 0;
    int i;

    // ===== ADC取樣 (8次取樣平均) =====
    // 多次取樣取平均可以減少雜訊影響，提高穩定性
    for(i = 0; i < 8; i++)
    {
        // 啟動ADC轉換 (ADCR bit 11: ADST)
        ADC->ADCR |= (1UL << 11);

        // 等待轉換完成 (ADST位元自動清除)
        while(ADC->ADCR & (1UL << 11));

        // 讀取ADC轉換結果 (12位元，範圍0~4095)
        // ADDR[channel]的低12位元 (bit 0~11) 為轉換結果
        adc_val += (ADC->ADDR[ADC_VR_CHANNEL] & 0xFFFF);
    }

    // 計算平均值
    adc_val = adc_val / 8;

    // ===== ADC值映射到擋板X座標 =====
    // ADC值範圍：0~4095
    // 擋板X座標範圍：0 ~ (128-16) = 0~112
    // 使用線性映射：paddle_x = (adc_val * 112) / 4096
    g_paddle.x = (adc_val * (LCD_W - PADDLE_W)) / 4096;
}

/***
 * @brief 蜂鳴器響聲
 * @note 蜂鳴器為Active-Low，低電位時響
 * @note 響聲持續時間約50ms
 */
void Beep(void)
{
    // 設定PB11為低電位 (蜂鳴器響)
    PB->DOUT &= ~(1UL << BUZZER_PIN);

    // 延遲50ms (50000微秒)
    CLK_SysTickDelay(50000);

    // 設定PB11為高電位 (蜂鳴器停止)
    PB->DOUT |= (1UL << BUZZER_PIN);
}

```

```

/***
 * @brief 碰撞檢測函數 (AABB：軸對齊邊界框)
 * @param rect 矩形物件 (擋板或障礙物)
 * @param ball 球體物件
 * @return 1表示碰撞，0表示未碰撞
 * @note 使用AABB碰撞檢測，將球體視為矩形進行檢測
 */
int Check_Collision(Rect *rect, BallObj *ball)
{
    // AABB碰撞檢測：兩個矩形重疊的條件
    // 矩形A和矩形B重疊，當且僅當：
    // - A的左邊 < B的右邊 且
    // - A的右邊 > B的左邊 且
    // - A的上邊 < B的下邊 且
    // - A的下邊 > B的上邊
    if (ball->x < rect->x + rect->w &&
        ball->x + ball->w > rect->x &&
        ball->y < rect->y + rect->h &&
        ball->y + ball->h > rect->y)           // 球體左邊 < 矩形右邊
                                                // 球體右邊 > 矩形左邊
                                                // 球體上邊 < 矩形下邊
                                                // 球體下邊 > 矩形上邊
    {
        return 1; // 發生碰撞
    }
    return 0; // 未碰撞
}

/***
 * @brief 繪製遊戲畫面
 * @note 清除整個LCD，然後繪製球體、擋板和障礙物
 * @note 使用fill_Rectangle函數繪製實心矩形
 */
void Draw_Game(void)
{
    // 清除整個LCD畫面
    clear_LCD();

    // ===== 1. 繪製球體 (8x8像素白色矩形) =====
    // fill_Rectangle參數 : x0, y0, x1, y1, fgColor, bgColor
    // fgColor=1 (白色), bgColor=0 (黑色)
    fill_Rectangle(g_ball.x, g_ball.y,
                  g_ball.x + BALL_SIZE - 1, g_ball.y + BALL_SIZE - 1,
                  1, 0);

    // ===== 2. 繪製擋板 (16x8像素白色矩形) =====
    fill_Rectangle(g_paddle.x, g_paddle.y,
                  g_paddle.x + PADDLE_W - 1, g_paddle.y + PADDLE_H - 1,
                  1, 0);

    // ===== 3. 繪製障礙物 (16x8像素白色矩形) =====
    fill_Rectangle(g_obstacle.x, g_obstacle.y,
                  g_obstacle.x + OBSTACLE_W - 1, g_obstacle.y + OBSTACLE_H - 1,
                  1, 0);
}

```

```
// =====
//          主程式
// =====
/***
 * @brief 主程式：打磚塊遊戲
 * @note 使用狀態機控制遊戲流程
 * @note 狀態轉換：INIT -> PLAYING -> GAMEOVER -> INIT
 */
int main(void)
{
    // ===== 系統初始化 =====
    Init_Hardware(); // 初始化所有硬體設備
    srand(123);      // 設定隨機數種子（固定種子可重現結果）

    // ===== 主遊戲迴圈 =====
    while(1)
    {
        // ===== 狀態1：初始狀態 =====
        if (g_state == STATE_INIT)
        {
            // 初始化遊戲資料（球體、擋板、障礙物位置）
            Init_Game_Data();

            // 繪製初始畫面
            Draw_Game();

            // 等待按鍵開始遊戲
            // 在等待期間持續更新擋板位置（讓使用者可以預先調整）
            while(ScanKey() == 0) {
                Update_Paddle_Pos(); // 根據ADC更新擋板位置
                Draw_Game();         // 更新畫面顯示
                CLK_SysTickDelay(50000); // 延遲50ms
            }

            // 按鍵按下，進入遊戲進行狀態
            g_state = STATE_PLAYING;
        }
        // ===== 狀態2：遊戲進行中 =====
        else if (g_state == STATE_PLAYING)
        {
            // 更新擋板位置（根據ADC可變電阻）
            Update_Paddle_Pos();

            // ===== 更新球體位置 =====
            g_ball.x += g_ball.dx; // X方向移動
            g_ball.y += g_ball.dy; // Y方向移動

            // ===== 邊界碰撞檢測與反彈 =====
            // 左邊界碰撞
            if (g_ball.x <= 0) {
                g_ball.x = 0;           // 限制X座標不超出左邊界
                g_ball.dx = -g_ball.dx; // X方向速度反向（反彈）
            }
            // 右邊界碰撞
            else if (g_ball.x >= LCD_W - BALL_SIZE) {
                g_ball.x = LCD_W - BALL_SIZE; // 限制X座標不超出右邊界
                g_ball.dx = -g_ball.dx;       // X方向速度反向（反彈）
            }

            // 上邊界碰撞
            if (g_ball.y <= 0) {
                g_ball.y = 0;           // 限制Y座標不超出上邊界
                g_ball.dy = -g_ball.dy; // Y方向速度反向（反彈）
            }
        }
    }
}
```

```

// 上邊界碰撞
if (g_ball.y <= 0) {
    g_ball.y = 0;           // 限制Y座標不超出上邊界
    g_ball.dy = -g_ball.dy; // Y方向速度反向 (反彈)
}

// 下邊界碰撞 (球體掉落底部，遊戲結束)
if (g_ball.y >= LCD_H - BALL_SIZE) {
    g_state = STATE_GAMEOVER; // 切換到遊戲結束狀態
    CLK_SysTickDelay(200000); // 延遲200ms
}

// ===== 物件碰撞檢測 =====
// 球體與擋板碰撞
if (Check_Collision(&g_paddle, &g_ball)) {
    g_ball.dy = -g_ball.dy;           // Y方向速度反向 (向上反彈)
    g_ball.y = g_paddle.y - BALL_SIZE; // 調整球體位置，避免穿透擋板
    Beep();                          // 發出蜂鳴器聲響
}

// 球體與障礙物碰撞
if (Check_Collision(&g_obstacle, &g_ball)) {
    g_ball.dy = -g_ball.dy; // Y方向速度反向 (反彈)
}

// 繪製更新後的遊戲畫面
Draw_Game();

// 延遲100ms，控制遊戲速度
CLK_SysTickDelay(100000);
}

// ===== 狀態3：遊戲結束 =====
else if (g_state == STATE_GAMEOVER)
{
    // 清除畫面並顯示遊戲結束訊息
    clear_LCD();
    printS(30, 24, "GAME OVER"); // 在座標(30, 24)顯示"GAME OVER"

    // 等待按鍵重新開始
    while(ScanKey() == 0);

    // 延遲500ms，避免按鍵誤觸
    CLK_SysTickDelay(500000);

    // 切換回初始狀態，準備下一輪遊戲
    g_state = STATE_INIT;
}
}
}

```