微處理機系統實習　　Lab1

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 Lab9-1（貪食蛇一）與 Lab9-2（貪食蛇二），目標為使用 Joystick（ADC 類比輸入）操控蛇的方向，並在 LCD 上以 2×2 像素格子構成 64×32 的地圖。兩個實驗都要求蛇移動順暢、不使用 clear_LCD 清空全畫面，而是以「覆蓋式重繪」方式維持畫面穩定。

Lab9-1 — 基本貪食蛇

設計內容如下：

1. 蛇初始長度為 16 格，位於 y 軸中央並朝右移動。
2. Joystick 推動方向會以 X/Y ADC 偏移量判斷移動方向，若推力小於半徑 1/3 則不動。
3. 禁止蛇反方向移動（避免立即死亡）。
4. 蛇碰撞自身後遊戲停止；碰到邊界則可沿著邊界方向行走，不可超界。
5. 按下 Reset（PC0）可重新啟動遊戲。

Lab9-2 — 進階貪食蛇

在 9-1 基礎上增加：

1. 推動方向後放開，蛇會持續往該方向移動（latched direction）。
2. 地圖上會產生一個隨機「水果」，只有一個，且不得出現在蛇身上。

3. 吃到水果後：
    - 蛇長度加 1
    - 分數 +10
    - 七段顯示器顯示分數（不補零）
4. 透過 Timer0 中斷達成穩定的七段顯示掃描，避免閃爍。

兩個實驗的主要概念包含：

- ADC Joystick 類比輸入與方向判斷
- 遊戲邏輯狀態機（方向、死亡、吃果實、重生）
- LCD 像素繪圖與覆蓋式重繪
- 隨機數生成與碰撞檢查
- Timer 中斷驅動的七段顯示動態掃描

二、【遭遇的問題】：

What problems you faced during design and implementation?

- Joystick ADC 值在中心時有雜訊，導致蛇會微幅抖動或不自主移動。
- 方向判斷使用 X/Y 差值時會出現「斜向難以判斷」的問題。
- 若立即反向推動，蛇會立刻死亡，需要加入「禁止反向」判斷。
- LCD 因反覆繪製蛇身，容易產生殘影或更新不完全。
- 在 Lab9-2，水果位置若未排除蛇身，有機率生成於蛇身上導致錯誤。
- 七段顯示器因無掃描中斷，更新速度不均會導致閃爍。
- 吃到水果後蛇變長，尾巴更新與重繪順序容易出錯。

三、【解決方法】：

How did you solve the problems?

1. 設定 joystick 休止區 ADC_THRES（700），並以距離平方判斷是否超出休止範圍，成功消除雜訊誤動作。
2. 使用 abs(dx) 與 abs(dy) 比較確定方向，使判斷明確落在上下左右四個方向。
3. 在方向更新時加入反向限制，例如：若 current=RIGHT，則不能切換到 LEFT。
4. 不使用 clear_LCD，而是以 draw_Snake_Block(x,y,0) 覆蓋尾部，再繪製新的蛇頭，確保畫面平滑。
5. 以迴圈檢查水果是否出現在蛇身，每次無效就重新隨機產生【Q2-final.c 已實作】。
6. 在 Lab9-2 透過 Timer0 ISR 驅動七段掃描，使每 2.5ms 更新一位，完全消除閃爍問題。
7. 每次移動後重新繪製整條蛇，避免尾部殘影與長度變化造成的破圖。

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?

- 因 LCD 單緩衝特性，在蛇高速移動時仍可能有輕微閃爍，需雙緩衝才可根本改善。

- Joystick 類比值仍會受環境噪訊影響，若無平均濾波（moving average），方向切換仍可能偶爾不穩。
- 當蛇長度過長（接近 100）時，重繪時間變長，畫面更新速度無法保持一致。
- 隨機水果仍可能出現在視覺上不明顯的位置（邊界陰影），需額外標示或改變顏色才能更明顯辨識。
- 七段顯示器僅能顯示數字，若要顯示負分或特殊符號需額外設計編碼表。

五、【程式碼】：

Lab9.1:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "NUC100Series.h"
#include "MCU_init.h"
#include "SYS_init.h"
#include "LCD.h"

// ==========================================
//               常數定義
// ==========================================
// 貪食蛇遊戲參數
#define SNAKE_LEN   16  // 蛇身固定長度（16格）
#define GRID_W      64  // 格子寬度（LCD寬度128除以2，每個格子2x2像素）
#define GRID_H      32  // 格子高度（LCD高度64除以2，每個格子2x2像素）

// 搖桿ADC參數
#define ADC_CENTER  2048  // ADC中心值（12位元ADC，範圍0-4095，中心為2048）
#define ADC_THRES   700   // 休止區域閾值（約為中心值的1/3，避免微小震動造成誤動作）

// ==========================================
//               方向列舉
// ==========================================
/**
 * @brief 移動方向列舉
 * @note DIR_STOP: 停止（搖桿在休止區域內）
 * @note DIR_UP, DIR_DOWN, DIR_LEFT, DIR_RIGHT: 四個移動方向
 */
typedef enum {
    DIR_STOP = 0,   // 停止
    DIR_UP,         // 向上
    DIR_DOWN,       // 向下
    DIR_LEFT,       // 向左
    DIR_RIGHT       // 向右
} Direction;

// ==========================================
//               全域變數
// ==========================================
// ADC相關變數（volatile確保編譯器不優化，因為在中斷中更新）
volatile uint8_t u8ADF;         // ADC中斷旗標（未使用）
volatile uint16_t X_ADC, Y_ADC; // X軸和Y軸ADC值（在中斷中更新）
volatile uint8_t  B_Button;     // 按鈕狀態（未使用）

// 蛇身座標陣列（index 0 = 尾部，index 15 = 頭部）
int8_t snake_x[SNAKE_LEN];  // 蛇身X座標（格子座標系統）
int8_t snake_y[SNAKE_LEN];  // 蛇身Y座標（格子座標系統）

// 方向控制變數
Direction current_dir = DIR_RIGHT;  // 當前移動方向
Direction next_dir = DIR_RIGHT;     // 搖桿要求的下一個方向
```

```
// ============================================
//                ADC中斷處理
// ============================================
/**
 * @brief ADC中斷服務程式
 * @note 當ADC轉換完成時自動觸發此中斷
 * @note 讀取X軸（通道0）和Y軸（通道1）的ADC值
 * @note 此函數由硬體自動呼叫，執行時間應盡量短
 */
void ADC_IRQHandler(void)
{
    uint32_t u32Flag;

    // 取得ADC中斷旗標
    u32Flag = ADC_GET_INT_FLAG(ADC, ADC_ADF_INT);

    // 檢查是否為ADC轉換完成中斷
    if(u32Flag & ADC_ADF_INT) {
        // 讀取X軸ADC值（通道0）
        X_ADC = ADC_GET_CONVERSION_DATA(ADC, 0);
        // 讀取Y軸ADC值（通道1）
        Y_ADC = ADC_GET_CONVERSION_DATA(ADC, 1);
    }

    // 清除ADC中斷旗標
    ADC_CLR_INT_FLAG(ADC, u32Flag);
}

/**
 * @brief 初始化ADC（雙通道：X軸和Y軸）
 * @note 啟用ADC通道0和1，對應PA0和PA1（搖桿輸入）
 * @note 啟用ADC中斷，自動更新X_ADC和Y_ADC變數
 */
void Init_ADC(void)
{
    // 開啟ADC，設定輸入模式和操作模式，啟用通道遮罩（通道0和1）
    ADC_Open(ADC, ADC_INPUT_MODE, ADC_OPERATION_MODE, ADC_CHANNEL_MASK);

    // 開啟ADC電源
    ADC_POWER_ON(ADC);

    // 啟用ADC轉換完成中斷（ADF: ADC Data Flag）
    ADC_EnableInt(ADC, ADC_ADF_INT);

    // 在NVIC中啟用ADC中斷
    NVIC_EnableIRQ(ADC_IRQn);

    // 啟動ADC連續轉換
    ADC_START_CONV(ADC);
}
```

```
// ==========================================
//                繪圖函數
// ==========================================
/**
 * @brief 繪製蛇身方塊（2x2像素）
 * @param gx 格子X座標（0-63）
 * @param gy 格子Y座標（0-31）
 * @param color 顏色（1=白色，0=黑色）
 * @note 將格子座標轉換為像素座標（乘以2）
 * @note 每個格子由2x2像素組成，繪製4個像素點
 */
void draw_Snake_Block(int8_t gx, int8_t gy, uint16_t color)
{
    int16_t px;  // 像素X座標
    int16_t py;  // 像素Y座標
    uint16_t bg; // 背景顏色

    // 格子座標轉換為像素座標（每個格子2x2像素）
    px = gx * 2;
    py = gy * 2;

    // 計算背景顏色
    // color=1（白色方塊）時，bg=0（黑色背景）
    // color=0（黑色方塊）時，bg=1（白色背景）
    bg = (color == 0) ? 1 : 0;

    // 繪製2x2像素方塊（4個像素點）
    draw_Pixel(px,     py,     color, bg);  // 左上
    draw_Pixel(px + 1, py,     color, bg);  // 右上
    draw_Pixel(px,     py + 1, color, bg);  // 左下
    draw_Pixel(px + 1, py + 1, color, bg);  // 右下
}
```

```
/**
 * @brief 初始化蛇的位置
 * @note 將蛇放置在螢幕中央，水平排列，初始方向向右
 * @note 蛇身長度固定為16格，從左到右排列
 */
void init_Snake(void)
{
    int i;
    int8_t start_x;  // 起始X座標（尾部）
    int8_t start_y;   // 起始Y座標（所有節點相同）

    // 計算起始位置，使蛇水平置中
    // 起始X = 格子寬度/2 - 蛇長/2，確保蛇在螢幕中央
    start_x = (GRID_W / 2) - (SNAKE_LEN / 2);
    // Y座標置中
    start_y = (GRID_H / 2);

    // 設定初始方向為向右
    current_dir = DIR_RIGHT;
    next_dir = DIR_RIGHT;

    // 清除LCD畫面
    clear_LCD();

    // 繪製初始蛇身（16格，從左到右水平排列）
    for(i = 0; i < SNAKE_LEN; i++) {
        // 設定每個節點的座標（X遞增，Y固定）
        snake_x[i] = start_x + i;  // X座標：start_x, start_x+1, ..., start_x+15
        snake_y[i] = start_y;      // Y座標：全部相同（水平排列）

        // 繪製白色方塊（color=1）
        draw_Snake_Block(snake_x[i], snake_y[i], 1);
    }
}
```

```
// ========================================
//                搖桿邏輯處理
// ========================================
/**
 * @brief 更新搖桿邏輯，計算下一個移動方向
 * @note 根據ADC值計算搖桿偏移，判斷移動方向
 * @note 使用休止區域避免微小震動造成誤動作
 * @note 禁止直接反向移動，防止意外死亡
 */
void update_Joystick_Logic(void)
{
    int32_t dx;         // X軸偏移量（相對於中心）
    int32_t dy;         // Y軸偏移量（相對於中心）
    uint32_t dist_sq;  // 距離平方（避免開平方運算）
    uint32_t thres_sq; // 閾值平方
    Direction req_dir = DIR_STOP;  // 要求的方向

    // 計算搖桿相對於中心的偏移量
    dx = (int32_t)X_ADC - ADC_CENTER;  // X軸偏移 (-2048 ~ +2047)
    dy = (int32_t)Y_ADC - ADC_CENTER;  // Y軸偏移 (-2048 ~ +2047)

    // 計算距離平方（使用平方避免開平方運算，提高效率）
    dist_sq = (dx*dx) + (dy*dy);
    // 計算閾值平方
    thres_sq = ADC_THRES * ADC_THRES;  // 700^2 = 490000

    // ========== 休止區域檢測 ==========
    // 如果搖桿在休止區域內（距離中心小於閾值），則停止移動
    if (dist_sq < thres_sq) {
        next_dir = DIR_STOP;
        return;
    }

    // ========== 方向判斷 ==========
    // 比較X和Y軸的偏移量，較大者決定移動方向
    if (abs(dx) > abs(dy)) {
        // 水平方向移動（X軸偏移較大）
        if (dx > 0) req_dir = DIR_RIGHT;  // 向右
        else req_dir = DIR_LEFT;           // 向左
    } else {
        // 垂直方向移動（Y軸偏移較大）
        if (dy > 0) req_dir = DIR_DOWN;    // 向下
        else req_dir = DIR_UP;             // 向上
    }

    // ========== 反向限制 ==========
    // 禁止直接反向移動，防止意外死亡
    // 例如：當前向右移動時，不能直接切換到向左
    if (current_dir == DIR_RIGHT && req_dir == DIR_LEFT) return;
    if (current_dir == DIR_LEFT && req_dir == DIR_RIGHT) return;
    if (current_dir == DIR_UP && req_dir == DIR_DOWN) return;
    if (current_dir == DIR_DOWN && req_dir == DIR_UP) return;

    // 設定下一個方向
    next_dir = req_dir;
}
```

```
// ========================================
//               主程式
// ========================================
/**
 * @brief 主程式：基礎貪食蛇遊戲
 * @note 使用搖桿控制蛇的移動方向
 * @note 固定蛇身長度16格
 * @note 碰撞邊界或自身時遊戲結束
 */
int32_t main (void)
{
    int i;
    int8_t head_x, head_y;  // 當前蛇頭座標
    int8_t new_x, new_y;    // 新蛇頭座標
    int valid_move;         // 移動有效性旗標

    int game_over = 0;      // 遊戲結束旗標

    // ========== 1. 系統初始化 ==========
    SYS_Init();             // 系統時鐘和基本設定
    SYS_UnlockReg();        // 解鎖暫存器寫入保護

    // 啟用SPI3和ADC時鐘
    // SPI3用於LCD通訊，ADC用於搖桿輸入
    CLK->APBCLK |= (1 << 15) | (1 << 28);  // bit 15: SPI3, bit 28: ADC

    // ========== 2. 多功能腳位設定 ==========
    // SPI3腳位設定（LCD通訊）
    SYS->GPD_MFP |= (SYS_GPD_MFP_PD8_SPI3_SS0 |    // PD8: SPI3片選
                     SYS_GPD_MFP_PD9_SPI3_CLK |     // PD9: SPI3時鐘
                     SYS_GPD_MFP_PD11_SPI3_MOSI0);  // PD11: SPI3資料輸出

    // ADC腳位設定（搖桿輸入）
    SYS->GPA_MFP |= (SYS_GPA_MFP_PA0_ADC0 |  // PA0: ADC通道0（X軸）
                     SYS_GPA_MFP_PA1_ADC1);    // PA1: ADC通道1（Y軸）

    SYS_LockReg();          // 重新鎖定暫存器寫入保護

    // ========== 3. LCD重置序列 ==========
    // LCD需要特定的重置序列才能正常初始化
    GPIO_SetMode(PD, BIT12 | BIT14, GPIO_MODE_OUTPUT);  // PD12和PD14設為輸出
    PD14 = 1;               // PD14設為高電位
    PD12 = 1; CLK_SysTickDelay(10000);   // PD12高電位，延遲10ms
    PD12 = 0; CLK_SysTickDelay(10000);   // PD12低電位，延遲10ms（重置）
    PD12 = 1; CLK_SysTickDelay(100000);  // PD12高電位，延遲100ms（穩定）

    // ========== 4. 週邊設備初始化 ==========
    Init_ADC();             // 初始化ADC（搖桿輸入）
    init_LCD();             // 初始化LCD顯示器
    clear_LCD();            // 清除LCD畫面

    // PC0設為輸入模式，作為重置按鈕
    GPIO_SetMode(PC, BIT0, GPIO_MODE_INPUT);

    // ========== 5. 遊戲初始化 ==========
    init_Snake();           // 初始化蛇的位置
```

```
// ========== 主遊戲迴圈 ==========
while(1) {
    // ========== 重置按鈕檢測 ==========
    // PC0按下（低電位）時重置遊戲
    if (PC0 == 0) {
        init_Snake();        // 重新初始化蛇的位置
        game_over = 0;       // 清除遊戲結束旗標
        CLK_SysTickDelay(500000);  // 延遲500ms，避免按鈕誤觸
    }

    // ========== 遊戲結束處理 ==========
    // 如果遊戲結束，停止移動，只等待重置
    if (game_over == 1) {
        CLK_SysTickDelay(200000);  // 延遲200ms
        continue;                  // 跳過後續處理
    }

    // ========== 更新搖桿邏輯 ==========
    // 根據ADC值計算下一個移動方向
    update_Joystick_Logic();

    // ========== 移動處理 ==========
    // 只有當方向不為STOP時才移動
    if (next_dir != DIR_STOP) {

        // 取得當前蛇頭座標（陣列最後一個元素）
        head_x = snake_x[SNAKE_LEN - 1];
        head_y = snake_y[SNAKE_LEN - 1];

        // 初始化新座標為當前座標
        new_x = head_x;
        new_y = head_y;
        valid_move = 1;  // 預設移動有效

        // 更新當前方向
        current_dir = next_dir;

        // ========== 計算新蛇頭位置 ==========
        // 根據當前方向計算新座標
        switch(current_dir) {
            case DIR_UP:    new_y = head_y - 1; break;    // 向上：Y減1
            case DIR_DOWN:  new_y = head_y + 1; break;    // 向下：Y加1
            case DIR_LEFT:  new_x = head_x - 1; break;    // 向左：X減1
            case DIR_RIGHT: new_x = head_x + 1; break;    // 向右：X加1
            default: break;
        }

        // ========== 邊界檢查 ==========
        // 檢查新位置是否超出螢幕範圍
        if (new_x < 0 || new_x >= GRID_W || new_y < 0 || new_y >= GRID_H) {
            valid_move = 0;  // 超出邊界，移動無效
        }
```

```
            // ========== 計算新蛇頭位置 ==========
            // 根據當前方向計算新座標
            switch(current_dir) {
                case DIR_UP:    new_y = head_y - 1; break;    // 向上：Y減1
                case DIR_DOWN:  new_y = head_y + 1; break;    // 向下：Y加1
                case DIR_LEFT:  new_x = head_x - 1; break;    // 向左：X減1
                case DIR_RIGHT: new_x = head_x + 1; break;    // 向右：X加1
                default: break;
            }

            // ========== 邊界檢查 ==========
            // 檢查新位置是否超出螢幕範圍
            if (new_x < 0 || new_x >= GRID_W || new_y < 0 || new_y >= GRID_H) {
                valid_move = 0;  // 超出邊界，移動無效
            }

            // ========== 自身碰撞檢查 ==========
            // 檢查新位置是否與蛇身重疊（不包括尾部，因為尾部會被清除）
            if (valid_move) {
                for (i = 0; i < SNAKE_LEN - 1; i++) {
                    // 如果新位置與任何身體節點重疊
                    if (snake_x[i] == new_x && snake_y[i] == new_y) {
                        valid_move = 0;  // 碰撞自身，移動無效
                        game_over = 1;    // 設定遊戲結束旗標
                        break;
                    }
                }
            }

            // ========== 執行移動 ==========
            if (valid_move) {
                // 清除舊尾部（繪製黑色方塊）
                draw_Snake_Block(snake_x[0], snake_y[0], 0);

                // 蛇身前移：每個節點移動到前一個節點的位置
                for (i = 0; i < SNAKE_LEN - 1; i++) {
                    snake_x[i] = snake_x[i+1];  // X座標前移
                    snake_y[i] = snake_y[i+1];  // Y座標前移
                }

                // 新增蛇頭：將新位置設為最後一個元素（蛇頭）
                snake_x[SNAKE_LEN - 1] = new_x;
                snake_y[SNAKE_LEN - 1] = new_y;

                // 繪製新蛇頭（白色方塊）
                draw_Snake_Block(new_x, new_y, 1);

                // 重繪整條蛇（避免殘影問題）
                // 這確保所有節點都正確顯示，即使有繪圖延遲
                for (i = 0; i < SNAKE_LEN; i++) {
                    draw_Snake_Block(snake_x[i], snake_y[i], 1);
                }
            }
        }

        // 延遲200ms，控制移動速度
        CLK_SysTickDelay(200000);
    }
}
```

Lab9.2:

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

// ---------------- 定義常數 ----------------
#define MAX_SNAKE_LEN 100
#define GRID_W      64
#define GRID_H      32

#define ADC_CENTER  2048
#define ADC_THRES   700

typedef enum {
    DIR_STOP = 0,
    DIR_UP,
    DIR_DOWN,
    DIR_LEFT,
    DIR_RIGHT
} Direction;

// ---------------- 外部函式宣告 ----------------
// 確保你有 Seven_Segment.c 在 Library 中
extern void OpenSevenSegment(void);
extern void ShowSevenSegment(uint8_t no, uint8_t number);
extern void CloseSevenSegment(void);

// ---------------- 全域變數 ----------------
volatile uint8_t u8ADF;
volatile uint16_t X_ADC, Y_ADC;
volatile uint8_t  B_Button;

int8_t snake_x[MAX_SNAKE_LEN];
int8_t snake_y[MAX_SNAKE_LEN];
uint16_t current_len = 16;

int8_t fruit_x = -1;
int8_t fruit_y = -1;

int score = 0;
int game_over = 0;

Direction current_dir = DIR_RIGHT;
Direction next_dir = DIR_RIGHT;

// ---------------- [新功能] Timer 掃描相關變數 ----------------
volatile int8_t g_DisplayBuf[4] = {-1, -1, -1, -1};
volatile uint8_t g_ScanIndex = 0;
```

```
// ---------------- Timer0 中斷服務程式 (ISR) ----------------
// 這個函式由硬體自動呼叫，用來解決七段顯示器閃爍問題
void TMR0_IRQHandler(void)
{
    // 1. 清除 Timer0 中斷旗標
    TIMER0->TISR = 1;

    // 2. 關閉所有顯示 (消影)
    CloseSevenSegment();

    // 3. 顯示目前的位數
    if (g_DisplayBuf[g_ScanIndex] != -1) {
        ShowSevenSegment(g_ScanIndex, g_DisplayBuf[g_ScanIndex]);
    }

    // 4. 準備下一次掃描的位數
    g_ScanIndex++;
    if (g_ScanIndex >= 4) g_ScanIndex = 0;
}

// ---------------- 初始化 Timer0 ----------------
void Init_Timer0_For_Scan(void)
{
    // 開啟 Timer0 時鐘
    CLK->APBCLK |= (1 << 2); // TMR0_EN

    // 選擇 Timer0 時鐘源為 HXT (12MHz)
    CLK->CLKSEL1 &= ~(0x7 << 8);

    // 設定 Timer0: Prescaler=11, CMP=2500 -> 2.5ms 中斷一次 (400Hz)
    TIMER0->TCSR = 0;
    TIMER0->TCSR |= (11 << 0);
    TIMER0->TCMPR = 2500;
    TIMER0->TCSR |= (1 << 29); // IE (Interrupt Enable)
    TIMER0->TCSR |= (1 << 27); // Periodic mode

    NVIC_EnableIRQ(TMR0_IRQn);

    // 啟動 Timer0
    TIMER0->TCSR |= (1 << 30); // CEN
}

// ---------------- 更新顯示緩衝區 ----------------
void Update_Score_Display(int val)
{
    // 個位數
    g_DisplayBuf[0] = val % 10;

    // 十位數
    if (val >= 10) g_DisplayBuf[1] = (val / 10) % 10;
    else g_DisplayBuf[1] = -1;

    // 百位數
    if (val >= 100) g_DisplayBuf[2] = (val / 100) % 10;
    else g_DisplayBuf[2] = -1;

    // 千位數
    if (val >= 1000) g_DisplayBuf[3] = (val / 1000) % 10;
    else g_DisplayBuf[3] = -1;
}
```

```
// ---------------- ADC 相關函式 ----------------
void ADC_IRQHandler(void)
{
    uint32_t u32Flag;
    u32Flag = ADC_GET_INT_FLAG(ADC, ADC_ADF_INT);
    if(u32Flag & ADC_ADF_INT) {
        X_ADC = ADC_GET_CONVERSION_DATA(ADC, 0);
        Y_ADC = ADC_GET_CONVERSION_DATA(ADC, 1);
    }
    ADC_CLR_INT_FLAG(ADC, u32Flag);
}

void Init_ADC(void)
{
    ADC_Open(ADC, ADC_INPUT_MODE, ADC_OPERATION_MODE, ADC_CHANNEL_MASK );
    ADC_POWER_ON(ADC);
    ADC_EnableInt(ADC, ADC_ADF_INT);
    NVIC_EnableIRQ(ADC_IRQn);
    ADC_START_CONV(ADC);
}

// ---------------- 繪圖與遊戲邏輯 ----------------

void draw_Snake_Block(int8_t gx, int8_t gy, uint16_t color)
{
    int16_t px = gx * 2;
    int16_t py = gy * 2;
    uint16_t bg = (color == 0) ? 1 : 0;

    draw_Pixel(px,     py,     color, bg);
    draw_Pixel(px + 1, py,     color, bg);
    draw_Pixel(px,     py + 1, color, bg);
    draw_Pixel(px + 1, py + 1, color, bg);
}

void spawn_Fruit(void)
{
    int valid = 0;
    int i;
    int8_t rx, ry;

    srand(X_ADC + Y_ADC + score);

    while (!valid) {
        valid = 1;
        rx = rand() % GRID_W;
        ry = rand() % GRID_H;

        for (i = 0; i < current_len; i++) {
            if (snake_x[i] == rx && snake_y[i] == ry) {
                valid = 0;
                break;
            }
        }
    }
    fruit_x = rx;
    fruit_y = ry;
    draw_Snake_Block(fruit_x, fruit_y, 1);
}
```

```
void init_Game(void)
{
    int i;
    int8_t start_x = (GRID_W / 2) - (16 / 2);
    int8_t start_y = (GRID_H / 2);

    current_dir = DIR_RIGHT;
    next_dir = DIR_RIGHT;
    current_len = 16;
    score = 0;
    game_over = 0;

    clear_LCD();

    for(i = 0; i < current_len; i++) {
        snake_x[i] = start_x + i;
        snake_y[i] = start_y;
        draw_Snake_Block(snake_x[i], snake_y[i], 1);
    }

    spawn_Fruit();
    Update_Score_Display(score); // 初始分數顯示
}

void update_Joystick_Logic(void)
{
    int32_t dx = (int32_t)X_ADC - ADC_CENTER;
    int32_t dy = (int32_t)Y_ADC - ADC_CENTER;
    uint32_t dist_sq = (dx*dx) + (dy*dy);
    uint32_t thres_sq = ADC_THRES * ADC_THRES;
    Direction req_dir = DIR_STOP;

    if (dist_sq < thres_sq) return;

    if (abs(dx) > abs(dy)) {
        if (dx > 0) req_dir = DIR_RIGHT;
        else req_dir = DIR_LEFT;
    } else {
        if (dy > 0) req_dir = DIR_DOWN;
        else req_dir = DIR_UP;
    }

    if (current_dir == DIR_RIGHT && req_dir == DIR_LEFT) return;
    if (current_dir == DIR_LEFT && req_dir == DIR_RIGHT) return;
    if (current_dir == DIR_UP && req_dir == DIR_DOWN) return;
    if (current_dir == DIR_DOWN && req_dir == DIR_UP) return;

    next_dir = req_dir;
}
```

```
// ----------------- 主程式 -----------------
int32_t main (void)
{
    int i;
    int8_t head_x, head_y, new_x, new_y;
    int valid_move;

    SYS_Init();
    SYS_UnlockReg();
    CLK->APBCLK |= (1 << 15) | (1 << 28);
    SYS->GPD_MFP |= (SYS_GPD_MFP_PD8_SPI3_SS0 | SYS_GPD_MFP_PD9_SPI3_CLK | SYS_GPD_MFP_PD11_SPI3_MOSI0);
    SYS->GPA_MFP |= (SYS_GPA_MFP_PA0_ADC0 | SYS_GPA_MFP_PA1_ADC1);
    SYS_LockReg();

    GPIO_SetMode(PD, BIT12 | BIT14, GPIO_MODE_OUTPUT);
    PD14 = 1;
    PD12 = 1; CLK_SysTickDelay(10000);
    PD12 = 0; CLK_SysTickDelay(10000);
    PD12 = 1; CLK_SysTickDelay(100000);

    Init_ADC();
    OpenSevenSegment();
    init_LCD();

    // [重要] PC0 設為 Quasi，防止浮接造成一直 Reset
    GPIO_SetMode(PC, BIT0, GPIO_MODE_QUASI);

    // [重要] 初始化 Timer0 來負責七段顯示器掃描
    Init_Timer0_For_Scan();

    init_Game();

    while(1) {
        // 重置鍵
        if (PC0 == 0) {
            init_Game();
            CLK_SysTickDelay(500000);
        }

        // 遊戲結束，雖然卡住，但 Timer 中斷仍會在背景更新顯示器
        if (game_over) {
            CLK_SysTickDelay(200000);
            continue;
        }

        update_Joystick_Logic();
```

```
if (next_dir != DIR_STOP) {

    head_x = snake_x[current_len - 1];
    head_y = snake_y[current_len - 1];
    new_x = head_x;
    new_y = head_y;
    valid_move = 1;

    current_dir = next_dir;

    switch(current_dir) {
        case DIR_UP:    new_y = head_y - 1; break;
        case DIR_DOWN:  new_y = head_y + 1; break;
        case DIR_LEFT:  new_x = head_x - 1; break;
        case DIR_RIGHT: new_x = head_x + 1; break;
        default: break;
    }

    if (new_x < 0 || new_x >= GRID_W || new_y < 0 || new_y >= GRID_H) {
        valid_move = 0;
        game_over = 1;
    }

    if (valid_move) {
        for (i = 0; i < current_len - 1; i++) {
            if (snake_x[i] == new_x && snake_y[i] == new_y) {
                valid_move = 0;
                game_over = 1;
                break;
            }
        }
    }
```

```
            if (valid_move) {
                if (new_x == fruit_x && new_y == fruit_y) {
                    // 吃到水果
                    score += 10;
                    Update_Score_Display(score); // 更新顯示 Buffer

                    if (current_len < MAX_SNAKE_LEN) current_len++;

                    snake_x[current_len - 1] = new_x;
                    snake_y[current_len - 1] = new_y;

                    draw_Snake_Block(new_x, new_y, 1);
                    spawn_Fruit();
                } else {
                    draw_Snake_Block(snake_x[0], snake_y[0], 0);
                    for (i = 0; i < current_len - 1; i++) {
                        snake_x[i] = snake_x[i+1];
                        snake_y[i] = snake_y[i+1];
                    }
                    snake_x[current_len - 1] = new_x;
                    snake_y[current_len - 1] = new_y;
                    draw_Snake_Block(new_x, new_y, 1);
                }

                // 斷尾修復
                for (i = 0; i < current_len; i++) {
                    draw_Snake_Block(snake_x[i], snake_y[i], 1);
                }
                draw_Snake_Block(fruit_x, fruit_y, 1);
            }
        }

        CLK_SysTickDelay(200000);
    }
}
```