CHAPTER

1

認識 ESP32

ESP32 是一款由具備 Wi-Fi 與 Bluetooth 網路的 MCU 所組成的平價晶片，使其可用於製作各種物聯網（**Internet of Things, IoT**）應用。本章要來看看多種 ESP32 開發板並學會 ESP32 的基本開發方法。

本章主題如下：

- ESP32 簡介
- 採用 ESP32 的開發板
- 設定開發環境
- 使用 Espressif SDK 來開發 ESP32 程式
- 開發用於 ESP32 開發板的草稿碼程式

1.1 技術要求

開始之前，請確認你已準備好以下項目：

- 安裝好作業系統的電腦，作業系統可為 Windows、Linux 或 macOS。
- 一片 ESP32 開發板，建議使用 Espressif 自家的 ESP-WROVER-KIT 開發板。

1.2 ESP32 簡介

ESP32 是由 Espressif Systems 公司所推出的平價晶片。ESP32 在其單晶片上整合了 Wi-Fi（2.4 GHz）與 Bluetooth 4.2，支援傳統 Bluetooth 來建立像是 L2CAP、SDP、GAP、SMP、AVDTP、AVCTP、A2DP (SNK) 與 AVRCP (CT) 等連線。ESP32 也支援藍牙低功耗（Bluetooth Low Energy, BLE），包含了 L2CAP、GAP、GATT、SMP 與基於 GATT 的藍牙規範。ESP32 晶片 / 模組的詳細產品線請參考以下網址：

`https://www.espressif.com/en/products/modules/esp32`

ESP32 有兩大板型：晶片型與模組型，兩者的尺寸與腳位數量不同。如何選用 ESP32 版型完全取決於你的專案設計與目的。想要設計並製作一個整合在 PCB 電路板中的 IoT 方案時，ESP32 板型的實際尺寸就是你必須考量的因素之一。各種 ESP32 晶片與模組板型請參考：

`https://www.espressif.com/en/products/modules`

接著要來介紹一些採用了 ESP32 晶片或 ESP32 模組的開發板。

1.3 採用 ESP32 的開發板

由於 ESP32 具備了晶片與模組兩種板型，市面上已有多款整合了 ESP32 晶片或 ESP32 模組的開發板。本書不是要介紹如何做出一片具備 ESP32 的開發板。反之，我們將採用市面上方便取得的現成開發板。

以 ESP32 為基礎的開發板可分成兩種款式。第一種是由 Espressif 原廠生產的開發板，另一種則是來自該公司的業務夥伴或個人自造者。來看看市面上一些現有的 ESP32 開發板。

◉ ESP32 原廠開發套件

整體而言，Espressif 提供了一系列可直接使用的 ESP32 開發板。我們不用再大費周章去洗 PCB 電路板以及焊接 ESP32 晶片等等。Espressif 所推出的 ESP32 系列開發板請參考：

`https://www.espressif.com/en/products/hardware/development-boards`

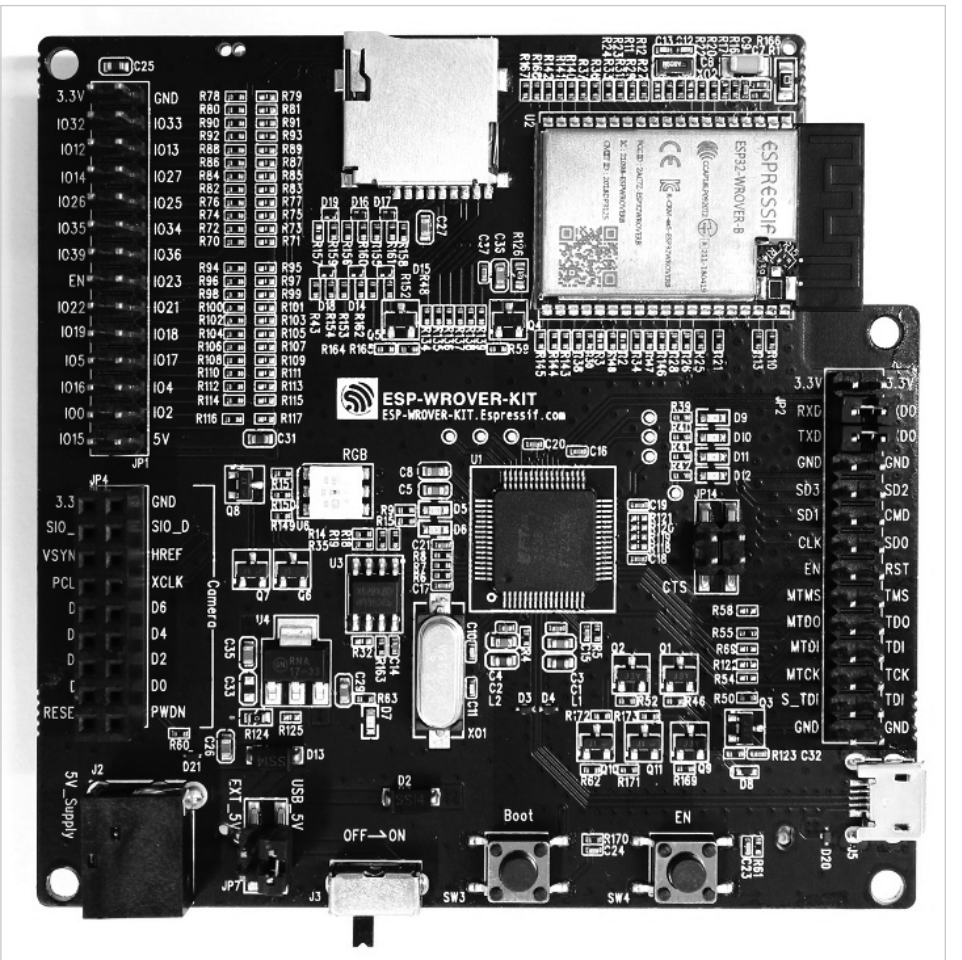
接著要介紹兩款 ESP32 開發板：ESP32- PICO-KIT 與 ESP-WROVER-KIT。ESP32-PICO-KIT 是一款體積輕巧的基礎開發板，可以直接搭配麵包板來接線。這片板子包含了像是 USB CP2102（v 4.0）與 CP2102N（v 4.1）等 ESP32 晶片。可以透過 USB 傳輸線將這片板子接上電腦。

ESP-WROVER-KIT 則是另一款功能完整的開發板，包含了各種感測器與模組。本板子採用了 ESP32-WROVER 來實作 ESP32 開發板的各項功能。ESP-WROVER-KIT 的主要功能如下：

- FT2232HL 的 JTAG 介面
- 相機接頭
- I/O 接頭

- RGB LED
- MicroSD 卡插槽
- LCD

ESP-WROVER-KIT 實體照片如下：



▲ 圖 1-1：ESP-WROVER-KIT

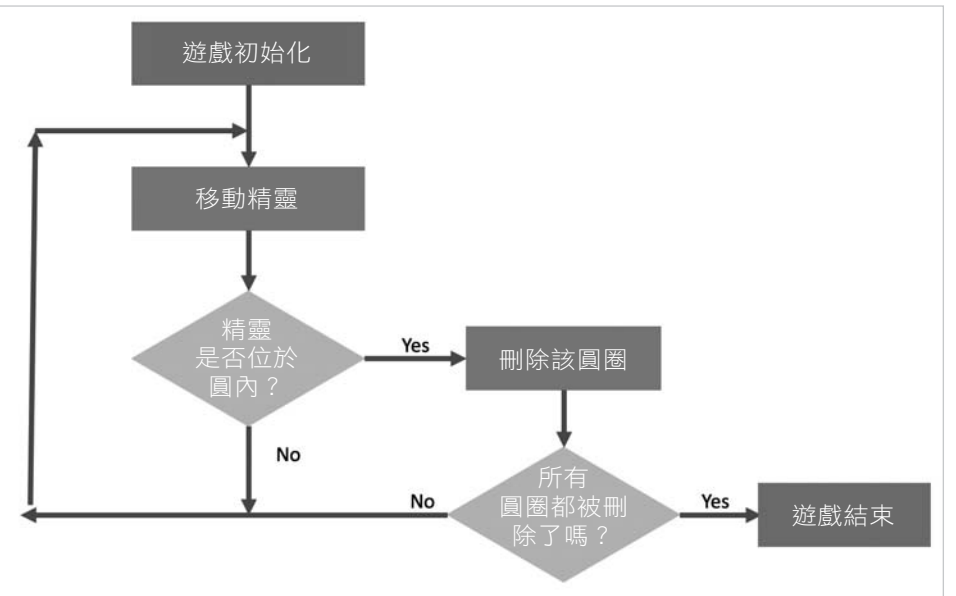
存檔之後，就可編譯並把程式燒錄到 ESP32 開發板了。都完成之後，你應該會聽到蜂鳴器發出聲響才對。

3.6 範例｜製作簡易嵌入式遊戲

本節要做一個簡易小遊戲，為此需要整合之前操作 LCD、類比搖桿與發聲蜂鳴器的相關經驗。在此會製作一個打球遊戲，一樣使用 ESP-WROVER-KIT ESP32 開發板來實作這個遊戲專案，開始吧！

◉ 遊戲情境

每個遊戲都有情境，有些遊戲還會定義使用者的等級。本專案也會建立一個簡易遊戲情境。這款遊戲的流程圖如圖 3-12：



▲ 圖 3-12：打球遊戲的遊戲情境

遊戲情境的製作步驟如下：

1. 遊戲一開始會出現一個隨機半徑的圓圈。這些圓圈是以精靈（sprite）的方式來呈現。
2. 設定球形精靈的位置於指定座標。
3. 使用者可操作類比搖桿來移動球形精靈。
4. 如果球跑到了圓圈裡面，讓發聲蜂鳴器發出聲響數秒鐘。
5. 如果並非如此，則無反應。
6. 如果所有圓圈都被刪除則代表遊戲完成，在 LCD 上顯示 `Game over`。

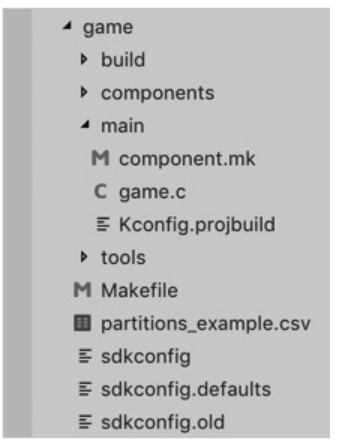
接著，要進行硬體接線並編寫遊戲程式了。

◉ 硬體接線

在此要整合 `joystick` 與 `buzzer` 專案的接線，請回顧本章先前的說明。

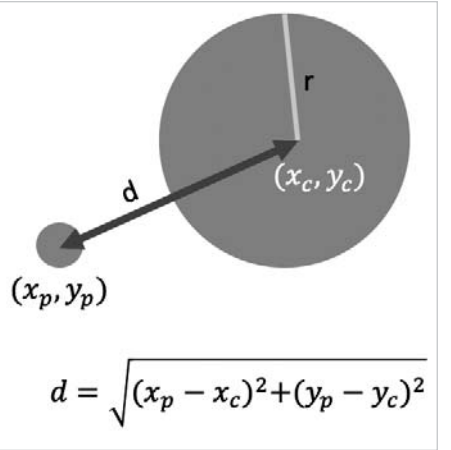
◉ 開發遊戲程式

現在，請建立一個名為 `game` 的專案。請複製第 2 章的天氣專案，專案結構如圖 3-13：



▲ 圖 3-13：遊戲的專案結構

有個簡單的方法來偵測球形精靈與圓圈是否發生碰撞。這需要計算精靈自身位置與圓心之間的距離，接著把點到圓心的距離與圓半徑進行比較，如果距離短於圓半徑，代表球形精靈位於圓圈之中。請參考圖 3-14：



▲ 圖 3-14：偵測碰撞的公式

現在要修改 game.c 中的 tft_demo() 函式。我們會在 tft_demo() 函式的最後呼叫 game_demo() 函式，如下：

```
void tft_demo() {

    ...
    game_demo();
}
```

本專案的遊戲情境是在 game_demo() 函式中實作：

1. 首先，初始化所有圓圈、ADC 與發聲蜂鳴器，如下：

```
static void game_demo()
{
  int x, y, r, i, n;
  n = 10;
  Circle_Sprite circles[n];
```

4.5 儲存感測器資料於 microSD 卡

技術上來說，我們可以把任何指定的資料存在 SD 卡與 microSD 卡中，當然也可以把感測器資料儲存於外部儲存裝置。

為了示範，我們要用 DHT 感測器來偵測溫度與濕度變化，硬體接線方式請參考第 2 章「在 LCD 上視覺化呈現資料與動畫」中的 dhtdemo 專案。本專案的情境是先偵測溫濕度變化，再把資料儲存在 microSD 卡中。

複製上一個 sdcard 專案並將其改名為 sdcarddht。其中的主程式也要由原本的 sdcard.c 改名為 sdcardht.c。在此要修改的地方是呼叫 dht_read_data() 函式來讀取 DHT 感測器的溫溼度數值。取得感測器資料之後，將其儲存在 sensor.txt 檔案中：

```
    ESP_LOGE(TAG, "Reading sensor data");

    if (dht_read_data(sensor_type, dht_gpio, &humidity, &temperature) == ESP_OK)
    {
        printf("Humidity: %d%% Temp: %d^C\n", humidity / 10, temperature / 10);
        FILE* f = fopen("/sdcard/sensor.txt", "a");
        if (f == NULL) {
            ESP_LOGE(TAG, "Failed to open file for writing");
            return;
        }
        fprintf(f, "Humidity: %d%% Temp: %d^C\n", humidity / 10, temperature / 10);
        fclose(f);
    }
    else
        printf("Could not read data from sensor\n");
```

現在，儲存程式。

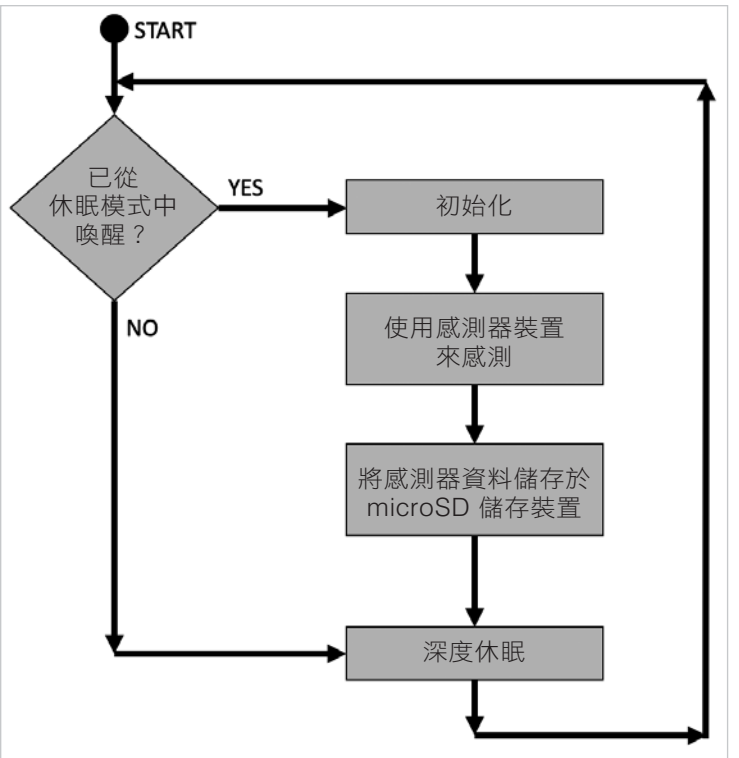
4.6 專案｜製作感測器監控紀錄器

本節要製作一個感測器監控紀錄器，會用到 ESP32 開發板與 DHT 感測器。在此，所有的感測器資料都會被儲存在 microSD 卡中的 CSV 檔裡面。後續可用 Excel 這類軟體來視覺化呈現感測器資料。

開始吧！

◉ 設計程式

技術上來說，本紀錄器專案的流程圖如下圖。在此會運用 ESP32 晶片的深度休眠功能來操作其休眠模式：



▲ 圖 4-6：紀錄器專案的程式設計

請將 request body 的數值改為 0，接著在 Postman 中按下 **Send** 按鈕，看看會發生什麼事。

5.7 智慧家庭專案

這個專案的目標是製作簡易的智慧家庭系統。所謂的智慧家庭是指能讓使用者控制家中諸多裝置的各種科技。這類應用會連接到許多感測器與裝置。我們可透過攝影機、溫度與電力用量這類感測器裝置來得知家中的資訊，另外也可以控制燈光為開啟或關閉。

本節將使用 ESP32 製作簡易的智慧家庭系統。圖 5-10 為使用 ESP32 開發板為核心之智慧家庭系統設計圖請參考。在此要把一些感測器與致動器裝置接上 ESP32 開發板，接著再透過網路進行控制。

如果想在戶外也能控制 ESP32 開發板的話，當然需要先啟動網路伺服器服務。下圖是透過網路伺服器發送給 ESP32 開發板的各個指令示意：

▲ **圖 5-10**：智慧家庭應用的簡易模型

簡單示範就好，在此會在 ESP32 開發板接上一顆 LED，並透過 HTTP 請求來控制 LED 亮滅。在此會定義 `/led` HTTP 請求來控制 LED，如果程式收到的請求主題中包含了數值 `1`，則 LED 亮起；收到數值 `0`，則讓 LED 熄滅。

專案開始！

◉ 硬體接線

本專案的硬體接線應該不會太麻煩；只需要在 ESP32 開發板的 IO12 腳位上接一顆 LED 就好。如果你所使用的 ESP32 開發板無法使用 IO12 腳位的話，請換成其他可用的腳位。

◉ 處理 HTTP 請求

現在要來編寫智慧家庭專案的程式了：

1. 建立名為 `smarthome` 的專案，主程式為 `smarthome.c`。

   首先，宣告專案所需的函式庫：

```
#include <esp_wifi.h>
#include <esp_event_loop.h>
#include <esp_log.h>
#include <esp_system.h>
#include <nvs_flash.h>
#include <sys/param.h>

#include <esp_http_server.h>
```

2. 定義用於 LED 的 GPIO 腳位與紀錄器：

```
static const char *TAG="APP";
#define LED1 12
```

3. 接著，透過 httpd_uri_t struct 來建立 HTTP POST 請求處理器。在此把請求方法定義為 HTTP_POST 搭配 /led 請求。另外也要把 led_post_handle() 函式送入請求處理器，如下：

```
httpd_uri_t led_post = {
    .uri = "/led",
    .method = HTTP_POST,
    .handler = led_post_handler,
    .user_ctx = NULL
};
```

4. 現在要實作 led_post_handler() 函式，負責讀取 HTTP 請求的訊息。接著，檢查 HTTP 請求主體是否包含數值 1 或 0。如果請求主體中包含了數值 1，就呼叫 gpio_set_level() 函式讓 LED 亮起來：

```
esp_err_t led_post_handler(httpd_req_t *req)
{
    char buf[100];
    int ret, remaining = req->content_len;

    while (remaining > 0) {
        buf[0] = '\0';
        if ((ret = httpd_req_recv(req, &buf, 1)) <= 0) {
            if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
                httpd_resp_send_408(req);
            }
            return ESP_FAIL;
        }
        buf[ret] = '\0';
        ESP_LOGI(TAG, "Recv HTTP => %s", buf);
        if (buf[0] == '1') {
            ESP_LOGI(TAG, "====================================");
            ESP_LOGI(TAG, ">>> Turn on LED");
            gpio_set_level(LED1, 1);
            httpd_resp_send_chunk(req, "Turn on LED", ret);
        }
        else
        if (buf[0] == '0') {
            ESP_LOGI(TAG, "====================================");
            ESP_LOGI(TAG, ">>> Turn off LED");
            gpio_set_level(LED1, 0);
```