Win 10 IoT 支援的硬體

2
CHAPTER

本章重點

本章節將介紹 Windows 10 IoT 支援的硬體。

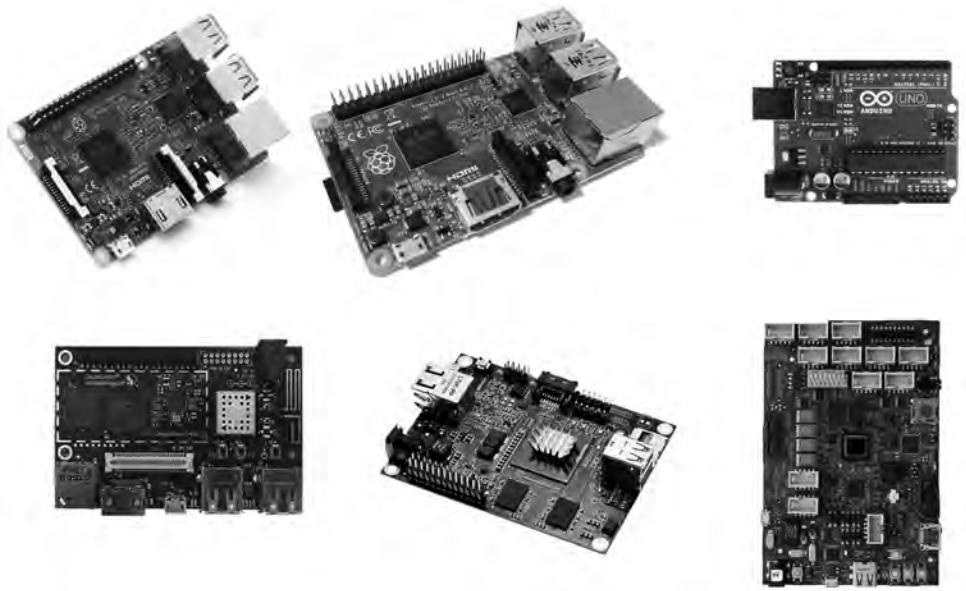

圖 2-0 Windows 10 IoT 支援的硬體

2.1 Windows 10 IoT Core 的硬體支援介紹

目前 Windows 10 IoT Core 的硬體支援有：

- Raspberry Pi 2 樹莓派 2
- Raspberry Pi 3 樹莓派 3
- MinnowBoard Max
- DragonBoard 410c

由硬體規格看來，Windows 10 同時支援多個版子，有 32 位元 ARM 架構，也有 64 位元 x86 架構，而又分為單核或為雙核。主要看開發者偏好哪種架構，以及所期望的價位與效能，從而去選擇樹莓派 2 和樹莓派 3、MinnowBoard Max 和 DragonBoard 410c 甚至 Arduino 都是可以支援的硬體，而本書將以樹莓派 2 為主，並經測試全部相容樹莓派 3。

2.2 Raspberry Pi 2 樹莓派 2

介紹

Raspberry Pi 2 Model B 在 2015 年 2 月 2 日釋出，並且連續 2 個月占據美國亞馬遜網站的電腦類產品銷售第一名。新版的樹莓派 2 使用 BCM2836 處理器 quad core ARMv7 的速度是 900MHz，現在售價 US$35。

樹莓派 2 的硬體為：

- 處理器 Broadcom BCM2836 ARMv7 Quad Core Processor 900MHz
- 記憶體 1GB RAM
- GPIO 的接腳延續前一版的排法
- 40pin 的 GPIO 接腳
- 網路 10/100 Ethernet Port

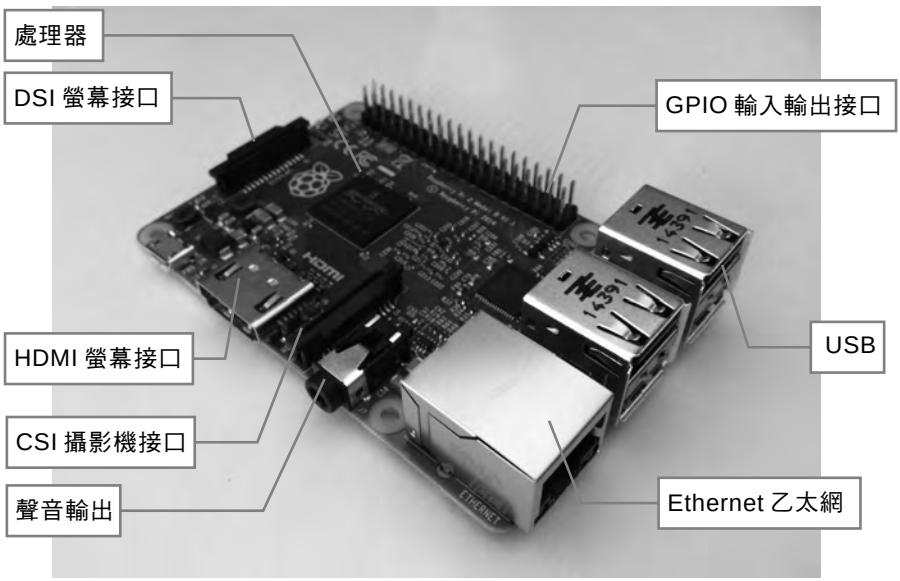

圖 2-1 Raspberry Pi 2 樹莓派 2 外觀

因為 Raspberry Pi 2 相當普及，所以本書所有的內容是針對 Raspberry Pi 2 撰寫，而樹莓派 2 詳細規格請看表 2-1。

表 2-1　樹莓派 2 的硬體

功能	規格
處理器	Broadcom BCM2836 ARMv7 四核心
處理器速度	四核心每個 900MHz
GPU	Videocore IV
記憶體	1GB SDRAM @ 450MHz
容量	microSD
USB	USB 2.0，4 個接口
電源	1.8A 5V
GPIO	40 Pin
大小	85 x 56 x 17 mm
重量	42 克

新的 SoC BCM2836

- 900MHz quad-core ARM Cortex-A7 CPU，大約有 6 倍的效能提昇。
- 記憶體容量加大到 1GB LPDDR2 SDRAM。
- 捨棄 PoP（package-on-package），而是將處理器和記憶體分別焊在板子的正反兩面。
- 因為採用 quad-core ARMv7 的處理器，所以會有較高的功耗，也就是比較耗電。

OS 的差異

但因為採用了 ARMv7 的處理器，所以可執行更多的 ARM GNU/Linux 版本，例如過去採用 ARMv6 指令集而無法執行 Ubuntu，現在也可以在 Raspberry Pi 2 上執行 Snappy Ubuntu Core。甚至可支援 Microsoft Windows 10 IoT，並且是免費提供，本書後面會有詳細的介紹，因為使用 ARMv7 的處理器，所以 Raspberry Pi 2 對 Android 作業系統的相容性就好很多。

相同的部分

- Model B+一樣的外型與尺寸，所以可以持續使用以前的保護外殼。
- Camera 的接線、LCE Display 的接線和 GPIO 40-pin 位置也相同。
- PCB 板固定螺絲開孔處相同。
- USB、Ethernet、A/V、HDMI、microSD 和 micro USB 位置相同，尺寸也相同。

2.3 Raspberry Pi 3 樹莓派 3

介紹

新的 Raspberry Pi 3 在 2016 年 2 月 28 日釋出，新版的樹莓派 3 使用的 Broadcom BCM2837 處理器是 ARM Cortex-A53 1.2Ghz 四核心處理器，並且有圖形處理器 GPU 部分為 400MHz VideoCore IV。Raspberry Pi 3 也加入了 BCM43438 晶片，裡面有 802.11n 無線網路與 Bluetooth 4.1 兩個新功能，速度是 900MHz，64 bit 並擁有 1GB LPDDR2 記憶體的 Raspberry Pi 3 售價為 35 元。

Raspberry Pi 3 的硬體為：

- Broadcom BCM2837 處理器是 ARM Cortex-A53 1.2Ghz 900MHz
- 記憶體 1GB RAM LPDDR2
- GPIO 的接腳延續前一版的排法
- 40pin 的 GPIO 接腳
- 網路 10/100 Ethernet Port

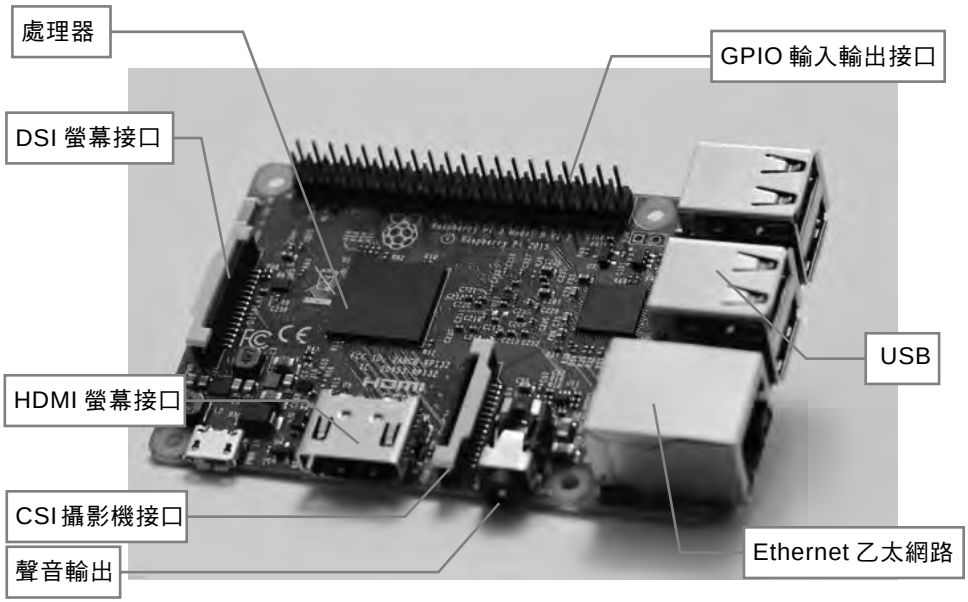

圖 2-2 Raspberry Pi 3 樹莓派 3 外觀

樹莓派 3 詳細規格請看表 2-2 的列表。

表 2-2 樹莓派 3 的硬體規格

功能	規格
處理器	Broadcom BCM2837 ARMv7 64 bit ARMv8 四核心
處理器速度	四核心每個 1.2GHz
GPU	Videocore IV 400MHz
記憶體	1GB SDRAM @ 450MHz
容量	microSD
USB	USB 2.0，4 個接口
電源	1.8A 5V
GPIO	40 Pin
大小	85 x 56 x 17 mm
重量	42 克

GPIO 接腳輸出控制

11 CHAPTER

本章重點

11.3 專題製作－霹靂燈專案

介紹

本章節將透過多個 LED 燈，做出霹靂燈的效果，而霹靂燈的效果說穿了，也就是讓所有的燈全部都亮起，並且依照指定的時間，讓其中的某一個燈光依序熄滅。

請準備 4 個 LED 燈，並且每一個 LED 燈都接上 220 歐姆的電阻，這樣可以避免 LED 燒壞。

硬體準備

-  Raspberry Pi 2 板子
-  4 個 LED
-  4 個 220 歐姆的電阻，顏色為紅紅棕，最後一色環金或銀
-  麵包板
-  數條接線

硬體接線

Raspberry Pi 2 接腳	元件接腳
Pin 7 / GPIO4	LED 長腳，並將 LED 短腳與電阻連接
Pin 29 / GPIO5	LED 長腳，並將 LED 短腳與電阻連接
Pin 31 / GPIO6	LED 長腳，並將 LED 短腳與電阻連接
Pin 32 / GPIO12	LED 長腳，並將 LED 短腳與電阻連接
GND	LED 短腳透過 220 ohm 電阻

硬體接線設計圖 ch11\11-3\11-3.fzz

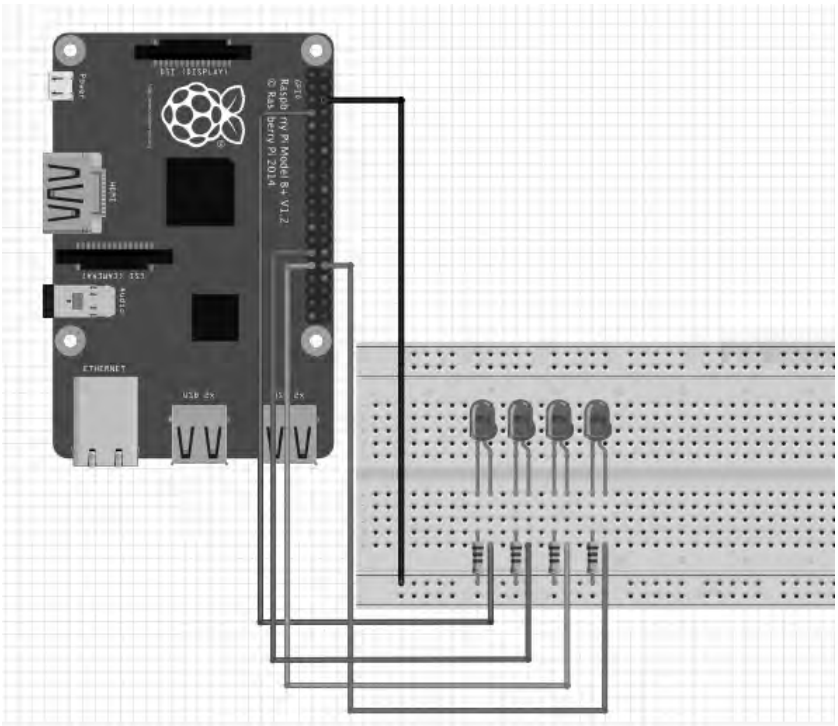

圖 11-7 硬體接線

說明

本專題指定 4 個 GPIO 接腳 GPIO4、5、6、12 為輸出，透過計時器定時每 0.5 秒呼叫一次，並且調整每一個接腳的電壓，這樣就能夠調整每一個 LED 燈。

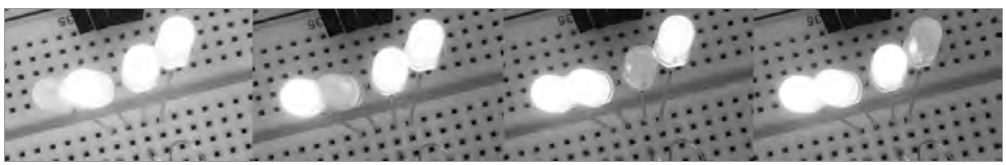

圖 11-8 調整 LED 燈光

實際範例

本程式執行後，就會把 GPIO5 接腳上的 LED 燈打開。

範例程式：ch011\11-3\MyIoTApp\MyIoTApp\MainPage.xaml.cs

```
1. ...                                      // using 定義，相同上一章所以省略
2. using Windows.Devices.Gpio;              // GPIO 函數 using 定義
3. namespace MyIoTApp
4. {
5.     public sealed partial class MainPage : Page
6.     {
7.         private GpioPin pin4;                // 接腳 4 變數
8.         private GpioPin pin5;                // 接腳 5 變數
9.         private GpioPin pin6;                // 接腳 6 變數
10.         private GpioPin pin12;               // 接腳 12 變數
11.         private DispatcherTimer timer1;      // 時間變數
12.         private int lightNo = 0;             // 霹靂燈狀態定義變數
13.         public MainPage()
14.         {
15.             this.InitializeComponent();      // 初始化元件
16.             GpioController gpio=GpioController.GetDefault();
                // 初始化 IoT 函數
17.             pin4 = gpio.OpenPin(4);          // 指定接腳 4
18.             pin5 = gpio.OpenPin(5);          // 指定接腳 5
19.             pin6 = gpio.OpenPin(6);          // 指定接腳 6
20.             pin12 = gpio.OpenPin(12);        // 指定接腳 12
21.             pin4.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 4 數位輸出
22.             pin5.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 5 數位輸出
23.             pin6.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 6 數位輸出
24.             pin12.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 12 數位輸出
25.
26.             timer1 = new DispatcherTimer();      // 計時器定義
27.            timer1.Interval = TimeSpan.FromMilliseconds(500);
28.            timer1.Tick += Timer_Tick;                    每 0.5 秒執行一次
29.             timer1.Start();                              // 計時器開始
                                  時間到的處理函數
30.         }
31.         private void Timer_Tick(object sender, object e)
            //定時呼叫的處理函數
32.         {
33.             if (lightNo == 0)                    // 霹靂燈狀態 0
34.             {
35.                 pin4.Write(GpioPinValue.Low);    // 設定接腳為低電位
36.                 pin5.Write(GpioPinValue.High);   // 設定接腳為高電位
37.                 pin6.Write(GpioPinValue.High)    // 設定接腳為高電位
38.                 pin12.Write(GpioPinValue.High); // 設定接腳為高電位
39.                 lightNo = 1;
40.             }
```

```
41.             else if (lightNo == 1)                // 霹靂燈狀態 1
42.             {
43.                 pin4.Write(GpioPinValue.High);    // 設定接腳為高電位
44.                 pin5.Write(GpioPinValue.Low);     // 設定接腳為低電位
45.                 pin6.Write(GpioPinValue.High);    // 設定接腳為高電位
46.                 pin12.Write(GpioPinValue.High);   // 設定接腳為高電位
47.                 lightNo = 2;                      // 下次霹靂燈狀態
48.             }
49.             else if (lightNo == 2)                // 霹靂燈狀態 2
50.             {
51.                 pin4.Write(GpioPinValue.High);    // 設定接腳為高電位
52.                 pin5.Write(GpioPinValue.High);    // 設定接腳為高電位
53.                 pin6.Write(GpioPinValue.Low);     // 設定接腳為低電位
54.                 pin12.Write(GpioPinValue.High);   // 設定接腳為高電位
55.                 lightNo = 3;                      // 下次霹靂燈狀態
56.             }
57.             else if (lightNo == 3)                // 霹靂燈狀態 3
58.             {
59.                 pin4.Write(GpioPinValue.High);    // 設定接腳為高電位
60.                 pin5.Write(GpioPinValue.High);    // 設定接腳為高電位
61.                 pin6.Write(GpioPinValue.High);    // 設定接腳為高電位
62.                 pin12.Write(GpioPinValue.Low);    // 設定接腳為低電位
63.                 lightNo = 0;                      // 下次霹靂燈狀態
64.             }
65.         }
66.         private void ClickMe_Click(object sender, RoutedEventArgs e)
            // 按鈕按下的反應
67.         {
68.             App.Current.Exit();                   // 結束程式
69.         }
70.
71.     }
72. }
```

程式說明

- 第 16-24 行：初始化 IoT 函數和設定 GPIO 接腳初始化動作。
- 第 26-29 行：設定計時器的時間。
- 第 31 行：計時器時間到時的處理函數。
- 第 35-39 行：設定每一個 LED 燈的狀態 0。
- 第 43-47 行：設定每一個 LED 燈的狀態 1。
- 第 51-55 行：設定每一個 LED 燈的狀態 2。
- 第 59-63 行：設定每一個 LED 燈的狀態 3。

執行結果

本範例執行之後的硬體結果如下圖所示。如果正確，執行程式後，因為接腳設定為高電位，就會將 LED 燈點亮。

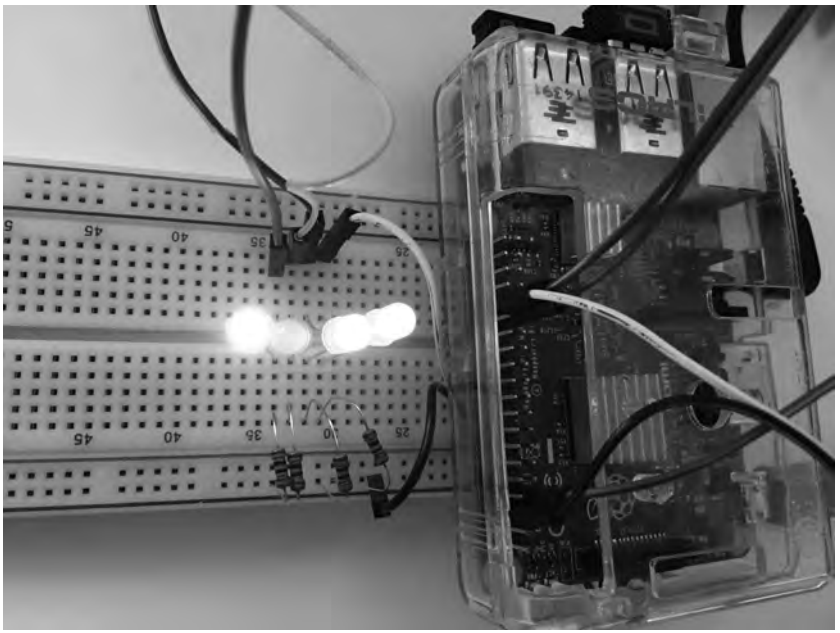

圖 11-9 執行結果

教學影片

完整的教學影片可以觀看 *11-3-gpio_Timer_LEDs*；硬體的執行情況請看影片 *11-3-gpio_Timer_LEDs-Hardware*，裡面都會有詳細的過程和教學；而 Demo 展示則請看 *11-3-gpio_Timer_LEDs-Hardware-Demo*。

延伸學習

霹靂燈的效果有很多種，請各位自行調整一下 GPIO，讓它變成不同效果顯示的霹靂燈，例如：全部的燈光只有一個會亮，並且從左到右依序做切換。

11.4 時間延遲的設計

介紹

在 Windows 10 IoT 程式中，對延遲的設計跟其他的程式不太一樣，以 Arduino 的程式來說，只要透過 delay(1); 就能夠停止一秒。但是 Windows 10 IoT 的 C# 語言，因為亦考量多工的情況，如要同時處理畫面的 UI 控制、時間的處理…等，不可能像 Arduino 的程式，停在 delay(1) 那一行就不用做其他事情。

所以在本次的實作中，將透過計時器，設定為 1 秒啟動一次，並且透過變數記錄時間，如果時間到了，就做出動作反應。

這裡將透過上一章的霹靂燈實驗，改變亮燈的時間，實作出延遲的效果。

硬體和接線

與上一章相同。

說明

本專題將指定 4 個 GPIO 接腳 GPIO4、5、6、12 為輸出，並透過計時器定時每 1 秒呼叫一次，並改變 Counter 變數。而霹靂燈有 4 個狀態，分別延遲 1 秒、2 秒、5 秒和 4 秒，請留意以下程式的延遲處理方法。

實際範例

本程式執行後，就會把 GPIO5 接腳上的 LED 燈打開。

範例程式：ch011\11-4\MyIoTApp\MyIoTApp\MainPage.xaml.cs

```
1. ...                                          // using 定義，同上一章所以省略
2. using Windows.Devices.Gpio;                  // GPIO 函數 using 定義
3. namespace MyIoTApp
4. {
5.     public sealed partial class MainPage : Page
6.     {
7.         private GpioPin pin4;                // 接腳 4 變數
8.         private GpioPin pin5;                // 接腳 5 變數
9.         private GpioPin pin6;                // 接腳 6 變數
10.         private GpioPin pin12;              // 接腳 12 變數
```

```
11.         private DispatcherTimer timer1; // 時間變數
12.         private int counter = 0;        // 現在時間的變數
13.         public MainPage()
14.         {
15.             this.InitializeComponent(); // 初始化元件
16.             GpioController gpio=GpioController.GetDefault();
                // 初始化 IoT 函數
17.             pin4 = gpio.OpenPin(4);    // 指定接腳 4
18.             pin5 = gpio.OpenPin(5);    // 指定接腳 5
19.             pin6 = gpio.OpenPin(6);    // 指定接腳 6
20.             pin12 = gpio.OpenPin(12);  // 指定接腳 12
21.             pin4.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 4 數位輸出
22.             pin5.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 5 數位輸出
23.             pin6.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 6 數位輸出
24.             pin12.SetDriveMode(GpioPinDriveMode.Output);
                // 接腳 12 數位輸出
25.
26.             timer1 = new DispatcherTimer();  // 計時器定義
27.             timer1.Interval = TimeSpan.FromMilliseconds(1000);
28.             timer1.Tick += Timer_Tick;       // 時間到的處理函數
29.             timer1.Start();                  // 計時器開始
30.             counter = 0;
31.         }
32.         private void Timer_Tick(object sender, object e)
            // 定時會呼叫的處理函數
33.         {
34.             if (counter == 0 || counter==0+1+2+5+4)
35.             {
36.                 pin4.Write(GpioPinValue.Low);   // 設定接腳為低電位
37.                 pin5.Write(GpioPinValue.High);  // 設定接腳為高電位
38.                 pin6.Write(GpioPinValue.High);  // 設定接腳為高電位
39.                 pin12.Write(GpioPinValue.High); // 設定接腳為高電位
40.                 counter = 0;
41.             }
42.             else if (counter == 0+1)
43.             {
44.                 pin4.Write(GpioPinValue.High);  // 設定接腳為高電位
45.                 pin5.Write(GpioPinValue.Low);   // 設定接腳為低電位
46.                 pin6.Write(GpioPinValue.High);  // 設定接腳為高電位
47.                 pin12.Write(GpioPinValue.High); // 設定接腳為高電位
48.             }
49.             else if (counter == 0+1+2)
50.             {
51.                 pin4.Write(GpioPinValue.High);  // 設定接腳為高電位
52.                 pin5.Write(GpioPinValue.High);  // 設定接腳為高電位
```

每 1 秒執行一次

狀態 0 顯示 1 秒

狀態 1 顯示 2 秒

狀態 2 顯示 5 秒

```
53.                 pin6.Write(GpioPinValue.Low);   // 設定接腳為低電位
54.                 pin12.Write(GpioPinValue.High); // 設定接腳為高電位
55.             }
56.           else if (counter == 0+1+2+5)
57.             {
58.                 pin4.Write(GpioPinValue.High);  // 設定接腳為高電位
59.                 pin5.Write(GpioPinValue.High);  // 設定接腳為高電位
60.                 pin6.Write(GpioPinValue.High);  // 設定接腳為高電位
61.                 pin12.Write(GpioPinValue.Low);  // 設定接腳為低電位
62.            }
63.             counter = counter + 1;
64.         }
65.         private void ClickMe_Click(object sender, RoutedEventArgs e)
           // 按鈕按下的反應
66.         {
67.             App.Current.Exit();                 // 結束程式
68.         }
69.
70.     }
71. }
```

狀態 3 顯示 4 秒

計時器時間的調整

程式說明

- 第 16-24 行：初始化 IoT 函數和設定 GPIO 接腳初始化動作。
- 第 26-29 行：設定計時器的時間。
- 第 30 行：設定計時器的時間。
- 第 31 行：計時器時間 Counter 初始化。
- 第 34-41 行：設定每一個 LED 燈的狀態 0。
- 第 42-48 行：設定每一個 LED 燈的狀態 1。
- 第 49-55 行：設定每一個 LED 燈的狀態 2。
- 第 56-62 行：設定每一個 LED 燈的狀態 3。

並且延遲的情況如下：

- 當 Counter 變數為 0，霹靂燈狀態 0，顯示 1 秒鐘。
- 當 Counter 變數為 1 (0+1)，霹靂燈狀態 1，顯示 2 秒鐘。
- 當 Counter 變數為 3 (0+1+2)，霹靂燈狀態 2，顯示 5 秒鐘。
- 當 Counter 變數為 8 (0+1+2+5)，霹靂燈狀態 3，顯示 4 秒鐘。

所以程式中會看到，counter 時間到了才去調整情況，且延遲的時間會寫在下一個狀態的 counter 變數判斷式之中。

注意 請留意最後一個狀態 3 的延遲 4 秒的時間，需要在狀態 0 判斷式之中。請看程式第 34 行：

```
if (counter == 0 ||  counter==0+1+2+5+4)
```

執行結果

本範例執行之後的硬體結果如下圖所示，如果正確執行程式，因透過計時器的關係，就能做出延遲的效果。

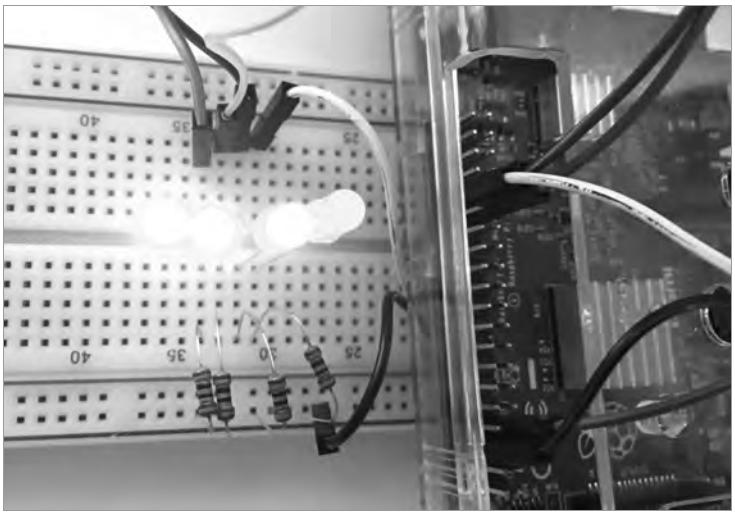

圖 11-10 執行結果

教學影片

完整的教學影片可以觀看 *11-4-gpio_Timer_delay*，裡面會有詳細的過程和教學；而 Demo 展示則請看 *11-4-gpio_Timer_delay-Hardware-Demo*。

延伸學習

因為 Windows 10 IoT 的延遲程式處理起來相當特別，請自行調整不同的時間，讓霹靂燈的效果改變，並且增加印象。