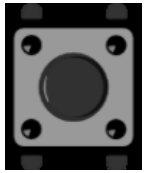
Chap 7
定時器 Timer

使用的 I/O 裝置



按鈕開關

火焰傳感器

光敏電阻

1602 I2C 液晶螢幕

7-1 定時器

定時器（Timer）中斷是一個非常實用的中斷機制，舉凡跟時間有關的控制動作幾乎都會使用到定時器，例如我們要寫出一個閃爍的LED燈，能固定的亮1秒鐘，暗1秒鐘，如此無限循環，這個程式相信對初學者而言一點都不困難，如範例7.1中的程式碼三二下就可輕鬆的解決，可是當我們要再加入其他的動作，例如在LED燈閃燈的同時要將數值1~100循環不停的輸出到PC的串列埠印出，如此一來範例7.1中的程式碼還適用嗎?。

【範例 7.1】

```
1  void setup()
2  {
3    pinMode(13, OUTPUT);           //設定數位接腳 13 為 OUTPUT 模式
4  }
5
6  void loop()
7  {
8    digitalWrite(13, HIGH);        //將 HIGH 寫到(輸出)接腳 13
9    delay(1000);                   //延遲 1000ms=1s
10   digitalWrite(13, LOW);         //將 LOW 寫到(輸出)接腳 13
11   delay(1000);                   //延遲 1000ms=1s
12 }
```

7-1 定時器

- 以單核單執行緒的Arduino而言，不管下面的範例怎麼修改，只要使用delay()函式一定寫不出邊閃爍邊印數值的動作，即使寫的出類似的動作也無法符合精確的時間要求，所以使用delay()函式來達到時間控制的目的似乎不是一個聰明的解決辦法，那要如何精確的控制時間呢？所幸定時器中斷提供了有效的解決方法。

【範例 7.1】

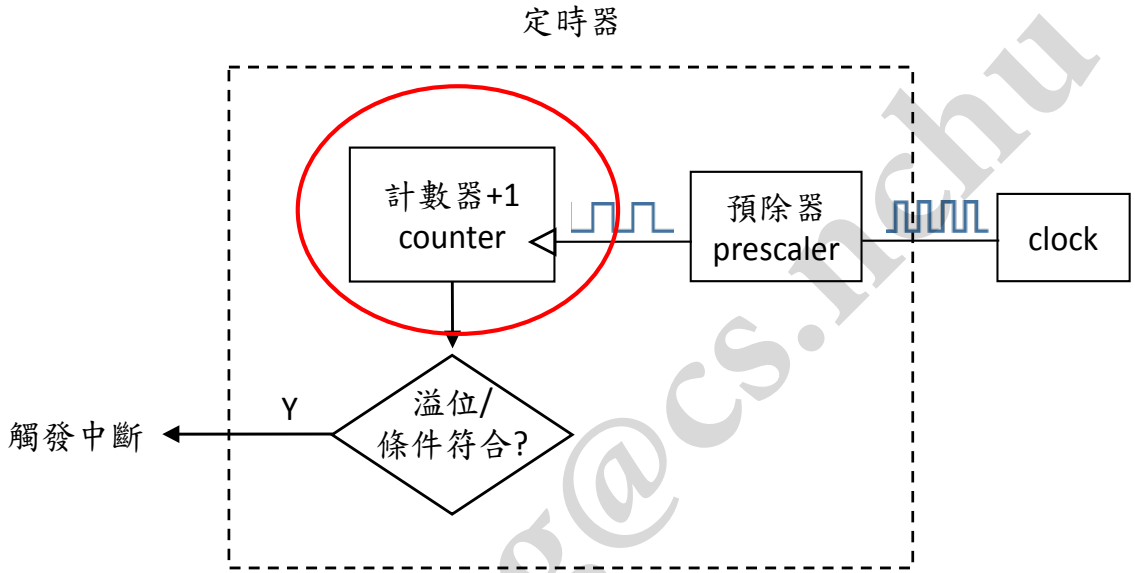
```
1  void setup()
2  {
3    pinMode(13, OUTPUT);           //設定數位接腳 13 為 OUTPUT 模式
4  }
5
6  void loop()
7  {
8    digitalWrite(13, HIGH);        //將 HIGH 寫到(輸出)接腳 13
9    delay(1000);                   //延遲 1000ms=1s
10   digitalWrite(13, LOW);         //將 LOW 寫到(輸出)接腳 13
11   delay(1000);                   //延遲 1000ms=1s
12 }
```

7-1 什麼是定時器?

- 一般日常生活中，我們都會使用定時器來量測某特定時間，提醒我們什麼時間該處理什麼事情，同樣的，在微控器中的定時器（Timer）也具有相同的概念跟用法，你可以把定時器想像成是MCU/MPU的鬧鐘，只要設定好一個時間點，等到時間一到就會開始響鈴提醒，當然不是真正的響起鬧鈴聲，對機器而言這會觸發一個中斷，只要我們把特定的動作或服務寫在中斷服務程式裡，如此一來便可達到在某特定時間執行某特定工作的目的。

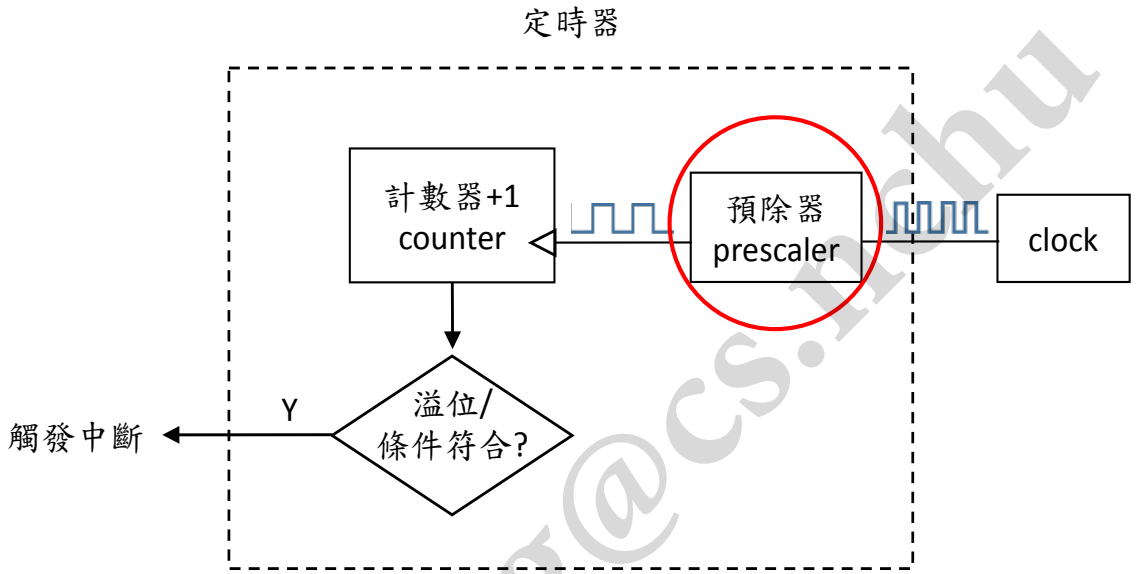
- 定時器的迷人之處在於它的運作是一種非同步的執行方式，與主程式之間是完全獨立互不相關的，當你的主程式在執行工作的時候，同一時間定時器可以在背景執行完成它既定的工作，完全不會干擾到前景主程式的執行，讓使用者感覺不到它的存在，比起使用迴圈或是時間函式來達到時間的控制，這種使用定時器的方式顯然是高端多了。

7-2 定時器的運作原理



- 定時器的核心單元包含一個計數暫存器（counter），而這個計數暫存器的內容值在每一次的時脈週期都會自動的加1，一直加到計數暫存器所能儲存的最大值，接著再加1就會發生溢位（overflow）的狀況，這時候除了計數暫存器的內容值會重置為0之外，定時器通常會設定旗標位元來表示溢位已經發生，當然除了溢位之外，定時器也可以設定其他觸發中斷的條件，例如：計數器的值達到我們指定的目標值的時候就觸發中斷。

7-2 定時器的運作原理



- 在定時器的運作中有一個非常關鍵的時間參數，那就是計數器要隔多久時間會執行加1的動作？通常這個時脈來源都是MCU上的系統時脈，一個脈波訊號就會觸發一次加1的動作。UNO晶片上16MHz的時脈頻率對定時器而言實在是太高了，為了解決這個問題，通常我們都需要把MCU上所提供的時脈頻率調降下來才能符合定時器的使用情境，這就是預先除頻電路prescaler的作用，簡稱「預除器」，說的白話一點，預除器的作用就是把定時器中計數加1的速度調慢下來。

7-3 UNO的定時器

表 7.1 Arduino UNO（ATmega328P）內含的 3 組定時器

定時器	計數器長度	計數值範圍	在 UNO 中預設的功能用途
Timer0	8-bit	0~255	(1) 使用在 Arduino 標準的時間函式 delay()、millis()、micros()，但是並不包括 delayMicroseconds()函式，因為它的時間計數與 Timer0 完全無關。為了避免干擾這些標準函式的正常功能，通常不建議修改 Timer0。 (2) 使用在類比接腳 pin 5、6 的 analogWrite()，實現 PWM 的輸出。
Timer1	16-bit	0~65535	(1) 使用在 Arduino 內建的伺服馬達函式庫 Servo Library。 (2) 使用在類比接腳 pin 9、10 的 analogWrite()，實現 PWM 的輸出。
Timer2	8-bit	0~255	(1) 使用在 Arduino 的聲波函式 tone()。 (2) 使用在類比接腳 pin 3、11 的 analogWrite()，實現 PWM 的輸出。

7-3 UNO的定時器中斷

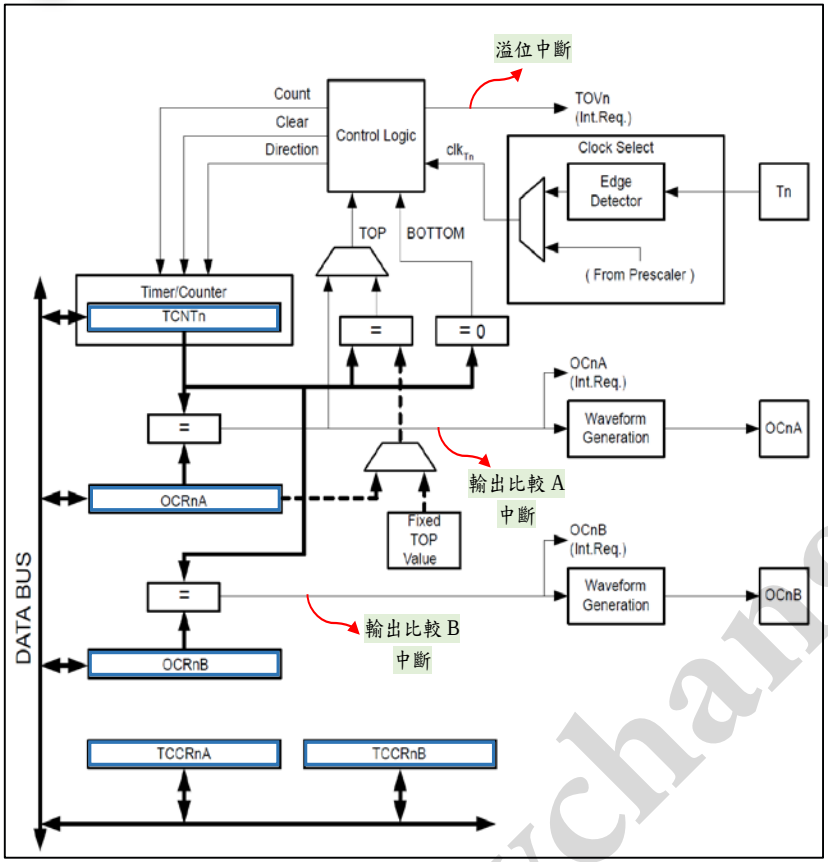
8	0x000E	TIMER2 COMPA (TIMER2_COMPA_vect)	Timer/Counter2 Compare Match A Timer2 定時器的比較相符中斷 A
9	0x0010	TIMER2 COMPB (TIMER2_COMPB_vect)	Timer/Counter2 Compare Match B Timer2 定時器的比較相符中斷 B
10	0x0012	TIMER2 OVF (TIMER2_OVF_vect)	Timer/Counter2 Overflow Timer2 定時器的溢位中斷
11	0x0014	TIMER1 CAPT (TIMER1_CAPT_vect)	Timer/Counter1 Capture Event Timer1 定時器的輸入比較相符中斷
12	0x0016	TIMER1 COMPA (TIMER1_COMPA_vect)	Timer/Counter1 Compare Match A Timer1 定時器的比較相符中斷 A
13	0x0018	TIMER1 COMPB (TIMER1_COMPB_vect)	Timer/Counter1 Compare Match B Timer1 定時器的比較相符中斷 B
14	0x001A	TIMER1 OVF (TIMER1_OVF_vect)	Timer/Counter1 Overflow Timer1 定時器的溢位中斷
15	0x001C	TIMER0 COMPA (TIMER0_COMPA_vect)	Timer/Counter0 Compare Match A Timer0 定時器的比較相符中斷 A
16	0x001E	TIMER0 COMPB (TIMER0_COMPB_vect)	Timer/Counter0 Compare Match B Timer0 定時器的比較相符中斷 B
17	0x0020	TIMER0 OVF (TIMER0_OVF_vect)	Timer/Counter0 Overflow Timer0 定時器的溢位中斷

Timer2

Timer1

Timer0

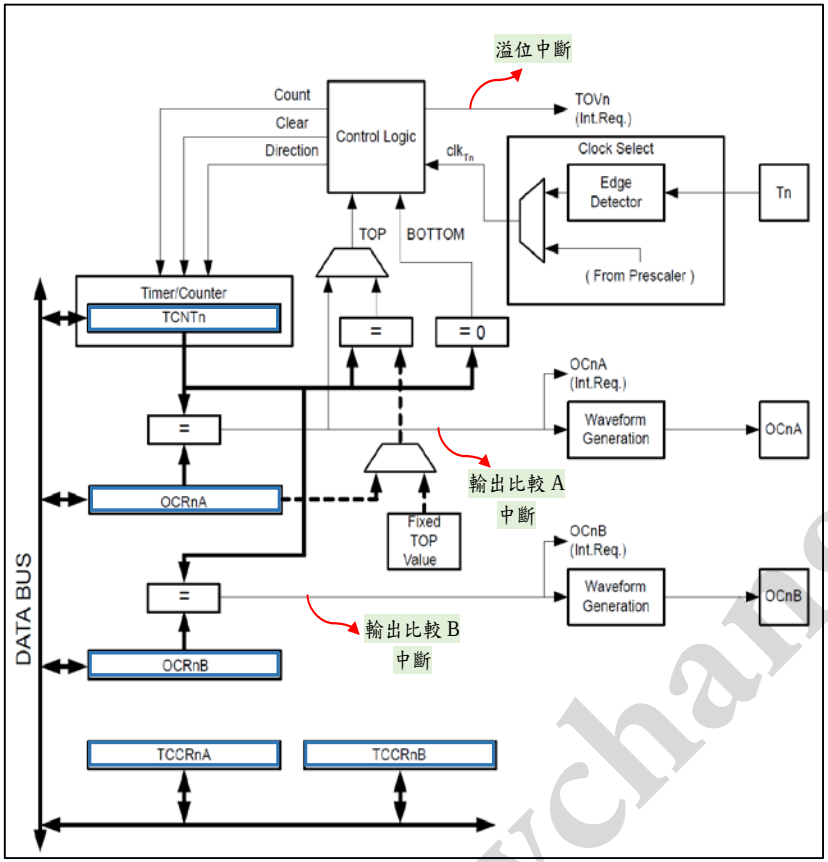
7-3-1 定時器相關暫存器



Arduino UNO中8位元定時器（Timer0 / Timer2）的電路圖

暫存器	說明
TCCRxA	【名稱】 Timer/Counter Control Register A，控制暫存器 A。 【功能】 設定定時器的運作模式，一般來說，如果只是單純的使用 Timer/Counter 功能，而不需要用到 PWM 輸出的話，只需把 TCCRxA 暫存器的內容值設定成 0x00 即可。
TCCRxB	【名稱】 Timer/Counter Control Register B，控制暫存器 B。 【功能】 主要是用來設定定時器的時脈來源以及預先除頻（預除器）prescaler 的倍數，配合時間的規畫 prescaler 共有 1/8、1/64、1/256 及 1/1024 四種有效的除頻倍數。
TCNTx	【名稱】 Timer/Counter Value Register，計數值暫存器。 【功能】 儲存定時器的計數值。
OCRxA	【名稱】 Output Compare Register A，輸出比較暫存器 A 【功能】 設定觸發中斷的目標值 A，當 TCNTx 的計數值等於 OCRxA 的內容值時就會觸發中斷。
OCRxB	【名稱】 Output Compare Register B，輸出比較暫存器 B 【功能】 設定觸發中斷的目標值 B，當 TCNTx 的計數值等於 OCRxB 的內容值時就會觸發中斷。

7-3-1 定時器相關暫存器



Arduino UNO中8位元定時器（Timer0 / Timer2）的電路圖

TIMSKx	【名稱】 Timer/Counter Interrupt Mask Register，中斷遮罩暫存器 【功能】 主要是用來啟用或停用定時器的中斷，包含輸出比較 A 中斷、輸出比較 B 中斷、以及溢位中斷，共三種中斷。
TIFRx	【名稱】 Timer/Counter Interrupt Flag Register，中斷旗標暫存器 【功能】 當定時器發生中斷的時候，會根據中斷的來源，將對應的旗標欄位設定為 1，一旦執行中斷之後，硬體就會自動將對應的旗標欄位清除為 0。
ICR	【名稱】 Input Capture Register，輸入暫存器 【功能】 只有 16 位元的定時器才有，可用來設定計數器的 TOP 值。

7-3-2 定時器 Timer0 的功能模式

表 7.3 Timer0 的功能模式

模式編號	WGM02	WGM01	WGM00	模式名稱	TOP 上限值	說明
0	0	0	0	Normal	0xFF	定時器 Normal 模式，在計數值溢位時發出溢位中斷。
1	0	0	1	PWM, phase correct	0xFF	
2	0	1	0	CTC	OCRA	定時器 CTC 模式，在計數值等於輸出比較暫存器 OCRA 的內容值時就會觸發中斷。
3	0	1	1	Fast PWM	0xFF	
4	1	0	0	-	-	保留沒有使用
5	1	0	1	PWM, phase correct	OCRA	
6	1	1	0	-	-	保留沒有使用
7	1	1	1	Fast PWM	OCRA	

7-3-2 定時器 Timer0 的功能模式

Bit	7	6	5	4	3	2	1	0
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

控制暫存器 A（TCCR0A/ TCCR2A）

Bit	7	6	5	4	3	2	1	0
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

控制暫存器 B（TCCR0B/ TCCR2B）

圖 7.3 8 位元定時器（Timer0 / Timer2）的控制暫存器

Bit	7	6	5	4	3	2	1	0
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

控制暫存器 A（TCCR1A）

Bit	7	6	5	4	3	2	1	0
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

控制暫存器 B（TCCR1B）

圖 7.3 16 位元定時器（Timer1）的控制暫存器

7-3-2 定時器 Timer1 的功能模式

表 7.3 Timer1 的功能模式

模式編號	WGM13	WGM12	WGM11	WGM10	模式名稱	TOP 上限值	OCR1x 內容更新點	TOV1 旗標設定點
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

7-3-3 預先除頻與時間的計算

- 在先前我們已經討論過，16MHz的頻率（也就是1秒有16x10^{6}個脈波）對定時器而言實在是太高了。Arduino UNO （ATmega328P）中可使用的除頻倍數如表7.x所示，有8倍、64倍、256倍、1024倍等4種，只要透過控制暫存器TCCRxB中CS12，CS11，CS10三個位元的設定，就可指定定時器的時脈來源及其除頻倍數。

表 7.x 定時器時脈的來源與前置除頻 prescaler 的設定

CS12	CS11	CS10	除頻倍數	說明
0	0	0		沒有時脈來源（停用定時器）
0	0	1	$clk_{I/O}/1$	時脈是來自 prescaler 電路除頻 1 倍的結果，沒有除頻的效果
0	1	0	$clk_{I/O}/8$	時脈是來自 prescaler 電路除頻 8 倍的結果，16 MHz/8=2 MHz
0	1	1	$clk_{I/O}/64$	時脈是來自 prescaler 電路除頻 64 倍的結果，16 MHz/64=250 KHz
1	0	0	$clk_{I/O}/256$	時脈是來自 prescaler 電路除頻 256 倍的結果，16 MHz/256=62.5 KHz
1	0	1	$clk_{I/O}/1024$	時脈是來自 prescaler 電路除頻 1024 倍的結果，16 MHz/1024=15.625 KHz
1	1	0		時脈是來自 T1 接腳的外部時脈來源，屬負緣（下降邊緣）觸發
1	1	1		時脈是來自 T1 接腳的外部時脈來源，屬正緣（下降邊緣）觸發

7-3-3 預先除頻與時間的計算

上限值 TOP 的計算

接下來，「多久時間產生一次中斷？」這個問題是程式設計者在使用定時器之前必須要決定的，有了這個時間條件，我們就可以計算出定時器中計數器要累計的上限值，也就是 TOP 值。假設我們要每 T 秒產生一次中斷，由 T 與 TOP 的關係可推得下列的計算方程式：

$$T = TOP \times Timer\ 的時脈週期$$

$$TOP = T \times Timer\ 的時脈頻率$$

$$TOP = T \times \frac{16M}{除頻倍數}$$

舉例而言，如果要使用 64 倍的預先除頻倍數，讓定時器固定每 1 毫秒（1 ms=0.001 s）產生 1 次中斷，則根據以上 TOP 值的計算公式可得：

$$TOP = 0.001 \times \frac{16M}{64}$$

$$TOP = 250$$

7-3-3 預先除頻與時間的計算

根據 TOP 值的計算公式，我們將 4 種有效的除頻倍數與 4 個常用的中斷時間，總共 16 種組合的 TOP 值列出在表 7.x 中，其中有小數的部分可以直接捨去，因為 8 位元的定時器（Timer0，Timer2）最大的計數值只到 255，所以只有標註*的組合可以使用；而 16 位元的定時器（Timer1）最大的計數值可以到 65535，因此可以使用的組合較多，如表中的灰色組合皆可使用。

表 7.x 各種除頻倍數與中斷時間組合的 TOP 上限值

	1 秒	0.1 秒	0.01 秒	0.001 秒
1024	15625	1562.5	156.25 *	15.625 *
256	62500	6250	625	62.5 *
64	250000	25000	2500	250 *
8	2000000	200000	20000	2000

注意： 1. *的組合只適用在 8 位元的定時器（Timer0，Timer2）

2. 灰色的組合適用在 16 位元的定時器（Timer1）

7-3-3 預先除頻與時間的計算

表 7.x 雖然列出了 4 個時間，但或許有人會問，如果中斷時間要 5 秒一次，甚至是 10 秒一次，那 TOP 值不就超過 65535 無法設定了？的確，像這種狀況確實無法一次設定到位，但是不要忘記時間是可以累計的，延續上一個 TOP=250 的範例，有了每 1 毫秒產生 1 次中斷的定時器之後，我們就可以利用這個 1 毫秒（0.001 秒）的定時器來完成任何定時執行的工作。例如：1 毫秒累積 500 次之後就等同 500 毫秒的效果，因此範例中第 7 行的敘述就是每 0.5 秒執行一次 Job_A，第 8 行的敘述就是每 1 秒執行一次 Job_B，第 9 行的敘述就是每 8 秒執行一次 Job_C。

```
1  /*****************************************************
2   * Timer0 的 COMPA 中斷服務程式，每 0.001 秒產生一次中斷
3   * ISR 函式名稱固定為 ISR(TIMER0_COMPA_vect)，不可修改
4   *****************************************************/
5  ISR(TIMER0_COMPA_vect)
6  {
7     cnt_A++; cnt_B++; cnt_C++;
8     if(cnt_A == 500) { cnt_A=0; Job_A( ); }  //每 0.5 秒就執行一次 Job_A
9     if(cnt_B == 1000){ cnt_B=0; Job_B( ); }  //每  1 秒就執行一次 Job_B
10    if(cnt_C == 8000){ cnt_C=0; Job_C( ); }  //每  8 秒就執行一次 Job_C
   }
```

7-3-3 預先除頻與時間的計算

```
1  /*****************************************************
2   * Timer0 的 COMPA 中斷服務程式，每 0.001 秒產生一次中斷
3   *****************************************************/
4  ISR(TIMER0_COMPA_vect)
5  {
6     cnt_A++; cnt_B++; cnt_C++;
7     if(cnt_A == 500) { cnt_A=0; flag_A=1; }
8     if(cnt_B == 1000){ cnt_B=0; flag_B=1; }
9     if(cnt_C == 8000){ cnt_C=0; flag_C=1; }
10 }
11
12 /*****************************************************
13  * loop
14  *****************************************************/
15 loop()
16 {
17    if(flag_A == 1) { Job_A(); flag_A=0; }  //每 0.5 秒就執行一次 Job_A
18    if(flag_B == 1) { Job_B(); flag_B=0; }  //每  1 秒就執行一次 Job_B
19    if(flag_C == 1) { Job_C(); flag_C=0; }  //每  8 秒就執行一次 Job_C
20 }
```

7-3-4 定時器的使用步驟

接下來我們將定時器的使用分解成固定的 4 個步驟，使用者只要依照下列的步驟進行設定就可以完成定時器的規劃跟使用。

Step 1. 決定使用哪一組定時器，並且進行相關暫存器的初始化

Step 2. 設定正確的模式

Step 3. 根據中斷時間，設定預先除頻倍數 prescaler 與正確的 TOP 上限值

Step 4. 致能對應的中斷

例如：程式要固定每 1 秒產生一次中斷。

(1) Step 1：決定使用哪一組定時器，並且進行初始化。

```
TCCR1A=0;
TCCR1B=0;
TCNT1=0;
```

因為中斷時間為 1 秒算是一個很大的時間單位，以 Arduino UNO 而言，只有 Timer1 有足夠大的計數器可以設定，所以我們選用 Timer1，記得在使用前先將 Timer1 的控制暫存器 A、B 與計數器皆初始化為 0。

7-3-4 定時器的使用步驟

(2) Step 2：設定定時器的模式。

```
TCCR1B = TCCR1B | (1<<3);      // 設定成 CTC 模式
```

或 `TCCR1B = TCCR1B | (1<<WGM12); // WGM12=3 定義在 time.h 中`

如果不是特殊的應用，一般而言我們都會將定時器設定成 CTC 模式，也就是在定時器的計數值累計到我們設定的上限值時，就會觸發中斷，同時將計數暫存器 TCNT 的值歸 0 重新計數。參考表 7.x，Timer1 的 CTC 模式有二種，我們要選用將 TOP 上限值存放在輸出比較暫存器 OCR1A 的模式，也就是編號 4 的模式，所以將 1 左移 3 個位元，(1<<3)，但為了增加程式碼的可讀性，我們將其改寫為(1<<WGM12)，其中 WGM12 定義在標頭檔 timer.h 中。

7-3-4 定時器的使用步驟

(3) Step 3：根據中斷時間，設定預先除頻倍數 prescaler 與正確的 TOP 上限值。

```
TCCR1B = TCCR1B | (1<<2);     //設定 prescaler=256
```

或

```
TCCR1B = TCCR1B | (1<<CS12); //CS12=2 定義在 time.h 中
OCR1A  = 62500;              //設定 TOP=62500
```

因為中斷時間為 1 秒，採用 256 倍的預先除頻倍數，根據公式：

$$\text{TOP} = \text{T} \times \frac{16\text{M}}{\text{除頻倍數}}$$

$$\text{TOP} = 1 \times \frac{16\text{M}}{256}$$

$$\text{TOP} = 62500$$

7-3-4 定時器的使用步驟

（4）Step 4：致能對應的中斷。

```
TIMSK1 = TIMSK1 | (1<<1);        //致能輸出比較中斷 A
或 TIMSK1 = TIMSK1 | (1<<OCIE1A); //OCIE1A=1 定義在 time.h 中
```

最後一步要記得打開（致能）輸出比較相符的中斷，因為我們採用 Timer1 的 CTC 模式是將上限值存放在 OCR1A 中，所以在計數器 TCNT1 累計等於 OCR1A 時就會滿足條件，這時候只有輸出比較 A 的中斷致能旗標（OCIE1A）為 1，才會觸發中斷，進一步的叫用中斷服務函式 ISR(TIMER1_COMPA_vect)；否則，在 OCIE1A 位元為 0 的情況下，雖然滿足了 TCNT1=OCR1A 的條件，但還是不會產生輸出比較相符的中斷。

範例 7-1

「試寫出一個閃爍的 LED 燈，能固定的亮 1 秒鐘，暗 1 秒鐘，並且在 LED 閃燈的同時將數值 1~100 循環不停的輸出到 PC 的串列埠印出。」

```
#define LED_PIN 13

/*****************************************************
 * Timer1的COMPA中斷服務程式，每1秒產生一次中斷
 *****************************************************/
ISR(TIMER1_COMPA_vect)
{
   digitalWrite(LED_PIN,!digitalRead(LED_PIN));  //反轉LED現在的狀態
}

/*****************************************************
 * 設定定時器
 *****************************************************/
void SetupTimer(void)
{
   //---step1:初始暫存器
   TCCR1A=0;
   TCCR1B=0;
   TCNT1=0;

   //---step2:設定CTC模式
   TCCR1B = TCCR1B | (1<<WGM12);  //將1左移3個位元，進行位元OR邏輯運算

   //---step3:設定預先除頻倍數prescaler與正確的TOP上限值
   //在prescaler=256，TOP=62500的設定下，會固定每一秒產生一次中斷
   TCCR1B = TCCR1B | (1<<CS12);    //prescaler=256
   OCR1A  = 62500;                 //設定TOP=62500

   //---step4:致能對應的中斷
   TIMSK1 = TIMSK1 | (1<<OCIE1A); //致能輸出比較中斷A
}
```

```
/*****************************************************
 * setup
 *****************************************************/
void setup()
{
   Serial.begin(9600);
   pinMode(LED_PIN,OUTPUT);
   digitalWrite(LED_PIN, HIGH);
   SetupTimer();  //設定定時器
}

/*****************************************************
 * loop
 *****************************************************/
void loop()
{
int i;
   for(i=0;i<=100;i++) Serial.println(i);
}
```

範例 7-2: 1602 I2C 液晶螢幕

模組特點：

1、通訊介面：I2C

2、I2C地址 ：0x27

3、接腳定義：VCC、GND、SDA、SCL

4、工作電壓：+5V

5、尺寸 16x2

6、螢幕顯示對比度可調

7、藍底白字

- 程式碼

範例 7-2: 1602 I2C 液晶螢幕

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Set the LCD address to 0x27 for a 16 chars and 2 line display
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
    // initialize the LCD
    lcd.begin();

    // Turn on the blacklight and print a message.
    lcd.backlight();
    lcd.print("Hello, world!");
}

void loop()
{
    // Do nothing here...
}
```