嵌入式系統

單元四：即時作業系統基本觀念(2)

授課教師：雲林科技大學 張慶龍 老師

單元學習目標與大綱

- 多工程式之溝通與同步
- Critical Section及其保護
- Timer/Tick Clock

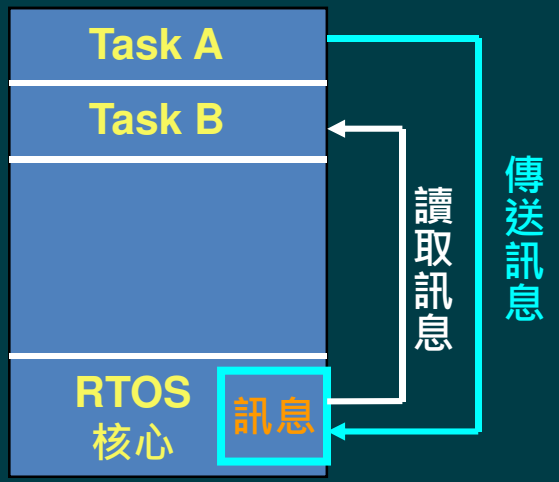
Part 1
多工程式之溝通與同步

Task溝通 (Communication)

- 多工程式(multitask programming)
  - 一個系統功能由多個Tasks分工合作來完成
  - 每個Task為一獨立程式，各有自己專屬的stack
    - 彼此看不到對方的區域變數
  - 為了完成系統工能，Task在執行過程，通常需與其它Task溝通
- 在RTOS環境下，Task間溝通方法
  - 訊息傳遞(message passing)
  - 共享記憶體(shared memory)

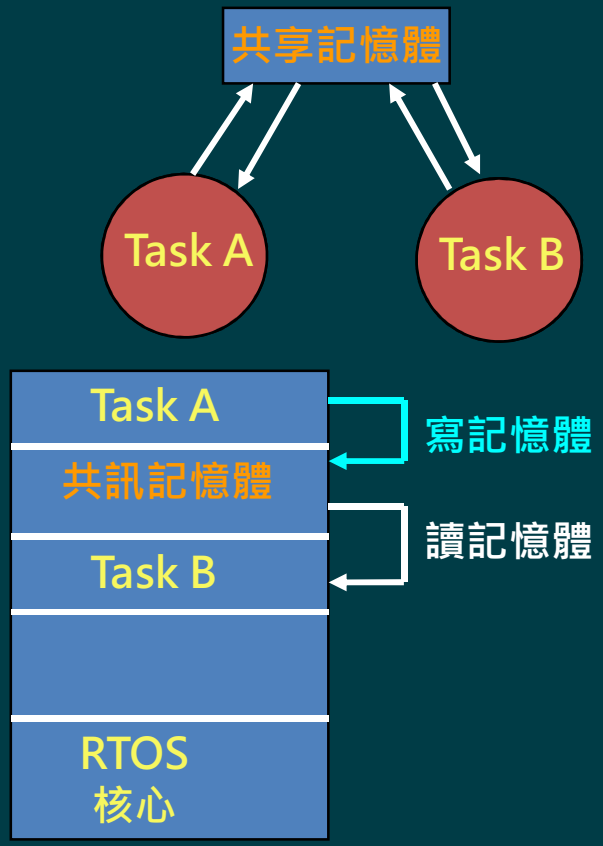
訊息傳遞

- Task間要先建立訊息通道
- 依賴RTOS維護此訊息通道
- 訊息內容是雙方Task瞭解即可
- Task B讀不到訊息時會進入 Waiting state (context switch)



共享記憶體

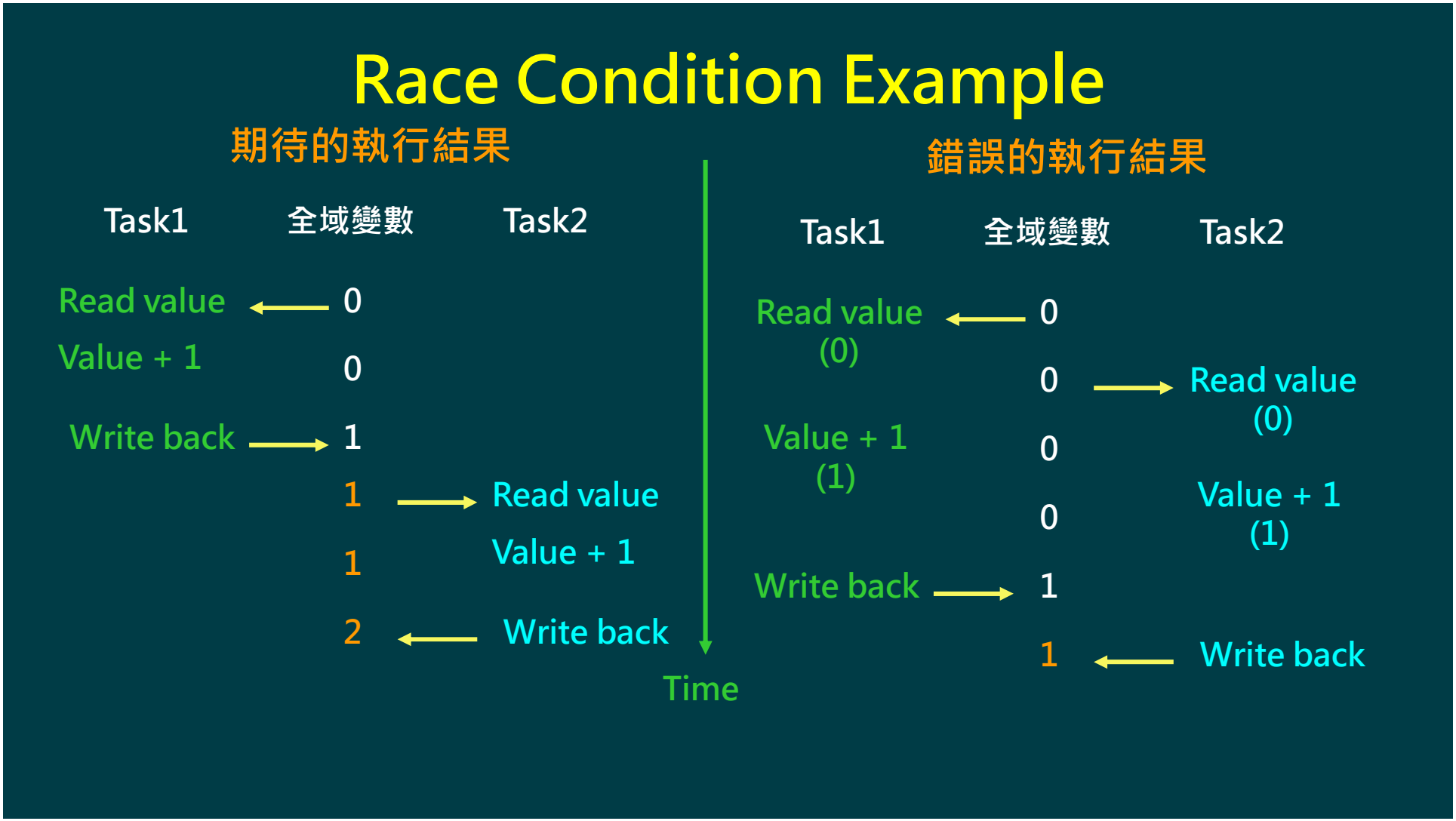
- 宣告全域變數(共享記憶體)，藉由變數值的改變達到溝通目的
- 所有的Tasks皆可對此全域變數存/取
- 無需RTOS核心管理
- 需對共享記憶體做互斥存取(mutual exclusive)
  - 否則會有Race condition，造成執行結果不正確



Software Race Condition

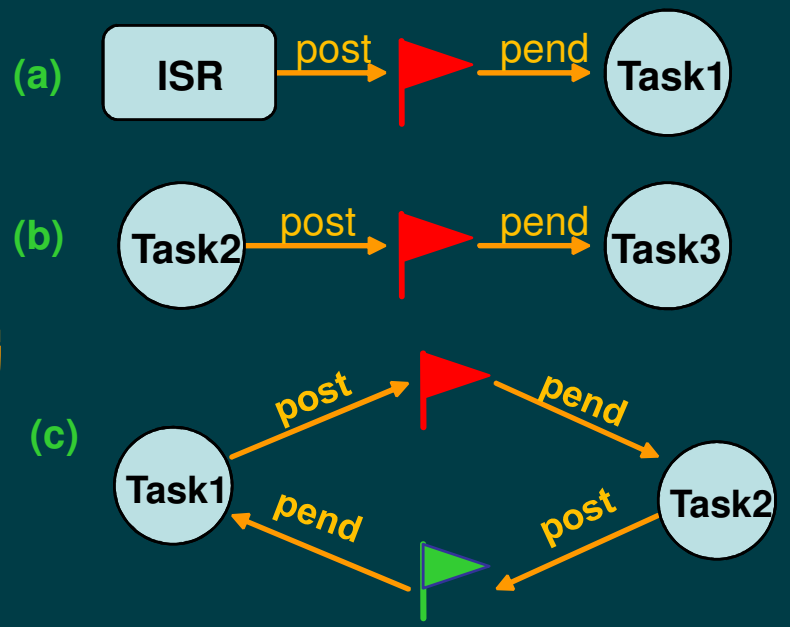
- 執行結果的正確性是依固定的Tasks執行順序而定
  - Tasks執行順序改變，執行之結果會受影響
  - 通常是多個Tasks會去存取相同的全域變數(global variable)
- Race condition為一動態bug
  - 很難被複製與除錯(debug)
  - 將於Tasks內加入log功能，race condition bug可能就不會出現
- 撰寫多工程式時，即要避免有race condition的狀況

Race Condition Example



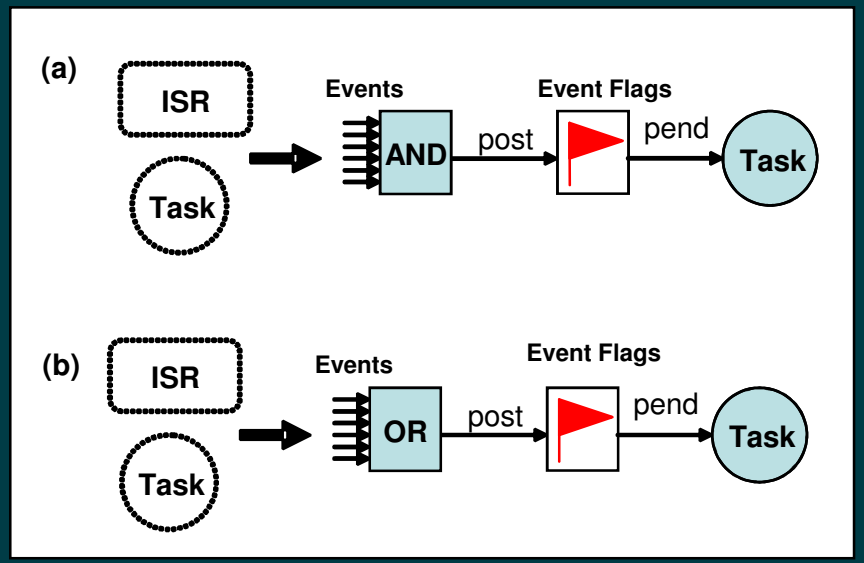
Task間同步(synchronization) (1)

- Semaphore(令牌)
  - 單一事件觸發的同步
    - ISR程式觸發Task
    - Task觸發Task
  - 不可由ISR觸發ISR與Task觸發ISR
    - 會造成ISR程式執行時間不確定➔影響系統即時性能力



Task間同步(synchronization) (2)

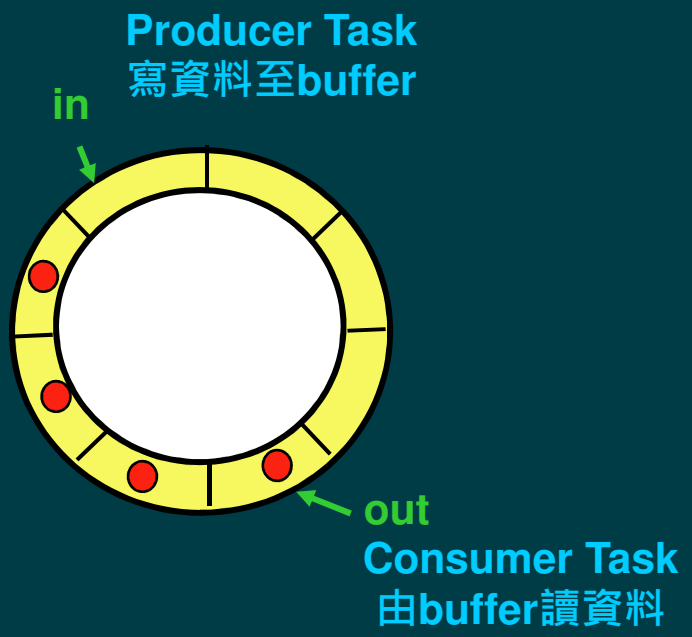
- **Event Flags**
  - 多事件觸發同步
    - 多事件皆發生時觸發Task執行 (AND事件)
    - 多事件中任一事件發生觸發Task執行(OR事件)



Part 2
Critical Section

Task 同步需求範例-
Producer-consumer problem

- Producer: 產生資訊供其它Task使用
- Consumer: 使用Producer所產生之資訊
- Circular buffer
  - in指標:指向下一個空的buffer空間位址
  - out指標:指向下一個要讀取資料之buffer空間位址
- Circular buffer 為共享記憶體(critical section)
  - 需要保護以避免 race condition
  - 當buffer空了，Consumer需等待
  - 當buffer滿了，Producer需等待



Critical Section

- 臨界區域(Critical Section)之定義：Tasks間共享之資源(resources)稱為臨界區域
  - 全域變數 (global variable)
  - 不可重入函數 (non reentrant function)
  - 周邊裝置(ex: UART、Printer)
- 多工環境，Task執行過程，隨時可能被其它task插斷
  - Critical section在執行過程被其它Task插斷，可能造成執行結果錯誤

```
task1()
{
  ....
  printf("This is task1");
  ...
 }

task2()
 {
    ...
    printf("This is task2");
    ...
 }
```

Results may be:

ThiThsi siis ttasaks k12

Critical Section設計需滿足三個性質

- Mutual exclusion(互斥)
- Progress (no deadlock)
  - 沒有在critical section內的tasks不能影響別的task進入critical section
- Bound waiting (no starvation)
  - 提出進入critical section的等待時間是有限的

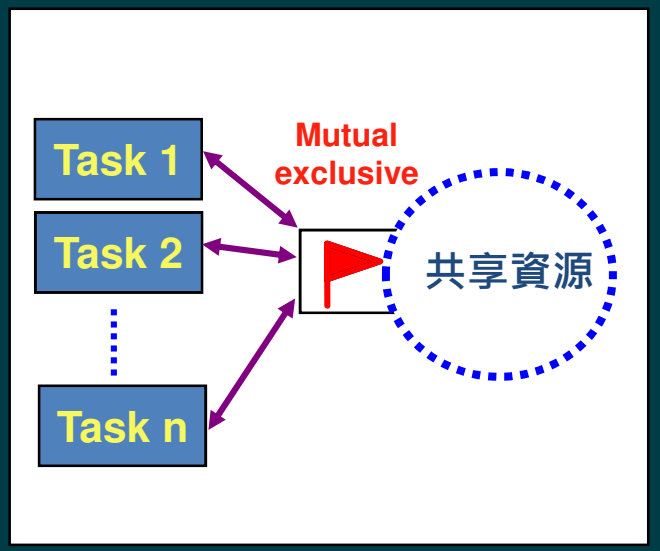
Critical Section 保護-mutual exclusive

- 確保在critical section內執行的Task，不被其它Task插斷(preemptive)
  - 即不會做context switch
- 要確保mutual exclusive之方法為
  - 進入臨界區前執行entry code；離開臨界區時執行exit code
- Entry/exit code主要可以是
  - disable 中斷(interrupt)
  - disable 排程(scheduling)
  - 使用Semaphore(令牌)

```
task()
{ ....
  while(1)
    {
       ...
       entry code
                  shared resources
                   (critical section)
       exit code
          remainder section
     }
}
```

Mutual Exclusive方法

- Entry/exit code主要可以是
  - disable 中斷(interrupt)
    - 不會有新事件產生➔不會新的Task ready
  - disable 排程(scheduling)
    - 系統不會做排程➔不會做context switch
  - 使用Semaphore(令牌)
    - 只有一個令牌
    - Entry code會試著拿令牌
      - 拿到令牌才能進入critical section
      - 拿不到令牌即進入等待(context switch
    - Exit code歸還令牌



Part 3
Timer/Tick Clock

RTOS為何需要Timer?

```
void main() {
        do forever{
            check keypad;
            measure temperature;
            control oven power;
            decrement timer;
            update display;
            wait for a fixed time;
    }
}
```

RTOS為何需要Timer?

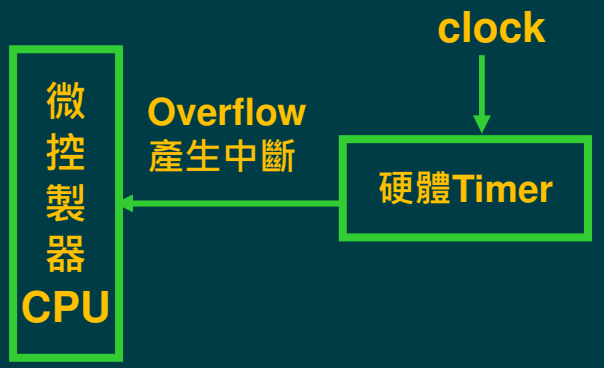
- Round-Robin排程器，每個Task執行一固定slice time即context switch，換下一個Task執行
- Task要周期性的polling周邊裝置之狀況
- Task在設定時間內，等待某事件發生，若時間已到，事件尚未發生，則Task將不繼續等待

CPU如何得知經過多少時間? (1)

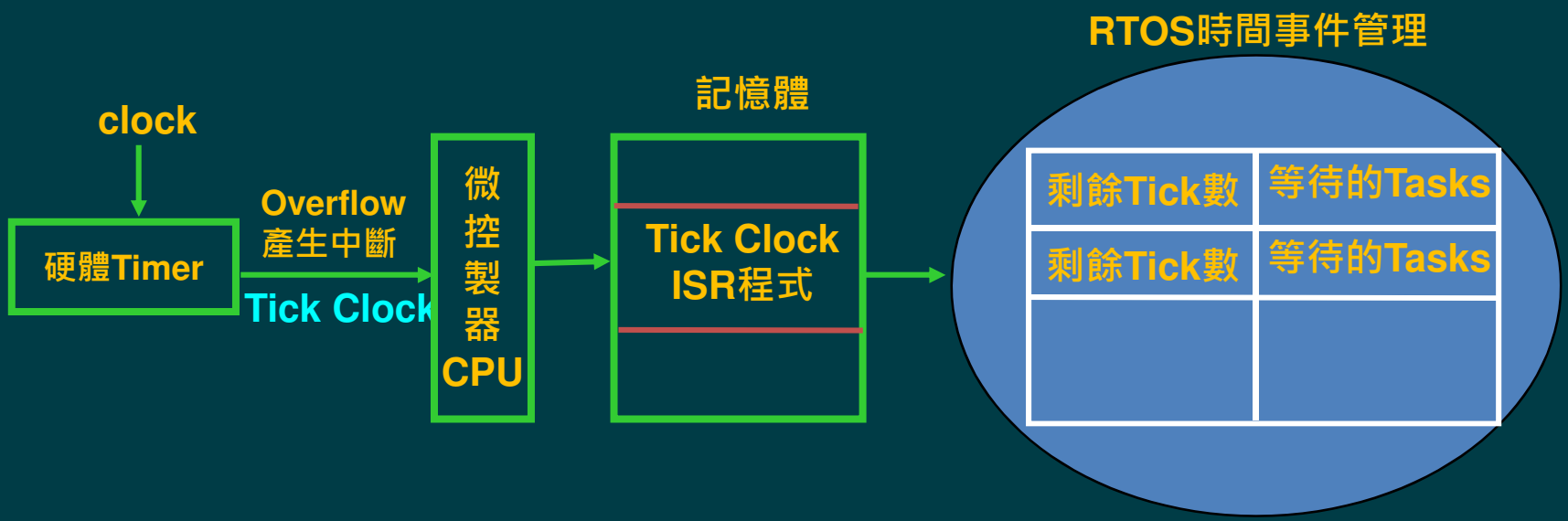
- 利用軟體迴圈
  - 計算迴圈所執行的指令數，換算為指令週期數 n
  - 在工作頻率固定下，指令週期時間為固定 t
  - 得到執行迴圈所需時間 $T = n \times t$
- 會浪費CPU Time
- 通常應用於8-bit CPU之較簡單的嵌入式系統環境

CPU如何得知經過多少時間? (2)

- 利用外部硬體計時器(Timer)
  - CPU可設定硬體Timer之計時時間
  - 當計時時間到達時(overflow)，硬體Timer會對CPU發出中斷訊號
  - CPU藉由中斷訊號得知所經過的時間
- 不會浪費CPU時間
- RTOS通常使用此架構管理內部之時間事件



RTOS時間事件管理(1)



RTOS時間事件管理(2)

- 硬體Timer中斷對CPU而言，稱為Tick Clock
- Tick Clock為RTOS內最小的時間單位
- RTOS依據Tick Clock來管理內部之時間事件

– 如圖，T1, T4 Task需再等待2個Tick Clocks
– T2, T3 Task需再等待 2+3 = 5個Tick Clocks
– 每次Tick Clock中斷，Tick ISR程式只需將時間事件管理表內的第一列的剩餘Tick數減一
– 若剩餘Tick數為0，代表時間事件已到期

時間事件管理表

剩餘Tick數	等待的Tasks
2	T1, T4
3	T2, T3

Clock Tick

- 為周期性的中斷訊號
- 提供RTOS最基本的時間單位
  - 提供Task使用之delay function
- 避免Task無限期等待事件的發生
  - 提供等待事件的Timeout時間
- Clock tick愈小，系統負擔愈大
- Clock tick愈大，時間解析度愈差

Clock Tick範例