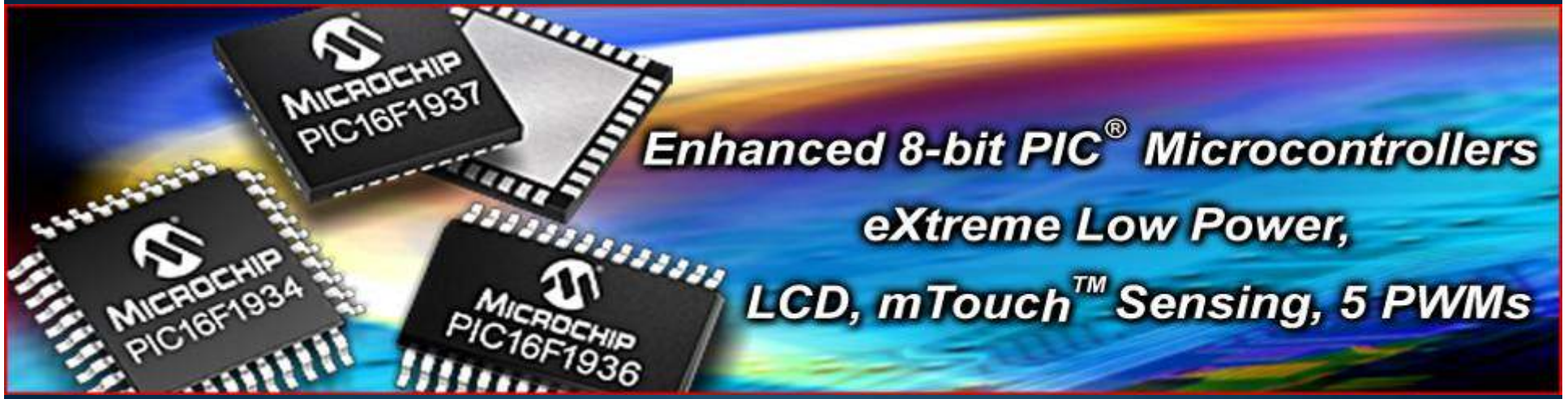


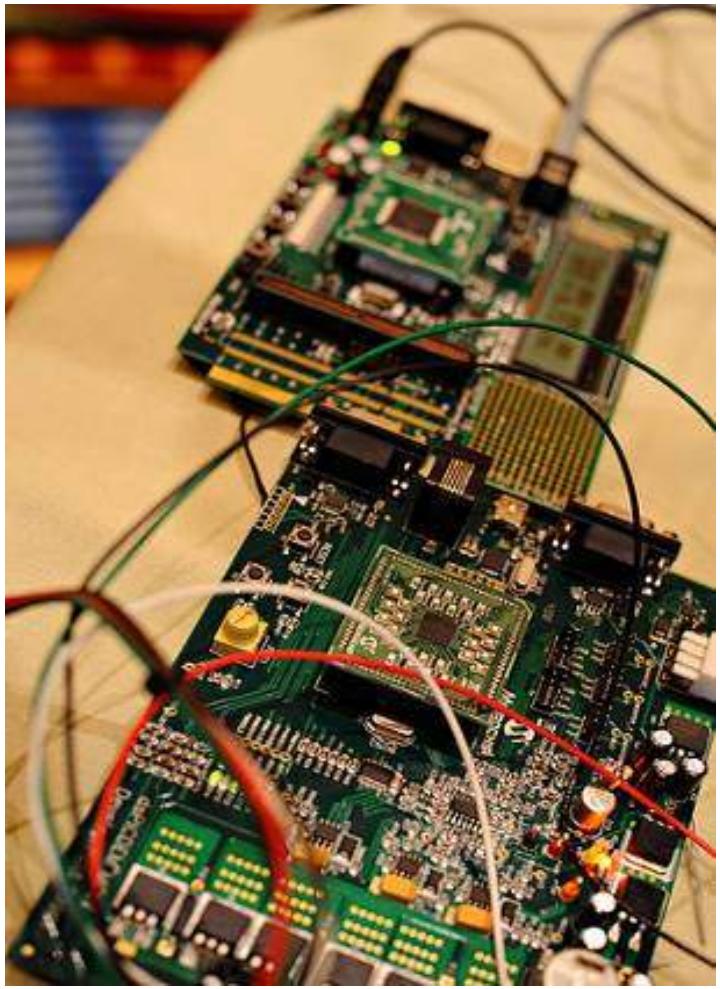
W301 Advance PICC Application

教育訓練課程

Enhanced PIC16F1xxx 介紹

What is important when designing with an 8-bit MCU?

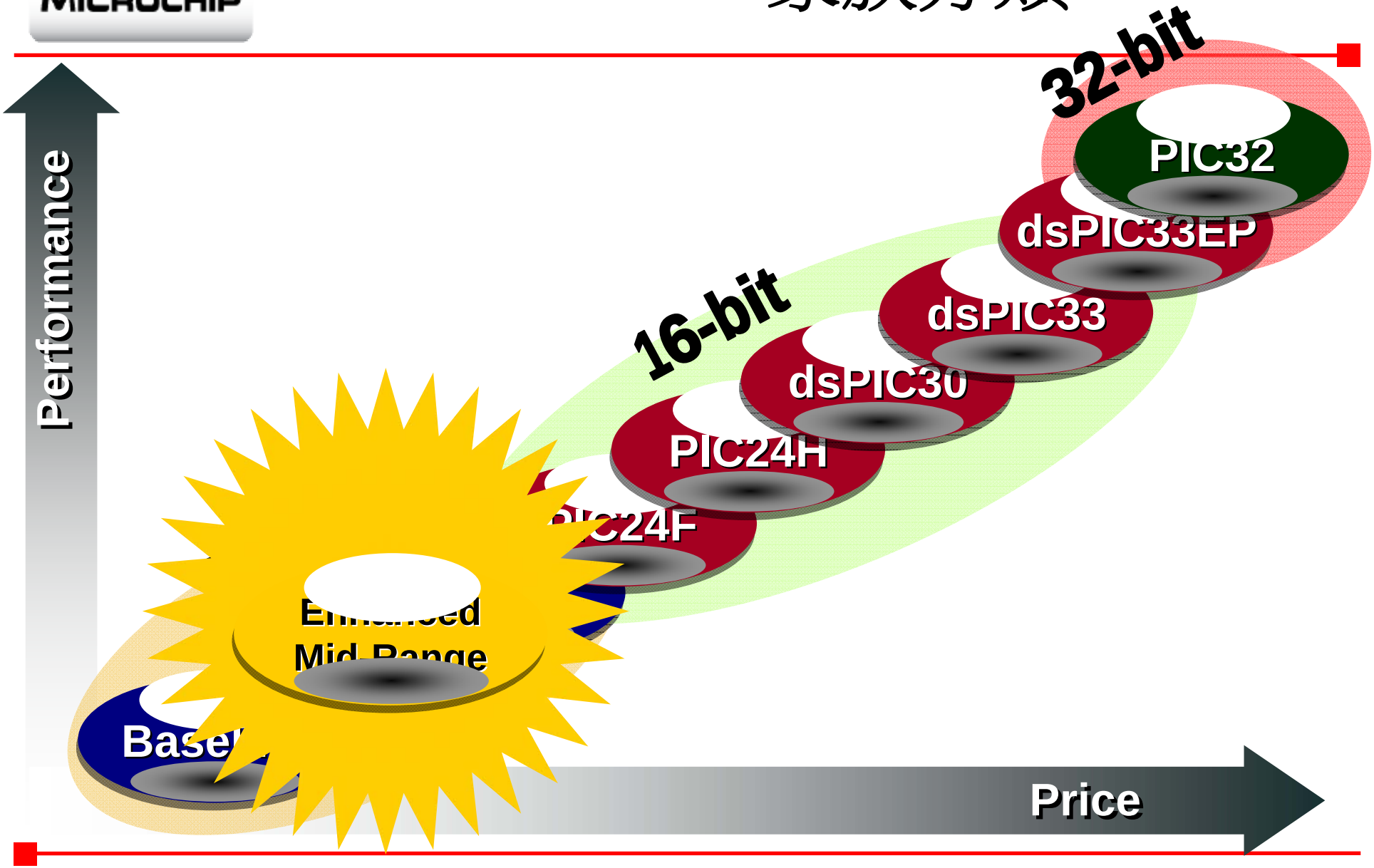
- 使用 C 語言撰寫
  - 開發更複雜的應用
  - 指令的編譯效率
- 低功耗支援
  - 提高電池的使用壽命
  - 符合綠色環保需求
- 性能
  - 執行效能與功耗關係
  - 增加執行效能 (MIPS)
  - 最小的中斷響應時間
  - 採最佳化編譯效果減少指令執行時間
- 延伸與擴充
  - 升級及轉移的便利性
- 正確的產品選擇
  - 眾多功能的整合與成本
  - 周邊的整合
  - 低單價、功能強大的開發工具



PIC16F1xxxx 介紹

- Overview
- Memory Map
- New Instructions
- Enhanced Indirect Addressing Features

PIC® 家族分類


32-bit
PIC32
dsPIC33EP
16-bit
dsPIC33
dsPIC30
PIC24H
Performance
Price

Enhanced PIC16 的目標

- 增加程式記憶的空間 **(Flash)**
- 增加特殊暫存器空間**(SFR)**及周邊
- 增加一般資料儲存空間 **(RAM)**
- 減少程式對程式頁及資料頁的切換
- 改進使用 **'C'** 的編譯效率
- 改善程式執行的效能

PIC16 新、舊架構比較

舊 PIC16Fxxx

High-Performance RISC CPU:

Only 35 instructions
- 所有指令執行時間爲一個指令週期，除了”分歧“指令

Operating speed:
- DC – **20MHz** oscillator/clock input
- DC – **200ns** instruction cycle

具中斷能力

8 層硬體中斷堆疊

直接，間接及相對定址模式

新 PIC16F1xxx

High-Performance RISC CPU:

Only 49 instructions
- 所有指令執行時間爲一個指令週期，除了”分歧“指令以外

Operating speed:
- DC – **32 MHz** oscillator/clock input
- DC – **125 ns** instruction cycle

具有背景自動儲存能力的中斷功能

16 層硬體中斷堆疊，內建堆疊溢位/借位的 **Reset** 功能

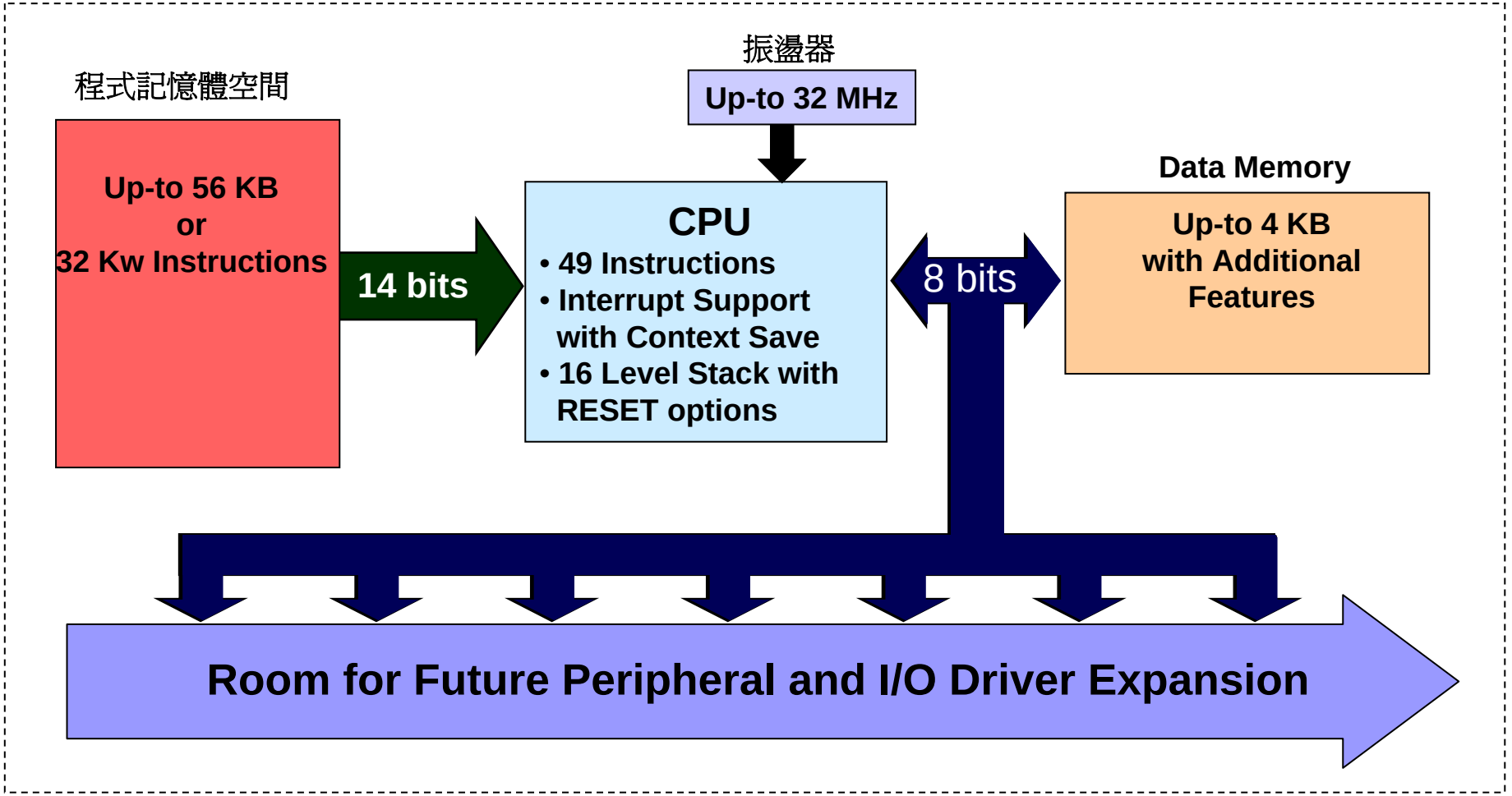
直接，間接及相對定址模式：
- 兩個可定址到 **16-bit FSRs** 暫存器
- **FSRs** 可讀取程式及資料空間

PIC 快速表較表

	PIC16	Enhanced PIC16	PIC18
Max GPR/SFR	336 / 110	2496 / 316	4096/159+ 有些周邊較多的 PIC18F 元件，部分的周邊並沒有放在 access bank.
Max Program	8Kx14	32Kx14	1Mx16
FSRs	1	2 可以透過 FSR 讀取程式記憶體	3
Instruction Count	35	49	75 加入擴展型指令共有 85 個指令
Stack	8	16 with over/under flow reset	31 with over/under flow reset
Interrupts	1	1 hardware context save	2 optional hardware context save
Program Memory Read	透過 RETLW 指令讀取，有些原件可以透過 EEPROM 周邊介面讀取 (PIC16F877)	透過 RETLW 指令讀取、也可以透過 EEPROM 周邊介面讀取或使用 FSR 定址方式讀取	所有元件須透過 TABLRD instructions.

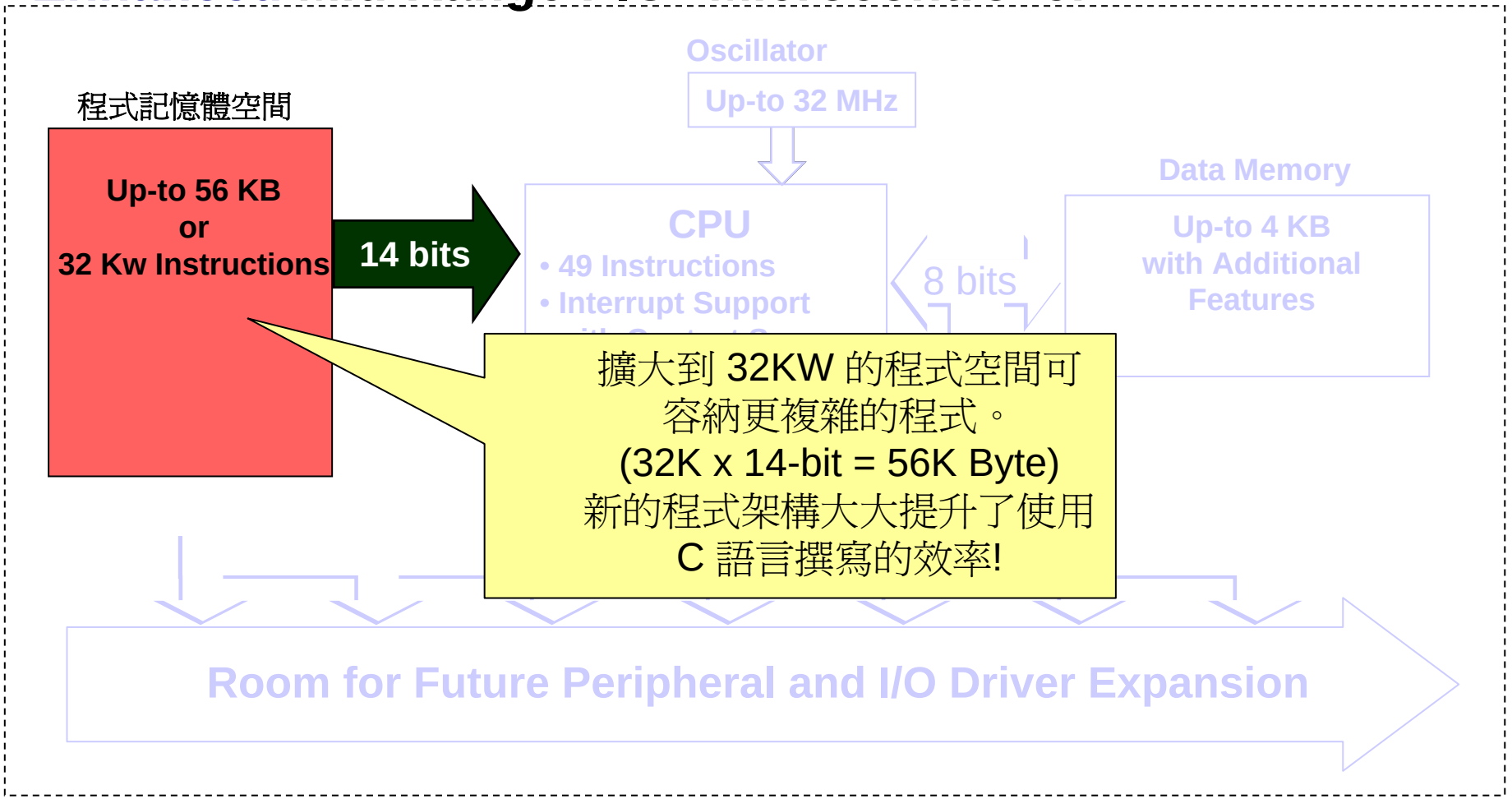
Enhanced PIC16F1xxx
基本架構

Enhanced Mid-Range PIC® Microcontroller



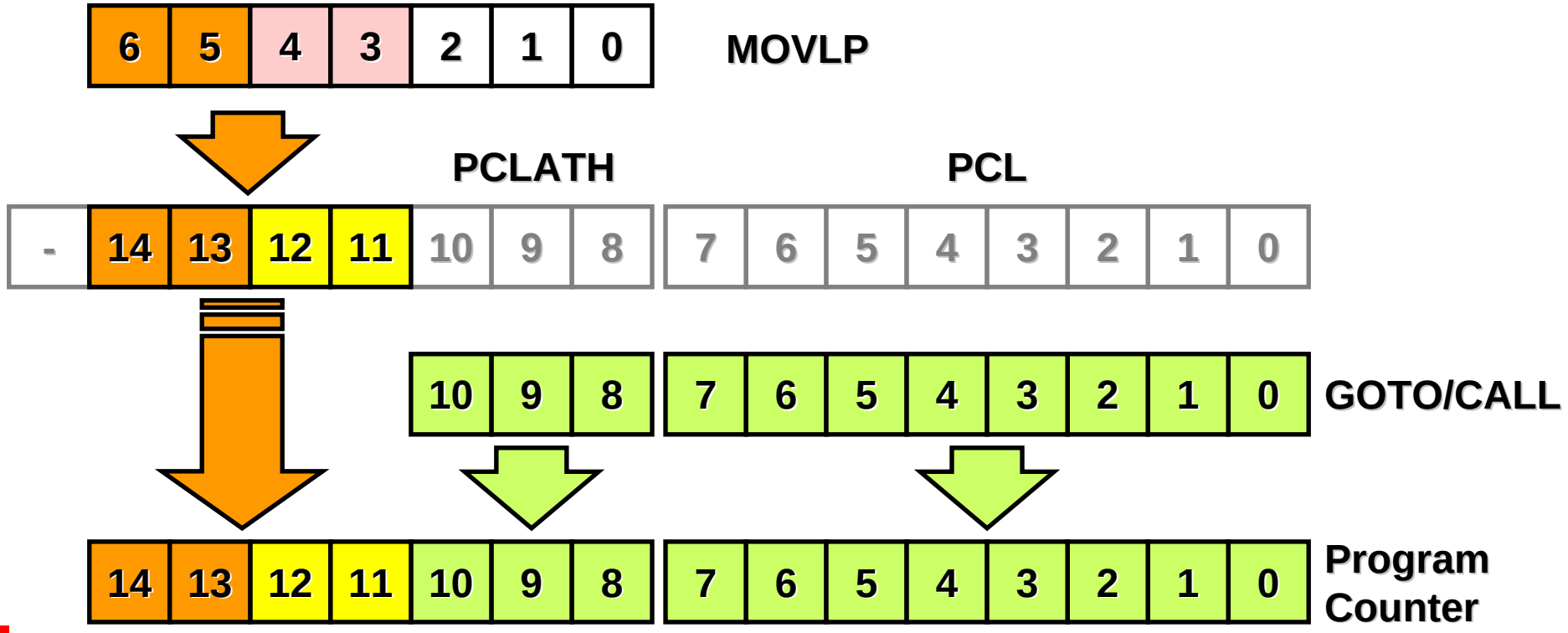
容量更大的程式空間 (Flash Memory)

Enhanced Mid-Range PIC® Microcontroller



程式可定址到 32KW 的程式空間

- 程式空間擴充到 **16** 個程式頁 **(page)** ，每頁受限於 **goto/call (11-bit** 位址長度**)** 指令仍維持 **2kw** 的大小。
- 使用 **MOVLP** 指令來簡化程式頁 **(Page)** 的切換 **(**舊的使用 **PAGESEL)**。



使用 MOVLP 做 Page 切換

- **MOVLP 指令**
  - 將 7 bit 直接值置入 PCLATH 暫存器裡
    - MOVLP HIGH *Lable*
  - 可以在一個令周期完成程式頁的切換，毋須動到 W reg. 暫存器。
  - MOVLP + CALL/GOTO 的組合將花掉 2 ~ 3 指令週期，其視野範圍可擴展到全部 32KW 的程式空間

相對位址的跳躍

- 對於短程的跳躍可以使用相對跳躍指令以減少程式對程式頁的切換動作
- 指令 **BRA N**
  - 直接听到目前 PC + N 位址
  - 跳躍的範圍 -256 <= N <= 255
  - PC + N 總長度爲 15 bit 位址，所以沒有換頁的問題
- 指令 **BRW**
  - 直接听到目前 PC + W (unsigned)
  - 做爲快速查表功能或 State Machines
  - PC + W總長度爲 15 bit 位址，所以沒有換頁的問題
- 指令 **CALLW**
  - 副程式呼叫功能 PCLATH, W
  - 快速查表功能 /State Machines/ 函數指標
  - 使用 PCLATH:W 直接位址方式

程式頁切換探討

我的組合語言寫法

```
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   goto My_function_loop
   return

Main
    do lots of stuff
   PAGESEL My_Function
   call My_Function
   do lots of other stuff

   end
```

PIC16 組譯方式

```
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   goto My_function_loop
   return

Main
    do lots of stuff
   movlw high My_Function
   movwf PCLATH
   call My_Function
   do lots of other stuff

   end
```

ENHANCED PIC16 組譯方式

```
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   goto My_function_loop
   return
```

Page Boundary

```
Main
    do lots of stuff
   movlp high My_Function
   call My_Function
   do lots of other stuff

   end
```

相對跳躍

原本的組合語言程式

另外插入的程式

```
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   goto My_function_loop
   return
```

程式頁邊界點變動造成呼叫原本的副程式 My_function_loop 時需要做程式頁的切換

```
Main
    do lots of stuff
    PAGESEL My_Function
    call My_Function
    do lots of other stuff
```

新架構下的組合語言程式

```
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   bra My_function_loop
   return
```

使用相對要越方式可以避免程式因跨業時所造成的錯誤問題

```
Main
    do lots of stuff
    PAGESEL My_Function
    call My_Function
    do lots of other stuff
    end
```

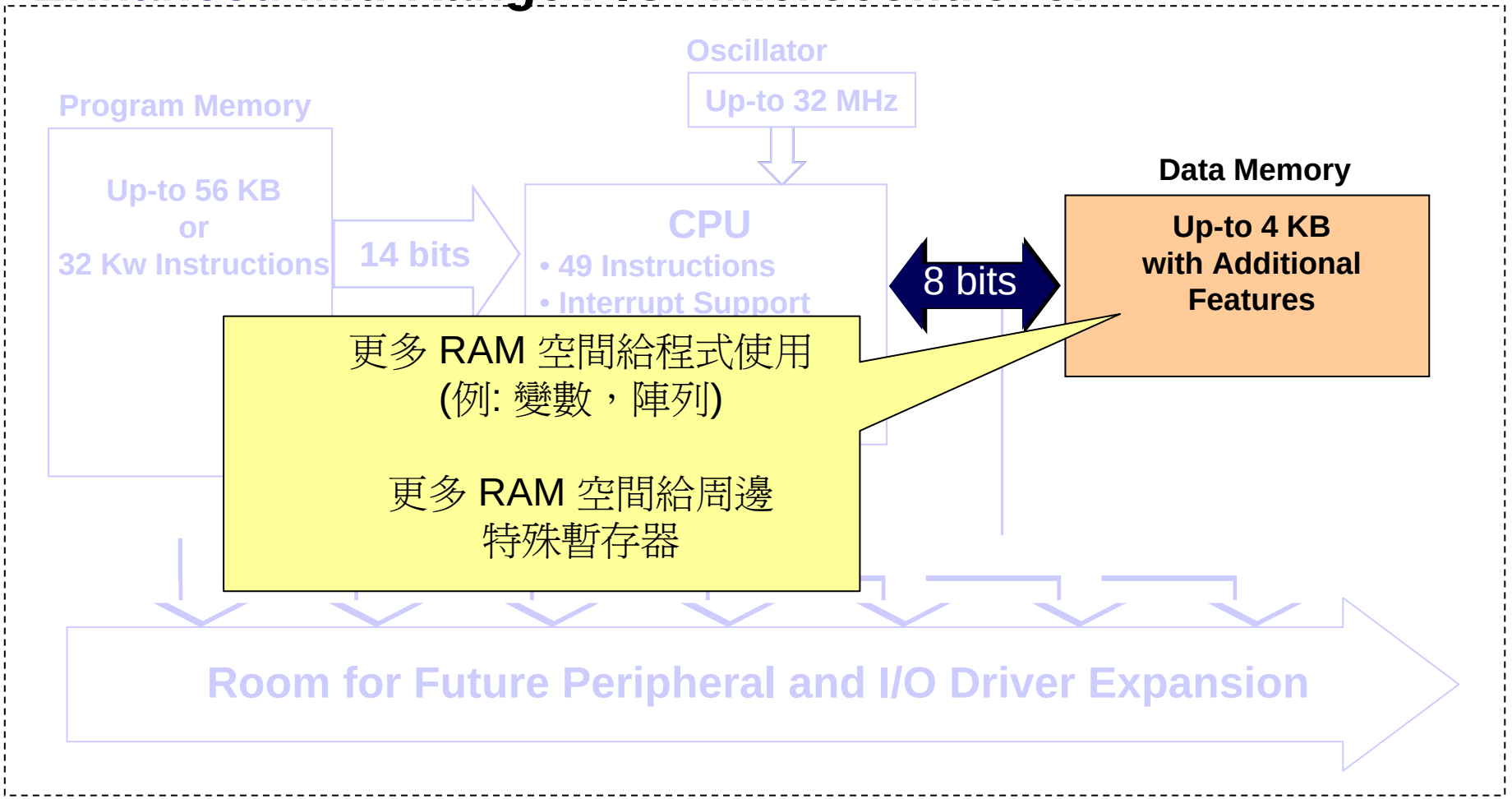
No relative CALL support. CALLW is not relative.

所以程式頁切換...

- 繼續使用 **PAGESEL** 的巨集指令
  - MPASM 會自動使用 MOVLP
- 更新所有的 **PCLATH** 的值
  - 確定 7 bit 新資料已更新在 PCLATH
- **轉換成相對跳躍方式**
  - 相對跳躍模式可避免程式頁切換的問題
  - 程式中因插入新程式時造成換頁問題

更多的資料記憶空間 (RAM & SFRs)

Enhanced Mid-Range PIC® Microcontroller



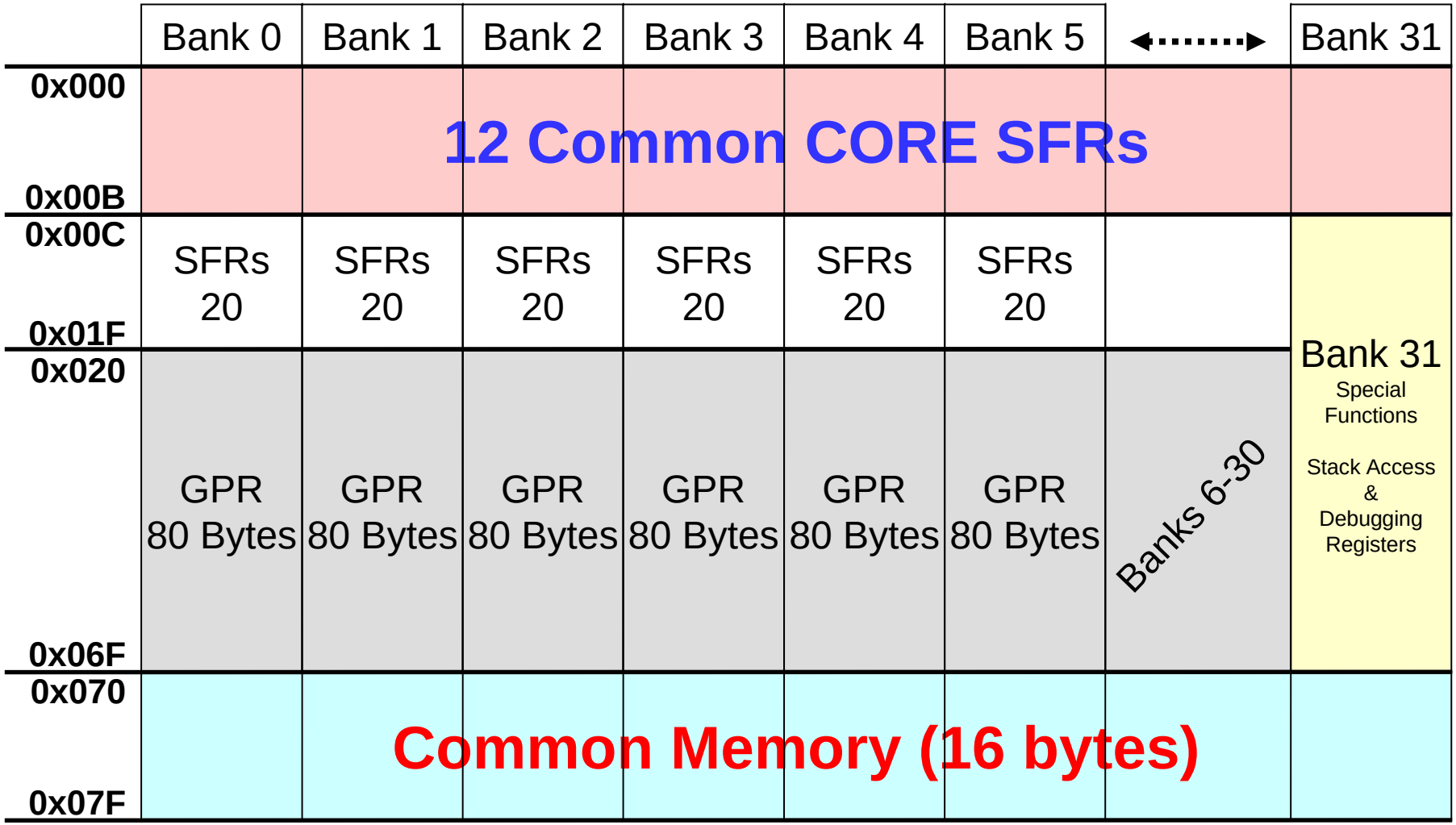
新的資料記憶體配置 (RAM)

- 總共 **32 banks** 資料記憶空間
  - 每個 **Bank** 一樣爲 **128Bytes**
  - 目前保留 **15 banks** 給以後新元件使用

- 更簡單使用的 **RAM :**
  - 保留每個 **Bank** 的最底下 **16 bytes RAM** 做爲共用 **RAM** 區域
  - 保留每個 **Bank** 的最初的 **12 bytes RAM** 做爲共用 **CPU registers.**
  - **SFRs** 及周邊佔用每個 **Bank** 位址 **0x0C** 到 **0x1F**

- 新加入的功能
  - **W** 爲 **RAM (0x09)** 的一員稱之爲 **“w reg” (**與 **PIC18** 一樣**)**
  - **Banks 16-30** 給以後新元件使用
  - **Bank 31** 給堆疊及除錯使用

RAM 的配置

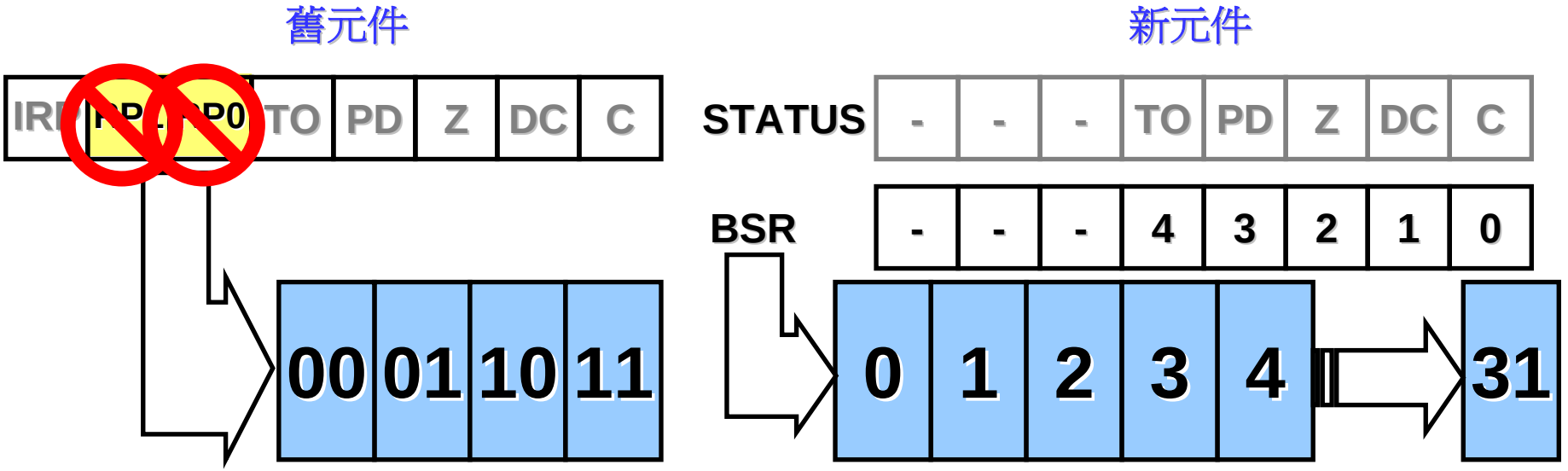

Bank 0
Bank 1
Bank 2
Bank 3
Bank 4
Bank 5
Bank 31
0x000
0x00B
12 Common CORE SFRs
0x00C
0x01F
SFRs 20
0x020
0x06F
GPR 80 Bytes
Banks 6-30
Bank 31
Special Functions
Stack Access & Debugging Registers
0x070
0x07F
Common Memory (16 bytes)

Common Core Registers

	Address	Register	Function
	0x00	INDF0	Indirect Register 0
NEW	0x01	INDF1	Indirect Register 1
	0x02	PCL	Program Counter Low
	0x03	STATUS	Status Register
NEW	0x04	FSR0 Low	File Select Register 0 Low Byte
NEW	0x05	FSR0 High	File Select Register 0 High Byte
NEW	0x06	FSR1 Low	File Select Register 1 Low Byte
NEW	0x07	FSR1 High	File Select Register 1 High Byte
NEW	0x08	BSR	Bank Select Register
NEW	0x09	WREG	Working Register
	0x0A	PCLATH	Program Counter Latch High
	0x0B	INTCON	Interrupt Control Register

有關 RAM Bank 的切換
直接定址模式

- 在舊的 **PIC16， Bank** 的切換是透過 **Status** 暫存器的 **RP0**及**RP1** 位元
- 在新的架構下，這兩個位元已經不存在了
- 新架構下，使用 **BSR** 暫存器來管理 **Bank** 的切換
- 使用新的 **MOVLB** 指令來做 **Bank** 的選擇



BANKSEL

我的組合語言

```
 data
Var1 res 1
Var2 res 1
Var3 res 1

  code

Main
    do lots of stuff
    BANKSEL Var1
    addwf  Var1
    do lots of other stuff


    end
```

實際在 PIC16 的組合語言

```
 data
Var1 res 1
Var2 res 1
Var3 res 1

  code

Main
    do lots of stuff
    bsf STATUS,RP0
    bcf STATUS,RP1
    addwf  Var1
    do lots of other stuff

    end
```

新架構下的組合語言

```
 data
Var1 res 1
Var2 res 1
Var3 res 1

  code

Main
    do lots of stuff
    movlb Var1 >> 7
    addwf  Var1
    do lots of other stuff


    end
```

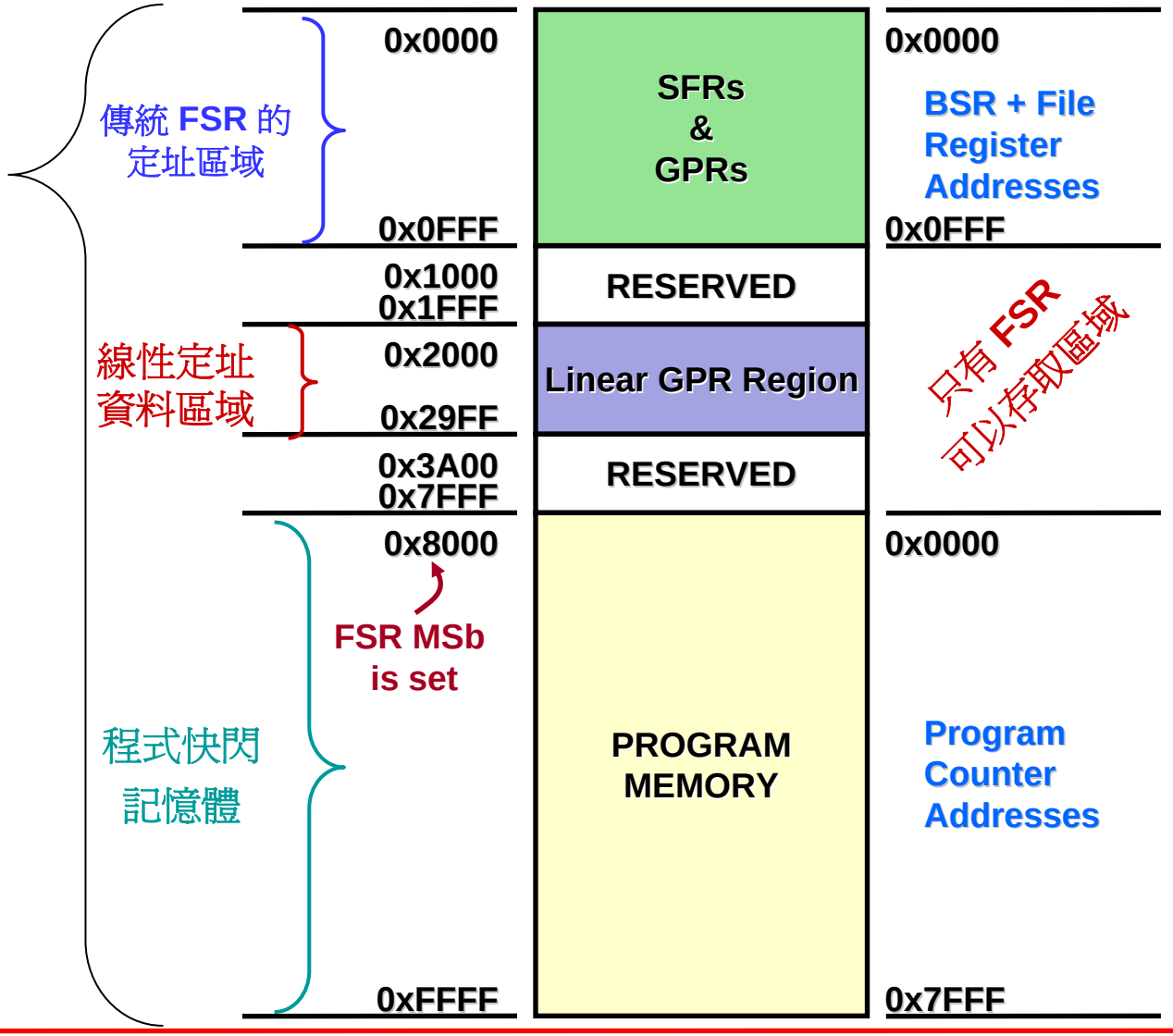
在做 Bank 的切換時可以減少一個指令的動作

間接定址模式 (FSR)

- **IRP bit** 已經不存在了，用 **FSRxH** 取代了
- 可以直接存取超過 **256 bytes**以上的資料，只要更新 **FSRxH** 暫存器
- 透過 **W** 暫存器的寫入，可以更快速有效的修改 **FSRxH** 暫存器
- 提供更有效率的查表指令
- **BANKISEL** 仍可繼續使用，直接修改 **FSRxH** 暫存器

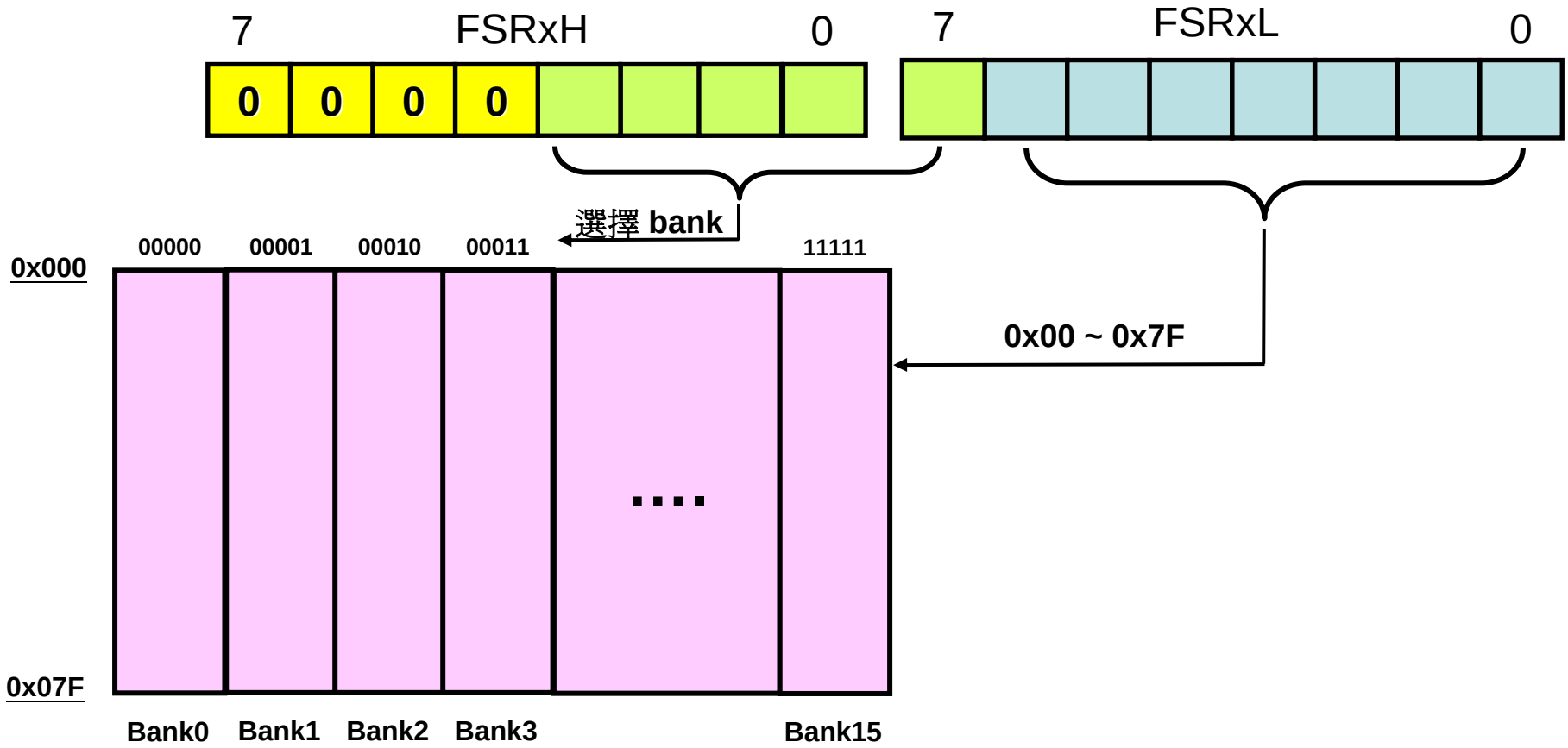
新架構下的 FSRs

- 兩組 **16-bit FSRs**
- 視野可達 **64K** 全範圍
  - 傳統定址資料區域
  - 線性定址資料區域
  - 程式快閃記憶體
- **FSR** 最高位元
  - = 0，RAM 空間
  - = 1，程式空間
- **FSRs** 有更多的功能支援以方便查表的應用



FSR 傳統定址

- 直接定址到 **RAM (0x0000 ~ 0x0FFF)**
  - 因爲每個 **banks SFR** 佔用最前面的位址，所以傳統的索引還是有 **bank** 設定問題



Lab1
FSR 傳統定址方式

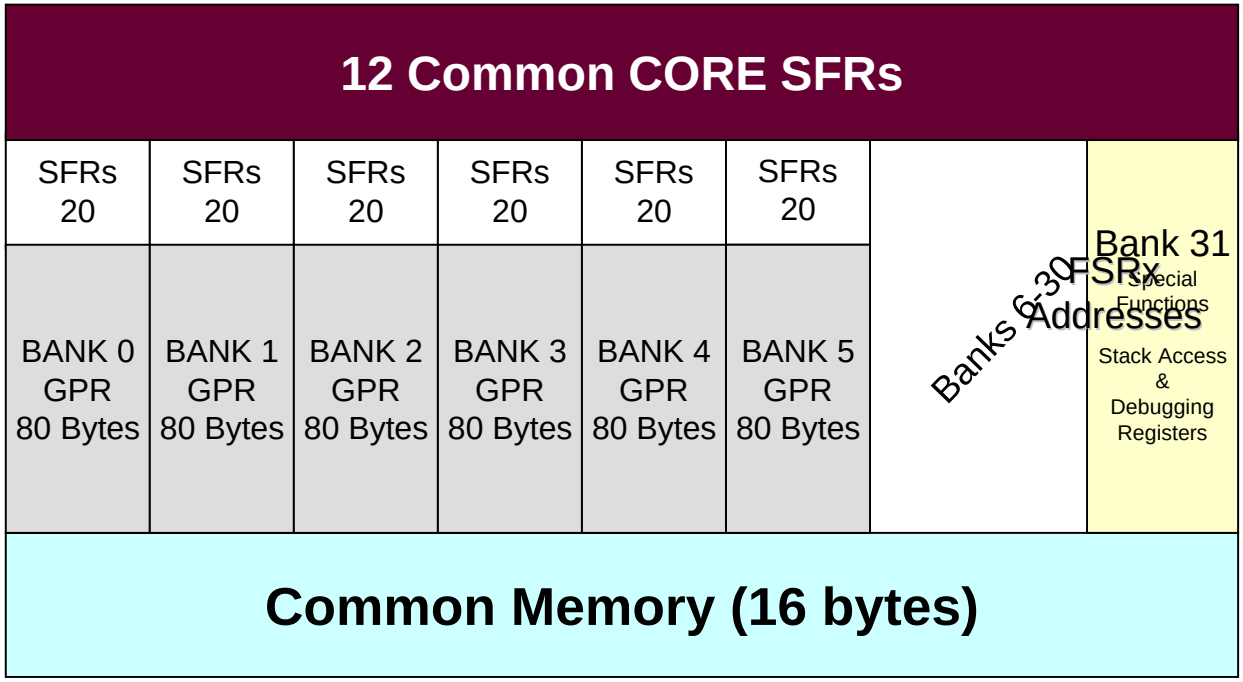
- 專案位置：
  - .. \Advance PICC Application Labs\Lab1 Traditional FSR\ Traditional FSR Access RAM.mcp
- 本實驗將以傳統的索引定址方式將四個 **Bank** 填上數值。
  - Bank0 (0x20 ~ 0x7F) 填入 0x00
  - Bank1 (0xA0) ~ 0xEF) 填入 0x01
  - Bank2 (0x120 ~ 0x17F) 填入 0x02
  - Bank3 (0x1A0) ~ 0x1EF) 填入 0x03
- 使用 **MPLAB SIM** 軟體模擬及開啓 **“File Registers”** 來監看執行結果。

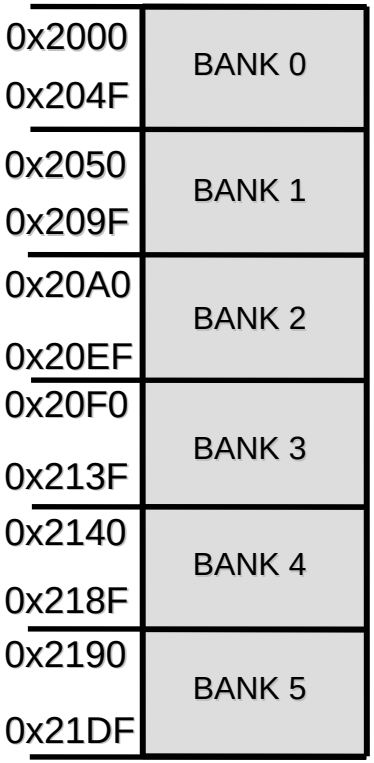
Lab1 程式說明
Traditional FSR.c

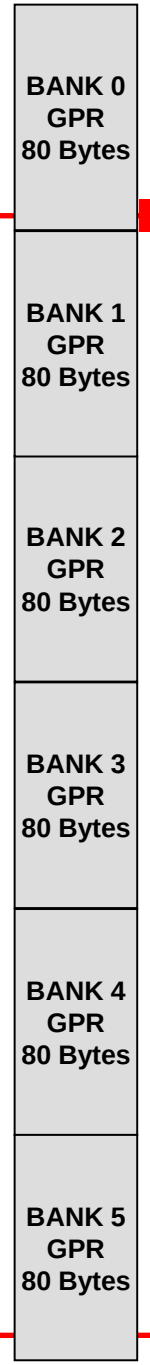
```
void main(void)
{
        OSCCONbits.IRCF=0B1110;         // 設定 Fosc =32MHz (8MHz *
    4 PLL)

        FSR1 = 0x20;                    // FSR 指向起始位址

    while ( FSR1 <= 0x1FF)              // 完成 4 個 Bank 的填值 ?
    {
        if (FSR1 >= 0x20 && FSR1 <= 0x7F) INDF1 = 0x00;
        if (FSR1 >= 0xA0 && FSR1 <= 0xEF) INDF1 = 0x01;
        if (FSR1 >= 0x120 && FSR1 <= 0x16F) INDF1 = 0x02;
        if (FSR1 >= 0x1A0 && FSR1 <= 0x1EF) INDF1 = 0x03;
        FSR1++;
    }
    while(1);

}
```

FSR 資料區線性定址

- 整合每個**BANK**裡 **80Bytes RAM** 爲線性定址區域
- **FSRx** 線性定址起始位址 **0x2000 ~ 0x29AF (BANK31** 不列入**)**
- 允許使用較大的資料堆疊、陣列、暫存區 **…** 等
- 不需切 **BANK** 直接**,** 透過 **FSRx** 就可以存取
- 定址範例 **: unsigned char InputBuffer[100] @ 0x2000;**







Lab2
FSR 線性定址方式

- 專案位置：
  - .. \Advance PICC Application Labs\ Lab2 Linear FSR\ Linear FSR.mcp
- 本實驗將以 **PIC16F1xxx** 新的線性索引定址方式來存取陣列 **InputBuffer[256]**。
  - 若陣列大小，小於、等於 **80 Bytes** 查詢資料：
    - 使用傳統 **Mid-Range** 方式放在 **BANKn** 的起始位址。
  - 若陣列大小，大於 **80 Bytes** 查詢資料：
    - 使用 **FSR** 線性定址方式 **(**可以支援巨大陣列功能**)**。
  - 程式中將使用到 **Common Memory** 來擺放指標。
- 使用 **MPLAB SIM** 軟體模擬及開啓 **“File Registers”** 來監看執行結果。

Lab2 程式說明

Linear FSR.c

```
unsigned char InputBuffer[256]    ;                  // 宣告陣列
unsigned char *near PTR ;                            // PTR 指標位址設在 Common Memory
near unsigned int j ;                                // 變數 j 放在 Common Memory

void main(void)
{
        OSCCONbits.IRCF=0B1110;                      // 選用 8MHz * 4 (PLLEN_ON) = 32MHz(Fosc)


    PTR = InputBuffer ;                              // 設定指標位址

    for ( j=0; j<=255; j++ )
    {
        *PTR =0xA5;                                  // 將陣列填入 0xA5
        PTR++;
    }

    while(1);

}
```

使用 FSR 查表應用

- 較慢的方法
  - 表格位址設定給 FSRx
  - 將索引值加到 FSRx 裡
  - 查表後再使用 RETLW 返回
  - 因沒有考慮 FSR0L 進位其範圍受限制

```
The_CODE
        movlw  high Table_start
        movwf  FSR0H
        movlw  low Table_start
        movwf  FSR0L
        movlw  3
        addwf  FSR0L
        movf   INDF0,w

Table_start
        DT 3,4,5,6,7,8,9
```

快速查表方式

- 這種查表方式有受限於 **256 Word** 的邊界限制

```
Table_Function
        movlw  high Table_start
        movwf  PCLATH
        movlw  3
        movwf  PCL
Table_start
        DT 3,4,5,6,7,8,9
```

```
The_CODE
        movlp  high Table_start
        movlw  3
        callw

Table_start
        DT 3,4,5,6,7,8,9
```

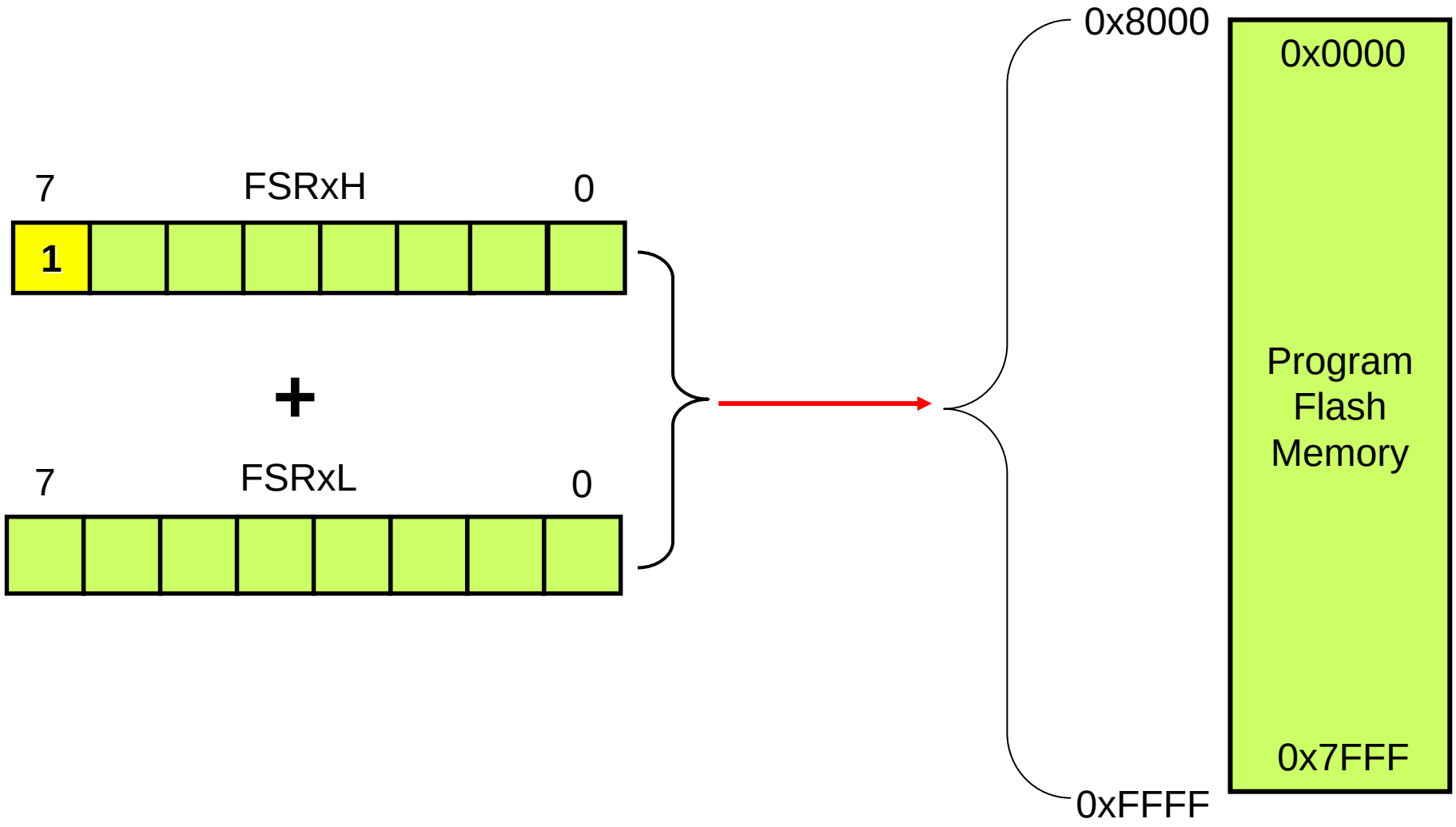
更多快速查表方式

- 沒有邊界限制的問題
- 查表的範圍不受限制

```
The_CODE
        movlw  high Table_start
        movwf  PCLATH
        movlw  low Table_start
        addwf  3
        btfss  STATUS,C
        incf   PCLATH,f
        movwf  PCL
Table_start
        DT 3,4,5,6,7,8,9
```

```
The_CODE
        movlp  high Table_start
        movlw  3
        brw
Table_start
        DT 3,4,5,6,7,8,9
```

FSR 程式記憶線性定址



Lab3
使用 FSR 程式線性查表方式

- 專案位置：
  - .. \Advance PICC Application Labs\Lab3 Program FSR\Program Lookup Table.mcp
- 本實驗將以 **PIC16F1xxx** 新的程式查表方式來測試查詢超過 **256 Bytes** 的表格
  - 查詢表格，小於 **255 Bytes** 查詢資料：
    - 使用傳統 **Mid-Range** 方式查表。
  - 查詢表格，大於 **255 Bytes** 查詢資料：
    - 使用 **FSR** 程式現行查表方式 **(**可以支援巨大查表功能**)**。
- 使用 **MPLAB SIM** 軟體模擬及開啓 **“File Registers”** 來監看執行結果。

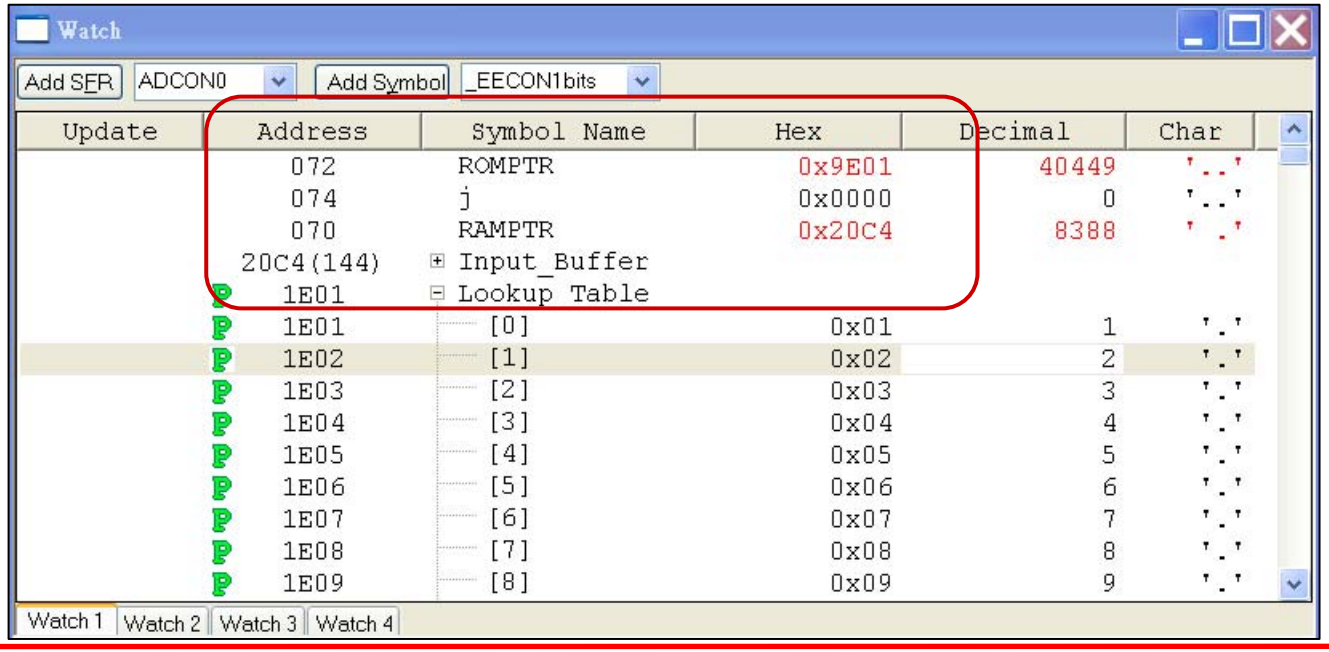
Lab3
FSR 程式線性查表方式

```
const unsigned char Lookup_Table[ ]  =
      {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
       18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
       34,35,36,37,38,39,40,41,42,43,44,45,56,57,48,49,50,
       51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
       68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,
       84,85,86,87,88,89,80,91,92,93,94,95,96,97,98,99,100,
      …….
      };
unsigned char Input_Buffer[300];
const unsigned char *near ROMPTR;          // 指標放在 Common Memory，指標指向 Pmemory
unsigned char *near RAMPTR ;               // PTR 指標位址設在 Common Memory
near unsigned int j ;                      // 變數 j 放在 Common Memory
void main(void)
{
      OSCCONbits.IRCF=0B1110;              // 選用 8MHz * 4 (PLLEN_ON) = 32MHz(Fosc)

      ROMPTR = Lookup_Table ;
      RAMPTR = Input_Buffer ;

      for (j=0; j<=300; j++)
      {
            *RAMPTR++ = *ROMPTR++;         // Copy data from ROM to RAM
      }
      while(1);
}
```

Lab3 指標示範

```
37      void main(void)
38      {
39          OSCCONbits.IRCF=0B1110;
40
41
42      ROMPTR = Lookup_Table ;
43      RAMPTR = Input_Buffer ;
44
45      for (j=0; j<=300; j++)
46      {
47          *RAMPTR++ = *ROMPTR++;
48      }
49
50      while(1);
```

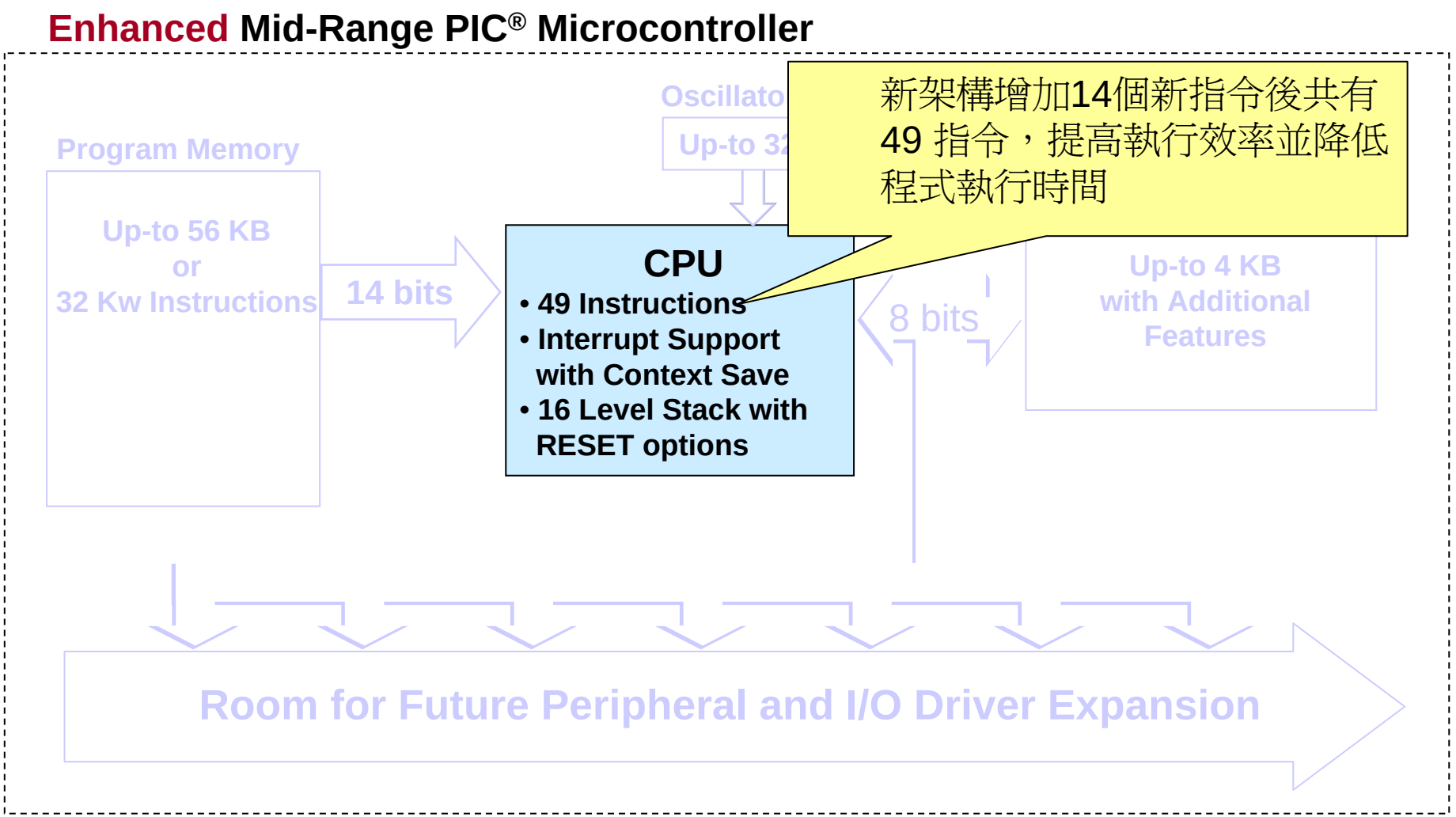
程式執行停在斷點處，觀察一下
ROMPTR & RAMPTR 的實際位址



Enhanced PIC16F1xxx

新功能

更加的性能表現


Enhanced Mid-Range PIC® Microcontroller
新架構增加14個新指令後共有49 指令，提高執行效率並降低程式執行時間
Oscillato
Up-to 32
Program Memory
Up-to 56 KB
or
32 Kw Instructions
14 bits
CPU
• 49 Instructions
• Interrupt Support with Context Save
• 16 Level Stack with RESET options
8 bits
Up-to 4 KB
with Additional
Features
Room for Future Peripheral and I/O Driver Expansion

新加入的指令

Mnemonic	Description
ADDWFC	Add W+F with Carry
SUBWFB	Subtract F-W with Borrow
LSLF	Logical Shift Left
LSRF	Logical Shift Right
ASRF	Arithmetic Shift Right
MOVLP	Move Literal to PCLATH
MOVLB	Move Literal to BSR
BRA	Branch Relative (signed)
BRW	Branch PC + W (unsigned)
CALLW	Call PCLATH:W
ADDFSR	Add Literal to FSRn (signed)
MOVIW	Move indirect to W
MOVWI	Move W to Indirect
RESET	Reset Hardware & Software

新增加的 FSR 指令
讓 C 編譯更有效率

- **ADDFSR 指令**
  - 將一個有號數的立即值加到所選定的 FSRx
  - [FSRxH:FSRxL] + k
  - 有號數範圍：K= -32 ~ +31

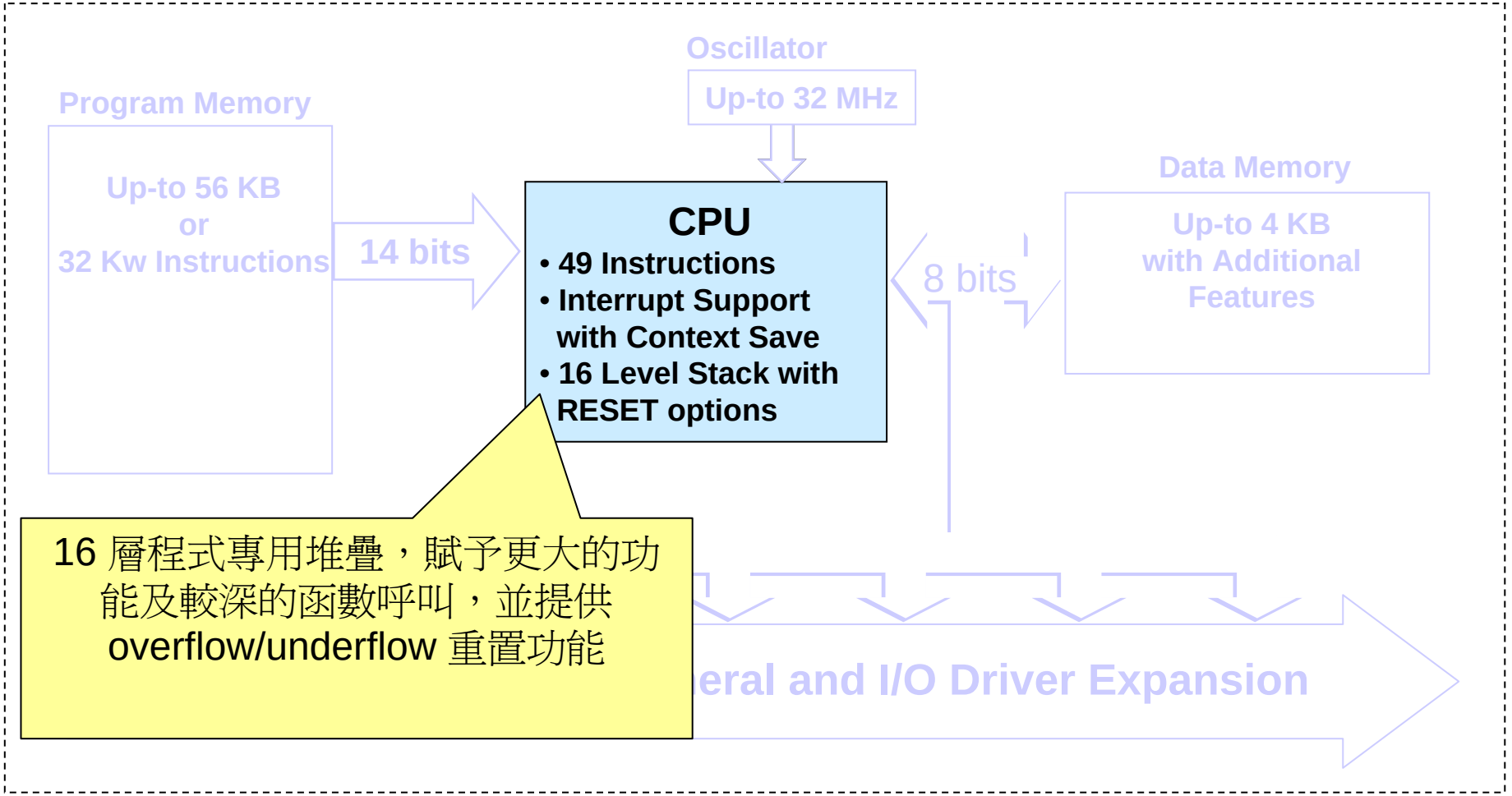
- **MOVIW/MOVWI – 將索引位址指到的資料搬到 W 或 將 W 的資料移到索引位址裡**
  - 增強模式
    - Pre/Post Increment
    - Pre/Post Decrement
    - Relative Offset
      - ✓ Same range as ADDFSR

MOVIW / MOVWI 語法

- **Standard**
  `MOVWI 0[INDF0]`
- **Pre Increment**
  `MOVIW ++INDF0`
- **Post Increment**
  `MOVWI INDF0++`
- **Pre Decrement**
  `MOVIW --INDF0`
- **Post Decrement**
  `MOVWI INDF0--`
- **Offset**
  `MOVWI k[INDF0]`
  -32 <= k <= 31

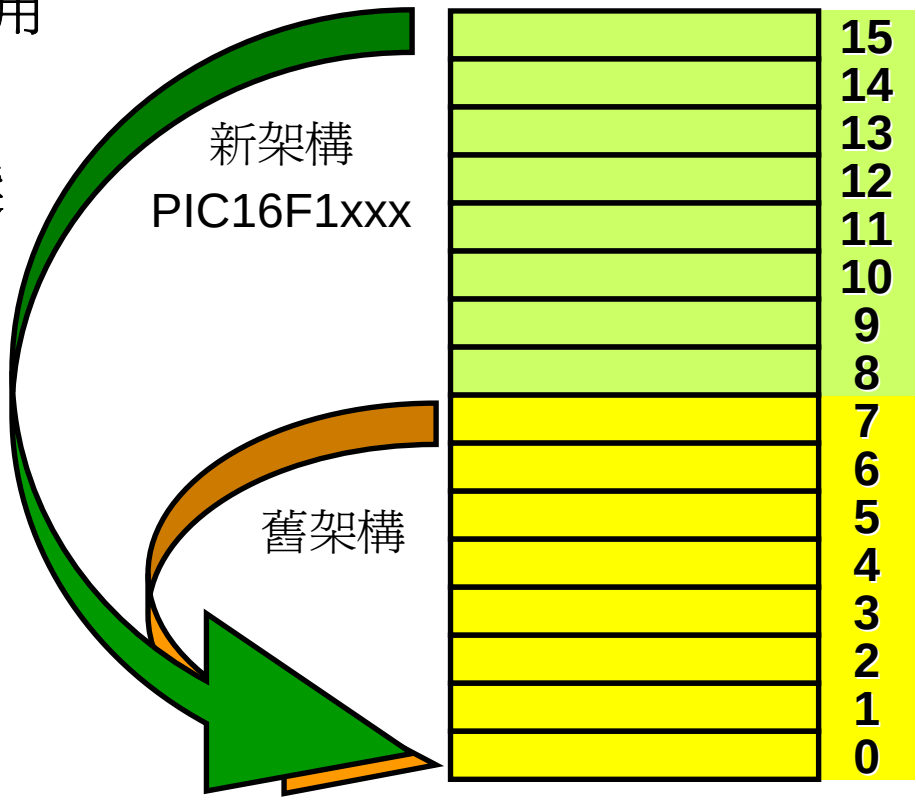
*紅色標示爲執行該指令後，**FSRx** 內容值不變*

改良後的堆疊表現


Enhanced Mid-Range PIC® Microcontroller
Oscillator
Up-to 32 MHz
Program Memory
Up-to 56 KB
or
32 Kw Instructions
14 bits
CPU
• 49 Instructions
• Interrupt Support with Context Save
• 16 Level Stack with RESET options
8 bits
Data Memory
Up-to 4 KB with Additional Features
16 層程式專用堆疊，賦予更大的功能及較深的函數呼叫，並提供 overflow/underflow 重置功能
eral and I/O Driver Expansion

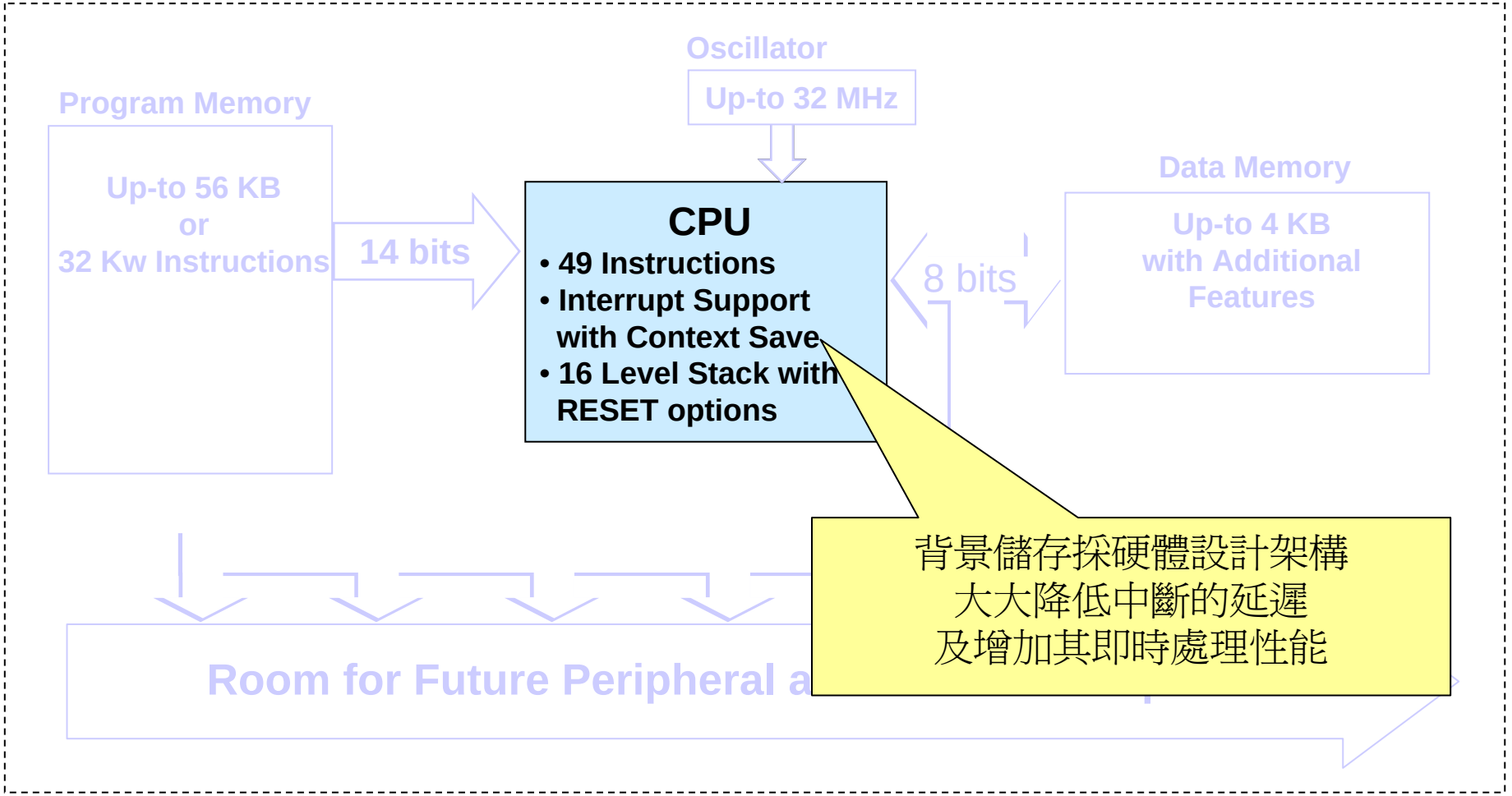
16 層的堆疊

- **16 層 15 bit 寬的堆疊深度**
- 硬體設計，呼叫函數及中斷使用
  - 存取 PC
- **Over/Underflow reset (可選擇)**



更快的中斷響應

Enhanced Mid-Range PIC® Microcontroller



快速存取中斷背景資料 shadow

- 中斷發生時背景資料將自動儲存到 **Bank 31** 的 **Shadow** 暫存器
  - **W → WREG_SHAD**
  - **STATUS → STATUS_SHAD**
  - **BSR → BSR_SHAD**
  - **FSR0 → FSR0_SHAD**
  - **FSR1 → FSR1_SHAD**
  - **PCLATH → PCLATH_SHAD**
- **RETFIE** 指令背景自動取回
- 快速背景儲存動作無法被關閉

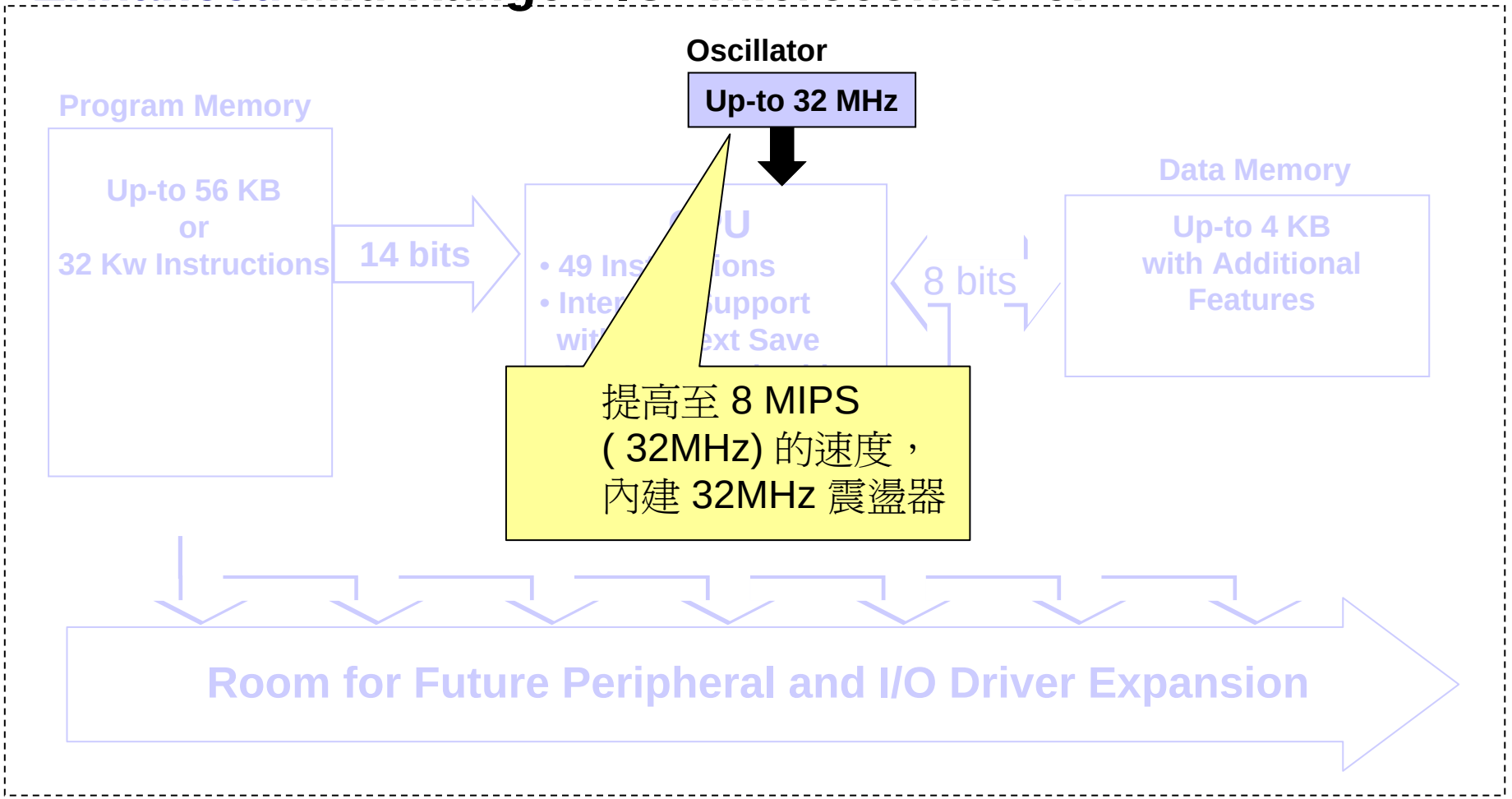
Accessing the context save shadow

- 中斷所儲存的背景值是放在 **bank 31** 且其值是可以被讀寫的

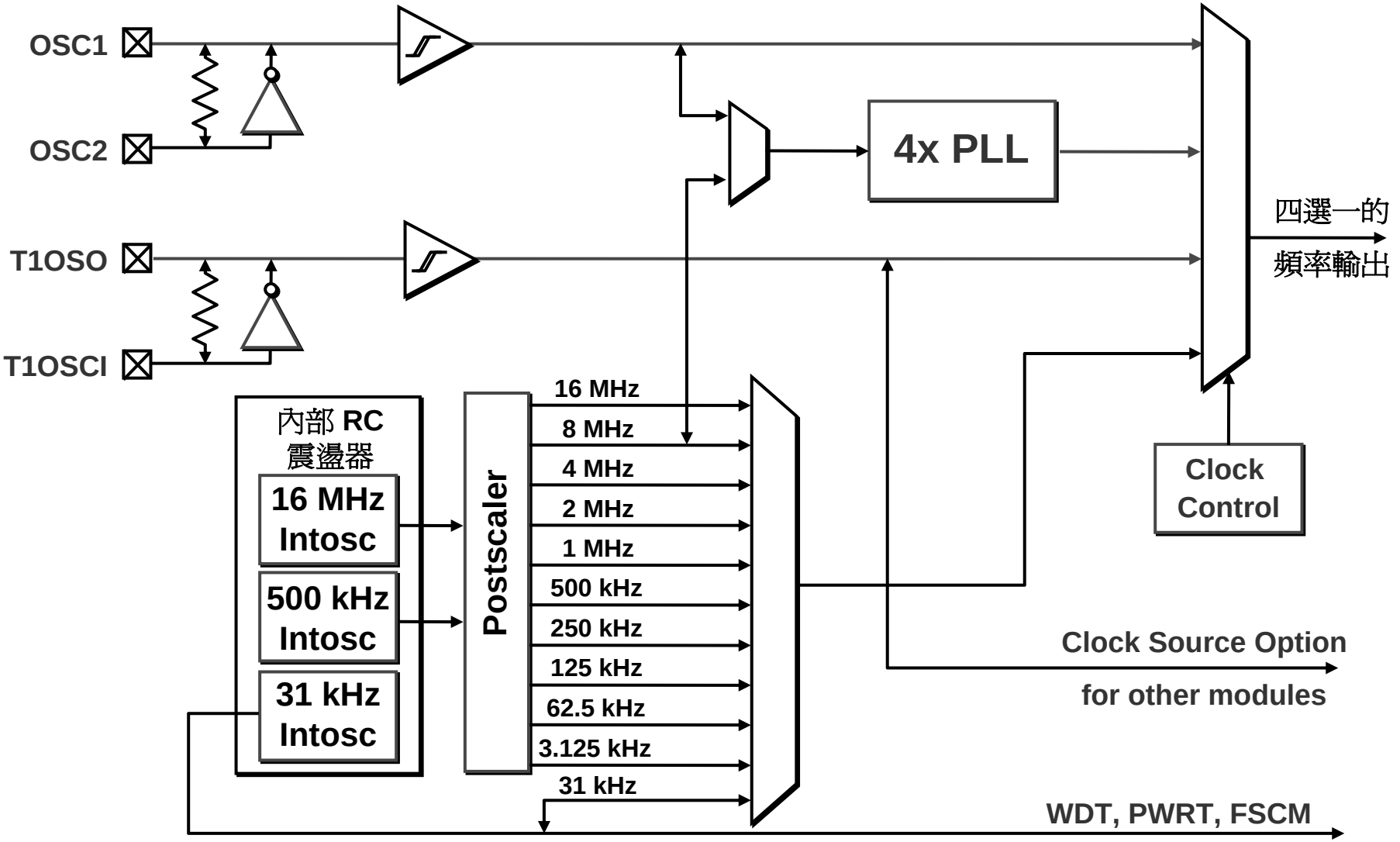
STATUS	STATUS_SHAD (儲存在 Bank 31)
FSR0 Low	FSR0L_SHAD
FSR0 High	FSR0H_SHAD
FSR1 Low	FSR1L_SHAD
FSR1 High	FSR1H_SHAD
BSR	BSR_SHAD
WREG	WREG_SHAD
PCLATH	PCLATH_SHAD

更快的執行速度

Enhanced Mid-Range PIC® Microcontroller

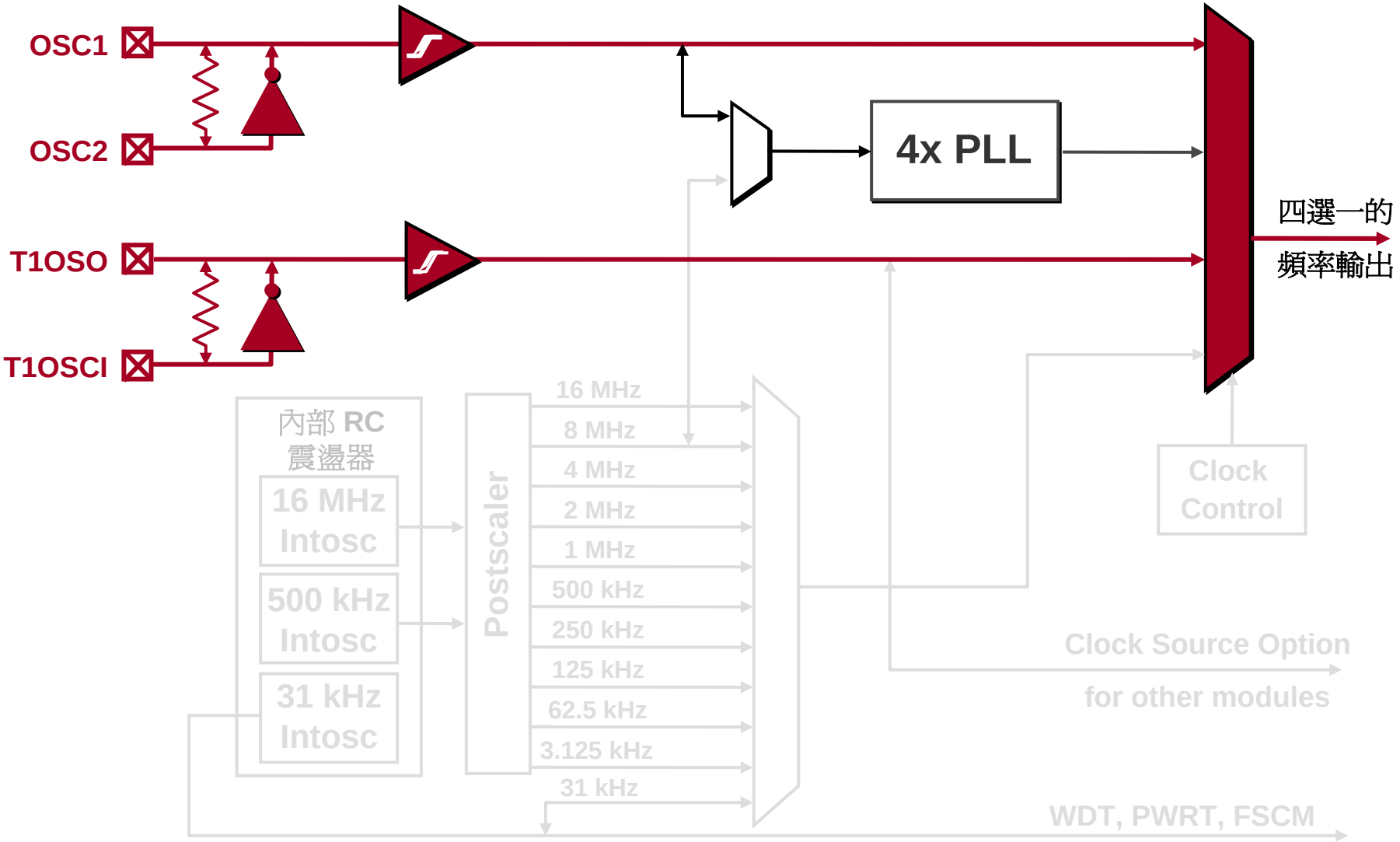


震盪器方塊圖

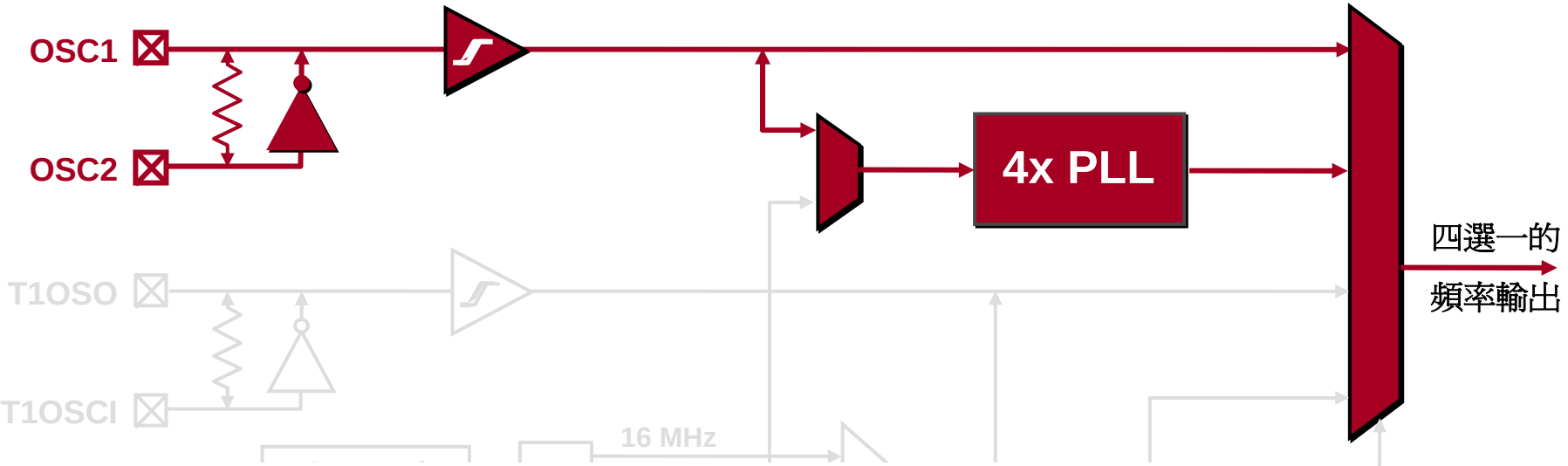

OSC1
OSC2
T1OSO
T1OSCI
4x PLL
四選一的
頻率輸出
內部 RC
震盪器
16 MHz
Intosc
500 kHz
Intosc
31 kHz
Intosc
Postscaler
16 MHz
8 MHz
4 MHz
2 MHz
1 MHz
500 kHz
250 kHz
125 kHz
62.5 kHz
3.125 kHz
31 kHz
Clock
Control
Clock Source Option
for other modules
WDT, PWRT, FSCM

兩種外接震盪器

外接主振盪 與 T1 外接低頻震盪



外接主振盪器與 PLL



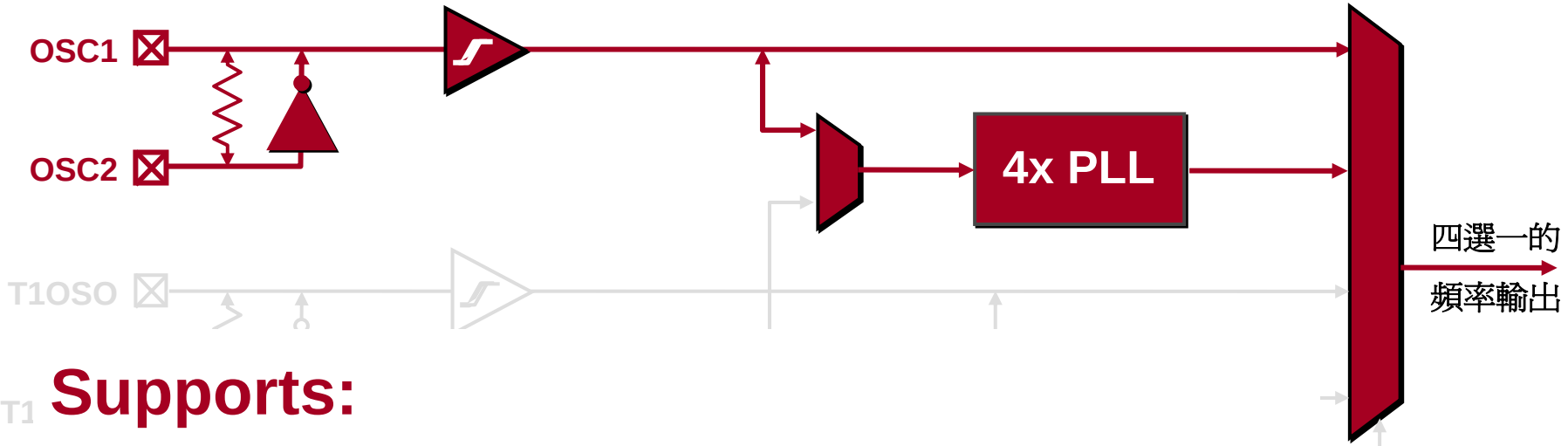
PLL 頻率輸入範圍:

Fosc – PLL 輸入頻率 4MHz ~ 8MHz

Fsys – 內建 vco 輸出頻率範圍 16 MHz ~ 32MHz

Delta clk – PLL 頻率抖動率 0.25%

外接主振盪器的種類



Supports:

EC – 外接震盪頻率輸入

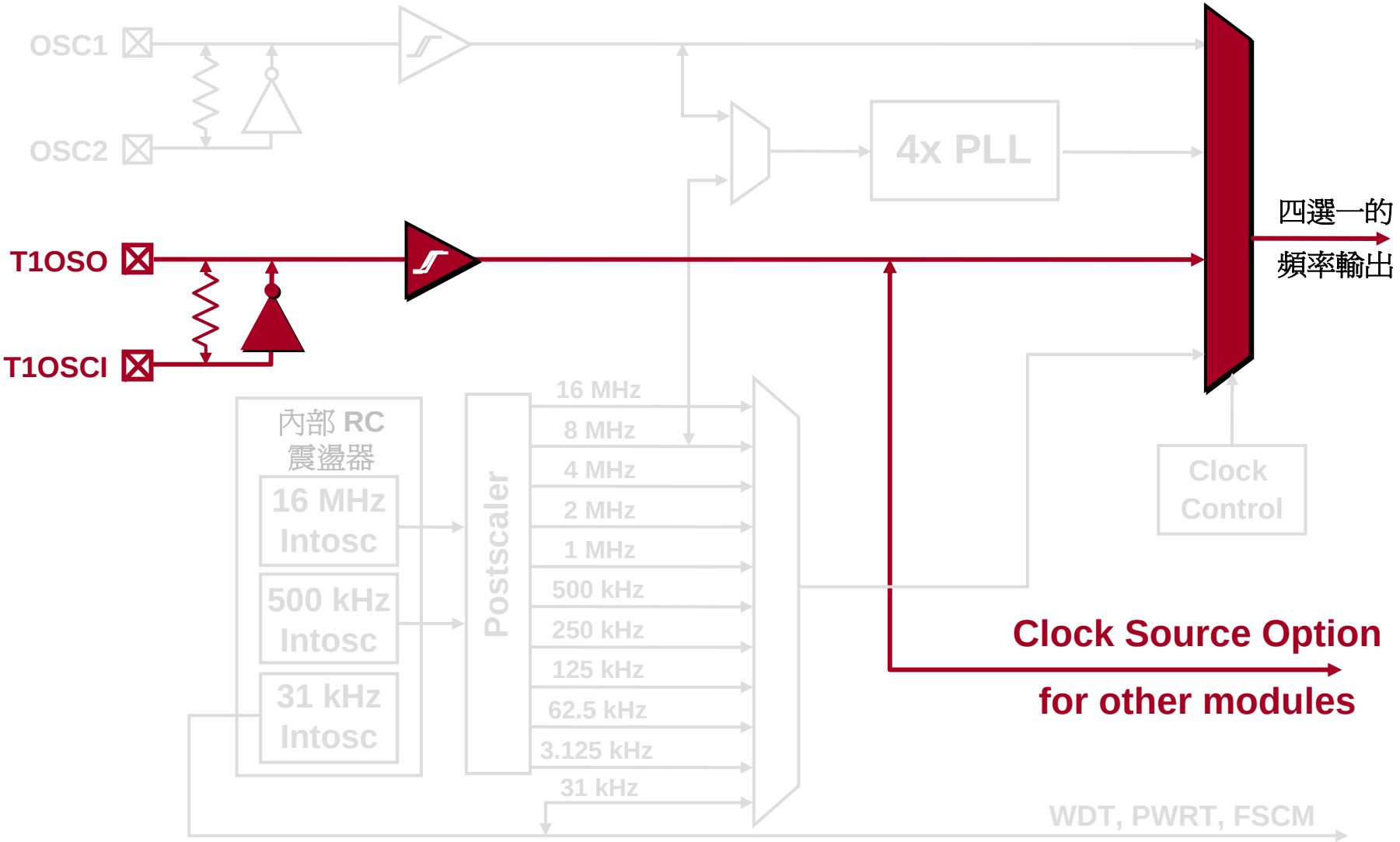
LP – 32 kHz 低功耗石英晶體

XT – 中間頻率 (< = 4MHz) 石英晶體 或 陶瓷震盪器

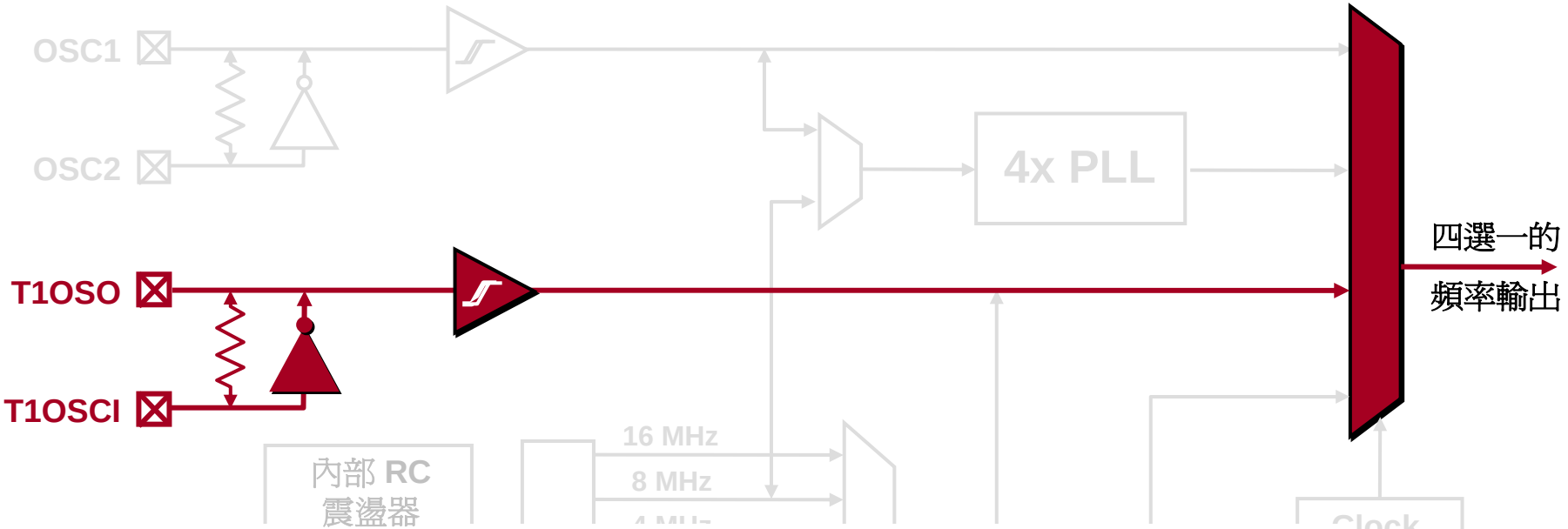
HS – 高頻率 (> 4MHz) 石英晶體 或 陶瓷震盪器

RC – 外接電阻電容式震盪器

Timer1 震盪器 (T1OSC)



Timer1 震盪器 (T1OSC)

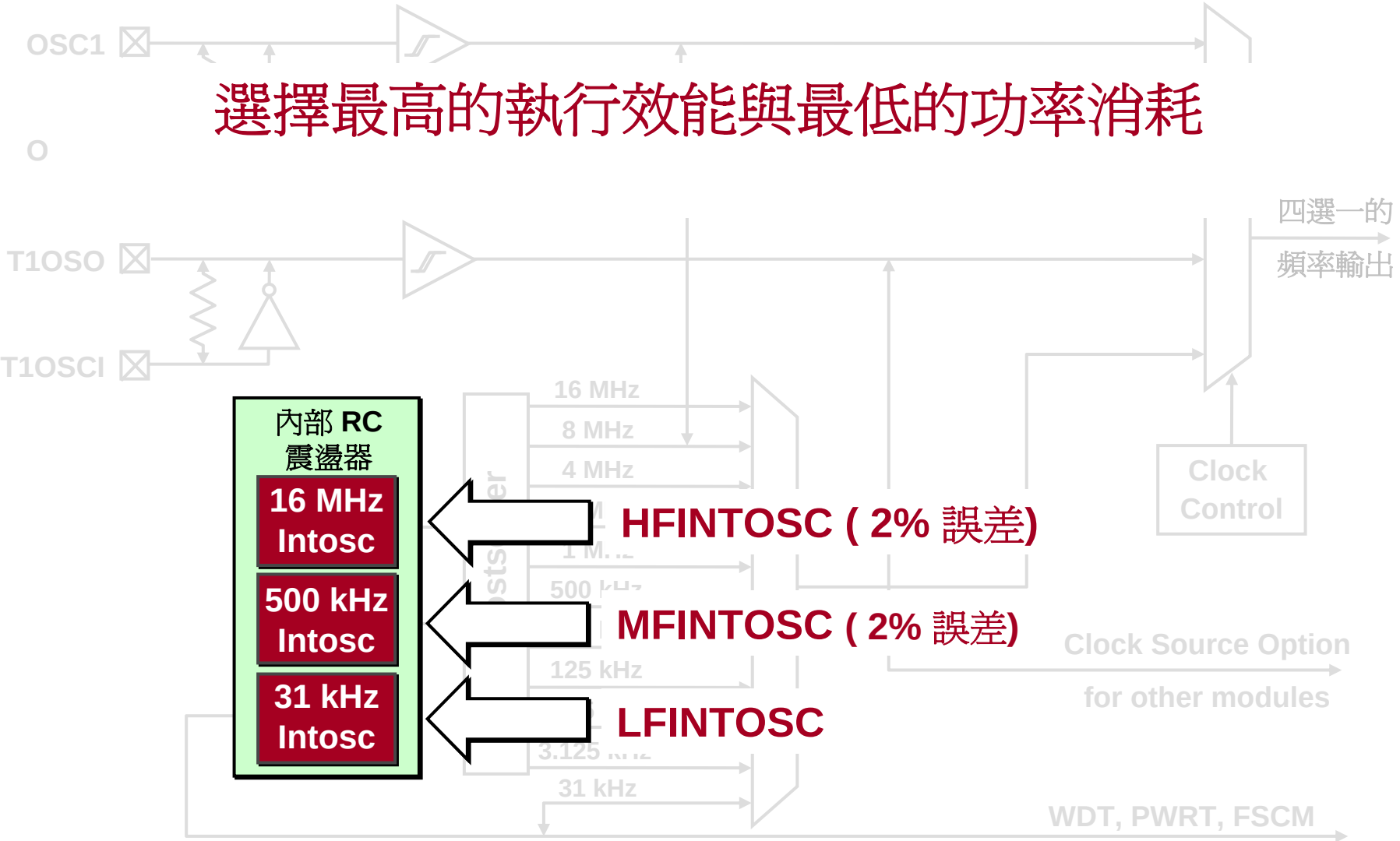


在 32.768 kHz 工作頻率下
一般的耗電流爲 500 nA, @ 3.0V

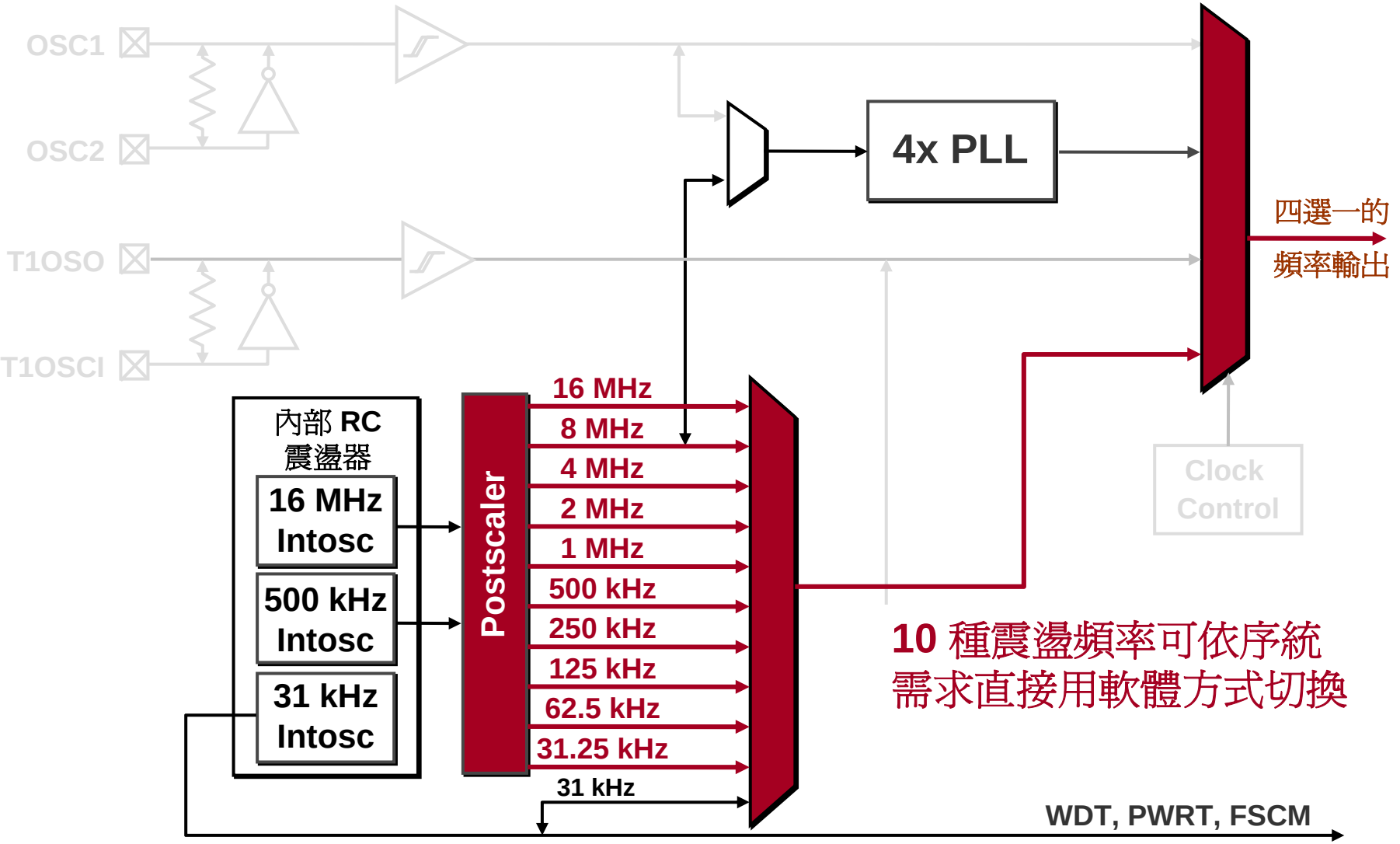
此種低功耗設計非常適用於
需精確計時的應用

三個內建 RC 震盪器

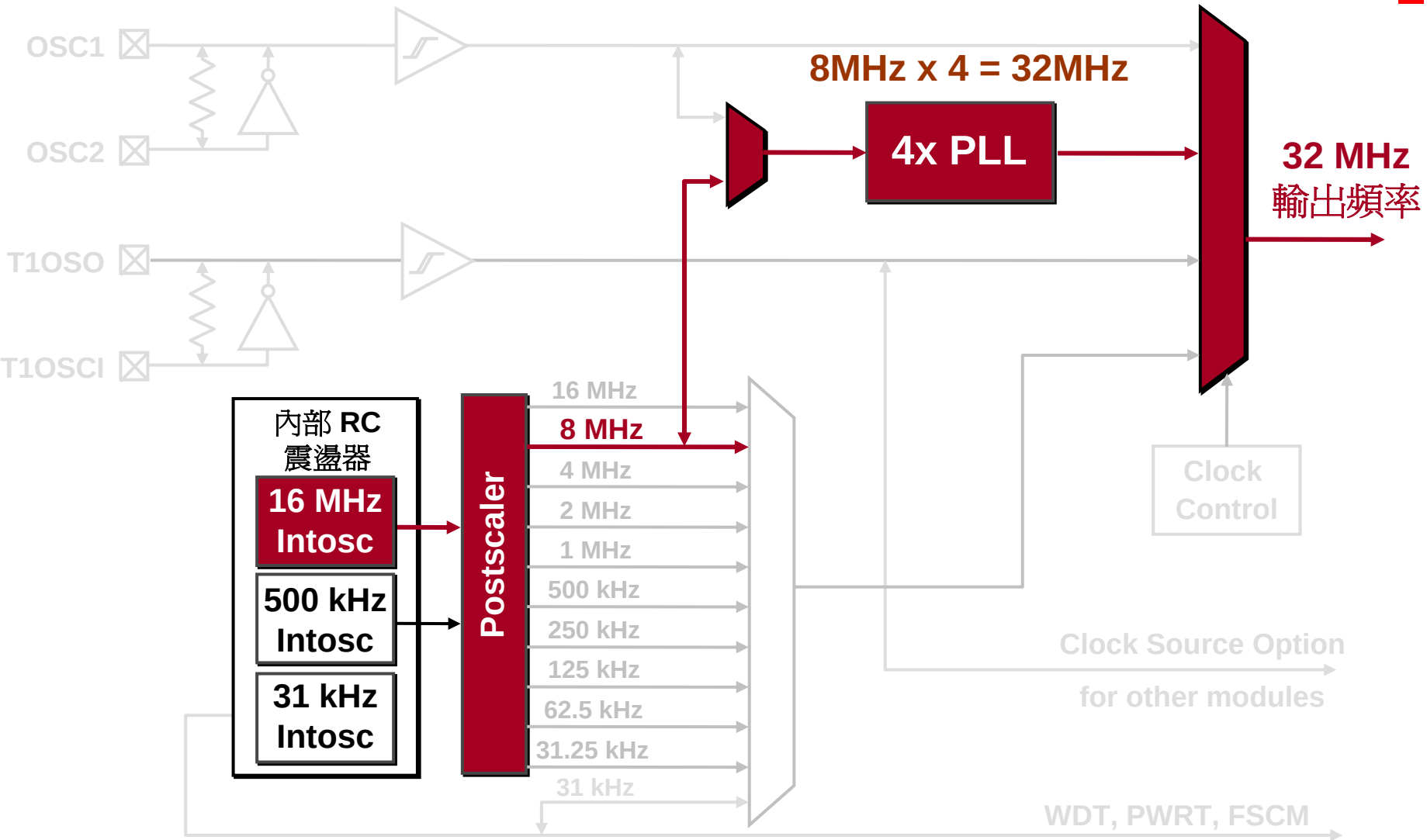
選擇最高的執行效能與最低的功率消耗



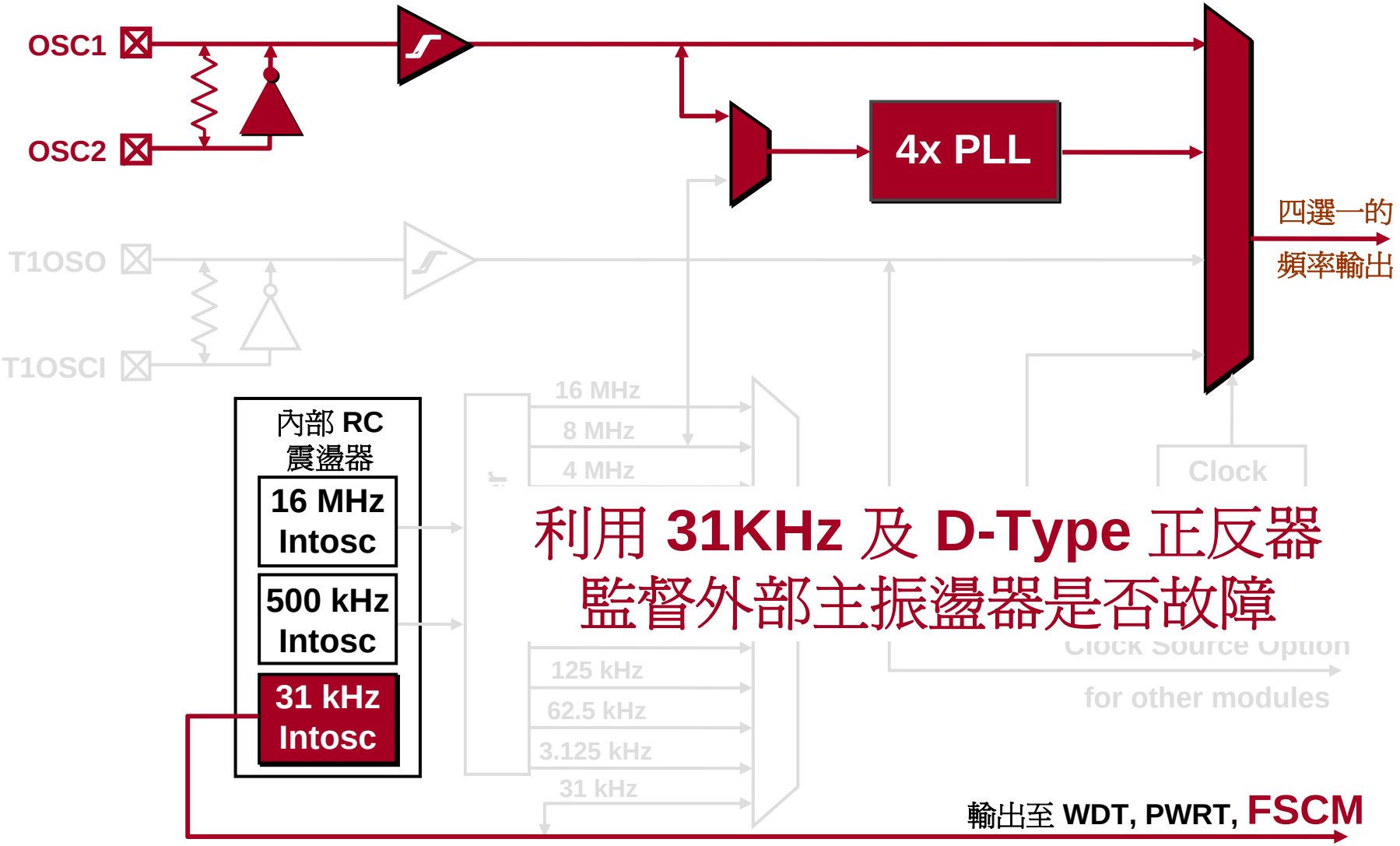
內建 RC 震盪器方塊


OSC1
OSC2
T1OSO
T1OSCI
4x PLL
四選一的
頻率輸出
內部 RC
震盪器
16 MHz
Intosc
500 kHz
Intosc
31 kHz
Intosc
Postscaler
16 MHz
8 MHz
4 MHz
2 MHz
1 MHz
500 kHz
250 kHz
125 kHz
62.5 kHz
31.25 kHz
31 kHz
Clock
Control
10 種震盪頻率可依序統
需求直接用軟體方式切換
WDT, PWRT, FSCM

使用 4 倍頻的內部震盪輸出

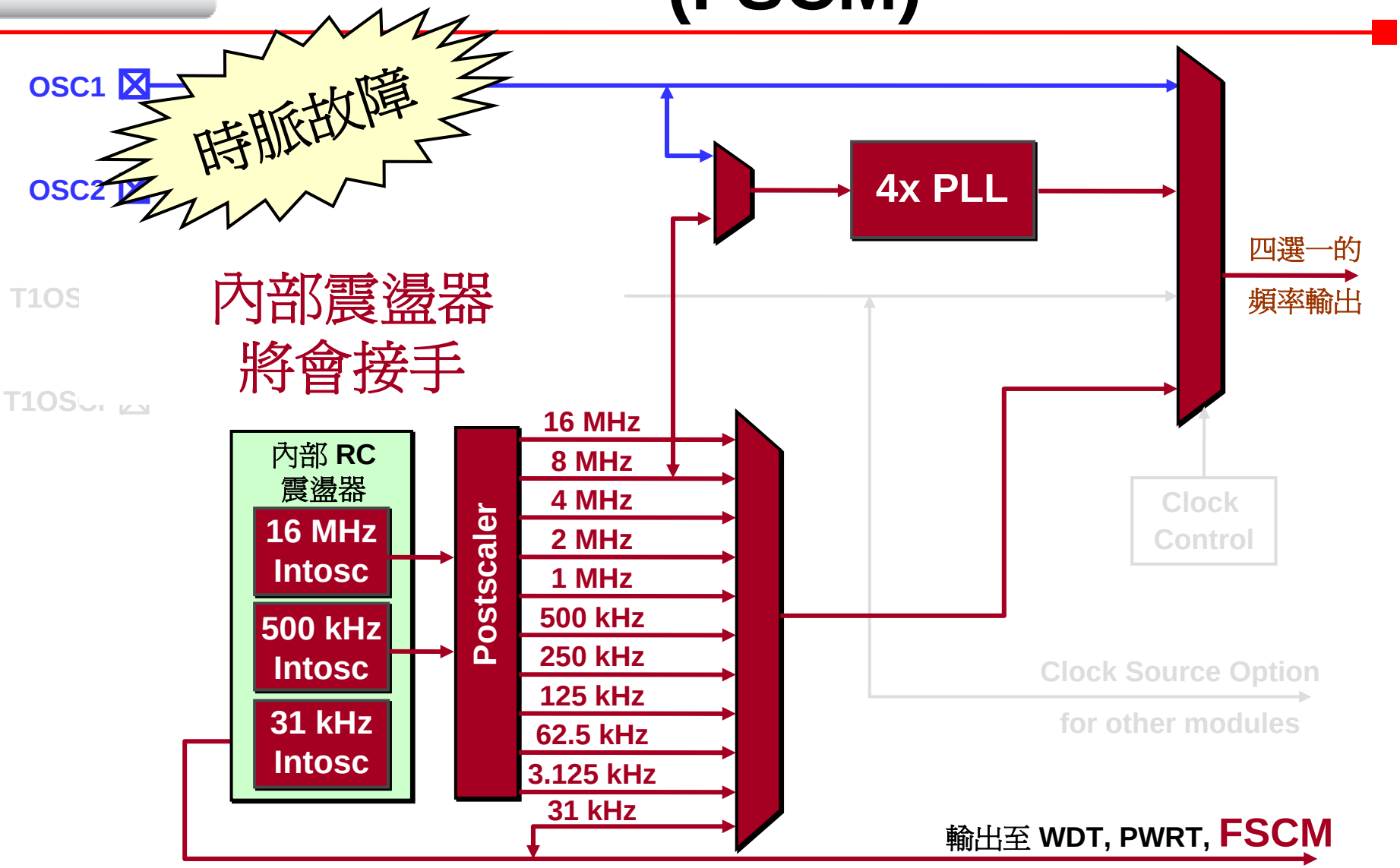


Fail-Safe Clock Monitor (FSCM)



利用 **31KHz** 及 **D-Type** 正反器
監督外部主振盪器是否故障

Fail-Safe Clock Monitor (FSCM)



PORT Peripherals

PORT 周邊暫存器

- 每一個 **PORT** 都有三個暫存器來運作

PORTx: PORTx Register

Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0

RBx<7:0>: PORTx I/O Pin bit
1 = PORTx pin is > VIH
0 = PORTx pin is < VIL

TRISx: PORTx TRI-STATE Register

TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0

TRISx<7:0>: PORTx Tri-State Control bit
1 = PORTx pin 設定成輸入腳功能 **(tri-stated)**
0 = PORTx pin 設成輸出腳功能

新加入 LATx 暫存器

- 每一個 **PORTx** 暫存器都有一個相對應的 **LATx** 暫存器
- 資料寫到 **PORTx**
  - 同時資料也寫到 LATx
- 自 **PORTx** 讀取資料
  - 實際是讀取外部 I/O 腳的數值

避免 **Read/Modify/Write** 的問題發生

PORTx: PORTx REGISTER

Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0

LATA: PORTA DATA LATCH REGISTER

LATx7	LATx6	LATx5	LATx4	LATx3	LATx2	LATx1	LATx0

帶有類比輸入的 PORTs

- 開機後的初始設定爲類比輸入腳功能，如需使用數位功能需做設定

ANSELx: PORTx Analog Select Register

ANSx7	ANSx6	ANSx5	ANSx4	ANSx3	ANSx2	ANSx1	ANSx0

ANSx<7:0>: Analog Select bit
1 = Pin 設定爲 **I/O PORT** 或數位週邊功能
0 = Pin 設定爲類比輸入功能腳 **(**開機後內定值**)**

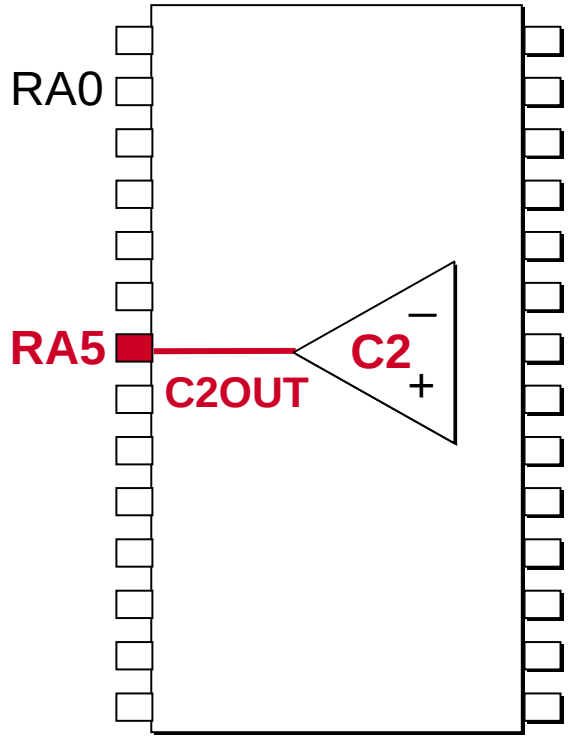
腳位替換功能

- 有些周邊的輸入、輸出腳可以被規劃在替換腳位上
  - $\overline{SS}$ (Slave Select)
  - CCP2
  - CCP3
  - Timer1 Gate Input
  - SR Latch $\overline{Q}$ Output
  - Comparator C2 Output

APFCON: Alternate Pin Function Control Register

	CCP3SEL	T1GSEL	P2BSEL	SRNQSEL	C2OUTSEL	SSSEL	CCP2SEL

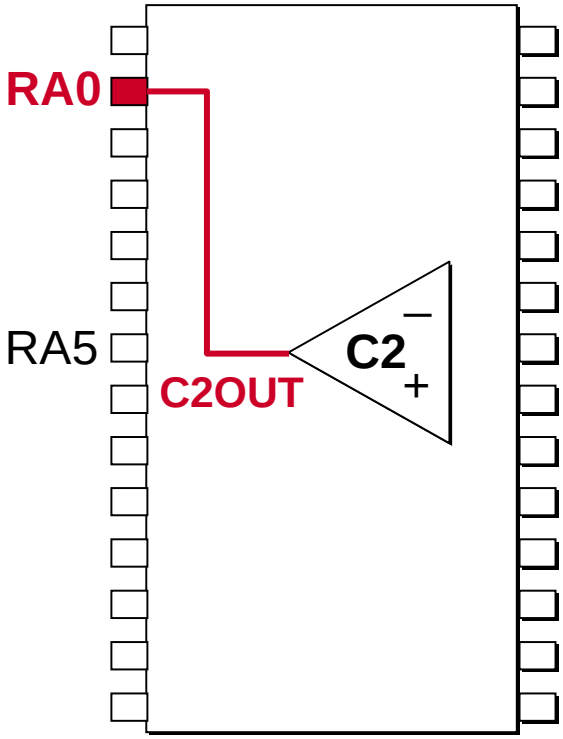
腳位的互換設定



APFCON: Alternate Pin Function Control Register

	CCP3SEL	T1GSEL	P2BSEL	SRNQSEL	0	SSSEL	CCP2SEL

腳位的互換設定



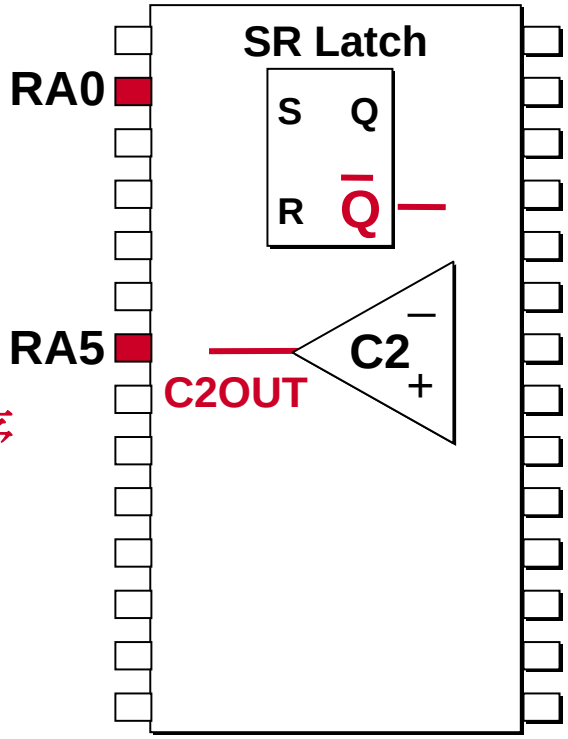
APFCON: Alternate Pin Function Control Register

	CCP3SEL	T1GSEL	P2BSEL	SRNQSEL	1	SSSEL	CCP2SEL

腳位的互換設定

多功能腳位的選擇是透過內部的多工選擇器來設定的

PORTx 腳位使用的優先順序請參考 **Data Sheet** 的標示

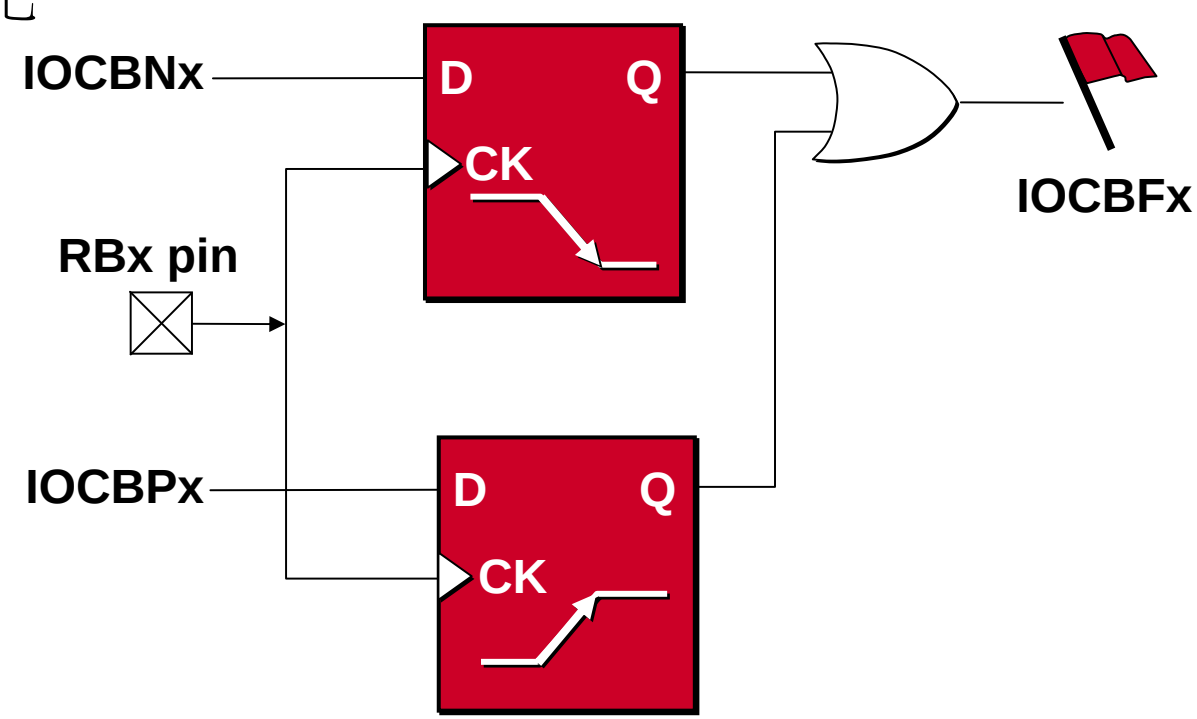


APFCON: Alternate Pin Function Control Register

	CCP3SEL	T1GSEL	P2BSEL	SRNQSEL	C2OUTSEL	SSSEL	CCP2SEL

Interrupt-on-Change

- 腳位上的位準改變可以觸發中斷
  - 上升、下降緣的改變
  - 上升緣位準變化
  - 下降緣位準變化



LCD Module

內建 LCD 模組的功能

- 內建 LCD 驅動模組可直接驅動 LCD 玻璃
- 彈性的 LCD 節點驅動數
  - 28 pins - up to 60 segments
  - 44 pins - up to 96 segments
  - 64 pins - up to 184 segments
  - 80 pins - up to 192 segments
- 多種 LCD 時脈選擇
- 內建 LCD 偏壓產生器
- 可支援 3V & 5V 的玻璃
- 反襯度使用軟體調整
- 低功耗

LCD 驅動模組

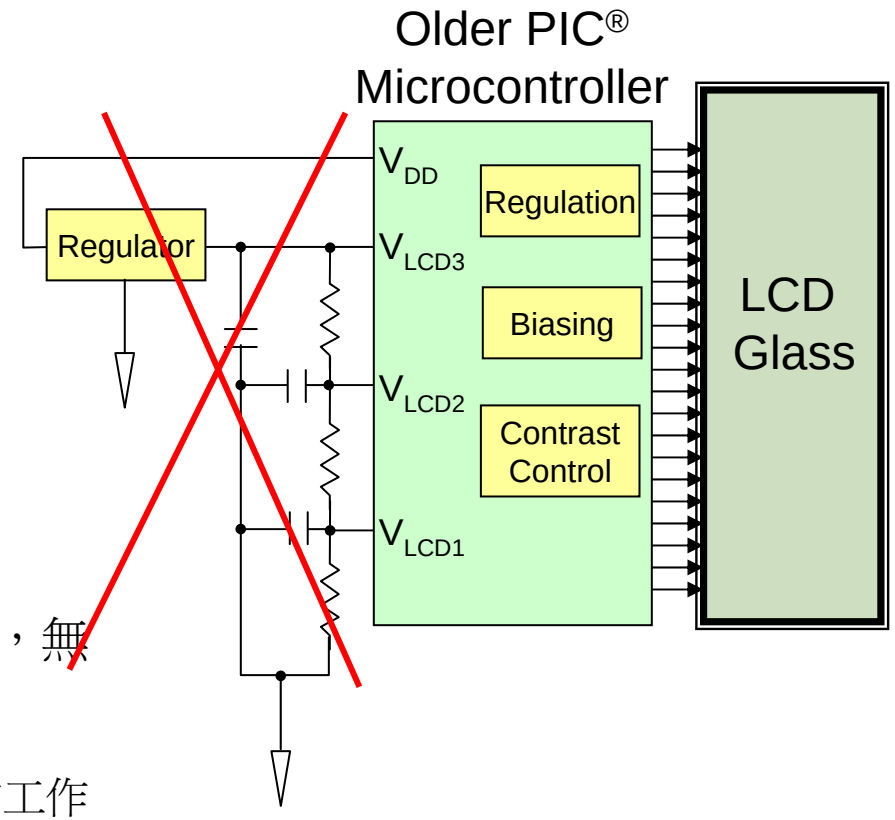
驅動的點數 = **Segments x Common**

PIC16F/LF1933/1936/1938	PIC16F/LF1934/1937/1939	PIC16F/LF1946/1947
15 x 4 = 60	**24 x 4 = 96**	**46 x 4 = 184**

- **Drives a number of LCD types:**
  - Static (1 common)
  - 1/2 multiplex (2 commons)
  - 1/3 multiplex (3 commons)
  - 1/4 multiplex (4 commons)

LCD 驅動模組

- 使用內建元件設計減少使用外部元件降低成本
  - 內建 LCD 偏壓設計
    - 無需使用外部零件做偏壓輸入
    - 3 種電流/功耗的設定
  - 內建反襯度控制
    - 無需使用外部零件調整
    - 7 種反襯度調整
  - 3V LCD 穩壓器
    - 無需使用外部 3V 穩壓器
    - 可以使用 5V 或 3V 的切換設計，無須外部零件
    - 使用內建的固定參考電壓：
      - ✓ 即使 V_{DD} 有漂移也可以穩定的工作



LCD Module 暫存器

- LCDCON : LCD 控制暫存器
- LCDPS　: LCD 相位暫存器
- LCDREF : LCD 參考電壓控制暫存器
- LCDCST : LCD 反襯度控制暫存器
- LCDSEn : LCD 節點驅動腳位選擇暫存器
  - 此暫存器之數目視元件而定，**PIC16F1937** 有 **24 Segments** 固有三個此類的暫存器
- LCDDATAn : LCD 顯示資料暫存器
  - **PIC16F1937** 有 **12** 個，**LCDDATA0 ~ LCDDATA11**
  - **3 Segment (Seg.0 ~ Seg. 23) x 4 Common**
- LCDRL : LCD Reference Ladder register

LCDSEn 節點腳位選擇暫存器

- **LCDSE0 ~ LCDSE2 暫存器**

LCDSE0

SE7	SE6	SE5	SE4	SE3	SE2	SE1	SE0

LCDSE1

SE15	SE14	SE13	SE12	SE11	SE10	SE9	SE8

LCDSE2

SE23	SE22	SE21	SE20	SE19	SE18	SE17	SE16

位元	功能
SEn	1 = Segment function of the pin is enabled 0 = I/O function of the pin is enabled

LCDCON 控制暫存器

- **LCDCON 控制暫存器**

LCDCON

LCDEN	SLPEN	WERR	---	CS1	CS0	LMUX1	LMUX0

位元	功能

LCDEN	1 = LCD 模組啓用 0 = LCD 模組關閉		
CS1 **CS0**	**LCD 時脈選擇**	00 = Fosc/256 1x = LFINTOSC (31KHz)	01 = T1OSC (Timer1)
LMUX1 **LMUX0**	**Commons 選擇**	00 = Static (COM0) 10 = 1/3 (COM<2:0>)	01 =1/2 (COM<1:0>) 11 =1/4 (COM<3:0>)

範例程式的設定值：0x8F

LCDPS 相位暫存器

- **LCDPS 暫存器**

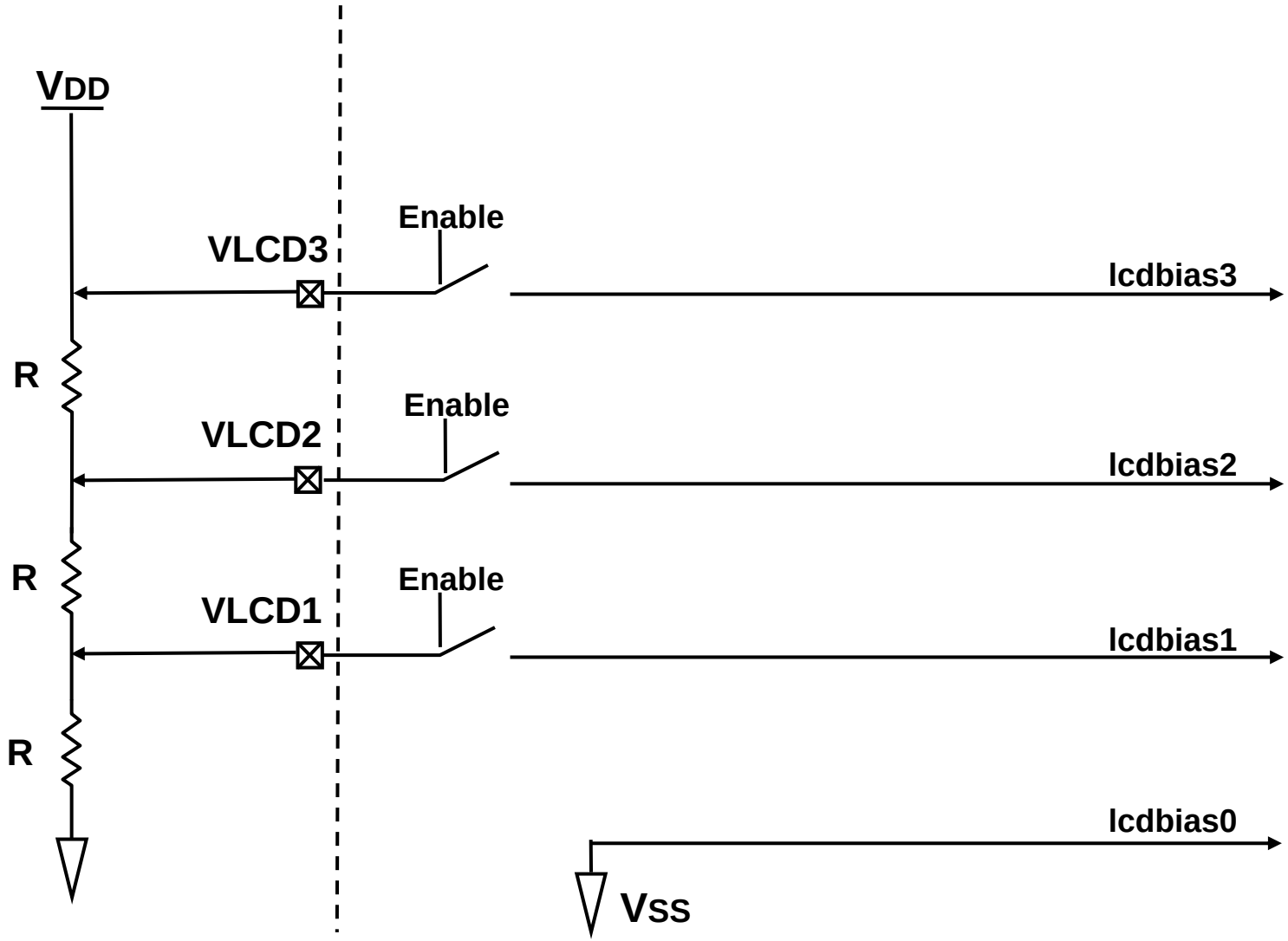
LCDPS		R-0/0	R-0/0				
EFT	BIASMD	LCDA	WA	LP3	LP2	LP1	LP0

位元	功能

EFT	**1 = Type-B Phase** **0 = Type-A Phase**
LCDA **WA**	**LCD 動作指示位元** **0 = Inactive** **1 = Active** **LCDDATAn 允許寫入指示位元** **1 = 允許資料寫入**
LP3 **LP0**	**4-bit 預除器 (1:1 ~ 1:16)**

範例程式的設定值：0x00

傳統使用外部的 LCD 偏壓電路



LCDREF 參考電壓控制暫存器

- **LCDREF 暫存器**

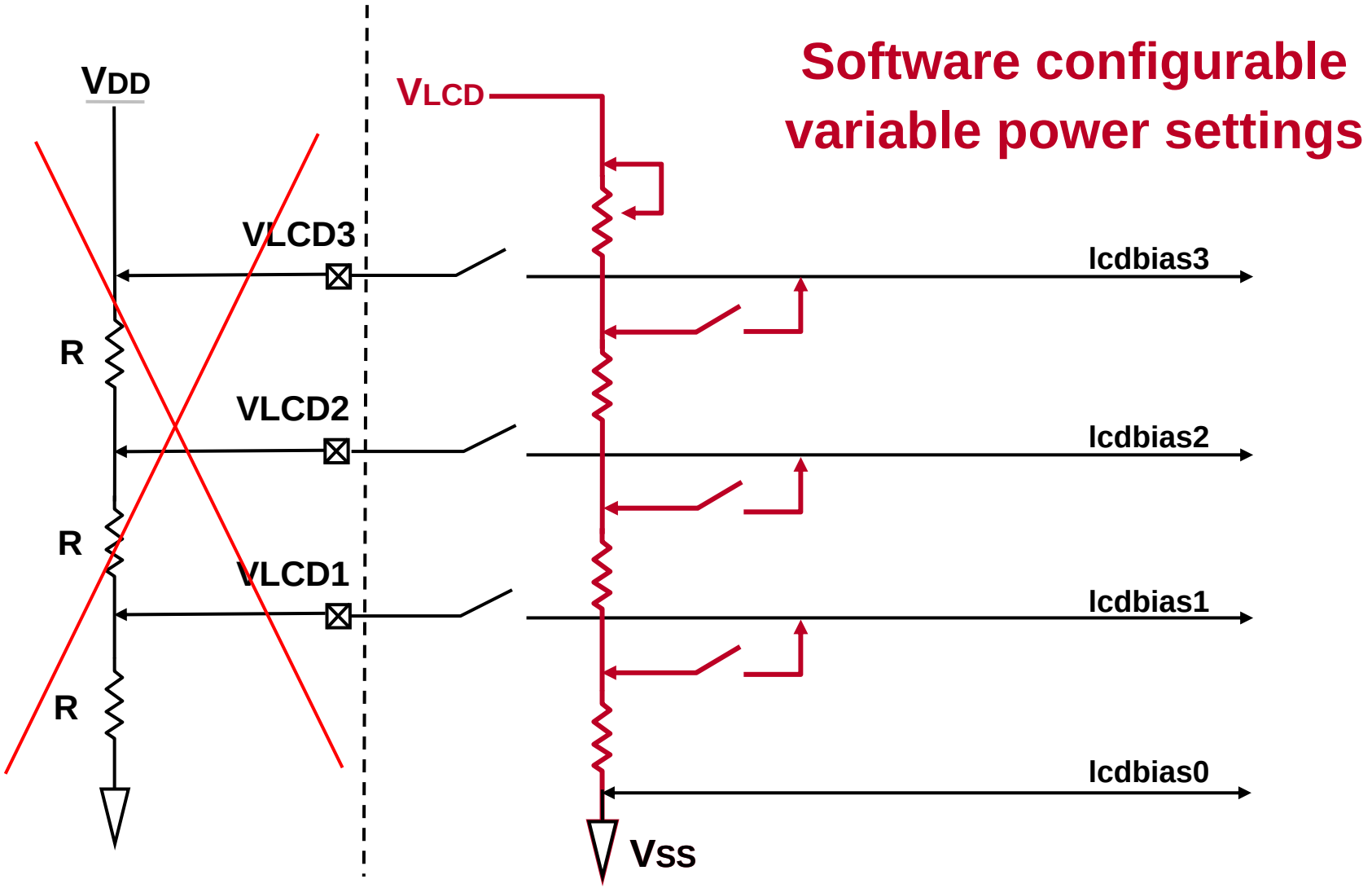
LCDREF

LCDIRE	LCDIRES	LCDIRI	---	VLCD3PE	VLCD2PE	VLCD1PE	---

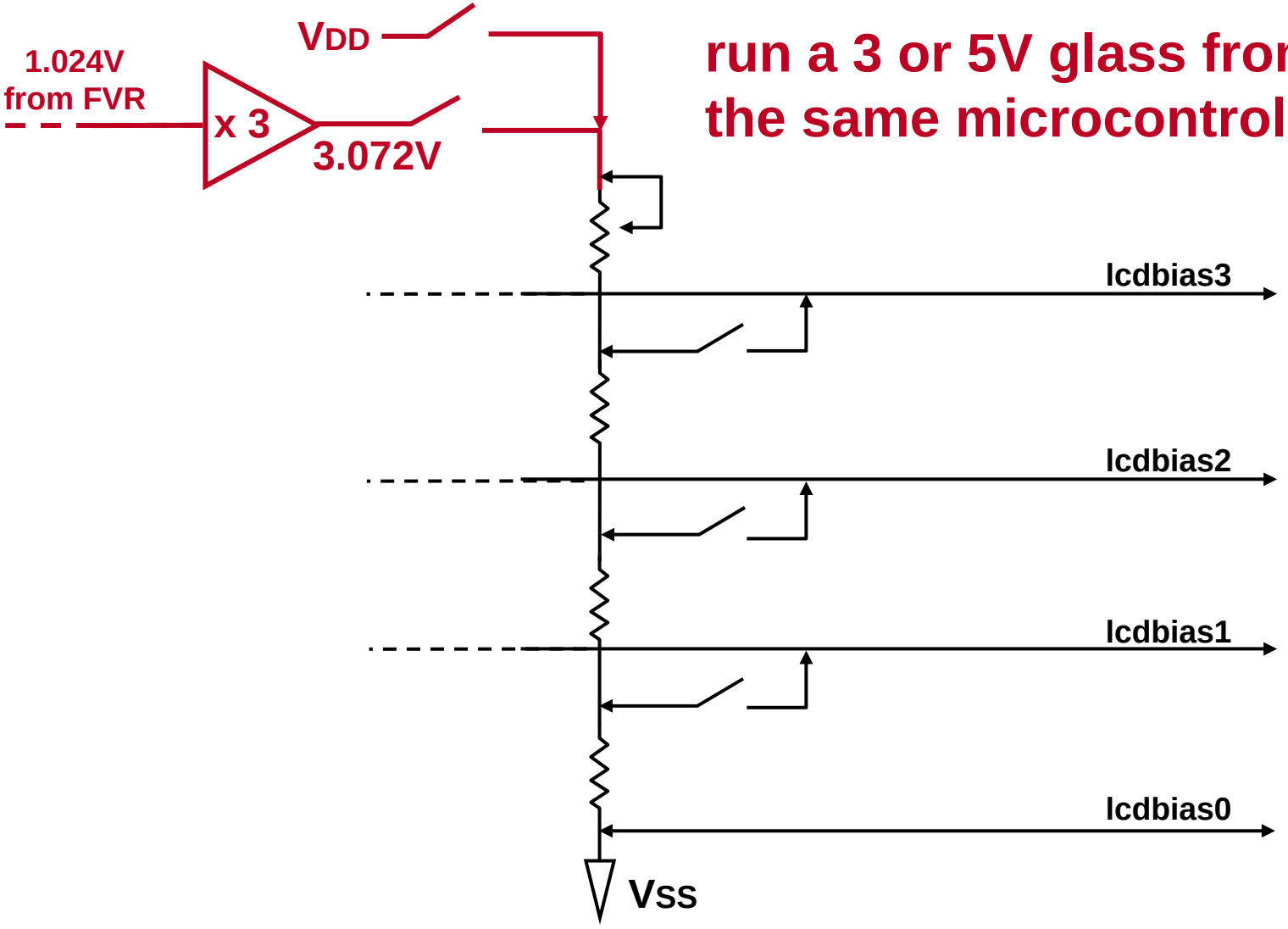
位元	功能
LCDIRE	**1 =** Internal LCD Reference is enabled and connected to the Internal Contrast Control circuit **0 =** Internal LCD Reference is disabled
LCDIRES If LCDIRE = 1:	0 = Internal LCD Contrast Control is powered by VDD 1 = Internal LCD Contrast Control is powered by a 3.072V output of the FVR.
VLCD3PE **VLCD1PE**	1 = The VLCDx pin is connected to the internal bias voltage LCDBIASx 0 = The VLCDx pin is not connected (使用內部偏壓)

範例程式的設定值：0xC0

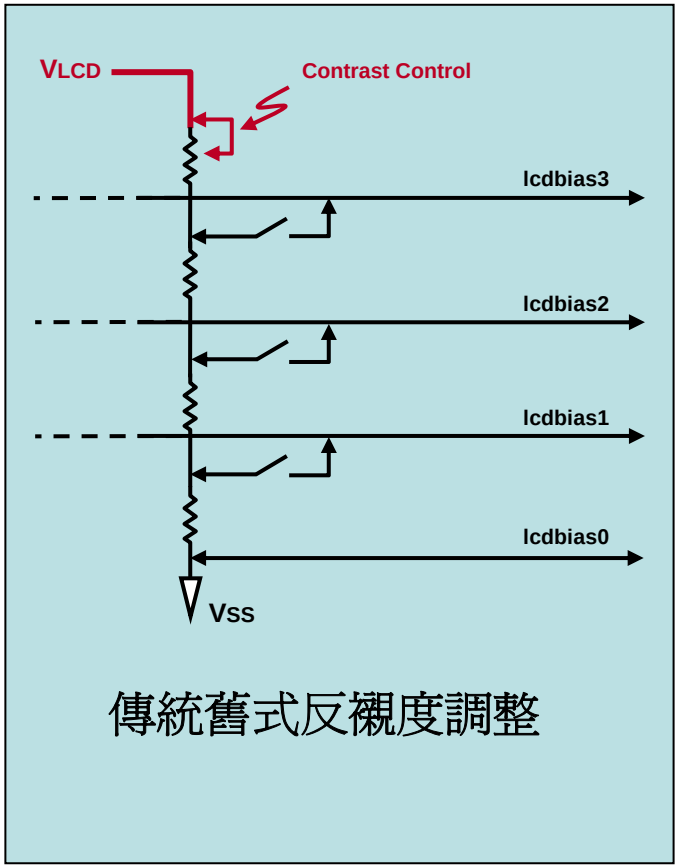
使用內建的 LCD 偏壓設計



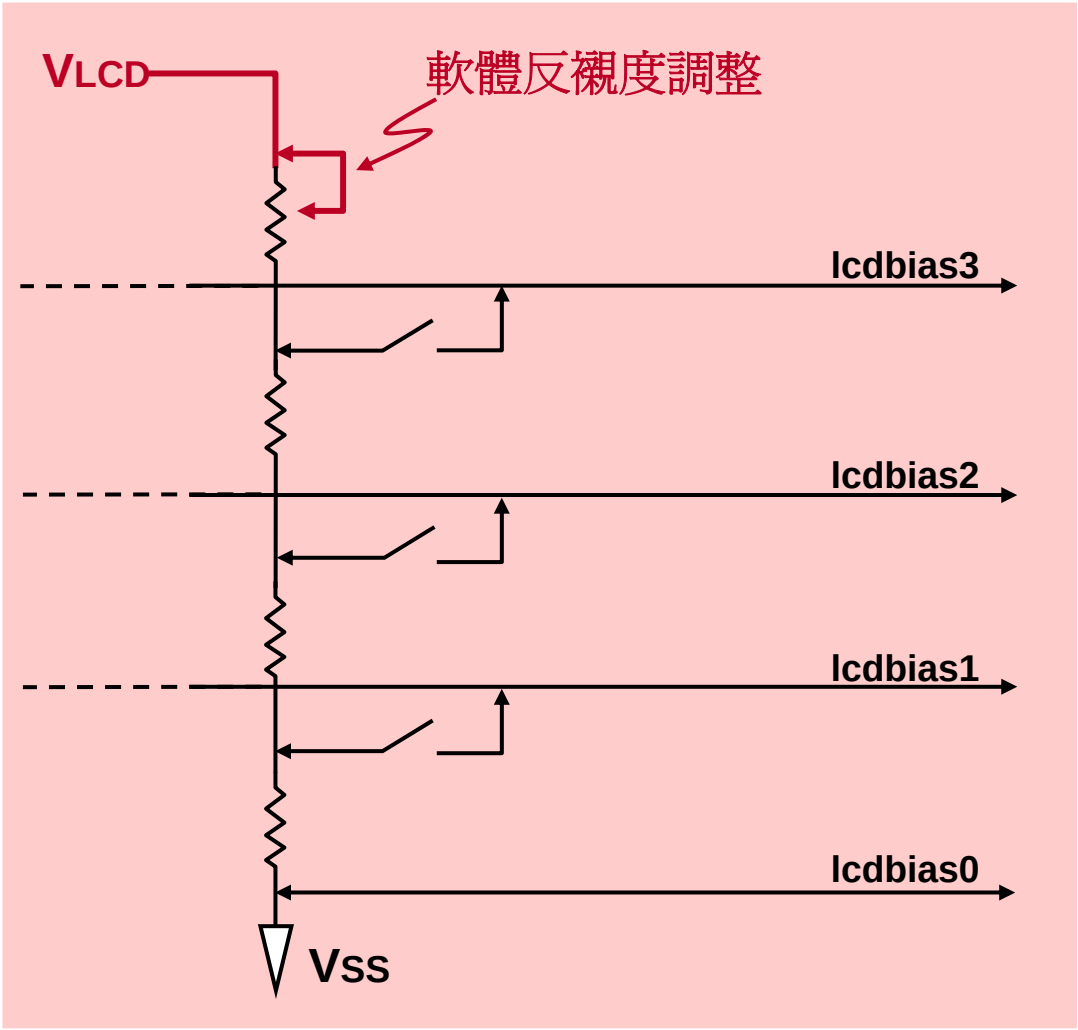
LCD 偏壓產生電路


VDD
1.024V
from FVR
x 3
3.072V
run a 3 or 5V glass from
the same microcontroller
lcdbias3
lcdbias2
lcdbias1
lcdbias0
Vss

反襯度控制電路



傳統舊式反襯度調整



LCDCST 反襯度控制暫存器

- **LCDREF 暫存器**

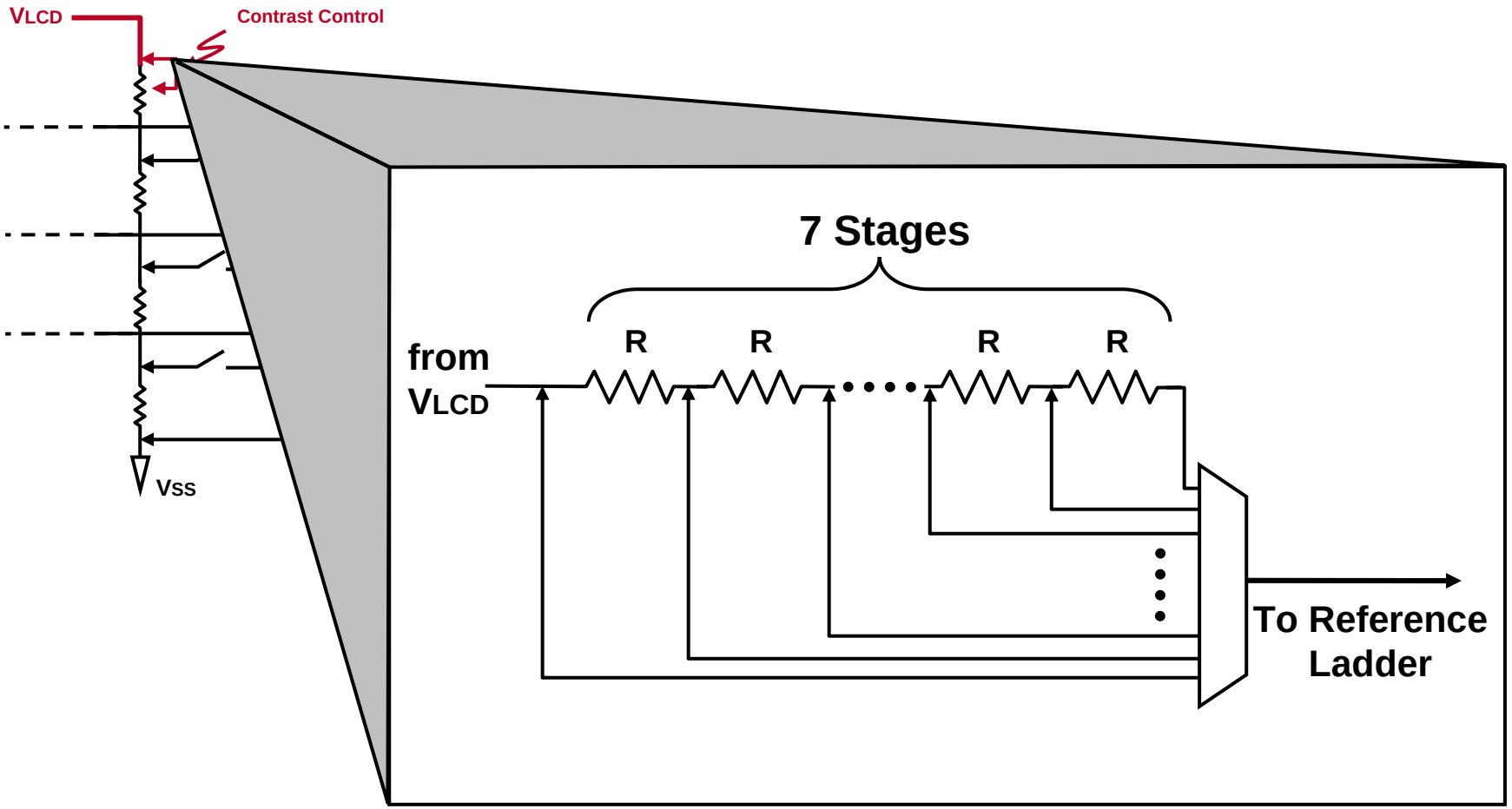
LCDREF

---	---	---	---	---	LCDCST2	LCDCST1	LCDCST0

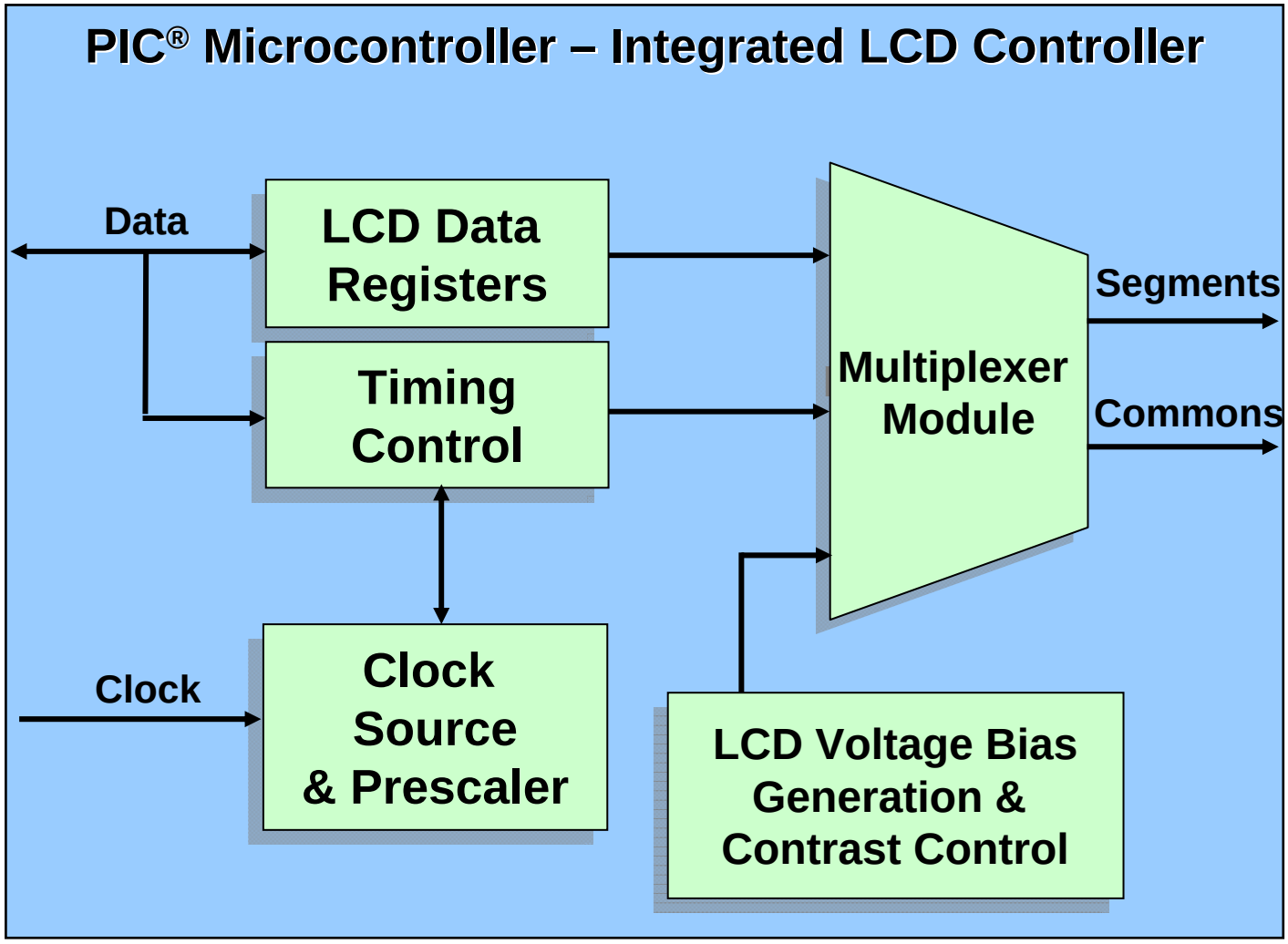
位元	功能
LCDCST2 **LCDCST0**	Selects the resistance of the LCD contrast control resistor ladder

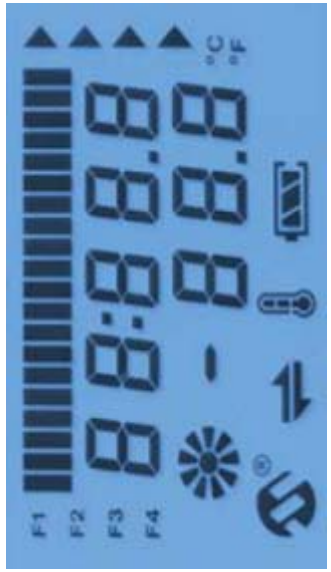
範例程式的設定值：0x00

軟體調整反襯度功能


VLCD
Contrast Control
Vss
7 Stages
from VLCD
R
R
R
R
To Reference Ladder

LCD Controller Block Diagram

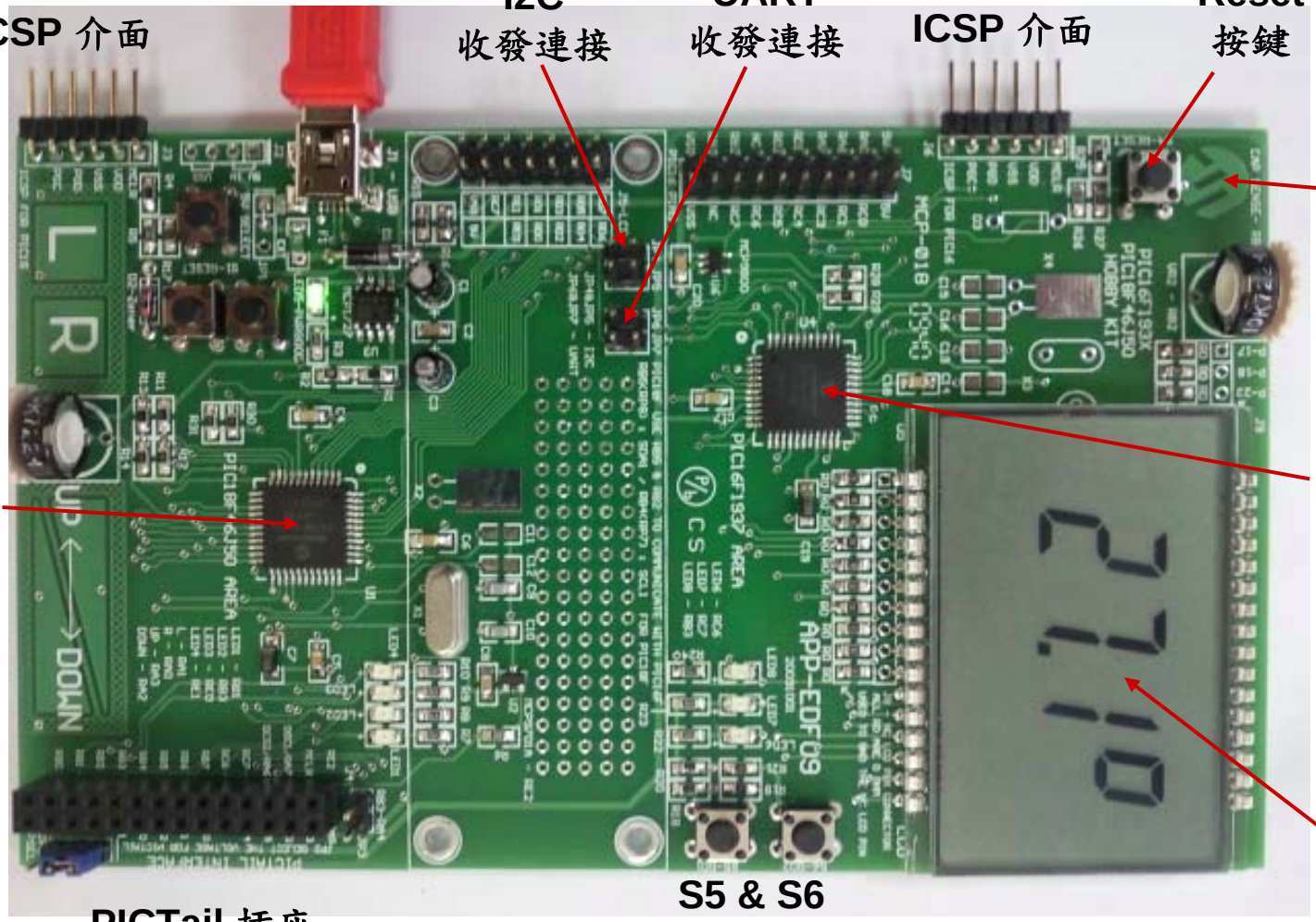




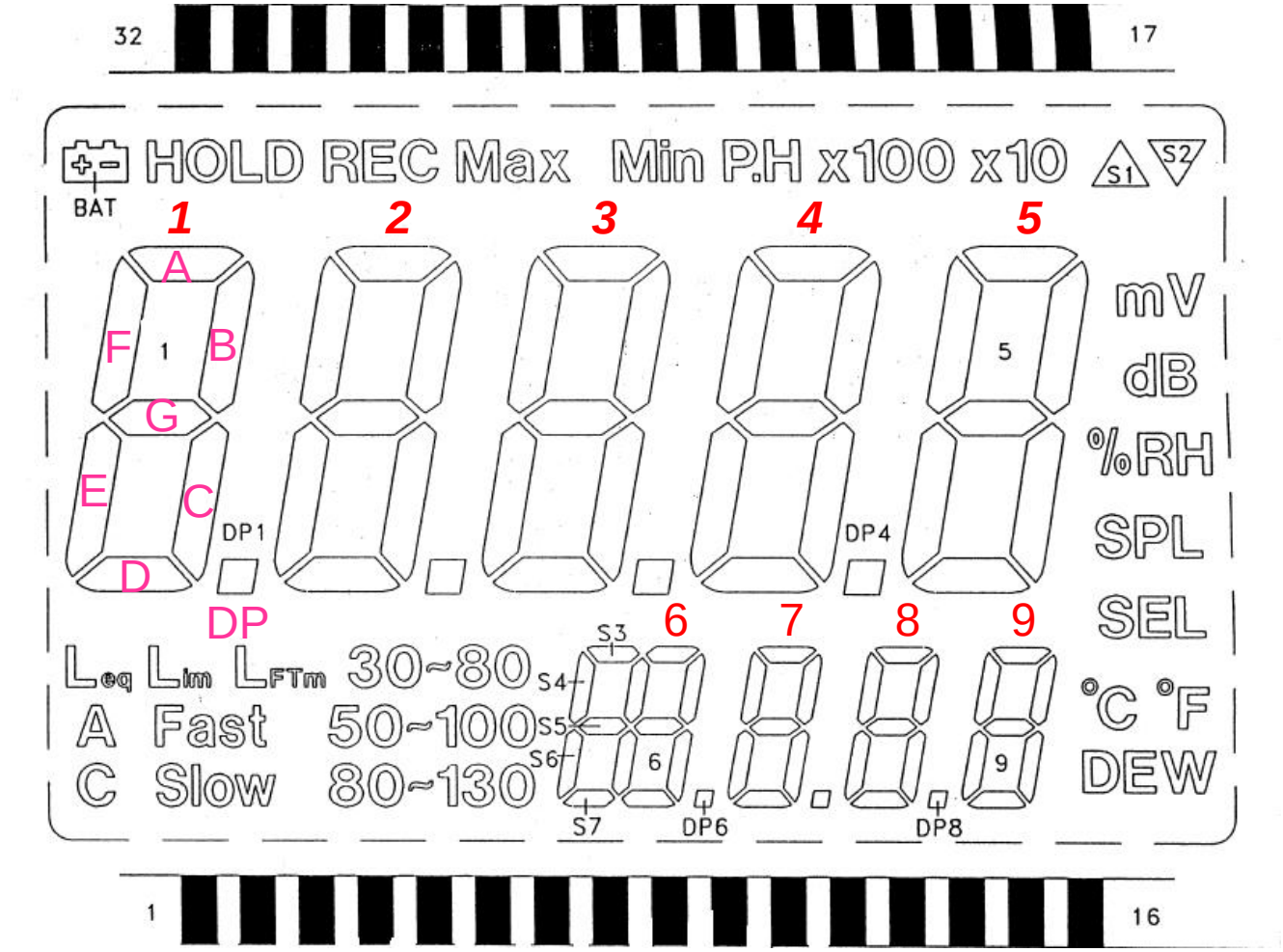
APP-EDF09 實驗板

- **APP-EDF09**
  - PIC16F1937 與 PIC18F46J50 兩個區塊
- **PIC16F1937 區塊 (使用 USB – J1 提供的 5V 工作)**
  - 直接驅動 Segment LCD Glass Display
  - 三個按鍵 (RESET * 1 以及兩個數位按鍵)
  - LED * 3
  - VR * 1 以便實習 ADC 的功能
  - MCP9800 溫度感測器 * 1，與 PIC16F1937 使用 I2C 界面溝通
  - 可以用兩組 Jumper 選擇用 I2C 或 UART 與 PIC18F46J50 通信
  - 保留的 Primary & Secondary Oscillator 零件位置 (X3 & X4)
  - 20-Pin 的擴充界面，可將 PIC16F1937 的許多信號引出以便測試
  - 電容式感測按鍵 * 1 (位於 Microchip Logo)
  - Microchip 臺灣網站提供使用手冊、電路圖及測試程式
    - http://www.microchip.com.tw/Taiwan_CAE/EVM_Pages/Index1.htm

APP-EDF09 實驗板



APP-EDF09 LCD 配置



- **3V** 工作電壓
- 工作溫度：
  - -10°C ~ 70°C
- 驅動方式：
  - ¼ Duty
  - 1/3 Bias
- 視野：**6** 點鐘方向
- **TN Type**

LCD 顯示腳位與 PIC 腳位
PIC16F1937 44-pin/TQFP

LCD PIN	*COM0*	*COM1*	*COM2*	*COM3*	*PIC16F1937 Segment*	*PIC16F1937 Pin*
1	/	Leq	Lim	LFTm	N.C.	N.C.
2	S7	S6	S5	S4	N.C.	N.C.
3	6D	6E	6G	6F	N.C.	N.C.
4	6DP	6C	6B	6A	N.C.	N.C.
5	7D	7E	7G	7F	N.C.	N.C.
6	7DP	7C	7B	7A	N.C.	N.C.
7	8D	8E	8G	8F	N.C.	N.C.
8	8DP	8C	8B	8A	N.C.	N.C.
9	9D	9E	9G	9F	N.C.	N.C.
10	/	9C	9B	9A	N.C.	N.C.
11	/	F	C	S3	SEG. 21	PIN 25
12	C	A	Slow	Fast	SEG. 5	PIN 24
13	COM0	/	/	/	COM0	PIN 14
14	/	COM1	/	/	COM1	PIN 15
15	/	/	COM2	/	COM2	PIN 21
16	/	/	/	COM3	COM3	PIN 38

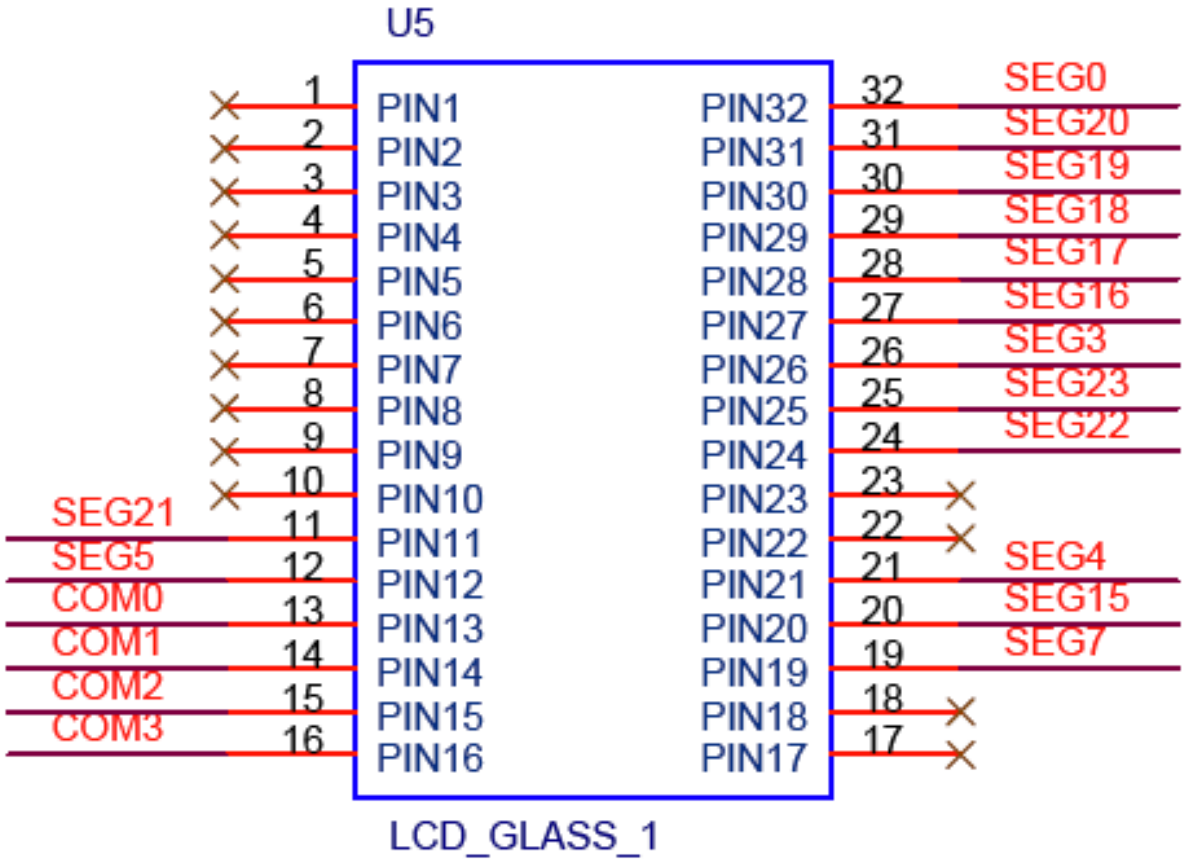
LCD 顯示腳位與 PIC 腳位

PIC16F1937 44-pin/TQFP

接上一頁

17	DEW	80~130	50~100	30~80	N.C.	N.C.
18	SEL	SPL	dB	%RH	N.C.	N.C.
19	m	V	S2	S1	SEG. 7	PIN 20
20	/	5C	5B	5A	SEG. 15	PIN 22
21	5D	5E	5G	5F	SEG. 4	PIN 23
22	/	/	/	/	N.C.	N.C.
23	P.H	x10	x100	Min	N.C.	N.C.
24	4DP	4C	4B	4A	SEG. 22	PIN 26
25	4D	4E	4G	4F	SEG. 23	PIN 27
26	3DP	3C	3B	3A	SEG. 3	PIN 36
27	3D	3E	3G	3F	SEG. 16	PIN 41
28	Max	REC	HOLD	BAT	SEG. 17	PIN 2
29	2DP	2C	2B	2A	SEG. 18	PIN 3
30	2D	2E	2G	2F	SEG. 19	PIN 4
31	1DP	1C	1B	1A	SEG. 20	PIN 5
32	1D (範例)	1E	1G	1F	SEG. 0	PIN 9

PIC16F1937 推動 LCD 電路圖



LCD 空腳部分都有接 0 ohm 電阻接地，以避免LCD 腳位浮接產生錯誤顯示。

LCDDATAx 顯示資料暫存器

- **LCDDATAx 暫存器 (PIC16F1937 有 12 個 LCDDATA 暫存器，可推動 96 個 Segments)**

LCDDATAx (LCDDATA0 ~ LCDDATA11)

SEGx-COMy	SEGx-COMy	SEGx-COMy	SEGx-COMy	SEGx-COMy	SEGx-COMy	SEGx-COMy	SEGx-COMy

位元	功能
SEGx-COMy	1 = Pixel on (dark) 0 = Pixel off (clear)

LCD 顯示節的對應表

LCD Function	COM0		COM1		COM2		COM3	
	LCDDATAx Address	LCD Segment	LCDDATAx Address	LCD Segment	LCDDATAx Address	LCD Segment	LCDDATAx Address	LCD Segment
SEG0	LCDDATA0, 0	1D	LCDDATA3, 0	1E	LCDDATA6, 0	1G	LCDDATA9, 0	1F
SEG1	LCDDATA0, 1		LCDDATA3, 1		LCDDATA6, 1		LCDDATA9, 1	
SEG2	LCDDATA0, 2		LCDDATA3, 2		LCDDATA6, 2		LCDDATA9, 2	
SEG3	LCDDATA0, 3	3DP	LCDDATA3, 3	3C	LCDDATA6, 3	3B	LCDDATA9, 3	3A
SEG4	LCDDATA0, 4	5D	LCDDATA3, 4	5E	LCDDATA6, 4	5G	LCDDATA9, 4	5F
SEG5	LCDDATA0, 5	C	LCDDATA3, 5	A	LCDDATA6, 5	Slow	LCDDATA9, 5	Fast
SEG6	LCDDATA0, 6		LCDDATA3, 6		LCDDATA6, 6		LCDDATA9, 6	
SEG7	LCDDATA0, 7	m	LCDDATA3, 7	V	LCDDATA6, 7	S2	LCDDATA9, 7	S1
SEG8	LCDDATA1, 0		LCDDATA4, 0		LCDDATA7, 0		LCDDATA10, 0	
SEG9	LCDDATA1, 1		LCDDATA4, 1		LCDDATA7, 1		LCDDATA10, 1	
SEG10	LCDDATA1, 2		LCDDATA4, 2		LCDDATA7, 2		LCDDATA10, 2	
SEG11	LCDDATA1, 3		LCDDATA4, 3		LCDDATA7, 3		LCDDATA10, 3	
SEG12	LCDDATA1, 4		LCDDATA4, 4		LCDDATA7, 4		LCDDATA10, 4	
SEG13	LCDDATA1, 5		LCDDATA4, 5		LCDDATA7, 5		LCDDATA10, 5	
SEG14	LCDDATA1, 6		LCDDATA4, 6		LCDDATA7, 6		LCDDATA10, 6	
SEG15	LCDDATA1, 7		LCDDATA4, 7	5C	LCDDATA7, 7	5B	LCDDATA10, 7	5A
SEG16	LCDDATA2, 0	3D	LCDDATA5, 0	3E	LCDDATA8, 0	3G	LCDDATA11, 0	3F
SEG17	LCDDATA2, 1	Max	LCDDATA5, 1	REC	LCDDATA8, 1	HOLD	LCDDATA11, 1	BAT
SEG18	LCDDATA2, 2	2DP	LCDDATA5, 2	2C	LCDDATA8, 2	2B	LCDDATA11, 2	2A
SEG19	LCDDATA2, 3	2D	LCDDATA5, 3	2E	LCDDATA8, 3	2G	LCDDATA11, 3	2F
SEG20	LCDDATA2, 4	1DP	LCDDATA5, 4	1C	LCDDATA8, 4	1B	LCDDATA11, 4	1A
SEG21	LCDDATA2, 5		LCDDATA5, 5	°F	LCDDATA8, 5	°C	LCDDATA11, 5	S3
SEG22	LCDDATA2, 6	4DP	LCDDATA5, 6	4C	LCDDATA8, 6	4B	LCDDATA11, 6	4A
SEG23	LCDDATA2, 7	4D	LCDDATA5, 7	4E	LCDDATA8, 7	4G	LCDDATA11, 7	4F

思考一下軟體要怎樣寫

- 我知道了 **LCD** 與 **PIC** 接腳的關係
  - 用那一個 Segment 與 COMx 去驅動那一節 LCD
- 由 **LCD** 顯示節的對應表我也知道了每一個 **LCD** 顯示節對應到 **PIC** 內部的顯示 **RAM (LCDDATAx.bit)**
- **LCDDATAx.bit** 與 **SEGx & COMx** 的關連?
- 我要如何定義這些 **LCDDATAx.bit** 的宣告讓程式容易驅動？
  - Hi-Tech PICC 已經定義了
  - 路徑 C:\Program Files\HI-TECH Software\PICC\9.82\include\pic16f1937.h

Lab4
pic16f1937.h LCD 宣告

Hi-Tech PICC 對 LCD Segment 的宣告方式採絕對位址(@方式)
將每一個顯示節的位元都加以宣告以方便程式的撰寫。

```
volatile unsigned char       LCDDATA0        @ 0x7A0;
// bit and bitfield definitions
volatile bit SEG0COM0        @ ((unsigned)&LCDDATA0*8)+0;
volatile bit SEG1COM0        @ ((unsigned)&LCDDATA0*8)+1;
volatile bit SEG2COM0        @ ((unsigned)&LCDDATA0*8)+2;
volatile bit SEG3COM0        @ ((unsigned)&LCDDATA0*8)+3;
volatile bit SEG4COM0        @ ((unsigned)&LCDDATA0*8)+4;
volatile bit SEG5COM0        @ ((unsigned)&LCDDATA0*8)+5;
volatile bit SEG6COM0        @ ((unsigned)&LCDDATA0*8)+6;
volatile bit SEG7COM0        @ ((unsigned)&LCDDATA0*8)+7;
```

LAB4

顯示數字的定義檔 (LCD_Defs.h)

- 程式裡對各個顯示節再宣告一次，讓所要驅動的 **LCD** 顯示節一目了然。
- **SEG_1D = SEG0COM0 = LCDDATA0.0** 這三種寫法都是驅動 **LCD** 玻璃上的第一個大數字的 **D Segment**，那一種寫法比較清楚**?**

```
#define     SEG_1A      SEG20COM3
#define     SEG_1B      SEG20COM2
#define     SEG_1C      SEG20COM1
#define     SEG_1D      SEG0COM0
#define     SEG_1E      SEG0COM1
#define     SEG_1F      SEG0COM3
#define     SEG_1G      SEG0COM2
#define     SEG_1DP     SEG20COM0
```

LAB4

用程式控制 LCD

- **SEG_1D = 1; //** 點亮第一個數字的 **D Segment**
- **SEG_1D = 0; //** 熄滅第一個數字的 **D Segment**

- 程式中要顯示 **0** 的字型：
  SEG_1A = 1 ; SEG_1B = 1 ; SEG_1C = 1 ; SEG_1D = 1 ; SEG_1E = 1 ;
  SEG_1F = 1 ; SEG_1G = 0 ;
- 程式中要顯示 **1** 的字型：
  SEG_1A = 0 ; SEG_1B = 1 ; SEG_1C = 1 ; SEG_1D = 0 ; SEG_1E = 0 ;
  SEG_1F = 0 ; SEG_1G = 0 ;
- 程式中要顯示 **2** 的字型：
  SEG_1A = 1 ; SEG_1B = 1 ; SEG_1C = 0 ; SEG_1D = 1 ; SEG_1E = 1 ;
  SEG_1F = 0 ; SEG_1G = 1 ;

顯示數字的函數

```
void DisplayDigit3(int Show_Number)
{
  switch (Show_Number)
                {

                case 0:
                                SEG_3A = 1 ; SEG_3B = 1 ; SEG_3C = 1 ; SEG_3D = 1 ; SEG_3E = 1 ; SEG_3F = 1 ; SEG_3G = 0 ;
                                break ;
                case 1:
                                SEG_3A = 0 ; SEG_3B = 1 ; SEG_3C = 1 ; SEG_3D = 0 ; SEG_3E = 0 ; SEG_3F = 0 ; SEG_3G = 0 ;
                                break ;
                case 2:
                                SEG_3A = 1 ; SEG_3B = 1 ; SEG_3C = 0 ; SEG_3D = 1 ; SEG_3E = 1 ; SEG_3F = 0 ; SEG_3G = 1 ;
                                break ;
                case 3:
                                SEG_3A = 1 ; SEG_3B = 1 ; SEG_3C = 1 ; SEG_3D = 1 ; SEG_3E = 0 ; SEG_3F = 0 ; SEG_3G = 1 ;
                                break ;
                case 4:
                                SEG_3A = 0 ; SEG_3B = 1 ; SEG_3C = 1 ; SEG_3D = 0 ; SEG_3E = 0 ; SEG_3F = 1 ; SEG_3G = 1 ;
                                break ;
                case 5:
                                SEG_3A = 1 ; SEG_3B = 0 ; SEG_3C = 1 ; SEG_3D = 1 ; SEG_3E = 0 ; SEG_3F = 1 ; SEG_3G = 1 ;
                                break ;
                case 6:
                                SEG_3A = 1 ; SEG_3B = 0 ; SEG_3C = 1 ; SEG_3D = 1 ; SEG_3E = 1 ; SEG_3F = 1 ; SEG_3G = 1 ;
                                break ;
                case 7:
                                SEG_3A = 1 ; SEG_3B = 1 ; SEG_3C = 1 ; SEG_3D = 0 ; SEG_3E = 0 ; SEG_3F = 0 ; SEG_3G = 0 ;
                                break ;
                case 8:
                                SEG_3A = 1 ; SEG_3B = 1 ; SEG_3C = 1 ; SEG_3D = 1 ; SEG_3E = 1 ; SEG_3F = 1 ; SEG_3G = 1 ;
                                break ;
                  :
```

Lab4
Drive LCD Glass

- 專案位置：
  - .. \Advance PICC Application Labs\Lab4 Drive LCD Glass\Drive LCD Glass.mcp
- 專案的裡程式
  - LCD Test.c
  - APP-EDF09 LCD.c
  - LCD_Defs.h
  - pic16f1937.h
- **PIC16F1937 工作在 5V，爲何可直接趨動 3V LCD 玻璃？**

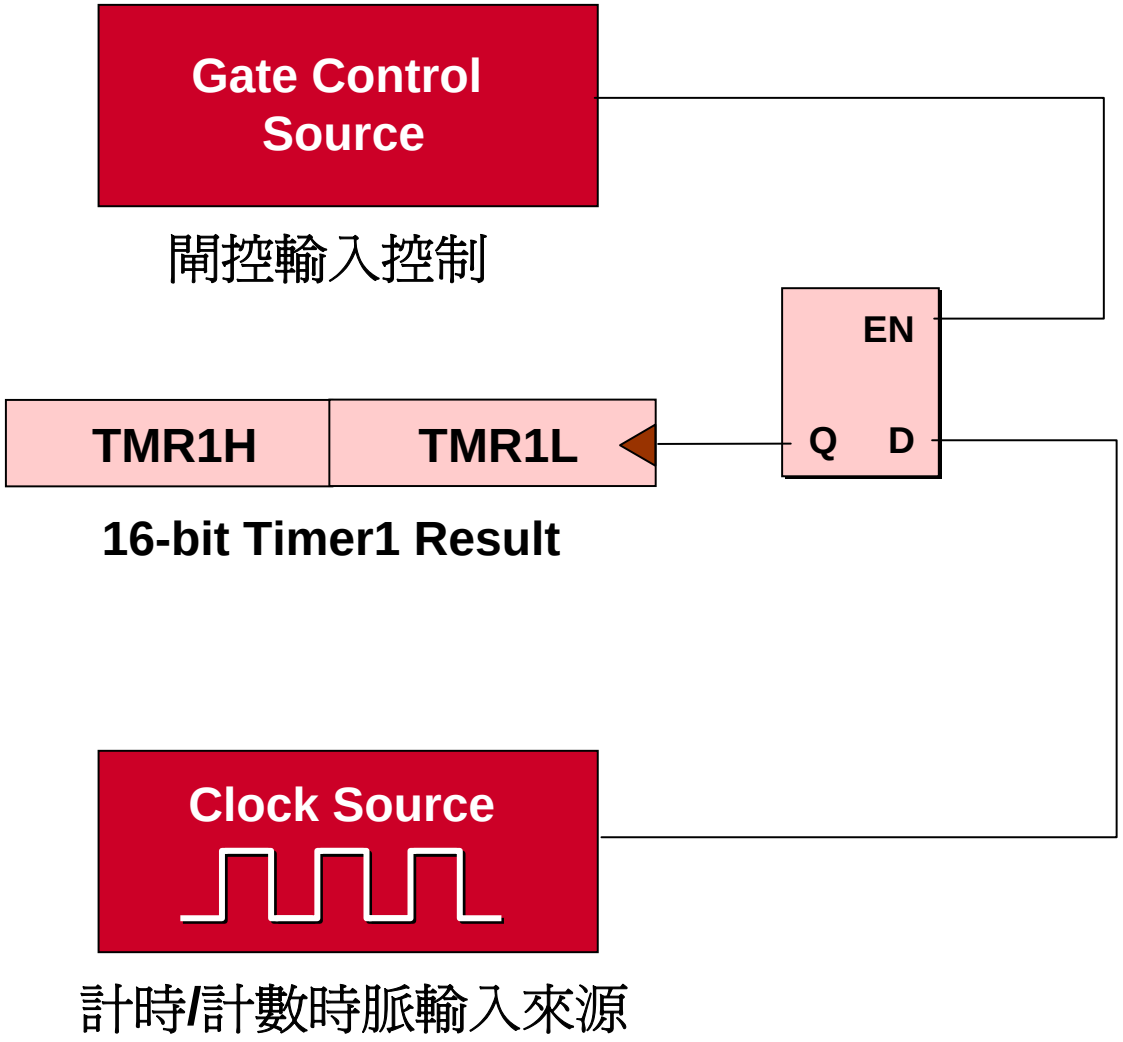
Lab4 Drive LCD Gless

- **PIC16F1937** 工作在 **5V**，直接趨動 **3V LCD** 玻璃
- 程式將使用 **Timer2 (1mS** 中斷**)** 來量測 **CSM** 電容觸控感應模組的震盪頻率，採移動式平均頻率值計算方式
  - 無接觸時：約 310KHz
  - 觸碰時：頻率開始將低，最低可達 100KHz
- 將所量測到的平均頻率顯示在 **LCD** 上
  - F=312（頻率顯示爲 321Khz）
- 展示：**APP-EDF009 + PICKit3 + MPLAB IDE v8.36 + Hi-Tech PICC PRO 9.65PL1.4612**

Timer1 Module
with Gate Control

Timer1 主體架構

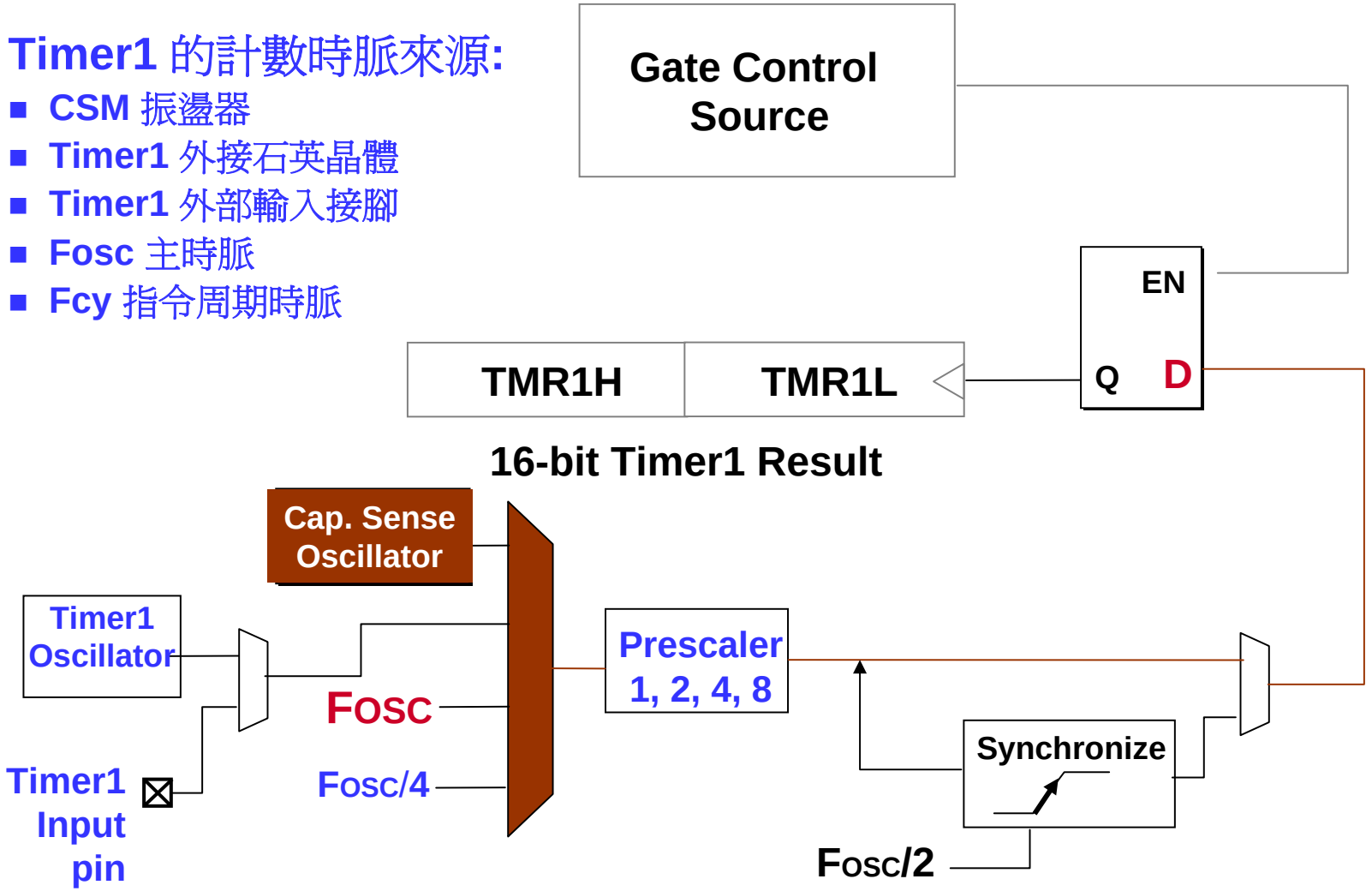
Gate Control + Clock Source



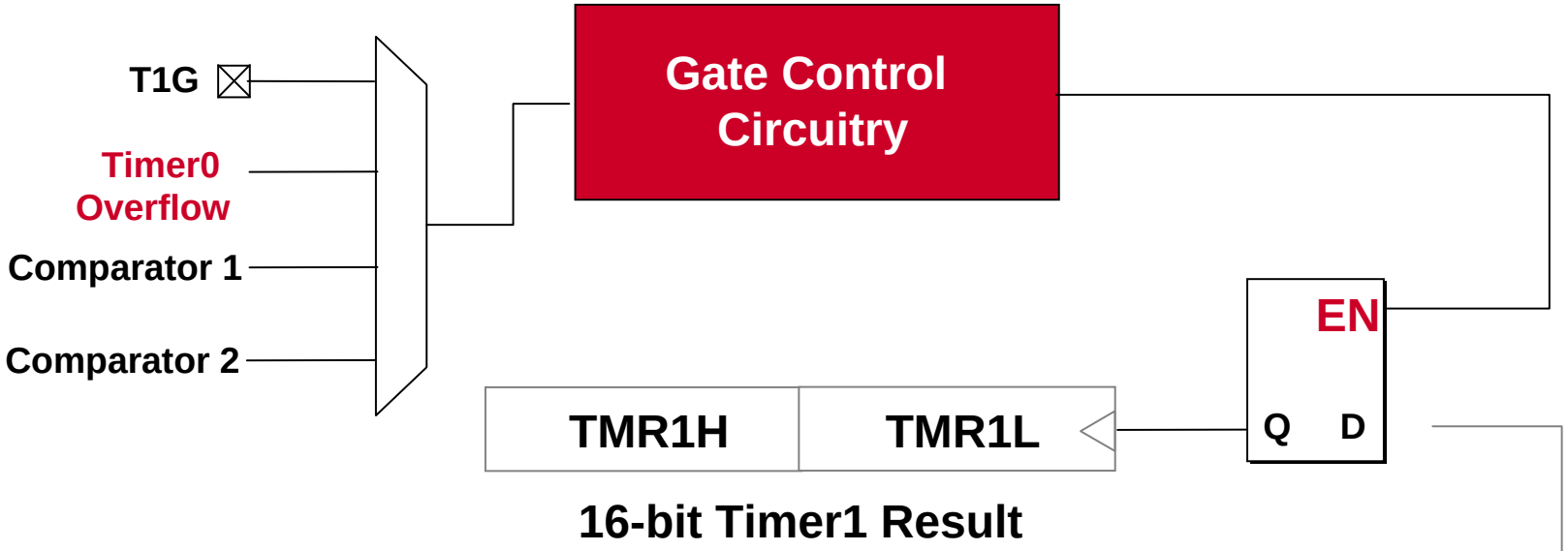
Timer1 計數時脈來源

Timer1 的計數時脈來源:

- CSM 振盪器
- Timer1 外接石英晶體
- Timer1 外部輸入接腳
- Fosc 主時脈
- Fcy 指令周期時脈



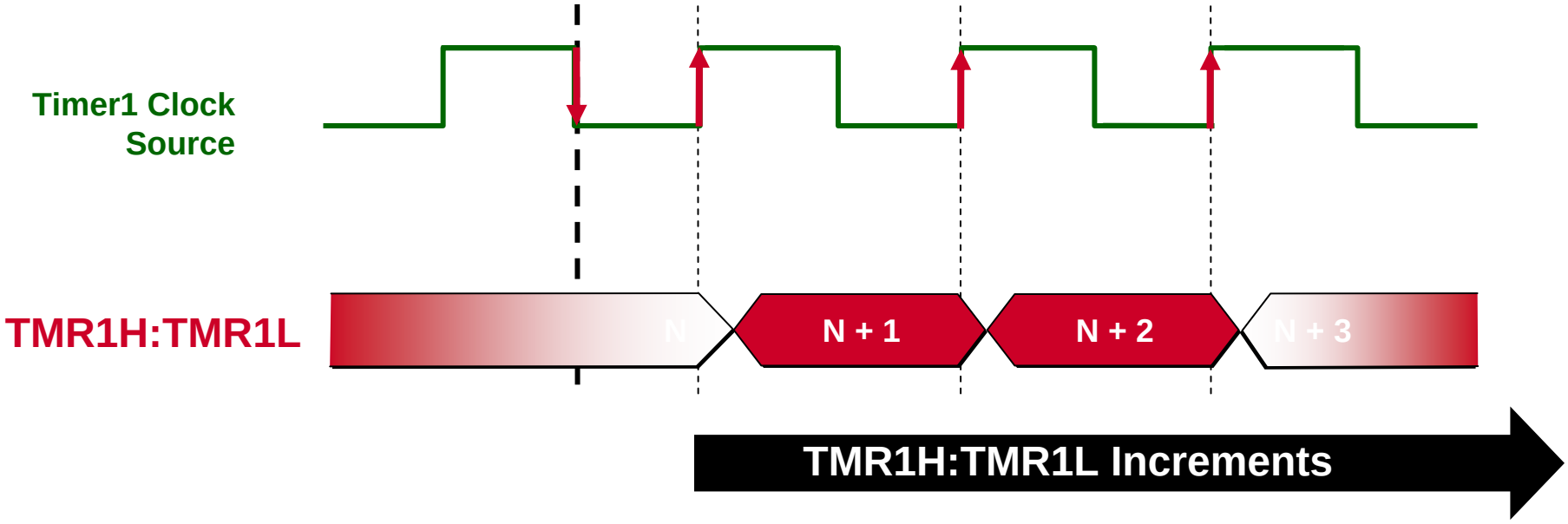
Timer1 閘控來源選擇



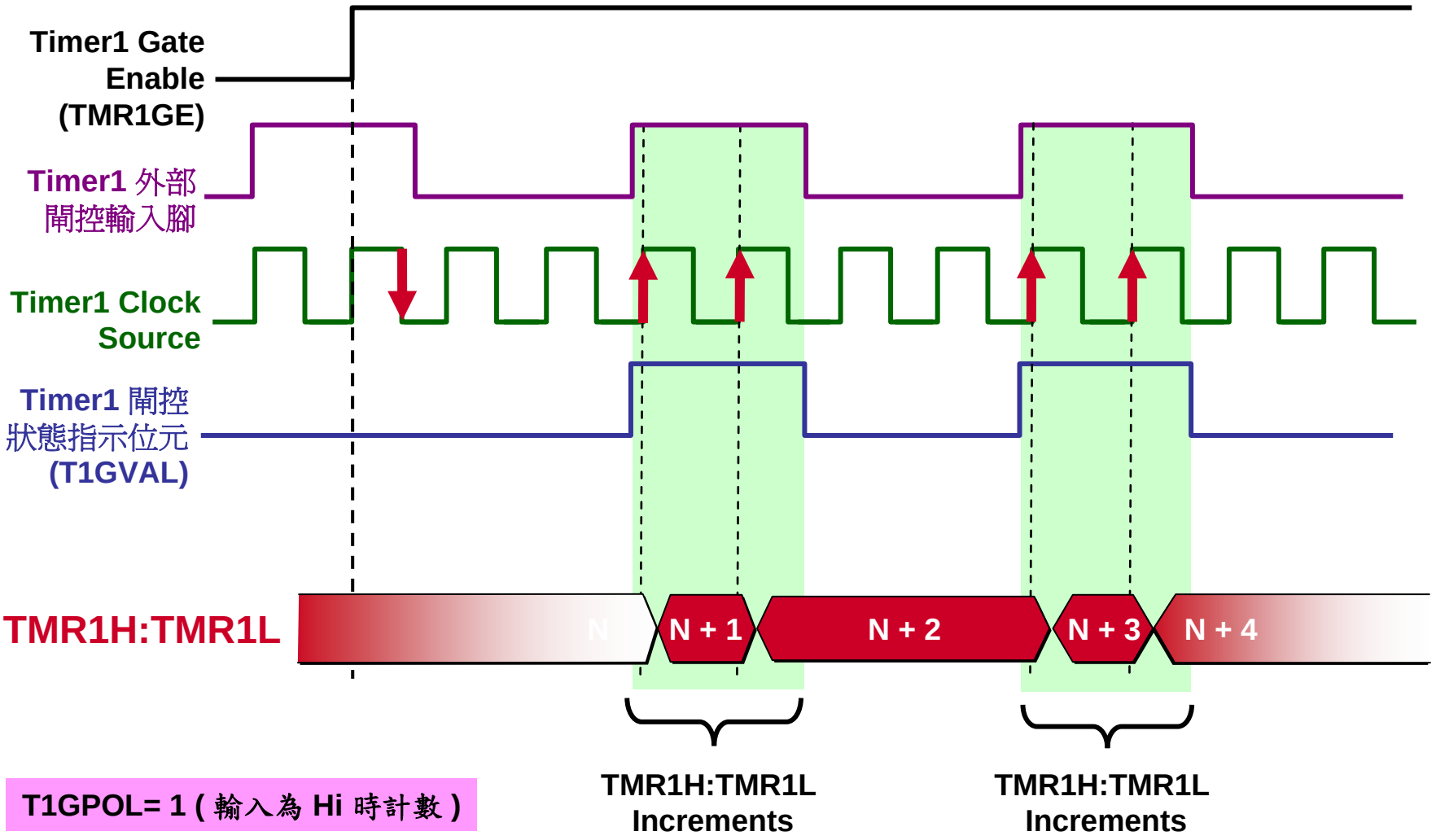
T1GSS<1:0>: Timer1 閘控來源選擇位元
00 = Timer1 閘控輸入腳
01 = Timer0 溢位
10 = 電壓比較器 1 (SYNCC1OUT)
11 = 電壓比較器 2 (SYNCC2OUT)

Timer1 Without Gate Control

- 不使用 **Timer1** 閘控的計時方式

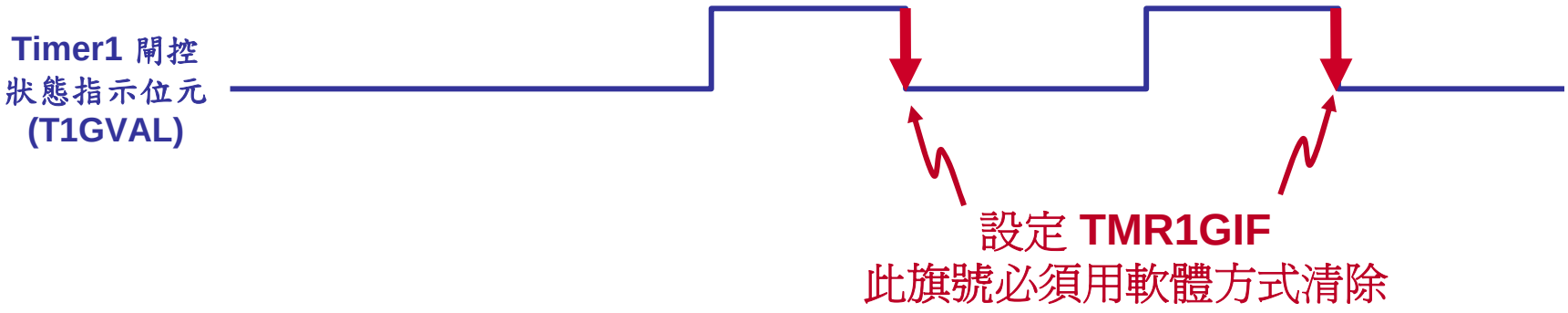


Timer1 閘控計時
(使用Timer1 外部值控輸入接腳)

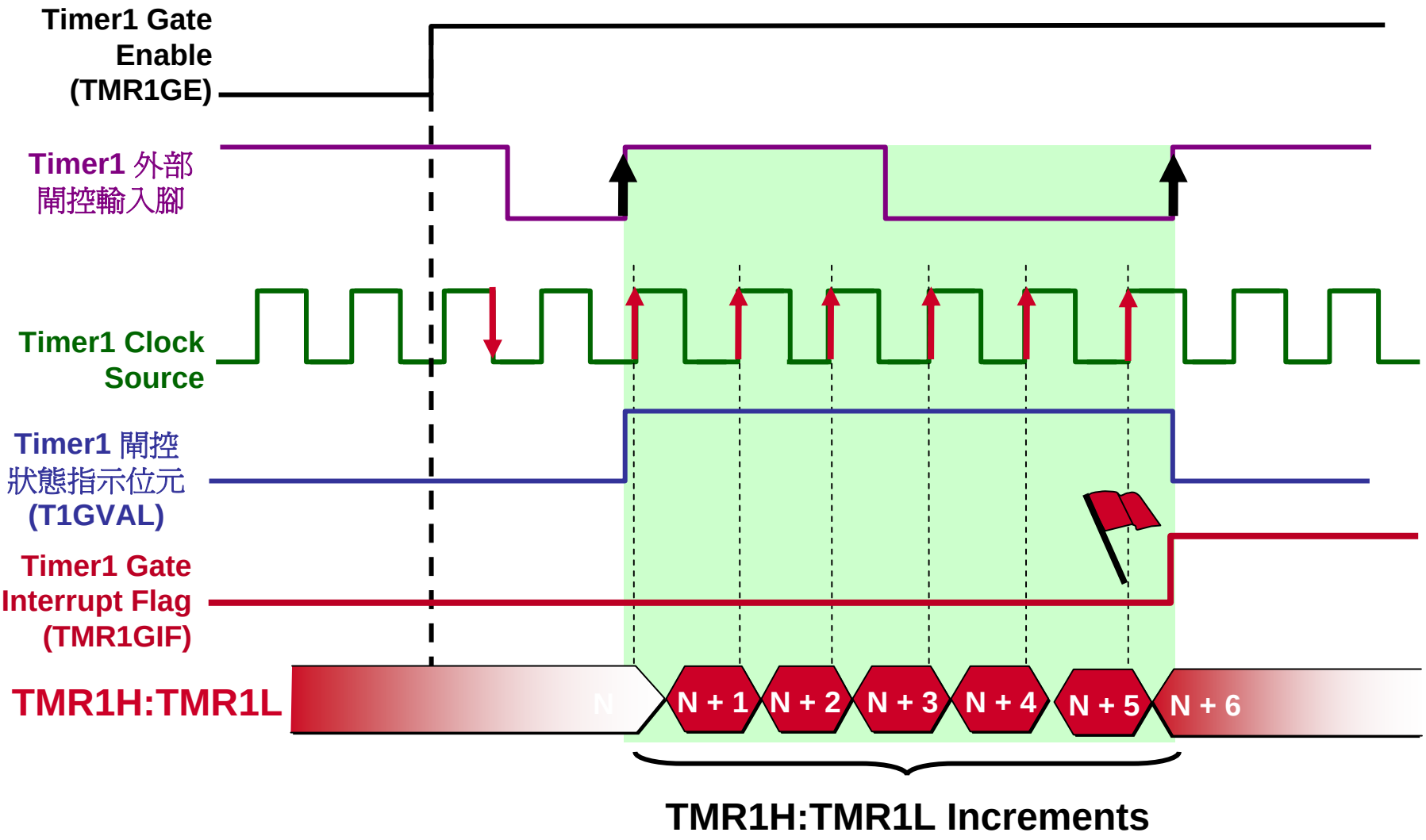


Timer1 狀態指示位元

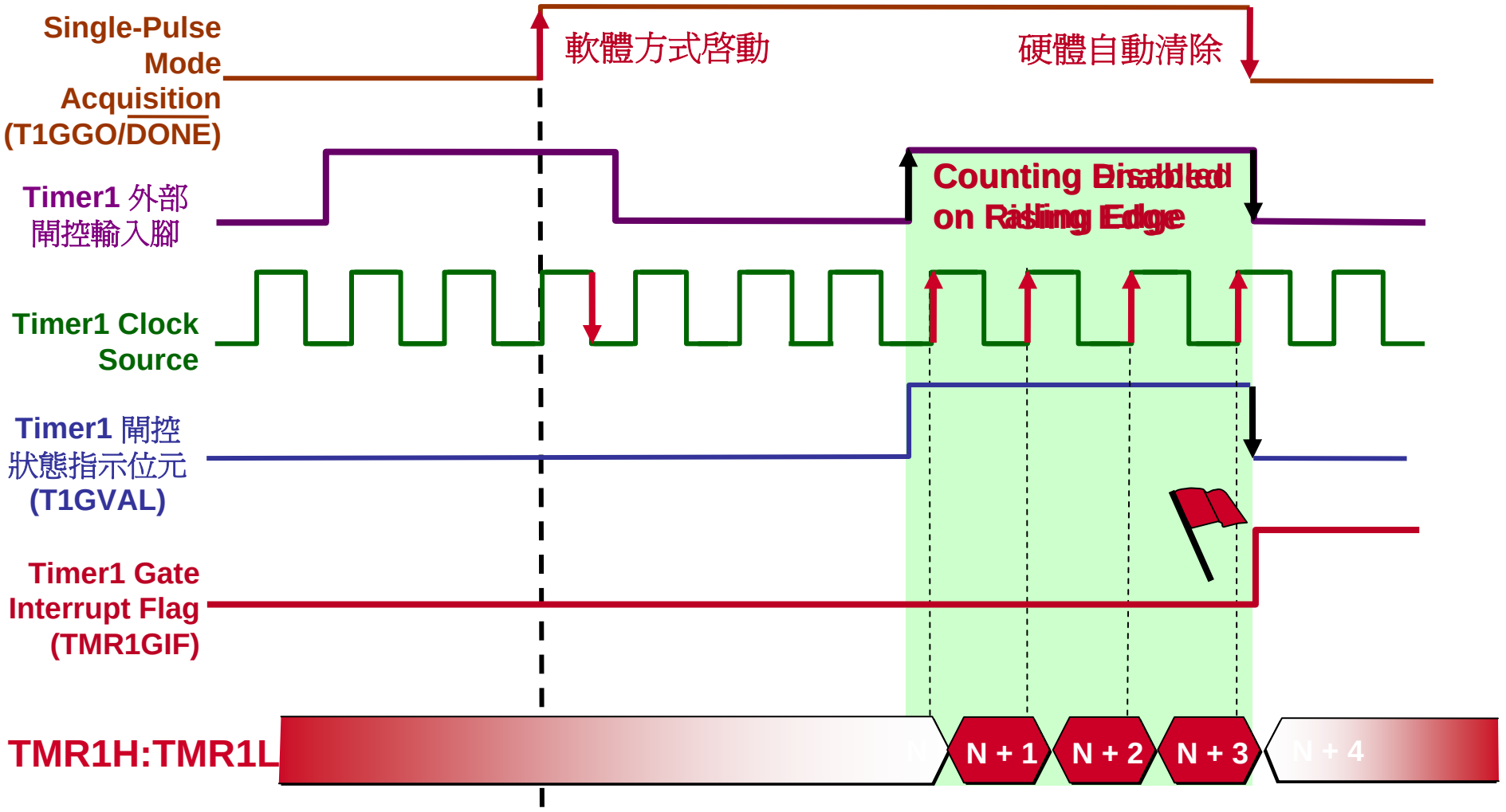
- 用來指示閘控目前是開啓的狀態，並在往上計數 **(TMR1H:TMR1L)**
- 每個外部閘控輸入的下緣將會產生 **Timer1 Gate Interrupt Flag (TMR1GIF)**



Timer1 閘控計時 (Toggle Mode)



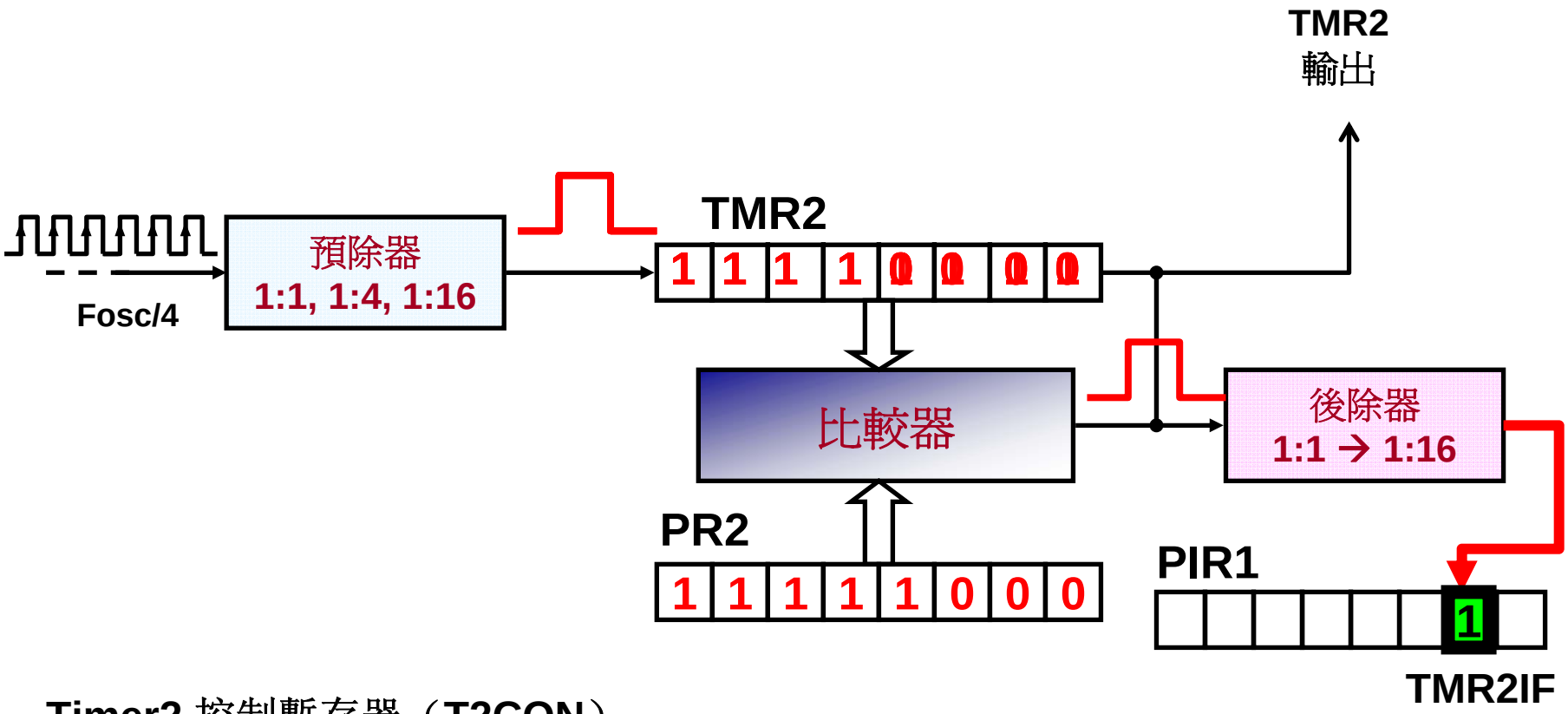
Timer1 閘控計時
(Single-Pulse Mode)



Timer2/4/6

- **8-bit** 計時器及周期產生器
- 有預除器 **(1:1, 1:4,1:16, 1:64)**
- 後除器之功能 **(1:1** 到 **1:16)**
- **PWM** 輸出模式下，基本的頻率來源
- **TMR2** 爲一可讀、寫具有自動載入功能的計時器
- **TMR2** 會自動加一並與設定的值相比；若相等則送出訊號至後除器或產生中斷，並自動將自己清除爲零，重新計時
- 可做爲**MSSP (SPI™)** 傳送速率的設定

Timer2 方塊圖



Timer2 控制暫存器（T2CON）

	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Capacitive Sensing Module

電容式觸控感應模組

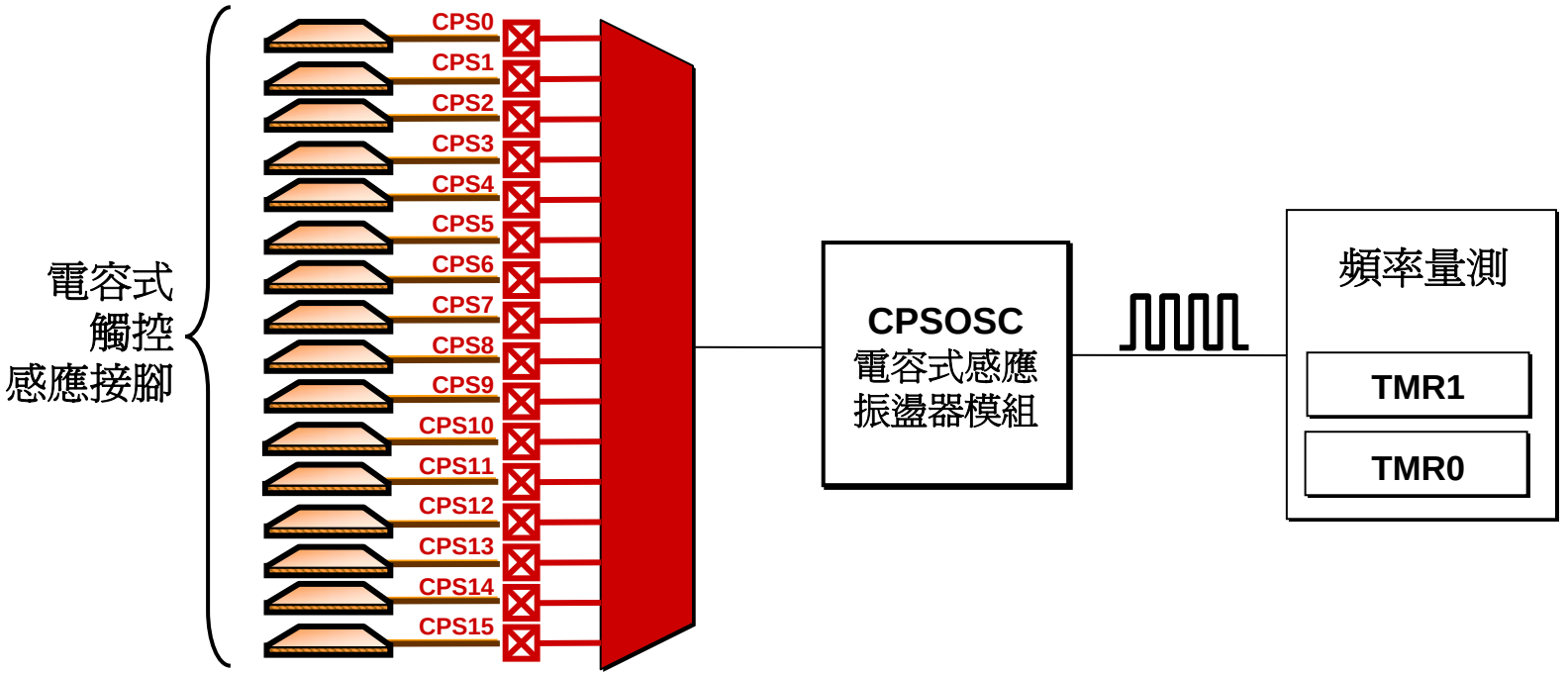
mTouch™

電容式觸控感應模組 (CSM)

- 硬體電路設計
  - 專屬的振盪器
  - 勿需外接零件，直接觸控感應介面 (接腳)
- 振盪器可以在睡眠模式下工作
  - **Timer1** 依然可以取的計數植

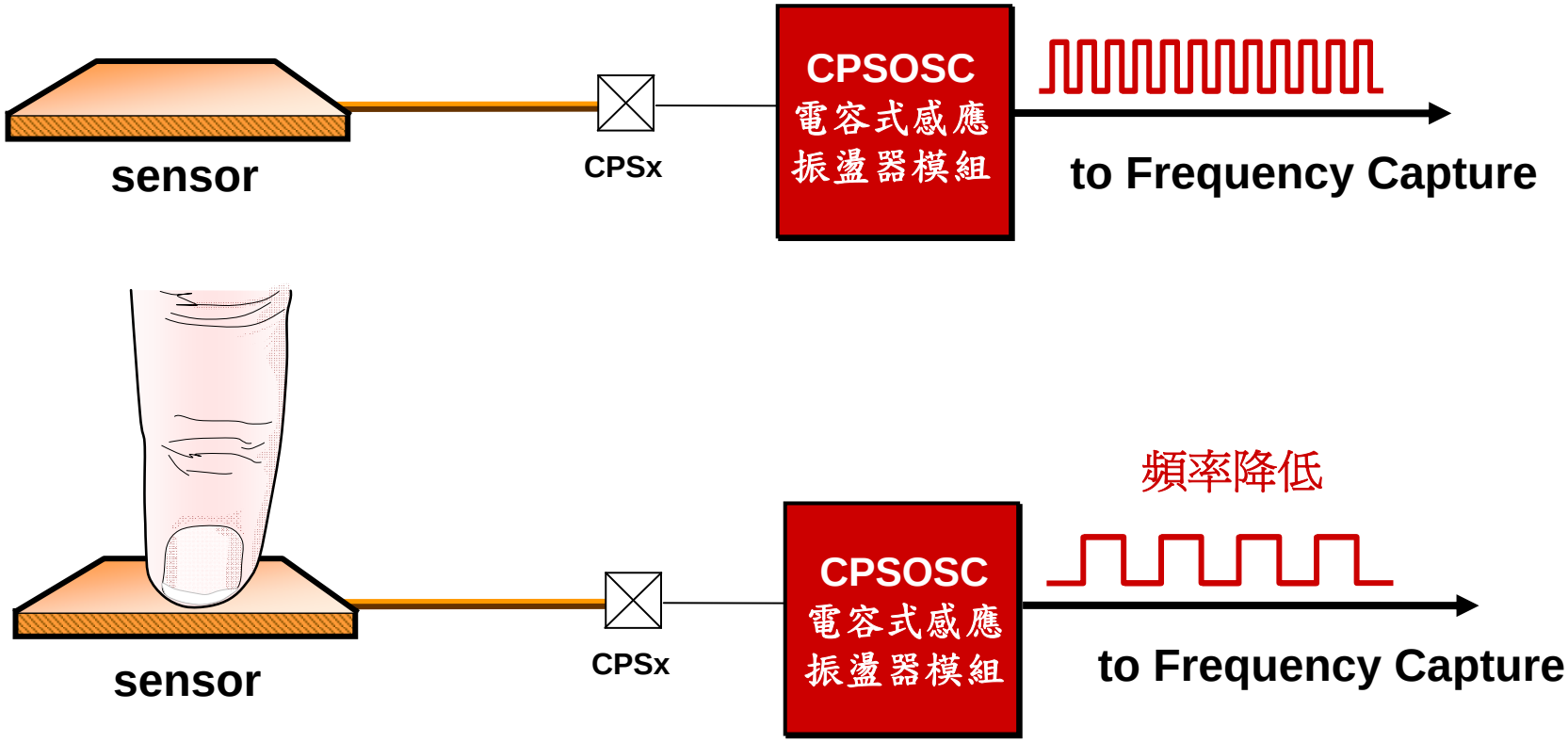
CSM 方塊圖

- 多達 16 個專屬觸控感應輸入腳
  - 軟體設定多工器擇一輸入腳
- 直接觸控感應介面
  - 勿需外接零件



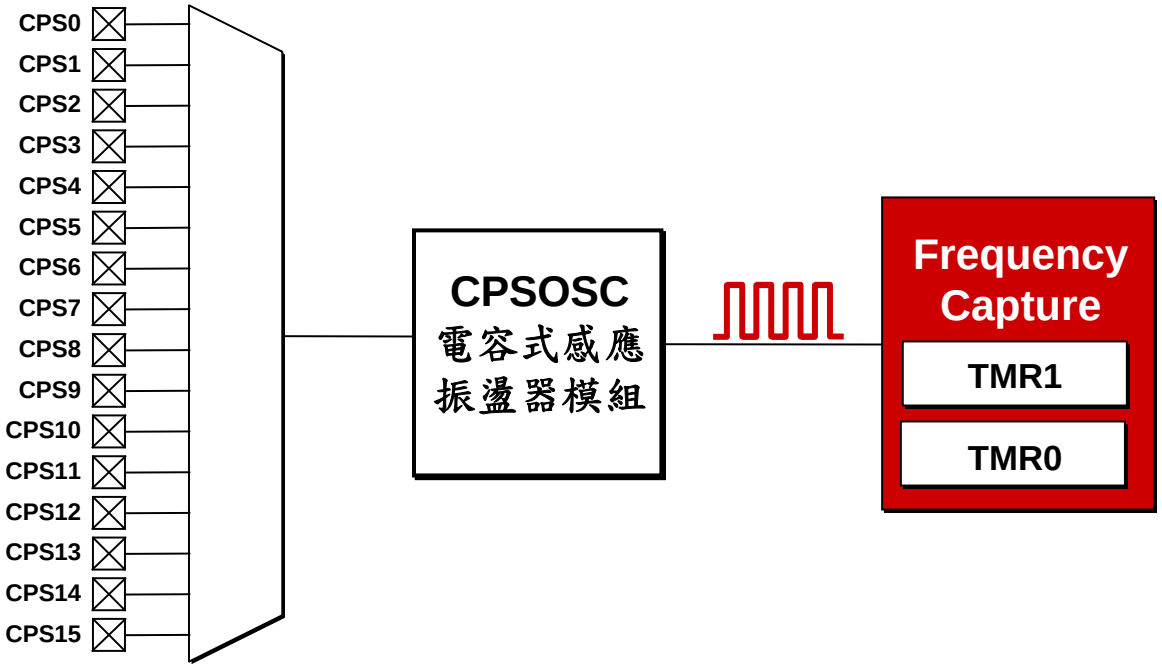
CSM
專屬振盪器

振盪器的頻率受外界的雜散電容引響
- 電容變大，頻率降低
- 電容減少，頻率升高

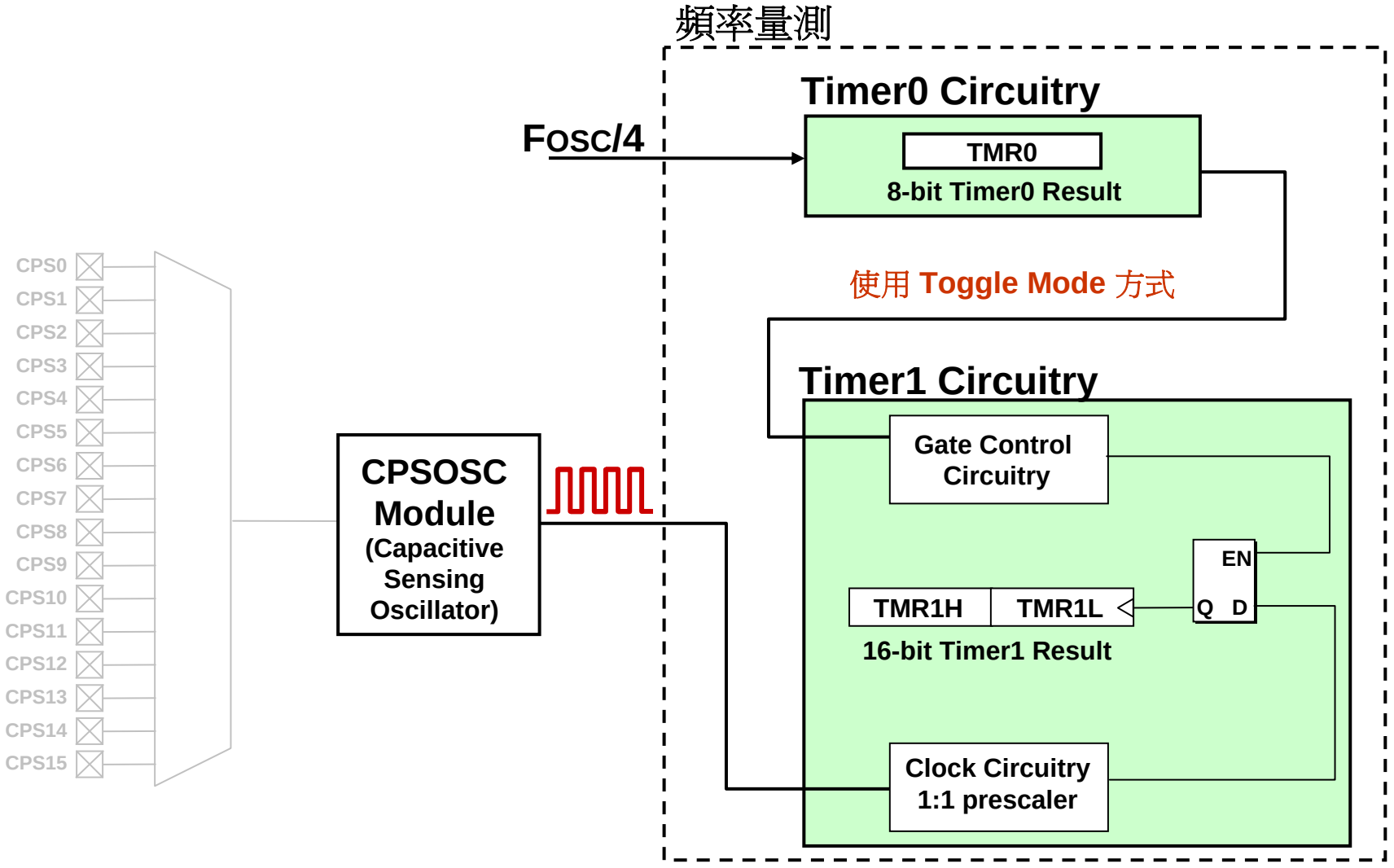


CSM
頻率的測量

- 使用 **Timer** 的計數器來擷取頻率
  - Timer0 8-bit Timer
  - Timer1 16-bit Timer (使用 Timer1)

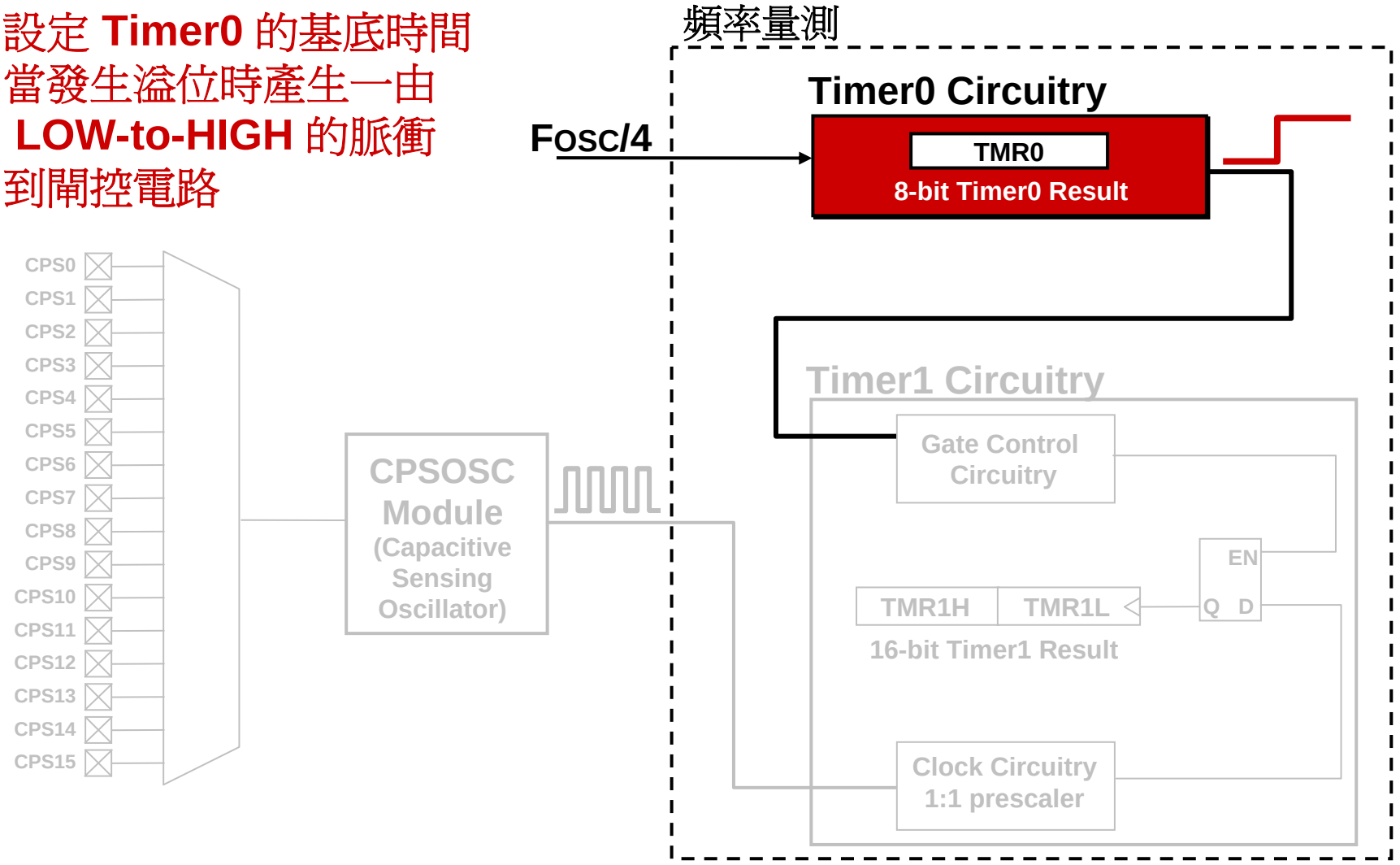


使用 Timers 作爲頻率量測方塊圖



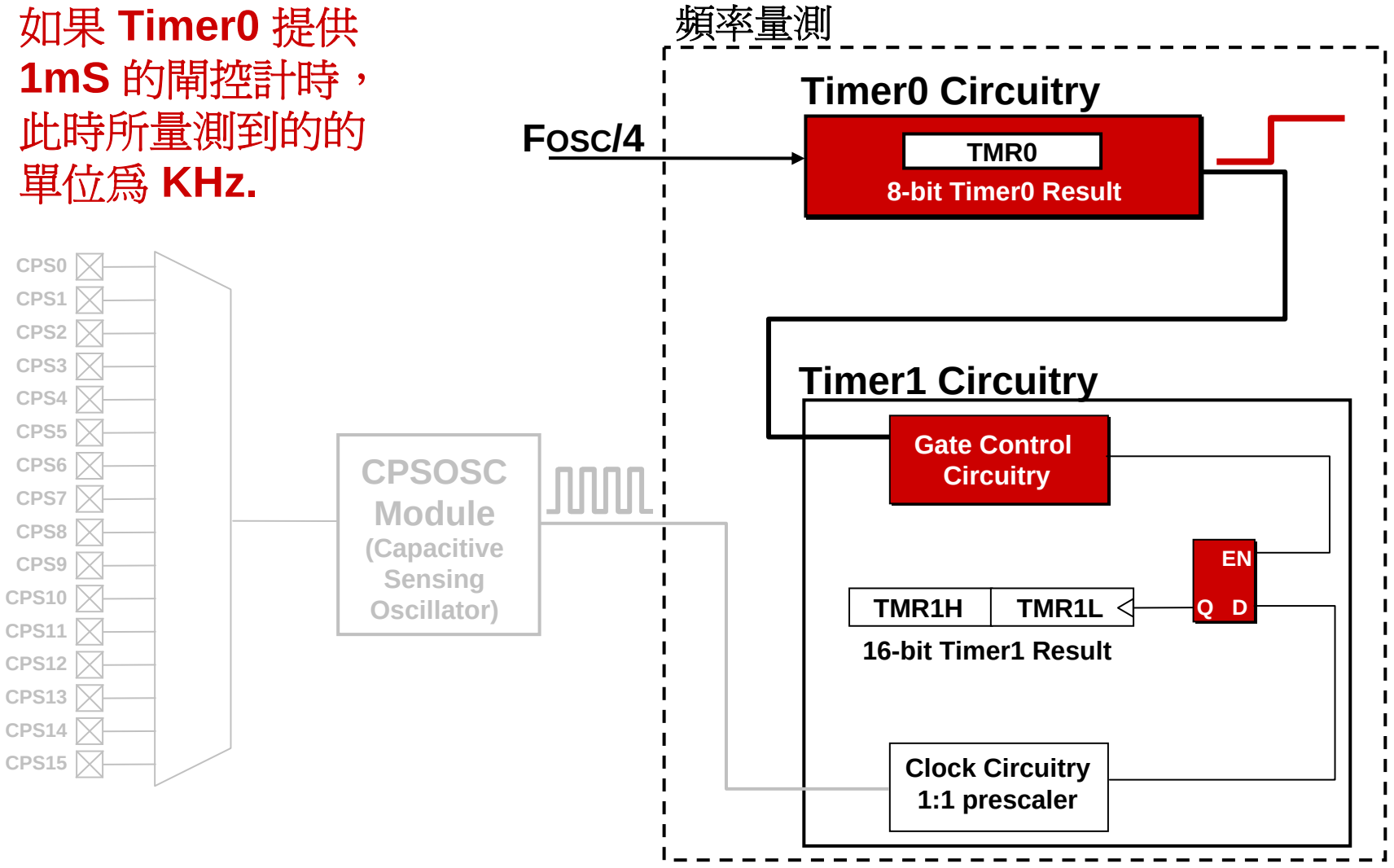
使用 Timers 作爲頻率量測方法

設定 **Timer0** 的基底時間
當發生溢位時產生一由
LOW-to-HIGH 的脈衝
到閘控電路



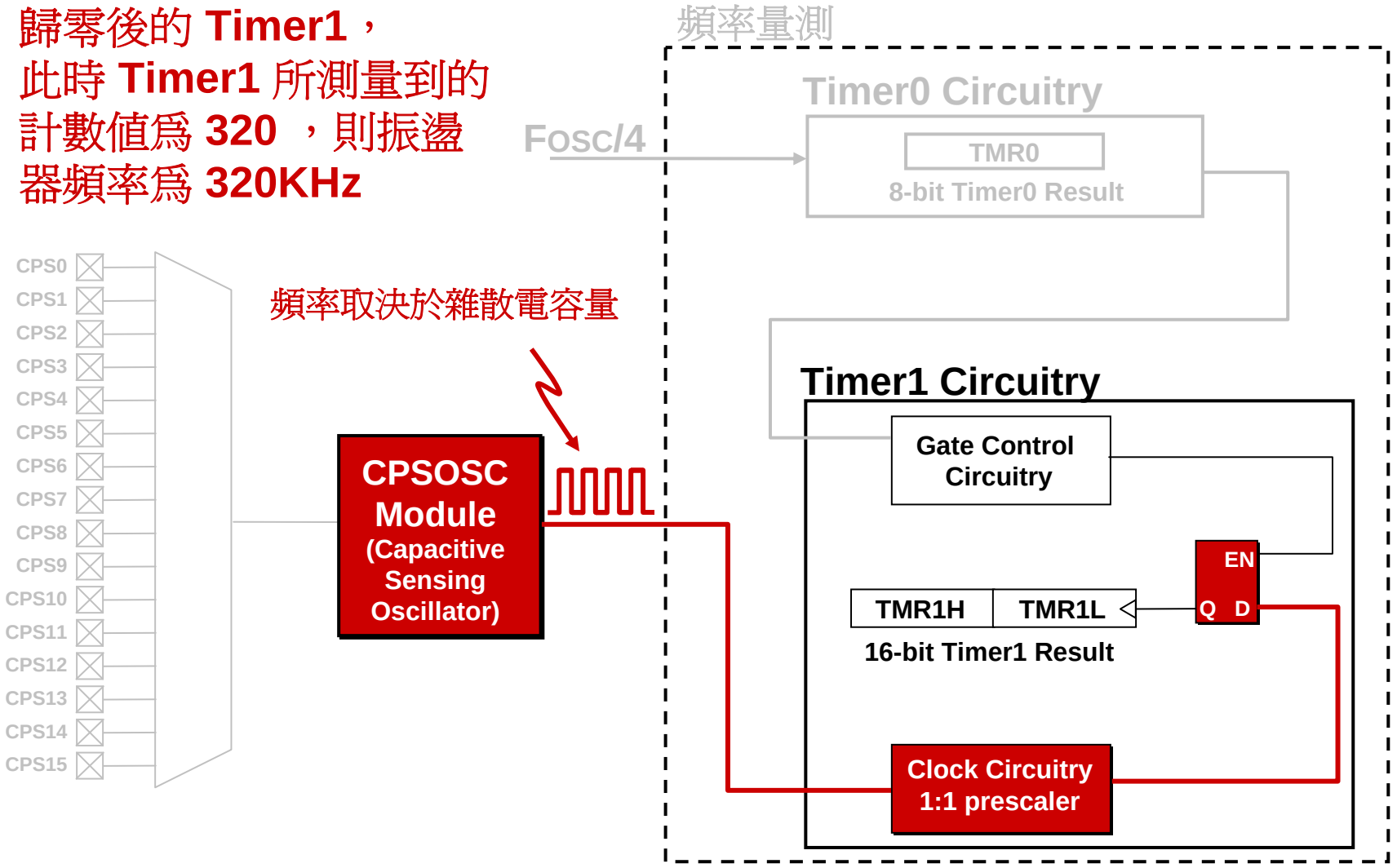
使用 Timers 作爲頻率量測方法

如果 **Timer0** 提供 **1mS** 的閘控計時，此時所量測到的的單位爲 **KHz.**



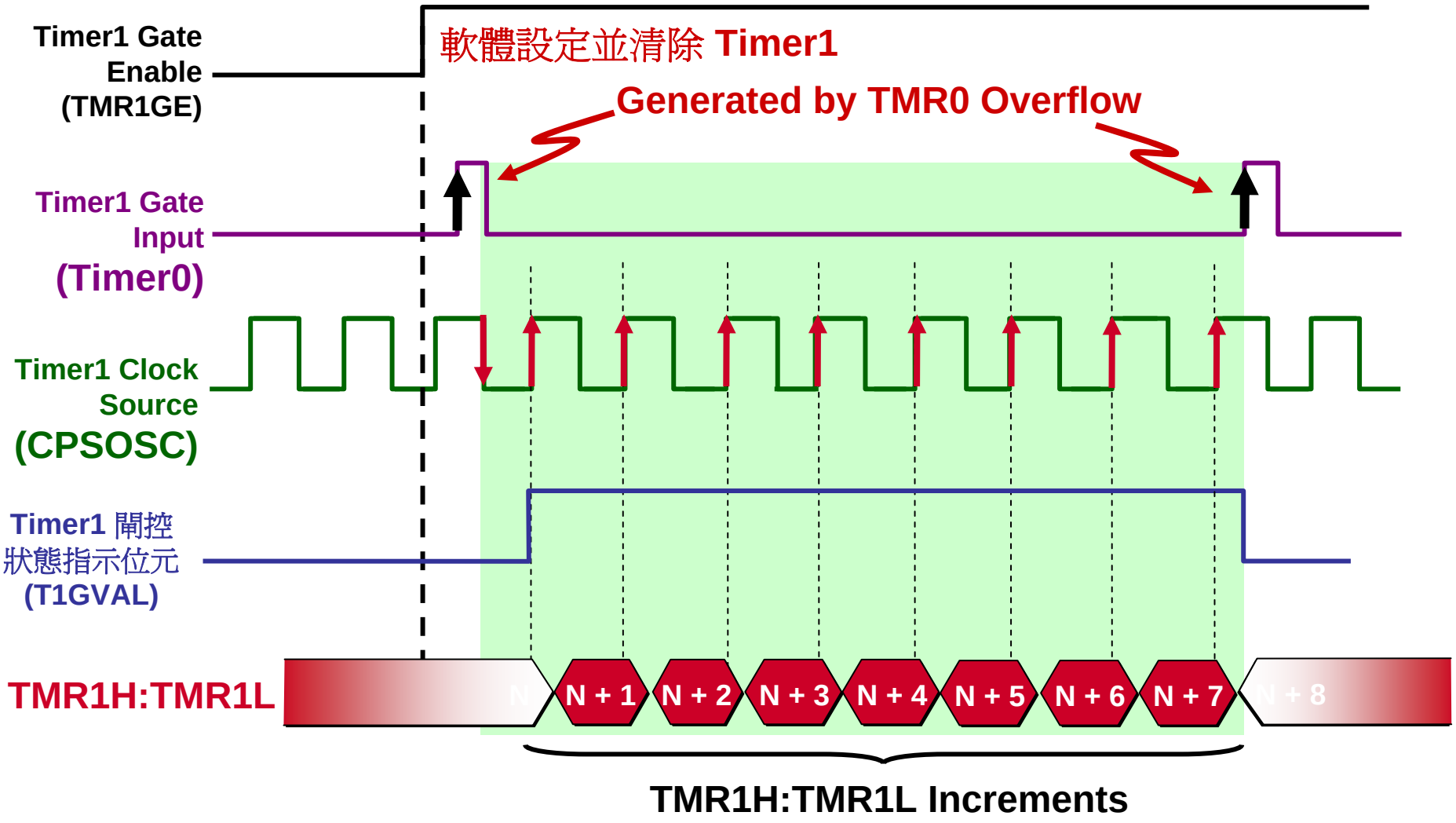
使用 Timers 作爲頻率量測方法

歸零後的 **Timer1**，
此時 **Timer1** 所測量到的
計數值爲 **320** ，則振盪
器頻率爲 **320KHz**



CSM 使用 Timer1 的閘控電路

Toggle Mode



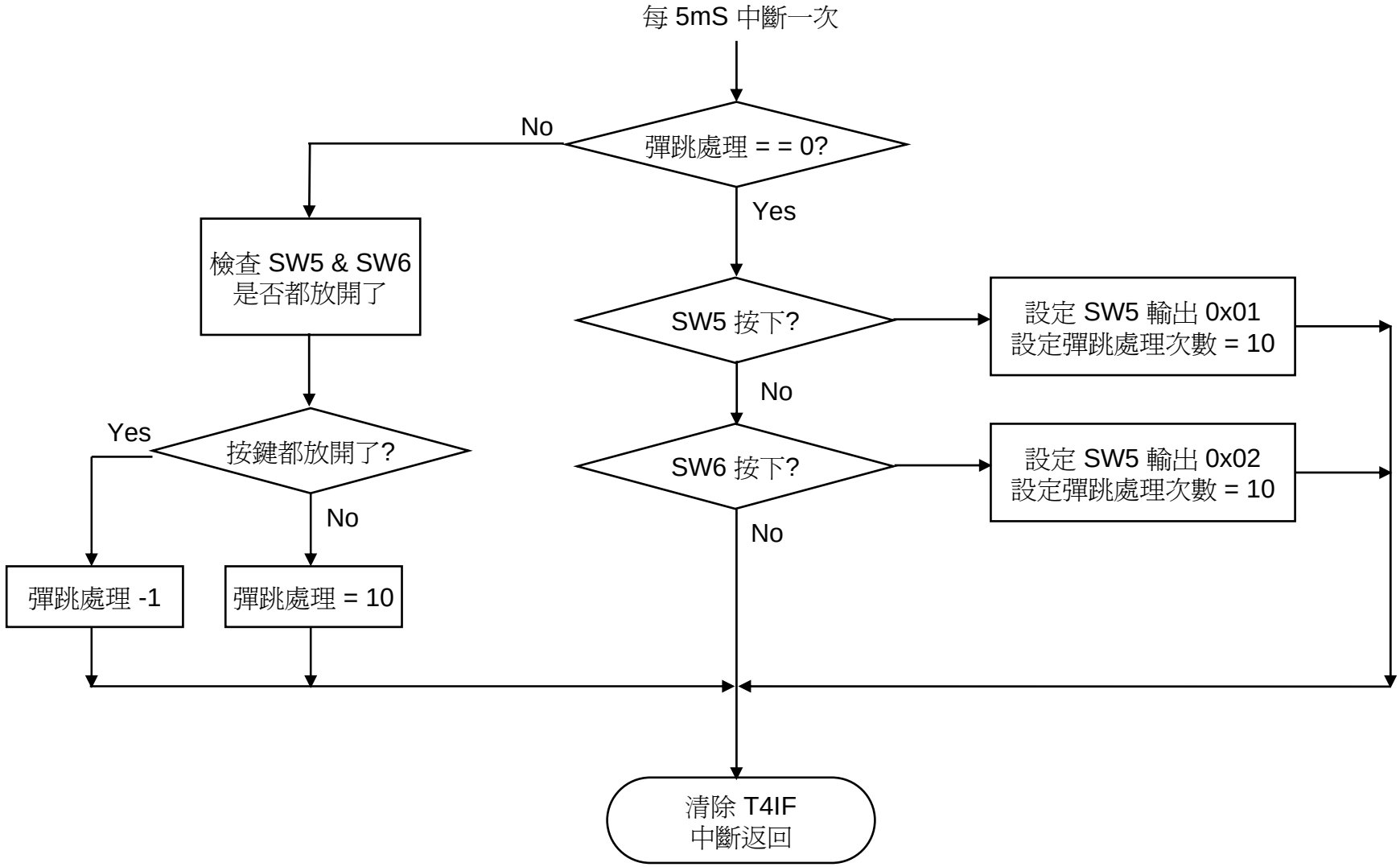
CSM 使用 Timer1 的閘控電路
Toggle Mode

- **T1GVAL 下降緣時觸發中斷**
  - 表示 1mS 計時到了
  1. 讀取 TMR1H:TMR1L
  2. 取平均值
  3. 清除 TMR1H:TMR1L 等下一次的攫取

Lab5 電容式觸控按鍵

- **Lab5 使用的周邊模組**
  - LCD 顯示 CSM 輸出的頻率
  - Timer2 提供 1mS 的中斷計時，做爲 CSM 的計時基準時間
  - Timer4 提供 5mS 的中斷計時，做爲一般按鍵 (S5 & S6) 的彈跳處理
  - Timer1 做爲 CSM 頻率計數器
  - 一個 CSM 觸控輸入 (Logo 圖示)

使用 Timer4 處理按鍵彈跳



Lab5 取平均值

- **Lab5** 是採用移動平均值的方式
  - 適用於較少 Channel 觸控輸入時使用
  - 最耗記憶體，Lab5 使用 unsigned int CSM_BUF[16] 的陣列方式用掉 32 Bytes
  - 優點是：速度快，可以去除最大值及最小值的脈衝誤差
- 如果觸控 **Channel** 多的話，可以用此方式
  - Temp = (Temp – Temp/16)

    Output = (Temp + New/16)
  - 檢查 New 的 <b3:b0> 的值是否 > 0x8，加一補此誤差(類似四捨五入的修正方式)

Lab5
CSM Touch & Timer

- 專案位置：
  - .. \Advance PICC Application Labs\Lab5 CSM Touch\CSM Touch & Timer.mcp
- 專案的裡程式
  - Main.c
  - APP-EDF09 LCD.c
  - CSM_Touch.c
  - LCD_Defs.h
  - Main.h

類比周邊介紹

固定參考電壓源

簡易型 **DAC**

電壓比較器

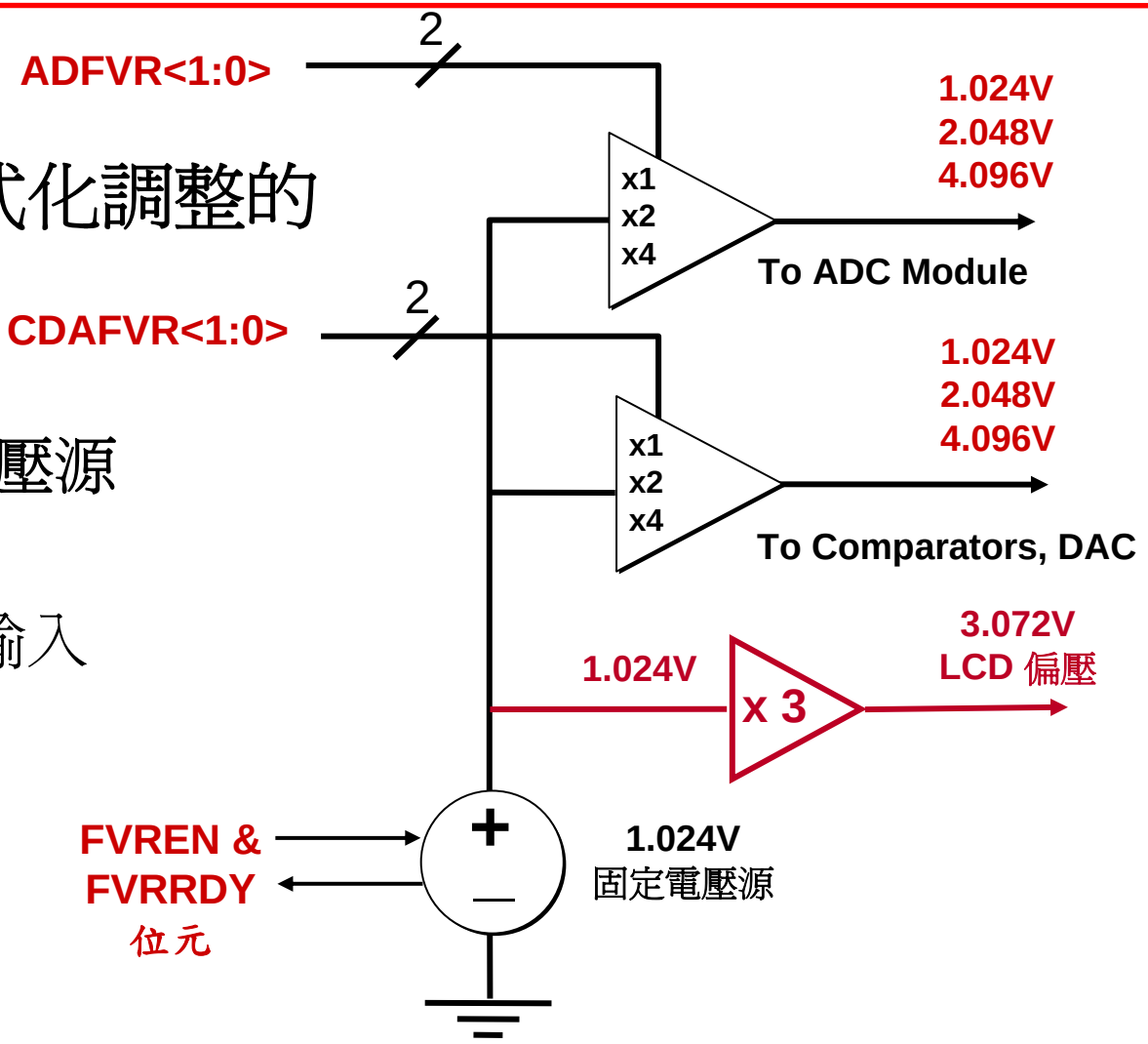
10-bit ADC

固定參考電壓源

- PIC16F1937 固定參考電壓源
  - **ADC** 輸入腳
  - **ADC** 正端參考電壓源
  - 比較器正端輸入
  - 可程式調整參考電壓
  - **LCD** 偏壓產生器

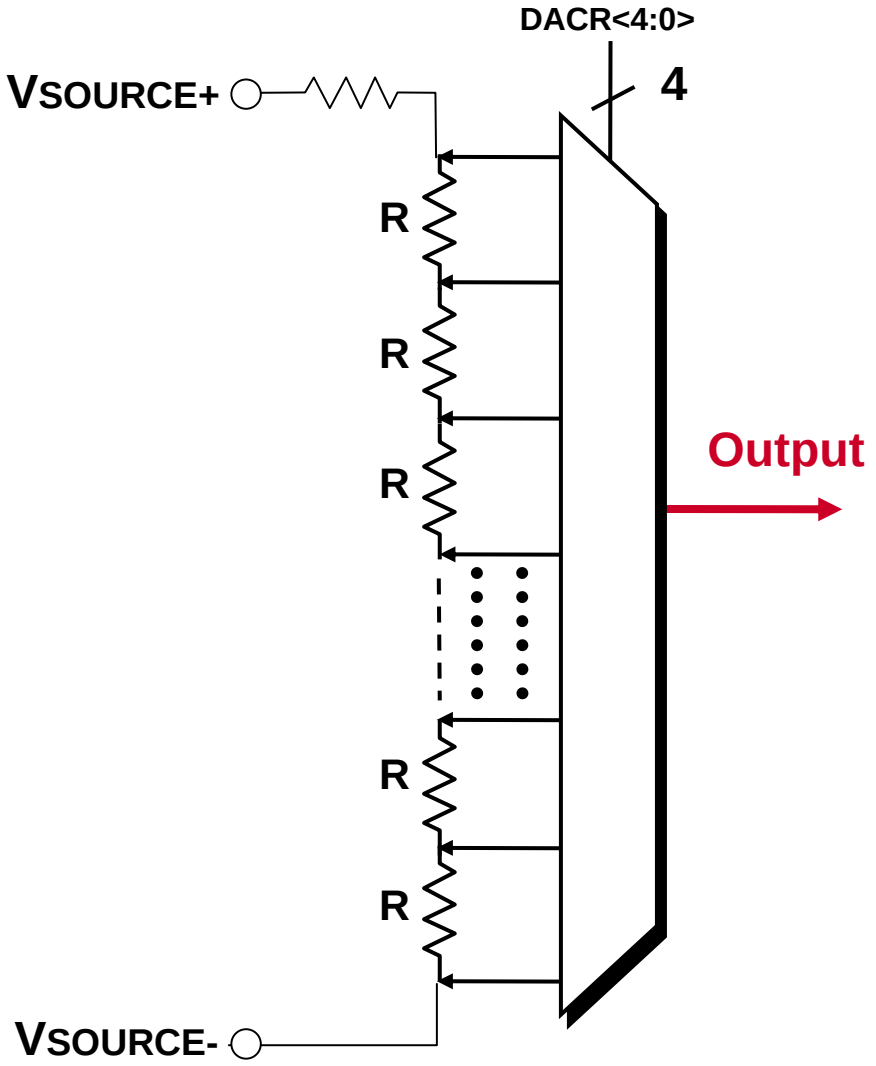
固定參考電壓源

- 兩組獨立的可程式化調整的放大器
  - ADC 電壓輸入
  - **ADC 正端參考電壓源**
  - DAC 轉換器
  - 比較器正端電壓輸入
  - **LCD 偏壓產生器**



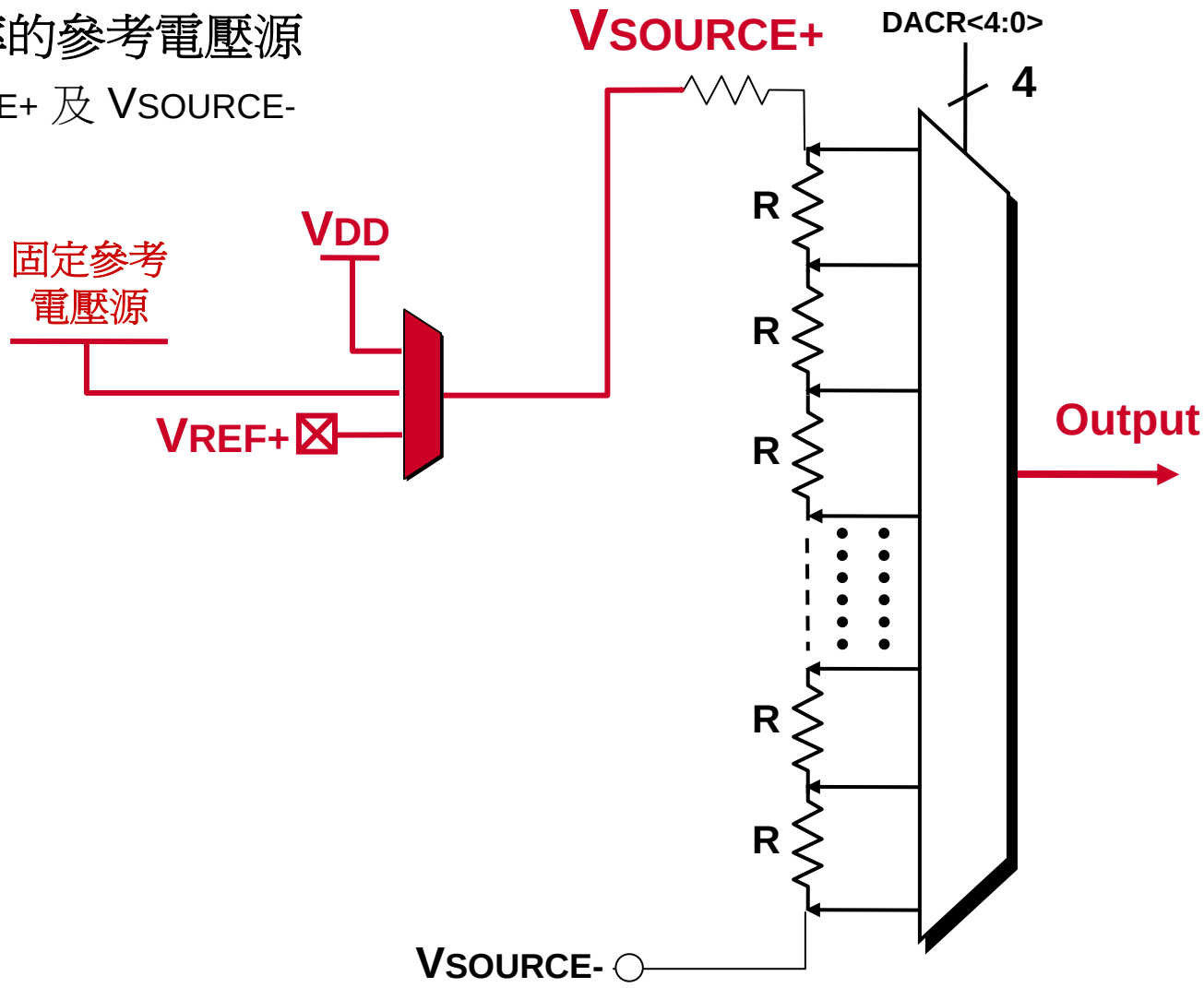
32 階 DAC 轉換器 (DAC) Module

- 提供一可變比率的參考電壓源
  - 使用 VSOURCE+ 及 VSOURCE-



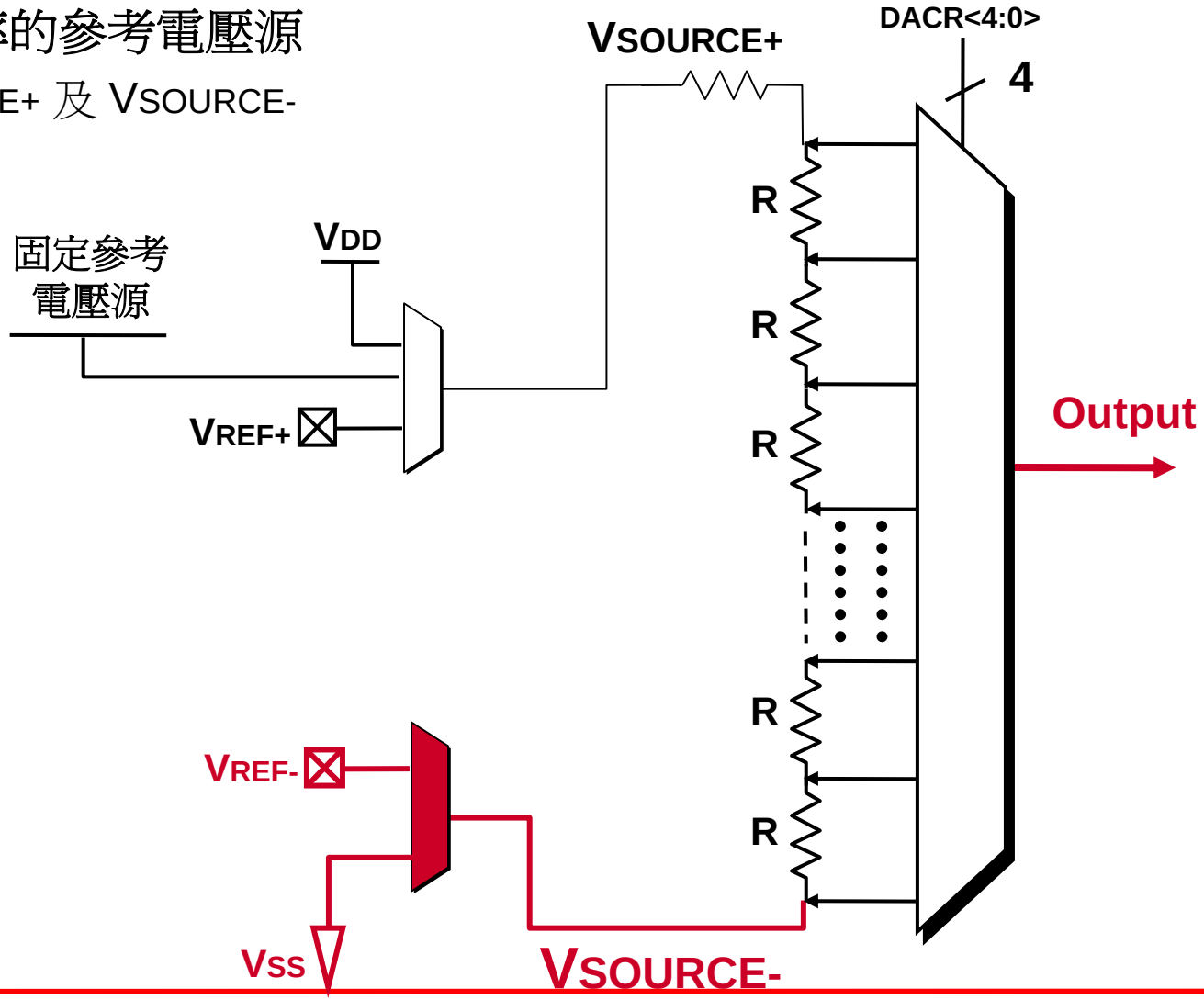
Digital-to-Analog Converter (DAC) Module

- 提供一可變比率的參考電壓源
  - 使用 VSOURCE+ 及 VSOURCE-



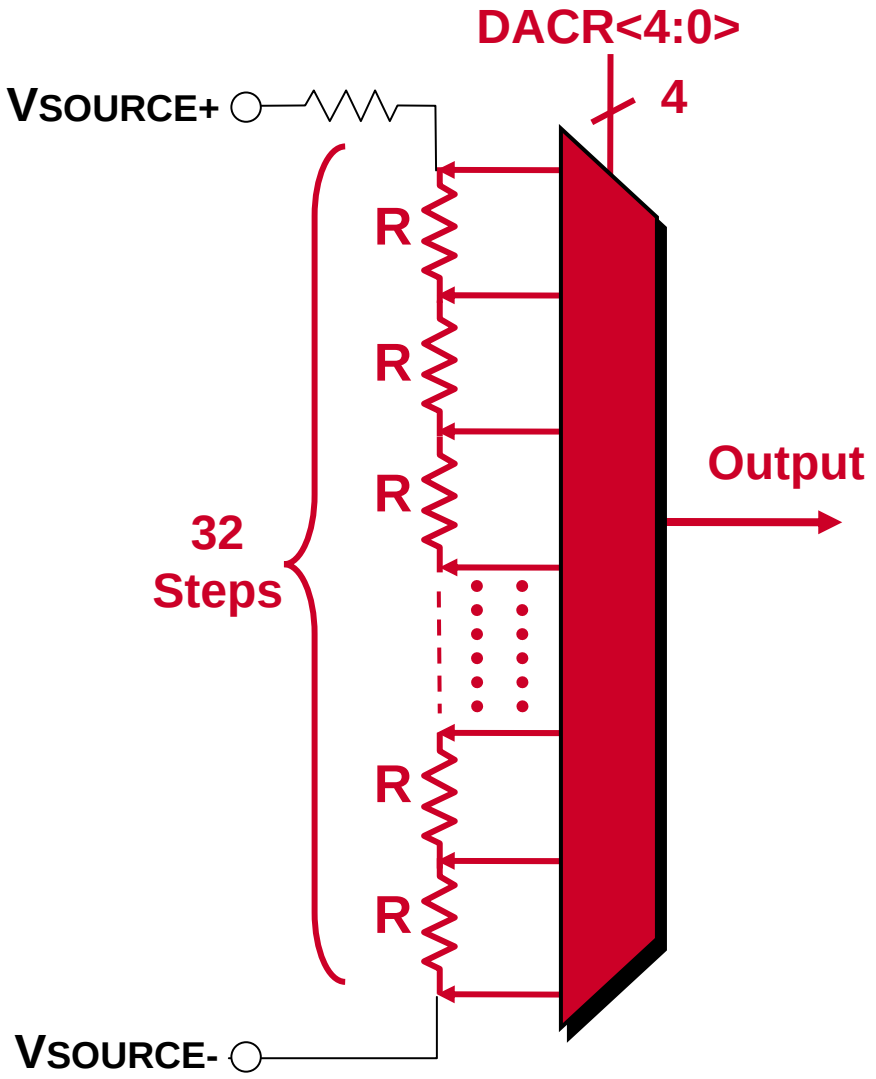
Digital-to-Analog Converter (DAC) Module

- 提供一可變比率的參考電壓源
  - 使用 VSOURCE+ 及 VSOURCE-



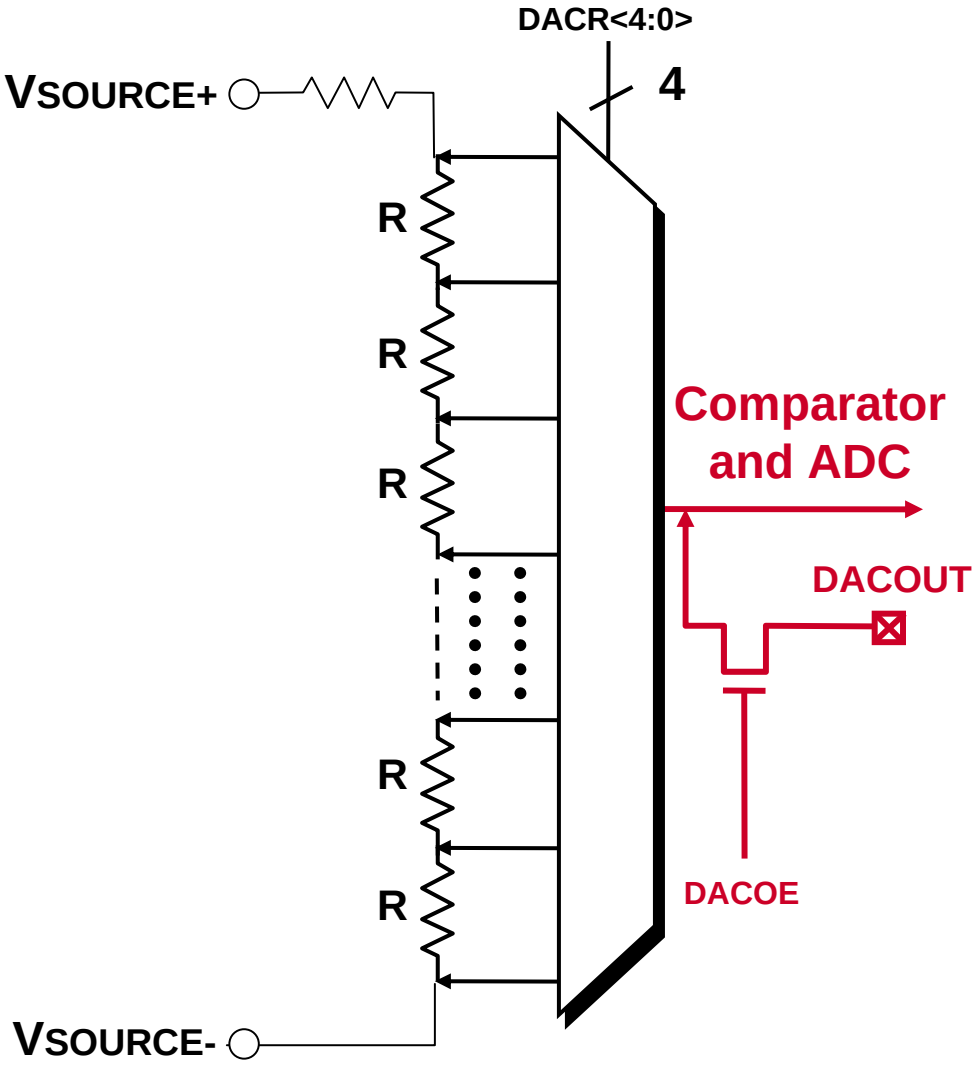
Digital-to-Analog Converter (DAC) Module

- 提供一可變比率的參考電壓源
  - 使用 VSOURCE+ 及 VSOURCE-
- **32** 階可選擇的輸出準位



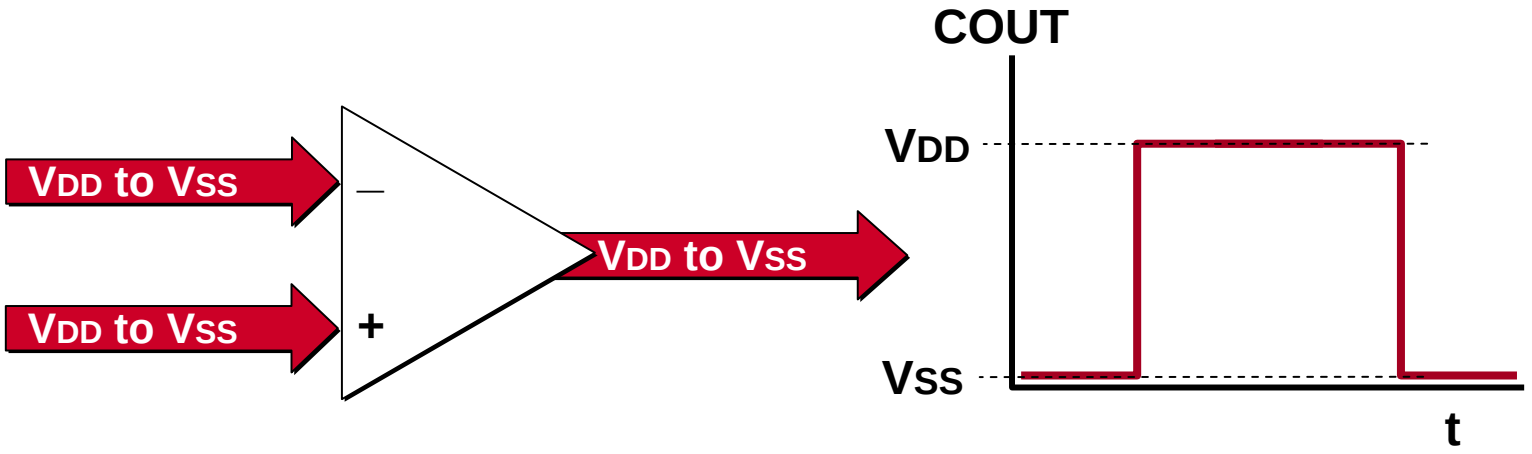
Digital-to-Analog Converter (DAC) Module

- 提供一可變比率的參考電壓源
  - 使用 VSOURCE+ 及 VSOURCE-
- **32** 階可選擇的輸出準位
- 輸出電壓可使用在：
  - 內部周邊：
    - **Comparator Non-Inverting Input**
    - **ADC Input**
  - 外部接腳輸出：
    - **DACOUT pin**



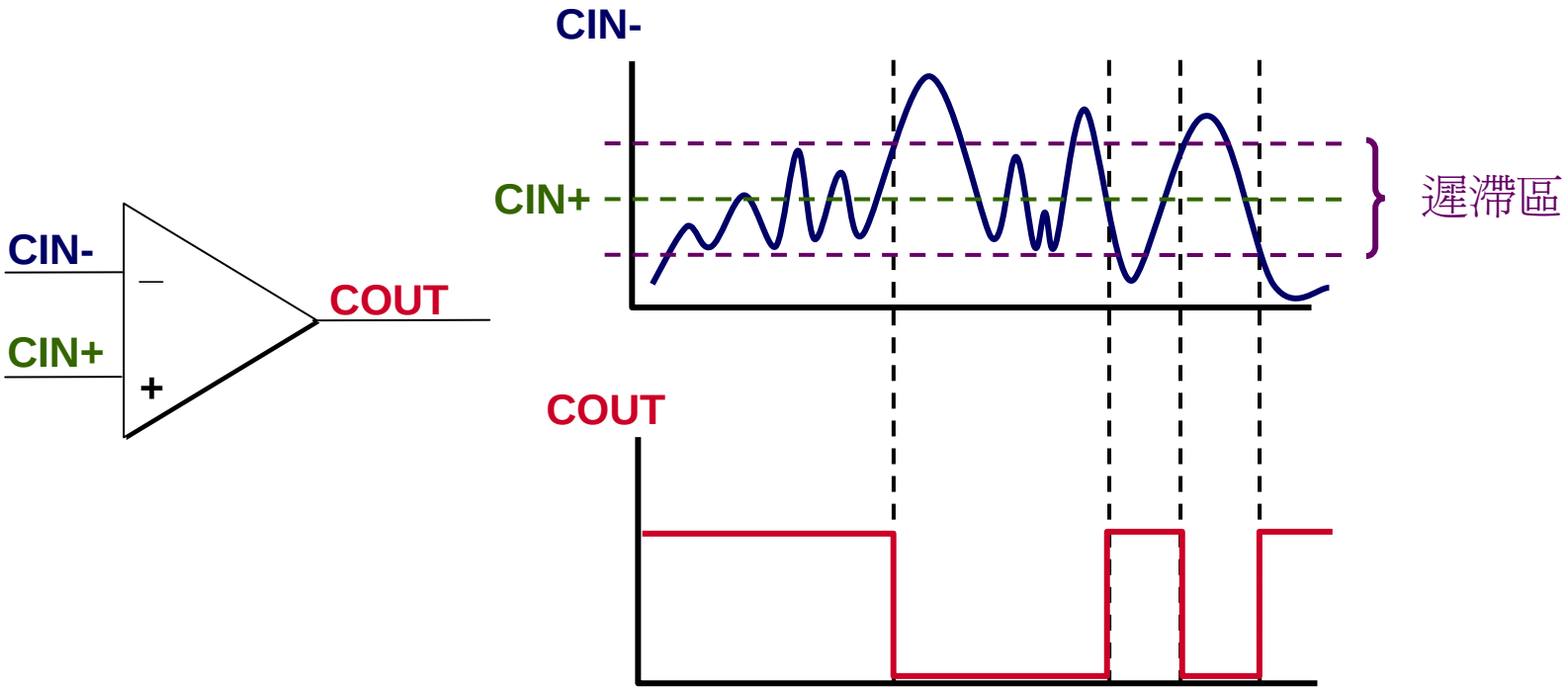
軌對軌的操作範圍

- **Now standard**
  - 可工作在低電壓下，達到最大的動態輸入及輸出位準



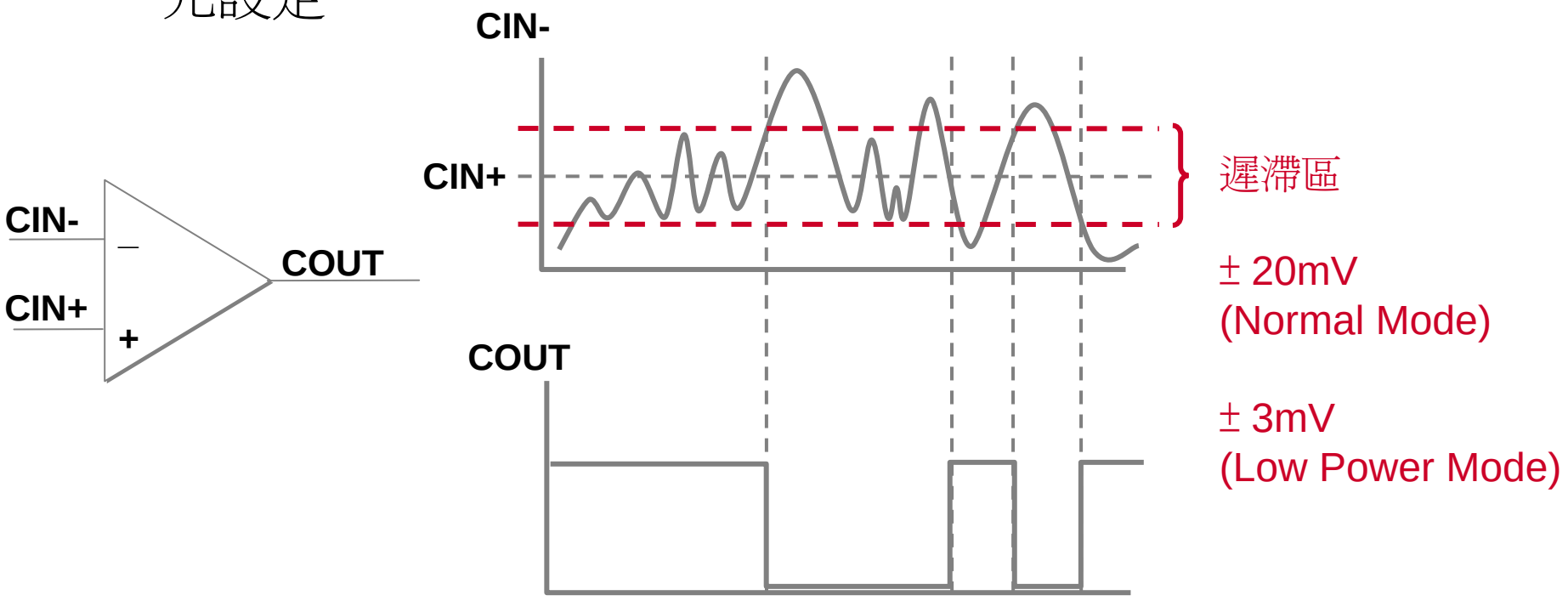
遲滯曲線 (Hysteresis)

- **Now Standard**
  - 可以用軟體方式設定遲滯電壓範圍 (1mV, 3mV, 20mV)



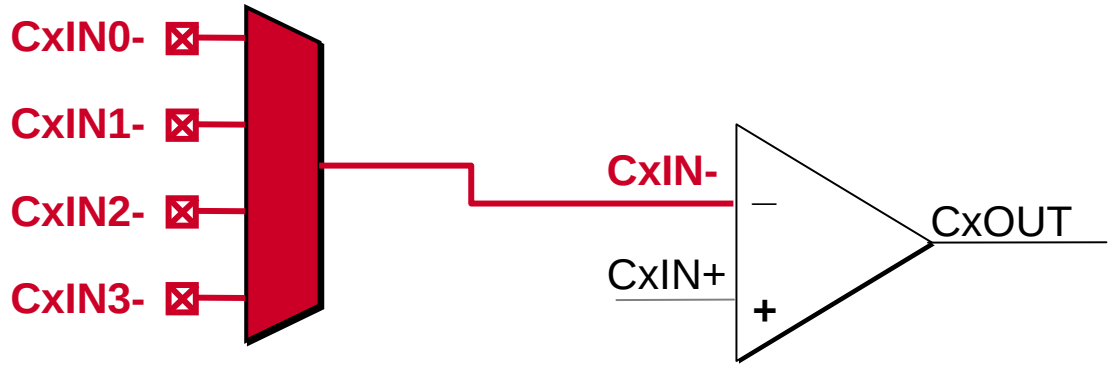
遲滯曲線 (Hysteresis)

- **Now Standard**
  - 可以用軟體方式設定遲滯電壓範圍 (1mV, 3mV, 20mV)
  - 遲滯電壓可以透過 CMxCON0<CxPS & CxHYS> 兩位元設定



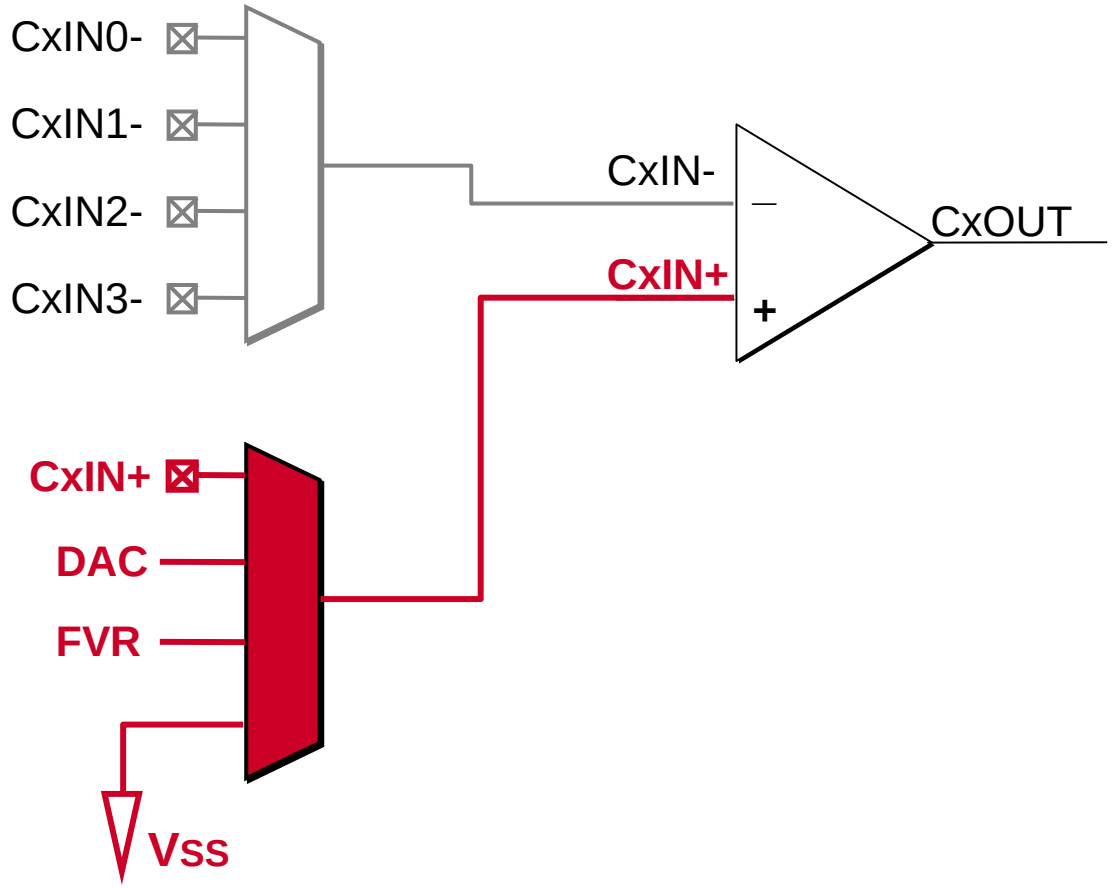
比較器輸入選擇

- 四種負端輸入來源選擇

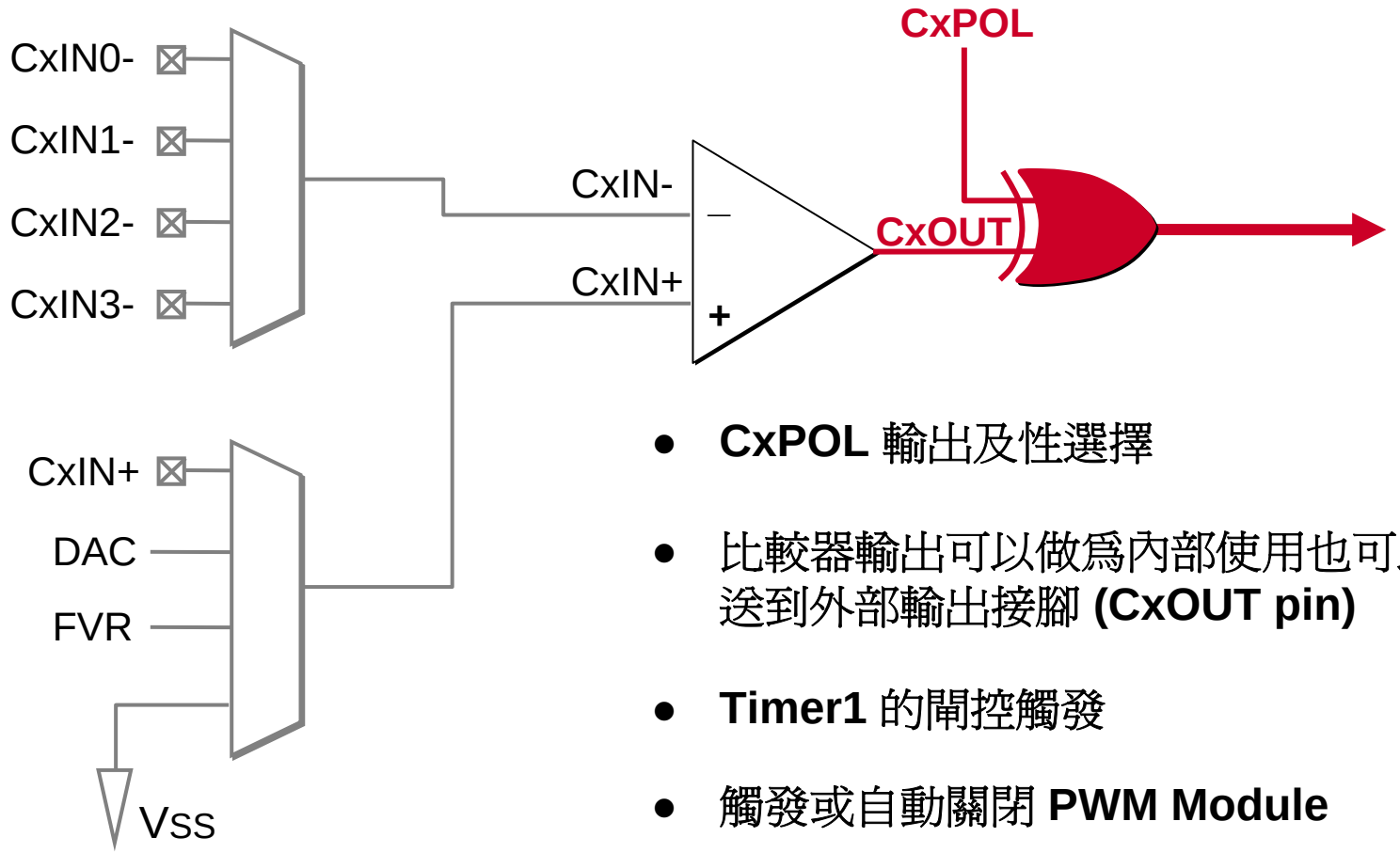


比較器輸入選擇

- 軟體方式選擇正端輸入



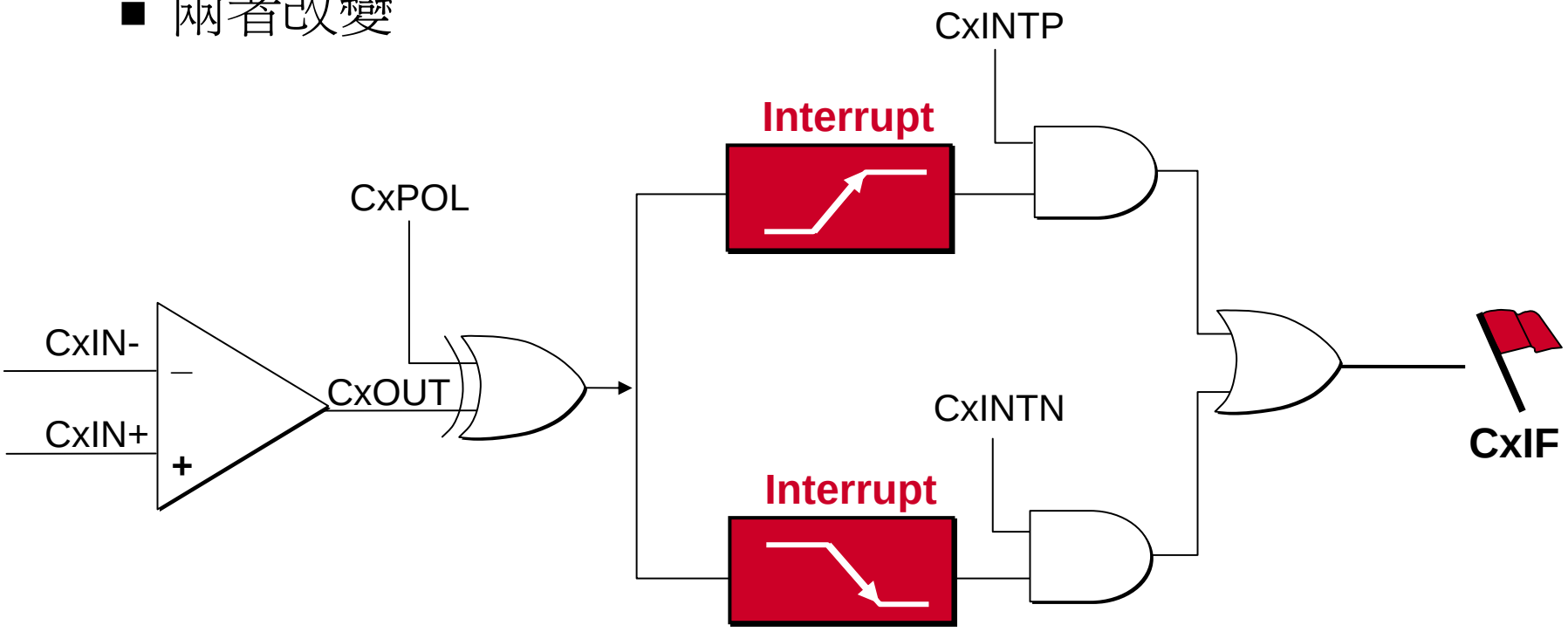
比較器輸出選擇



- **CxPOL** 輸出及性選擇
- 比較器輸出可以做爲內部使用也可以送到外部輸出接腳 **(CxOUT pin)**
- **Timer1** 的閘控觸發
- 觸發或自動關閉 **PWM Module**

比較器中斷

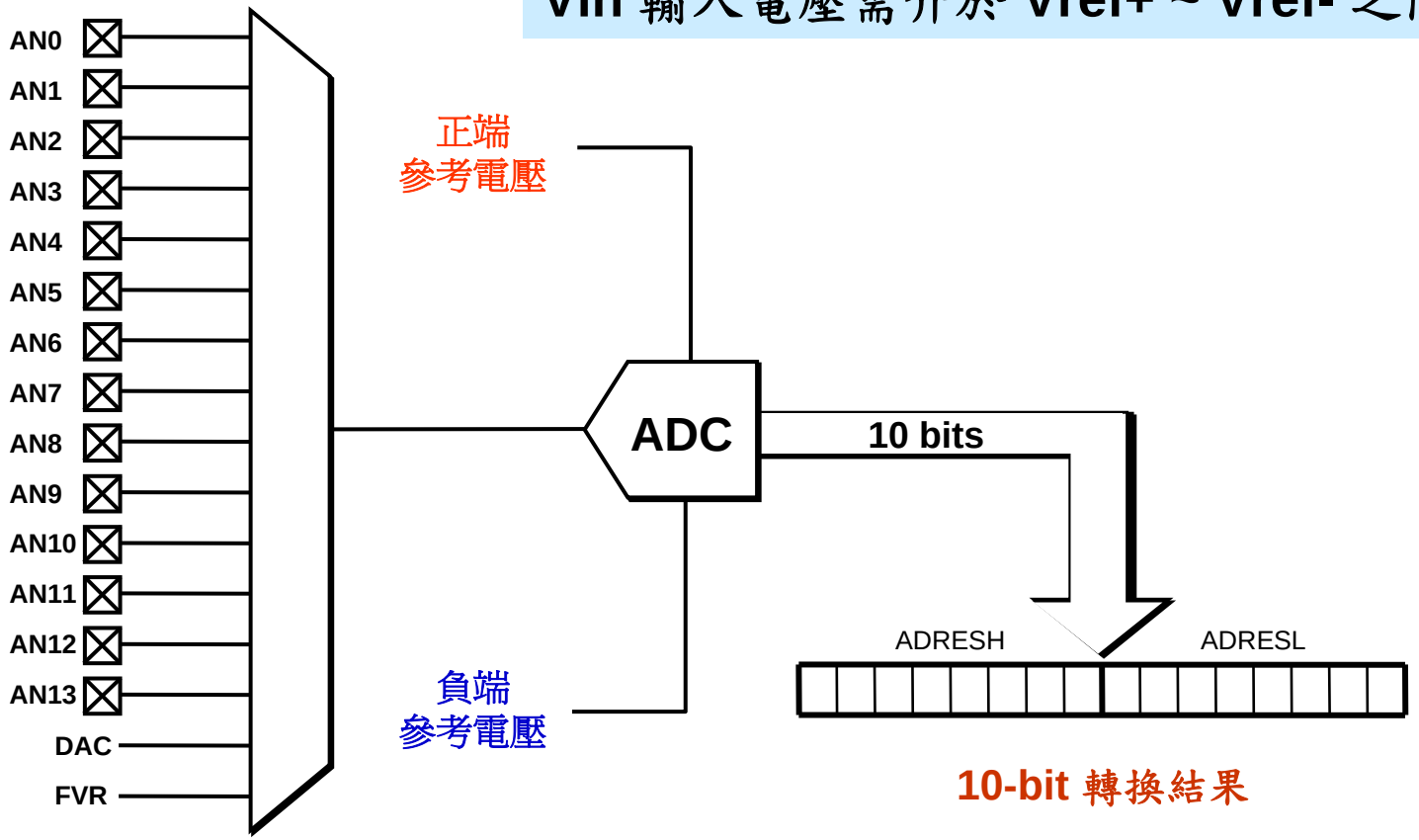
- **Interrupt-on-Change** 位準改變中斷觸發
  - 上升緣
  - 下降緣
  - 兩者改變



ADC 概述

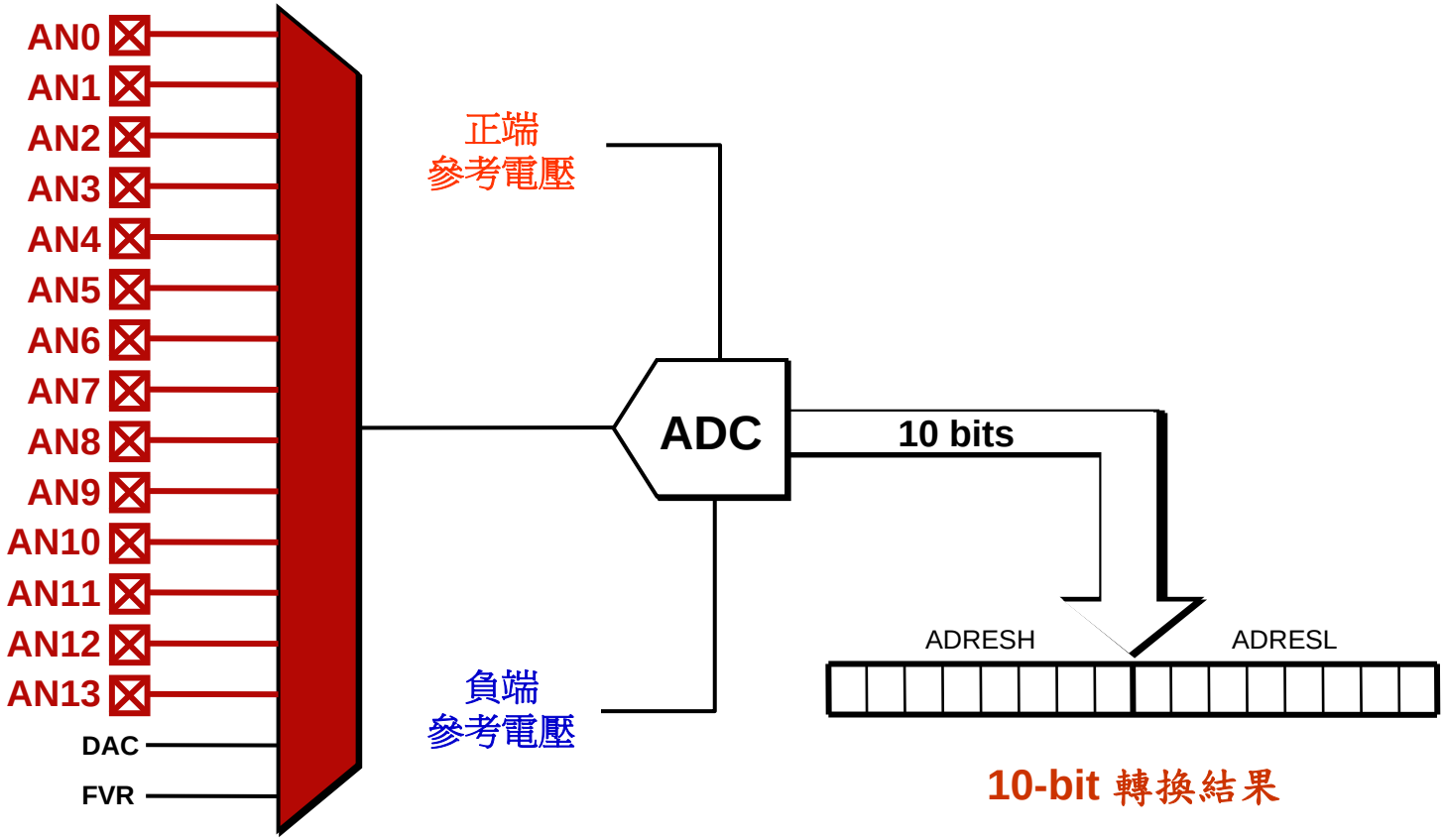
- 轉換值 = [(Vin – Vref-) / (Vref+ – Vref-)] x 1023

Vin 輸入電壓需介於 Vref+ ~ Vref- 之間。



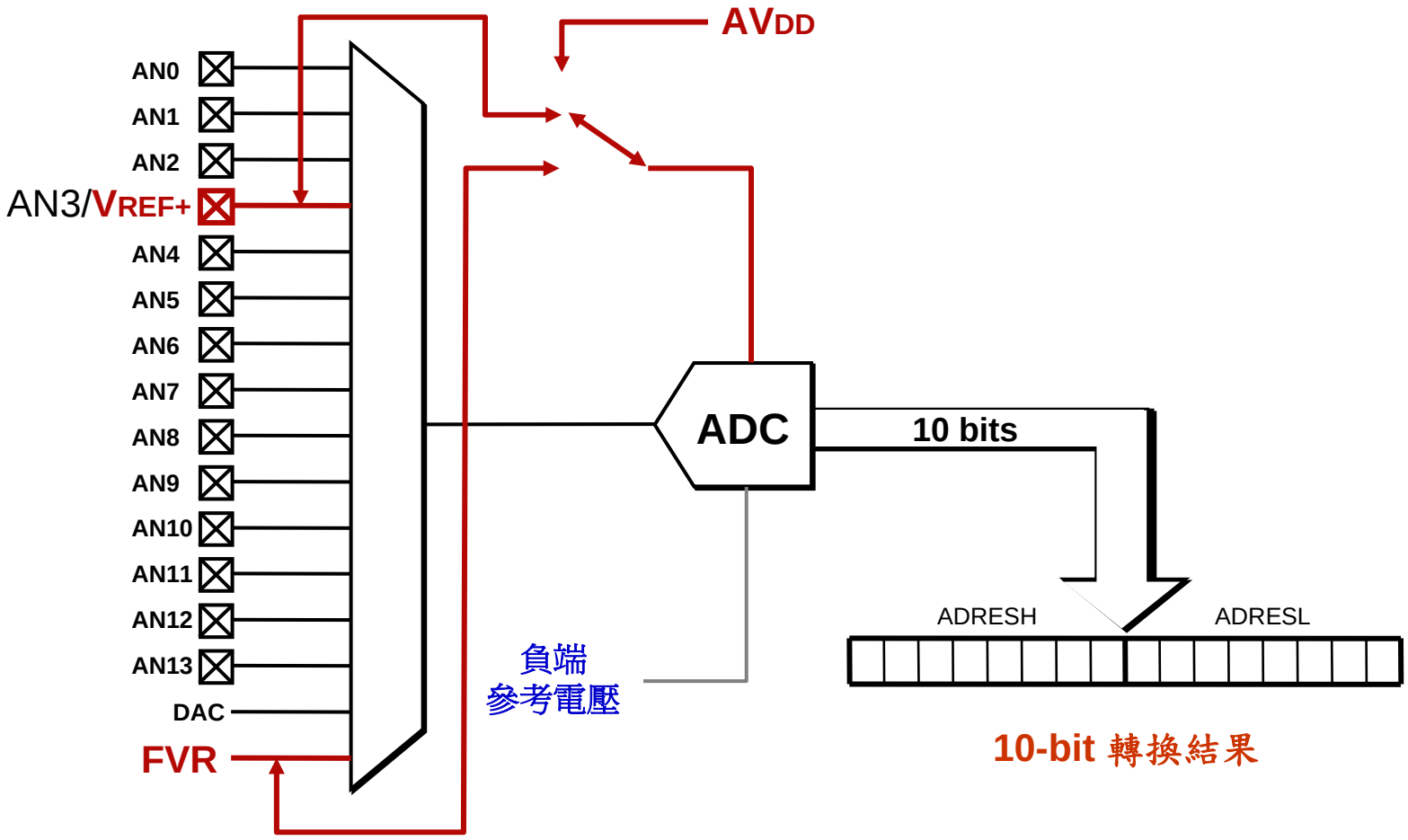
ADC 概述

- 可達 **14** 個類比轉換輸入通道選擇



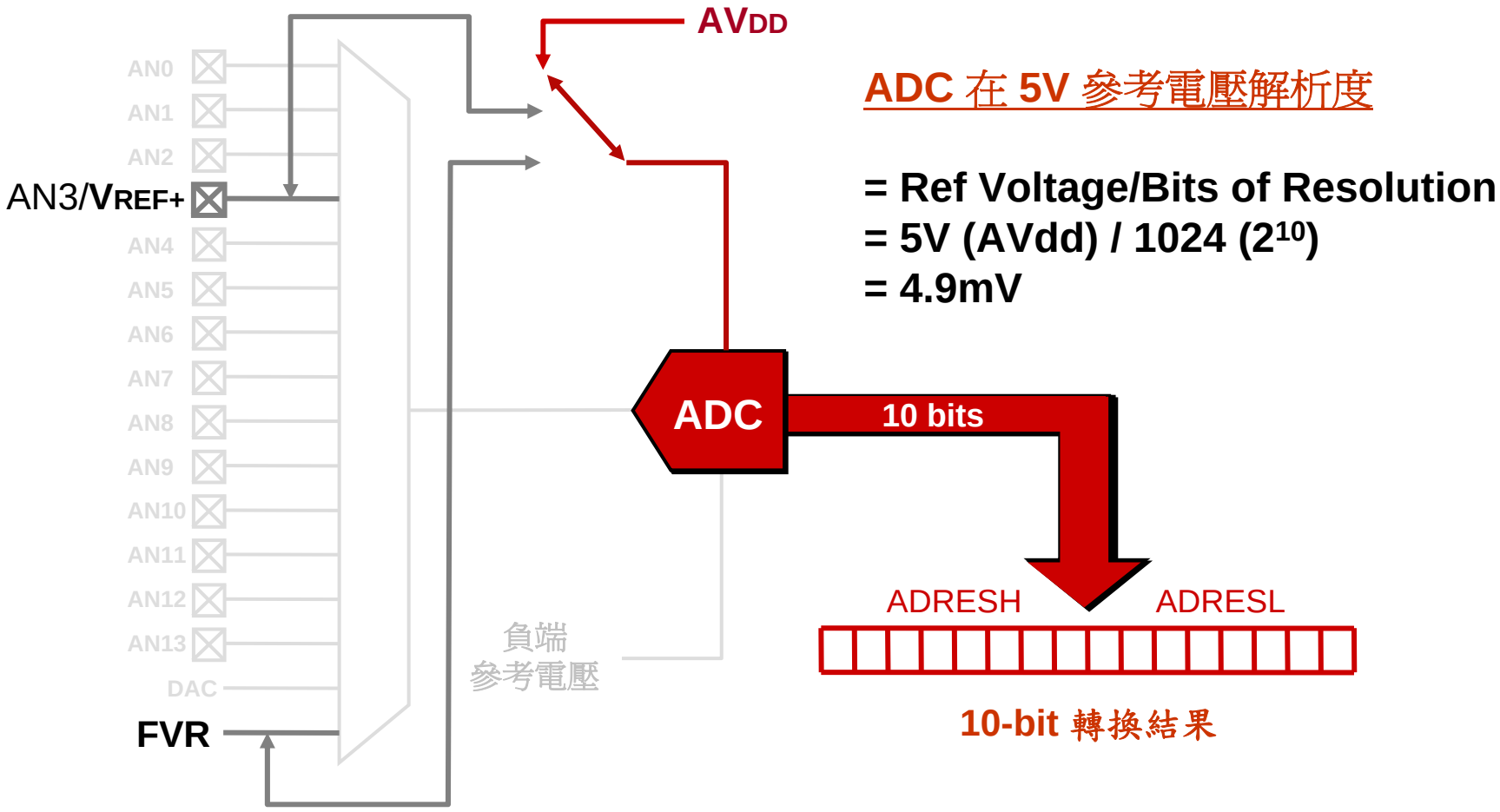
ADC 概述

- 軟體設定正端參考電壓源



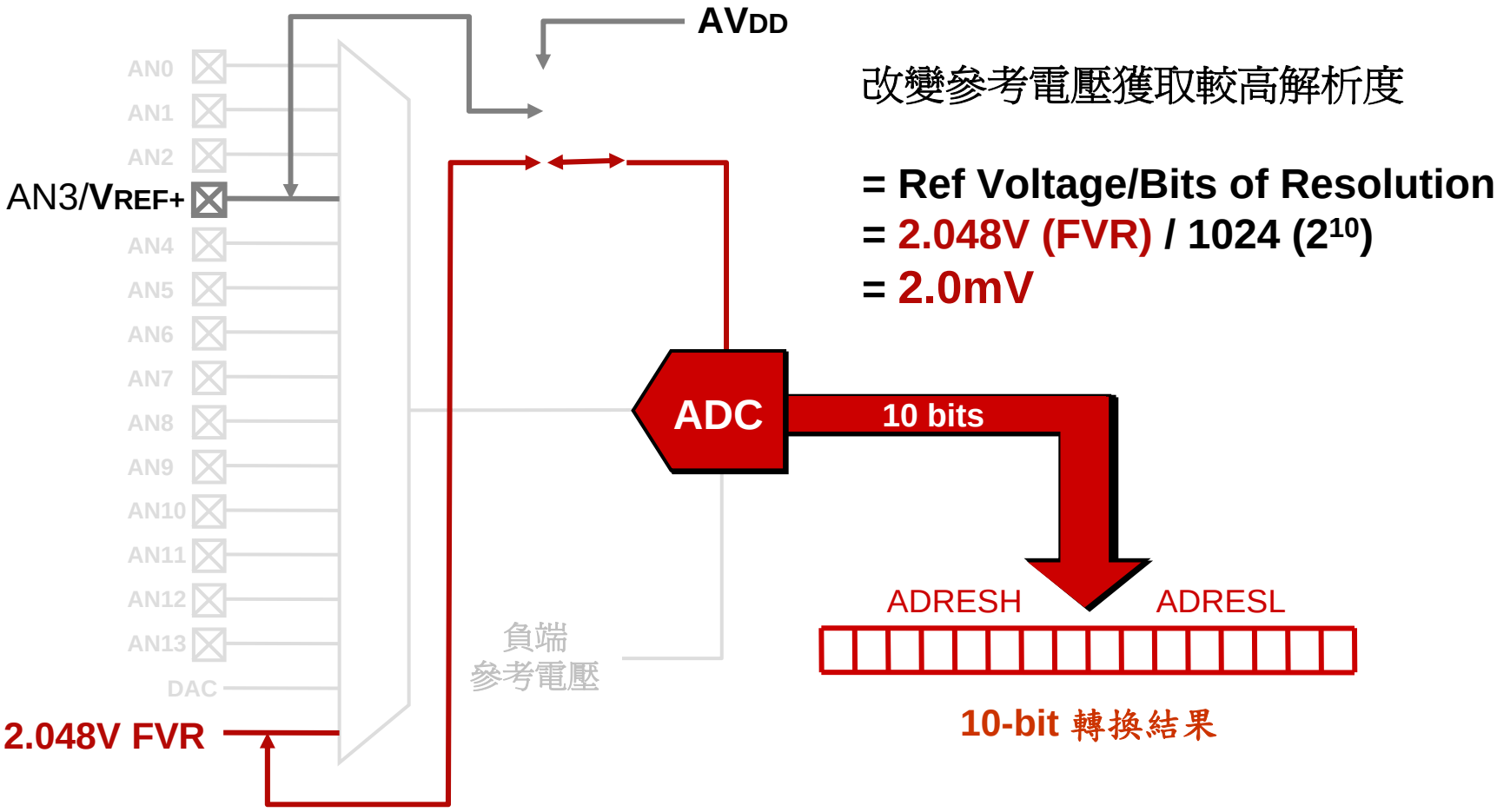
ADC 概述

- 利用參考電壓的設定增加解析度



ADC 概述

- 利用參考電壓的設定增加解析度

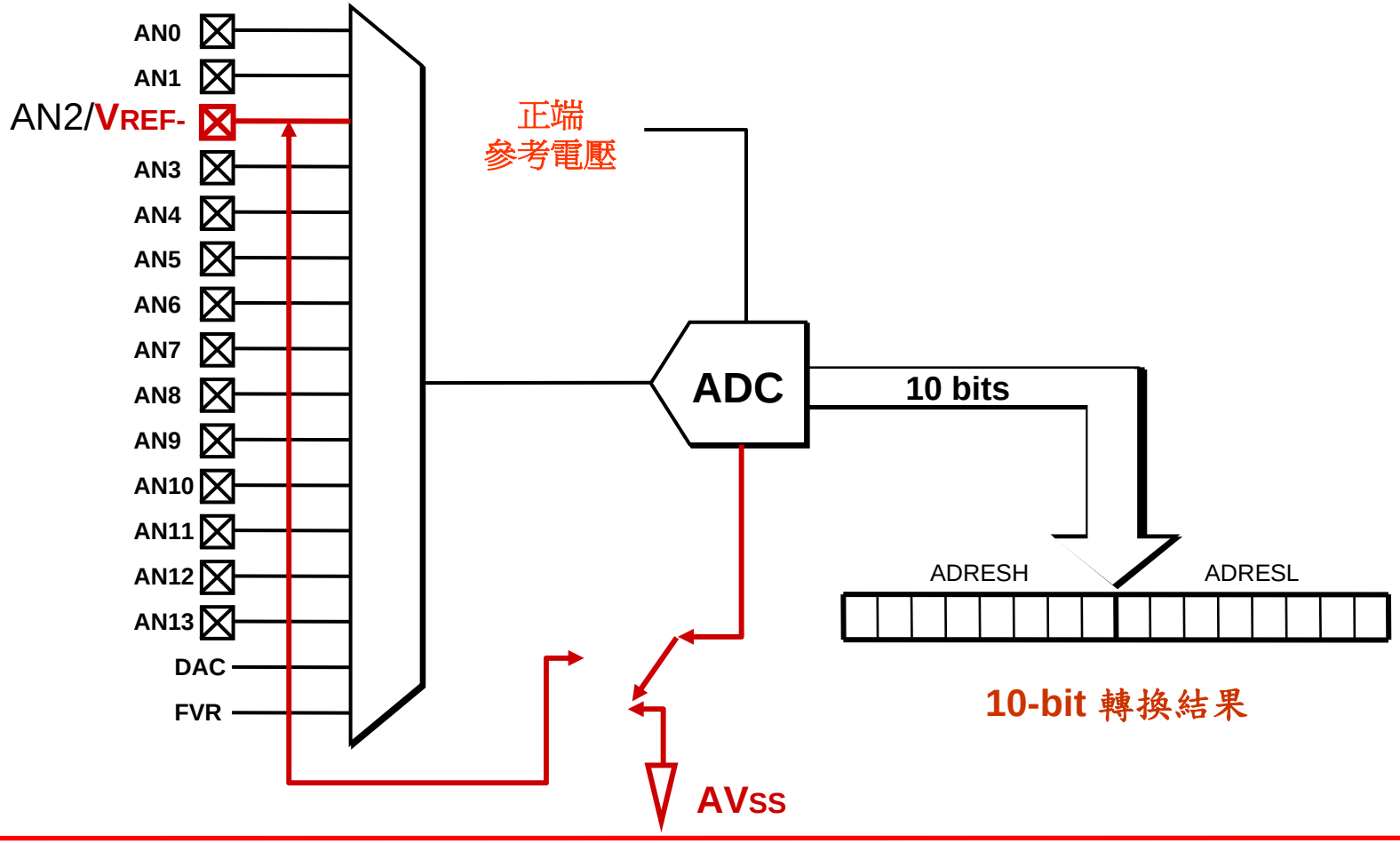


改變參考電壓獲取較高解析度

= Ref Voltage/Bits of Resolution
= 2.048V (FVR) / 1024 (2^{10})
= 2.0mV

ADC 概述

- 負端參考電壓設定



使用 **ADC**

- PIC16F1937 ADC 暫存器
  - ANSELA : PORTA 的類比輸入或 I/O 功能設定
  - ANSELB : PORTB 的類比輸入或 I/O 功能設定
  - ANSELE : PORTE 的類比輸入或 I/O 功能設定
  - ADCON0 : A/D CONTROL REGISTER 0
  - ADCON1 : A/D CONTROL REGISTER 1
  - ADRESH : ADC RESULT REGISTER HIGH
  - ADRESL : ADC RESULT REGISTER LOW

ADC 暫存器

- **ADCON0 控制暫存**

ADCON0

	CHS$	CHS3	CHS2	CHS1	CHS0	$\overline{\text{GO/DONE}}$	ADON

位元	功能
CHSx	類比通道選擇位元 **(AN0 ~ AN13)**
$\overline{\text{GO/DONE}}$	**1 = A/D 轉換正在進行** **0 = A/D 轉換完成**
ADON	**啟動 ADC 模組**

ADC 暫存器

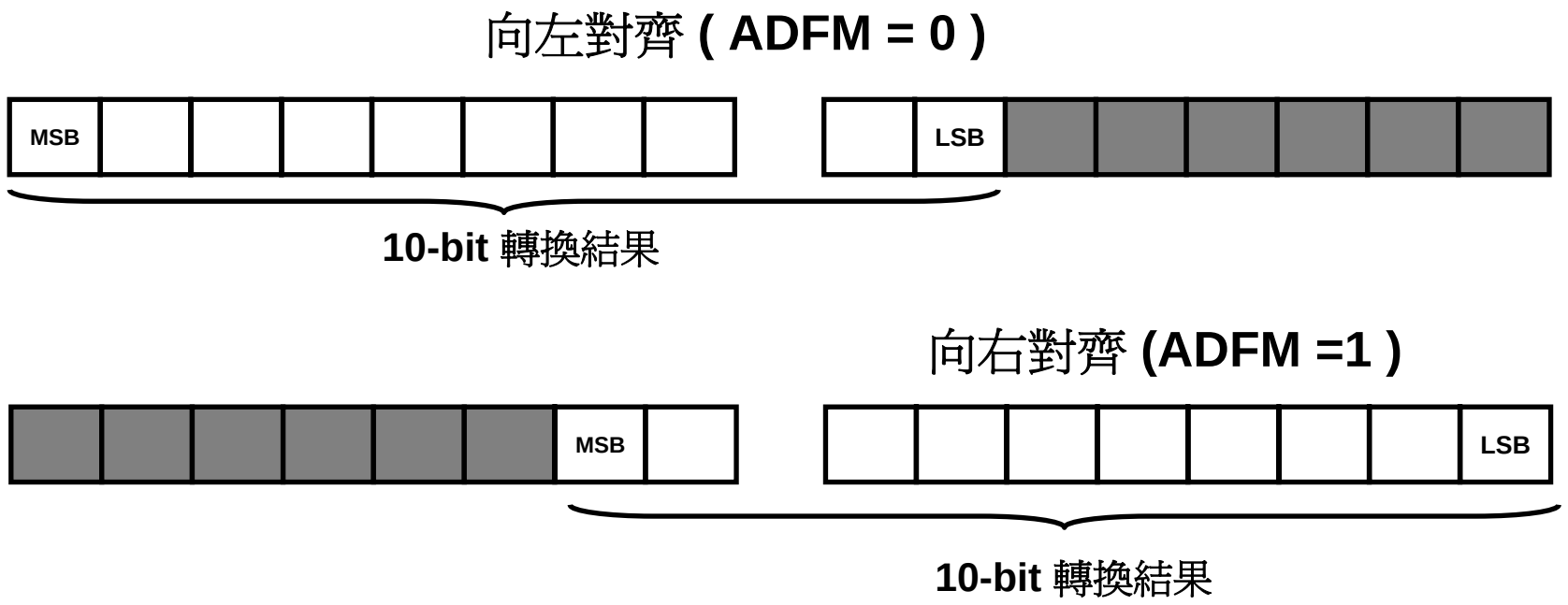
ADCON1

ADFM	ADCS2	ADCS1	ADCS0	---	ADNREF	ADPREF1	ADPREF0

位元	功能
ADFM	**AD** 轉換結果調整 **1 =** 向右靠齊調整， **0 =** 向左靠齊調整
ADCSx	**AD** 的 T_{AD} 時間設定 **(** T_{AD} 設定在 **Data Sheet** 裡規定不可小於 **1uS)** **(** 取樣時間須大於 **9.5uS)**
ADNREF **ADPREF0~1**	**AD** 負端參考電壓設定 **AD**正端參考電壓選擇

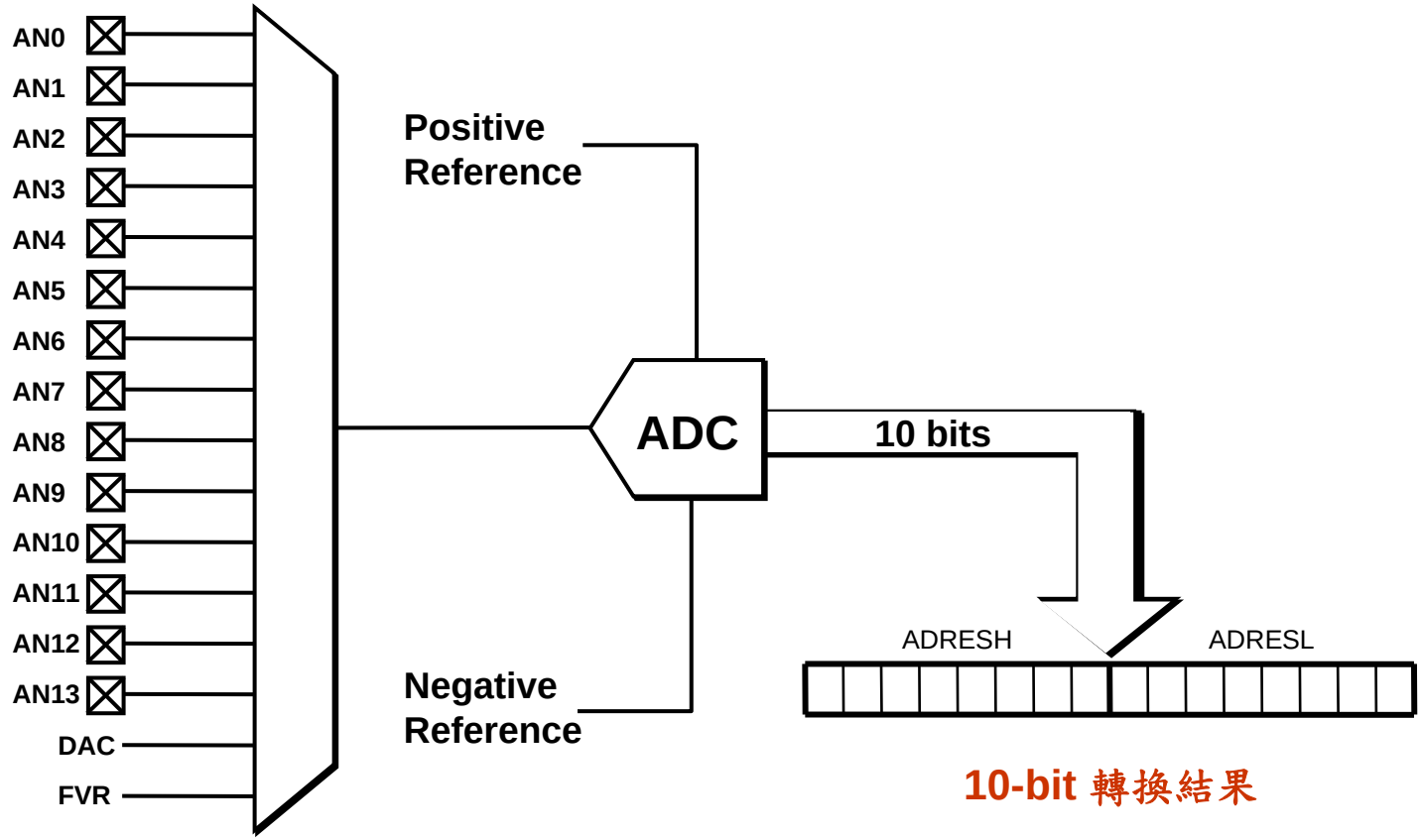
ADFM 位元

- 轉換完成後，**ADC** 轉換結果被放到個結果暫存器 **ADRESH** 和 **ADRESL**
- **10-bit ADC** 轉換結果可以向左對齊也可向右對齊



ADC 應用範例

- VDD 電壓偵測



Lab6 顯示輸入電壓值

- 修改原先的 **Lab5** 程式，加入 **ADC** 轉換並顯示 **VR2** 的電壓值
- 利用 **S5 & S6** 按鍵做功能切換動作
  - S5 : LED6 (綠色) 點亮，並顯示 VR2 電壓值
  - S6 : LED7 (紅色) 點亮，並顯示 CSM 的頻率

Lab6 程式

- 在 **Lab6** 專案裡的程式
  - 10-bit ADC.c ：ADC 電壓轉換與顯示
  - APP_EDF09 LCD.c : LCD 顯示函數
  - CSM_Touch.c : 中斷函數，CSM 頻率量測及 5mS 計時器，按鍵彈跳處理
  - Main.c : 主控程式及CSM頻率顯示
  - LCD_Defs.h : LCD 函數雛形宣告
  - Main.h : 函數雛形宣告

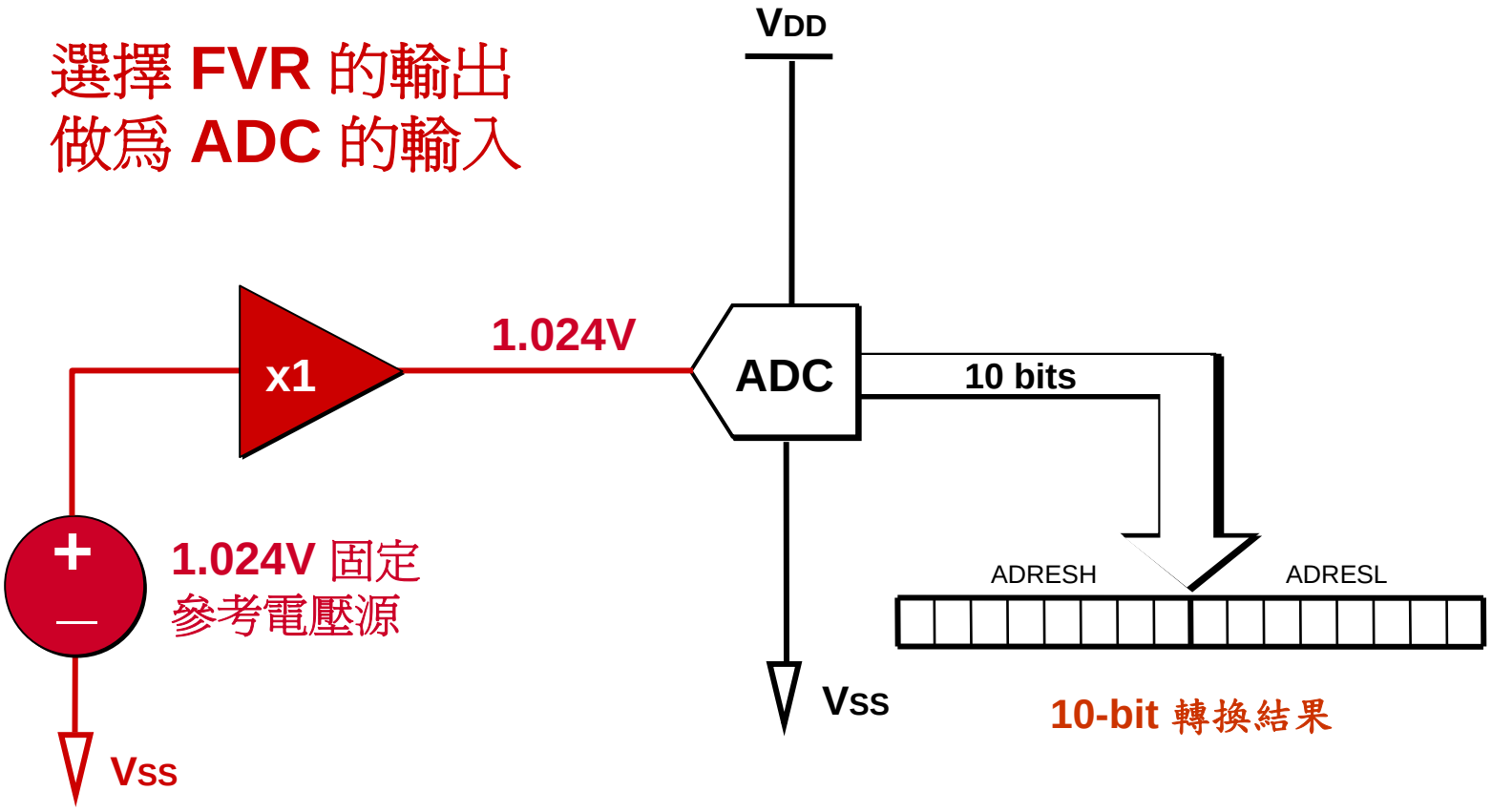
Lab6
ADC Voltage Meter

- 專案位置：
  - .. \Advance PICC Application Labs\Lab6 Voltage Meter\ADC for Voltage Meter.mcp

- 程式裡顯示電壓值到 **mV** 是怎樣做到的?
  - **Disp_Temp = (((unsigned long) Temp_Buff) * 4883) / 100 ;**
  - **5V / 1024 = 4.883 mV ,** 要顯示 **4.9mV** 的精度
  - 不要使用浮點數運算，只要用整數運算即可

ADC與參考電壓應用範例

- 使用 **1.024V** 參考電壓做 **Vdd** 掉電偵測



ADC與參考電壓應用範例

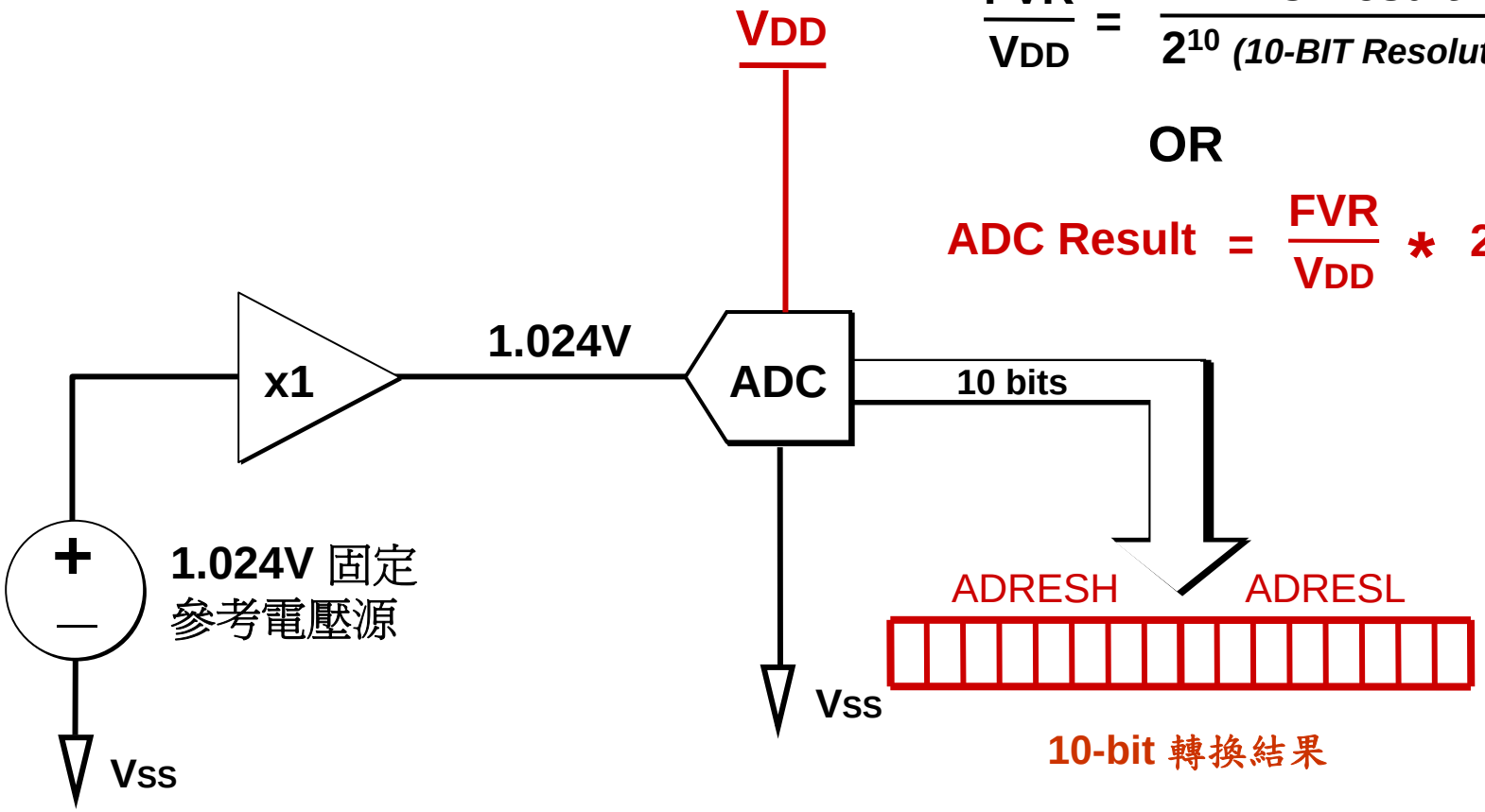
- VDD 掉電偵測

VDD vs. 1.024V (FVR)

$$\frac{FVR}{V_{DD}} = \frac{\text{ADC Result}}{2^{10}\ \textit{(10-BIT Resolution)}}$$

OR

$$\text{ADC Result} = \frac{FVR}{V_{DD}} * 2^{10}$$



ADC與參考電壓應用範例

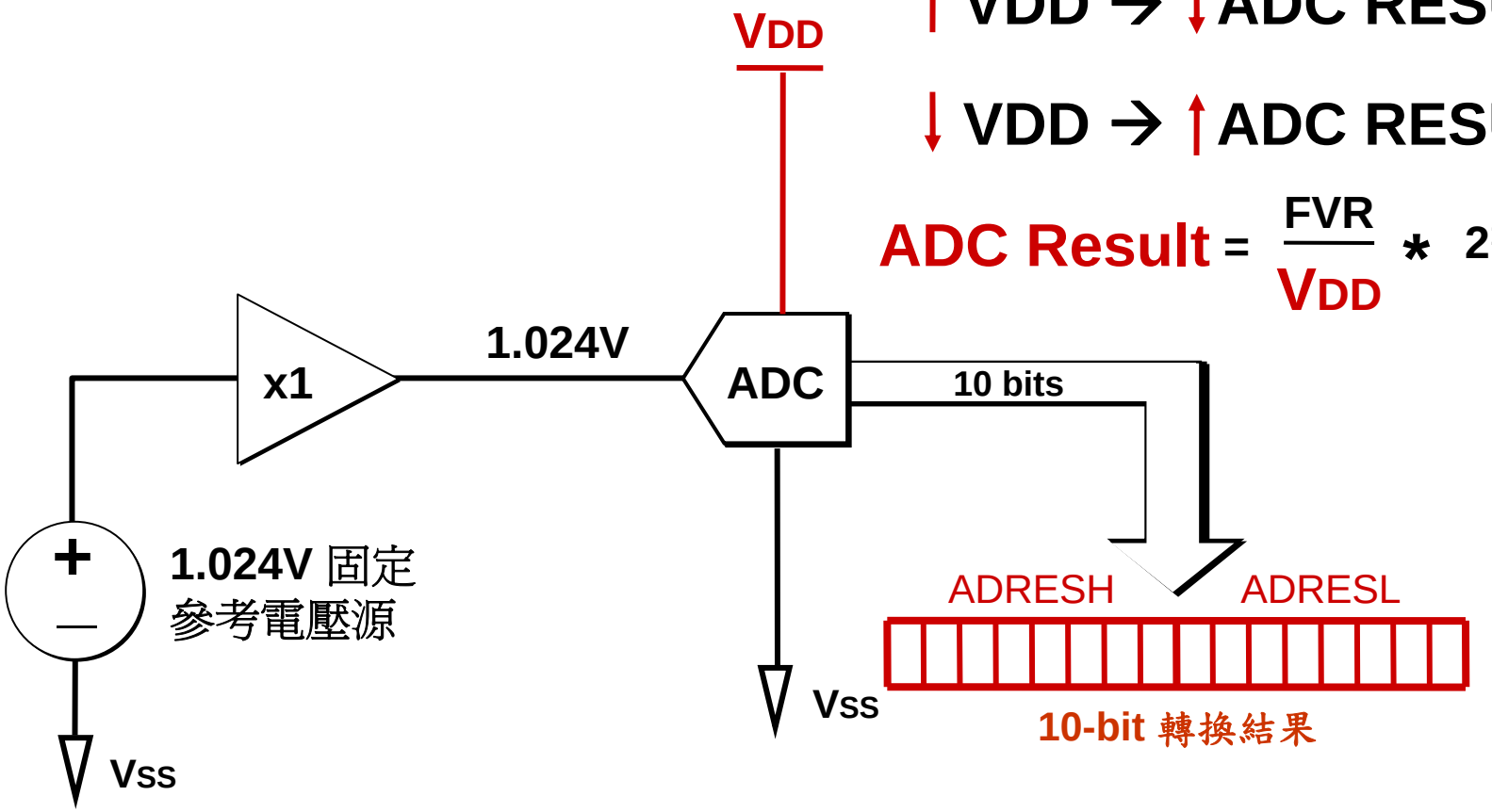
- VDD 掉電偵測

VDD vs. 1.024V (FVR)

↑ VDD → ↓ ADC RESULT

↓ VDD → ↑ ADC RESULT

$$\text{ADC Result} = \frac{\text{FVR}}{V_{DD}} * 2^{10}$$



MICROCHIP

I²C 介面操作

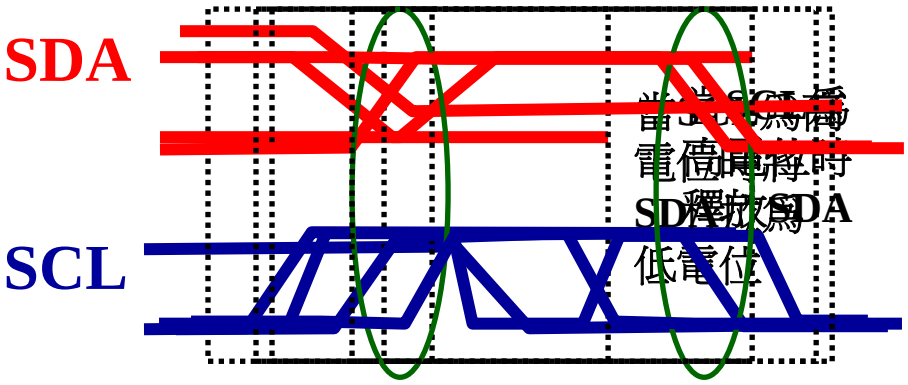
讀取 MCP9801 溫度感應器

MSSP 通訊模組

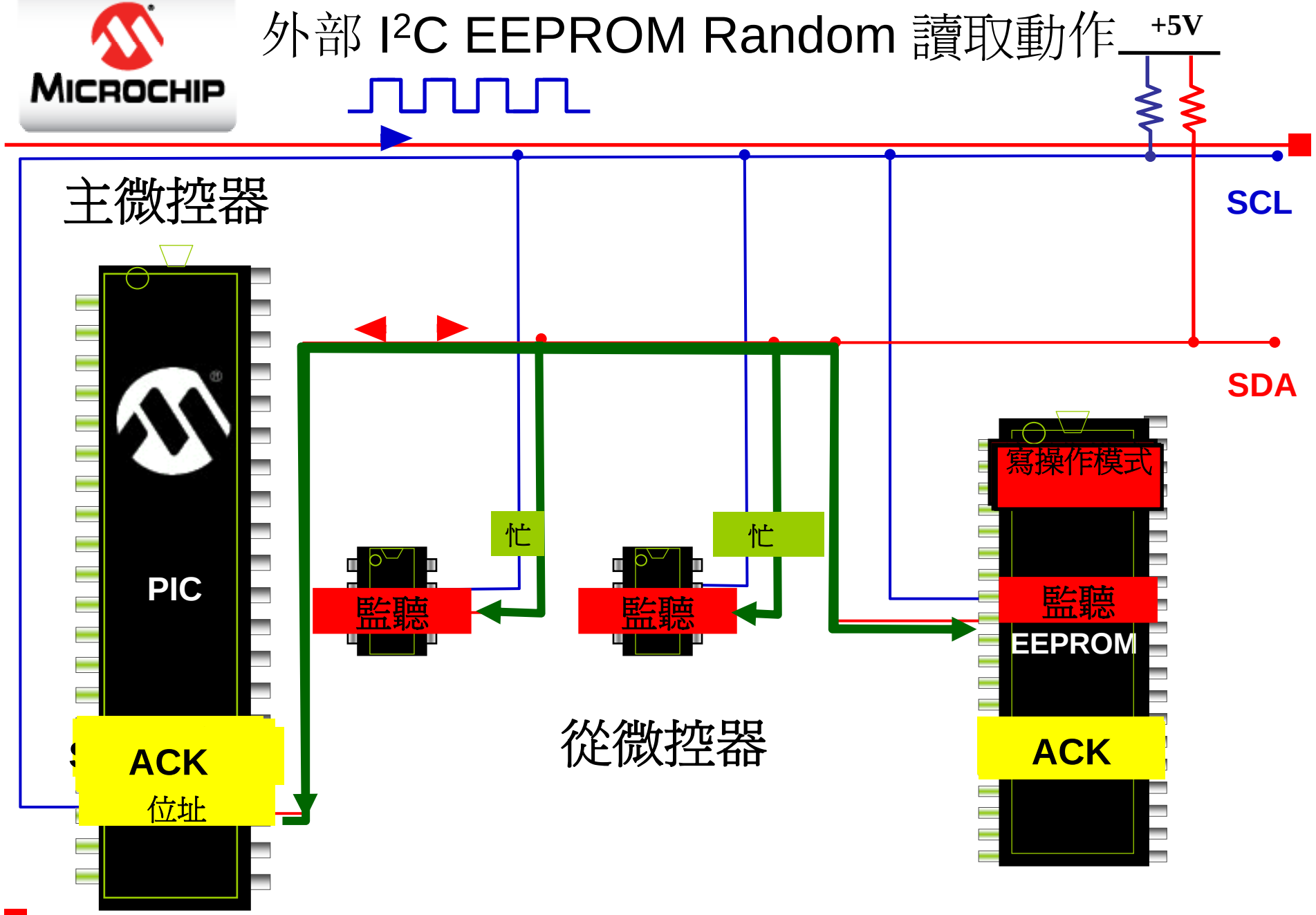
- **MSSP** 模組有以下兩種工作模式：
  - 串列周邊介面（**SPI**）
  - **I²C™**
    - 主模式 **(Master Mode)**
    - 從模式 **(Slave Mode)**（帶有廣播地址呼叫）
- **I²C** 介面通過硬體可以支援以下模式：
  - 主模式 (Master Mode)
  - 多主機模式 (Multi-Master Mode)
  - 從模式 (Slave Mode)

I²C 訊號條件

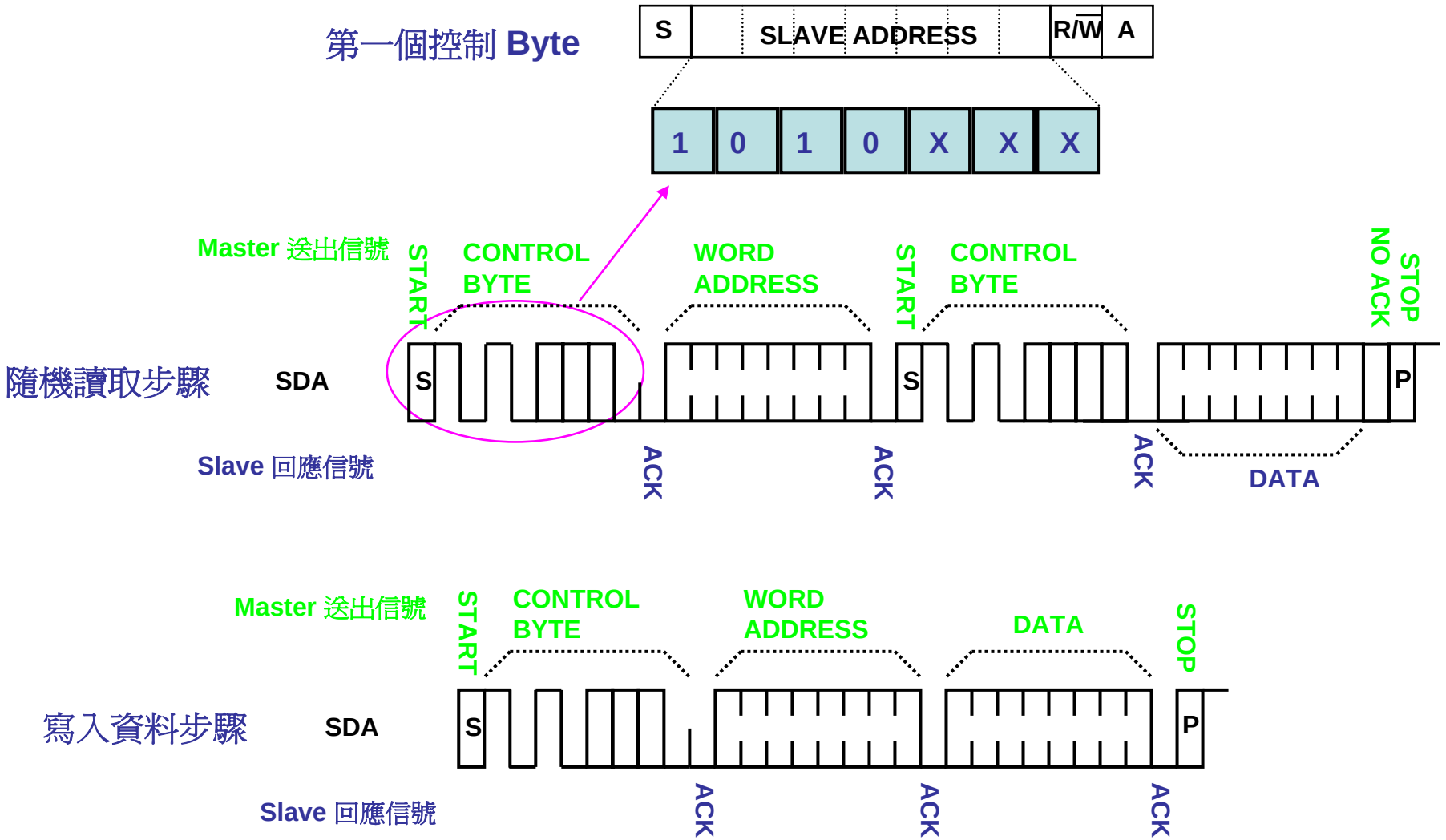
- 條件：
  - 啓動（S）
  - 停止（P）
  - 回應（Ack）
  - 重覆啓動（R）
  - 否定或無回應（Nack）



外部 I²C EEPROM Random 讀取動作

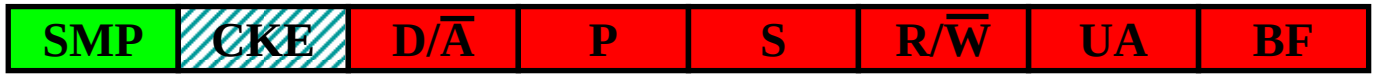


24LC02B 基本時序圖



MSSP 控制暫存器 (一)

- 有3 個相關的控制暫存器
  1. MSSP 狀態暫存器（SSPSTAT）



控制位元　　檢測位元（旗標）

位元	功能
SMP	斜率控制位元 (1= 100KHz, 0 = 400KHz Speed Rate)
CKE	I²C 模式下不使用
D/A	資料或位址指示位元(I2C Slave 專用)
P	檢測到停止條件
S	檢測到啓動條件
R/W̄	Slave 端：讀/寫命令，Master 端：正在發送
UA	地址需要更新 (10-bit Address)
BF	SSPBUF 暫存器滿

MSSP 控制暫存器(二)

SSPM3	SSPM2	SSPM1	SSPM0	模式
0	0	0	0	SPI主模式，時鐘 = FOSC/4
0	0	0	1	SPI主模式，時鐘 = FOSC/16
0	0	1	0	SPI主模式，時鐘 = FOSC/64
0	0	1	1	SPI主模式，時鐘 = TMR2輸出/2
0	1	0	0	SPI從模式，時鐘 = SCK接脚，啟動SS接脚控制
0	1	0	1	SPI從模式，時鐘 = SCK接脚，禁止SS接脚控制，SS可用作I/O接脚
0	1	1	0	I2C從模式，7-bit Address Mode
0	1	1	1	I2C從模式， 10-bit Address Mode
1	0	0	0	I2C主模式，時鐘 = FOSC / (4 * (SSPADD+1))
1	0	0	1	保留
1	0	1	0	保留
1	0	1	1	I2C 固件控制的主模式（從微控器空間）
1	1	0	0	保留
1	1	0	1	保留
1	1	1	0	I2C從模式，7-bit Address，並致能起始位元和停止位元中斷
1	1	1	1	I2C 從模式，10-bit Address，並致能起始位元和停止位元中斷

MSSP 控制暫存器 (三)

3. MSSP 控制暫存器 2 （SSPCON2）

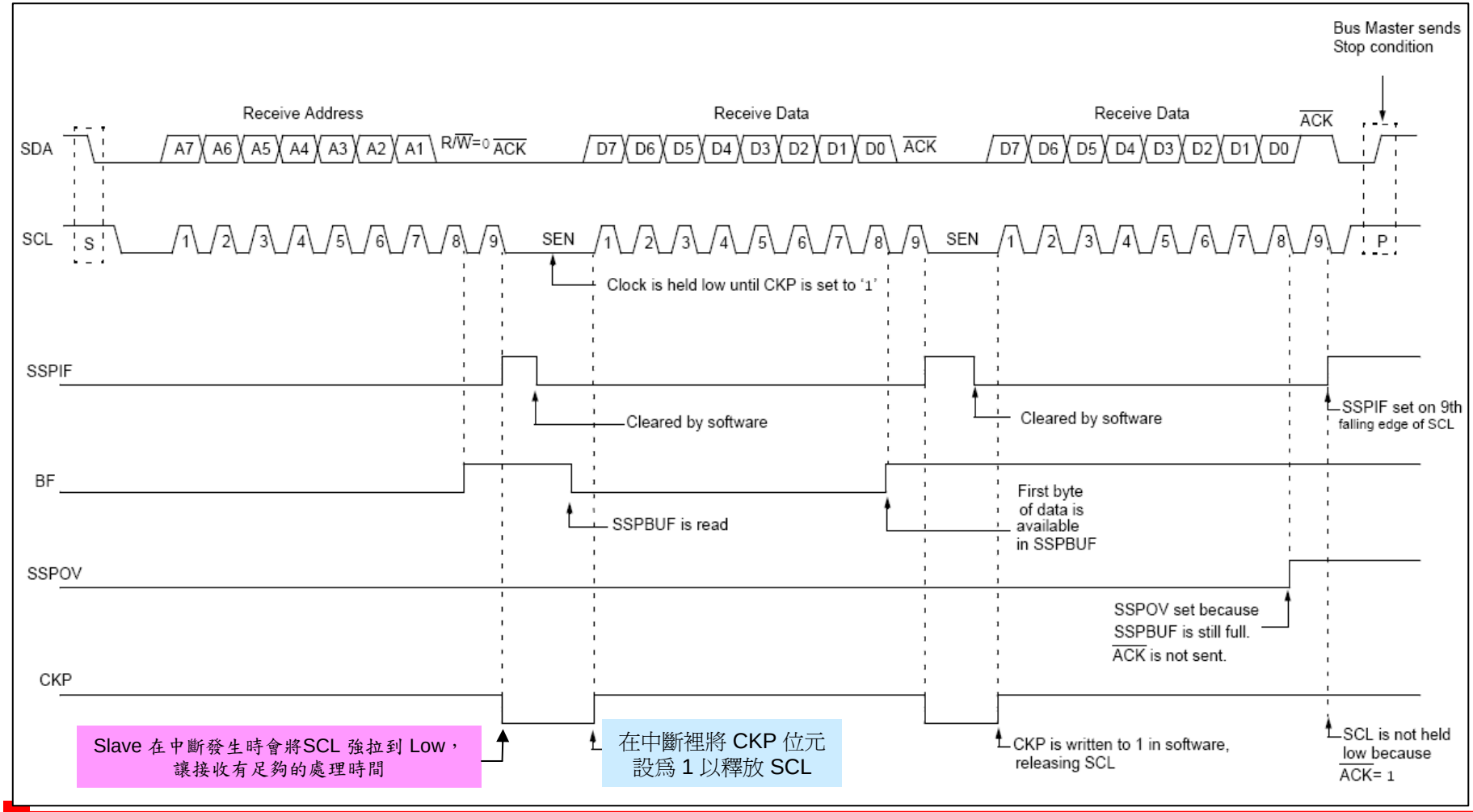
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN

控制位元　　檢測位元（旗標）

位元	功能
GCEN	接收到共用位址時**(0x00)**產生中斷（**Slave Only**）
ACKSTAT	指示接收端 **Slave** 的回應信號；**0 = ACK , 1 = NACK (**發送模式**)**
ACKDT	設定 **ACK/NACK** 位元資料；**0 = ACK , 1 = NACK**
ACKEN	發出 **ACK/NACK** 條件（發送**ACKDT**位元）**(Master Mode)**
RCEN	啓動接收模式 **(Master Mode)**
PEN	發出停止條件 **(Master Mode)**
RSEN	發出重複啓動條件 **(Master Mode)**
SEN	發出啓動條件 **(Master Mode)**

I²C 7-bit Address 接收時序

- **I²C SLAVE, 7-BIT ADDRESS, RECEPTION**



與 I²C 相關的暫存器

- I²C Slave Address（SSPADD）：
  - Slave Mode：
    - ✓ 設定 PIC 微控器的 I²C Slave Address
    - ✓ 與接收到的位址值進行比較 (ADD0 不參與比較)
  - Master Mode：
    - ✓ 用於計算 I²C 系統的時脈速率（速率）

$$\text{速率} = \frac{Fosc}{4\times(SPADD+1)}$$

注意：**Fosc** 是振盪器頻率，而非指令週期 **T_{CY}**

MSSP 中斷

- 發生以下事件時，**PIR1暫存器中的 SSPIF 中斷旗標會設 1**
  - 啓動條件
  - 停止條件
  - 資料傳輸 (Byte) 已發送/已接收
  - 發送 ACK/NACK
  - 重覆啓動條件

當設定了 PIE1<SSPIE> 位元以及 INTCON 中的GIE 和 PEIE 位元時，才會產生 SSP 的中斷。

MCP9800 I²C 溫度感應器

- 溫度對數位轉換，**I²C** 介面輸出
- 精確度: **0.5°C (室溫)，1°C (-10°C ~ 85°C)**
- 可選擇 **9 ~ 12 bit** 的解析度輸出
- 工作電壓 : **2.7V ~5.5V**
- 包裝 : SOT-23-5, MSOP-8, SOIC-8

- **APP-EDF09 使用 MCP9800A5T-M/OT**
  - A5 : I2C Slave Address
  - T : Tape & Reel
  - M : -55°C ~ +125°C
  - OT : SOT-23 5 lead

MCP9800 設定

- **MCP9800**
  - 採用 Σ Δ ADC 轉換架構
  - 設定溫度解析度爲 12-Bit，1Lsb = 0.0625ºC
  - 12-bit 解析度下，轉換時間不可小於 600mS
  - 內建四個可被存取的暫存器
    - Temperature Register (MSB + LSB) (0x00)
    - Config Register (Point Address 0x01)
    - Hysteresis Register (0x02)
    - LIMIT-SET Register (0x03)
  - MCP9800 內部暫存器是採用指標暫存器來存取的

MCP9800 溫度讀取

- 設定 **Config** 暫存器的 **I²C** 命令
  - Start + A5 address w/ write + Pointer + Config Data + Stop
- 讀取 **12-bit** 溫度值
  - Start + Address w/ Write + pointer + Re-Start + Address w/ Read + Read MSB + Read Lsb + Stop

	2's 二進制輸出值	十進制輸出	溫度值
12-bit 解析度	0111 1101 0000 uuuu	2000	+125°C
	0000 0000 0001 uuuu	1	+0.0625°C
	0000 0000 0000 uuuu	0	0°C
	1111 1111 1111 uuuu	- 1	- 0.0625°C
	1100 1001 0000 uuuu	- 880	- 55°C

Lab7
用 MCP9800 做一數位溫度計

- 專案位置：
  - .. \Advance PICC Application Labs\Lab7 Temperature Meter\MCP9800 Temperature Meter.mcp

- **Hi-Tech PICC 只提供軟體 I²C 沒有使用 MSSP 模組的 I²C 函數?**
  - 將 C18 所提供的 I2C 函數原始程式改成 PIC16F1xxx 的函數
  - I2C 程式 : MCP9800 IIC Function.c
  - Head File : MCP9800 IIC.h

Lab7 中對 MCP9800 定義

- **MCP9800 IIC.h**

```
#define   SCL          RC3          // I2C 接腳
#define   SDA          RC4
#define   SCL_DIR      TRISC3
#define   SDA_DIR      TRISC4

// MCP9800 各項命令需參考 Data Sheet
#define   MCP9800_Addr_Read      0b10011011
#define   MCP9800_Addr_Write     0b10011010
#define   MCP9800_Point_Temp     0b00000000
#define   MCP9800_Point_Conf     0b00000001
#define   MCP9800_Point_Hyst     0b00000010
#define   MCP9800_Point_Limit    0b00000011
```

Lab7 使用 I²C 函數

- **void Initialize_I2C_Master(void);**
  - 設定 MSSP 爲 I2C Master, 7-bit Address,
  - 100KHz Speed Rate
- **void Read_Temperature(void);**
  - 讀取 12-bit 溫度值到溫度 Buffer
- **void Write_MCP9800_Config(void);**
  - 設定MCP9800 爲 12-bit 解析度
  - Shutdown Mode 關閉
- **void IdleI2C(void);**
  - 測試 I2C Bus 是否閒置。
  - SSPCON2 的 ACKEN, RCEN, PEN, RSEN & SEN == 0
- **void StartI2C(void);**
  - 送出 Start Condition

Lab7 使用 I²C 函數

- **unsigned char WriteI2C(unsigned char);**
  - 將傳入的參數送到 I2C Bus，並回傳狀態
    - -1 : Write Collision
    - -2 : 沒收到對方的回應 (Nack)
    - 0 : 傳送正常
- **unsigned char ReadI2C(void);**
  - 連續送出八個 SCL 時脈給 Slave
  - 讀取一個 Byte 資料
- **void AckI2C(void);**
  - 送出一個 Ack 訊號
- **void NotAckI2C(void);**
  - 送出一個 Nack 訊號

Lab7 使用 I²C 函數

- **void RestartI2C(void);**
  - 送出 Repeat Start Condition
- **void StopI2C(void);**
  - 送出 Stop Condition
- **void I2C_Done(void);**
  - I2C 每做完一個動作後的檢查函數，以確保下一動作時間點的正確性

```
void I2C_Done(void)
{
      while (!SSPIF);     // Completed the action when the SSPIF is Hi.
      SSPIF=0;            // Clear SSPIF
}
```