實驗一 基礎I/O實驗

【目的】

_ 瞭解 NuEdu-SDK-M45x 系列處理器基本特性及規格
_ 瞭解 ARM® Cortex®-Mx 系列處理器的優點
_ Keil uVersion開發工具軟體的使用

【實驗背景】

C 語言

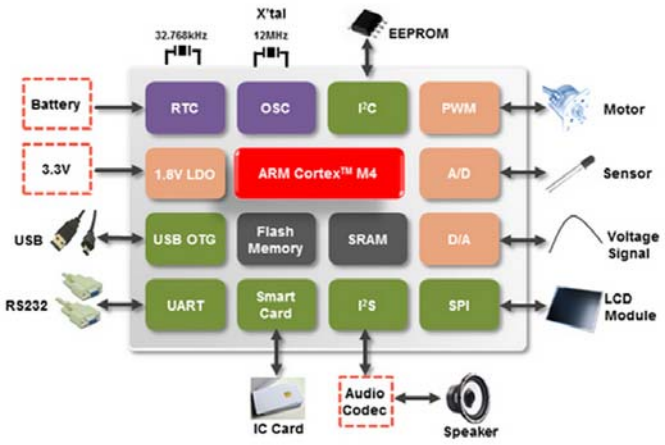
【原理與說明】

一、瞭解 NuEdu-SDK-M45x 處理器

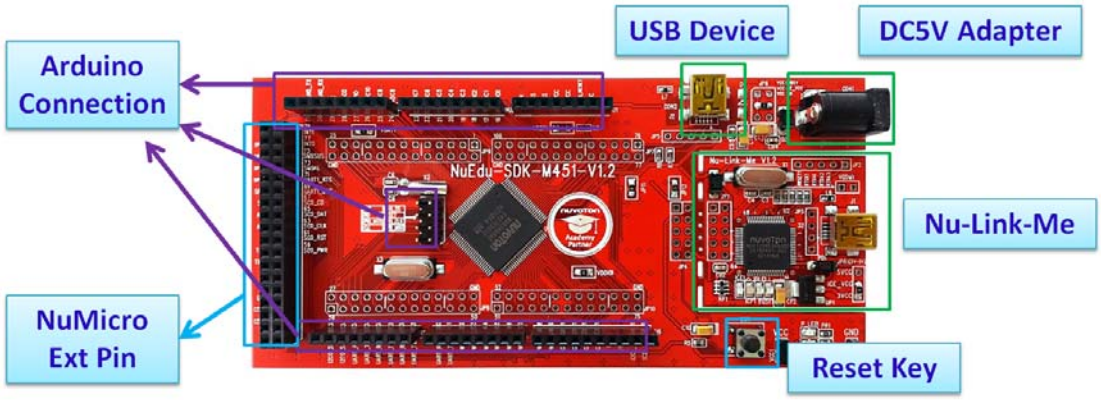
基本特性及規格簡介：

- **Core**
  - ARM® Cortex™-M4 with DSP and FPU
  - Max frequency of 72 MHz
  - Operating voltage: 2.5V to 5.5V
  - Temperature range: -40℃ ~ 105℃
- **Memory**
  - 128/256 KB of Flash Memory
  - 32 KB of SRAM
  - Data Flash configurable
- **12-bit ADC (up to 16 channels)**
- **12-bit DAC**
- **16-bit PWM (up to 12 channels)**
- **Timers**
  - Four timers
  - RTC (optional)
- **Connectivity**
  - USB 2.0 OTG (optional)
  - CAN (optional)
  - Up to 5 UARTs
  - Up to three SPIs
  - Up to two I²Cs (up to 1 MHz)
  - Smart card interfaces (optional)
  - Up to two I²S interfaces
- **Security**
  - CRC engine
- **Clock Control**
  - 4 to 24 MHz external crystal oscillator
  - 32.768 kHz crystal oscillator
  - 22.1184 MHz internal RC oscillator
  - 10 kHz internal RC oscillator

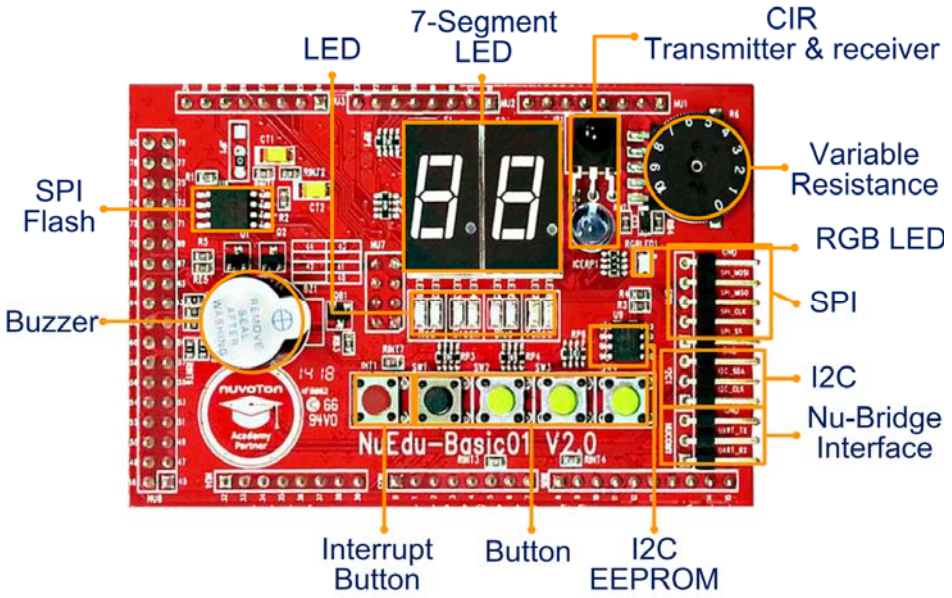
• M453VG6AE 功能方塊圖：



• NuEdu-EVB-M451 週邊硬體組成：



• NuEdu-Basic01週邊硬體組成:



二、_ 瞭解 M45x 系列處理器的優點

M45x系列處理器的優點:

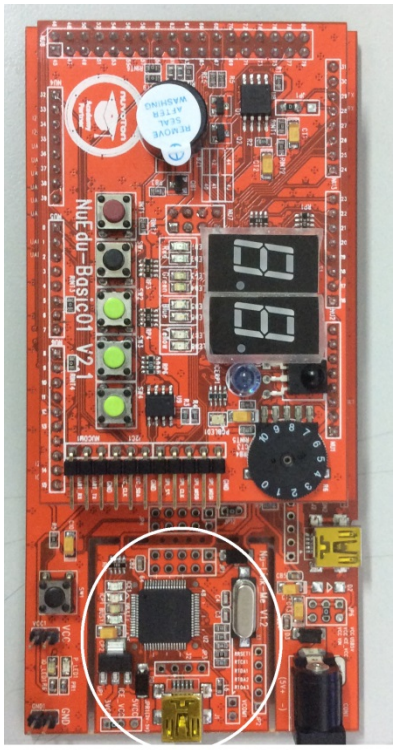
三、Keil uVersion開發工具軟體的使用

詳見power point

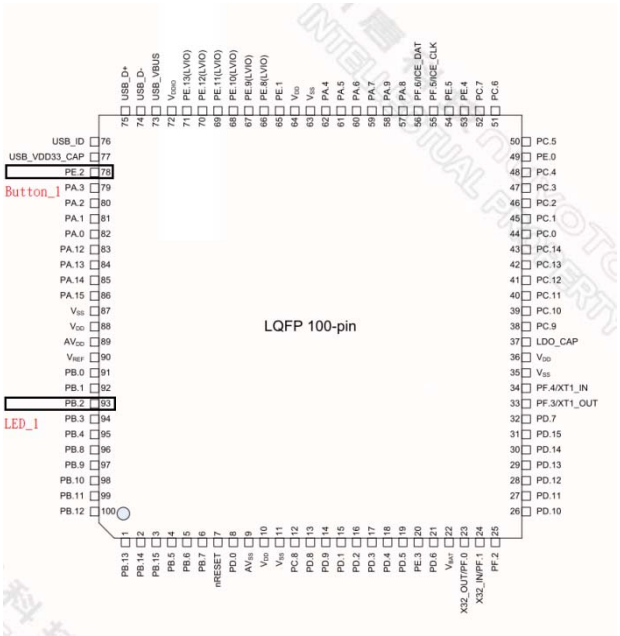
四、LED控制實驗

範例實驗:

使用button控制LED亮暗;從以下兩張圖可以看出想要控制Button_1及LED_1必須要控制M453VG6AE這顆IC的PE.2和PB.2這兩個BIT。



Debugger



承上段，PE.2(Button)必須被設定為輸入；PB.2(LED)必須被設定為輸出，參考下圖的暫存器格式資料可以知道:

1. PE_MODE暫存器的BIT5和BIT4必須設為0和0
2. PB_MODE暫存器的BIT5和BIT4必須設為0和1
3. PE_DINOFF暫存器的BIT18代表輸入的電位(0高1低)
4. PB_DOUT暫存器的BIT2控制LED_1的亮暗(0亮1暗)。

Bits	Description	
[2n+1:2n] n=0,1..15	**MODEn**	**Port A-F I/O Pin[N] Mode Control** Determine each I/O mode of Px.n pins. 00 = Px.n is in Input mode. 01 = Px.n is in Push-pull Output mode. 10 = Px.n is in Open-drain Output mode. 11 = Px.n is in Quasi-bidirectional mode. **Note1:** The initial value of this field is defined by CIOINI (CONFIG0 [10]). If CIOINI is set to 0, the default value is 0xFFFF_FFFF and all pins will be quasi-bidirectional mode after chip powered on. If CIOINI is set to 1, the default value is 0x0000_0000 and all pins will be input mode after chip powered on. **Note2:** Max. n=15 for port A/B/C/D. Max. n=14 for port E. Max. n=7 for port F.

Bits	Description	
[n+16] n=0,1..15	**DINOFF[n]**	**Port A-F Pin[N] Digital Input Path Disable Control** Each of these bits is used to control if the digital input path of corresponding Px.n pin is disabled. If input is analog signal, users can disable Px.n digital input path to avoid input current leakage. 0 = Px.n digital input path Enabled. 1 = Px.n digital input path Disabled (digital input tied to low). **Note:** Max. n=15 for port A/B/C/D. Max. n=14 for port E. Max. n=7 for port F.
[15:0]	**Reserved**	Reserved.

Bits	Description	
[31:16]	**Reserved**	Reserved.
[n] n=0,1..15	**DOUT[n]**	**Port A-F Pin[N] Output Value** Each of these bits controls the status of a Px.n pin when the Px.n is configured as Push-pull output, Open-drain output or Quasi-bidirectional mode. 0 = Px.n will drive Low if the Px.n pin is configured as Push-pull output, Open-drain output or Quasi-bidirectional mode. 1 = Px.n will drive High if the Px.n pin is configured as Push-pull output or Quasi-bidirectional mode. **Note:** Max. n=15 for port A/B/C/D. Max. n=14 for port E. Max. n=7 for port F.

五、LED顯示控制程式碼：

範例程式:

```
#include "M451Series.h"

int __main()
{
  /*---------initial--------*/
  PE_MODE |= ~(BIT5 + BIT4);
  PB_MODE |= ~(BIT5 + (~BIT4));

  while(1)
  {
    /*write your code here*/
    if(PE2==0)
      PB2=0;//set LED high
    else
      PB2=1;//set LED low
  }
}
```

註:暫存器位置(GPIO_BA = 0PB000)

Port A-F I/O Mode Control (Px_MODE)

Register	Offset	R/W	Description	Reset Value
PA_MODE	GPIO_BA+0x000	R/W	PA I/O Mode Control	0xXXXX_XXXX
PB_MODE	GPIO_BA+0x040	R/W	PB I/O Mode Control	0xXXXX_XXXX
PC_MODE	GPIO_BA+0x080	R/W	PC I/O Mode Control	0xXXXX_XXXX
PD_MODE	GPIO_BA+0x0C0	R/W	PD I/O Mode Control	0xXXXX_XXXX
PE_MODE	GPIO_BA+0x100	R/W	PE I/O Mode Control	0xXXXX_XXXX
PF_MODE	GPIO_BA+0x140	R/W	PF I/O Mode Control	0x0000_XXXX

Port A-F Digital Input Path Disable Control (Px_DINOFF)

Register	Offset	R/W	Description	Reset Value
PA_DINOFF	GPIO_BA+0x004	R/W	PA Digital Input Path Disable Control	0x0000_0000
PB_DINOFF	GPIO_BA+0x044	R/W	PB Digital Input Path Disable Control	0x0000_0000
PC_DINOFF	GPIO_BA+0x084	R/W	PC Digital Input Path Disable Control	0x0000_0000
PD_DINOFF	GPIO_BA+0x0C4	R/W	PD Digital Input Path Disable Control	0x0000_0000
PE_DINOFF	GPIO_BA+0x104	R/W	PE Digital Input Path Disable Control	0x0000_0000
PF_DINOFF	GPIO_BA+0x144	R/W	PF Digital Input Path Disable Control	0x0000_0000

Port A-F Data Output Value (Px_DOUT)

Register	Offset	R/W	Description	Reset Value
PA_DOUT	GPIO_BA+0x008	R/W	PA Data Output Value	0x0000_FFFF
PB_DOUT	GPIO_BA+0x048	R/W	PB Data Output Value	0x0000_FFFF
PC_DOUT	GPIO_BA+0x088	R/W	PC Data Output Value	0x0000_FFFF
PD_DOUT	GPIO_BA+0x0C8	R/W	PD Data Output Value	0x0000_FFFF
PE_DOUT	GPIO_BA+0x108	R/W	PE Data Output Value	0x0000_FFFF
PF_DOUT	GPIO_BA+0x148	R/W	PF Data Output Value	0x0000_00FF

六、實作：

課堂上實作內容：

利用4個按鍵控制設計出7個LED的亮暗模式。

實驗報告：

報告內容包含程式碼(加註解)、程式碼流程、心得。

實驗二 中斷處理程式與時鐘

【目的】

壹、了解中斷處理

貳、了解Timer設定流程用以產生定時中斷

參、學習如何透過配置相關的暫存器來產生定時中斷及七段顯示器

【實驗背景】

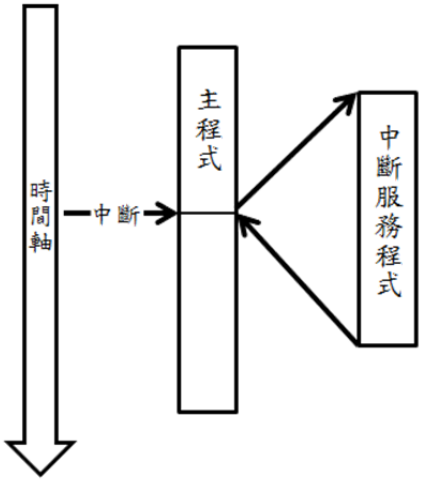
- C 語言

【原理與說明】

壹、了解中斷處理

1. 中斷介紹：

所謂中斷是指處理器接收到外圍硬體的非同步信號，或者來自軟體的同步信號而進行相應的硬體／軟體處理。處理器每執行完一個指令，它就會檢查中斷暫存器看看是否有中斷發生，當有中斷發生的時候，處理器會根據中斷向量表來決定要執行哪一個中斷服務程式。當處理器發現有中斷發生時，它會停止現在執行的程式，然後跳去執行中斷服務程式，中斷服務程式執行完後，那麼處理器要跳回剛剛發生中斷的地方繼續執行，可參考圖一的示意圖。不過有一點同學要注意，當處理器要執行中斷服務程式前，必需先將當前的暫存器程系統資訊先儲存好，這樣子處理完中斷服務程式後才可以恢復到原來的狀態。



圖一 中斷處理示意圖

貳、了解Timer設定流程用以產生定時中斷

1. Clock Controller：

時鐘控制器為整個晶片提供時鐘源，包括系統時鐘和所有週邊設備時鐘。其向量位址位於4000_0200h 到 4000_0280h之間，而整體架構圖如下所示。本次實驗只需要將Timer0的時鐘開啟並設定Timer0的時鐘來源(紅色部分)。

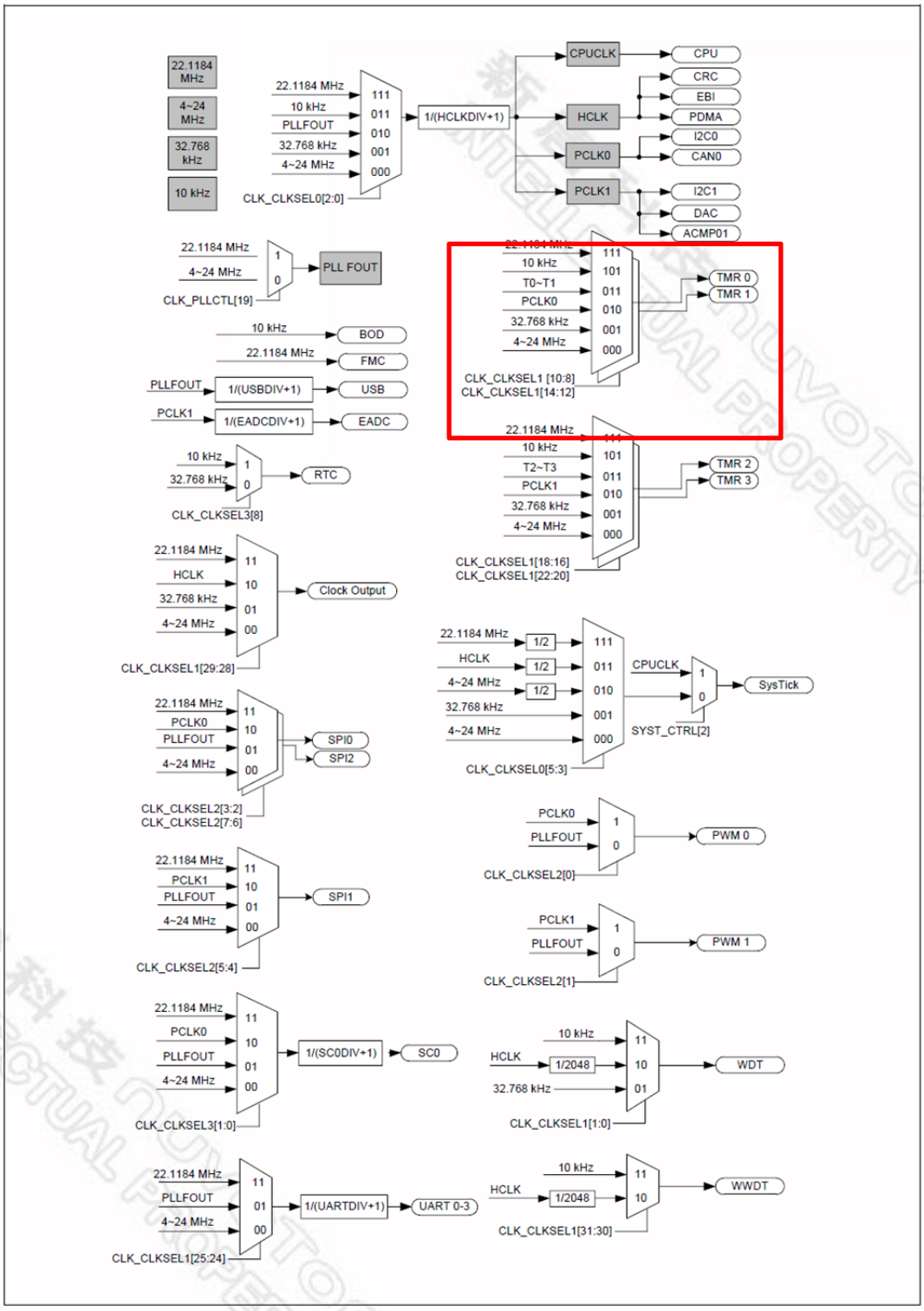


圖2-1 Clock Controller架構圖

設定完Timer0的時鐘來源後就開始設定Timer0在經過幾個clock cycle才會產生一次中斷。此次的Timer0 clock為HXT(12 MHz)，若我們想每 1 ms產生一次中斷，則必須設定成每經過12000個clock cycle產生一次中斷。以下來介紹Timer Controller的架構、向量位址以及如何設定成每經過約12000個clock cycle產生一次中斷。

2. Timer Controller：

Timer Controller包含 4 組 32-bit定時器(TIMER0~TIMER3)。其向量位址位於4005_0000h 到 4005_003Ch 及4005_1000h 到 4005_103Ch之間，而整體架構圖如下所示。

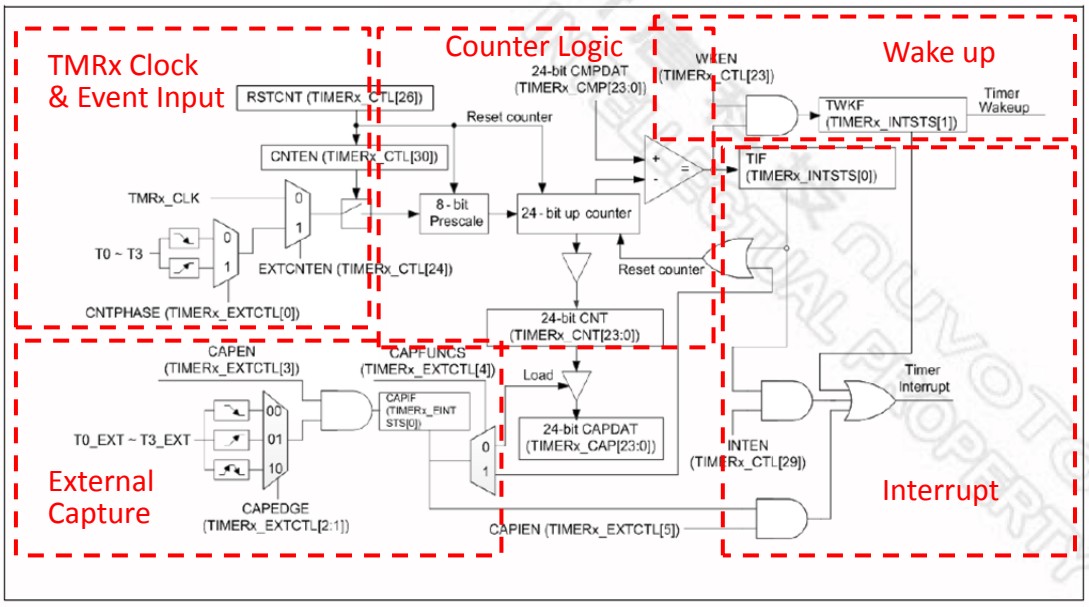


圖2-2 Timer Controller架構圖

本次實驗只需要設定Timer0，首先設定Timer0的計數模式。共有以下四種計數模式：(本次實驗Timer0的計數模式為periodic模式)

(1). One-shot模式

當CNTEN置1，則CNT計數器會由0開始計數，之後每個Timer0 clock cycle皆加1(本次實驗的Timer0 clock為HXT)，一旦CNT計數器的值達到CMPDAT的值時，TIF 標誌將變為1，此時CNT的值和CNTEN位將由定時器控制器自動清零，然後定時器計數操作停止。並且在INTEN位使能，則定時器中斷訊號產生並送到NVIC通知CPU。(示意圖如下)

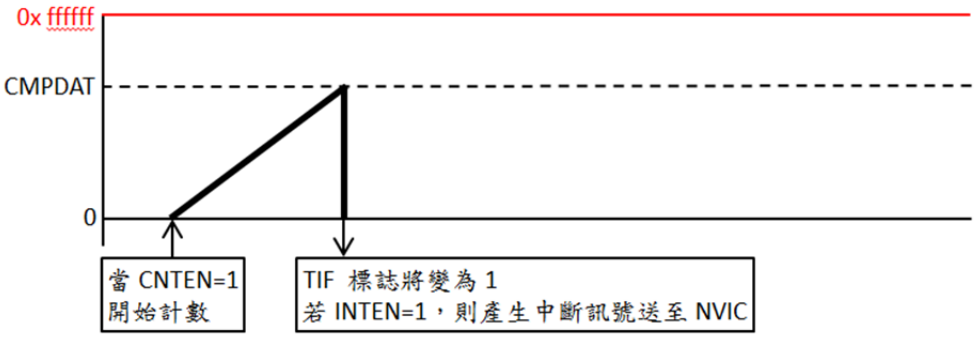


圖2-3 One-shot模式示意圖

(2). Periodic模式

當CNTEN置1，則CNT計數器會由0開始計數，之後每個Timer0 clock cycle皆加1(本次實驗的Timer0 clock為HXT)，一旦CNT計數器的值達到CMPDAT的值時，TIF 標誌將變為1，此時CNT的值將由定時器控制器自動清零，然後定時器計數重新開始，直至CNTEN由軟件清0。並且在INTEN位使能，則定時器中斷訊號產生並送到NVIC通知CPU。(示意圖如下)

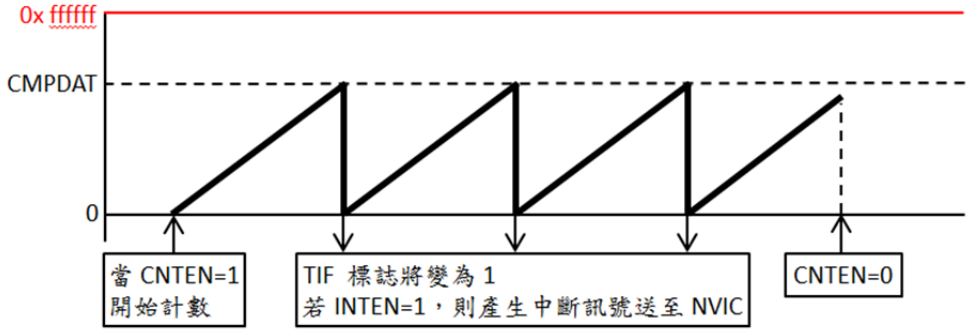


圖2-4 Periodic模式示意圖

(3). Toggle-Output模式

Toggle-output模式的計數操作大部分與Periodic模式是一樣的，除了該模式當TIF 標誌將變為1時，有相關的T0～T3腳位來輸出信號以 50% 的占空週期反復改變。(示意圖如下)

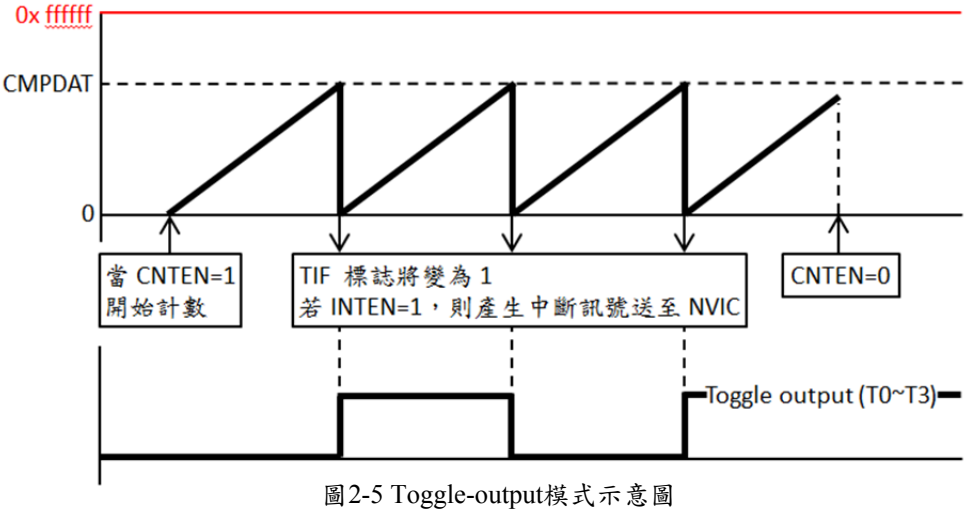


圖2-5 Toggle-output模式示意圖

(4). Continuous Counting模式

當CNTEN置1，則CNT計數器會由0開始計數，之後每個Timer0 clock cycle皆加1(本次實驗的Timer0 clock為HXT)，一旦CNT計數器的值達到CMPDAT的值時，TIF 標誌將變為1，並且在INTEN位使能，則定時器中斷訊號產生並送到NVIC通知CPU。此時CNT的值將繼續上數到$(2^{24}-1)$後由定時器控制器自動清零，然後定時器計數重新開始，直至

CNTEN由軟件清0。在該模式，用户可以立刻改變不同的CMPDAT值，而不需要停止Timer計數和重新開始Timer計數。(示意圖如下)

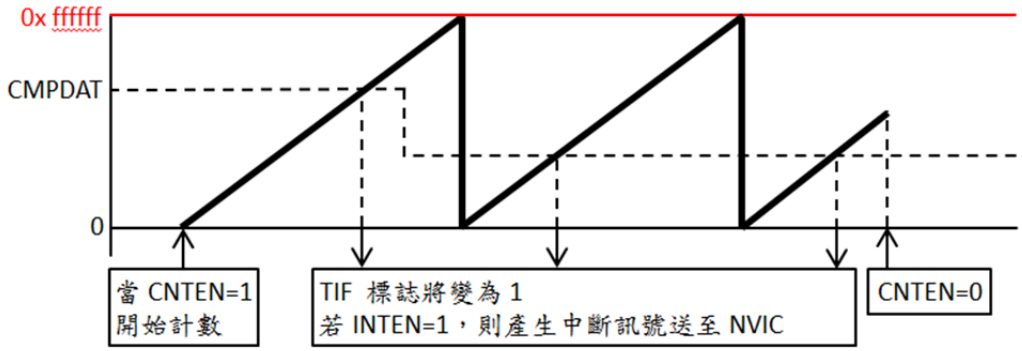


圖2-6 Continuous counting模式示意圖

當CNT與CMPDAT的值相同時，TIF旗標會變為1，此時用來判斷是否會產生中斷的INTEN(TIMER0_CTL[29])位元設為1時則會將此次中斷送至NIVC，接下來介紹何謂NIVC以及其向量位址。

3. NVIC(嵌套向量中斷控制器)：

NVIC與處理器內核介面的緊密耦合，能夠使降低中斷延遲處理以及動態分配中斷優先處理順位。其向量位址位於E000_E100h 到 E000_EF04h之間。而本次的實驗僅需要將Timer0的中斷功能開啟。

4. 中斷的語法：

```
void TMR0_IRQHandler(void)
{
        TIMER0_INTSTS = TIMER0_INTSTS | (0x3);      // clear Timer0 interrupt flag
}
```

參、學習如何透過配置相關的暫存器來產生定時中斷

1. Clock Controller：

Clock Controller中控制Timer0開啟的向量位址下表所示。

CLK_APBCLK0 : 0x4000_0208

Reserved			EADCCKEN	USBDCKEN	OTGCKEN	Reserved	CAN0CKEN
31	30	29	28	27	26	25	24
Reserved				UART3CKEN	UART2CKEN	UART1CKEN	UART0CKEN
23	22	21	20	19	18	17	16
Reserved	SPI2CKEN	SPI1CKEN	SPI0CKEN	Reserved		I2C1CKEN	I2C0CKEN
15	14	13	12	11	10	9	8
ACMP01CKEN	CLKOCKEN	TMR3CKEN	TMR2CKEN	TMR1CKEN	TMR0CKEN	RTCCKEN	WDTCKEN
7	6	5	4	3	2	1	0

位元	功能描述	
[2]	**TMR0CKEN**	**Timer0** 時鐘使能位元 0 = 禁用 Timer0 時鐘 1 = 使能 Timer0 時鐘 ✓

表2-1 CLK_APBCLK0之TMR0CKEN向量位址及功能表

Clock Controller中設定Timer0的時鐘來源的向量位址下表所示。

CLK_CLKSEL1 : 0x4000_0214

WWDTSEL		CLKOSEL		Reserved		UARTSEL	
31	30	29	28	27	26	25	24
Reserved	TMR3SEL			**Reserved**	TMR2SEL		
23	22	21	20	19	18	17	16
Reserved	TMR1SEL			**Reserved**	TMR0SEL		
15	14	13	12	11	10	9	8
Reserved						WDTSEL	
7	6	5	4	3	2	1	0

位元	功能描述	
[10:8]	**TMR0SEL**	**TIMER0** 時鐘源選擇 000 = 時鐘源為外部 4~24 MHz高速晶振時鐘(HXT) ✓ 001 = 時鐘源為外部 32.768 kHz低速晶振時鐘(LXT) 010 = 時鐘源為 PCLK0 011 = 時鐘源為外部時鐘 T0 管腳. 101 = 時鐘源為10 kHz 內部低速振盪器時鐘(LIRC) 111 = 時鐘源為內部 22.1184 MHz 高速振盪器時鐘(HIRC) 其它 = 保留

表2-2 CLK_CLKSEL1之TMR0SEL向量位址及功能表

2. Timer Controller：

本次實驗需要設定Timer0計數模式、將Timer0的中斷功能開啟和最後將Timer0 CNTEN置1以便開始計數，其向量位址如下表所示。

TIMER0_CTL : 0x4005_0000

ICEDEBUG	CNTEN	INTEN	OPMODE[1:0]		RSTCNT	ACTSTS	EXTCNTEN
31	30	29	28	27	26	25	24
WKEN	TGLPINSEL	TRGEADC	TRGDAC	TRGPWM	TRGSSEL	WKTKEN	Reserved
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
PSC[7:0]							
7	6	5	4	3	2	1	0

位元	功能描述	
30	**CNTEN**	**TIMER0**計數使能位元 0 = 停止/暫停計數 1 = 開始計數 ✓
29	**INTEN**	**TIMER0**中斷使能位元 0 = 禁用計時器中斷 1 = 使能計時器中斷 ✓
[28:27]	**OPMODE**	**TIMER0**計數模式選擇 00 = 計時器工作在單觸發模式 (one-shot) 01 = 計時器工作在週期模式 (Periodic) ✓ 10 = 計時器工作在Toggle-output 模式 11 = 計時器工作在連續計數模式(Continuous Counting）

表2-4 Timer0 Controller之TIMER0_CTL向量位址及功能表

Timer0的CMPDAT的向量位址下表所示。

TIMER0_CMP : 0x4005_0004							
Reserved							
31	30	29	28	27	26	25	24
CMPDAT [23:16]							
23	22	21	20	19	18	17	16
CMPDAT [15:8]							
15	14	13	12	11	10	9	8
CMPDAT [7:0]							
7	6	5	4	3	2	1	0

位元	功能描述	
[23:0]	**CMPDAT**	**TIMER0**比較值 CMPDAT是24位元比較暫存器。當內部 24位元上數計數器的值等於CMPDAT的值時，TIF為1。 ➢ 不能向CMPDAT裡寫 0x0 或 0x1，否則內核將運行到未知狀態。 ➢ 當Timer工作在 continuous counting 模式，即使軟體寫一個新的值到CMPDAT，24位元上數計數計時器將保持繼續計數。如果Timer工作在其他模式，如果軟體寫一個新的值到CMPDAT，Timer將使用新比較值並退出當前計數，開始重新計數。

表2-5 Timer0 Controller之TIMER0_CMP向量位址及功能表

3. NVIC(嵌套向量中斷控制器)：

開啟Timer0的中斷功能的向量位置如下表所示。

NVIC_ISER2 : 0xE000_E104							
IRQ63	**IRQ62**	**IRQ61**	**IRQ60**	**IRQ59**	**IRQ58**	**IRQ57**	**IRQ56**
31	30	29	28	27	26	25	24
IRQ55	**IRQ54**	**IRQ53**	**IRQ52**	**IRQ51**	**IRQ50**	**IRQ49**	**IRQ48**
23	22	21	20	19	18	17	16
IRQ47	**IRQ46**	**IRQ45**	**IRQ44**	**IRQ43**	**IRQ42**	**IRQ41**	**IRQ40**
15	14	13	12	11	10	9	8
IRQ39	**IRQ38**	**IRQ37**	**IRQ36**	**IRQ35**	**IRQ34**	**IRQ33**	**IRQ32**
7	6	5	4	3	2	1	0

位元	功能描述	
0	**IRQ32**	**TIMER0**中斷使能 0 = 無效 1 = 使能 Timer0 中斷 ✓

表2-7 NVIC之ISER2向量位址及功能表

4. 中斷的語法：

控制中斷旗標的向量位置如下表所示。

TIMER0_INTSTS : 0x4005_0008							
Reserved							
31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved						**TWKF**	**TIF**
7	6	5	4	3	2	1	0

位元	功能描述	
1	**TWKF**	**TIMER0**喚醒旗幟 0 = 計時器不會引起CPU 喚醒 ✓ 1 = 如果計時器中斷信號產生， CPU 從空閒或掉電模式喚醒 注意: 該位元必須通過軟體寫1清零
0	**TIF**	**TIMER0**中斷使能位元 當內部 24-bit上數計數Timer0 CNT的值與Timer0比較值CMPDAT相同時，該位元會顯示其中斷狀態。.

		0 = 無影響 ✓ 1 = CNT與CMPDAT的值相同 注意：該位元寫1清零。

表2-8 Timer0 Controller之TIMER0_INTSTS向量位址及功能表

5. 七段顯示器：

首先介紹本次實驗的顯示介面──七段顯示器，其電路圖如下所示。

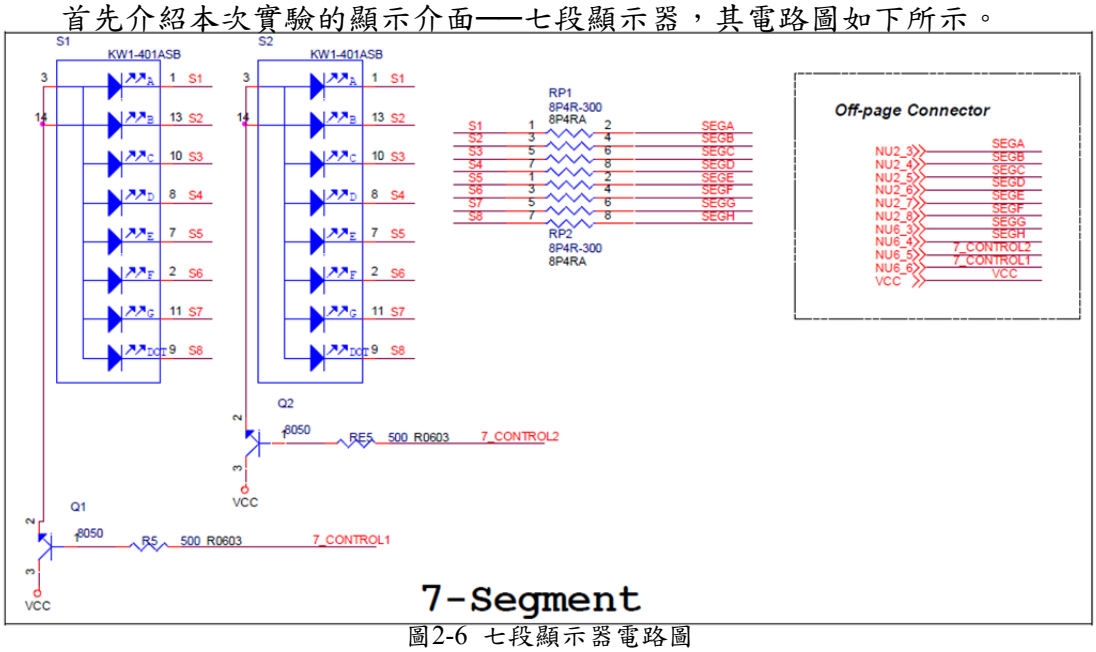


圖2-6 七段顯示器電路圖

此七段顯示器為共陰極七段顯示器，因此若要使其發亮則要將控制的腳位設為0。兩個七段顯示器的控制則是利用BJT開關實現，因此若要使Seven segment 2發亮則要將SEG_CONTROL2設為1。七段顯示器控制腳位如下表所示。

7-Segment	SEG_CONTROL1	SEG_CONTROL2	
SEG_A	S1.A	S2.A	PB.11
SEG_B	S1.B	S2.B	PB.12
SEG_C	S1.C	S2.C	PB.13
SEG_D	S1.D	S2.D	PB.14
SEG_E	S1.E	S2.E	PB.15
SEG_F	S1.F	S2.F	PB.5
SEG_G	S1.G	S2.G	PD.11
SEG_H	S1.H	S2.H	PF.2
	PD.8	PC.8	GPIO

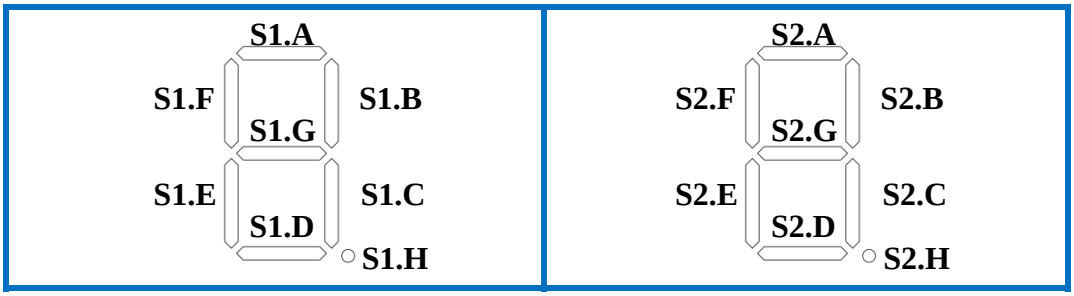


表2-9 七段顯示器腳位表

本次實驗需要將GPIO腳位設為推挽輸出模式，而GPIO向量位址位於4000_4000h 到 4000_4180h之間。其功能如下表所示。

PX_MODE : GPIO_BA_PX+0x00							
MODE15		MODE14		MODE13		MODE12	
31	30	29	28	27	26	25	24
MODE11		MODE10		MODE9		MODE8	
23	22	21	20	19	18	17	16
MODE7		MODE6		MODE5		MODE4	
15	14	13	12	11	10	9	8
MODE3		MODE2		MODE1		MODE0	
7	6	5	4	3	2	1	0

位元	功能描述	
[2n+1:2n] n=0,1..15	**MODEn**	**PA~PF I/O Pin[N]** 模式控制 00 = PX.n腳位為輸入模式(Input mode) 01 = PX.n腳位為推挽輸出模式(Push-pull mode) ✓ 10 = PX.n腳位為開漏模式(Open-drain mode) 11 = PX.n腳位為准雙向模式(Quasi-bidirectional mode)

表2-11 GPIO之PX_MODE向量位址及功能表

【範例程式說明】

此實驗提供之範例程式使處理器的七段顯示器顯示秒數。(每1ms中斷一次)

【範例程式】

範例程式:

```
#include "M451Series.h"

uint32_t volatile TimerCounter = 0;
unsigned char _7_seg_buf[2];
unsigned char seven_segment_pattern[10] = { 0xC0,      //7-Segment shows 0
                                            0xF9,      //7-Segment shows 1
                                            0xA4,      //7-Segment shows 2
                                            0xB0,      //7-Segment shows 3
                                            0x99,      //7-Segment shows 4
                                            0x92,      //7-Segment shows 5
                                            0x82,      //7-Segment shows 6
                                            0xD8,      //7-Segment shows 7
                                            0x80,      //7-Segment shows 8
                                            0x98 };    //7-Segment shows 9
```

```c
/*---------------------------------------------------------------------------------------------------------*/
/*  TMR0 IRQ handler                                                                                       */
/*---------------------------------------------------------------------------------------------------------*/
void TMR0_IRQHandler(void)
{
    TimerCounter++;         // plus 1 each Timer0 interrupt
    TIMER0_INTSTS |= (0x3); // clear Timer0 interrupt flag
}
/*---------------------------------------------------------------------------------------------------------*/
/*  Open Interrupt                                                                                         */
/*---------------------------------------------------------------------------------------------------------*/
void Open_Interrupt(void)
{
    CLK_APBCLK0 |= (1<<2);          //enable Timer0
    CLK_CLKSEL1 &= ~(7<<8);         //set HXT as Timer0 clock source
    TIMER0_CTL = (1<<27)|(1<<29);  //set Timer0 OPMODE as periodic mode and enable Timer0 interupt
    TIMER0_CMP = 0x2ef0;            //set Timer0 CMPDAT
    NVIC_ISER2 |= 0x1;              //enable interupt
}
/*---------------------------------------------------------------------------------------------------------*/
/*  Initial Seven_Segment                                                                                  */
/*---------------------------------------------------------------------------------------------------------*/
void Seven_Segment_Initial(void)
{
    PB_MODE = (0x1 << 22)|(0x1 << 24)|(0x1 << 26)|(0x1 << 28)|(0x1 << 30)|(0x1 << 10);
    PD_MODE = (0x1 << 22)|(0x1 << 16);
    PF_MODE = (0x1 << 4);
    PC_MODE = (0x1 << 16);
    //set Seven_Segment as Push-pull Output mode
    _7_seg_buf[0]=0xFF;
    _7_seg_buf[1]=0xFF;
}
/*---------------------------------------------------------------------------------------------------------*/
/*  7_SEG_SCAN                                                                                             */
/*---------------------------------------------------------------------------------------------------------*/
void _7_SEG_SCAN(void)
{
    static uint32_t OldCounte=0, _7_seg_scan_state=0;
    uint32_t no;

    if(TimerCounter - OldCounte >=5)
    {
        _7_seg_scan_state = (_7_seg_scan_state==0)? 1:0;
        PD8 = (_7_seg_scan_state==1)? 1:0;   //turning on/off 7-Segment control1
        PC8 = (_7_seg_scan_state==0)? 1:0;   //turning on/off 7-Segment control2
        no =_7_seg_buf[_7_seg_scan_state];
        PB11=((no & 0x01)>>0);
        PB12=((no & 0x02)>>1);
        PB13=((no & 0x04)>>2);
        PB14=((no & 0x08)>>3);
        PB15=((no & 0x10)>>4);
        PB5 =((no & 0x20)>>5);
        PD11=((no & 0x40)>>6);
        PF2 =((no & 0x80)>>7);
        OldCounte = TimerCounter;
    }   //scan _7_seg_buf every 5ms
}
/*---------------------------------------------------------------------------------------------------------*/
/*  7_SEG_PUT                                                                                              */
/*---------------------------------------------------------------------------------------------------------*/
void _7_SEG_PUT(int i, uint32_t no)
{
  _7_seg_buf[i]= seven_segment_pattern[no];
}
/*---------------------------------------------------------------------------------------------------------*/
/*  MAIN function                                                                                          */
/*---------------------------------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------*/
/*  MAIN function                                                                                  */
/*-------------------------------------------------------------------------------------------------*/
int main(void)
{

    uint32_t TimerOldCounter = 0;
    uint32_t Time_sec = 0;

    Open_Interrupt();         //Open interrupt
    Seven_Segment_Initial();  //Initial Seven_Segment
    TIMER0_CTL |= (1<<30);    //Start Timer0
    while(1)
    {
        if(TimerCounter-TimerOldCounter>=1000)
        {
            Time_sec = (Time_sec >= 59)? 0 : (Time_sec+1);
            _7_SEG_PUT(0,Time_sec % 10);
            _7_SEG_PUT(1,Time_sec / 10);
            TimerOldCounter = TimerCounter;
        }
        _7_SEG_SCAN();
    }
}
```

【實作內容】

1. **課堂上實作內容：**

   利用七段顯示器顯示時間，按下不同按鍵時七段顯示器會分別顯示秒數、分數或是時數。

2. **額外bonus：**

   做出任何有創意或額外的功能。

3. **實驗報告：**

   報告內容包含程式碼(加註解)、程式碼流程、心得。