計時器/計數器
Timer/Counter

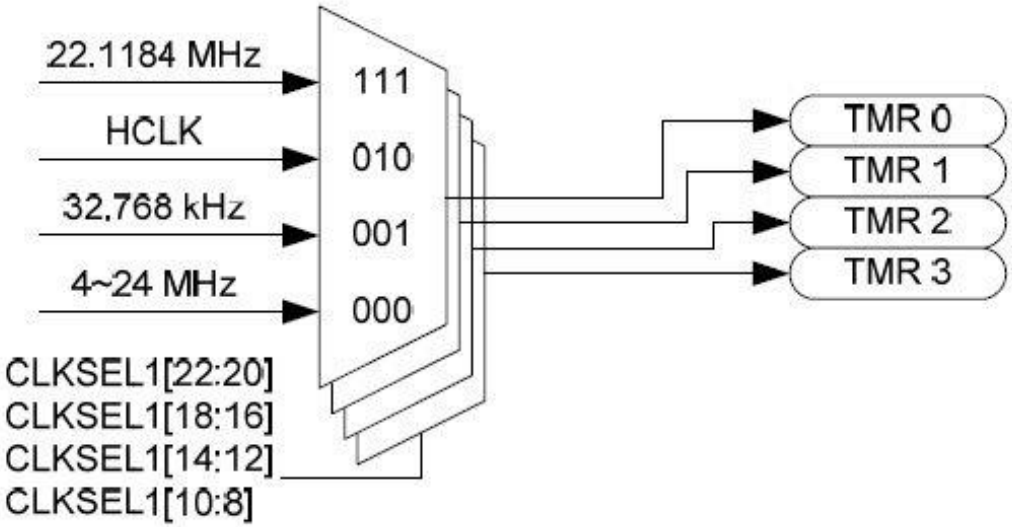
- 2013/4/3

課程大綱 (Lesson Outline)

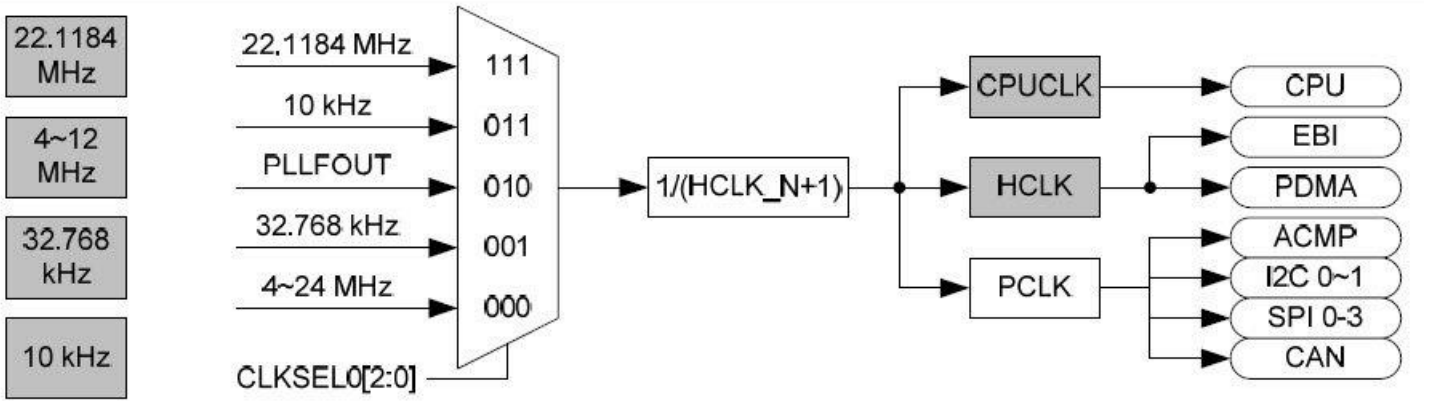
- **Cortex-M0 MCU 計數器/看門狗計數/計時時序 設計原理**
  - 計數器(Timer)
  - 看門狗(WatchDog)
  - 時序(RTC)
- **實習範例 : Smpl_Timer_WDT_RTC**
- **實習範例 : Timer計數器 – 單次模式、週期模式、循環模式**
  - 程式修改為使用三個Timer 把Timer0,1,2 以不同速度計時及顯示至LCD
- **範例練習: WDT看門狗計數器**
  - 利用範例修改出只使用WDT 看門狗之程式
- **範例練習: RTC計時時序**
  - 利用範例修改出只使用RTC時序計時之應用

時序控制器和計時器

- 參閱
- 技術手冊 5.3 時序控制器
- 技術手冊 5.10 計時器
- 計時器的時序源有22MHz,外部32K,外部4-24M,HCLK
- HCLK的時序源有22MHz,10KHz,外部32K,外部4-24M, PLLFOUT

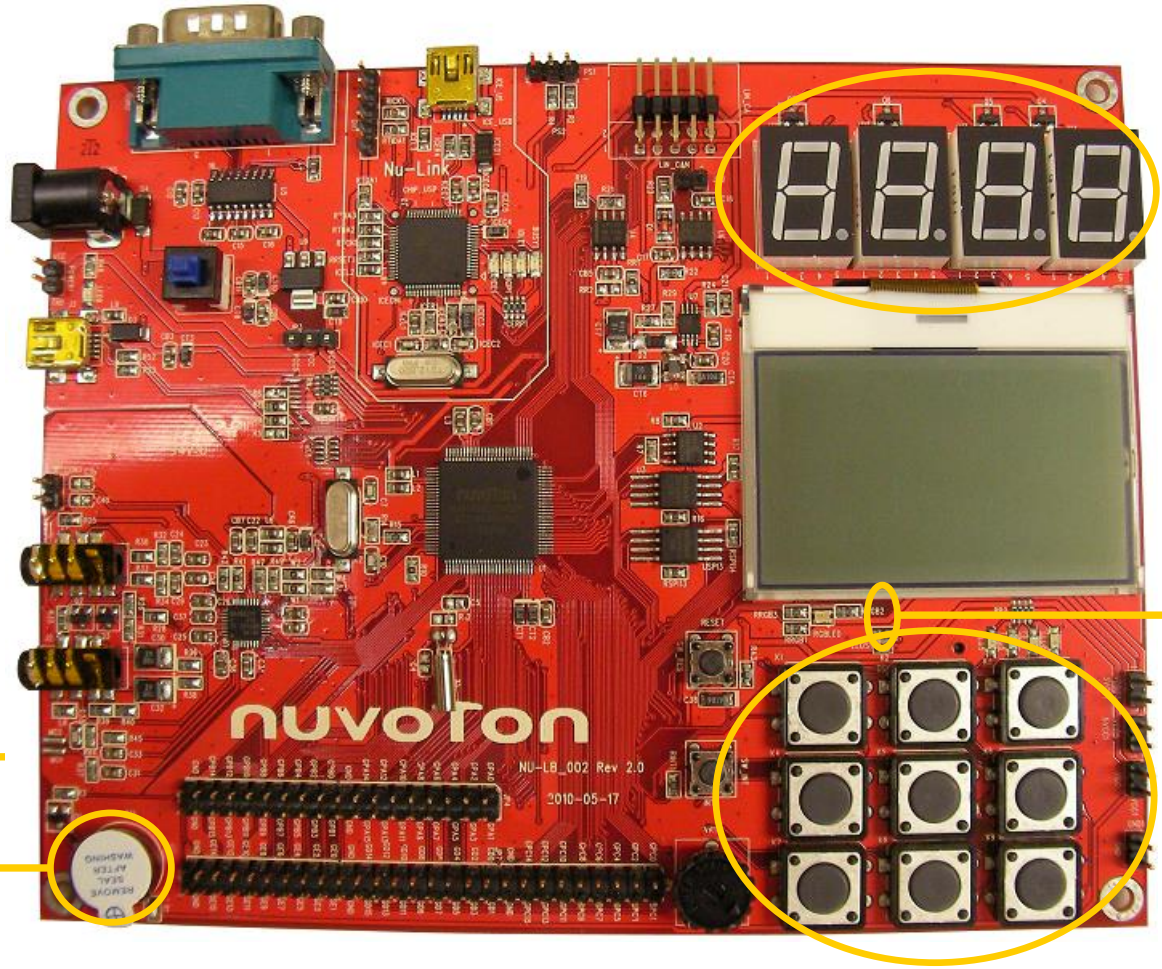


時序控制器和計時器



- HCLK的時序源有22MHz,10KHz,外部32K,外部4-24M, PLLFOUT
- PLLFOUT可使用PLLCON設定各種頻率(max 50MHz)。

NU-LB-NUC140 開發板



7 segment LEDs

RGB LED
GPA12,13,14

Keypad

SW Int
GPB15

Buzzer
GPB11

Unlockreg() & lockreg()

Unlockreg()

Unlock the locked registers before access

暫存器鎖定鍵地址暫存器(RegLockAddr) :

有些系統控制暫存器需要被保護起來，以防止誤操作而影響芯片運行，這些暫存器在reset後是鎖定的。用戶可以連續依次寫入“59h”, “16h” “88h”到0x5000_0100解鎖

LOCKREG();

//向 “0x5000_0100”寫入任何值，就可以重新上鎖保護暫存器

PWRCON: Power Down Control Register

- 功能：System Power Down Control Register (1: enable)
- 設定時序來源，10K,22M,32K預設為開啟。外部4-24M必須啟動後，才能使用。

Bits	function	Descriptions
3	OSC10K_EN	Internal 10 kHz Low Speed Oscillator Enable
2	OSC22M_EN	Internal 22.1184 MHz High Speed Oscillator Enable
1	XTL32K_EN	External 32.768 kHz Low Speed Crystal Enable
0	XTL12M_EN	External 4~24 MHz High Speed Crystal Enable

Function: DrvSYS_SetOscCtrl

- 功能：enable/disable internal oscillator or external crystal.
- 函數：int32_t DrvSYS_SetOscCtrl (E_SYS_CHIP_CLKSRC eClkSrc, int32_t i32Enable)
- 參數：eClkSrc: 暫存器PWRCON
  E_SYS_XTL12M:[0]外部4-24M振盪器
  E_SYS_XTL32K:[1]外部32.768K振盪器
  E_SYS_OSC22M:[2]內部22.1184M振盪器
  E_SYS_OSC10K:[3]內部10K振盪器
- 參數： i32Enable:1: Enable / 0: Disable.
- 範例：DrvSYS_SetOscCtrl(E_SYS_XTL12M, 1);
- 與下列指令的作用相同
- SYSCLK->PWRCON.XTL12M_EN = 1;

CLKSTATUS: Clock status Register

- 功能：Clock status monitor Register (1:stable)

Bits	function	Descriptions
4	OSC22M_STB	Internal 22.1184 MHz High Speed Oscillator Clock Source Stable Flag
3	OSC10K_STB	Internal 10 kHz Low Speed Oscillator Clock Source Stable Flag
2	PLL_STB	Internal PLL Clock Source Stable Flag
1	XTL32K_STB	External 32.768 kHz Low Speed Crystal Clock Source Stable Flag
0	XTL12M_STB	External 4~24 MHz High Speed Crystal Clock Source Stable Flag

Function: DrvSYS_GetChipClockSourceStatus

- 功能：monitor if the chip clock source stable or not.
- 函數：int32_t DrvSYS_GetChipClockSourceStatus (E_SYS_CHIP_CLKSRC eClkSrc)
- 參數：eClkSrc: 暫存器CLKSTATUS
  - E_SYS_XTL12M:[0]外部4-24M振盪器
  - E_SYS_XTL32K :[1]外部32.768K振盪器
  - E_SYS_OSC22M :[4]內部22.1184M振盪器
  - E_SYS_OSC10K :[3]內部10K振盪器
  - E_SYS_PLL:[2]內部PLL時序源
- 範例： DrvSYS_GetChipClockSourceStatus(E_SYS_XTL12M)
- 與下列指令的作用相同
- SYSCLK->CLKSTATUS.XTL12M_STB

CLKSEL0: Clock Source Select Control Register 0

Bits	function	Descriptions
5:3	STCLK_S	**Cortex_M0 SysTick clock source select** 000 = external 4~24 MHz high speed crystal clock 001 = external 32.768 kHz low speed crystal clock 010 = external 4~24 MHz high speed crystal clock/2 011 = Clock source from HCLK/2 111 = internal 22.1184 MHz high speed oscillator clock/2
2:0	HCLK_S	**HCLK clock source select** 000 = external 4~24 MHz high speed crystal clock 001 = external 32.768 kHz low speed crystal clock 010 = Clock source from PLL clock 011 = internal 10 kHz low speed oscillator clock 111 = internal 22.1184 MHz high speed oscillator clock

Function: DrvSYS_SelectHCLKSources

- 功能：select HCLK clock source.
- 函數：int32_t DrvSYS_SelectHCLKSource(uint8_t u8ClkSrcSel)
- 參數：u8ClkSrcSel: 暫存器CLKSEL0

  0: External 12M clock

  1: External 32K clock

  2: PLL clock

  3: Internal 10K clock

  7: Internal 22M clock
- 範例： DrvSYS_SelectHCLKSource(0);
- 與下列指令的作用相同
- SYSCLK->CLKSEL0.HCLK_S = 0;
- SystemCoreClockUpdate();

CLKSEL1: Clock Source Select Control Register 1

Bits	function	Descriptions
[10:8]	**TMR0_S**	**TIMER0** 時序源選擇 000 = 時序源為外部 4~24 MHz 晶振時序 001 = 時序源為外部 32.768 kHz 晶振時序 010 = 時序源為 HCLK 111 = 時序源為內部 22.1184 MHz 振盪器時序
[7:4]	**Reserved**	保留
[3:2]	**ADC_S**	**ADC** 時序源選擇 00 = 時序源為外部 4~24 MHz 晶振時序 01 = 時序源來自 PLL 時序 10 = 時序源為 HCLK 11 = 時序源為內部 22.1184 MHz 振盪器時序

APB 設備時序使能控制寄存器 (APBCLK)

- 功能：該寄存器各位用於使能/禁用外設控制器時序。

Bits		描述
[4]	TMR2_EN	Timer2 時序使能控制 1 = 使能 Timer2 時序 0 = 禁用 Timer2 時序
[3]	TMR1_EN	Timer1 時序使能控制 1 = 使能 Timer1 時序 0 = 禁用 Timer1 時序
[2]	TMR0_EN	Timer0 時序使能控制 1 = 使能 Timer0 時序 0 = 禁用 Timer0 時序

定時器控制寄存器 (TCSR)

Bits	symbol	描述
[31]	**DBGACK_TMR**	**ICE debug mode acknowledge Disable**（寫保護位） 0 = ICE除錯模式回應影響定時器計數。 當ICE除錯模式回應時，定時器計數將被鎖定。 1 = ICE除錯模式回應禁用。 無論ICE除錯模式回應與否，定時器計數將持續下去。
[30]	**CEN**	**Timer Enable Bit** 1 = 開始計數 0 = 停止/暫停計數 注1：在停止狀態，設置 CEN = 1 將致能 24-位向上計數器從上次停止的計數值繼續計數。 注2：在 one-shot 模式下 (MODE [28:27] =00)，當相應的定時中斷產生時 (IE [29] =1)，該位元由硬體自動清零。

定時器控制寄存器 (TCSR)

Bits	symbol	描述
[29]	**IE**	**Interrupt Enable Bit** 1 = 致能定時器中斷 0 = 禁用定時器中斷 當定時器中斷致能，當計數值與TCMPR寄存器內數值相同是，觸發中斷。
[28:27]	**MODE**	00:當定時器定義為單觸發模式 (one-shot) 時，定時器溢出僅觸發中斷一次（如果 IE 致能），進入中斷後 CEN 由硬件自動清除為 0。
		01:當定時器定義為週期模式 (period) 時，定時器每次溢出都觸發相應中斷（如果 IE 致能）。
		10:當定時器工作在 toggle 模式，中斷信號週期性產生（如果 IE 致能）。且相應的信號 (tout) 以占空比為 50% 週期改變。
		11:定時器工作在連續計數 (continuous counting) 模式。相應的中斷在 TDR = TCMPR 時產生（如果 IE 致能）。 然而，24位的向上定時器繼續計數而不會復位。

定時器控制寄存器 (TCSR)

Bits	symbol	描述
[26]	**CRST**	**Timer Reset Bit** 0 = 該位寫 0 無效。 1 = 重置定時器的 8-位預分頻計數器，內部 24-位向上計數器和 CEN 位。
[25]	**CACT**	**Timer Active Status Bit**（只讀） 該位表示當前定時器計數器的狀態。 0 = 定時器未激活 1 = 定時器激活
[24]	**CTB**	**Counter Mode Enable Bit** 該位是計數模式致能位元。當定時器用作事件計數器時，該位需要設置成 1 且定時器作為事件計數器外部觸發引腳。計數器根據 TX_PHASE 定義的相位在外部管腳的上升沿/下降沿檢測相位。 1 = 致能計數器模式 0 = 禁用計數器模式

定時器控制寄存器 (TCSR)

Bits	symbol	描述
[16]	TDR_EN	**Data Load Enable** TDR_EN 被設定後，當計數器計數時，TDR（Timer 數據寄存器）將被 24-位元向上計數器的值不斷更新。 1 = Timer 數據寄存器更新致能。 0 = Timer 數據寄存器禁用更新。
[7:0]	PRESCALE	**Pre-scale Counter** 時序輸入根據 PRESCALE +1 進行預分頻。如果 PRESCALE=0，不進行預分頻。

定時器比較寄存器 (TCMPR)

Bits	symbol	描述
[23:0]	**TCMP**	定時器比較值 TCMP 是24-位元比較寄存器。當內部 24-位元向上計數器的值與 TCMP 的值相等時，如果TCSR.IE[29]=1，產生中斷請求。 TCMP 的值定義定時器計數的週期時間。 週期時間 =（定時器時序輸入的週期）* (8-bit PRESCALE + 1) * (24-bit TCMP) 注1：不能向 TCMP 裡寫 0x0 或 0x1，否則內核將運行到未知狀態。 注2：當定時器工作在 continuous counting 模式時，如果軟體寫一個新的值到 TCMP，24-位元向上計數定時器將繼續計數。 如果定時器工作在其他模式，如果軟件寫一個新的值到 TCMP，定時器將使用新比較值並退出當前計數，開始重新計數。

定時器中斷狀態寄存器 (TISR)

Bits	symbol	描述
[0]	**TIF**	定時器中斷旗標 此位元顯示定時器中斷狀態。 當內部 24-位元向上計數定時器符合定時器比較值 (TCMP)，硬體設定TIF=1。 用軟體寫入1，將該位元清除為0。

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Bits	符號	描述
[31:0]	**SETPEND**	寫 1，由軟件控制掛起相應中斷。每位元代表 IRQ0 ~ IRQ31 的中斷號（向量號：從16到47）。 寫 0 無效 讀取該寄存器返回當前等待處理的中斷狀態。

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Function	Description
DrvTIMER_Init	User must to call this function before any timer operations after system boot up.
DrvTIMER_Open	Open the specified timer channel with specified operation mode.
DrvTIMER_Close	close the timer channel.
DrvTIMER_SetTimerEvent	Install the interrupt callback function of the specified timer channel.
DrvTIMER_ClearTimerEvent	Clear the timer event of the specified timer channel.
DrvTIMER_EnableInt	enable the specified timer interrupt.
DrvTIMER_DisableInt	disable the specified timer interrupt

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Function	Description
DrvTIMER_GetIntFlag	Get the interrupt flag status from the specified timer channel.
DrvTIMER_ClearIntFlag	Clear the interrupt flag of the specified timer channel.
DrvTIMER_Start	Start to count the specified timer channel
DrvTIMER_GetIntTicks	get the number of interrupt occurred after the timer interrupt function is enabled.
DrvTIMER_ResetIntTicks	clear interrupt ticks to 0.
DrvTIMER_Delay	add a delay loop by specified interrupt ticks of the timer channel.
DrvTIMER_OpenCounter	open the timer channel with the specified operation mode.

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Function	Description
DrvTIMER_StartCounter	Start counting of the specified timer channel
DrvTIMER_GetCounters	get the current counters of the specified timer channel.
DrvTIMER_OpenCapture	initial the external timer capture source and set to start catpure or reset specified timer counter.
DrvTIMER_CloseCapture	close the external timer capture source.
DrvTIMER_SelectExternalMode	select to run capture function or reset the timer counter.
DrvTIMER_SelectCaptureEdge	configure the detect edge of timer capture mode.

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Function	Description
DrvTIMER_EnableCaptureInt	enable the timer external interrupt function.
DrvTIMER_DisableCaptureInt	disable the timer external interrupt function
DrvTIMER_EnableCapture	enable the specified capture function.
DrvTIMER_DisableCapture	disable the specified capture function
DrvTIMER_GetCaptureData	get the capture value of the specified timer channel.
DrvTIMER_GetCaptureIntFlag	Get the external interrupt flag status from the specified timer channel.
DrvTIMER_ClearCaptureIntFlag	Clear the external interrupt flag of the specified timer channel

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Function	Description
DrvTIMER_EnableCaptureDebounce	Enable the debounce function of specified external capture input source
DrvTIMER_DisableCaptureDebounce	Disable the debounce function of specified external capture input source.
DrvTIMER_EnableCounterDebounce	Enable the debounce function of specified external counter input source.
DrvTIMER_DisableCounterDebounce	Disable the debounce function of specified external counter input source.
DrvTIMER_SelectCounterDetectPhase	configure the counter detect phase of specified source.
DrvTIMER_GetVersion	Get the version number of Timer/WDT driver.

IRQ0 ~ IRQ31 Set-Pending Control Register (NVIC_ISPR)

Function	Description
DrvWDT_Open	Enable WDT engine clock and set WDT time-out interval.
DrvWDT_Close	stop/disable WDT relative functions.
DrvWDT_InstallISR	install WDT interrupt service routine.
DrvWDT_Ioctl	start/stop the WDT, enable/disable WDT interrupt function, enable/disable WDT time-out wake up function, enable/disable system reset

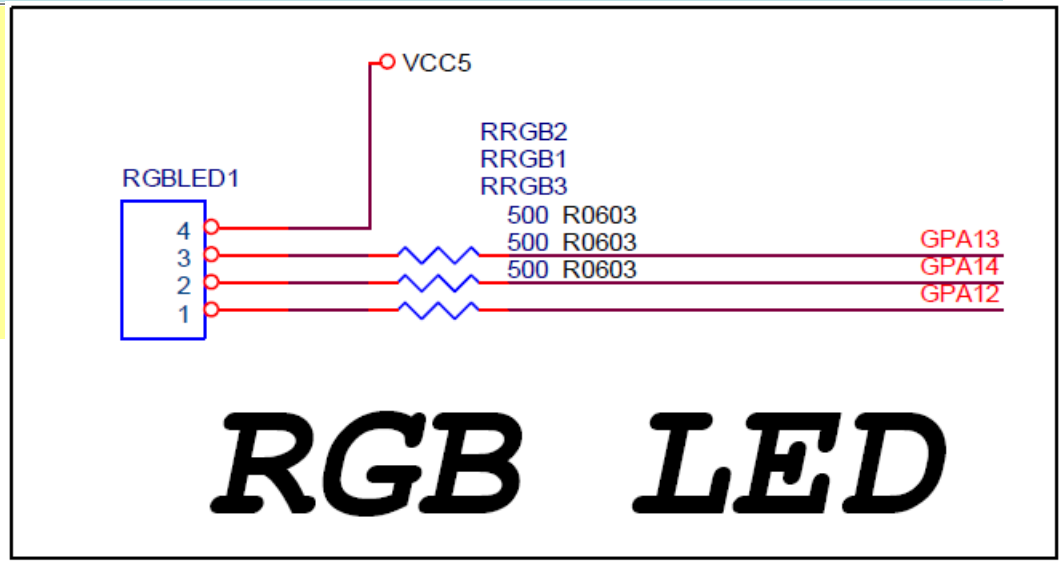
7.1Test_RGBled 學習板電路圖

- RGB LED : GPA12,13,14

GPA12 : Blue　0 = on, 1 = off

GPA13 : Green　0 = on, 1 = off

GPA14 : Red　0 = on, 1 = off



寫一程式輪流顯示藍色，綠色，紅色LED，每個LED亮一秒。

1. 使用Timer 0控制時間，每秒產生一次中斷。
2. 在Timer 0中斷時，切換顯示的LED。
3. 藍色LED :GPA[12]，綠色LED :GPA[13]，紅色LED :GPA[14]

7.1Test_RGBled (1/x)

```
int main (void)
{
        uint32_t freq;

        // Initial LED: set GPIO GPA12,13,14 to output mode
        // initial GPIO GPA pin  12 to output mode
        DrvGPIO_Open(E_GPA, 12, E_IO_OUTPUT);
        // initial GPIO GPA pin  13 to output mode
        DrvGPIO_Open(E_GPA, 13, E_IO_OUTPUT);
        // initial GPIO GPA pin  14 to output mode
        DrvGPIO_Open(E_GPA, 14, E_IO_OUTPUT);
```

7.1Test_RGBled (2/x)

```
// Initial 12M and TIMER0
        Init12M();  //initial external 12M
        freq=1;
        InitTIMER0(freq);
// Main loop
        while (1)
        {

        }
}
```

7.1Test_RGBled (3/x)

```
void InitTimer0(uint32_t freq)
{
        // Select TIMER0 clock source
        // 0=extgernal, 1=32k,2=hclk,7=22M
        DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0);

        // Enable TIMER0 engine clock
        DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);

        // User must to call this function before any timer operations.
        DrvTIMER_Init ();
```

7.1Test_RGBled (4/x)

```
        // Open the specified timer channel with specified operation mode.
        DrvTIMER_Open (E_TMR0, freq, E_PERIODIC_MODE);
        // enable the specified timer interrupt.
        DrvTIMER_EnableInt (E_TMR0);
        // Install the interrupt callback function
        //1= Number of timer interrupt occurred
        //0= A parameter of the callback function
        DrvTIMER_SetTimerEvent (E_TMR0, 1,
(TIMER_CALLBACK) TMR_Callback, 0);
        // Start to count the specified timer channel.
        DrvTIMER_Start (E_TMR0);
}
```

7.1Test_RGBled (5/x)

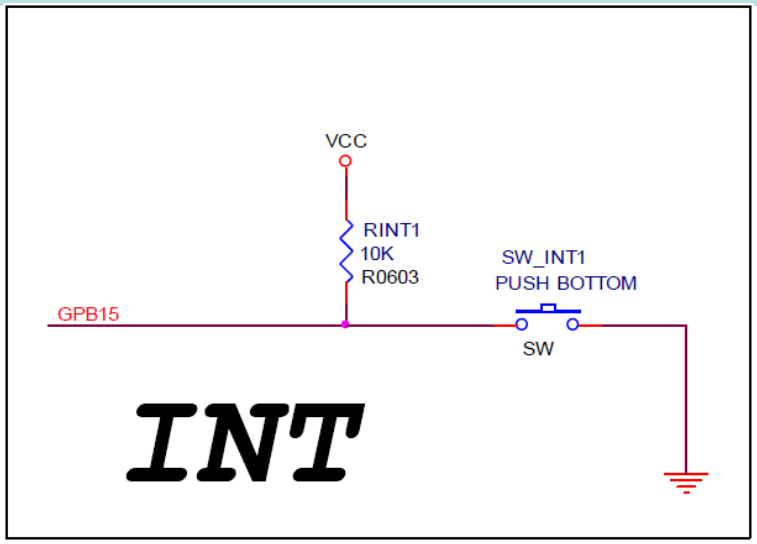
```
void TMR_Callback(void) // Timer0 interrupt subroutine
{
        //set GPIOA[LedLighted]=1, LED off
        DrvGPIO_SetBit(E_GPA,LedLighted);
        //next LED
        LedLighted++;
        if(LedLighted == 15)LedLighted=12;
        //set GPIOA[LedLighted]=0, LED on
        DrvGPIO_ClrBit(E_GPA,LedLighted);
}
```

7.1Test_RGBled (6/x)

```
// Initial external 12M
void Init12M(void)
{
        UNLOCKREG();
        //1: enable, 0: disable
        DrvSYS_SetOscCtrl(E_SYS_XTL12M,1);
        //SYSCLK->PWRCON.XTL12M_EN = 1;
        LOCKREG();
}
```

7.2Test_Interrupt_SWInt_RGBled

- RGB LED : GPA12,13,14
  - **GPA12** : Blue 0 = on, 1 = off
  - **GPA13** : Green 0 = on, 1 = off
  - **GPA14** : Red 0 = on, 1 = off
- SW Int : GPB15
  - **GPB15**: 0=pressed, 1= not pressed



寫一程式，LED閃爍亮0.25秒，暗0.25秒。當按鍵按下時，閃爍藍色LED。按鍵再按下時，閃爍綠色LED。按鍵再按下時，閃爍紅色LED。每次按鍵按下時，LED依序循環。

GPIOB[15]作為輸入，設定為外部中斷EINT1，啟用中斷，藉由中斷程式來執行。

DrvGPIO_GetBit

- 功能：Get the pin value from the specified input GPIO pin.
- 函數：int32_t DrvGPIO_GetBit(E_DRVGPIO_PORT port, int32_t i32Bit)
- 參數：Port: E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA(+0), E_GPB(+0x40), E_GPC(+0x80), E_GPD(+0xC0) and E_GPE(+0x100).
- 參數： i32Bit: Specify pin of the GPIO port. It could be 0~15.
- 範例：DrvGPIO_GetBit(E_GPB, 15)　　//get GPIOB bit15
- 　　與下列指令的作用相同
- 　　GPIOB->PIN & (1<<15) //get GPIOB bit15

DrvGPIO_EnableDebounce

- 功能：Enable the debounce function。
- 函數：int32_t DrvGPIO_EnableDebounce(E_DRVGPIO_PORT port, int32_t i32Bit)
- 參數：Port: specify GPIO port. It could be E_GPA(+0), E_GPB(+0x40), E_GPC(+0x80), E_GPD(+0xC0) and E_GPE(+0x100).
- 參數： i32Bit: Specify pin of the GPIO port. It could be 0~15.
- 範例：DrvGPIO_EnableDebounce(E_GPB, 15);
- 如同指令：
- GPIOB->DBEN |= (1<<15); //DBEN[15]=1, de-bounce enabled
- GPIO_DBNCECON->DBNCECON.ICLK_ON=1; //Interrupt clock On mode

DrvGPIO_SetDebounceTime

- 功能：Set the interrupt debounce sampling time。
- 函數：int32_t DrvGPIO_SetDebounceTime(uint32_t u32CycleSelection, E_DRVGPIO_DBCLKSRC ClockSource)
- 參數： u32CycleSelection : The number of sampling cycle selection,0-15.
- 參數： ClockSource : E_DBCLKSRC_HCLK or E_DBCLKSRC_10K.
- 範例：DrvGPIO_SetDebounceTime(4,E_DBCLKSRC_10K);
- 如同指令：
- //debounce time=16/10K=16*0.1ms=1.6ms
- GPIO_DBNCECON->DBNCECON.DBCLKSEL = 4;
- //sample once per 2^4=16 clocks
- GPIO_DBNCECON->DBNCECON.DBCLKSRC = 1; //10KHz

使用EINT1中斷的初始設定

- 1.設定中斷允許：下降邊緣,低電位或上升邊緣,高電位
- GPIOB->IEN |= (1<<15)　　//fall edge, low level
- GPIOB->IEN |= (1<<(15+16))　//rise edge, high level
- 2.設定觸發模式：
- GPIOB->IMD &= ~(1<<15);　　//=0, edge trigger
- GPIOB->IMD |= (1<<15);　　//=1, level trigger
- 3.設定EINT1優先權
- NVIC->IPR[0] = (2ul<<30);　　//11=最高, 00=最低
- 4.設定IRQ4中斷允許
- NVIC->ISER[0]=(1<<3);
- 上述步驟，可以用函數直接設定：
- DrvGPIO_EnableEINT1(E_IO_FALLING, E_MODE_EDGE, EINT1Callback); //下降邊緣，邊緣觸發，中斷處理程式 EINT1Callback

DrvGPIO_EnableEINT1()

功能： Enable the interrupt function for EINT1。
函數： void DrvGPIO_EnableEINT1(E_DRVGPIO_INT_TYPE TriggerType, E_DRVGPIO_INT_MODE Mode, GPIO_EINT1_CALLBACK pfEINT1Callback)
參數：TriggerType: interrupt trigger type(E_IO_RISING, E_IO_FALLING or E_IO_BOTH_EDGE).
參數： Mode: interrupt mode (E_MODE_EDGE or E_MODE_LEVEL).
參數： pfEINT1Callback : pfEINT1Callback (function pointer of the external INT1 callback function).
範例：DrvGPIO_EnableEINT1(E_IO_FALLING, E_MODE_EDGE, EINT1Callback);
說明：當SW INT按下時，原來程式中斷，此時GPIOB_ISRC=0x8000，IRQ3_SRC=1，MCU_IRQ=0x8，執行EINT1_IRQHandler (DRVGPIO.c)。先清除GPIOB_ISRC[15]=0，則IRQ3_SRC=0，MCU_IRQ=0x0，再呼叫函數pfEINT1Callback。

EINT1的中斷服務程式

- 在DrvGPIO.c有提供中斷服務程式
- void EINT1_IRQHandler(void)
- {
- /* EINT0 = GPB15. Clear the interrupt flag */
- GPIOB->ISRC = 1UL << 15; //清除Interrupt Source Flag
- if (_pfEINT1Callback) //如果有設定callback函數
- _pfEINT1Callback(); //呼叫callback函數
- }

- callback函數必須撰寫中斷的處理程式
- void EINT1Callback(void)
- {
- ….
- }

7.2Test_SWInt_RGBled (1/x)

```
int main (void)
{       uint32_t freq;
        // Initial LED
        // Initial GPIO GPA12(blue LED) to output mode
        DrvGPIO_Open(E_GPA, 12, E_IO_OUTPUT);

        // Initial GPIO GPA13(green LED) to output mode
        DrvGPIO_Open(E_GPA, 13, E_IO_OUTPUT);

        // Initial GPIO GPA14(red LED) to output mode
        DrvGPIO_Open(E_GPA, 14, E_IO_OUTPUT);
```

7.2Test_SWInt_RGBled (2/x)

```
// Initial SW EINT1
// Initial GPIO GPB15(SW Int) to input mode
DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);
//Enable the debounce function
DrvGPIO_EnableDebounce(E_GPB, 15);
//Set the interrupt debounce sampling time & clock source
//The target debounce time is (2^4)*(1/(10k)) s = 16*0.1 ms
DrvGPIO_SetDebounceTime(4,E_DBCLKSRC_10K);
// Configure external interrupt
  DrvGPIO_EnableEINT1(E_IO_FALLING,
E_MODE_EDGE, EINT1Callback);
```

7.2Test_SWInt_RGBled (3/x)

```
        // Initial 12M and Timer 0
        Init12M();
        freq=4;             //T=1/4=0.25s
        InitTIMER0(freq);
        // Main loop
        while (1)
        {
        }
}
```

7.2Test_SWInt_RGBled (4/x)

```
void EINT1Callback(void)
{
        // turn off lighted LED
         DrvGPIO_SetBit(E_GPA,led_n);

        // next LED
        led_n++;
        if(led_n > 14)led_n=12;

        // turn on next LED
        DrvGPIO_ClrBit(E_GPA,led_n);
}
```

7.2Test_SWInt_RGBled (5/x)

```
void TMR_Callback(void) // Timer0 interrupt subroutine
{
        //xor 0: unchange; xor 1: 0->1, 1->0
        GPIOA->DOUT ^= (1<<LedLighted);

}
```

7.2Test_SWInt_RGBled (6/x)

```
void InitTIMER0(uint32_t freq)
{
        // Select TIMER0 clock source
        // 0=extgernal, 1=32k,2=hclk,7=22M
        DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC,0);

        // Enable TIMER0 engine clock
        DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);

        // User must to call this function before any timer operations.
        DrvTIMER_Init ();
```

7.2Test_SWInt_RGBled (7/x)

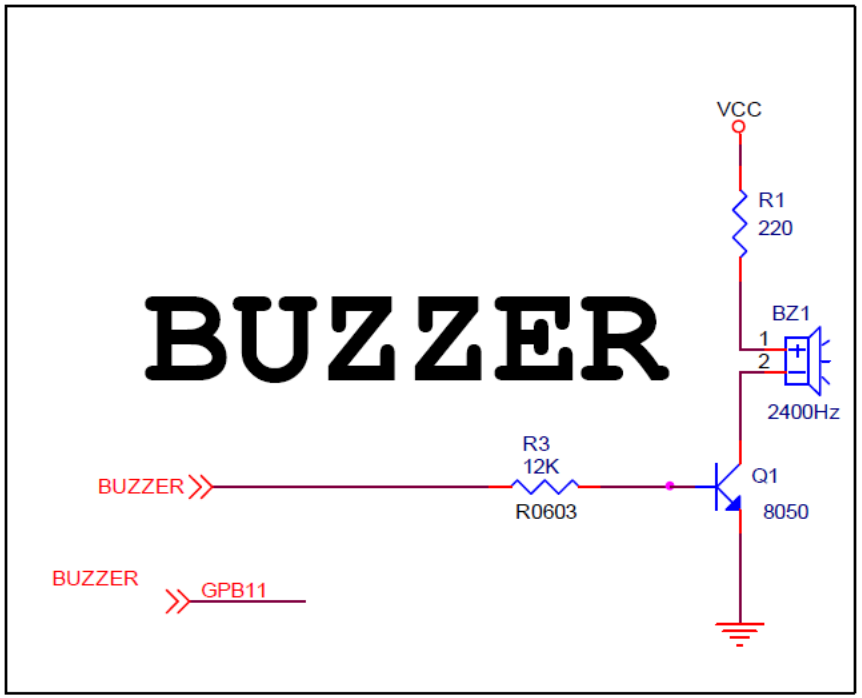
```
    // Open the specified timer channel with specified operation mode.
    DrvTIMER_Open (E_TMR0, freq, E_PERIODIC_MODE);

    // enable the specified timer interrupt.
    DrvTIMER_EnableInt (E_TMR0);

    // Install the interrupt callback function
    DrvTIMER_SetTimerEvent (E_TMR0, 1,
(TIMER_CALLBACK) TMR_Callback, 0);

    // Start to count the specified timer channel.
    DrvTIMER_Start (E_TMR0);
}
```

7.2Test_SWInt_RGBled (8/x)

```
// Initial external 12M
void Init12M(void)
{
        UNLOCKREG();
        //1: enable, 0: disable
        DrvSYS_SetOscCtrl(E_SYS_XTL12M,1);
        //SYSCLK->PWRCON.XTL12M_EN = 1;
        LOCKREG();
}
```

7.3Test_SWInt_Buzzer–

- BUZZER : GPB11
  - **GPB11** : 0 = on, 1 = off
- R1電阻太大，聲音太小聽不到。將R1短路，可聽到聲音。
- SW Int : GPB15
  - **GPB15**: 0=pressed, 1= not pressed



寫一程式，當按鍵按下時，蜂鳴器發出間斷(0.25秒on，0.25秒off)聲音。鬆開則沒有聲音。

GPIO pin GPB11=0 turn on BUZZER

GPIO pin GPB11=1 turn off BUZZER

7.3Test_TIMER_SWInt_Buzzer (1/x)

```
int main (void)
{
        uint32_t freq;
        // Initial SW EINT1
        // initial GPIO pin GPB15 (SW int ) to input mode
        DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);
        // Initial buzzer
        // initial GPIO pin GPB11 for controlling Buzzer to output mode
        DrvGPIO_Open(E_GPB, 11, E_IO_OUTPUT);
        // Initial Timer
        Init12M();
        freq=4;
        InitTimer0(freq);
```

7.3Test_TIMER_SWInt_Buzzer (2/x)

```
// Main loop
while (1)
{

}
```

7.3Test_TIMER_SWInt_Buzzer (3/x)

```
void InitTimer0(uint32_t freq)
{
        // Select TIMER0 clock source
        // 0=extgernal, 1=32k,2=hclk,7=22M
        DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0);

        // Enable TIMER0 engine clock
        DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);

        // User must to call this function before any timer operations.
        DrvTIMER_Init ();
```

7.3Test_TIMER_SWInt_Buzzer (4/x)

```
    // Open the specified timer channel with operation mode.
    DrvTIMER_Open (E_TMR0, freq, E_PERIODIC_MODE);

    // enable the specified timer interrupt.
    DrvTIMER_EnableInt (E_TMR0);

    // Install the interrupt callback function
    DrvTIMER_SetTimerEvent (E_TMR0, 1,
(TIMER_CALLBACK) TMR_Callback, 0);

    // Start to count the specified timer channel.
    DrvTIMER_Start (E_TMR0);
}
```

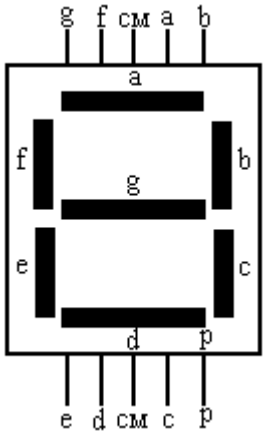
7.3Test_TIMER_SWInt_Buzzer (5/x)

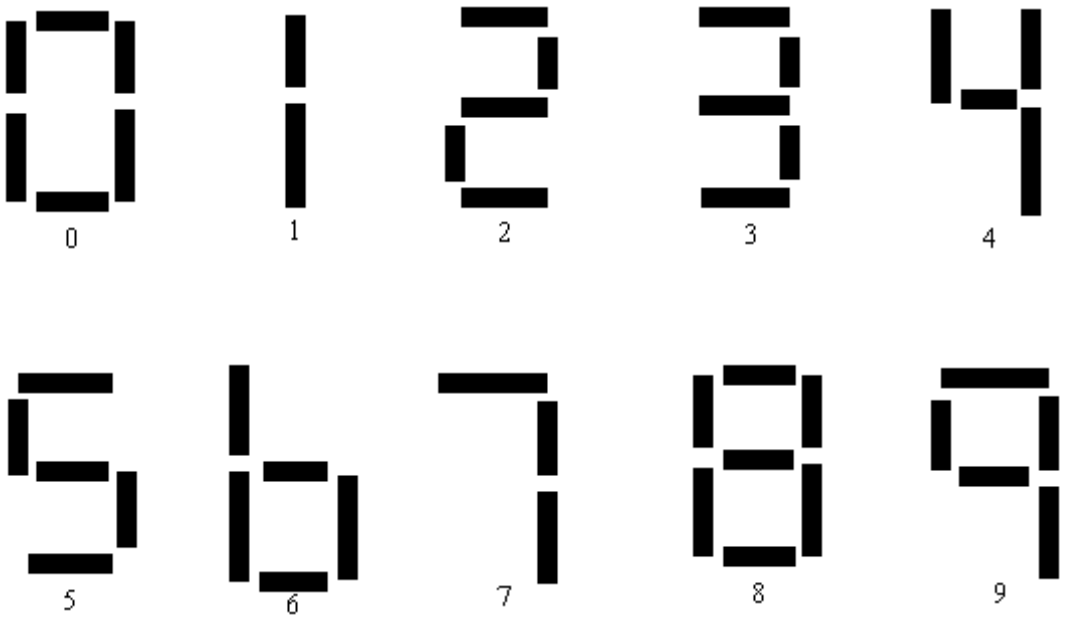
```
void TMR_Callback(void) // Timer0 interrupt subroutine
{
    //change GPIOB[11] output state
    //xor 0: unchange; xor 1: 0->1, 1->0
     if(DrvGPIO_GetBit(E_GPB,15))
        DrvGPIO_SetBit(E_GPB,15); //buzzer off
    else
        GPIOB->DOUT ^= (0x1<<11); //buzzer change state (on/off)
}
```

7.3Test_TIMER_SWInt_Buzzer (6/x)

```
// Initial external 12M
void Init12M(void)
{
        UNLOCKREG();
        //1: enable, 0: disable
        DrvSYS_SetOscCtrl(E_SYS_XTL12M,1);
        //SYSCLK->PWRCON.XTL12M_EN = 1;
        LOCKREG();
}
```

7.4Test_7seg–七段顯示器





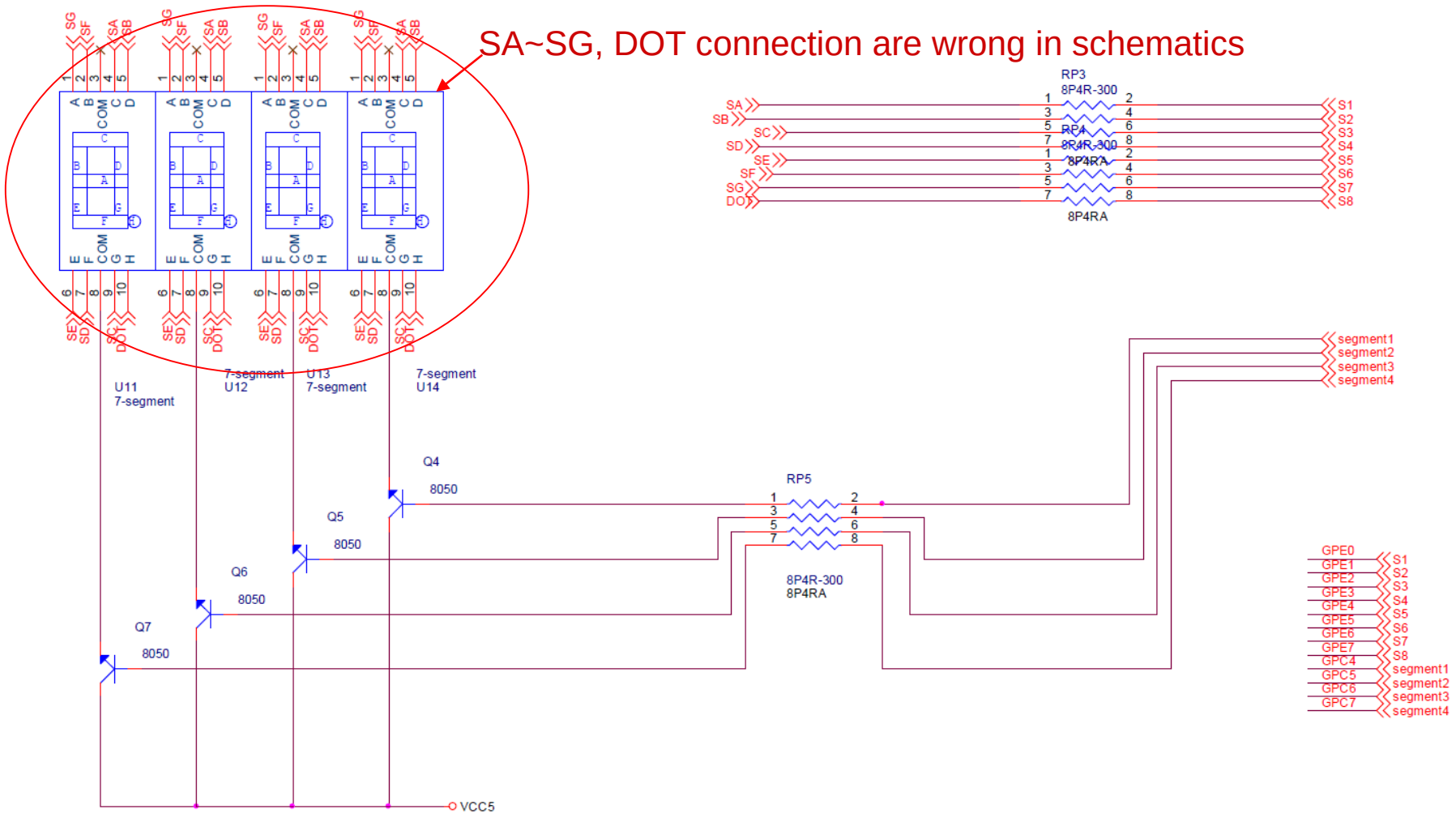
每個7段顯示器有8個LED，分別標示為a,b,c,d,e,f,g,p。

如果將a,b,c,d,e,f,g,p分別對應到GPIO的bit 0,1,2,3,4,5,6,7，則可由GPIO控制LED的明暗。

共陽極7段顯示器由低電位控制，共陰極7段顯示器由高電位控制。

顯示0000-9999，間隔0.5秒。

7.4Test_7seg– 7 段顯示器線路圖

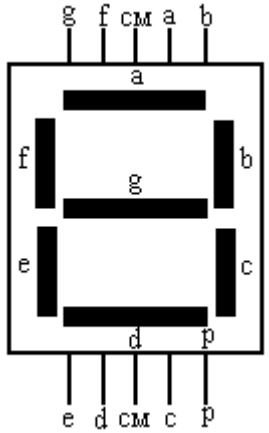


7.4Test_7seg– Control Pins used for

GPC4~7 control which 7-segment to turn on (1 = on, 0 = off)

- GPC4 : 1st 7-segment (LSB)
- GPC5 : 2nd 7-segment
- GPC6 : 3th 7-segment
- GPC7 : 4th 7-segment (MSB)

GPE0~7 control each segment to turn on (0 = on, 1 = off)



	g	**e**	**d**	**b**	**a**	**f**	**dp**	**c**	
	7	6	5	4	3	2	1	0	
0	1	0	0	0	0	0	1	0	0x82
1	1	1	1	0	1	1	1	0	0xEE
2	0	0	0	0	0	1	1	1	0x07
3	0	1	0	0	0	1	1	0	0x46
4	0	1	1	0	1	0	1	0	0x6A
5	0	1	0	1	0	0	1	0	0x52

7-Segment LED Driver

NUC100SeriesBSP/NuvotonPlatform_Keil/Src/NUC1xx-LB002/Seven_Segment.c

- #define SEG_N0 0x82 // define segment led on/off for digit 0
- #define SEG_N1 0xEE // define segment led on/off for digit1
- #define SEG_N2 0x07
- #define SEG_N3 0x46
- #define SEG_N4 0x6A
- #define SEG_N5 0x52
- #define SEG_N6 0x12
- #define SEG_N7 0xE6
- #define SEG_N8 0x02
- #define SEG_N9 0x62
- // array of segment led on/off value for digit 0~9
- unsigned char SEG_BUF[10]={SEG_N0, SEG_N1, SEG_N2, SEG_N3, SEG_N4, SEG_N5, SEG_N6, SEG_N7, SEG_N8, SEG_N9};

7-Segment LED Driver - show_seven_segment

- 功能：將number顯示在第no個7段顯示器
- 函數： void show_seven_segment(unsigned char no, unsigned char number)
- 參數： no:第no個7段顯示器，(0-3)
- 參數：number: 顯示的數字，(0-9)
- 範例：show_seven_segment(3,digit)
- 與下列指令的作用相同
- GPIOE->DOUT &= ~(0xFF); //clear 7 seg., GPIOE bit0-7
- GPIOE->DOUT |= SEG_BUF[digit[3]];//show 7 seg.,
- GPIOC->DOUT |= (1<<(3+4)); //display 4th 7 seg.,GPIOC bit 7=1

7-Segment LED Driver - close_seven_segment

- 功能：關閉7段顯示器。
- 函數：void close_seven_segment(void)
- 參數：
- 範例： close_seven_segment();
- 與下列指令的作用相同
- GPIOC->DOUT &= ~(0xF<<4);

7.4Test_7seg– main() (1/x)

```
int32_t main (void)
{
    seven_segment_open();          // Initial 7-seg.
    Init12M();                     // initial external 12M
    tm0freq=1000;                  //T=1/1000=1ms
    InitTimer0(tm0freq);           // Initial TIMER 0
// Display 0000-9999
    show_period=500;    //LED light time = 500ms
    show_number=show_period*tm0freq/1000;   // 500ms/1ms=500
    show_number_ith=show_number;      // the ith
    light_7seg=3;                     // the ith lighted LED
    count_number=0;                   // count 0000-9999
    while(1);
}
```

7.4Test_7seg (2/x)

```
void seven_segment_open(void)
{
    //Initial GPIOE [7:0] to output mode for 7-seg.
    DrvGPIO_Open(E_GPE, 0, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 1, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 2, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 3, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 4, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 5, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 6, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 7, E_IO_OUTPUT);
```

7.4Test_7seg (3/x)

```
    //Initial GPIOC [7:4] to output mode for nth 7-seg.
    DrvGPIO_Open(E_GPC, 4, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPC, 5, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPC, 6, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPC, 7, E_IO_OUTPUT);
}
```

7.4Test_7seg– seven_segment_open() (4/x)

```
void TMR_Callback(void) // Timer0 interrupt subroutine
{
    uint32_t value;
    // count_number=0000-9999
    if(!show_number_ith--)
    {
        show_number_ith=show_number; //reset show_number

        if(count_number++ == 9999)count_number=0;
        // seperate number to single digit
        seg_data(count_number);
    }

    // change lighted LED
    seg_display();
}
```

7.4Test_7seg (5/x)

```
// seperate number to single digit
void seg_data(int16_t value)
{
        digit_7seg[3] = value / 1000;
        value = value - digit_7seg[3] * 1000;
        digit_7seg[2] = value / 100;
        value = value - digit_7seg[2] * 100;
        digit_7seg[1] = value / 10;
        value = value - digit_7seg[1] * 10;
        digit_7seg[0] = value; //4 states
}
```

7.4Test_7seg (6/x)

```
// display digit on 7-seg.
void seg_display(void)
{
    close_seven_segment(); //clear GPIOC bit4-7
   //GPIOC->DOUT &= ~(0xF<<4);//clear GPIOC[7:4] bit4-7

    show_seven_segment(light_7seg,digit_7seg[light_7seg]);
    //GPIOE->DOUT &= ~(0xFF);          //clear 7 seg., GPIOE[7:0]=0
    //GPIOE->DOUT |= SEG_BUF[digit[3]];//show 7 seg.,GPIOE[7:0]=
    //GPIOC->DOUT |= (1<<(3+4));   //display 4th 7 seg.,GPIOC[7]=1

    if(!light_7seg--)light_7seg=3;
}
```

7.4Test_7seg– TMR0_IRQHandler() (7/x)

```
void InitTIMER0(uint32_t freq)
{
        // Select TIMER0 clock source
        // 0=extgernal, 1=32k,2=hclk,7=22M
        DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0);

        // Enable TIMER0 engine clock
        DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);

        // User must to call this function before any timer operations.
        DrvTIMER_Init ();
```

7.4Test_7seg (8/x)

```
        // Open the specified timer channel with operation mode.
        DrvTIMER_Open (E_TMR0, freq, E_PERIODIC_MODE);

        // enable the specified timer interrupt.
        DrvTIMER_EnableInt (E_TMR0);

        // Install the interrupt callback function
        DrvTIMER_SetTimerEvent (E_TMR0, 1,
(TIMER_CALLBACK) TMR_Callback, 0);

        // Start to count the specified timer channel.
        DrvTIMER_Start (E_TMR0);
}
```

7.4Test_7seg (9/x)

```
// Initial external 12M
void Init12M(void)
{
        UNLOCKREG();
        //1: enable, 0: disable
        DrvSYS_SetOscCtrl(E_SYS_XTL12M,1);
        //SYSCLK->PWRCON.XTL12M_EN = 1;
        LOCKREG();
}
```

7.5Test_7seg_Keypad–學習板電路圖

Control Pins used for 3x3 Keypad

Column control : GPA2, 1, 0

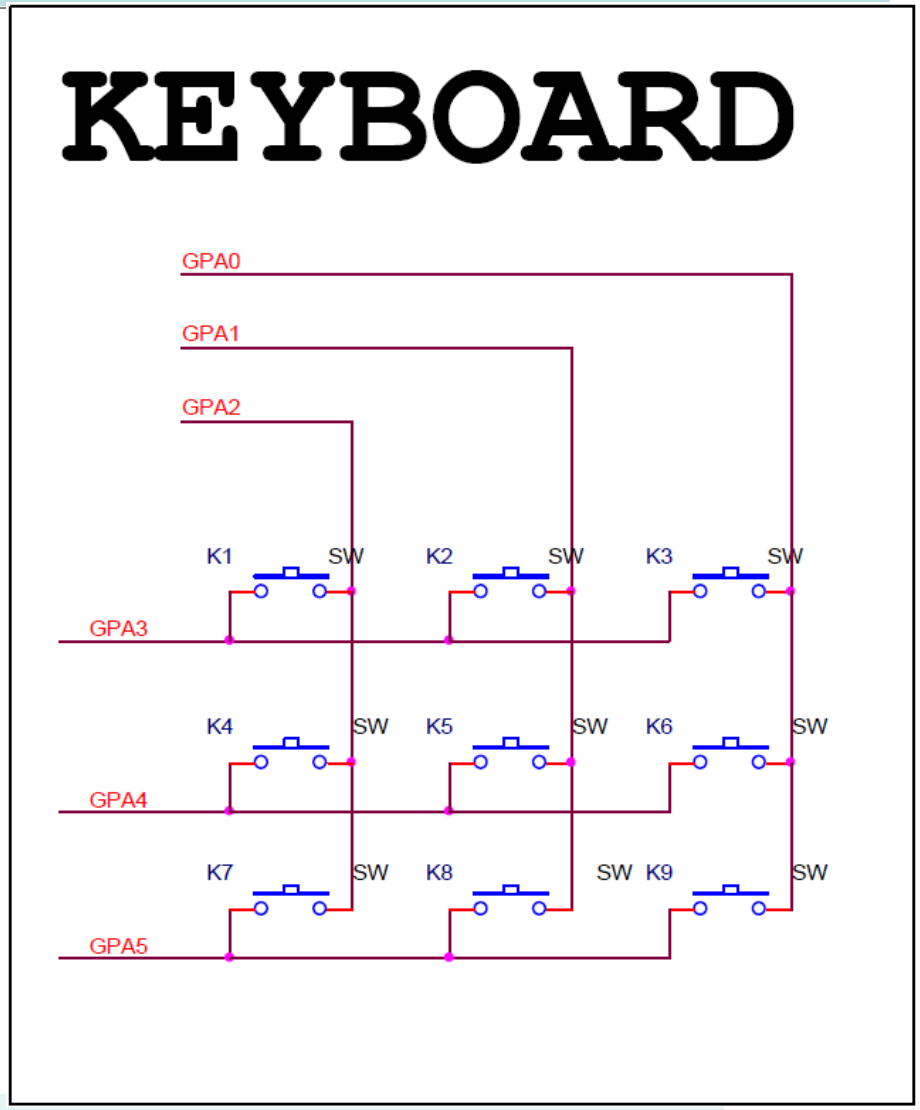
Raw control : GPA 3, 4, 5

- Key1 = GPA3 + GPA2
- Key2 = GPA3 + GPA1
- Key3 = GPA3 + GPA0
- Key4 = GPA4 + GPA2
- Key5 = GPA4 + GPA1
- Key6 = GPA4 + GPA0
- Key7 = GPA5 + GPA2
- Key8 = GPA5 + GPA1
- Key9 = GPA5 + GPA0

寫一程式，當按鍵按下時，顯示最後出現的4個數字。

GPA[2:0]依序輸出低電位(011, 101, 110)供按鍵讀取

GPA[5:3]讀取低電位



7.5Test_7seg_Keypad–學習板電路圖

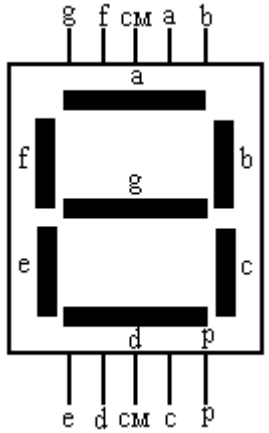
- 鍵盤按下時輸入信號為0，鬆開時輸入信號為1
- 通常按下或鬆開時間都會大於40ms，彈跳時間在10ms以內就會結束
- 若每5ms紀錄一次按鍵的狀態，當有按鍵完成時(按下_鬆開)，會出現00xx11的狀態，xx為彈跳期間。
- 程式設計1：
- 1.timer每5ms中斷一次，
- 2.檢查keypad是否有按鍵。若有按鍵，記錄按鍵的數字，和其狀態為0；否則紀錄狀態1。
- 3.檢查累積的狀態是否=00xx11。是：紀錄對應的數字，reset鍵盤狀態。
- 4.顯示下一個7seg.的數字
- 缺點：按鍵鬆開才能判斷一個數字。

7.5Test_7seg_Keypad– 7-seg. Control Pins

GPC4~7 control which 7-segment to turn on (1 = on, 0 = off)

- GPC4 : 1st 7-segment (LSB)
- GPC5 : 2nd 7-segment
- GPC6 : 3th 7-segment
- GPC7 : 4th 7-segment (MSB)

GPE0~7 control each segment to turn on (0 = on, 1 = off)



	g	**e**	**d**	**b**	**a**	**f**	**dp**	**c**	
	7	6	5	4	3	2	1	0	
0	1	0	0	0	0	0	1	0	0x82
1	1	1	1	0	1	1	1	0	0xEE
2	0	0	0	0	0	1	1	1	0x07
3	0	1	0	0	0	1	1	0	0x46
4	0	1	1	0	1	0	1	0	0x6A
5	0	1	0	1	0	0	1	0	0x52

ScanKeyDriver - OpenKeyPad

- 功能：將ScanKey的控制GPIOA[5:0]設定為雙向
- 函數： void OpenKeyPad(void)
- 範例：OpenKeyPad ()
- 與下列指令的作用相同
- GPIOA->PMDPMD0 = 0x03;
- GPIOA->PMDPMD1 = 0x03;
- GPIOA->PMDPMD2 = 0x03;
- GPIOA->PMDPMD3 = 0x03;
- GPIOA->PMDPMD4 = 0x03;
- GPIOA->PMDPMD5 = 0x03;

myNUC1xx-LB_002.c - KeyPadOpen

- 功能：將ScanKey的控制GPIOA[2:0]設定為輸出，GPIOA[5:3]設定為輸入
- 函數： void KeyPadOpen (void)
- 範例：KeyPadOpen ()
- 與下列指令的作用相同
- GPIOA->PMDPMD0 = 0x01;
- GPIOA->PMDPMD1 = 0x01;
- GPIOA->PMDPMD2 = 0x01;
- GPIOA->PMDPMD3 = 0x00;
- GPIOA->PMDPMD4 = 0x00;
- GPIOA->PMDPMD5 = 0x00;

7-Segment LED Driver - show_seven_segment

- 功能：將number顯示在第no個7段顯示器
- 函數： void show_seven_segment(unsigned char no, unsigned char number)
- 參數： no:第no個7段顯示器，(0-3)
- 參數：number: 顯示的數字，(0-9)
- 範例：show_seven_segment(3,digit)
- 與下列指令的作用相同
- GPIOE->DOUT &= ~(0xFF); //clear 7 seg., GPIOE bit0-7
- GPIOE->DOUT |= SEG_BUF[digit[3]];//show 7 seg.,
- GPIOC->DOUT |= (1<<(3+4)); //display 4th 7 seg.,GPIOC bit 7=1

7-Segment LED Driver - close_seven_segment

- 功能：關閉7段顯示器。
- 函數：void close_seven_segment(void)
- 參數：
- 範例： close_seven_segment();
- 與下列指令的作用相同
- GPIOC->DOUT &= ~(0xF<<4);

7.5Test_7seg_Keypad– (1/x)

```
#include "NUC1xx.h"
#include "Driver\DrvSYS.h"
#include "Driver\DrvGPIO.h"
#include "Driver\DrvTIMER.h"
#include "Seven_Segment.h"
#include "scankey.h"

    uint32_t tm0freq; //T=1/250=4ms
    uint8_t light_7seg_ith;
    uint8_t digit_7seg[4];

    uint8_t keynumber=0;
    uint8_t keypress=0;
    uint8_t keyrepeat=0xFF;
```

7.5Test_7seg_Keypad– (2/x)

```
int32_t main (void)
{
    //Initial GPIOC [7:4] to output mode
    seven_segment_open();   //set GPIOC_PMD[7:4]
    close_seven_segment();   //tuen off 7-seg. GPIOC[7:4]
    //Initial GPIOA [5:0]=11,QUASI mode
    OpenKeyPad();
    Init12M();
    tm0freq=250;            // frequency=250,4ms
    InitTimer0(tm0freq); // Initial Timer
    // main loop
    light_7seg_ith=3;
    while(1);
}
```

7.5 Timer 0 interrupt routine (3/x)

```
void seven_segment_open(void)
{   //Initial GPIOE [7:0] to output mode for 7-seg.
    DrvGPIO_Open(E_GPE, 0, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 1, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 2, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 3, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 4, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 5, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 6, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPE, 7, E_IO_OUTPUT);

    //Initial GPIOC [7:4] to output mode for nth 7-seg.
    DrvGPIO_Open(E_GPC, 4, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPC, 5, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPC, 6, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPC, 7, E_IO_OUTPUT);
}
```

7.5 Timer 0 interrupt routine (4/x)

```
// Initial external 12M
void Init12M(void)
{
        UNLOCKREG();
        //1: enable, 0: disable
        DrvSYS_SetOscCtrl(E_SYS_XTL12M,1);
        //SYSCLK->PWRCON.XTL12M_EN = 1;
        LOCKREG();
}
```

7.5 Timer 0 interrupt routine (5/x)

```
void InitTimer0(uint32_t freq)
{
        // Select TIMER0 clock source
        // 0=extgernal, 1=32k,2=hclk,7=22M
        DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0);

        // Enable TIMER0 engine clock
        DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);

        // User must to call this function before any timer operations.
        DrvTIMER_Init ();
```

7.5 Timer 0 interrupt routine (6/x)

```
    // Open the specified timer channel with operation mode.
    DrvTIMER_Open (E_TMR0, freq, E_PERIODIC_MODE);

    // enable the specified timer interrupt.
    DrvTIMER_EnableInt (E_TMR0);

    // Install the interrupt callback function
    DrvTIMER_SetTimerEvent (E_TMR0, 1,
(TIMER_CALLBACK) TMR_Callback, 0);

    // Start to count the specified timer channel.
    DrvTIMER_Start (E_TMR0);
}
```

7.5Test_7seg_Keypad– (7/x)

```
void TMR_Callback(void) // Timer0 interrupt subroutine
{
    //286clock,12.9us(22M),23.8us(12M)
    Scankeypad();

    //30clock,1.36us(22M),2.5us(12M)
    seg_keypad();
    //813clock,36.8us(22M),67.8us(12M)
    seg_display();
    while(keybufferptr)keybufferptr--;
}
```

RTC05： Test_RTC 8/x

```
void Scankeypad(void)
{
        int8_t irow,keyno=0;
        for (irow=1; irow <4; irow++)
        {
                GPIOA->DOUT |= (0x7);          //xxxx-x111
                GPIOA->DOUT &= ~(1<<(3-irow));  //011, 101, 110
                //scan row1 GPIOA[3] for 1,2,3
                if((GPIOA->PIN & (0x1<<3)) == 0)keyno = irow;
                //scan row2 GPIOA[4] for 4,5,6
                if((GPIOA->PIN & (0x1<<4)) == 0)keyno = irow+3;
                //scan row3 GPIOA[5] for 7,8,9
                if((GPIOA->PIN & (0x1<<5)) == 0)keyno = irow+6;
        }
```

RTC05： Test_RTC 9/x

```
//record key status
if (keyno)
{    // key pressed
     if (keyno == keynumber )                    // same key
          keystatus = (keystatus<<1 );           // record key status=0
     else //different key
          {
               keystatus = 0xFE;// initial key status=HHHHHHHL
               keynumber=keyno;// record key number
          }
}
else  // key not pressed
{     keystatus = (keystatus<<1 | 1 );// record key status=1
}
```

RTC05： Test_RTC 10/x

```
// if keystatus = LLxxHH, the key is pressed and released
        if( (keystatus & 0x33) == 0x3)
        {
                keybuffer[keybufferptr++]=keynumber;
                keystatus = 0xFF; //reset key status=HHHHHHHH
                keynumber=0;      //reset key number
                keypress=0;       //key released
        }
}
```

RTC05： Test_RTC 11/x

```
void seg_keypad(void)
{
        uint8_t irow;
        //: store keypad number to 7-seg
        if(keybufferptr)
        {
                for (irow=3; irow >0; irow--)
                        digit_7seg[irow]=digit_7seg[irow-1];
                digit_7seg[0]=keybuffer[0];
        }
}
```

RTC05： Test_RTC 12/x

```
void seg_display(void)
{
        close_seven_segment(); //clear GPIOC bit4-7
        //GPIOC->DOUT &= ~(0xF<<4);//clear GPIOC[7:4] bit4-7
        show_seven_segment(light_7seg,digit[light_7seg]);
        //GPIOE->DOUT &= ~(0xFF);  //clear 7 seg.,
        //GPIOE->DOUT |= SEG_BUF[digit[3]];//show 7 seg.,
        //GPIOC->DOUT |= (1<<(3+4));//display 4th 7 seg.
        if(!light_7seg--)light_7seg=3;
}
```

7.6Test_7seg_Keypad–學習板電路圖

Control Pins used for 3x3 Keypad

Column control : GPA2, 1, 0

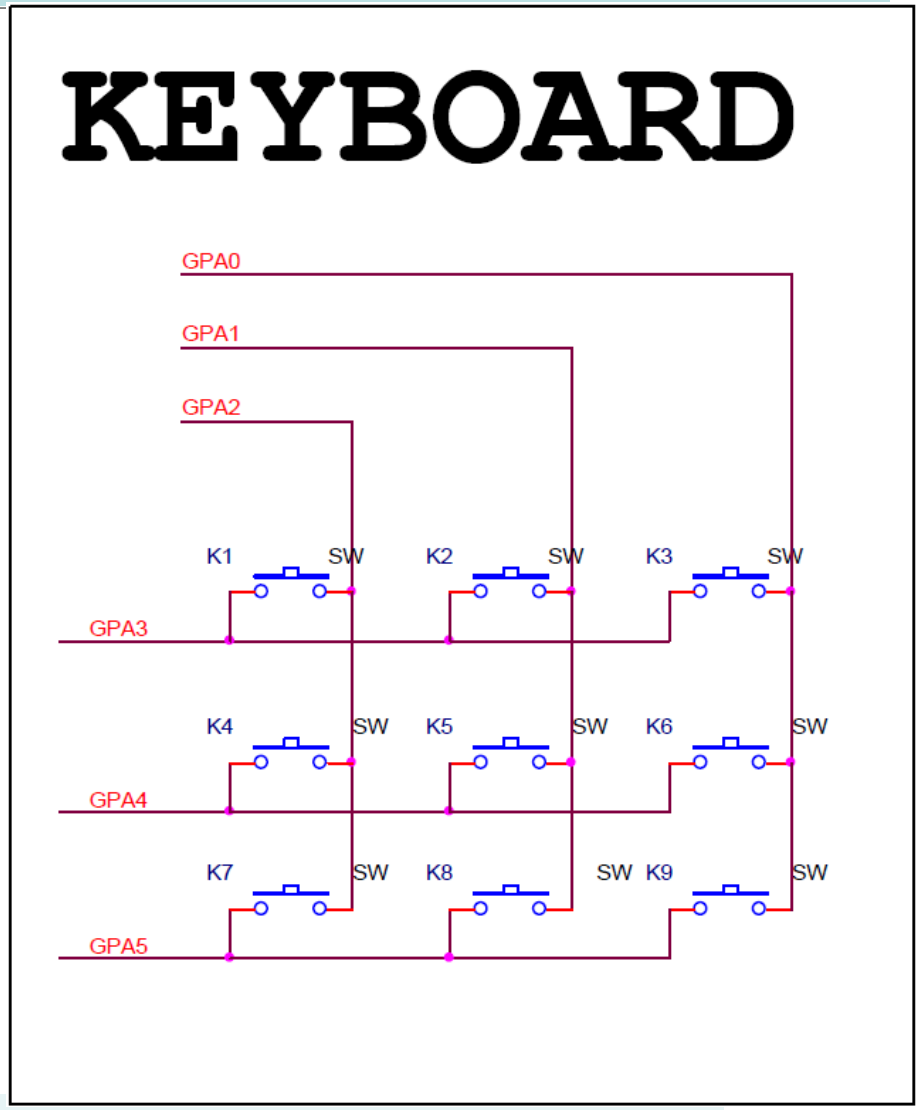
Raw control : GPA 3, 4, 5

- Key1 = GPA3 + GPA2
- Key2 = GPA3 + GPA1
- Key3 = GPA3 + GPA0
- Key4 = GPA4 + GPA2
- Key5 = GPA4 + GPA1
- Key6 = GPA4 + GPA0
- Key7 = GPA5 + GPA2
- Key8 = GPA5 + GPA1
- Key9 = GPA5 + GPA0

寫一程式，當按鍵按下時，顯示最後出現的4個數字。

GPA[2:0]依序輸出低電位(011, 101, 110)供按鍵讀取

GPA[5:3]讀取低電位



7.6Test_7seg_Keypad

- 鍵盤按下時輸入信號為0，鬆開時輸入信號為1
- 通常按下或鬆開時間都會大於40ms，彈跳時間在10ms以內就會結束
- 若每5ms紀錄一次按鍵的狀態，當有按鍵完成時(按下_鬆開)，若出現0000表示按鍵已經按下，出現11表示按鍵鬆開。
- 程式設計2：
- 1.timer每5ms中斷一次，
- 2.檢查keypad是否有按鍵。若有按鍵，記錄按鍵的數字，和其狀態為0；否則紀錄狀態1。
- 3.檢查按鍵狀態。若按鍵未按下(狀態=0)且累積的狀態=0000，則按鍵按下(紀錄對應的數字，按鍵狀態=1)。若按鍵已按下(狀態=1)且累積的狀態=11，則按鍵鬆開(按鍵狀態=0)。
- 4.顯示下一個7seg.的數字

7.6Test_7seg_Keypad 1/x

```
void Scankeypad(void)
{
        int8_t irow,keyno=0;
        for (irow=1; irow <4; irow++)
        {
                GPIOA->DOUT |= (0x7);          //xxxx-x111
                GPIOA->DOUT &= ~(1<<(3-irow));  //011, 101, 110
                //scan row1 GPIOA[3] for 1,2,3
                if((GPIOA->PIN & (0x1<<3)) == 0)keyno = irow;
                //scan row2 GPIOA[4] for 4,5,6
                if((GPIOA->PIN & (0x1<<4)) == 0)keyno = irow+3;
                //scan row3 GPIOA[5] for 7,8,9
                if((GPIOA->PIN & (0x1<<5)) == 0)keyno = irow+6;
        }
```

7.6Test_7seg_Keypad 2/x

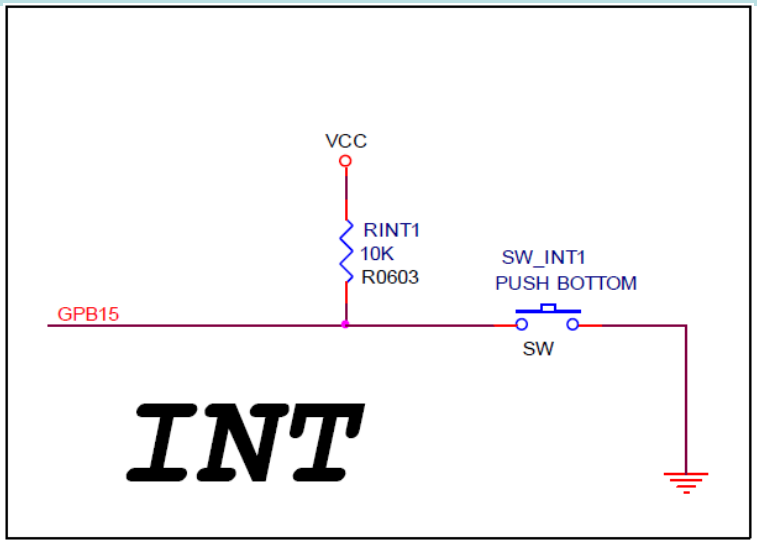
```
//record key status
if (keyno)
{    // key pressed
     if (keyno == keynumber )                    // same key
          keystatus = (keystatus<<1 );           // record key status=0
     else //different key
          {
               keystatus = 0xFE;// initial key status=HHHHHHHL
               keynumber=keyno;// record key number
          }
}
else  // key not pressed
{     keystatus = (keystatus<<1 | 1 );// record key status=1
}
```

7.6Test_7seg_Keypad 3/x

```
// if get 4 continuous low level, the key is pressed.
                    if( !keypress && !(keystatus & 0x0F) )
                    {
                            keypress=1;
                            keybuffer[keybufferptr++]=keynumber;
                    }
// check key released
                    if(keypress && ((keystatus & 0x3)==3) )
                    {
                            keystatus = 0xFF; //reset key status
                            keynumber=0;      //reset key number
                            keypress=0;       //key released
                    }
}
```

7.7Test_7seg_SWInt_Bounce–計算彈跳的接觸時間

- SW Int: GPB15
  - **GPB15**: 0=pressed, 1= not pressed

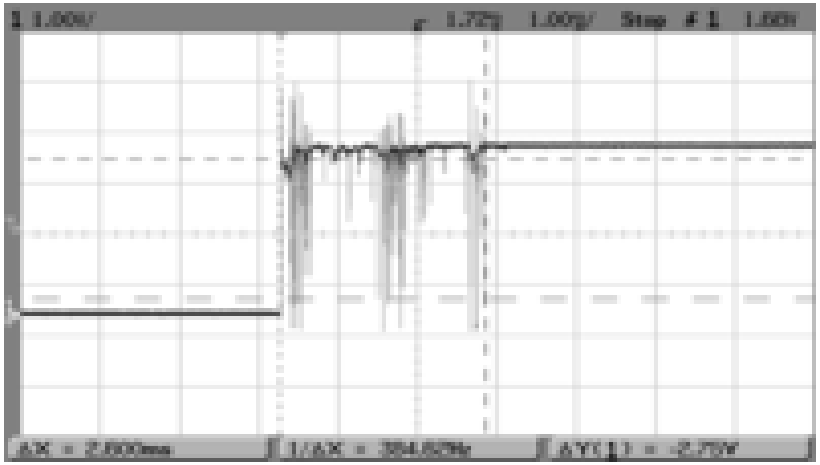


寫一程式，當按鍵按下時，啟動計時器，紀錄一秒內每一次電壓變化的時間點。然後將每一段時間顯示在7段顯示器。

顯示的時間以msec為單位，顯示小數點。

7.7Test_7seg_SWInt_Bounce–計算彈跳的次數

- 接觸彈跳(bounce)是機械開關常見的問題。當開關接觸時，由於動量和彈性造成彈跳。
- 彈跳時間通常在10ms以內。
- 彈跳可以藉由硬體電路消除，或使用軟體處理。



- 鍵盤的掃描頻率是 2.64ms，共掃描 8 次= 2.64ms * 8 = 約 21.12ms，與下次間隔0.88ms，合計22ms。
- 軟體：第一次偵測到低電位，延遲20ms，再檢測一次是否為低電位?若是高電位表示一個雜訊，不予處理。若是低電位，表示一個有效的按鍵。

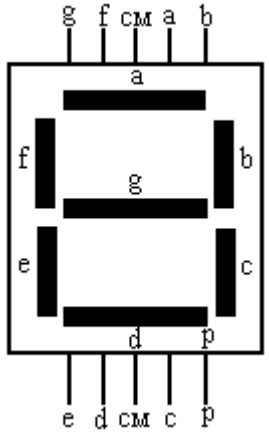
http://www.embedded.com/electronics-blogs/break-points/4024944/The-secret-life-of-switches

Test_7seg_SWInt_Bounce– Control Pins used for

GPC4~7 control which 7-segment to turn on (1 = on, 0 = off)

- GPC4 : 1st 7-segment (LSB)
- GPC5 : 2nd 7-segment
- GPC6 : 3th 7-segment
- GPC7 : 4th 7-segment (MSB)

GPE0~7 control each segment to turn on (0 = on, 1 = off)



	g	e	d	b	a	f	dp	c	
	7	6	5	4	3	2	1	0	
0	1	0	0	0	0	0	1	0	0x82
1	1	1	1	0	1	1	1	0	0xEE
2	0	0	0	0	0	1	1	1	0x07
3	0	1	0	0	0	1	1	0	0x46
4	0	1	1	0	1	0	1	0	0x6A
5	0	1	0	1	0	0	1	0	0x52

7.7Test_7seg_SWInt_Bounce (1/x)

```
int main (void)
{
    int32_t  time_limit,value;

    seven_segment_open(); //743 states
    //GPIOE->PMD.PMD0 = 1; GPIOE[7:0]=output
    //GPIOC->PMD>PMD4 = 1; GPIOC[7-4]=output
    close_seven_segment(); //clear GPIOC[7:0] //151 states

    // Initial GPIO GPB15(SW Int) to input mode
    DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);
    //GPIOB->PMD.PMD15 = 0x00;            //13 states

    while (1)
    {
        value = 0;
```

7.7Test_7seg_SWInt_Bounce (2/x)

```
while (1)
{

    // wait for SW_INT button pressed or time limited
    // get GPIOB bit15 GPIOB->PIN & 0x8000
    time_limit=1500000;
    //while((DrvGPIO_GetBit(E_GPB,15)!=0) && (--time_limit)); //39 states
    while(((GPIOB->PIN & (1<<15))!=0)  && (--time_limit)); //14 states
    //if time limited, the  key did not press
    if(! time_limit) break;
    //the key pressed, imcrease times
    value++;
    //wait for key release
    while((GPIOB->PIN & (1<<15))==0 ); //14 states
}
```

7.7Test_7seg_SWInt_Bounce (3/x)

```
    if(value)
    {       //show bounce times
            show_seven_segment(0,value); //448 states
            // delay
            Delay(5000000); //30013 states
            //turn off 7-seg.
            close_seven_segment(); //clear GPIOC bit4-7 //151 states
    }
  }
}
```

General Disclaimer

The Lecture is strictly used for educational purpose.

MAKES NO GUARANTEE OF VALIDITY

- **The lecture cannot guarantee the validity of the information found here.** The lecture may recently have been changed, vandalized or altered by someone whose opinion does not correspond with the state of knowledge in the relevant fields. Note that most other encyclopedias and reference works also have similar disclaimers.

No formal peer review

- The lecture is not uniformly peer reviewed; while readers may correct errors or engage in casual peer review, they have no legal duty to do so and thus all information read here is without any implied warranty of fitness for any purpose or use whatsoever. Even articles that have been vetted by informal peer review or *featured article* processes may later have been edited inappropriately, just before you view them.

No contract; limited license

- Please make sure that you understand that the information provided here is being provided freely, and that no kind of agreement or contract is created between you and the owners or users of this site, the owners of the servers upon which it is housed, the individual Wikipedia contributors, any project administrators, sysops or anyone else who is in *any way connected* with this project or sister projects subject to your claims against them directly. You are being granted a limited license to copy anything from this site; it does not create or imply any contractual or extracontractual liability on the part of Wikipedia or any of its agents, members, organizers or other users.
- There is **no agreement or understanding between you and the content provider** regarding your use or modification of this information beyond the Creative Commons Attribution-Sharealike 3.0 Unported License (CC-BY-SA) and the GNU Free Documentation License (GFDL);

General Disclaimer

Trademarks

- Any of the trademarks, service marks, collective marks, design rights or similar rights that are mentioned, used or cited in the lectures are the property of their respective owners. Their use here does not imply that you may use them for any purpose other than for the same or a similar informational use as contemplated by the original authors under the CC-BY-SA and GFDL licensing schemes. Unless otherwise stated , we are neither endorsed by nor affiliated with any of the holders of any such rights and as such we cannot grant any rights to use any otherwise protected materials. Your use of any such or similar incorporeal property is at your own risk.

Personality rights

- The lecture may portray an identifiable person who is alive or deceased recently. The use of images of living or recently deceased individuals is, in some jurisdictions, restricted by laws pertaining to personality rights, independent from their copyright status. Before using these types of content, please ensure that you have the right to use it under the laws which apply in the circumstances of your intended use. *You are solely responsible for ensuring that you do not infringe someone else's personality rights.*