General Purpose IO

- 2013/4/4

課程大綱 (Lesson Outline)

- **Cortex-M0 MCU之基本輸出/輸入原理**

- **實習範例1 : LED顯示控制**
- **實習範例 2: 按鍵+ LED顯示控制**
- **實習範例 3:按鍵+蜂鳴器控制**
- **實習範例 4:在七段顯示器顯示0000-9999**
- **實習範例 5: 3x3按鍵+七段顯示器**
- **實習範例 6: 顯示彈跳的次數**

General Purpose I/O (GPIO) Features

- NuMicro NUC140有80個通用I/O接腳，這些接腳同時具有其他功能，使用前必須先設定。
- 80個接腳分為GPIOA, GPIOB, GPIOC, GPIOD 與 GPIOE，每個PORT最多16個接腳。
- 每個接腳都是獨立的，都有對應的暫存器位元來設定接腳功能模式與資料。
- I/O接腳上的I/O類型可由軟體獨立地設定為高阻態輸入，推挽輸出，開漏輸出或准雙端模式。
- reset之後，所有接腳的I/O類型均為準雙端模式。
- 每個 I/O接腳有一個電阻值110 KΩ~300 KΩ的弱上拉電阻接到VDD上，VDD的範圍從 5.0 V 到 2.5 V。

General Purpose I/O (GPIO) Features

- 輸入模式
- [說明] 設定GPIOx_PMD[n]=00 為輸入模式，則 I/O 接腳為高阻抗，GPIOx_PIN[n]反映引腳狀態。
- [設定] GPIOx->PMD.PMDn=00b
- [查詢] (GPIOx->PIN & (1<<n))的值反映相應GPIOx[n]的狀態。

- 輸出模式
- [說明] 設定GPIOx_PMD[n]=01 為輸出模式，則 I/O 接腳支持數位輸出功能，有source/sink電流能力。
- [設定] GPIOx->PMD.PMDn=01b
- [輸出] GPIOx->DOUT &= ~(1<<n) //set GPIOx[n]=0
- GPIOx->DOUT |= (1<<n) //set GPIOx[n]=1
- [查詢] (GPIOx->PIN & (1<<n))的值反映相應GPIOx[n]的狀態。

General Purpose I/O (GPIO) 接腳模式

- 開漏模式
- [說明] 設定GPIOx_PMD[n]=10 為Open_Drain模式，則 I/O 接腳支援sink電流，需要外加上拉電阻。
- [設定] GPIOx->PMD.PMDn=10b
- [輸出] GPIOx->DOUT &= ~(1<<n) //set GPIOx[n]=0
- GPIOx->DOUT |= (1<<n) //set GPIOx[n]=1
- [查詢] (GPIOx->PIN & (1<<n))的值反映相應GPIOx[n]的狀態。

General Purpose I/O (GPIO) 接腳模式

- 準雙端模式
- [說明] 設定GPIOx_PMD[n]=11 為準雙端模式，則 I/O 接腳支持輸入/數位輸出功能，但 source 電流僅有30-200uA 。准雙端輸出是 80C51 及其系列產品共有的模式。要實現數位輸入，需要先將 GPIOx[n] 設定為1(reset後， GPIO->PMD=0xFFFFFFFF, GPIOx->DOUT =0xFFFF)。
- [設定] GPIOx->PMD.PMDn=11b
- [輸出] GPIOx->DOUT &= ~(1<<n) //set GPIOx[n]=0
- GPIOx->DOUT |= (1<<n) //set GPIOx[n]=1
- [查詢] (GPIOx->PIN & (1<<n))的值反映相應GPIOx[n]的狀態。

GPIOA Register

寄存器	偏移量	R/W	描述	復位後的值
GPIOA_PMD	GP_BA+0x000	R/W	接腳 I/O 模式控制	0xFFFF_FFFF
GPIOA_OFFD	GP_BA+0x004	R/W	接腳 OFF 數位致能	0x0000_0000
GPIOA_DOUT	GP_BA+0x008	R/W	數據輸出值	0x0000_FFFF
GPIOA_DMASK	GP_BA+0x00C	R/W	數據輸出寫屏蔽	0x0000_0000
GPIOA_PIN	GP_BA+0x010	R	接腳數值	0x0000_XXXX
GPIOA_DBEN	GP_BA+0x014	R/W	去抖動致能	0x0000_0000
GPIOA_IMD	GP_BA+0x018	R/W	中斷模式控制	0x0000_0000
GPIOA_IEN	GP_BA+0x01C	R/W	中斷致能	0x0000_0000
GPIOA_ISRC	GP_BA+0x020	R/W	中斷源標誌	0xXXXX_XXXX

GPIOA_PMD: I/O Mode Control

設定GPIO引腳的輸入/輸出模式

bit	Symbol	Descriptions
[2n+1:2n]	PMDn	**GPIOx I/O Pin[n] Mode Control**
[1:0]	PMD0	00 = GPIO port [n] pin is in INPUT mode
[3:2]	PMD1	01 = GPIO port [n] pin is in OUTPUT mode
[5:4]	PMD2	10 = GPIO port [n] pin is in Open-Drain mode
[7:6]	PMD3	11 = GPIO port [n] pin is in Quasi-bidirectional mode
…	…	After RESET, PMD=0xFFFFFFFF

GPIOA_OFFD

If input is analog signal, users can disable GPIO digital input path to avoid creepage.

bit	Symbol	Descriptions
[31:16]	OFFD	**GPIOx Pin[n] OFF 數位輸入通道致能** 用於控制 bit 對應的 GPIO 的數位輸入通道是否禁用。若輸入為類比信號，可以關閉數位輸入通道避免漏電。 1 = 禁用 IO 數位輸入通道（數位輸入拉低） 0 = 使能 IO 數位輸入通道 After RESET, OFFD=0

GPIOA_DOUT

控制引腳的輸出為0/1

bit	Symbol	Descriptions
[n] [0] [1] [2] [3] …	DOUT[n] DOUT[0] DOUT[1] DOUT[2] DOUT[3] …	**GPIOx Pin[n] Output Value** if the GPIO pin is configures as output, open-drain and quasi-mode: 1 = GPIO port [A/B/C/D/E] Pin[n] will drive High 0 = GPIO port [A/B/C/D/E] Pin[n] will drive Low After RES

GPIOA_DMASK

保護引腳的輸出，不被改變

bit	Symbol	Descriptions
[n]	DMASK[n]	**PORT[A/B/C/D/E] 數據輸出寫入遮蔽**
[0]	DMASK[0]	用於保護相應暫存器 GPIOx_DOUT bit[n]。在設置
[1]	DMASK[1]	DMASK bit[n] 為1，相應的GPIOx_DOUT[n] bit 被
[2]	DMASK[2]	保護。寫入信號會被遮蔽，不能向保護位元寫入資
[3]	DMASK[3]	料。
…	…	1 = 相應的 GPIOx_DOUT[n] bit 被保護
		0 = 相應的 GPIOx_DOUT[n] bit 能被更新
		After RESET, DMASK=0

GPIOA_PIN

查詢引腳的狀態0/1

bit	Symbol	Descriptions
[n]	PIN[n]	**Port [A/B/C/D/E] Pin Values**
[0]	PIN[0]	Each bit of the register reflects the actual status of the
[1]	PIN[1]	respective GPIO pin.
[2]	PIN[2]	Bit=1, the corresponding pin status is high,
[3]	PIN[3]	Bit=0, the pin status is low.
…	…	After RESET, PIN=0xFFFF

GPIOA_DBEN

啟用去彈跳功能

bit	Symbol	Descriptions
[n]	DBEN[n]	**Port [A/B/C/D/E] 輸入信號去抖動致能**
[0]	DBEN[0]	DBEN[n] 用於致能相應位元的去抖動功能。如果輸入信號脈衝寬度不能被兩個連續的去抖動採樣週期所採樣，則輸入信號被視為信號抖動，從而不會觸發中斷。去抖動的時鐘源由 DBNCECON[4] 控制，去抖動的採樣週期由 DBNCECON[3:0] 控制。 DBEN[n] 僅用於邊沿觸發 (edge-trigger)中斷，不能用於電平觸發 (level trigger)中斷。 1 = 致能 bit[n] 去抖動功能 0 = 禁用 bit[n] 去抖動功能 去抖動功能對邊沿觸發中斷有效，對於電平觸發中斷模式該致能位元不起作用 。 After RESET, DBEN=0
[1]	DBEN[1]	
[2]	DBEN[2]	
[3]	DBEN[3]	
…	…	

GPIOA_IMD

設定引腳中斷的觸發模式

bit	Symbol	Descriptions
[n]	IMD[n]	**Port [A/B/C/D/E]**邊沿或電平檢測中斷控制
[0]	IMD[0]	IMD[n] 用於控制中斷是電平觸發或邊沿觸發。若為邊沿觸發中斷，觸發源可由去抖動控制；若為電平觸發中斷，輸入源由一個 HCLK 時鐘採樣並產生中斷。
[1]	IMD[1]	1 = 電平觸發中斷
[2]	IMD[2]	0 = 邊沿觸發中斷
[3]	IMD[3]	如果設置管腳為電平觸發中斷，則在寄存器 GPIOx_IEN 中，只能設置一個電平。若設置了兩個電平去觸發中斷，則設置被忽略，不會產生中斷。
…	…	去抖動功能對於邊沿觸發有效，對於電平觸發中斷無效。
		After RESET, IMD=0

GPIOA_IEN

啟用引腳中斷

bit	Symbol	Descriptions
[n+16]	IR_EN[n]	**Port [A/B/C/D/E]輸入上升沿或輸入高電平中斷致能** 當設置 IR_EN[n] 位為 1： 中斷是電平觸發模式， PIN[n] 為高電平時將產生中斷。 中斷是邊沿觸發模式，PIN[n] 由低電平上升到高電平產生中斷 1 =致能 PIN[n] 高電平或由低電平到高電平變化的中斷 0 = 禁用 PIN[n] 高電平或由低電平到高電平變化的中斷 After RESET,, IR_EN=0
[n]	IF_EN[n]	**Port [A/B/C/D/E]**輸入下降沿或輸入低電平中斷致能 當設置 IF_EN[n] 位為 1： 中斷是電平觸發模式， PIN[n] 的狀態為低電平時將產生中斷。 中斷是邊沿觸發模式，PIN[n] 的狀態由高電平到低電平變化時將產生中斷。 1 =致能 PIN[n] 低電平或由高電平到低電平變化的中斷 0 = 禁用 PIN[n] 低電平或由高電平到低電平變化的中斷 After RESET,, IF_EN=0

GPIOA_ISRC

讀取/清除引腳中斷來源

bit	Symbol	Descriptions
[n]	ISRC[n]	**Port [A/B/C/D/E]** Interrupt Source Flag
[0]	ISRC[0]	讀：
[1]	ISRC[1]	1 = 表示GPIOx[n] 產生中斷
[2]	ISRC[2]	0 = 表示GPIOx[n] 沒有中斷
[3]	ISRC[3]	寫：
…	…	1= 清除相應的未處理中斷
		0= 無動作
		After RESET=0

GPIOA_DBNCECON

讀取/清除引腳中斷來源

bit	Symbol	Descriptions
[5]	ICLK_ON	中斷用時鐘 On 模式 1 = reset後,所有I/O接腳邊緣偵測為啟用 0 = 只有I/O接腳對應的GPIOx_IEN為1,則邊緣偵測為啟用 關掉此位元，可節省耗電
[4]	DBCLKSRC	選擇去抖動計數器時鐘源 1 = 去抖動計數器時鐘源為內部 10 kHz 振蕩器 0 = 去抖動計數器時鐘源為 HCLK
[3:0]	DBCLKDEL	選擇去抖動採樣週期 0:採樣中斷輸入每 1 clocks 1 次 1:採樣中斷輸入每 2 clocks 1 次 2:採樣中斷輸入每 4 clocks 1 次 3:採樣中斷輸入每 8 clocks 1 次 4:採樣中斷輸入每 16 clocks 1 次 15:採樣中斷輸入每 2^{15} clocks 1 次

GPIO Driver

_DRVGPIO_DOUT	control I/O Bit Output/Input Control Register of the specified pin.
	_DRVGPIO_DOUT (E_GPA, 1) = 1;
	u8PinValue = _DRVGPIO_DOUT (E_GPB, 3);

GPA_[n] / GPB_[n] / GPC_[n] / GPD_[n] / GPE_[n]	same as _DRVGPIO_DOUT macro but without any parameters GPA_1 = 1; u8PinValue = GPB_3;

GPIO Driver

DrvGPIO_ Open	Set GPIO pin to operation mode. GPIOx_PMD DrvGPIO_Open (E_GPA, 0, E_IO_OUTPUT); DrvGPIO_Open (E_GPA, 1, E_IO_INPUT);
DrvGPIO_ Close	set the pin to quasi-bidirectional mode. GPIOx_PMD DrvGPIO_Close (E_GPA, 0);
DrvGPIO_ SetBit	Set GPIO pin to 1. GPIOx_DOUT DrvGPIO_SetBit (E_GPA, 0);
DrvGPIO_ GetBit	Get GPIO pin value. GPIOx_PIN i32BitValue = DrvGPIO_GetBit (E_GPA, 1);
DrvGPIO_ ClrBit	Set GPIO pin to 0. GPIOx_DOUT DrvGPIO_ClrBit (E_GPA, 0);

GPIO Driver

DrvGPIO_ SetPortBits	Set the output port value to the specified GPIO port. GPIOx_DOUT DrvGPIO_SetPortBits (E_GPA, 0x1234);
DrvGPIO_ GetPortBits	Get the input port value from the specified GPIO port. GPIOx_PIN i32PortValue = DrvGPIO_GetPortBits (E_GPA);
DrvGPIO_ GetDoutBit	Get the bit value from the specified Data Output Value Register. GPIOx_DOUT i32BitValue = DrvGPIO_GetDoutBit (E_GPA, 1);
DrvGPIO_ GetPortDoutBits	Get the port value from the specified Data Output Value Register. GPIOx_DOUT i32PortValue = DrvGPIO_GetPortDoutBits (E_GPA);

GPIO Driver

DrvGPIO_ SetPortMask	protect the write data function of the corresponding GPIO pins. GPIOx_DMASK DrvGPIO_SetPortMask (E_GPA, 0x11);
DrvGPIO_ GetPortMask	Get the port value from the specified Data Output Write Mask Register. GPIOx_DMASK i32MaskValue = DrvGPIO_GetPortMask (E_GPA);
DrvGPIO_ ClrPortMask	remove the write protect function of the corresponding GPIO pins. GPIOx_DMASK DrvGPIO_ClrPortMask (E_GPA, 0x11);

GPIO Driver

DrvGPIO_ SetBitMask	protect the write data function of the corresponding GPIO pin. GPIOx_DMASK DrvGPIO_SetBitMask (E_GPA, 0);
DrvGPIO_ GetBitMask	Get the bit value from the specified Data Output Write Mask Register. GPIOx_DMASK i32MaskValue = DrvGPIO_GetBittMask (E_GPA, 0);
DrvGPIO_ ClrBitMask	remove the write protect function of the the corresponding GPIOpin. GPIOx_DMASK DrvGPIO_ClrBitMask (E_GPA, 0);

GPIO Driver

DrvGPIO_ EnableDigitalInputBit	Enable IO digital input path of the specified GPIO input pin. GPIOx_OFFD DrvGPIO_EnableDigitalInputBit (E_GPA, 0);
DrvGPIO_ DisableDigitalInputBit	Disable IO digital input path of the specified GPIO input pin. GPIOx_OFFD DrvGPIO_DisableDigitalInputBit (E_GPA, 0);

GPIO Driver

DrvGPIO_ EnableDebounce	Enable the de-bounce function of the specified GPIO input pin. GPIOX_DBEN DrvGPIO_EnableDebounce (E_GPA, 0);
DrvGPIO_ DisableDebounce	Disable the de-bounce function of the specified GPIO input pin. GPIOX_DBEN DrvGPIO_DisableDebounce (E_GPA, 0);
DrvGPIO_ SetDebounceTime	Set the interrupt de-bounce sampling time based on the de-bounce counter clock source. DrvGPIO_SetDebounceTime (4, E_DBCLKSRC_10K); GPIOX_DBNCECON
DrvGPIO_ GetDebounceSampleCycle	get the number of de-bounce sampling cycle selection. GPIOX_DBNCECON i32CycleSelection = DrvGPIO_GetDebounceSampleCycle ();

GPIO Driver

DrvGPIO_ EnableInt	Enable the interrupt function of the specified GPIO pin. GPIOX_IEN DrvGPIO_EnableInt (E_GPB, 13, E_IO_RISING, E_MODE_EDGE);
DrvGPIO_ DisableInt	Disable the interrupt function of the specified GPIO pin. GPIOX_IEN DrvGPIO_DisableInt (E_GPB, 13);
DrvGPIO_ SetIntCallback	Install the interrupt callback function for GPA/GPB port and GPC/GPD/GPE port, DrvGPIO_SetIntCallback (GPABCallback, GPCDECallback);

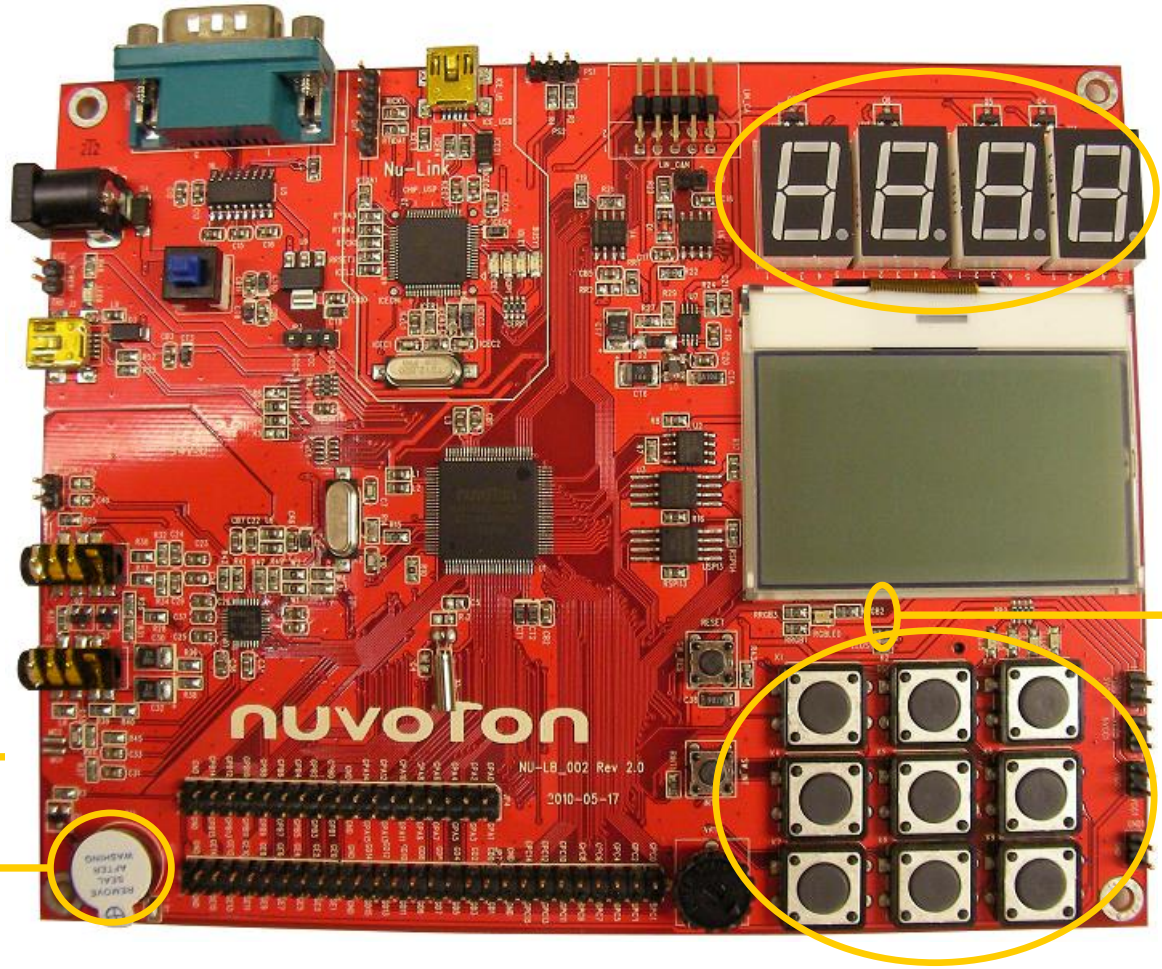
GPIO Driver

DrvGPIO_ EnableEINT0	Enable the interrupt function for external GPIO interrupt from /INT0(GPB.14) pin. GPIOx_IEN DrvGPIO_EnableEINT0 (E_IO_BOTH_EDGE, E_MODE_EDGE, EINT1Callback);
DrvGPIO_ DisableEINT0	Disable the interrupt function for external GPIO interrupt from /INT0 (GPB.14) pin. GPIOx_IEN DrvGPIO_DisableEINT0 ();
DrvGPIO_ EnableEINT1	Enable the interrupt function for external GPIO interrupt from /INT1(GPB.15) pin. GPIOx_IEN DrvGPIO_EnableEINT1 (E_IO_FALLING, E_MODE_LEVEL, EINT1Callback);
DrvGPIO_ DisableEINT1	Disable the interrupt function for external GPIO interrupt from /INT1(GPB.15) pin. GPIOx_IEN DrvGPIO_DisableEINT1 ();

GPIO Driver

DrvGPIO_ GetIntStatus	Get the port value from the specified Interrupt Trigger Source Indicator Register. GPIOx_ISRC i32INTStatus = DrvGPIO_GetIntStatus (E_GPA);
DrvGPIO_ InitFunction	Initialize the specified function and configure the relative pins for specified function used. DrvGPIO_InitFunction (E_FUNC_UART0);
DrvGPIO_ GetVersion	return the version number of GPIO driver i32GPIOVer = DrvGPIO_GetVersion ();

NU-LB-NUC140 開發板



7 segment LEDs

RGB LED
GPA12,13,14

Keypad

SW Int
GPB15

Buzzer
GPB11

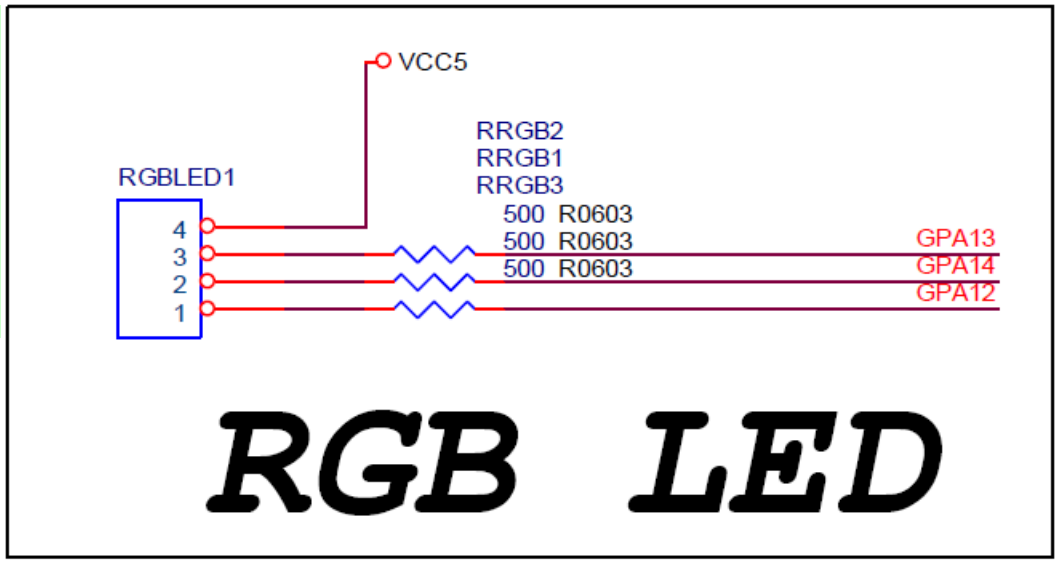
3.1Test_RGBled 學習板電路圖

- 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，設定藍色，綠色，紅色LED依序發亮0.5秒，然後全暗0.5秒；反覆執行。LED分別由GPIO A的bit12,13,14控制，輸出0時發亮，輸出1時，不亮。

DrvGPIO_Open

- 功能：設定GPIO接腳的Input/Output，有四種模式：輸入、輸出、開漏、準雙向。
- 函數：int32_t DrvGPIO_Open(E_DRVGPIO_PORT port, int32_t i32Bit, E_DRVGPIO_IO mode)
- 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.
- 參數： Mode: E_DRVGPIO_IO, set the specified GPIO pin to be E_IO_INPUT(00), E_IO_OUTPUT(01), E_IO_OPENDRAIN(10) or E_IO_QUASI(11) mode.
- 範例：DrvGPIO_Open(E_GPA, 12, E_IO_OUTPUT); //61states
- 與下列指令的作用相同
- GPIOA->PMD.PMD12 = 0x1; //13states

DrvGPIO_SetBit

- 功能：設定GPIO接腳的值=1。
- 函數： int32_t DrvGPIO_SetBit(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_SetBit(E_GPA,13); //GPIOA_13=1 off, 35states
- 與下列指令的作用相同
- GPIOA->DOUT |= (1<<13); //10states

DrvGPIO_ClrBit

- 功能：設定GPIO接腳的值=0。
- 函數： int32_t DrvGPIO_ClrBit(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_ClrBit(E_GPA,12); //GPIOA_12=0 on, 35states
- 與下列指令的作用相同
- GPIOA->DOUT &= ~(0x1<<12); //10states

DrvSYS_Delay

- 功能：generate the delay time (us)。
- 函數：void DrvSYS_Delay (uint32_t us);
- 參數： **us**: Delay time. The maximal delay time is 335000 us.
- 範例：DrvSYS_Delay (5000); //delay 5000us
- 與下列指令的作用相同
- 說明：DrvSYS_Delay使用SysTick作為計數器，最大可以設定2^{24}=16777215；時脈使用HCLK，若HCLK為22MHz，則數22次為1us。當計數器設定16777215，相當於762600us。也就是說DrvSYS_Delay(762600)是最大的設定值。
- 若HCLK=12MHz，最大值DrvSYS_Delay(1398101) 。
- 若HCLK=50MHz，最大值DrvSYS_Delay(335544) 。

3.1Test_RGBled (1/5)

```
int main (void)
{
    // Initial GPIO GPA12,13,14 to output mode
    DrvGPIO_Open(E_GPA, 12, E_IO_OUTPUT);
    //GPIOA->PMD.PMD12 = 0x01;
    DrvGPIO_Open(E_GPA, 13, E_IO_OUTPUT);
     //GPIOA->PMD.PMD13 = 0x01;
    DrvGPIO_Open(E_GPA, 14, E_IO_OUTPUT);
    //GPIOA->PMD.PMD14 = 0x01;
    while (1)
          {
```

3.1Test_RGBled (2/5)

```
// set RGBled to Blue

// GPA12 = Blue,  0 : on, 1 : off
DrvGPIO_ClrBit(E_GPA,12);
//GPIOA->DOUT &= ~(0x1<<12);

DrvGPIO_SetBit(E_GPA,13);
//GPIOA->DOUT |= (1<<13);

DrvGPIO_SetBit(E_GPA,14);
//GPIOA->DOUT |= (1<<14);

DrvSYS_Delay(500000); //delay 500000us=500ms=0.5s
```

3.1Test_RGBled (3/5)

```
// set RGBled to Green

DrvGPIO_SetBit(E_GPA,12);
//GPIOA->DOUT |= (1<<12);

// GPA13 = Green, 0 : on, 1 : off
DrvGPIO_ClrBit(E_GPA,13);
//GPIOA->DOUT &= ~(0x1<<13);

DrvGPIO_SetBit(E_GPA,14);
//GPIOA->DOUT |= (1<<14);
DrvSYS_Delay(500000); //delay 500000us=500ms=0.5s
```

3.1Test_RGBled (4/5)

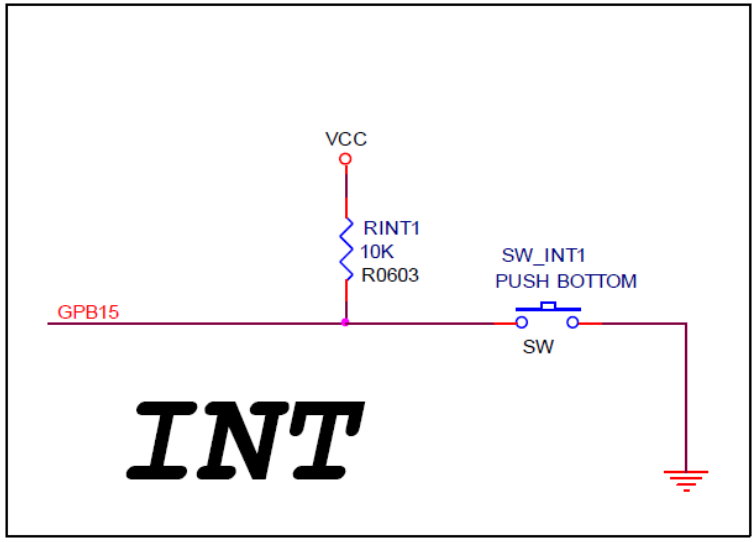
```
// set RGBled to Red

DrvGPIO_SetBit(E_GPA,12);
//GPIOA->DOUT |= (1<<12);

DrvGPIO_SetBit(E_GPA,13);
//GPIOA->DOUT |= (1<<13);

// GPA14 = Red,   0 : on, 1 : off
DrvGPIO_ClrBit(E_GPA,14);
//GPIOA->DOUT &= ~(0x1<<14);

DrvSYS_Delay(500000); //delay 500000us=500ms=0.5s
```

3.1Test_RGBled (5/5)

```
        // set RGBled to off
        DrvGPIO_SetBit(E_GPA,12); // GPA12 = Blue,  0 : on, 1 : off
        //GPIOA->DOUT |= (1<<12);

        DrvGPIO_SetBit(E_GPA,13); // GPA13 = Green, 0 : on, 1 : off
        //GPIOA->DOUT |= (1<<13);

        DrvGPIO_SetBit(E_GPA,14); // GPA14 = Red,   0 : on, 1 : off
        //GPIOA->DOUT |= (1<<14);

        DrvSYS_Delay(500000); //delay 500000us=500ms=0.5s
    }
}
```

3.2Test_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，當按鍵按下時，紅色LED閃爍(on/off各0.5秒) 。鬆開則綠色LED閃爍(on/off各0.5秒)。

GPIOB bit15作為GPIO輸入

檢查GPIOB_PIN[15]=0時，表示按鍵已經按下

DrvGPIO_Open

- 功能：設定GPIO接腳的Input/Output，有四種模式：輸入、輸出、開漏、準雙向。
- 函數：int32_t DrvGPIO_Open(E_DRVGPIO_PORT port, int32_t i32Bit, E_DRVGPIO_IO mode)
- 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.
- 參數： Mode: E_DRVGPIO_IO, set the specified GPIO pin to be E_IO_INPUT(00), E_IO_OUTPUT(01), E_IO_OPENDRAIN(10) or E_IO_QUASI(11) mode.
- 範例：DrvGPIO_Open(E_GPB, 15, E_IO_INPUT); //61states
- 與下列指令的作用相同
- GPIOB->PMD.PMD12 = 0x0; //13states

DrvGPIO_SetBit

- 功能：設定GPIO接腳的值=1。
- 函數： int32_t DrvGPIO_SetBit(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_SetBit(E_GPA,13); //GPIOA_13=1 off, 35states
- 與下列指令的作用相同
- GPIOA->DOUT |= (1<<13); //10states

DrvGPIO_ClrBit

- 功能：設定GPIO接腳的值=0。
- 函數： int32_t DrvGPIO_ClrBit(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_ClrBit(E_GPA,13); //GPIOA_13=0 on, 35states
- 與下列指令的作用相同
- GPIOA->DOUT &= ~(0x1<<13); //10states

DrvSYS_Delay

- 功能：generate the delay time (us)。
- 函數：void DrvSYS_Delay (uint32_t us);
- 參數： **us**: Delay time. The maximal delay time is 335000 us.
- 範例：DrvSYS_Delay (5000); //delay 5000us
- 與下列指令的作用相同
- 說明：DrvSYS_Delay使用SysTick作為計數器，最大可以設定2^{24}=16777215；時脈使用HCLK，若HCLK為22MHz，則數22次為1us。當計數器設定16777215，相當於762600us。也就是說DrvSYS_Delay(762600)是最大的設定值。
- 若HCLK=12MHz，最大值DrvSYS_Delay(1398101) 。
- 若HCLK=50MHz，最大值DrvSYS_Delay(335544) 。

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

3.2Test_SWInt_RGBled (1/4)

```
int main (void)
{
        // Initial GPIO GPA13(blue LED) to output mode
        DrvGPIO_Open(E_GPA, 13, E_IO_OUTPUT);
        //GPIOA->PMD.PMD13 = 0x01;

        // Initial GPIO GPA14(red LED) to output mode
        DrvGPIO_Open(E_GPA, 14, E_IO_OUTPUT);
        //GPIOA->PMD.PMD14 = 0x01;

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

3.2Test_SWInt_RGBled (2/4)

```
while (1)
   {
        if (DrvGPIO_GetBit(E_GPB,15)==0)     // 當(按鍵按下)
        //while((GPIOB->PIN & (1<<15))==0)
        {       //關綠色LED，閃紅色LED
                // turn off green LED, bit13=1
                DrvGPIO_SetBit(E_GPA,13);
                //GPIOA->DOUT |= (1<<13);
                // turn on red LED, bit14=0
                DrvGPIO_ClrBit(E_GPA,14);
                //GPIOA->DOUT &= ~(0x1<<14);
                DrvSYS_Delay(500000);
```

3.2Test_SWInt_RGBled (3/4)

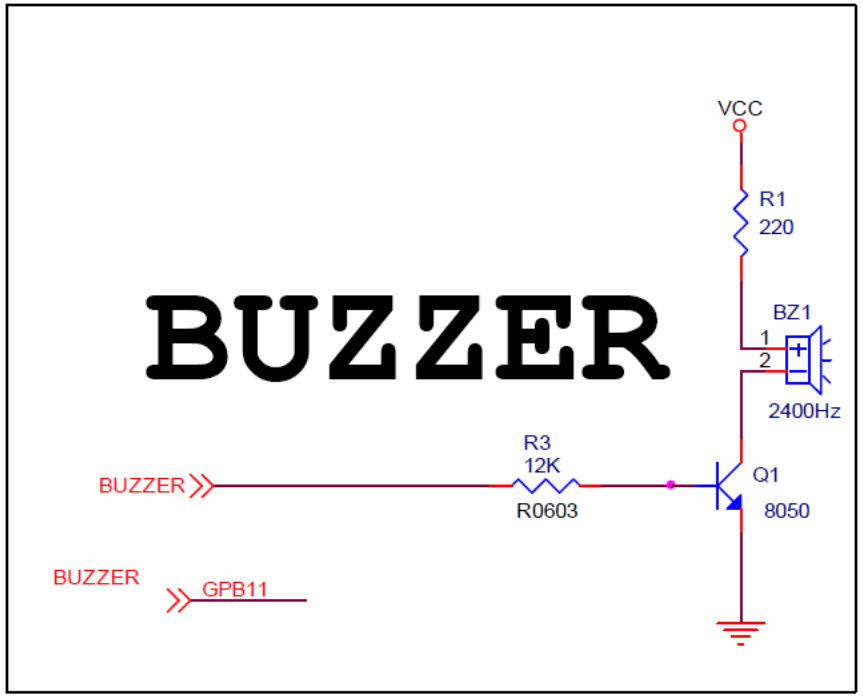
```
            // turn off red LED, bit14=1
            DrvGPIO_SetBit(E_GPA,14);
            //GPIOA->DOUT |= (1<<14);
            DrvSYS_Delay(500000);
    }
    else
    {       // set green flashing
            // turn off red LED, bit14=1
            DrvGPIO_SetBit(E_GPA,14);
            //GPIOA->DOUT |= (1<<14);
            // turn on green LED, bit13=0
```

3.2Test_SWInt_RGBled (4/4)

```
            DrvGPIO_ClrBit(E_GPA,13);       //35states
            //GPIOA->DOUT &= ~(0x1<<13);    //10 states
            DrvSYS_Delay(500000);
            // turn off green LED, bit14=0
            DrvGPIO_SetBit(E_GPA,13);       //35states
            //GPIOA->DOUT |= (1<<13);       //10 states
            DrvSYS_Delay(500000);
             }
      }
}
```

3.3Test_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，蜂鳴器發出聲音。鬆開則沒有聲音。

GPIO pin GPB11=0 turn on BUZZER

GPIO pin GPB11=1 turn off BUZZER

DrvGPIO_Open

- 功能：設定GPIO接腳的Input/Output，有四種模式：輸入、輸出、開漏、準雙向。
- 函數：int32_t DrvGPIO_Open(E_DRVGPIO_PORT port, int32_t i32Bit, E_DRVGPIO_IO mode)
- 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.
- 參數： Mode: E_DRVGPIO_IO, set the specified GPIO pin to be E_IO_INPUT(00), E_IO_OUTPUT(01), E_IO_OPENDRAIN(10) or E_IO_QUASI(11) mode.
- 範例：DrvGPIO_Open(E_GPB, 15, E_IO_INPUT); //61states
- 與下列指令的作用相同
- GPIOB->PMD.PMD12 = 0x0; //13states

DrvGPIO_SetBit

- 功能：設定GPIO接腳的值=1。
- 函數： int32_t DrvGPIO_SetBit(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_SetBit(E_GPB,11); //GPIOB_11=1 off, 35states
- 與下列指令的作用相同
- GPIOB->DOUT |= (1<<11); //10states

DrvGPIO_ClrBit

- 功能：設定GPIO接腳的值=0。
- 函數： int32_t DrvGPIO_ClrBit(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_ClrBit(E_GPB,11); //GPIOB_11=0 on, 35states
- 與下列指令的作用相同
- GPIOB->DOUT &= ~(0x1<<11); //10states

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

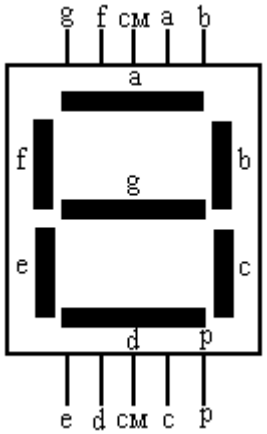
3.3Test_SWInt_Buzzer (1/2)

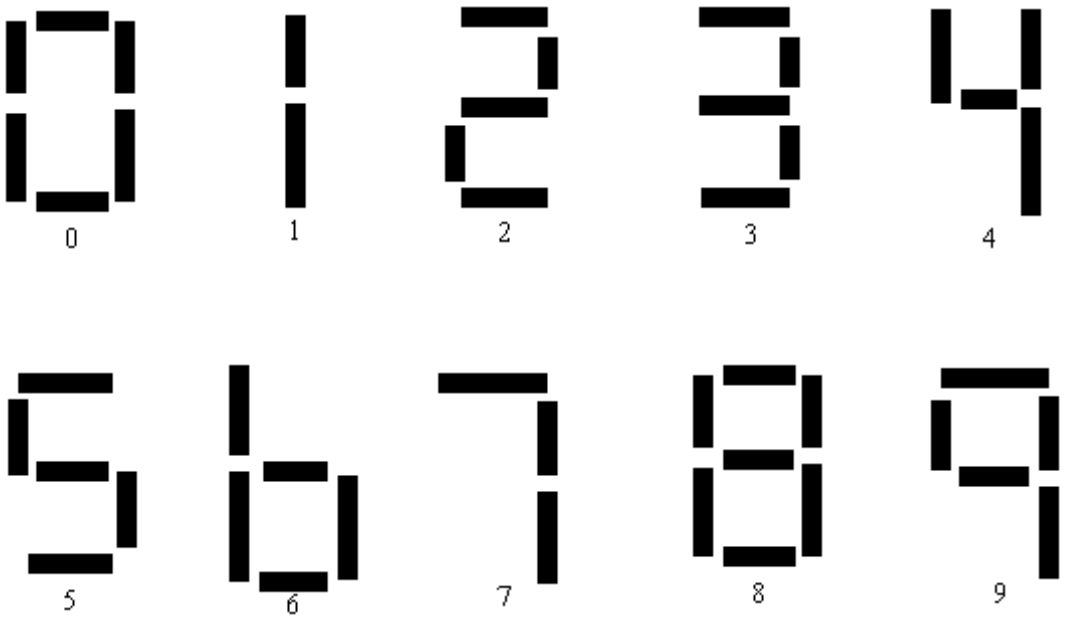
```
int main (void)
{
        // initial GPIO pin GPB15 (SW int ) to input mode
        DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);
        //GPIOB->PMD.PMD15 = 0x00;

        // initial GPIO pin GPB11 for controlling Buzzer to output mode
        DrvGPIO_Open(E_GPB, 11, E_IO_OUTPUT);
        //GPIOB->PMD.PMD11 = 0x01;
```

3.3Test_SWInt_Buzzer (2/2)

```
    while (1)
    {
      if(DrvGPIO_GetBit(E_GPB,15)==0)  // if SW_INT button pressed
      //if((GPIOB->PIN & (1<<15))==0)
            {    // GPB11 = 0, turn on Buzzer
                 DrvGPIO_ClrBit(E_GPB,11);
                 //GPIOB->DOUT &= ~(0x1<<11);
            }  else
            {
                 // GPB11 = 1, turn off Buzzer
                 DrvGPIO_SetBit(E_GPB,11);
                 //GPIOB->DOUT |= (1<<11);
    }
}
```

3.4Test_7seg–七段顯示器





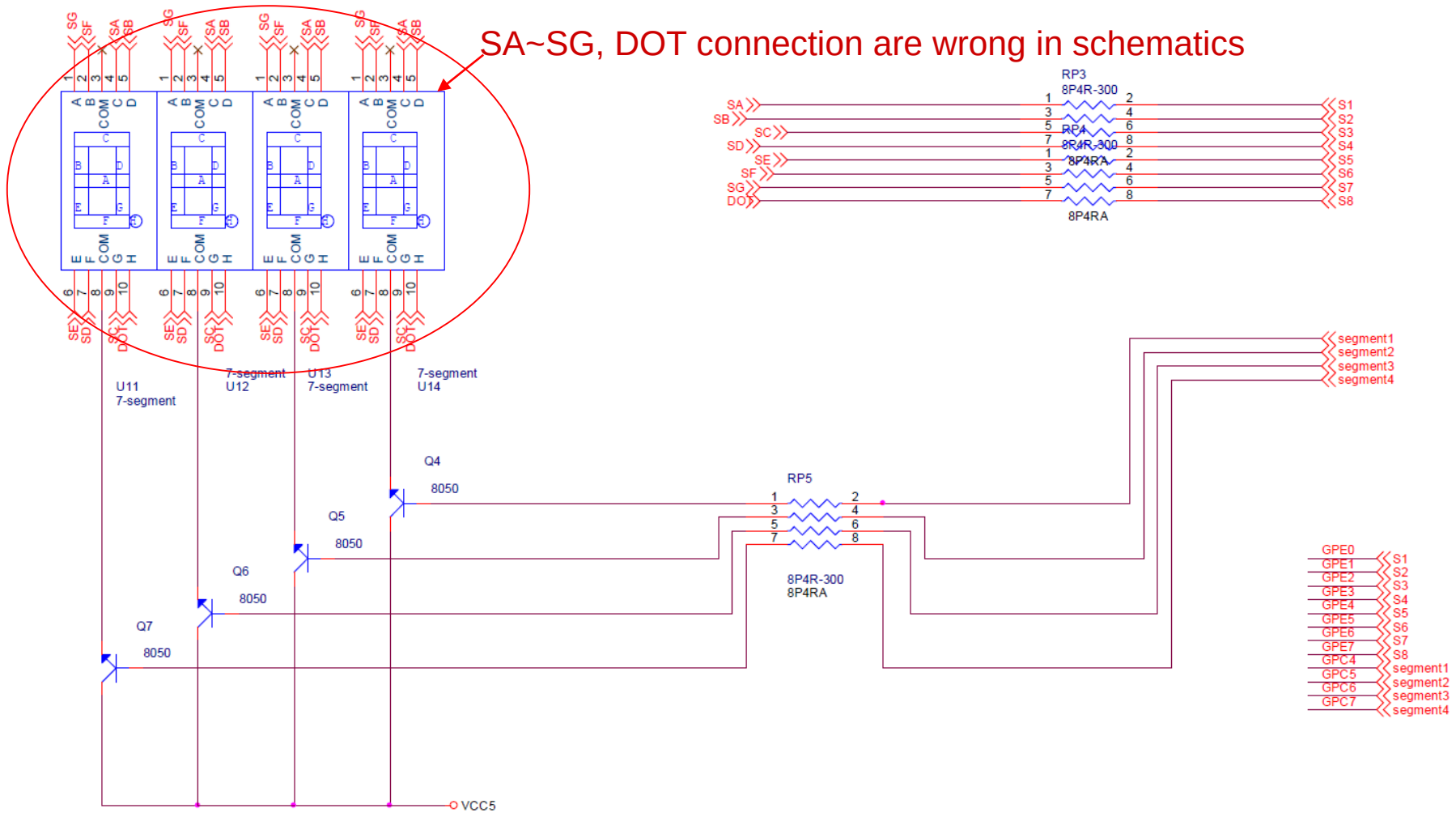
每個7段顯示器有8個LED，分別標示為a,b,c,d,e,f,g,p。

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。

寫一程式，逐一顯示0000-9999。

Test_7seg– 7 段顯示器線路圖

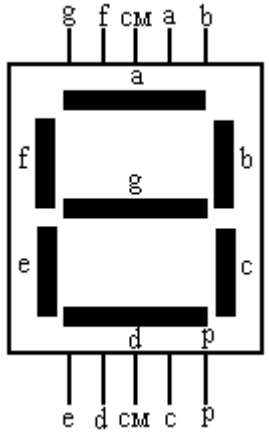


Test_7seg– Control Pins used for

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);

seven_segment_open

- 功能：設定7段顯示器的LED為輸出。
- 函數：void seven_segment_open(void)
- 參數：
- 範例： seven_segment_open();
- 與下列指令的作用相同

```
//Initial GPIOE [7:0] to input mode for 7-seg.
for (ith=0; ith<8; ith++)
  { DrvGPIO_Open(E_GPE, ith, E_IO_OUTPUT); }
//Initial GPIOC [7:4] to output mode for nth 7-seg.
for (ith=4; ith<8; ith++)
  { DrvGPIO_Open(E_GPC, ith, E_IO_OUTPUT); }
```

3.4Test_7seg– main() (1/x)

```
int32_t main (void)
{
    int32_t i;

    seven_segment_open();
    //GPIOE->PMD.PMD0 = 1; GPIOE[7:0]=output
    //GPIOE->PMD.PMD1 = 1; GPIOE[7:0]=output
    //GPIOC->PMD>PMD4 = 1; GPIOC[7-4]=output

    for (i=0; i<10000; i++)          // counting 0~9999
    {
       seg_display(i);       // display value to  7-segment display
    }
}
```

3.4Test_7seg– seg_display()　　(2/x)

```
void seg_display(int16_t value)
{
    int8_t digit[4];
    int32_t itimes=50;

    //將顯示在7段顯示器的值先算出來，存在陣列
    digit[3] = value / 1000;
    value = value - digit[3] * 1000;
    digit[2] = value / 100;
    value = value - digit[2] * 100;
    digit[1] = value / 10;
    value = value - digit[1] * 10;
    digit[0] = value;
```

3.4Test_7seg– seg_display() (3/x)

```
while(itimes--){
    //turn off 7-seg., clear GPIOC bit4-7
    close_seven_segment();
    //GPIOC->DOUT &= ~(0xF<<4);

    show_seven_segment(3,digit);
    //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
     DrvSYS_Delay(5000); //delay 5000us=5ms
```

3.4Test_7seg– seg_display() (4/x)

```
            close_seven_segment(); //clear GPIOC bit4-7
            show_seven_segment(2,digit[2]);
             DrvSYS_Delay(5000); //delay 5000us=5ms

            close_seven_segment(); //clear GPIOC bit4-7
            show_seven_segment(1,digit[1]);
             DrvSYS_Delay(5000); //delay 5000us=5ms

            close_seven_segment(); //clear GPIOC bit4-7
            show_seven_segment(0,digit[0]);
             DrvSYS_Delay(5000); //delay 5000us=5ms
    }
}
```

3.5Test_7seg_Keypad–學習板電路圖

Control Pins used for 3x3 Keypad

Column control : GPA2, 1, 0

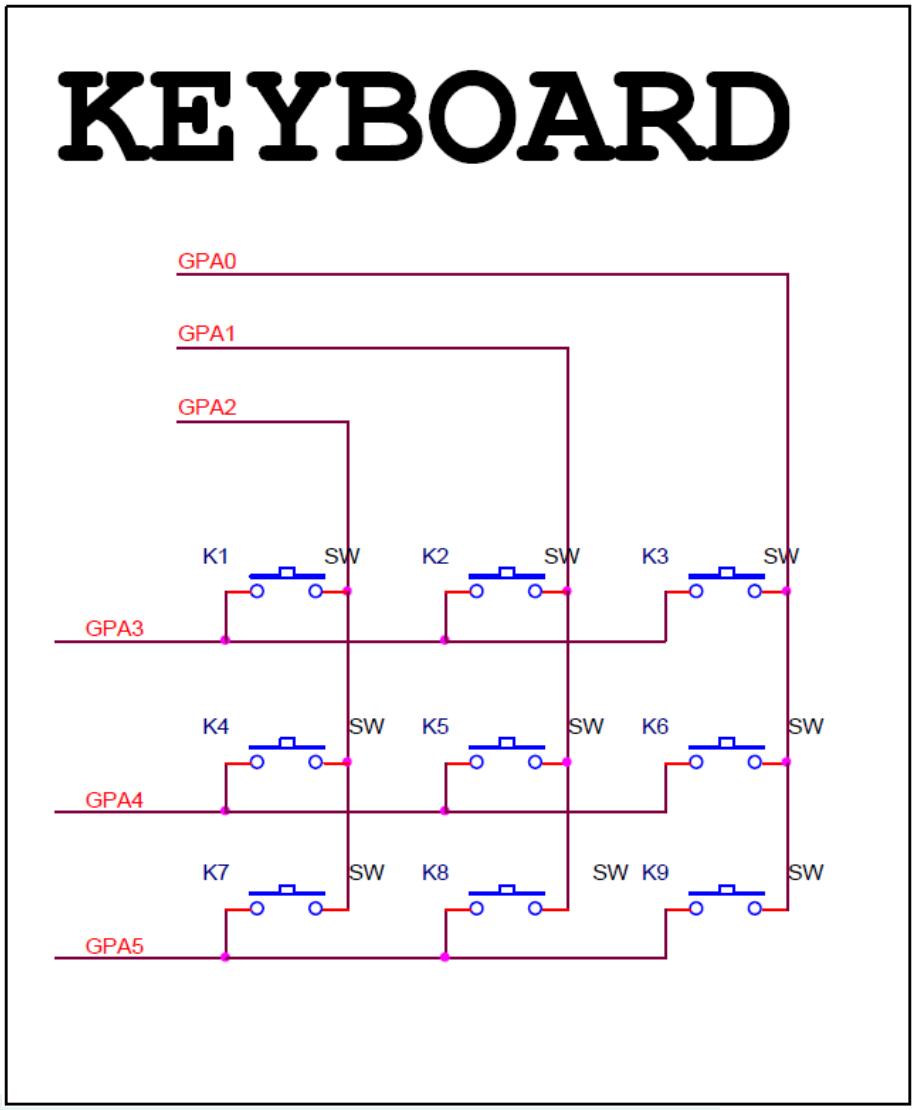
Raw control : GPA 3, 4, 5

- 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，顯示對應的數字0-9。

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

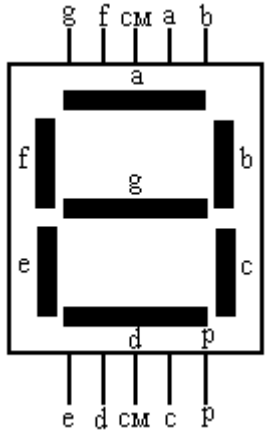


Test_7seg_Keypad– 7-seg. Control Pins

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()
- 範例：OpenKeyPad ()
- 與下列指令的作用相同
- GPIOA->PMD.PMD0 = 0x03;
- GPIOA->PMD.PMD1 = 0x03;
- GPIOA->PMD.PMD2 = 0x03;
- GPIOA->PMD.PMD3 = 0x03;
- GPIOA->PMD.PMD4 = 0x03;
- GPIOA->PMD.PMD5 = 0x03;

ScanKeyDriver - Scankey

- 功能：讀取按鍵對應的值1-9
- 函數： void ScanKey (void)
- 範例：ScanKey ()
-

ScanKeyDriver - CloseKeyPad

- 功能：將ScanKey的控制GPIOA[5:0]設定為雙向
- 函數： void CloseKeyPad ()
- 範例：CloseKeyPad()
- 與下列指令的作用相同
- GPIOA->PMD.PMD0 = 0x03;
- GPIOA->PMD.PMD1 = 0x03;
- GPIOA->PMD.PMD2 = 0x03;
- GPIOA->PMD.PMD3 = 0x03;
- GPIOA->PMD.PMD4 = 0x03;
- GPIOA->PMD.PMD5 = 0x03;

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;

Scankey_GetKey

- 功能：讀取按鍵對應的值1-9
- 函數： void ScanKey_GetKey (void)
- 範例：ScanKey_GetKey ()
- 與下列指令的作用相同

```
number = 0;
for (irow=1; irow <4; irow++)
{
        GPIOA->DOUT |= (0x7);          //xxxx-x111
        GPIOA->DOUT &= ~(1<<(3-irow));  //011, 101, 110
        if((GPIOA->PIN & (0x1<<3)) == 0)number = irow;
        if((GPIOA->PIN & (0x1<<4)) == 0)number = irow+3;
        if((GPIOA->PIN & (0x1<<5)) == 0)number = irow+6;
}
return number;
```

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);

seven_segment_open

- 功能：設定7段顯示器的LED為輸出。
- 函數：void seven_segment_open(void)
- 參數：
- 範例： seven_segment_open();
- 與下列指令的作用相同
- GPIOE->PMD.PMD0 = 1;
- …
- GPIOE->PMD.PMD7 = 1;
- GPIOC->PMD.PMD4 = 1;
- …
- GPIOC->PMD.PMD7 = 1;

3.5Test_7seg_Keypad– (1/x)

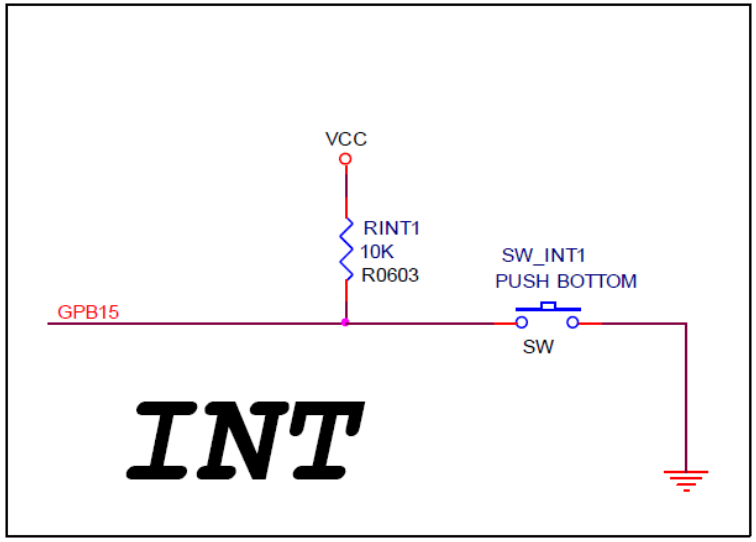
```
int32_t main (void)
{
    //Initial GPIOC [7:4] to output mode
    seven_segment_open();  //set GPIOC_PMD[7:4] //743 states

    close_seven_segment();  //tuen off 7-seg. GPIOC[7:4] //157 states
    //GPIOC->DOUT &= ~(0xF<<4);

    //Initial GPIOA [5:0]=11,QUASI mode
    OpenKeyPad(); //383 states
    //GPIOA->PMD.PMD0 = (0x3);
    //GPIOA->PMD.PMD1 = (0x3);
    //GPIOA->PMD.PMD2 = (0x3);
    //GPIOA->PMD.PMD3 = (0x3);
    //GPIOA->PMD.PMD4 = (0x3);
    //GPIOA->PMD.PMD5 = (0x3);
```

3.5Test_7seg_Keypad– (2/x)

```
while(1)
    {
        number = Scankey(); //7380 states

        if(number)show_seven_segment(0,number);//448 states
        //GPIOE->DOUT &= ~(0xFF); //clear 7 seg., GPIOE bit0-7=0000000
        //GPIOE->DOUT |= SEG_BUF[number];//show 7 seg.,GPIOE bit0-7=
        //GPIOC->DOUT |= (1<<(0+4));     //display 1st 7 seg.,GPIOC bit 7=1

        //Delay(5000);                  //30013 states
    }

}
```

3.6Test_7seg_SWInt_Bounce–計算彈跳的次數

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

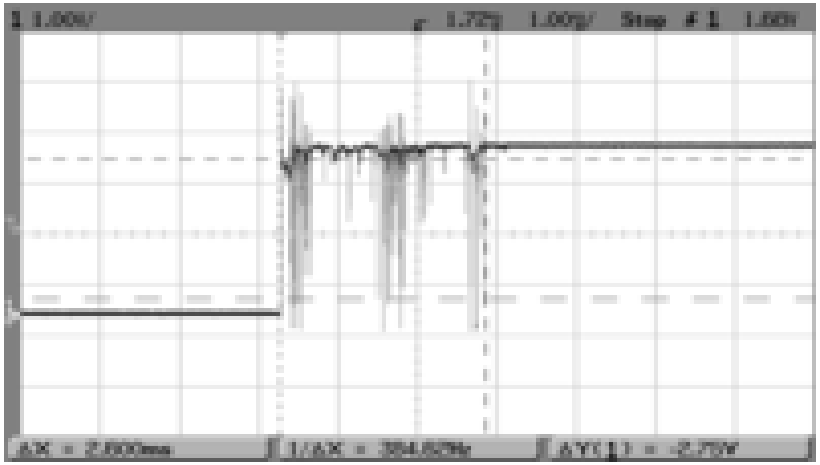


寫一程式，當按鍵按下時，計算彈跳(bounce)的次數。然後將次數顯示在7段顯示器。

設定為GPIO，檢查GPIOB_DOUT[15]=0時，表示按鍵已經按下

3.6Test_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。



- 按下時，會有彈跳發生，通常在10ms內結束。一般延遲20ms。
- 鬆開時，也會有彈跳現象。一般延遲10ms就足夠處理。
- 按鍵主要的接觸時間約60ms-150ms，或更長。這是較難處理的地方。

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

3.6Test_7seg_SWInt_Bounce–計算彈跳的次數

- 軟體處理1：
- 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。若是低電位，表示一個有效的按鍵。
- 等待按鍵鬆開，出現高電位，延遲10ms。若是低電位，表示雜訊。

- 軟體處理2：
- 使用TIMER計時，每5ms測量一次按鍵。
- 若連續出現4次低電位，表示按鍵已經按下。
- 等待連續出現4次高電位， 表示按鍵已經鬆開。

- 軟體處理3：
- 使用TIMER計時，每5ms測量一次按鍵。
- 若連續出現2次低電位，間格2次，再出現2次高電位，表示已經完成一次按鍵。缺點是必須等按鍵鬆開(可能很久)，才能判斷出來。

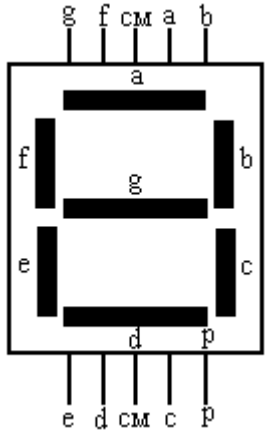
http://www.embedded.com/electronics-blogs/break-points/4024944/The-secret-life-of-switches

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

seven_segment_open

- 功能：設定7段顯示器的LED為輸出。
- 函數：void seven_segment_open(void)
- 參數：
- 範例： seven_segment_open();
- 與下列指令的作用相同
- GPIOE->PMD.PMD0 = 1;
- …
- GPIOE->PMD.PMD7 = 1;
- GPIOC->PMD.PMD4 = 1;
- …
- GPIOC->PMD.PMD7 = 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-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

3.6Test_7seg_SWInt_Bounce (1/x)

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

    //: SYST_RVR=SysTick reload value, max=2^24=16M
    SysTick->LOAD = 10000 * CyclesPerUs; //: 10000us=10ms
    //: SYST_CVR=SysTick current value=0 always
    SysTick->VAL  = 0;                          //: (core_cm0.h) always be 0
    //: enable SysTick and clock source,
    SysTick->CTRL = (1<<2) | (1); //: (core_cm0.h)

     seven_segment_open(); //set GPIO pin output for 7-seg. LED
    close_seven_segment(); //clear GPIOC[7:0]

    // Initial GPIO GPB15(SW Int) to input mode
    DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);
```

3.6Test_7seg_SWInt_Bounce (2/x)

```
while (1)
{
    //set SysTick 10000us=10ms
    SysTick->LOAD = 10000 * CyclesPerUs;
    // wait for key pressed
    while(DrvGPIO_GetBit(E_GPB,15)!=0) ;
    // press first time
    value=1;
    // wait for key released
    while(DrvGPIO_GetBit(E_GPB,15)==0)
```

3.6Test_7seg_SWInt_Bounce (3/x)

```
while (1)
{       //: reset SysTick time=0
        SysTick->VAL  = (0x00);
        //wait key not pressed and not time limit
        while(((GPIOB->PIN & (1<<15))!=0)  && (!(SysTick->CTRL
& (1 << 16))));
        //if time limited, the  key did not press
        if((GPIOB->PIN & (1<<15))==0)
        {           // key pressed, increase times
                    value++;
                    //wait for key release
                    while((GPIOB->PIN & (1<<15))==0);
        } else   // time limit, break
        {           break;
        }
}
```

3.6Test_7seg_SWInt_Bounce (4/x)

```
            show_seven_segment(0,value);
            DrvSYS_Delay(700000);
    //700000us=700ms=0.7sec.,nax=762600
            close_seven_segment(); //clear GPIOC bit4-7
    }
}
```

3.6Test_7seg_SWInt_Bounce (5/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);
}
```

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.*

數據類型相關的編程風格

- 必須用typedef顯式標識出各數據類型的長度和符號特性，避免直接使用標準數據類型。
- 一個32位的整數係統，可定義如下：
- typedef char chat_t；
- typedef sigrled char int8_t；
- typedef signed short intl6_t；
- typedef signed int int32_t；
- typedef signed long int64_t；
- typedef unsitgned chat uint8_t；
- typedef unsigned short uint16_t;
- typedef unsigned int uint32_t;
- typedef unsigned 1ong uint64_t；