中斷
Interrupt

- 2013/3/5

課程大綱 (Lesson Outline)

- **Cortex-M0 MCU 中斷設計原理**
  - 中斷向量(Interrupt Vector)
- **實習範例 :**每次SW按下時，LED依序循環發亮。
  - 程式
- **範例練習:**
  - 利用範例修改
- **範例練習:**
  - 利用範例修改

Nested Vectored Interrupt Controller (NVIC)

- Cortex-M0 提供中斷控制器，用於整體管理異常，稱之為「嵌套向量中斷控制器 (NVIC)」。
- NVIC 結構支持32(IRQ[31:0]) 個離散中斷，每個中斷可以支持 4 級離散中斷優先級。
- 當接受任何中斷時，ISR的開始地址可從內存的向量表中取得。當取得開始地址時，NVIC 將自動保存處理狀態到堆疊中，包括以下寄存器「PC, PSR, LR, R0~R3, R12」 的值。在ISR結束時，NVIC 將從堆疊中恢複相關寄存器的值。
- NVIC 支持末尾連鎖 (Tail Chaining)，有效處理背對背中斷(back-to-back interrupts)，即無需保存和恢復當前狀態從而減少在切換當前ISR時的延遲時間。NVIC 還支持遲到(Late Arrival)，改善同時發生的ISR的效率。

異常模式和系統中斷映射

- NuMicro NUC100 系列支持表5-2 所列的異常模式。與所有中斷一樣，軟件可以對其中一些中斷設置4級優先級。最高優先級為「0」，最低優先級為「3」，所有用戶可配置的優先級的默認值為「0」。

異常名稱	向量編號	優先級
Reset	1	-3
NMI	2	-2
Hard Fault	3	-1
Reserved	4 ~ 10	保留
SVCall	11	可配置
Reserved	12 ~ 13	保留
PendSV	14	可配置
SysTick	15	可配置
Interrupt (IRQ0 ~ IRQ31)	16 ~ 47	可配置

系統中斷映射

向量號	中斷號	中斷名稱	源 IP	中斷描述
16	0	BOD_OUT	Brown-Out	欠壓檢測中斷
17	1	WDT_INT	WDT	看門狗定時器中斷
18	2	EINT0	GPIO	PB.14 管腳上的外部信號中斷
19	3	EINT1	GPIO	PB.15 管腳上的外部信號中斷
20	4	GPAB_INT	GPIO	PA[15:0]/PB[13:0] 的外部信號中斷
21	5	GPCDE_INT	GPIO	PC[15:0]/PD[15:0]/PE[15:0] 的外部信號中斷
22	6	PWMA_INT	PWM0~3	PWM0, PWM1, PWM2 與 PWM3 中斷
23	7	PWMB_INT	PWM4~7	PWM4, PWM5, PWM6 與 PWM7 中斷

系統中斷映射

向量號	中斷號	中斷名稱	源 IP	中斷描述
24	8	TMR0_INT	TMR0	Timer 0 中斷
25	9	TMR1_INT	TMR1	Timer 1 中斷
26	10	TMR2_INT	TMR2	Timer 2 中斷
27	11	TMR3_INT	TMR3	Timer 3 中斷
28	12	UART02_INT	UART0/2	UART0 與 UART2 中斷
29	13	UART1_INT	UART1	UART1 中斷
30	14	SPI0_INT	SPI0	SPI0 中斷
31	15	SPI1_INT	SPI1	SPI1 中斷

系統中斷映射

向量號	中斷號	中斷名稱	源 IP	中斷描述
32	16	SPI2_INT	SPI2	SPI2 中斷
33	17	SPI3_INT	SPI3	SPI3 中斷
34	18	I2C0_INT	I2C0	I2C0 中斷
35	19	I2C1_INT	I2C1	I2C1 中斷
36	20	CAN0_INT	CAN0	CAN0 中斷
37	21	Reserved	Reserved	保留
38	22	Reserved	Reserved	保留
39	23	USB_INT	USBD	USB 2.0 FS 設備中斷

系統中斷映射

向量號	中斷號	中斷名稱	源 IP	中斷描述
40	24	PS2_INT	PS/2	PS/2 中斷
41	25	ACMP_INT	ACMP	模擬比較器-0 或 模擬比較器-1 中斷
42	26	PDMA_INT	PDMA	PDMA 中斷
43	27	I2S_INT	I2S	I2S 中斷
44	28	PWRWU_INT	CLKC	從掉電狀態喚醒的時鐘控制器中斷
45	29	ADC_INT	ADC	ADC 中斷
46	30	Reserved	Reserved	保留
47	31	RTC_INT	RTC	RTC 中斷

NVIC 控制寄存器

寄存器	偏移量	描述	復位後的值
NVIC_ISER	SCS_BA+0x100	IRQ0 ~ IRQ31 設置致能控制寄存器	0x0000_0000
NVIC_ICER	SCS_BA+0x180	IRQ0 ~ IRQ31 清致能控制寄存器	0x0000_0000
NVIC_ISPR	SCS_BA+0x200	IRQ0 ~ IRQ31 設置掛起控制寄存器	0x0000_0000
NVIC_ICPR	SCS_BA+0x280	IRQ0 ~ IRQ31 清掛起控制寄存器	0x0000_0000
NVIC_IPR0	SCS_BA+0x400	IRQ0 ~ IRQ3 優先級控制寄存器	0x0000_0000
NVIC_IPR1	SCS_BA+0x404	IRQ4 ~ IRQ7 優先級控制寄存器	0x0000_0000
NVIC_IPR2	SCS_BA+0x408	IRQ8 ~ IRQ11 優先級控制寄存器	0x0000_0000
NVIC_IPR3	SCS_BA+0x40C	IRQ12 ~ IRQ15 優先級控制寄存器	0x0000_0000
NVIC_IPR4	SCS_BA+0x410	IRQ16 ~ IRQ19 優先級控制寄存器	0x0000_0000
NVIC_IPR5	SCS_BA+0x414	IRQ20 ~ IRQ23 優先級控制寄存器	0x0000_0000
NVIC_IPR6	SCS_BA+0x418	IRQ24 ~ IRQ27 優先級控制寄存器	0x0000_0000
NVIC_IPR7	SCS_BA+0x41C	IRQ28 ~ IRQ31 優先級控制寄存器	0x0000_0000

IRQ0 ~ IRQ31 Set-Enable Control Register (NVIC_ISER)

Bits	符號	描述
[31:0]	**SETENA**	致能一個或者多個中斷，每位元代表 IRQ0 ~ IRQ31 的中斷號（向量號：從16到47）。 寫 1致能相關中斷 寫 0 無效 讀取該寄存器返回當前致能狀態。

IRQ0 ~ IRQ31 Clear-Enable Control Register (NVIC_ICER)

Bits	符號	描述
[31:0]	**CLRENA**	禁用一個或者多個中斷，每位元代表 IRQ0 ~ IRQ31 的中斷號（向量號：從16到 47）。 寫 1 禁用相應中斷 寫 0 無效 讀取該寄存器返回當前致能狀態。

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 Clear-Pending Control Register (NVIC_ICPR)

Bits	符號	描述
[31:0]	**CLRPEND**	寫 1 清除，由軟件控制清除等待處理的中斷，每位代表 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 ~ IRQ3 Interrupt Priority Register (NVIC_IPR0)

Bits	符號	描述
[31:30]	**PRI_3**	IRQ3 優先級, **EINT1** 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_2**	IRQ2 優先級, **EINT0** 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_1**	IRQ1 優先級, **WDT_INT** 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_0**	IRQ0 優先級, **BOD_OUT** 「0」表示最高優先級 &「3」表示最低優先級

IRQ4 ~ IRQ7 Interrupt Priority Register (NVIC_IPR1)

Bits	符號	描述
[31:30]	**PRI_7**	IRQ7 優先級, **PWMB_INT** 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_6**	IRQ6 優先級, **PWMA_INT** 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_5**	IRQ5 優先級, **GPCDE_INT** 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_4**	IRQ4 優先級, **GPAB_INT** 「0」表示最高優先級 &「3」表示最低優先級

IRQ8 ~ IRQ11 Interrupt Priority Register (NVIC_IPR2)

Bits	符號	描述
[31:30]	**PRI_11**	IRQ11 優先級, **TMR3_INT** 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_10**	IRQ10 優先級, **TMR2_INT** 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_9**	IRQ9 優先級, **TMR1_INT** 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_8**	IRQ8 優先級, **TMR0_INT** 「0」表示最高優先級 &「3」表示最低優先級

IRQ12 ~ IRQ15 Interrupt Priority Register (NVIC_IPR3)

Bits	符號	描述
[31:30]	**PRI_15**	IRQ15 優先級, **SPI1_INT** 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_14**	IRQ14 優先級, **SPI0_INT** 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_13**	IRQ13 優先級, **UART1_INT** 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_12**	IRQ12 優先級, **UART02_INT** 「0」表示最高優先級 &「3」表示最低優先級

IRQ16 ~ IRQ19 Interrupt Priority Register (NVIC_IPR4)

Bits	符號	描述
[31:30]	**PRI_19**	IRQ19 優先級 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_18**	IRQ18 優先級 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_17**	IRQ17 優先級 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_16**	IRQ16 優先級 「0」表示最高優先級 &「3」表示最低優先級

IRQ20 ~ IRQ23 Interrupt Priority Register (NVIC_IPR5)

Bits	符號	描述
[31:30]	**PRI_23**	IRQ23 優先級 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_22**	IRQ22 優先級 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_21**	IRQ21 優先級 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_20**	IRQ20 優先級 「0」表示最高優先級 &「3」表示最低優先級

IRQ24~ IRQ27 Interrupt Priority Register (NVIC_IPR6)

Bits	符號	描述
[31:30]	**PRI_27**	IRQ27 優先級 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_26**	IRQ26 優先級 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_25**	IRQ25 優先級 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_24**	IRQ24 優先級 「0」表示最高優先級 &「3」表示最低優先級

IRQ28 ~ IRQ31 Interrupt Priority Register (NVIC_IPR7)

Bits	符號	描述
[31:30]	**PRI_31**	IRQ31 優先級 「0」表示最高優先級 &「3」表示最低優先級
[23:22]	**PRI_30**	IRQ30 優先級 「0」表示最高優先級 &「3」表示最低優先級
[15:14]	**PRI_29**	IRQ29 優先級 「0」表示最高優先級 &「3」表示最低優先級
[7:6]	**PRI_28**	IRQ28 優先級 「0」表示最高優先級 &「3」表示最低優先級

Interrupt Source Identity Register (IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ_SRC	0	Bit2: 0 Bit1: 0 Bit0: BOD_INT
[2:0]	IRQ_SRC	1	Bit2: 0 Bit1: 0 Bit0: WDT_INT
[2:0]	IRQ_SRC	2	Bit2: 0 Bit1: 0 Bit0: EINT0 – PB.14 上的外部中斷 0
[2:0]	IRQ_SRC	3	Bit2: 0 Bit1: 0 Bit0: EINT1 – PB.15 上的外部中斷 1

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ4_SRC	4	Bit2: 0 Bit1: GPB_INT Bit0: GPA_INT
[2:0]	IRQ5_SRC	5	Bit2: GPE_INT Bit1: GPD_INT Bit0: GPC_INT
[3:0]	IRQ6_SRC	6	Bit3: PWM3_INT Bit2: PWM2_INT Bit1: PWM1_INT Bit0: PWM0_INT
[3:0]	IRQ7_SRC	7	Bit3: PWM7_INT Bit2: PWM6_INT Bit1: PWM5_INT Bit0: PWM4_INT

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ8_SRC	8	Bit2: 0 Bit1: 0 Bit0: TMR0_INT
[2:0]	IRQ9_SRC	9	Bit2: 0 Bit1: 0 Bit0: TMR1_INT
[2:0]	IRQ10_SRC	10	Bit2: 0 Bit1: 0 Bit0: TMR2_INT
[2:0]	IRQ11_SRC	11	Bit2: 0 Bit1: 0 Bit0: TMR3_INT

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ12_SRC	12	Bit2: 0 Bit1: 0 Bit0: URT0_INT
[2:0]	IRQ13_SRC	13	Bit2: 0 Bit1: 0 Bit0: URT1_INT
[2:0]	IRQ14_SRC	14	Bit2: 0 Bit1: 0 Bit0: SPI0_INT
[2:0]	IRQ15_SRC	15	Bit2: 0 Bit1: 0 Bit0: SPI1_INT

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ16_SRC	16	Bit2: 0 Bit1: 0 Bit0: SPI2_INT
[2:0]	IRQ17_SRC	17	Bit2: 0 Bit1: 0 Bit0: SPI3_INT
[2:0]	IRQ18_SRC	18	Bit2: 0 Bit1: 0 Bit0: I2C0_INT
[2:0]	IRQ19_SRC	19	Bit2: 0 Bit1: 0 Bit0: I2C1_INT

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ20_SRC	20	Bit2: 0 Bit1: 0 Bit0: CAN0_INT
[2:0]	IRQ21_SRC	21	保留
[2:0]	IRQ22_SRC	22	保留
[2:0]	IRQ23_SRC	23	Bit2: 0 Bit1: 0 Bit0: USB_INT

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ24_SRC	24	Bit2: 0 Bit1: 0 Bit0: PS2_INT
[2:0]	IRQ25_SRC	25	Bit2: 0 Bit1: 0 Bit0: ACMP_INT
[2:0]	IRQ26_SRC	26	Bit2: 0 Bit1: 0 Bit0: PDMA_INT
[2:0]	IRQ27_SRC	27	Bit2: 0 Bit1: 0 Bit0: I2S_INT

Interrupt Source Identity Register(IRQn_SRC)

Bits	暫存器	INT-Num	描述
[2:0]	IRQ28_SRC	28	Bit2: 0 Bit1: 0 Bit0: PWRWU_INT
[2:0]	IRQ29_SRC	29	Bit2: 0 Bit1: 0 Bit0: ADC_INT
[2:0]	IRQ30_SRC	30	保留
[2:0]	IRQ31_SRC	31	Bit2: 0 Bit1: 0 Bit0: RTC_INT

NMI Interrupt Source Select Control Register (NMI_SEL)

Bits	符號	描述
[8]	NMI_EN	NMI 中斷致能 1 = 允許 NMI 中斷 0 = 禁止 NMI 中斷
[4:0]	NMI_SEL	NMI 中斷源選擇 通過設置 NMI_SEL 可以在週邊設備中斷中選擇 Cortex-M0 的 NMI 中斷。

MCU Interrupt Request Source Register (MCU_IRQ)

Bits	符號	描述
[31:0]	MCU_IRQ	MCU IRQ 來源寄存器 MCU_IRQ 從週邊設備收集所有中斷，並向 Cortex-M0 內核產生同步中斷，產生此中斷的模式有兩種，分別是正常模式和測試模式。 MCU_IRQ 從每個週邊設備收集所有中斷和同步這些中斷，然後觸發 Cortex-M0 中斷。 MCU_IRQ[n] 為 0 時：置 MCU_IRQ[n] 為 1，Cortex_M0 NVIC[n] 將產生一個中斷。. MCU_IRQ[n] 為 1 時（意味著有中斷請求）：置 MCU_IRQ[n] 為 1，將清除中斷；置 MCU_IRQ[n] 為 0 則無效。

(core_cm0.h) NVIC_EnableIRQ(IRQn_t IRQn)

- Enable a device specific interupt in the NVIC interrupt controller.

```
static __INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[0] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* enable interrupt */
}
```

(core_cm0.h) NVIC_DisableIRQ(IRQn_t IRQn)

- Disable the interrupt line for external interrupt specified

```
static __INLINE void NVIC_DisableIRQ(IRQn_Type IRQn)
{
\  NVIC->ICER[0] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* disable interrupt */
}
```

(core_cm0.h) NVIC_GetPendingIRQ (IRQn_t IRQn)

- Read the interrupt pending bit for a device specific interrupt source

```
static __INLINE uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)
{
  return((uint32_t) ((NVIC->ISPR[0] & (1 << ((uint32_t)(IRQn) & 0x1F)))?1:0)); /* Return 1 if pending else 0 */
}
```

0	BOD_IRQn	8	TMR0_IRQn	16	SPI2_IRQn	24	PS2_IRQn
1	WDT_IRQn	9	TMR1_IRQn	17	SPI3_IRQn	25	ACMP_IRQn
2	EINT0_IRQn	10	TMR2_IRQn	18	I2C0_IRQn	26	PDMA_IRQn
3	EINT1_IRQn	11	TMR3_IRQn	19	I2C1_IRQn	27	I2S_IRQn
4	GPAB_IRQn	12	UART0_IRQn	20	CAN0_IRQn	28	PWRWU_IRQn
5	GPCDE_IRQn	13	UART1_IRQn	21	CAN1_IRQn	29	ADC_IRQn
6	PWMA_IRQn	14	SPI0_IRQn	22	SD_IRQn	30	DAC_IRQn
7	PWMB_IRQn	15	SPI1_IRQn	23	USBD_IRQn	31	RTC_IRQn

(core_cm0.h) NVIC_SetPendingIRQ (IRQn_t IRQn)

- Set the pending bit for the specified interrupt.

```
static __INLINE void NVIC_SetPendingIRQ(IRQn_Type IRQn)
{
  NVIC->ISPR[0] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* set interrupt pending */
}
```

0	BOD_IRQn	8	TMR0_IRQn	16	SPI2_IRQn	24	PS2_IRQn
1	WDT_IRQn	9	TMR1_IRQn	17	SPI3_IRQn	25	ACMP_IRQn
2	EINT0_IRQn	10	TMR2_IRQn	18	I2C0_IRQn	26	PDMA_IRQn
3	EINT1_IRQn	11	TMR3_IRQn	19	I2C1_IRQn	27	I2S_IRQn
4	GPAB_IRQn	12	UART0_IRQn	20	CAN0_IRQn	28	PWRWU_IRQn
5	GPCDE_IRQn	13	UART1_IRQn	21	CAN1_IRQn	29	ADC_IRQn
6	PWMA_IRQn	14	SPI0_IRQn	22	SD_IRQn	30	DAC_IRQn
7	PWMB_IRQn	15	SPI1_IRQn	23	USBD_IRQn	31	RTC_IRQn

(core_cm0.h) NVIC_ClearPendingIRQ (IRQn_t IRQn)

- Clear the pending bit for an external interrupt

```
static __INLINE void NVIC_ClearPendingIRQ(IRQn_Type IRQn)
{
  NVIC->ICPR[0] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* Clear pending interrupt */
}
```

0	BOD_IRQn	8	TMR0_IRQn	16	SPI2_IRQn	24	PS2_IRQn
1	WDT_IRQn	9	TMR1_IRQn	17	SPI3_IRQn	25	ACMP_IRQn
2	EINT0_IRQn	10	TMR2_IRQn	18	I2C0_IRQn	26	PDMA_IRQn
3	EINT1_IRQn	11	TMR3_IRQn	19	I2C1_IRQn	27	I2S_IRQn
4	GPAB_IRQn	12	UART0_IRQn	20	CAN0_IRQn	28	PWRWU_IRQn
5	GPCDE_IRQn	13	UART1_IRQn	21	CAN1_IRQn	29	ADC_IRQn
6	PWMA_IRQn	14	SPI0_IRQn	22	SD_IRQn	30	DAC_IRQn
7	PWMB_IRQn	15	SPI1_IRQn	23	USBD_IRQn	31	RTC_IRQn

(core_cm0.h) d NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)

- Set the priority for the specified interrupt.

```
static __INLINE void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
{
  if(IRQn < 0) {
    SCB->SHP[_SHP_IDX(IRQn)] = (SCB->SHP[_SHP_IDX(IRQn)] &
~(0xFF << _BIT_SHIFT(IRQn))) |
        (((priority << (8 - __NVIC_PRIO_BITS)) & 0xFF) <<
_BIT_SHIFT(IRQn)); }
  else {
    NVIC->IPR[_IP_IDX(IRQn)] = (NVIC->IPR[_IP_IDX(IRQn)] & ~(0xFF
<< _BIT_SHIFT(IRQn))) |
        (((priority << (8 - __NVIC_PRIO_BITS)) & 0xFF) <<
_BIT_SHIFT(IRQn)); }
}
```

(core_cm0.h) NVIC_GetPriority (IRQn_t IRQn)a

- Read the priority for the specified interrupt.

```
static __INLINE uint32_t NVIC_GetPriority(IRQn_Type IRQn)
{


  if(IRQn < 0) {
    return((uint32_t)((SCB->SHP[_SHP_IDX(IRQn)] >> _BIT_SHIFT(IRQn)
) >> (8 - __NVIC_PRIO_BITS)));  } /* get priority for Cortex-M0 system
interrupts */
  else {
    return((uint32_t)((NVIC->IPR[_IP_IDX(IRQn)] >> _BIT_SHIFT(IRQn) )
>> (8 - __NVIC_PRIO_BITS)));  } /* get priority for device specific
interrupts  */
}
```

(core_cm0.h) uint32_t SysTick_Config(uint32_t ticks)

- Initialise the system tick timer and its interrupt and start the system tick timer / counter in free running mode to generate periodical interrupts.

```
static __INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if (ticks > SysTick_LOAD_RELOAD_Msk)  return (1);
  SysTick->LOAD  = (ticks & SysTick_LOAD_RELOAD_Msk) - 1;
  NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);
  SysTick->VAL   = 0;
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;
  return (0);
}
```

(core_cm0.h) void NVIC_SystemReset (void)

- Initiate a system reset request to reset the MCU

```
static __INLINE void NVIC_SystemReset(void)
{
  SCB->AIRCR  = ((0x5FA << SCB_AIRCR_VECTKEY_Pos)      |
                 SCB_AIRCR_SYSRESETREQ_Msk);
  __DSB();                          /* Ensure completion of memory access */
  while(1);                         /* wait until reset */
}
```

中斷向量編號 (Interrupt Vector Number)

- defined in startup_NUC1xx.h
- NVIC_EnableIRQ(WDT_IRQn);

```
typedef enum IRQn
{
//******  Cortex-M0 Processor Exceptions Numbers
//NonMaskableInt_IRQn        = -14,    /*!< 2 Non Maskable Interrupt
//HardFault_IRQn             = -13,    /*!< 3 Cortex-M0 Hard Fault Interrupt
//SVCall_IRQn                = -5,     /*!< 11 Cortex-M0 SV Call Interrupt
//PendSV_IRQn                = -2,     /*!< 14 Cortex-M0 Pend SV Interrupt
//SysTick_IRQn               = -1,     /*!< 15 Cortex-M0 System Tick Interrupt
//********************* NUC1xx Interrupt Numbers ********************
BOD_IRQn                   = 0,
WDT_IRQn                   = 1,
…
RTC_IRQn                   = 31
} IRQn_Type;
```

中斷處理常式 (Interrupt Handler)

```
__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler
```

中斷處理常式 (Interrupt Handler)

```
; External Interrupts
; maximum of 32 External Interrupts are possible
    DCD    BOD_IRQHandler
    DCD    WDT_IRQHandler
    DCD    EINT0_IRQHandler
    DCD    EINT1_IRQHandler
    DCD    GPAB_IRQHandler
    DCD    GPCDE_IRQHandler
    DCD    PWMA_IRQHandler
    DCD    PWMB_IRQHandler
    DCD    TMR0_IRQHandler
    DCD    TMR1_IRQHandler
    DCD    TMR2_IRQHandler
    DCD    TMR3_IRQHandler
    DCD    UART02_IRQHandler
    DCD    UART1_IRQHandler
    DCD    SPI0_IRQHandler
    DCD    SPI1_IRQHandler
    DCD    SPI2_IRQHandler
    DCD    SPI3_IRQHandler
    DCD    I2C0_IRQHandler
    DCD    I2C1_IRQHandler
    DCD    CAN0_IRQHandler
    DCD    Default_Handler
    DCD    Default_Handler
    DCD    USBD_IRQHandler
    DCD    PS2_IRQHandler
    DCD    ACMP_IRQHandler
    DCD    PDMA_IRQHandler
    DCD    I2S_IRQHandler
    DCD    PWRWU_IRQHandler
    DCD    ADC_IRQHandler
    DCD    Default_Handler
    DCD    RTC_IRQHandler
```

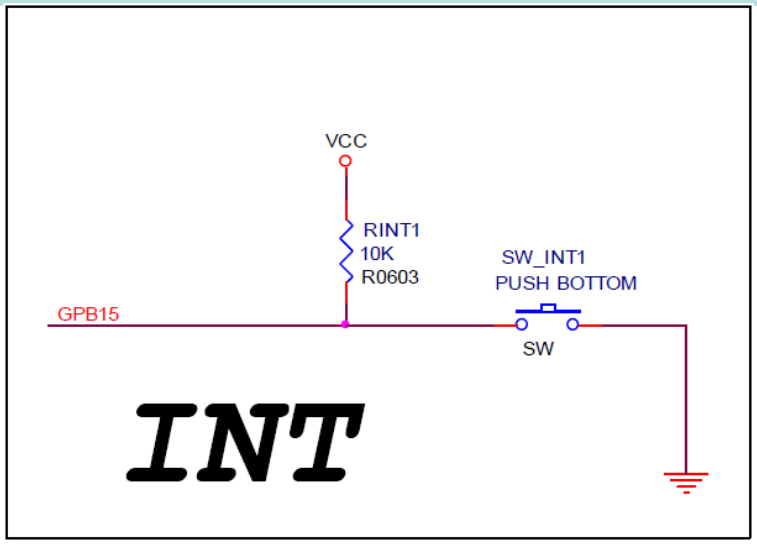
Interrupt Basics

- Pending: if the trigger condition has set the flag but the interrupt service routine has not been called yet, which can happen if the main program has disabled interrupts or another interrupt service routine is running.
- Mask: Every interrupt also has a mask bit, which enable or disables that individual interrupt.
- Global Interrupt: enable/disable all interrupts
- **Interrupt Configuration Steps**

  1. Configure The Peripheral
  2. Reset Interrupt Flag
  3. Enable Interrupt
  4. Enable Global Interrupt,

4.2Test_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，當按鍵按下時，顯示藍色LED。按鍵再按下時，顯示綠色LED。按鍵再按下時，顯示紅色LED。每次按鍵按下時，LED依序循環發亮。

GPIOB[15]作為輸入，可以設定為外部中斷EINT1

設定為INT1，啟用中斷，藉由中斷程式來執行。

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_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。

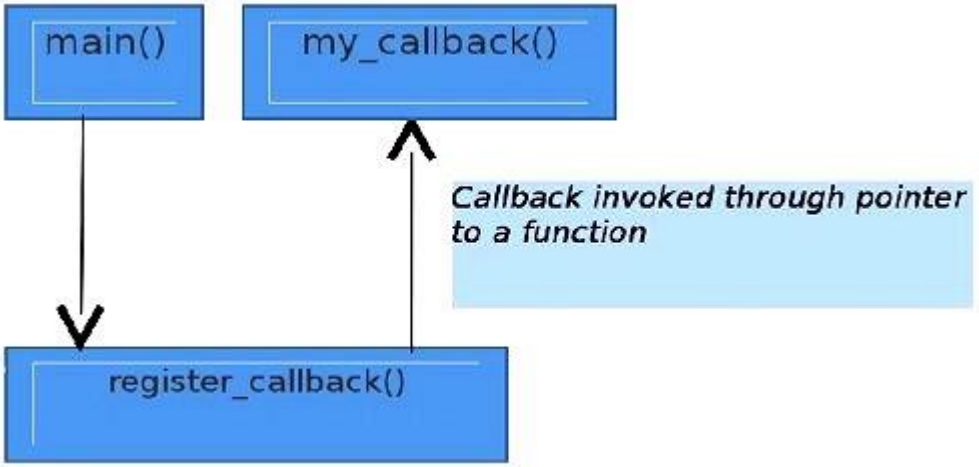
EINT1的中斷服務程式

- 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)
- {
- ….
- }

Call back function

The higher layer function calls a lower layer function as a normal call and the callback mechanism allows the lower layer function to call the higher layer function through a pointer to a callback function.



- Callback functions can also be used to create a library that will be called from an upper-layer program, and in turn, the library will call user-defined code on the occurrence of some event.
- Interrupt -> ISRs -> callback. why?
- To have more control and flexibility if ISRs are pre-built.

4.2Test_Interrupt_SWInt_RGBled (1/x)

```
#include <stdio.h>
#include "NUC1xx.h"
#include "Driver\DrvGPIO.h"
#include "Driver\DrvSYS.h"

    int8_t led_n=14;
```

4.2Test_Interrupt_SWInt_RGBled (2/x)

```
int main (void)
{
    // Initial GPIO GPA12,13,14 to output mode for RGB LED
    DrvGPIO_Open(E_GPA, 12, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPA, 13, E_IO_OUTPUT);
    DrvGPIO_Open(E_GPA, 14, E_IO_OUTPUT);

    //set GPIOB[15] to input mode for SW_EINT1
    DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);

    //Enable the debounce function for SW_EINT1
    DrvGPIO_EnableDebounce(E_GPB, 15);
```

4.2Test_Interrupt_SWInt_RGBled (3/x)

```
    //Set the interrupt debounce sampling time & clock source
    //The target debounce time is (2^4)*(1/(10*1000)) s = 16*0.1 ms
    //The system will sampling interrupt input once per 1.6 ms.
    DrvGPIO_SetDebounceTime(4,E_DBCLKSRC_10K);

    /* Configure external interrupt */
    DrvGPIO_EnableEINT1(E_IO_FALLING, E_MODE_EDGE,
    EINT1Callback);

    /* Waiting for interrupts */
   while(1);
}
```

4.2Test_Interrupt_SWInt_RGBled (4/x)

```
void EINT1Callback(void)
{
    // turn off lighted LED
    DrvGPIO_SetBit(E_GPA,led_n);
    //GPIOA->DOUT |= (1<<led_n);

    // next LED
    led_n++;
    if(led_n > 14)led_n=12;  //(GPA 12,13,14)

    // turn on next LED
    DrvGPIO_ClrBit(E_GPA,led_n);
    //GPIOA->DOUT &= ~(0x1<<led_n);
}
```

4.5Test_interrupt_7seg_Keypad–

Control Pins used for 3x3 Keypad

Column control : GPA2, 1, 0

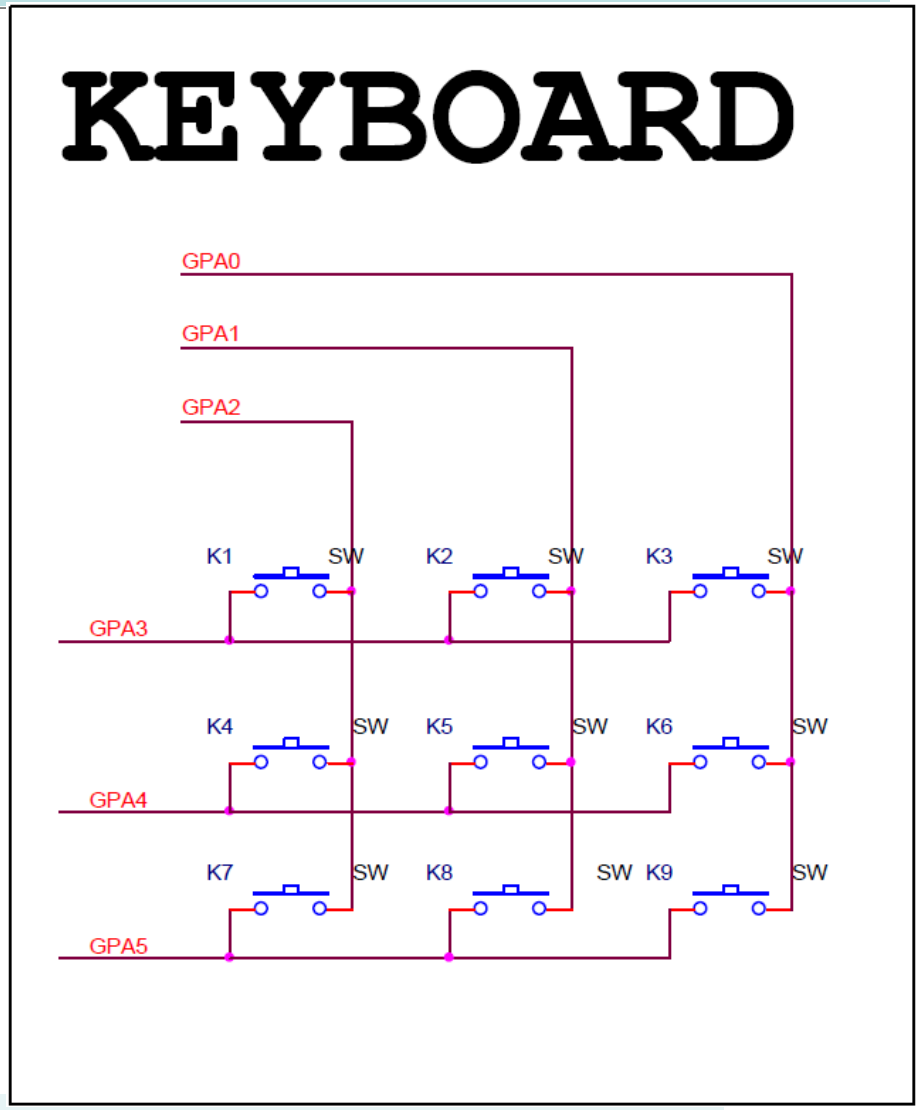
Raw control : GPA 3, 4, 5

- 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，在7段顯示器顯示對應的數字1-9。

GPA[2:0]依序輸出低電位，011,010,110,011,…

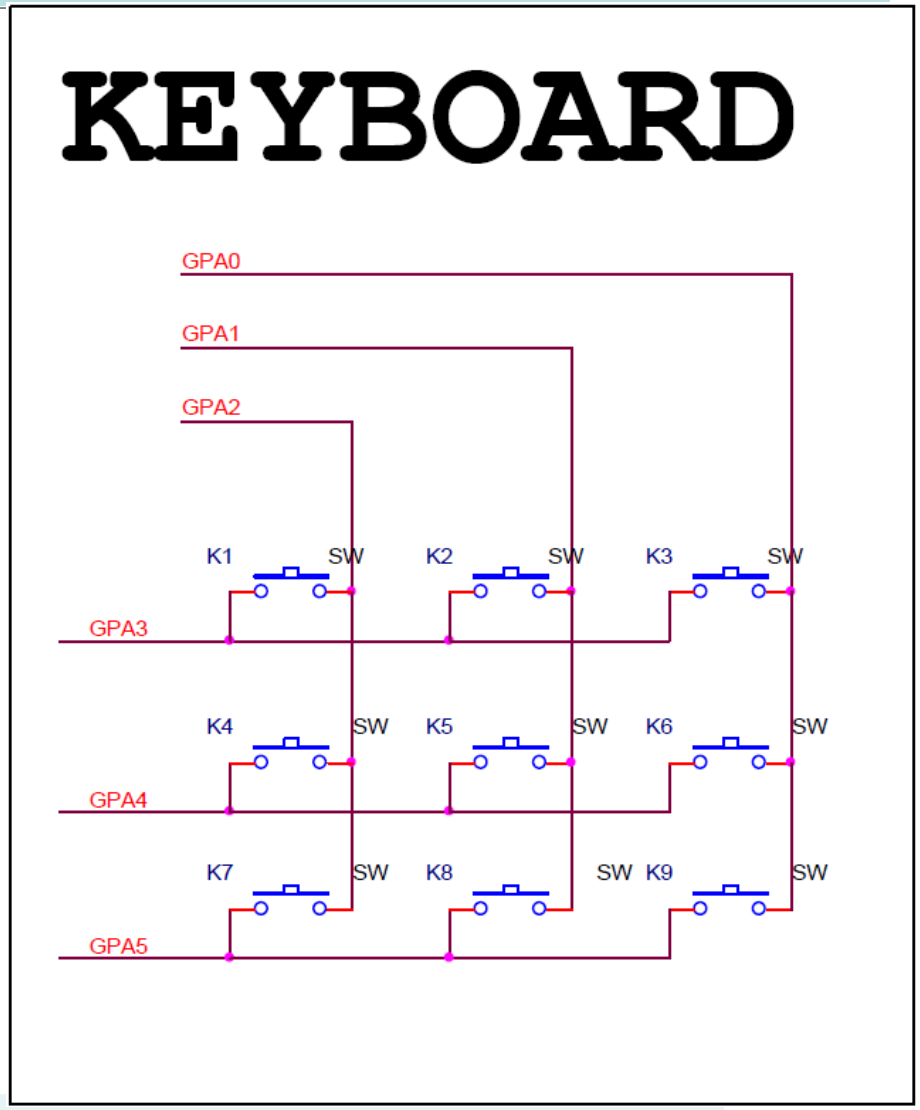
GPA[5:3]使用中斷處理按鍵



4.5Test_interrupt_7seg_Keypad–

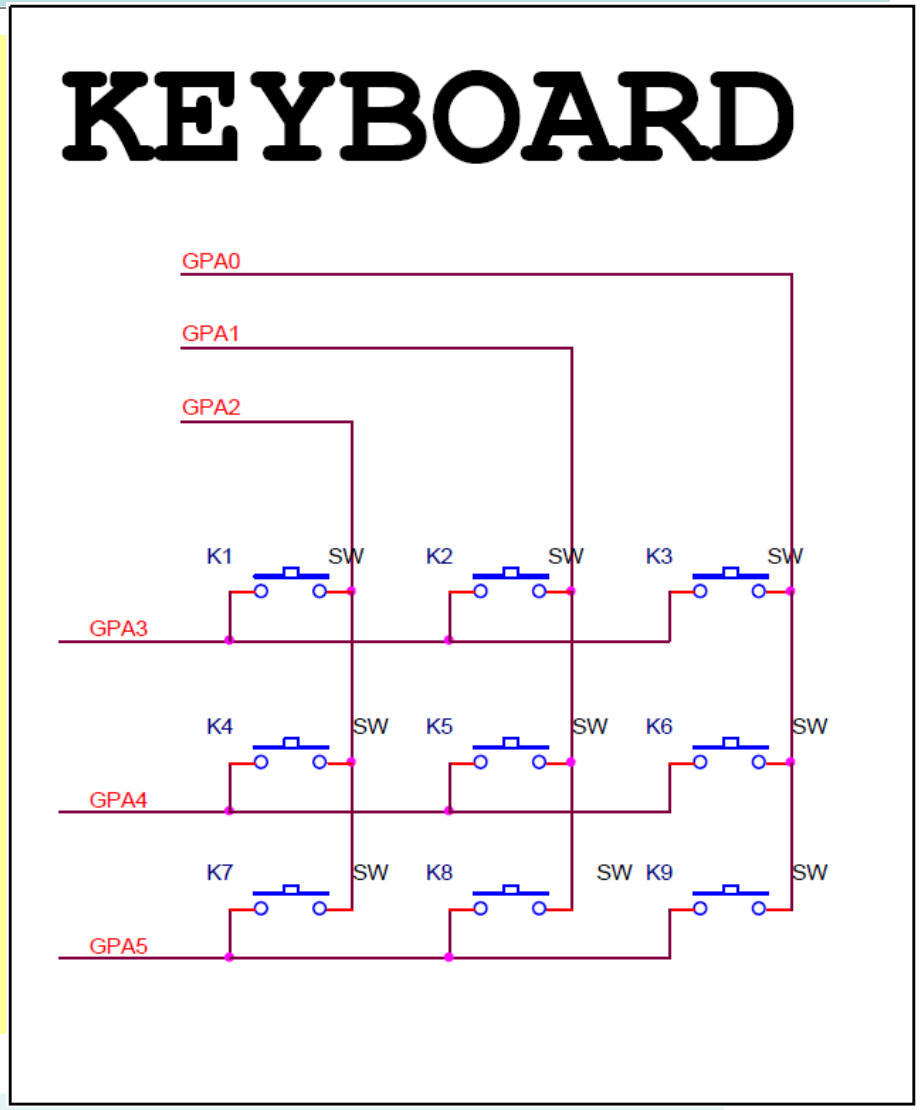
GPA	2,	1,	0
GPA 3	1	2	3
GPA 4	4	5	6
GPA 5	7	8	9

GPA	5	4	3	2	1	0
1	1	1	0	0	1	1
2	1	1	0	1	0	1
3	1	1	0	1	1	0
4	1	0	1	0	1	1
5	1	0	1	1	0	1
6	1	0	1	1	1	0
7	0	1	1	0	1	1
8	0	1	1	1	0	1
9	0	1	1	1	1	0



鍵盤掃描程式

- GPA[2:0]先輸出011，即column GPA2為低電位
- GPA[5:3]讀取資料，若GPA3=0，表示按下的是1；若GPA4=0，表示按下的是4；若GPA5=0，表示按下的是7；
- 然後GPA[2:0]先輸出101，即column GPA1為低電位，根據讀取的值，判斷按鍵是2,5,7
- 使用中斷程式判斷按鍵的位置
- 離開中斷服務程式前，用NVIC_ClearPendingIRQ清除IRQ信號，避免馬上又進入同一中斷服務程式。

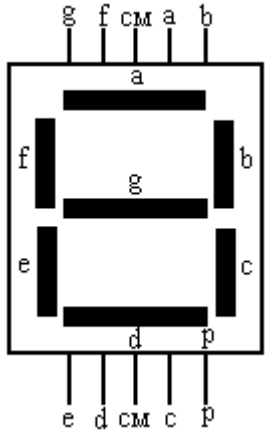


Test_interrupt_7seg_Keypad–

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

使用GPIO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
- GPIOA->IEN |= (1<<3)　　//fall edge, low level
- GPIOA->IEN |= (1<<(3+16))　　//rise edge, high level
- 2.設定觸發模式：
- GPIOA->IMD &= ~(1<<3);　　//=0, edge trigger
- GPIOA->IMD |= (1<<3);　　//=1, level trigger
- 3.設定優先權
- NVIC->IPR[1] = (2ul<<6);　　//11=最高, 00=最低
- 4.設定IRQ4中斷允許
- NVIC->ISER[0]=(1<<4);
- 上述步驟，可以用函數直接設定：
- DrvGPIO_EnableInt(E_GPA, 3, E_IO_FALLING, E_MODE_EDGE); //GPIOA[3]，下降邊緣，邊緣觸發

DrvGPIO_EnableINT()

功能： Enable the interrupt function for GPIO pin。
函數： int32_t DrvGPIO_EnableInt(E_DRVGPIO_PORT port, int32_t i32Bit, E_DRVGPIO_INT_TYPE TriggerType, E_DRVGPIO_INT_MODE Mode)
參數：port: (E_GPA, E_GPB, E_GPC, E_GPD and E_GPE)
參數：i32Bit: (0-31)
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).
範例： DrvGPIO_EnableInt(E_GPA, 3, E_IO_FALLING, E_MODE_EDGE);
說明：當GPIOA[3]的接腳為下降邊緣時，執行GPAB_IRQHandler。根據DrvGPIO_SetIntCallback設定的callback函數，再呼叫callback函數。

設定GPAB和GPCDE的callback函數

- 設定GPIO A,B和GPIO C,D,E的callback函數
- void DrvGPIO_SetIntCallback(GPIO_GPAB_CALLBACK pfGPABCallback, GPIO_GPCDE_CALLBACK pfGPCDECallback)
- {
- _pfGPABCallback = (void (*)(uint32_t, uint32_t))pfGPABCallback;
- _pfGPCDECallback = (void (*)(uint32_t, uint32_t, uint32_t))pfGPCDECallback;
- }

- GPIO A,B產生中斷時，執行GPAB_IRQHandler，再呼叫GPABCallback函數，其中必須撰寫中斷的處理程式。
- GPIO C,D,E產生中斷時，執行GPCDE_IRQHandler，再呼叫GPCDECallback函數，其中必須撰寫中斷的處理程式。

GPAB的中斷服務程式

- 在DrvGPIO.c有提供中斷服務程式
- void GPAB_IRQHandler(void)
- {
- volatile uint32_t u32GPAStatus, u32GPBStatus;
- /* Keep the interrupt source */
- u32GPAStatus = GPIOA->ISRC;
- u32GPBStatus = GPIOB->ISRC;
- /* Avoid to clear EINT0/EINT1 INT flag */
- u32GPBStatus = u32GPBStatus & ~(0x3 << 14);
- /* Clear the interrupt */
- GPIOA->ISRC = u32GPAStatus;
- GPIOB->ISRC = u32GPBStatus;

GPAB的中斷服務程式

```
/* Call the callback function of GPIOAB interrupt */
    if ( _pfGPABCallback )
        _pfGPABCallback(u32GPAStatus, u32GPBStatus);
}
```

callback函數必須撰寫中斷的處理程式

```
void GPABCallback(uint32_t u32GPAStatus, uint32_t u32GPBStatus)
{
        ….
}
```

GPCDE的中斷服務程式

- 在DrvGPIO.c有提供中斷服務程式
- void GPCDE_IRQHandler(void)
- {
- volatile uint32_t u32GPCStatus, u32GPDStatus, u32GPEStatus;
- /* Keep the interrupt source */
- u32GPCStatus = GPIOC->ISRC;
- u32GPDStatus = GPIOD->ISRC;
- u32GPEStatus = GPIOE->ISRC;
- /* Clear the interrupt */
- GPIOC->ISRC = u32GPCStatus;
- GPIOD->ISRC = u32GPDStatus;
- GPIOE->ISRC = u32GPEStatus;

GPCDE的中斷服務程式

```
/* Call the callback function of GPIOAB interrupt */
   if ( _pfGPCDECallback )
       _pfGPCDECallback(u32GPCStatus, u32GPDStatus, u32GPEStatus);
}
```

- callback函數必須撰寫中斷的處理程式

```
void GPCDECallback(uint32_t u32GPCStatus, uint32_t u32GPDStatus,
uint32_t u32GPEStatus)
{
        ….
}
```

(core_cm0.h) NVIC_ClearPendingIRQ (IRQn_t IRQn)

- Clear the pending bit for an external interrupt

```
static __INLINE void NVIC_ClearPendingIRQ(IRQn_Type IRQn)
{
  NVIC->ICPR[0] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* Clear pending interrupt */
}
```

0	BOD_IRQn	8	TMR0_IRQn	16	SPI2_IRQn	24	PS2_IRQn
1	WDT_IRQn	9	TMR1_IRQn	17	SPI3_IRQn	25	ACMP_IRQn
2	EINT0_IRQn	10	TMR2_IRQn	18	I2C0_IRQn	26	PDMA_IRQn
3	EINT1_IRQn	11	TMR3_IRQn	19	I2C1_IRQn	27	I2S_IRQn
4	GPAB_IRQn	12	UART0_IRQn	20	CAN0_IRQn	28	PWRWU_IRQn
5	GPCDE_IRQn	13	UART1_IRQn	21	CAN1_IRQn	29	ADC_IRQn
6	PWMA_IRQn	14	SPI0_IRQn	22	SD_IRQn	30	DAC_IRQn
7	PWMB_IRQn	15	SPI1_IRQn	23	USBD_IRQn	31	RTC_IRQn

4.5Test_interrupt_7seg_Keypad–

```
int32_t main (void)
{
    int8_t icolumn;
    //Initial 7-seg.
    seven_segment_open();//set GPIOC_PMD[7:4]
    close_seven_segment();//(Seven_Segment.c)turn off 7-seg.

    //Initial KeyPad
    OpenKeyPad(); //(ScanKey.c)

    //set GPIOA[5:3] interrupt
    DrvGPIO_EnableInt(E_GPA, 3, E_IO_FALLING, E_MODE_EDGE);
    DrvGPIO_EnableInt(E_GPA, 4, E_IO_FALLING, E_MODE_EDGE);
    DrvGPIO_EnableInt(E_GPA, 5, E_IO_FALLING, E_MODE_EDGE);
    // set GPIOAB callback function
    DrvGPIO_SetIntCallback(GPABCallback, GPCDECallback) ;
```

4.5Test_interrupt_7seg_Keypad–

```
    while(1)
    {
        for (icolumn=1; icolumn <4; icolumn++)
        {
            //turn off scan keypad
            GPIOA->DOUT |= (0x7);        //xxxx-x111
            //turn on scan  keypad for ith column
            GPIOA->DOUT &= ~(1<<(3-icolumn));//011, 101, 110
        }
    }

}
```

4.5seven_segment_open(void) –

```
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);
}
```

4.5GPABCallback() – GPIO A & B Routine

```
void GPABCallback(uint32_t u32GPAStatus, uint32_t u32GPBStatus)
{
    int8_t number;
    // check key from keypad
    switch ((GPIOA->DOUT & 0x07) | (~u32GPAStatus & 0x38))
     {
          case 0x33: number=1; break;
          case 0x35: number=2; break;
          case 0x36: number=3; break;
          case 0x2B: number=4; break;
          case 0x2D: number=5; break;
          case 0x2E: number=6; break;
          case 0x1B: number=7; break;
          case 0x1D: number=8; break;
          case 0x1E: number=9; break;
          default: number=0;
     }
```

4.5GPABCallback() – GPIO A & B Routine

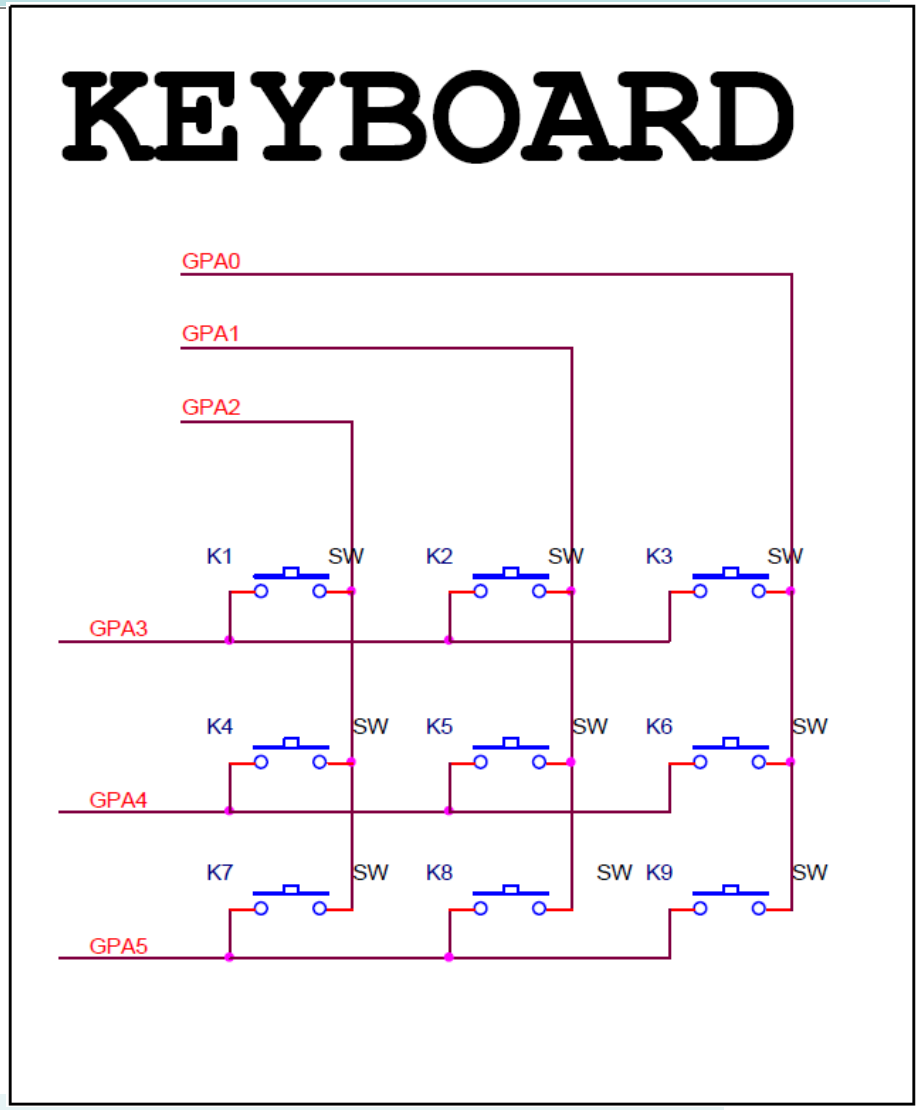
```
    // display number on 7-seg.
    show_seven_segment(0,number);//(Seven_Segment.c)

    // clear pending IRQ4
    NVIC_ClearPendingIRQ(GPAB_IRQn); //IRQ4
}
```

問題：

1. 當離開中斷服務程式時，按鍵可能還沒有鬆開，會再次進入中斷服務程式。到鬆開為止，可能會產生多次的中斷。

4.5b Test_interrupt_7seg_Keypad

- 範例4.5必須在main()不斷的送出鍵盤掃描信號，不僅增加CPU的執行時間，也增加程式的複雜度。
- 如果在掃描信號同時送出0(不用變化)，當任何一個鍵按下時，都會產生中斷，再送出掃描信號判斷是哪一個按鍵。
- 不僅省下掃描時間，程式集中管理，也較容易。



4.5b Test_interrupt_7seg_Keypad–

```
int32_t main (void)
{
    int8_t icolumn;
    //Initial 7-seg.
    seven_segment_open();//set GPIOC_PMD[7:4]
    close_seven_segment();//(Seven_Segment.c)turn off 7-seg.

    //Initial KeyPad
    OpenKeyPad(); //(ScanKey.c)

    //set GPIOA[5:3] interrupt
    DrvGPIO_EnableInt(E_GPA, 3, E_IO_FALLING, E_MODE_EDGE);
    DrvGPIO_EnableInt(E_GPA, 4, E_IO_FALLING, E_MODE_EDGE);
    DrvGPIO_EnableInt(E_GPA, 5, E_IO_FALLING, E_MODE_EDGE);
    // set GPIOAB callback function
    DrvGPIO_SetIntCallback(GPABCallback, GPCDECallback) ;
```

4.5b Test_interrupt_7seg_Keypad–

```
    while(1)
    {
        for (icolumn=1; icolumn <4; icolumn++)
        {
            //turn off scan keypad
            GPIOA->DOUT |= (0x7);       //xxxx-x111
            //turn on scan_keypad for ith column
            GPIOA->DOUT &= ~(1<<(3-icolumn));//011, 101, 110
        }
    }

}
```

4.5b seven_segment_open(void) –

```
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);
}
```

4.5b GPABCallback() – GPIO A & B Routine

```
void GPABCallback(uint32_t u32GPAStatus, uint32_t u32GPBStatus)
{
    int8_t number;
    // check key from keypad
    switch ((GPIOA->DOUT & 0x07) | (~u32GPAStatus & 0x38))
     {
          case 0x33: number=1; break;
          case 0x35: number=2; break;
          case 0x36: number=3; break;
          case 0x2B: number=4; break;
          case 0x2D: number=5; break;
          case 0x2E: number=6; break;
          case 0x1B: number=7; break;
          case 0x1D: number=8; break;
          case 0x1E: number=9; break;
          default: number=0;
     }
```

4.5b GPABCallback() – GPIO A & B Routine

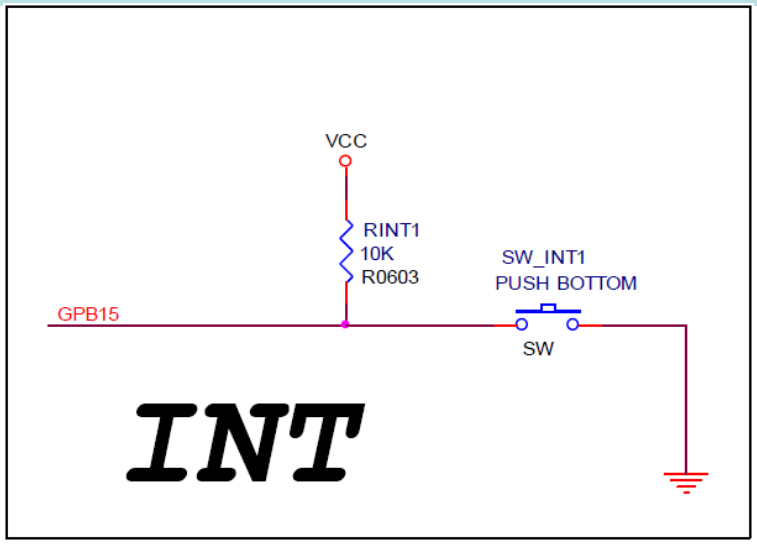
```
    // display number on 7-seg.
    show_seven_segment(0,number);//(Seven_Segment.c)
    //set GPA[2:0]=000
    GPIOA->DOUT &= ~(0x7);
    // clear pending IRQ4
    NVIC_ClearPendingIRQ(GPAB_IRQn); //IRQ4
}
```

問題：

1. 當離開中斷服務程式時，按鍵可能還沒有鬆開，會再次進入中斷服務程式。到鬆開為止，可能會產生多次的中斷。

3.6Test_7seg_SWInt_Bounce–計算彈跳的次數

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



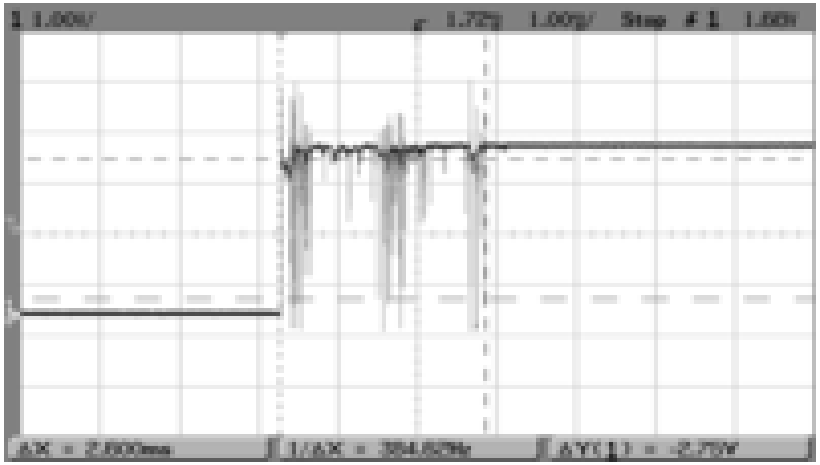
寫一程式，當按鍵按下時，計算彈跳(bounce)的次數。然後將次數顯示在7段顯示器。

GPIOB bit15作為輸入，設定為EINT1

設定為EINT1，啟用中斷，藉由中斷程式來執行。

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。



- 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。

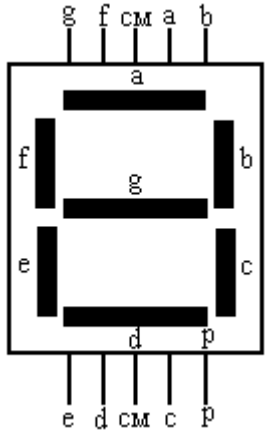
http://www.embedded.com/electronics-blogs/break-points/4024944/The-secret-life-of-switches

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

myNUC1xx-LB_002.c - 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

Test_7seg_SWInt_Bounce (1/4)

```
int32_t  value;

int main (void)
{
    // Initial 7-seg.
    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
    // Configure external interrupt
    DrvGPIO_EnableEINT1(E_IO_FALLING, E_MODE_EDGE,
    EINT1Callback);
```

Test_7seg_SWInt_Bounce (2/4)

```
    while (1)
    {

        // wait for key pressed, and count bounce
        value = 0;
        Delay(5000000); //30013 states

        // display key pressed number
        show_seven_segment(0,value); //448 states
        Delay(5000000); //30013 states
        close_seven_segment(); //clear GPIOC bit4-7 //151 states
    }
}
```

Initial GPIOx for 7-seg. (3/4)

```
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);
}
```

SW EINT1 interrupt routine (4/4)

```
void EINT1Callback(void)
{
    value++;
}
```

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.*