EIE3810 Microprocessor System Design Laboratory

Lab 4. Interrupt

School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen

2023-2024 Term 1

1. Objectives

In Lab 4, we will spend the 2-week session to study the following:

- To study the external interrupt setting of Cortex-M3, and read key input by an external interrupt
- To use an interrupt to read data from USART and shown onto LCD

2. Basics

Polling is a method to keep checking event status and discover which status has been changed. Subroutines will then be employed after the status changed. Programmers commonly use busy-wait loops (polling) to check if the status is changed. However, polling wastes most microprocessor's computation resource and makes programs slow.

Interrupt (or exception as in Cortex-M3) is a better solution for status checking. A certain event may trigger the interrupt controller; the processor will then put down the current job and serve the request. Thus, the processor does not need to waste time polling the status. The subroutine serving interrupt request is named interrupt handler, which should be programmed for individual exception number.

A higher priority interrupt can interrupt other handlers if the new request priority is higher than the current processing.

There are a few sources that can issue an interrupt request:

- External hardware EXTI, e.g., key pressing, data received from communication port;
- User software requests, e.g., user program requests operation system services;
- Emergency system fault, e.g., hardware fault, memory fault, usage fault and non-maskable interrupts (NMI).

Cortex-M3 has a powerful interrupt controller called NVIC (nested vector interrupt controller). It can handle 240 external interrupts (EXTIs) maximally.

STM32F10x has 60 maskable interrupt channels; 16 levels of programmable priority and nested interrupts. Table 1 shows some external interrupts on the development board.

Table 1. Keys, LEDs, USART and pin arrangement

Item	Input/Output	Operation	Connected to uP	Interrupt #
Key_Up	Input	Press=High	PA0	EXTI-0
Key0	Input	Press=Low	PE4	EXTI-4
Key1	Input	Press=Low	PE3	EXTI-3
Key2	Input	Press=Low	PE2	EXTI-2
DS0(LED0)	Output	Low=Lit	PB5	
DS1(LED1)	Output	Low=Lit	PE5	
USART1	Input/Output		PA9/PA10	USART1

Let's check EXTI-2 as an example. The procedures are elaborated as below.

1) Set priority group
   In Cortex-M3, NVIC has 240 registers NVIC_IPRx (8-bit each) to assign the priority from 0 to 255 to each of the available interrupts. 0 is the highest priority, and 255 is the lowest. Fig. 1 shows 32 of such registers. PRI_8 is NVIC_IPR8. It can be set for the priority level of the interrupt EXTI2, which is in position #8.

	31 24	23 16	15 8	7 0
E000E400	PRI_3	PRI_2	PRI_1	PRI_0
E000E404	PRI_7	PRI_6	PRI_5	PRI_4
E000E408	PRI_11	PRI_10	PRI_9	PRI_8
E000E40C	PRI_15	PRI_14	PRI_13	PRI_12
E000E410	PRI_19	PRI_18	PRI_17	PRI_16
E000E414	PRI_23	PRI_22	PRI_21	PRI_20
E000E418	PRI_27	PRI_26	PRI_25	PRI_24
E000E41C	PRI_31	PRI_30	PRI_29	PRI_28

Position	Priority	Type of priority	Acronym	Description	Address
⋮					
0	7	settable	WWDG	Window watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
⋮					
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
⋮					
59	66	settable	DMA2_Channel4_5	DMA2 Channel4 and DMA2 Channel5 global interrupts	0x0000_012C

Fig. 1 NVIC interrupt priority registers and the corresponding interrupts

But in STM32F103, only the higher 4 bits [7:4] of the priority register are used, so it can have 16-level of exception priorities.

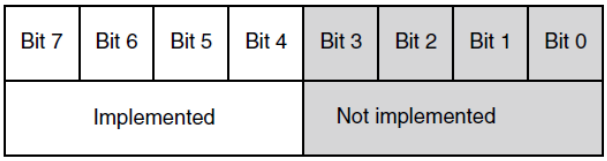

Fig. 2 Effective 4 bits of an interrupt priority register

In Cortex-M3, the priority is further controlled by 3-bit [10:8] PRIGROUP setting in AIRCR, i.e. Application Interrupt and Reset Control Register (Fig. 3)

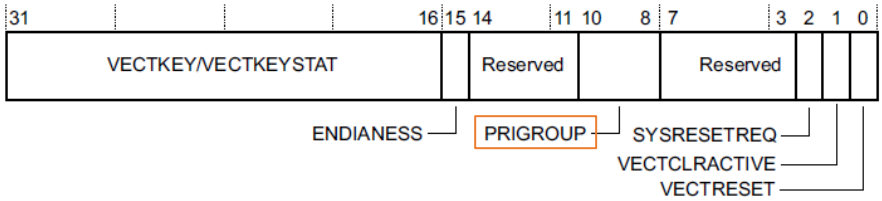

Fig. 3 Application Interrupt and Reset Control Register bit assignments

[10:8] PRIGROUP Interrupt priority grouping field:

PRIGROUP	Split of pre-emption priority from subpriority
0	7.1 indicates seven bits of pre-emption priority, one bit of subpriority
1	6.2 indicates six bits of pre-emption priority, two bits of subpriority
2	5.3 indicates five bits of pre-emption priority, three bits of subpriority
3	4.4 indicates four bits of pre-emption priority, four bits of subpriority
4	3.5 indicates three bits of pre-emption priority, five bits of subpriority
5	2.6 indicates two bits of pre-emption priority, six bits of subpriority
6	1.7 indicates one bit of pre-emption priority, seven bits of subpriority
7	0.8 indicates no pre-emption priority, eight bits of subpriority.

Fig. 4 Interrupt priority grouping field (PRIGROUP)

There are two types of priorities, i.e., pre-emption priority and subpriority. By setting the PRIGROUP (bits [10:8] in AIRCR), you can set different numbers of pre-emption priority and subpriority. The priorities of the interrupts are set by the rules below:

- If the pre-emption priority is higher (i.e., the pre-emption priority-bit number is smaller) that interrupt will be operated first. Interrupt with higher pre-emption priority can pause interrupts with lower pre-emption priority.
- If the interrupts have the same pre-emption priority, then they cannot pause each other. But the interrupt with higher subpriority (i.e., smaller by subpriority bits) will run first when two interrupts are activated simultaneously.
- If both pre-emption priority and subpriority bits are equivalent, they cannot pause each other. But the interrupt with a smaller position number will have a higher priority with two interrupts are activated simultaneously.

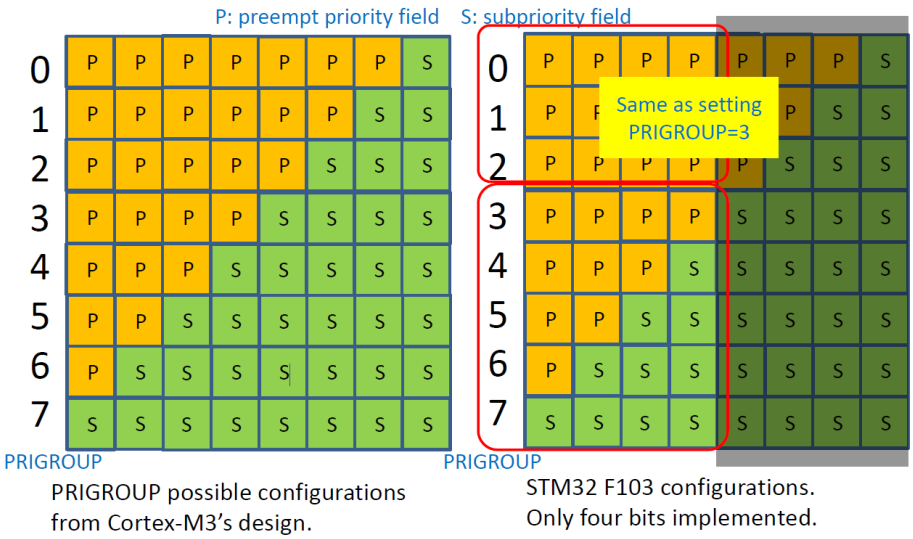

Fig. 5 Pre-emption priority and subpriority

The left part of Fig. 5 is the division of pre-emption priority and subpriority bits for Cortex-M3. In STM32F103, because the lower 4 bits are not used, setting PRIGROUP to 0, 1, 2, and 3 have the same effect.

In this experiment, we set PRIGROUP=5. That means in the register NVIC_IPR2, bits [7:6] and bits [5:4] indicate pre-exemption priority and subpriority, respectively.

To access AIRCR, you need to use SCB->AIRCR. Check core_cm3.h (Fig. 6)

```
714 /* Memory mapping of Cortex-M3 Hardware */
715 #define SCS_BASE            (0xE000E000)                          /*!< System Control Space Base Address */
716 #define ITM_BASE            (0xE0000000)                          /*!< ITM Base Address                  */
717 #define CoreDebug_BASE      (0xE000EDF0)                          /*!< Core Debug Base Address           */
718 #define SysTick_BASE        (SCS_BASE +  0x0010)                  /*!< SysTick Base Address              */
719 #define NVIC_BASE           (SCS_BASE +  0x0100)                  /*!< NVIC Base Address                 */
720 #define SCB_BASE            (SCS_BASE +  0x0D00)                  /*!< System Control Block Base Address */
721
722 #define InterruptType       ((InterruptType_Type *) SCS_BASE)     /*!< Interrupt Type Register           */
723 #define SCB                 ((SCB_Type *)           SCB_BASE)     /*!< SCB configuration struct          */
724 #define SysTick             ((SysTick_Type *)       SysTick_BASE) /*!< SysTick configuration struct      */
725 #define NVIC                ((NVIC_Type *)          NVIC_BASE)    /*!< NVIC configuration struct         */
726 #define ITM                 ((ITM_Type *)           ITM_BASE)     /*!< ITM configuration struct          */
727 #define CoreDebug           ((CoreDebug_Type *)     CoreDebug_BASE) /*!< Core Debug configuration struct */
728
```

```
155 typedef struct
156 {
157   __I  uint32_t CPUID;    /*!< Offset: 0x00  CPU ID Base Register                                  */
158   __IO uint32_t ICSR;     /*!< Offset: 0x04  Interrupt Control State Register                      */
159   __IO uint32_t VTOR;     /*!< Offset: 0x08  Vector Table Offset Register                          */
160   __IO uint32_t AIRCR;    /*!< Offset: 0x0C  Application Interrupt / Reset Control Register        */
161   __IO uint32_t SCR;      /*!< Offset: 0x10  System Control Register                               */
162   __IO uint32_t CCR;      /*!< Offset: 0x14  Configuration Control Register                        */
163   __IO uint8_t  SHP[12];  /*!< Offset: 0x18  System Handlers Priority Registers (4-7, 8-11, 12-15) */
164   __IO uint32_t SHCSR;    /*!< Offset: 0x24  System Handler Control and State Register             */
165   __IO uint32_t CFSR;     /*!< Offset: 0x28  Configurable Fault Status Register                    */
166   __IO uint32_t HFSR;     /*!< Offset: 0x2C  Hard Fault Status Register                            */
167   __IO uint32_t DFSR;     /*!< Offset: 0x30  Debug Fault Status Register                           */
168   __IO uint32_t MMFAR;    /*!< Offset: 0x34  Mem Manage Address Register                           */
169   __IO uint32_t BFAR;     /*!< Offset: 0x38  Bus Fault Address Register                            */
170   __IO uint32_t AFSR;     /*!< Offset: 0x3C  Auxiliary Fault Status Register                       */
171   __I  uint32_t PFR[2];   /*!< Offset: 0x40  Processor Feature Register                            */
172   __I  uint32_t DFR;      /*!< Offset: 0x48  Debug Feature Register                                */
173   __I  uint32_t ADR;      /*!< Offset: 0x4C  Auxiliary Feature Register                            */
174   __I  uint32_t MMFR[4];  /*!< Offset: 0x50  Memory Model Feature Register                         */
175   __I  uint32_t ISAR[5];  /*!< Offset: 0x60  ISA Feature Register                                  */
176 } SCB_Type;
```

Fig. 6 Coding in core_cm3.h

2) Initialize EXTI2

In this initialization stage, we need the following steps:

- Enable the clock of GPIOE (for Key2, which is connected to EXTI-2), and set PE2 as input pull-up. In this step, when you press Key2, the MCU can get the decreasing signal from high to low voltage.
- Enable the clock of AFIO (Fig. 7), as external interrupts registers are in AFIO (alternate-function I/O).

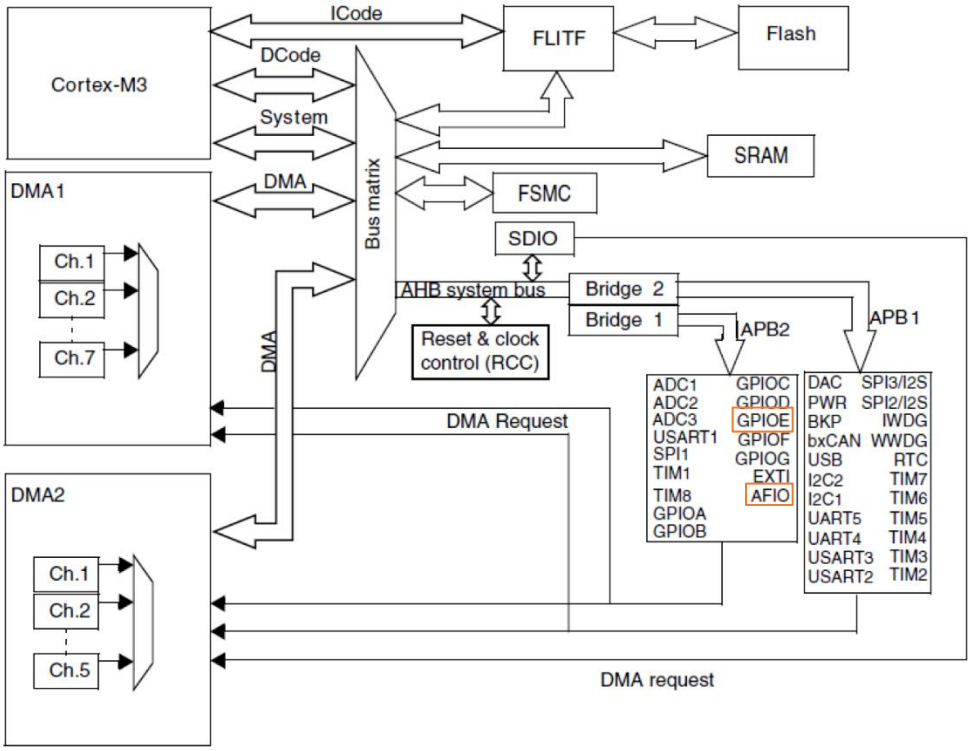

Fig. 7 System architecture

- Set external interrupt configuration register in AFIO for PE2.
  As shown in Fig. 8, EXTI-2 can come from PA2 to PG2. AFIO_EXTICR1 should be set to make the choice.

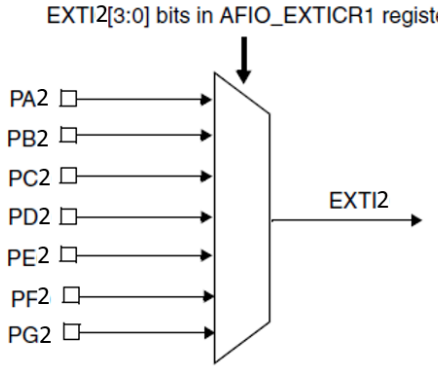

Fig. 8 Selection of PA2-PG2 to EXTI2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]:** EXTI x configuration (x= 0 to 3)
These bits are written by software to select the source input for EXTIx external interrupt.
Refer to *Section 10.2.5: External interrupt/event line mapping*
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

Fig. 9 AFIO_EXTICR1

- Set interrupt mask register of EXTI2 (Line 2). If Line x is masked, the interrupt will not be received.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Interrupt Mask on line x
0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

Fig. 10 Interrupt mask register

- Set falling trigger selection register of EXTI2 (Line2). If the input is a falling edge, the interrupt will be activated. You may also try the rising trigger selection register. Refer to *RM0008* for rising trigger selection register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Falling trigger event configuration bit of line x
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.
Note: Bit 19 used in connectivity line devices and is reserved otherwise.

Fig. 11 Falling trigger selection register

- Set interrupt priority register of EXTI2. To set this, you need to observe the data structure in Fig. 12 and coding in Fig. 16. NVIC->IP[8] is the priority register of EXTI2.

```
typedef struct
{
  __IO uint32_t ISER[8];          /*!< Offset: 0x000  Interrupt Set Enable Register           */
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];          /*!< Offset: 0x080  Interrupt Clear Enable Register         */
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];          /*!< Offset: 0x100  Interrupt Set Pending Register          */
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];          /*!< Offset: 0x180  Interrupt Clear Pending Register        */
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];          /*!< Offset: 0x200  Interrupt Active bit Register           */
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];          /*!< Offset: 0x300  Interrupt Priority Register (8Bit wide) */
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;             /*!< Offset: 0xE00  Software Trigger Interrupt Register     */
} NVIC_Type;
```

Fig. 12 NVIC_Type structure

- Set the Interrupt Set-Enable Register (ISER) to enable EXTI2. Observe Fig. 12. There are eight 32-bit ISERs (for the 240 external interrupts). For each one, the 32 bits are defined as in Fig. 13. For EXTI2, the interrupt position is #8 (refer to Fig. 1), so you can enable it by setting bit #8 in NVIC->ISER[0]. Reference can be taken from *Cortex-M3 Technical Reference Manual* (Page 8-12)

Bits	Field	Function
[31:0]	SETENA	Interrupt set enable bits. For write operation: 1 = enable interrupt 0 = no effect. For read operation: 1 = enable interrupt 0 = disable interrupt Writing 0 to a SETENA bit has no effect. Reading the bit returns its current enable state. Reset clears the SETENA fields.

Fig. 13 Interrupt set-enable register

3) Program the EXTI2 interrupt handler
The subroutine named EXTI2_IRQHandler() is called when EXTI2 interrupted is activated. You can write the coding to serve EXTI2 interrupt. But at the end of the subroutine, the bit for EXTI2 in the pending register (Fig. 14) of EXTI should be cleared.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **PRx:** Pending bit
0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a '1' into the bit.

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

Fig. 14 EXTI_PR

3. Experiments

3.1 Experiment 1: EXTI2 interrupt and onboard Key2

In this experiment, DS1 is programmed to flash periodically. Press Key2 will generate an interrupt and it will pause DS1 flashing, and flash DS0 instead. After the handler operation of Key2 is completed, DS1 can flash again.

```
26  int main(void)
27  {
28    EIE3810_LED_Init();
29    EIE3810_NVIC_SetPriorityGroup(5);//Set PRIGROUP
30    EIE3810_Key2_EXTIInit();//Initialize Key2 as an interrupt input
31    DS0_OFF;
32    while(1)
33    {
34      Delay(5000000);
35      DS1_ON;
36      Delay(5000000);
37      DS1_OFF;
38      count++;//Just count.
39    }
40  }
41
```

Fig. 15 main()

```
42  void EIE3810_Key2_EXTIInit(void)
43  {
44    RCC->APB2ENR |= 1<<6; //Add comments
45    GPIOE->CRL &=0xFFFFF0FF; //Add comments
46    GPIOE->CRL |=0x00000800; //Add comments
47    GPIOE->ODR |=1<<2; //Add comments
48    RCC->APB2ENR |=0x01; //Add comments
49    AFIO->EXTICR[0] &=0xFFFFF0FF; //Add comments
50    AFIO->EXTICR[0] |=0x00000400; //Add comments
51    EXTI->IMR |= 1<<2; //Add comments
52    EXTI->FTSR |= 1<<2; //Add comments
53    NVIC->IP[8] = 0x65; //Add comments
54    NVIC->ISER[0] |= (1<<8);  //Add comments
55  }
56
```

Fig. 16 EIE3810_Key2_EXTIInit()

```
57  void EIE3810_NVIC_SetPriorityGroup(u8 prigroup)
58  {
59    u32 temp1, temp2;
60    temp2= prigroup&0x00000007;
61    temp2 <<=8; //Add comments to explain "Why?"
62    temp1 = SCB->AIRCR; //Add comments
63    temp1 &=0x0000F8FF; //Add comments
64    temp1 |=0x05FA0000; //Add comments
65    temp1 |=temp2;
66    SCB->AIRCR = temp1;
67  }
68
```

Fig. 17 EIE3810_NVIC_SetPriorityGroup()

```
69  void EXTI2_IRQHandler(void)
70  {
71    u8 i;
72
73    for (i=0;i<10;i++)
74    {
75      DS0_ON;
76      Delay(3000000);
77      DS0_OFF;
78      Delay(3000000);
79    }
80    EXTI->PR = 1<<2;  //Add comments
81  }
82
```

Fig. 18 EIE3810_IRQHandler()

Procedures:

1) Type in the codes as in Fig. 15-18.
2) Refer to Lab 1, complete the LED initialization "EIE3810_LED_Init()" and define "DS0_ON, DS0_OFF, DS1_ON, DS1_OFF" by yourself.
3) Compile and download it to your project board.
4) Press Key2 once after reset.
5) Change falling trigger selection register to rising trigger selection register. You shall be able to find the rising trigger selection register in *RM0008*.

[Demonstration] When you have completed 4) and 5), demonstrate to the instructor or TAs that your program works.

[In Report] Include your test result. This time, it may be difficult to show that your experiment is successful by snapshots of the representative frames of video. But you can use a logic analyzer to obtain some figures like below. This is a good way to demonstrate to the reader of your report that your experiment is successful.

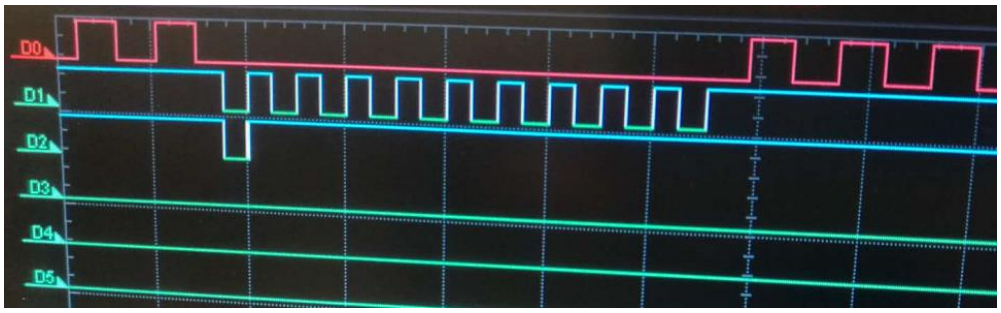

Fig. 19 Observation by a logic analyzer (1)

[Question]

(1) We set NVIC->IP[8] = 0x65 here. What other values can have the same priority effect as pre-emption priority=0b01, and subpriority=0b10?

(2) What is the difference in the observed signals on the logic analyzer screen for falling edge triggering and rising edge triggering?

(3) When you use the rising edge to trigger the interrupt, you will sometimes get the signals like in Fig. 20. Do you see the problem (falling edge trigger, instead of rising edge trigger)? Investigate this problem and explain why that occurs?

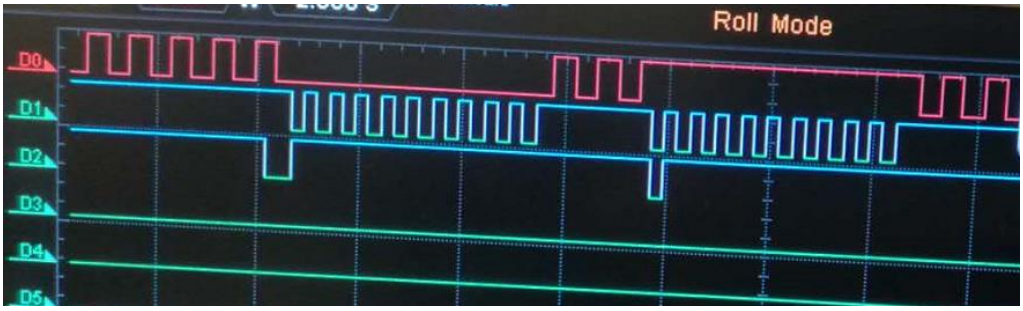

Fig. 20 Observation by logic analyzer (2)

3.2 Experiment 2: EXTI interrupt and on board Key_Up

Set another interrupt with Key_Up and drive LED DS1.

```
27  int main(void)
28  {
29    EIE3810_LED_Init();
30    EIE3810_NVIC_SetPriorityGroup(5);//Set PRIGROUP
31    EIE3810_Key2_EXTIInit();//Initialize Key2 as an interrupt input
32    EIE3810_KeyUp_EXTIInit(); //Initialize KeyUp as an interrupt input
33    DS0_OFF;
34    DS1_OFF;
35    while(1)
36    {
37      count++;//Just count.
38    }
39  }
40
```

Fig. 21 main() for Experiment 2

Procedures:

1) Modify main() to Fig. 21. Remove DS1 flashing behavior.
2) Compose your EIE3810_KeyUp_EXTIInit() for Key_Up.
3) Refer to EXTI2_IRQHandler(), compose a handler for Key_Up. Hint: the name of the subroutine should be EXTI0_IRQHandler().
4) Set the priority of Key_Up to 0x75.
5) Compile and download your project to the board.
6) Press Key_Up once after reset. Explain your observation.
7) Press Key_Up during Key2 handler is running. Explain your observation.
8) Press Key2 during Key_Up handler is running. Explain your observation.

[Demonstration] When you have completed 7) and 8), demonstrate to the instructor or TAs that your program works.

[In Report] Use a logic analyzer to measure the 4 signals (two keys and two LEDs), and include your test results.

[Question] Do you think it is better to set a rising edge trigger or a falling edge trigger for Key_Up? Why?

3.3 Experiment 3: Test interrupts priority

Procedures:

1) Set the priority of Key_Up to 0x95.
2) Compile and download your project to the board.
3) Press Key_Up during Key2 handler is running. Explain your observation.
4) Press Key2 during Key_Up handler is running. Explain your observation.
5) Set the priority of Key_Up to 0x35.
6) Compile and download your project to the board.
7) Press Key_Up during Key2 handler is running. Explain your observation.
8) Press Key2 during Key_Up handler is running. Explain your observation.

[Demonstration] When you have completed 3), 4), 7) and 8), demonstrate to the instructor or TAs that your program works.

[In Report] Use logic analyzer to measure the 4 signals (two keys and two LEDs), and include your test results.

3.4 Experiment 4: UART reading based on interrupt

In Lab 2, we have learned to transmit data by USART. To read data from USART, you can use a while loop to continuously check if there is data from the RX pin, but this is not efficient, as it wastes a lot of computational resources.

USART also has interrupts. You can find the interrupt position number in Fig. 1. In this experiment, we will use USART1, and the interrupt position # is 37. Some characters transmitted from the computer to STM32 will be shown onto the LCD. You will also be able to develop some coding to check and show the register value onto LCD.

Procedures:

1) Copy the project folder of Experiment 4 in Lab 3
2) Modify main() in main.c based on Fig. 22.

```
13  int main(void)
14  {
15      EIE3810_clock_tree_init();
16      EIE3810_LED_Init();
17      EIE3810_TFTLCD_Init();
18      Delay(500000);
19      EIE3810_TFTLCD_Clear(WHITE);
20      EIE3810_NVIC_SetPriorityGroup(5);  //Set PRIGROUP
21      EIE3810_USART1_init(72, 9600);
22      EIE3810_USART1_EXTIInit();
23      USART_print(1,"1234567890");
24      while(1)
25      {
26        USART_print(1,"EIE3810_Lab4");
27        while (!((USART1->SR >> 7) & 0x1));
28        Delay(10000000);
29      }
30
31  }
```

Fig. 22 main() for Experiment 5

3) Copy the related .c and .h files needed for LCD and USART (in Lab 2 and Lab 3) into the board folder and add them into your project. Do not forget to the #include to add the header into the main.c.
4) Type EIE3810_USART1_EXTIInit() into EIE3810_USART.c based on Fig. 23.

```
94  void EIE3810_USART1_EXTIInit(void)
95  {
96    NVIC->IP[37]  =0X65;       //Add comments
97    NVIC->ISER[1] |=1<<5;      //Add comments
98  }
```

Fig. 23 EIE3810_USART1_EXTIInit()

5) At the end of the subroutine EIE3810_USART1_init(u32 pclk1, u32 baud rate), add a line to (1) enable receive interrupt and (2) enable receiver. Refer to the register of USART1->CR1 in *RM0008* to set the bit correctly.
6) Complete your USART1_IRQHandler() for USART1 interrupt as in Fig. 24. When 'Q' is received, turn on LED0. When 'H' is received, turn off LED0. You can use eagleCom.exe to send the letter as shown in Fig. 25.

```
101  void USART1_IRQHandler(void)
102  {
103    u32 buffer;
104    if(USART1->SR &(1<<5)) //Add comments
105      {
106        buffer=USART1->DR; //Add comments
107        if(buffer=='Q')
108          { //Turn on LED0
109            }
110        else if(buffer=='H')
111          { //Turn off LED0
112            }
113    }
114  }
```

Fig. 24 USART1_IRQHandler()

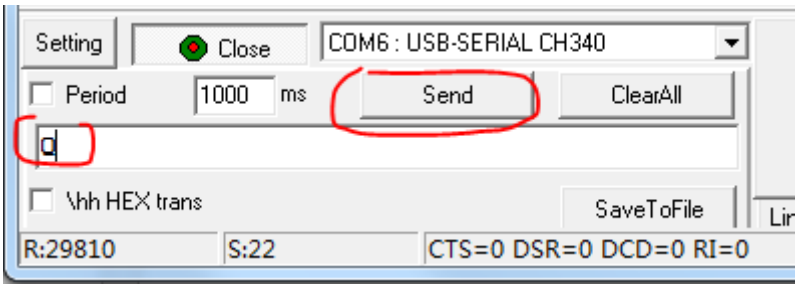

Fig. 25 Send letter 'Q' by eagleCom.exe

7) It will be helpful to show the register's value on to the LCD, so that you can check if the setting is correct. Revise USART1_IRQHandler(), so that when 'R' is received, LCD can show the registers' value of USART1->CR1, USART1->DR, and letter 'R', as shown in Fig. 26. The higher 16 bits are reserved, so you can just show the lower 16 bits. The first and third rows are values of USART1->CR1 and USART1->DR in binary format. The second and fourth rows are the indices of each bit. The background and foreground colors change iteratively, so that you may feel easier to locate the bits.

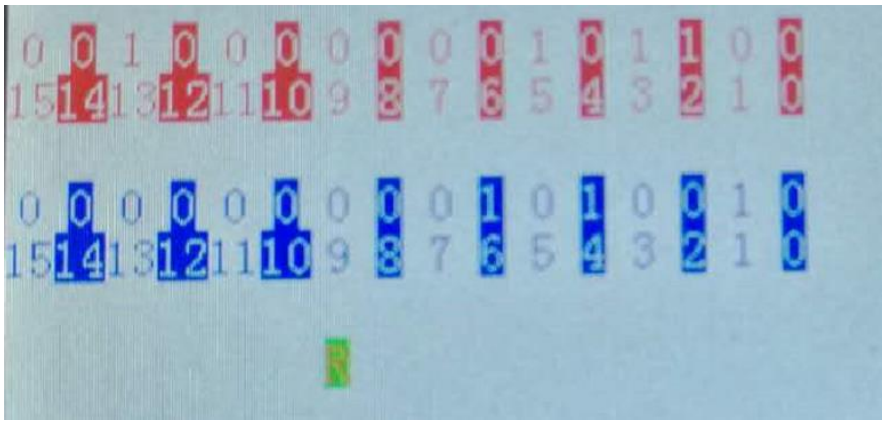

Fig. 26 LCD image

[Demonstration] When you have completed 6) and 7), demonstrate to the instructor or TAs that your program works.

[In Report] Include your test results. You do not need to use logic analyzer this time.

4. Lab Report and Source Code

Submit the report softcopy and your code (complete project folder of each experiment) in zip format to Blackboard by the deadline below:

- Tuesday Class (L02): 15:00, Tuesday, November 14, 2023
- Friday Class (L01): 9:00, Friday, November 17, 2023

Each day of late submission will result in 10% deduction in the report and source code raw marks.