Operating System Lab Assignment 02: Modification, Deploy Processor Exceptions and Interrupts; Design and Implementation of EXTI (external Interrupt).

Dr. Mosaddek Tushar, Professor
Computer Science and Engineering, University of Dhaka,
Version 1.0 (Premature)
Due Date: Following week

Aug 06, 2023

Contents

1 Objectives and Policies **1**
1.1 General Objectives 1
1.2 Assessment Policy 2

2 What to do? **2**

3 Deployment of processor exception and interrupt **2**
3.1 Implementation of NVIC Interrupt services 3
3.1.1 NVIC functions need to add to your kernel (if not there) 3

4 Design and deploy functions for implementation of EXTI and software interrupt. **6**
4.1 EXTI interrupt handler function 8
4.2 Configuration File 8
4.3 Testing and Submission 9

Appendices **10**

A DUOS Directory Structure – Tree **10**

1 Objectives and Policies

1.1 General Objectives

The objectives of the lab assignment are to understand and have hands-on training to understand the underline architecture of the Operation System, use microprocessors features, use knowledge learned in the Operating System class, and develop an infrastructure for future tech industries in Bangladesh. The broader objectives are to use AI and Machine learning techniques (where applicable) on top of the designed environment to build exportable products in Bangladesh. We (including students) gradually include all the functionalities ready for industries.

1.2 Assessment Policy

The assignment has three level objectives (i) primary objectives, (ii) advanced objectives, and (iii) optional boost objectives. Every student must complete the primary objective; however, they can attempt advanced and optional Boost-up objectives. The advanced objective will be a primary objective in the subsequent assignment. Further, you can achieve five (5) marks for completing the optional objectives and add these marks at the end of the semester with your total lab marks. However, you cannot get more than 100% assigned for the labs. The current lab does not contain advanced or optional boost objectives.

2 What to do?

- Deploy processor exception and interrupt, writing test routine, and modify
- Implement EXTI and Software Interrupt
- Write test programs for EXTI and Software Interrupts

3 Deployment of processor exception and interrupt

The following sub-section describes the data structure and given function to handle process exceptions and peripheral interrupts. Add the implementation to the DUOS and write your test program. The test program must have the following components. If the given implementation of the interrupt/exception has errors, correct it. In addition, keep the interrupt handle function as simple as possible. Avoid using 'kprintf' or try to minimize the use of it as much as possible. Create two files sys.h and sys.c that contains all the interrupt management codes for this assignment

- sys.h in 'src/kern/arch/stm32f446re/include/sys' directory and
- sys.c in 'src/kern/arch/stm32f446re/sys' directory

To do so, you must write a 'header – test_interrupt.h' and implementation functions in '.c – test_interrupt.c' files in 'src/kern/include/testprog/' and 'src/kern/lib/testprog/' sub directories accordingly.

(a) Test HardFault_Handler – enable, create an event for hard fault to be activated, print message when occur, disable and show the status. You may need to modify 'HardFault_Handler (..)' service routine in 'src/kern/arch/stm32f446re/sys/stm32_startup.c'.

(b) Enable and disable 'systick' interrupt and demonstrate. See the essential functions give below.

(c) Modify the priority of the 'SYSTICK' interrupt and use '__set_BASEPRI(..)' to disable interrupt with priority lower than a certain priority number and clear. Demonstrate 'SYSTICK ' functionality. Use

  - __NVIC_GetPriority(IRQn_TypeDef IRQn) – to see the current priority.

- __NVIC_DisableIRQn(IRQn_TypeDef IRQn) – to disable and interrupt
- __NVIC_SetPriority (IRQn_TypeDef IRQn,uint32_t priority) – to set priority of an interrupt
- __NVIC_EnableIRQn(IRQn_TypeDef IRQn) – to enable an interrupt

(d) Write a test function to reboot the Operating system including the micro-controller. This function emulate and activate the 'Reset_Handler'

(e) Write and modify if you find any problems in the interrupt management functions.

The following subsections describes the detail implementation of the interrupt management functions.

3.1 Implementation of NVIC Interrupt services

*****Important: You may need 'STM32F-Programming Manual'**

NVIC – nested interrupt vector provides configurable interrupt abilities to the processor. The processor facilitates low latency exceptions and interrupts handling and control power management. ARM Cortex-M has several exceptions and interrupts to handle system events for particular tasks and error conditions. ARM Cortex-M4 has 15 exceptions and 240 interrupts. The first exceptions are internal to the processor, and the remaining 240 interrupts are to accomplish tasks for external events, including peripherals and the outside world. The NVIC supports 256 levels of programmable dynamic priorities. ARM Cortex-M further segregates interrupt-priority bits into groups (preemptive) and subgroups. The interrupts assign the same group or preemption with higher priority subgroup never suspended while executing.

The program running in privileged mode has full access to the NVIC. But you can cause interrupts to enter a pending state in user mode if you enable this capability using the Configuration Control Register. Any other user mode access causes a bus fault. Unless otherwise stated, you can access all NVIC registers using byte, halfword, and word accesses. NVIC registers reside within the SCS (System Control Space). All NVIC registers and system debug registers are little-endian regardless of the endianness state of the processor. You configure the number of interrupts and bits of interrupt priority during implementation. The software can choose only to enable a subset of the configured number of interrupts and can decide how many bits of the configured preferences to use.

We already define the exceptions, interrupts, and handlers in stm32_statup.h file. However, the definitions of the interrupts handlers are weakly defined and map to the default handler. The system developers must write the task of an interrupt or exception according to their needs. Nevertheless, the prerequisites are developing a kernel function set to enable, disable, and prioritize the interrupt assignment. To do this, you must acquire knowledge of interrupt vectors and the configuration of the registers. The functions to use the interrupts in the ARM Cortex-M processor that you need to implement are listed below.

3.1.1 NVIC functions need to add to your kernel (if not there)

Before going to the function detail, you need to define the data structure for SCB, SCS, NVIC, and IRQn_Type to access the registers. You must express all data structure and address definitions in the 'stm32_peps.h' file and function details in 'sys.c'.

1. **SCB data structure:** The starting address for SCB is '0xE000ED00'

```
typedef struct
{
  volatile uint32_t CPUID;   // CPUID Base Register 0x0
  volatile uint32_t ICSR;    // Interrupt Control and State Register 0x4
  volatile uint32_t VTOR;    // Vector Table Offset Register 0x8
  volatile uint32_t AIRCR;   // Application Interrupt and Reset Control Register 0xC
  volatile uint32_t SCR;     // System Control Register   0x10
  volatile uint32_t CCR;     // Configuration and Control Register 0x14
  volatile uint8_t SHPR[12]; // Exception priority setting for system exceptions
  volatile uint32_t SHCSR;   // System Handler Control and State Register  0x24
  volatile uint32_t CFSR;    // Configurable Fault Status Register combined of MemManage
      // Fault Status Register, BusFault Status Register, UsageFault Status Register 0x28
  volatile uint32_t HFSR;    // HardFault Status Register 0x2C
  volatile uint32_t DFSR;    //Hint information for causes  of debug events
  volatile uint32_t MMFAR;    // MemManage Fault Address Register  0x34
  volatile uint32_t BFAR;    // BusFault Address Register  0x38
  volatile uint32_t AFSR;    // Auxiliary Fault Status Register 0x3C
  volatile uint32_t PFR[2];  // Read only information on available processor features
  volatile uint32_t DFR;     // Read only information on available debug features
  volatile uint32_t AFR;     // Read only information on available auxiliary features
  volatile uint32_t MMFR[4]; // Read only information on available memory model features
  volatile uint32_t ISAR[5]; //
  uint32_t RESERVED1[5];    //
  volatile uint32_t CPACR;   // Coprocessor access control register 0x88
 } SCB_TypeDef;

```

2. **NVIC Data Structure**: Starting address: ‘0xE000E100’. Verify with the ‘Cortex-M4 programming manual’ if the data structure is acceptable to the byte spacing.

```
typedef struct
{
  //define NVIC register compenenets -- use volatile data type
  volatile uint32_t ISER[8]; /*!< Offset: 0x000  addr: 0xE000E100 Interrupt Set Enable
    Register*/
  uint32_t RESERVED0[24];
  volatile uint32_t ICER[8]; /*!< Offset: 0x080  addr: 0xE000E180 Interrupt Clear Enable
    Register*/
  uint32_t RSERVED1[24];
  volatile uint32_t ISPR[8]; /*!< Offset: 0x100  addr: 0xE000E200 Interrupt Set Pending
    Register*/
  uint32_t RESERVED2[24];
  volatile uint32_t ICPR[8]; /*!< Offset: 0x180  addr: 0xE000E280 Interrupt Clear Pending
     Register*/
  uint32_t RESERVED3[24];
  volatile uint32_t IABR[8]; /*!< Offset: 0x200  addr: 0xE000E300 Interrupt Active bit
    Register*/
  uint32_t RESERVED4[56];
  volatile uint8_t  IP[240]; /*!< Offset: 0x300  addr: 0xE000E400 Interrupt Priority
    Register (8Bit wide) */
  uint32_t RESERVED5[644];
  volatile  uint32_t STIR;   /*!< Offset: 0xE00  addr: 0xE000EF00 Software Trigger
    Interrupt Register*/
}NVIC_TypeDef;

```

3. **Priority gouping**: Register AIRCR in SCB is used to determine the number of bits for priority prouping. See lecture slides

4. You also need **PRIMASK**, **FAULTMASK** and **BASEPRI** register to mask or unmask interrupts

5. Interrupt Data Structure

```
    enum IRQn_TypeDef {
      NonMaskableInt_IRQn = -14,
      HardFault_IRQn = -13,
      MemoryManagement_IRQn = -12,
      BusFault_IRQn = -11,
      UsageFault_IRQn = -10,
      SecureFault_IRQn = -9,
      SVCall_IRQn = -5,
      DebugMonitor_IRQn = -4,
      PendSV_IRQn = -2,
      SysTick_IRQn = -1,
      WWDG_STM_IRQn = 0,
      PVD_STM_IRQn = 1,
      ......
      ......
    }

```

1. void __NVIC_SetPriority (IRQn_TypeDef IRQn,uint32_t priority): This function takes two arguments (i) interrupt number and sets the priority to the interrupt. Note that priority in the above NVIC register is 8-bit. The priority puts the preference to the ISR executing before the lower (higher number) priority interrupts.

2. uint32_t __NVIC_GetPriority(IRQn_TypeDef IRQn): Return the priority set to the interrupt.

3. void __NVIC_EnableIRQn(IRQn_TypeDef IRQn): enable interrupt given as argument or interrupt number (IRQn_typeDef) – data structure (enumerator) defined earlier.

4. void __NVIC_DisableIRQn(IRQn_TypeDef IRQn): Disable interrupt.

5. void __disable_irq(): Masking interrupts ('__disable_irq()') – all interrupts other than HardFault, NMI, and reset.

6. void __set_BASEPRI(uint32_t value): __set_BASEPRI(uint32_t value) function mask interrupt number greater and equal to the given interrupt priority as an argument.

7. void __enable_irq(): Enable (unmask) all interrupts

8. void __unset_BASEPRI(uint32_t value): __unset_BASEPRI(uint32_t value) function unmask interrupts greater or equal to the given argument/priority number.

9. void __set_PRIMASK(uint32_t priMask): Prevent all interrupt without non-maskable interrupt.

10. uint32_t get_PRIMASK(void): Return value of the PRIMASK register

11. void __enable_fault_irq(void) and void __set_FAULTMASK(uint32_t faultMask): Enable all interrupt including FaultMask.

12. void __disable_fault_irq(void): Disable or prevent all interrupt including FaultMask

13. uint32_t __get_FAULTMASK(void): This function will return the status of the masking value of FaultMask register

14. void __clear_pending_IRQn(IRQn_TypeDef IRQn): Clear interrupt pending bit
15. uint32_t __get_pending_IRQn(IRQn_TypeDef IRQn): It returns the pending status of an interrupt.
16. uint32_t __NVIC_GetActive (IRQn_TypeDef IRQn): This function return the active status of the interrupt.

Note: Some of the functions may be semantically identical. However, different expressions embellish your code and enhance usability and interpretation.

4 Design and deploy functions for implementation of EXTI and software interrupt.

EXTI interrupt enables recognizing the outside world for data communication. For example, a camera sends an interrupt by changing the status of the interrupt pin from '0' to '1' or '1' to '0'. In this case, EXTI interrupt is very helpful to understand and capture data from the camera. It saves a few microprocessor clocks used for polling pins for data availability. Another example is the keyboard that sends an ASCII byte and an interrupt to the system to enable recognition of a character. The EXTI can be presents using the following data structure. Add data structure of the two system component in 'src/kern/arch/stm32f446re/include/stm32_peps.h'. Comments given.
Memory map: Address 0x40013C00

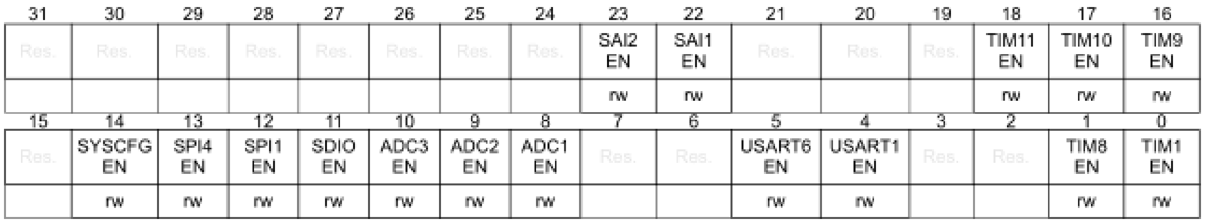
```
typedef struct
{
  volatile uint32_t  IMR;   /*!<EXTI Interrupt mask enable Register, Address offset : 0x00 */
  volatile uint32_t  EMR;   /*!<EXTI Event mask enable Register, Address offset : 0x04 */
  volatile uint32_t  RTSR;  /*!<EXTI Rising Trigger Select Register, Address offset : 0x08 */
  volatile uint32_t  FTSR;  /*!<EXTI Falling Edge Selection Regiter, Address offset : 0x0C */
  volatile uint32_t  SWIER; /*!<EXTI Software Interrupt Event regiter, Address offset : 0x10 */
  volatile uint32_t  PR;   /*!<EXTI Pending Regiter, Address offset : 0x1C */
} EXTI_TypeDef;
```

and SYSCFG data structure to be used: Memory map address: 0x40013800

```
  typedef struct
  {
  volatile uint32_t MEMRMP;  /*!< SYSCFG memory remap register,  Address offset: 0x00  */
  volatile uint32_t PMC; /*!< SYSCFG peripheral mode configuration register,  Address offset: 0
    x04 */
  volatile uint32_t EXTICR[4];  /*!< SYSCFG external interrupt configuration registers, Address
     offset: 0x08-0x14 */
  uint32_t  RESERVED[2];  /*!< Reserved, 0x18-0x1C */
  volatile uint32_t CMPCR;      /*!< SYSCFG Compensation cell control register, Address offset:
     0x20*/
  uint32_t RESERVED1[2]; /*!< Reserved, 0x24-0x28 */
  volatile uint32_t CFGR;     /*!< SYSCFG Configuration register, Address offset: 0x2C */
  } SYSCFG_TypeDef;
```

The configuration for EXTIx describes in the following steps: Please see the reference manual

STEP 1: Enable SYSCFG in RCC APB2EN bus register bit[14]

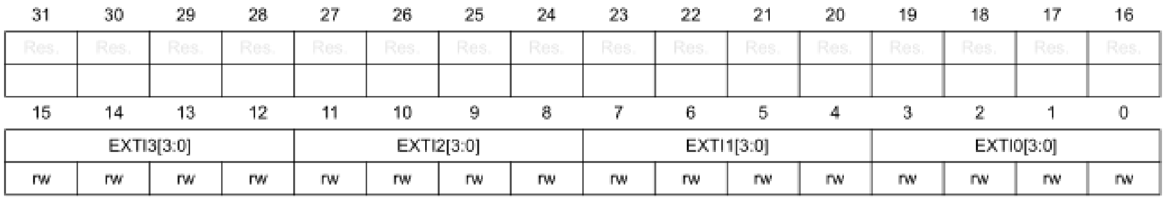


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2 EN	SAI1 EN	Res.	Res.	Res.	TIM11 EN	TIM10 EN	TIM9 EN
								rw	rw				rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Res.	SYSCFG EN	SPI4 EN	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Res.	Res.	USART6 EN	USART1 EN	Res.	Res.	TIM8 EN	TIM1 EN
	rw	rw	rw	rw	rw	rw	rw			rw	rw			rw	rw

Figure 1: APB2ENR

To enable SYSCFG clock set '1' to bit[14] and '0' to disable.

STEP 2: Configure the EXTI configuration register in the SYSCFG_EXTICRx register(s) [x=1··· 4]

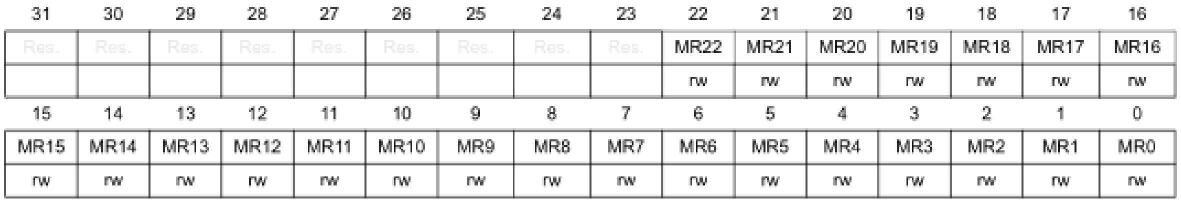


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw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

Figure 2: SYSCFG_EXTICR1 *Register*[1]

For sensing the eternal interrupt on PIN 0-15 of port A,··· H. Set corresponding bits[0···3]. For example, to set PA0 for interrupt, select '0000' to bit position from 0 to 3 of register SYSCFG _EXTICR1. For PB0, set '0001', PC0: '0010', and so on. For pin PA10, set '0000' to bit position from 8 to 11 of the EXTICR3 register. you must set the pin pull up or down – see GPIO_PUPDR register.

STEP 3: Enable EXTI using interrupt Mask Register [IMR]

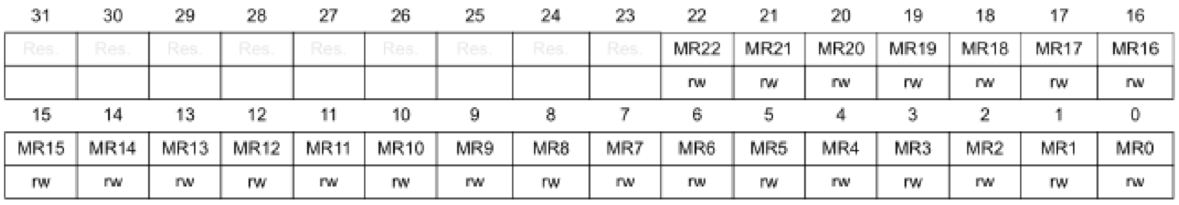


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 3: EXTI_IMR register

See the reference manual

STEP 4: Configure EXTI_EMR register

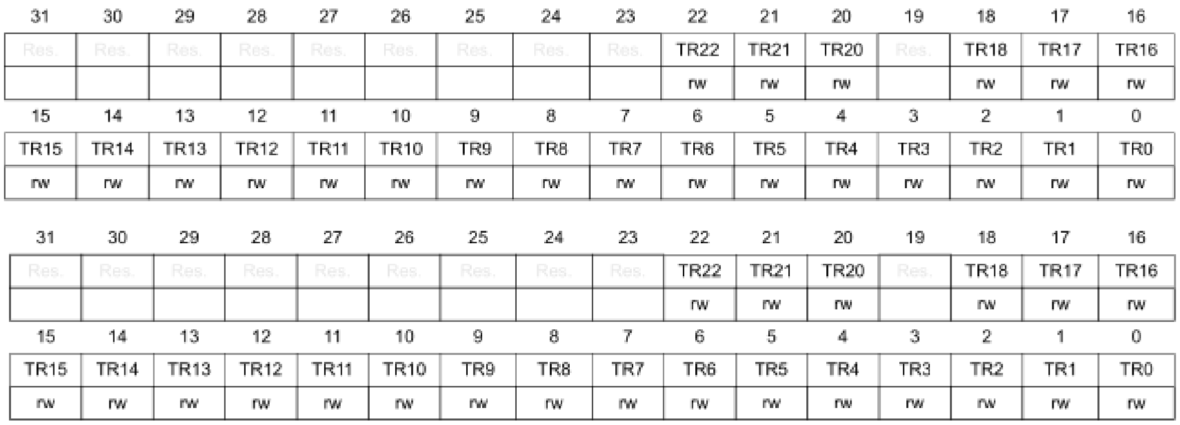


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 4: EXTI EMR register

See the reference manual

STEP 5: Configure rising edge or falling edge



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TR22	TR21	TR20	Res.	TR18	TR17	TR16
									rw	rw	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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TR22	TR21	TR20	Res.	TR18	TR17	TR16
									rw	rw	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

Figure 5: EXTI RTSR and FTSR registers

See the reference manual. Rising edge means changing pin (input) status from '0' (0V) to '1' (3.3/5.0V) and falling edge means changing from '1' to '0'.

STEP 6: Set interrupt priority for the configured EXTI : Use interrupt set priority function given above;

STEP 7: Enable EXTI : Use interrupt enable function given above;

Note that the interrupt number for the EXTIx are assiged in 'src/kern/arch/stm32f446re/include/stm32_startup.h'

4.1 EXTI interrupt handler function

File 'stm32_startup.h' contains the 'EXTIx_Handler()' functions prototype with weak attributes. Write function detail in the 'stm32_startup.c' file. The interrupt handler functions must clear the pending bit (PR register in EXTI˙TypeDef) before executing any other code.

4.2 Configuration File

Your 'sys.h' and 'sys.c' should contain the configuration for EXTIx interrupt and the function name is 'EXTI_Init(port, pin)'. Whenever you need use debugger for configuration.

4.3 Testing and Submission

Write a program for testing the EXTI interrupt and demonstrate. Submit your code (all update) to the class website. The link `https://www.youtube.com/watch?v=TMvS42ru9Gs&t=56s` contains a helpful video for EXTI.

Appendices

A DUOS Directory Structure – Tree

```
duos
└── src
    ├── compile
    │   ├── Makefile
    │   ├── mapfile
    │   ├── object
    │   └── target
    ├── doc
    │   └── Readme.txt
    └── kern
        ├── arch
        ├── dev
        ├── include
        ├── kmain
        ├── lib
        ├── proc
        ├── syscall
        ├── thread
        └── vfs
duos
└── src
    ├── compile
    │   ├── Makefile
    │   ├── mapfile
    │   ├── object
    │   └── target
    ├── doc
    │   └── Readme.txt
    └── kern
        ├── arch
        ├── dev
        ├── include
        ├── kmain
        ├── lib
        ├── proc
        ├── syscall
        ├── thread
        └── vfs
duos
└── src
    ├── compile
    │   ├── Makefile
    │   ├── mapfile
    │   ├── object
    │   └── target
    ├── doc
    │   └── Readme.txt
    └── kern
        ├── arch
        ├── dev
        ├── include
        ├── kmain
        ├── lib
        ├── proc
        ├── syscall
        ├── thread
        └── vfs
duos
└── src
    ├── compile
    │   ├── Makefile
    │   ├── mapfile
    │   ├── object
    │   └── target
    ├── doc
    │   └── Readme.txt
    └── kern
        ├── arch
        ├── dev
        ├── include
        ├── kmain
        ├── lib
        ├── proc
        ├── syscall
        ├── thread
        └── vfs
```

Alert: You can discuss with your classmates, however, do not copy code from others.