Interrupts

Lecture 18

Josh Brake
Harvey Mudd College

Learning Objectives

By the end of this lecture you will be able to:

- Explain the basic exception model used on ARM Cortex-M4 processors
- Configure interrupts to quickly respond to information from on-board peripherals like GPIO pins and timers

Outline

- Interrupts and Exceptions
  - The exception model in ARM Cortex-M4 processors
  - The Nested Vector Interrupt Controller (NVIC)
  - Configuring interrupts on ARM Cortex-M4
- Activity
  - Toggle LED with switch using polling and interrupts

ARM Cortex-M4 Exception Model

Sources of exceptions

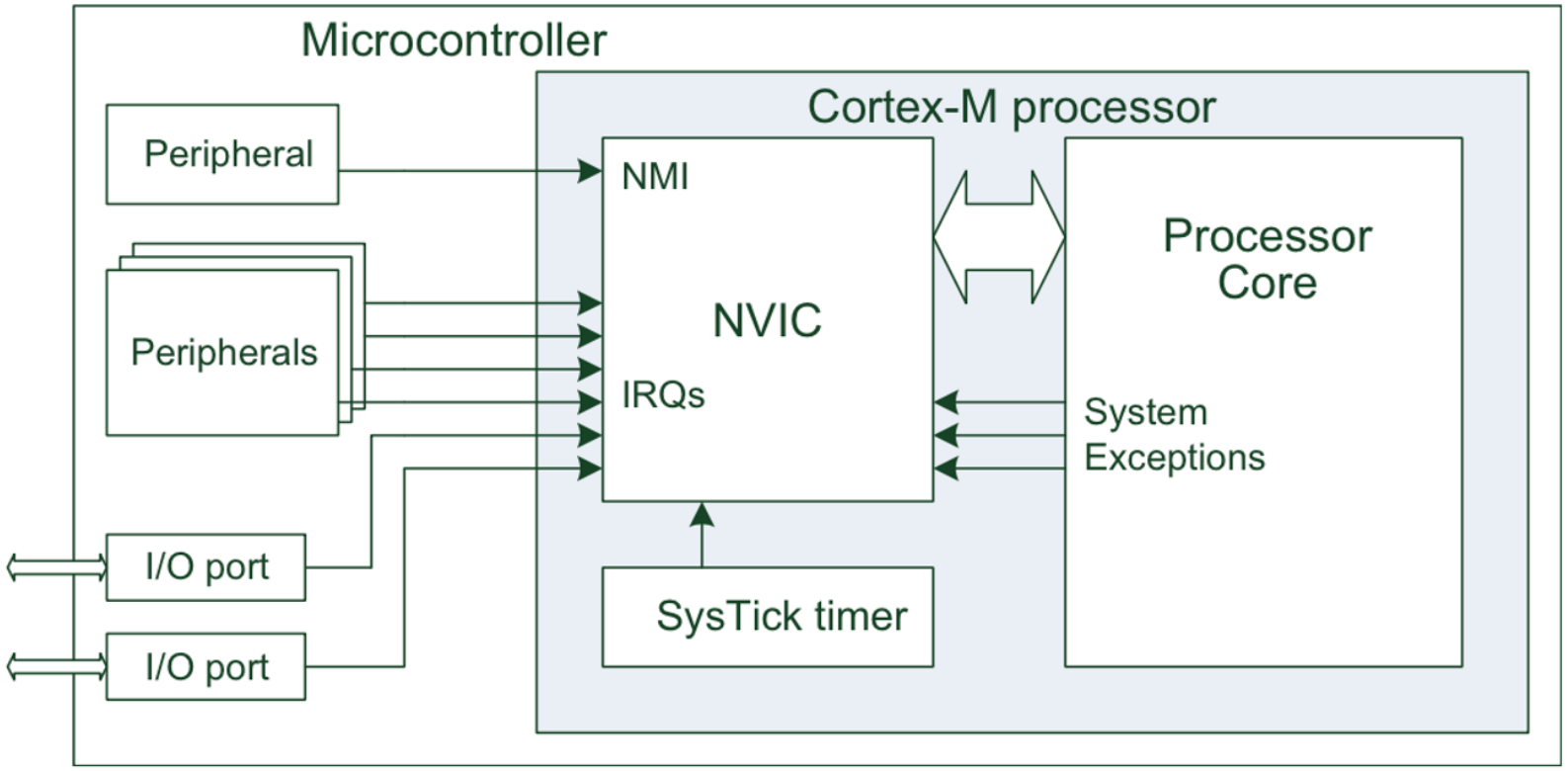


FIGURE 7.1

Various sources of exceptions in a typical microcontroller

Figure 7.1, p.230 The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors

Interrupt Servicing Sequence

1. Peripheral _____ interrupt request
2. Processor _____ currently operating task
3. Processor executes an _____ to service the peripheral and optionally clear the interrupt request
4. Processor _____ previously suspended task.

Vector Table

- Table of addresses to _____ that are placed starting at address `0x0000000`.
- Each memory location contains the _____ of different exception/interrupt handlers

Table 46. STM32L41xxx/42xxx/43xxx/44xxx/45xxx/46xxx vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
11	18	settable	DMA1_CH1	DMA1 channel 1 interrupt	0x0000 006C
12	19	settable	DMA1_CH2	DMA1 channel 2 interrupt	0x0000 0070
13	20	settable	DMA1_CH3	DMA1 channel 3 interrupt	0x0000 0074
14	21	settable	DMA1_CH4	DMA1 channel 4 interrupt	0x0000 0078
15	22	settable	DMA1_CH5	DMA1 channel 5 interrupt	0x0000 007C
16	23	settable	DMA1_CH6	DMA1 channel 6 interrupt	0x0000 0080
17	24	settable	DMA1_CH7	DMA1 channel 7 interrupt	0x0000 0084
18	25	settable	ADC1_2	ADC1 and ADC2(2) global interrupt	0x0000 0088
19	26	settable	CAN1_TX(1)	CAN1_TX interrupts	0x0000 008C
20	27	settable	CAN1_RX0(1)	CAN1_RX0 interrupts	0x0000 0090
21	28	settable	CAN1_RX1(1)	CAN1_RX1 interrupt	0x0000 0094
22	29	settable	CAN1_SCE(1)	CAN1_SCE interrupt	0x0000 0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK/TIM15	TIM1 Break/TIM15 global interrupts	0x0000 00A0
25	32	settable	TIM1_UP/TIM16	TIM1 Update/TIM16 global interrupts	0x0000 00A4
26	33	settable	TIM1_TRG_COM	TIM1 trigger and commutation interrupt	0x0000 00A8
27	34	settable	TIM1_CC	TIM1 capture compare interrupt	0x0000 00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	36	settable	TIM3(3)	TIM3 global interrupt	0x0000 00B4
30	37	settable	-	Reserved	0x0000 00B8
31	38	settable	I2C1_EV	I2C1 event interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I2C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV(4)	I2C2 event interrupt	0x0000 00C4
34	41	settable	I2C2_ER(4)	I2C2 error interrupt	0x0000 00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000 00CC
36	43	settable	SPI2(4)	SPI2 global interrupt	0x0000 00D0
37	44	settable	USART1	USART1 global interrupt	0x0000 00D4
38	45	settable	USART2	USART2 global interrupt	0x0000 00D8
39	46	settable	USART3(4)	USART3 global interrupt	0x0000 00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0
41	48	settable	RTC_ALARM	RTC alarms through EXTI line 18 interrupts	0x0000 00E4
42	49	settable	-	Reserved	0x0000 00E8
43	50	settable	-	Reserved	0x0000 00EC

RM0394 p. 322 Table 46

Nested Vector Interrupt Controller

- On Cortex-M4 the NVIC supports up to 240 IRQs, a Non-Maskable Interrupt, a SysTick timer interrupt, and a number of system exceptions.
- When handling and IRQ, some of the registers are stored on the stack automatically and are automatically restored. This allows exception handlers to be written as normal C functions.
- _____ refers to the fact that we have different priorities and therefore can handle an interrupt with a higher priority in the middle of handling an interrupt of a lower priority.
- _____ refers to the fact that the interrupt service handlers are addresses pointing to functions.

Interrupt Priorities

- Interrupt priority levels allow us to define which interrupts can pre-empt others
- Cortex-M processors support three fixed highest-priority levels and up to 256 level of programmable priority.
  - However, the actual number of available levels is chip dependent since implementing all 256 levels can be costly in terms of power and speed.
- Three negative priorities (hard fault, NMI, and reset) can pre-empt any other exceptions

Interrupt Priorities

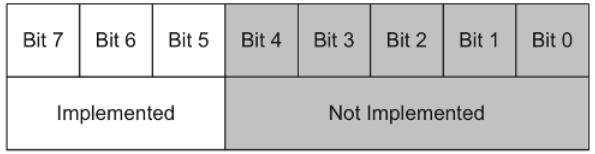


FIGURE 7.2

A priority-level register with 3 bits implemented (8 programmable priority levels)

Figure 7.2 p. 236 The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors

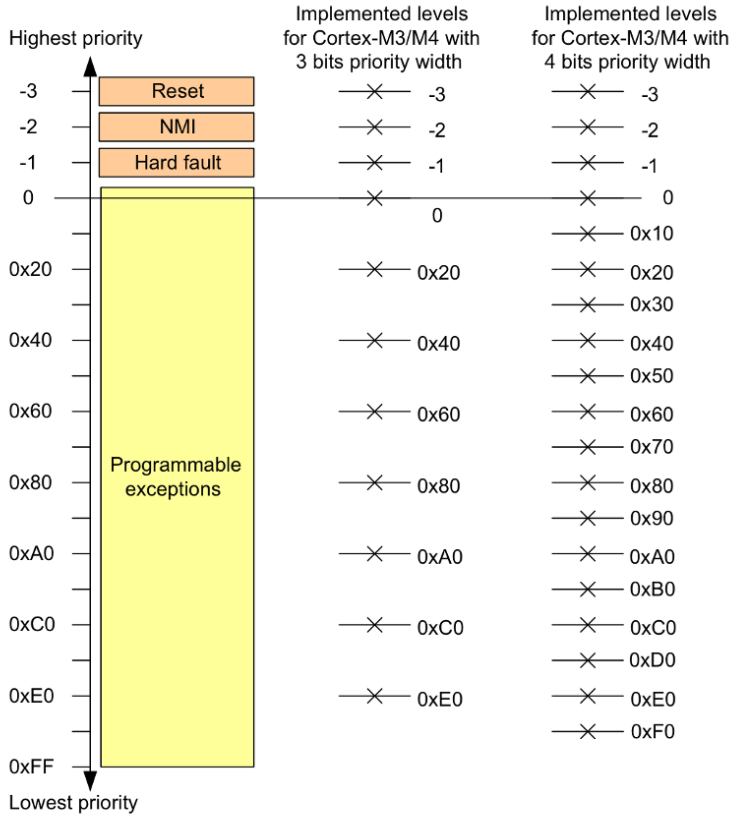


FIGURE 7.4

Available priority levels with 3-bit or 4-bit priority width

Figure 7.4 p. 237 The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors

Exception Definitions

Table 7.3 CMSIS-Core Exception Definitions

Exception Number	Exception Type	CMSIS-Core Enumeration (IRQn)	CMSIS-Core Enumeration Value	Exception Handler Name
1	Reset	-	-	Reset_Handler
2	NMI	NonMaskableInt_IRQn	−14	NMI_Handler
3	Hard Fault	HardFault_IRQn	−13	HardFault_Handler
4	MemManage Fault	MemoryManagement_IRQn	−12	MemManage_Handler
5	Bus Fault	BusFault_IRQn	−11	BusFault_Handler
6	Usage Fault	UsageFault_IRQn	−10	UsageFault_Handler
11	SVC	SVCall_IRQn	−5	SVC_Handler
12	Debug Monitor	DebugMonitor_IRQn	−4	DebugMon_Handler
14	PendSV	PendSV_IRQn	−2	PendSV_Handler
15	SYSTICK	SysTick_IRQn	−1	SysTick_Handler
16	Interrupt #0	(device-specific)	0	(device-specific)
17	Interrupt #1 - #239	(device-specific)	1 to 239	(device-specific)

Table 7.3 p.234 The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors

Interrupt Setup

1. Enable _____.
2. Set the _____ (optional).
3. Enable _____ in the peripheral that triggers the interrupt.
4. Enable the interrupt in the _____.

The name of the ISR needs to match the name used in the vector table so that the linker can place the starting address of the ISR into the vector table correctly.

Handling an interrupt

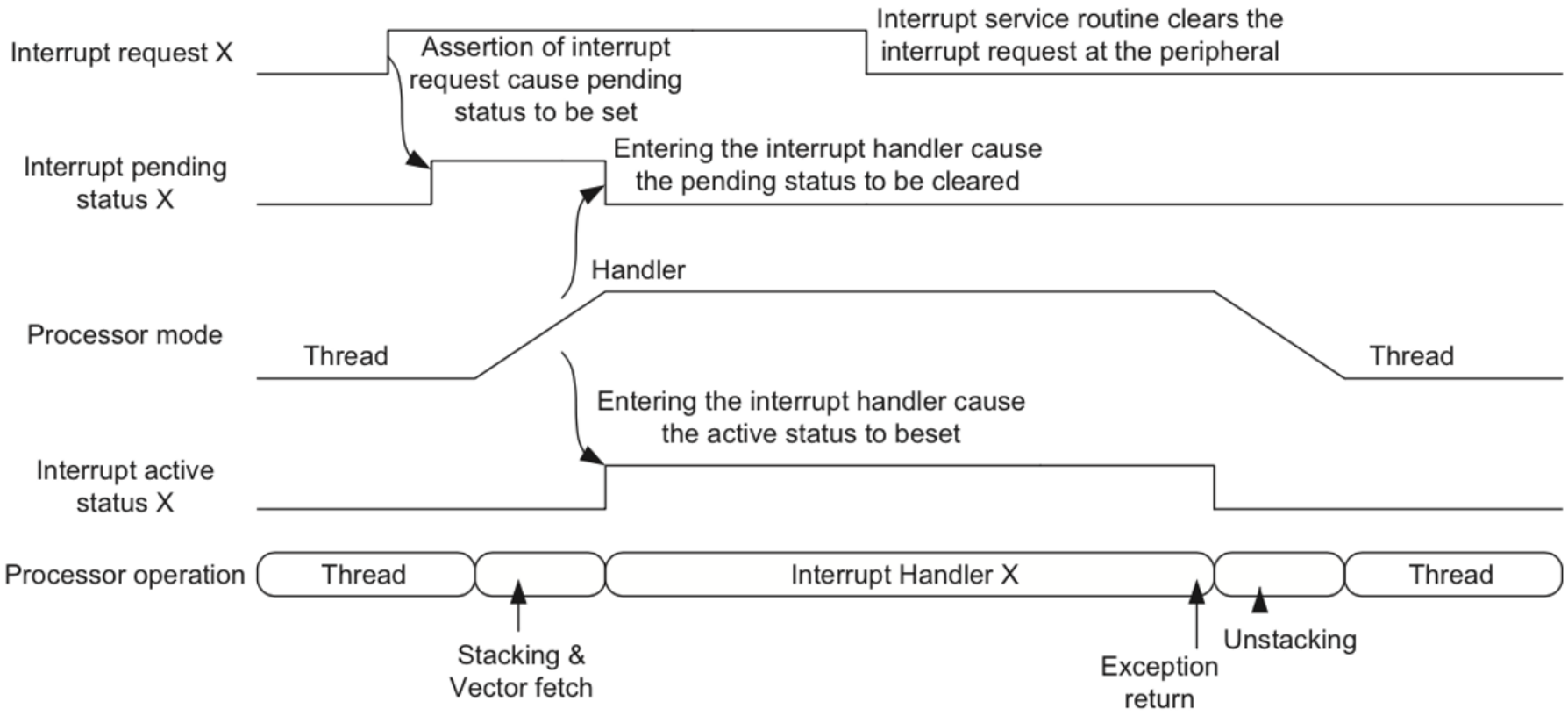


FIGURE 7.14

A simple case of interrupt pending and activation behavior

Relevant Files in Common Microcontroller Software Interface Standard (CMSIS)

`core_cm4.h` - Definitions which are global to the Cortex-M4

- e.g., NVIC_Type which specifies the NVIC registers
- Sidenote: Documentation for this is in the Cortex-M4 user manual, not in the reference manual or datasheet.

`cmsis_gcc.h` - Compiler specific definitions

- e.g., the specific directive syntax necessary to force functions to be inline. `void __enable_irq(void)` and `void __disable_irq(void)` are defined in `cmsis_gcc.h`
- These are compiler specific and use the `cpsie i` (enable) and `cpsid i` disable special assembly instructions.

`stm32l432xx.h` - Device specific configurations.

- e.g., the number of NVIC priority bits

NVIC Memory Location

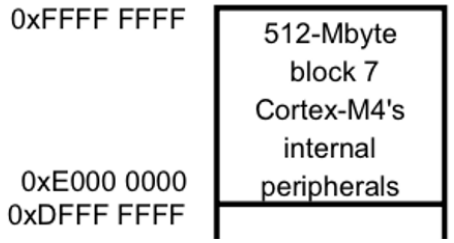
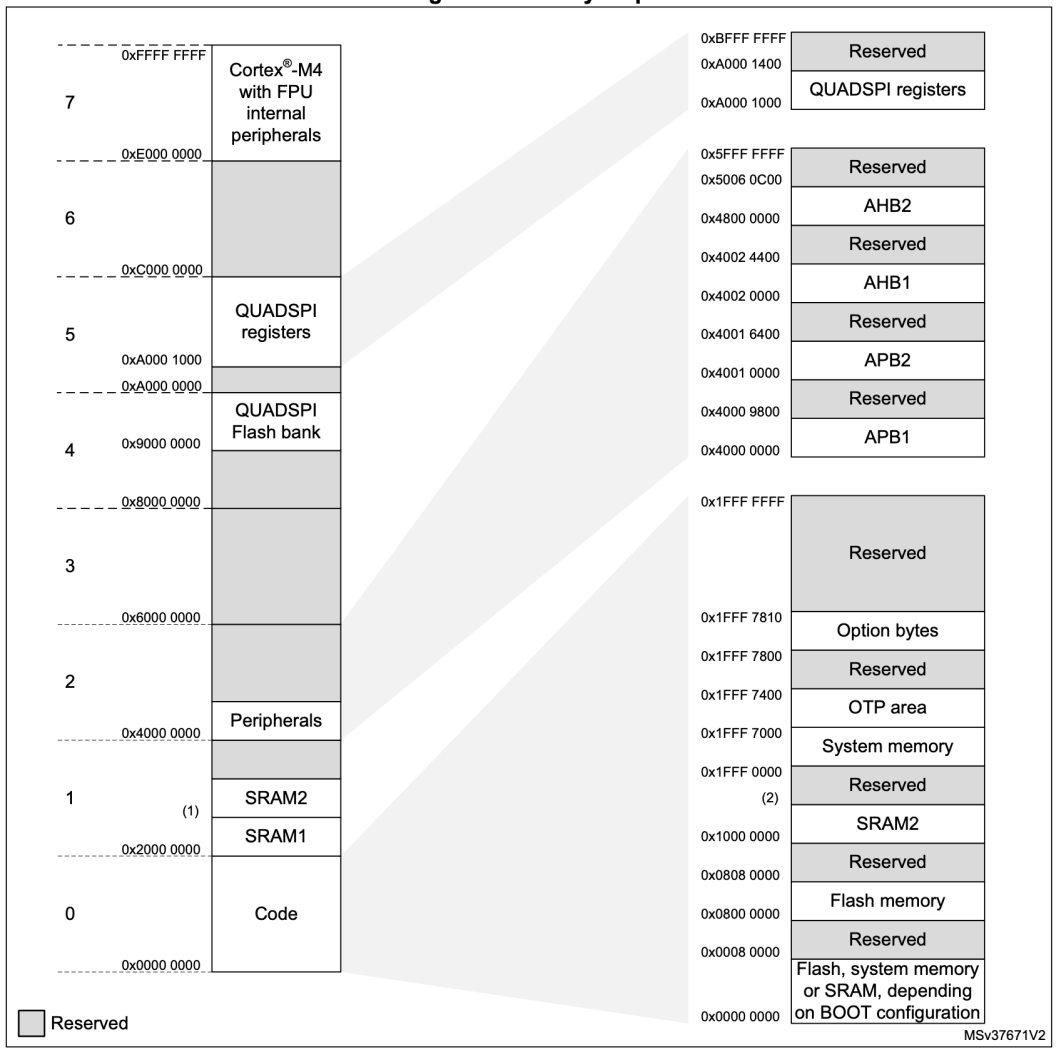


Figure 2. Memory map



RM0394 p. 67

Core Registers for NVIC

6.3 NVIC programmers model

Table 6-1 shows the NVIC registers.

Table 6-1 NVIC registers

Address	Name	Type	Reset	Description
0xE000E004	ICTR	RO	-	*Interrupt Controller Type Register, ICTR*
0xE000E100 - 0xE000E11C	NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0E000xE19C	NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E21C	NVIC_ISPR0 - NVIC_ISPR7	RW	0x00000000	Interrupt Set-Pending Registers
0xE000E280 - 0xE000E29C	NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0 - NVIC_IABR7	RO	0x00000000	Interrupt Active Bit Register
0xE000E400 - 0xE000E41F	NVIC_IPR0 - NVIC_IPR59	RW	0x00000000	Interrupt Priority Register

The following sections describe the NVIC registers whose implementation is specific to this processor. Other registers are described in the *ARMv7M Architecture Reference Manual*.

core_cm4.h

```
403   /**
404     \brief  Structure type to access the Nested Vectored Interrupt Controller (NVIC).
405    */
406   typedef struct
407   {
408     __IOM uint32_t ISER[8U];               /*!< Offset: 0x000 (R/W)  Interrupt Set Enable Register */
409           uint32_t RESERVED0[24U];
410     __IOM uint32_t ICER[8U];               /*!< Offset: 0x080 (R/W)  Interrupt Clear Enable Register */
411           uint32_t RESERVED1[24U];
412     __IOM uint32_t ISPR[8U];               /*!< Offset: 0x100 (R/W)  Interrupt Set Pending Register */
413           uint32_t RESERVED2[24U];
414     __IOM uint32_t ICPR[8U];               /*!< Offset: 0x180 (R/W)  Interrupt Clear Pending Register */
415           uint32_t RESERVED3[24U];
416     __IOM uint32_t IABR[8U];               /*!< Offset: 0x200 (R/W)  Interrupt Active bit Register */
417           uint32_t RESERVED4[56U];
418     __IOM uint8_t  IP[240U];               /*!< Offset: 0x300 (R/W)  Interrupt Priority Register (8Bit wide) */
419           uint32_t RESERVED5[644U];
420     __OM  uint32_t STIR;                   /*!< Offset: 0xE00 ( /W)  Software Trigger Interrupt Register */
421   }  NVIC_Type;
```

cmsis_gcc.h

```
118  /* ##########################  Core Function Access  ########################### */
119  /** \ingroup  CMSIS_Core_FunctionInterface
120      \defgroup CMSIS_Core_RegAccFunctions CMSIS Core Register Access Functions
121    @{
122   */
123
124  /**
125    \brief   Enable IRQ Interrupts
126    \details Enables IRQ interrupts by clearing the I-bit in the CPSR.
127             Can only be executed in Privileged modes.
128   */
129  __STATIC_FORCEINLINE void __enable_irq(void)
130  {
131    __ASM volatile ("cpsie i" : : : "memory");
132  }
133
134
135  /**
136    \brief   Disable IRQ Interrupts
137    \details Disables IRQ interrupts by setting the I-bit in the CPSR.
138             Can only be executed in Privileged modes.
139   */
140  __STATIC_FORCEINLINE void __disable_irq(void)
141  {
142    __ASM volatile ("cpsid i" : : : "memory");
143  }
```

stm32l432xx.h

```
/**
 * @brief STM32L4XX Interrupt Number Definition, according to the selected device
 *        in @ref Library_configuration_section
 */
typedef enum
{
/******  Cortex-M4 Processor Exceptions Numbers ***********************************************************
  NonMaskableInt_IRQn         = -14,    /*!< 2 Cortex-M4 Non Maskable Interrupt
  HardFault_IRQn              = -13,    /*!< 3 Cortex-M4 Hard Fault Interrupt
  MemoryManagement_IRQn       = -12,    /*!< 4 Cortex-M4 Memory Management Interrupt
  BusFault_IRQn               = -11,    /*!< 5 Cortex-M4 Bus Fault Interrupt
  UsageFault_IRQn             = -10,    /*!< 6 Cortex-M4 Usage Fault Interrupt
  SVCall_IRQn                 = -5,     /*!< 11 Cortex-M4 SV Call Interrupt
  DebugMonitor_IRQn           = -4,     /*!< 12 Cortex-M4 Debug Monitor Interrupt
  PendSV_IRQn                 = -2,     /*!< 14 Cortex-M4 Pend SV Interrupt
  SysTick_IRQn                = -1,     /*!< 15 Cortex-M4 System Tick Interrupt
/******  STM32 specific Interrupt Numbers *****************************************************************
  WWDG_IRQn                   = 0,      /*!< Window WatchDog Interrupt
  PVD_PVM_IRQn                = 1,      /*!< PVD/PVM3/PVM4 through EXTI Line detection Interrupts
  TAMP_STAMP_IRQn             = 2,      /*!< Tamper and TimeStamp interrupts through the EXTI line
  RTC_WKUP_IRQn               = 3,      /*!< RTC Wakeup interrupt through the EXTI line

...
} IRQn_Type;
```

Interrupt Activity

Activity: GPIO Pin Interrupts

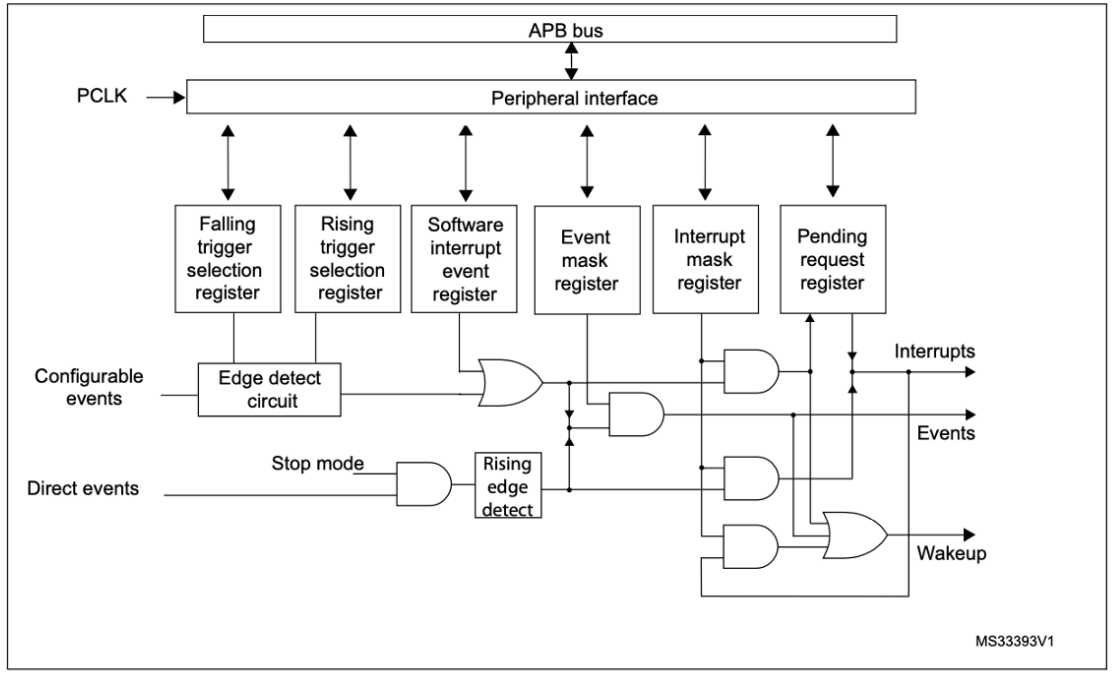
- Download the code from the course Github.
- Build and upload button_polling.c

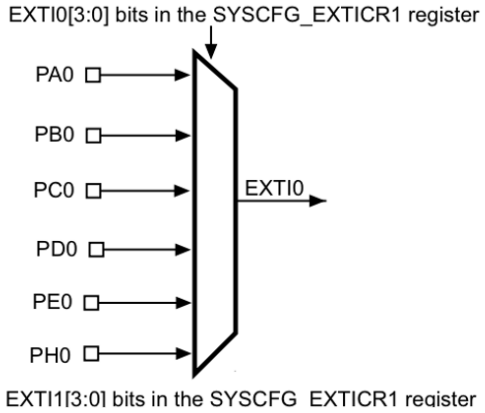
button_polling.c

```c
// button_polling.c
// Josh Brake
// jbrake@hmc.edu
// 10/31/22

/*
  This program polls the user button on the Nucleo-L432KC board and has a
  delay within the main loop to simulate the problems with polling for
  catching events.
*/

#include "main.h"

int main(void) {
    // Enable LED as output
    gpioEnable(GPIO_PORT_B);
    pinMode(LED_PIN, GPIO_OUTPUT);

    // Enable button as input
    gpioEnable(GPIO_PORT_A);
    pinMode(BUTTON_PIN, GPIO_INPUT);
    GPIOA->PUPDR |= _VAL2FLD(GPIO_PUPDR_PUPD2, 0b01); // Set PA2 as pull-up

    // Initialize timer
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
    initTIM(DELAY_TIM);

    int volatile cur_button_state = digitalRead(BUTTON_PIN);
    int volatile led_state = 0;
    int volatile prev_button_state = cur_button_state;

    while(1){
        prev_button_state = cur_button_state;
        cur_button_state = digitalRead(BUTTON_PIN);
        if ((prev_button_state == 1) && (cur_button_state == 0)) {
            led_state = !led_state;
            digitalWrite(LED_PIN, led_state);
        }
        delay_millis(DELAY_TIM, 200);
    }
}
```

External Interrupt/Event Controller (EXTI)

Figure 28. Configurable interrupt/event block diagram

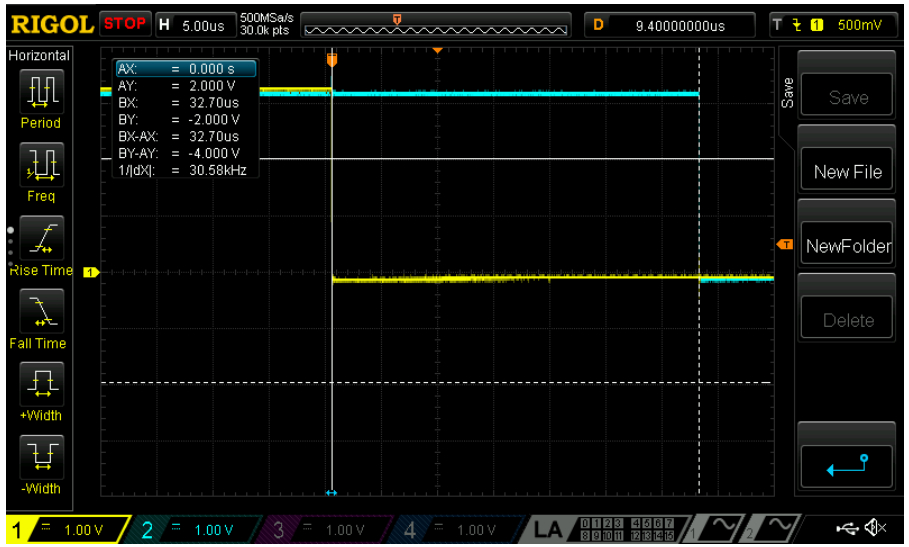


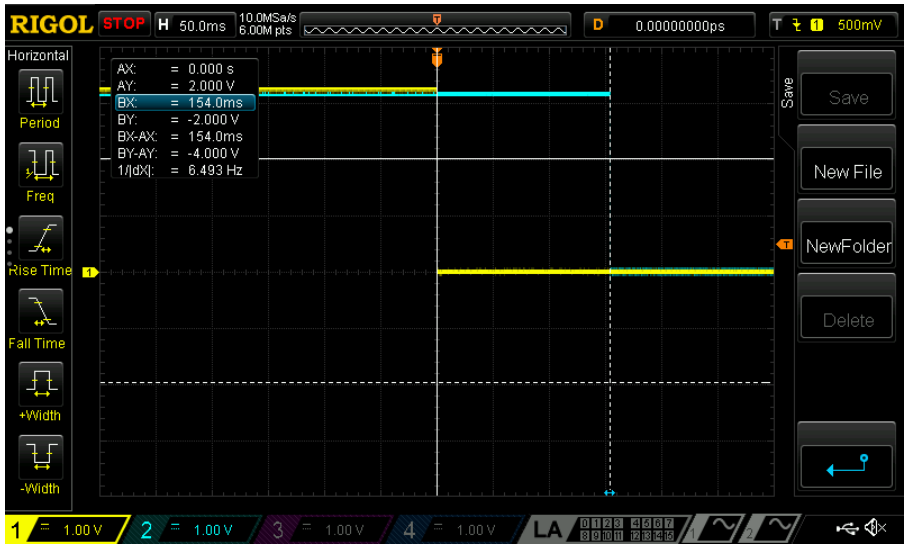


Results

1. Configure EXTI controller
2. Define IRQ handler name
3. Capture trace on oscilloscope demonstrating latency between button press and LED response for three cases:
    - Polling with 200 ms delay
    - Polling with no delay (comment out delay)
    - Interrupt driven

Results: Interrupt vs. Polling with 200 ms Delay





Results: Interrupt vs. Polling with No Delay