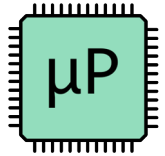
Interrupts

Lecture 12

Microprocessor-based Systems (E155)

Prof. Josh Brake



Learning Objectives

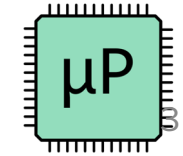
By the end of this lecture you will be able to:

- 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

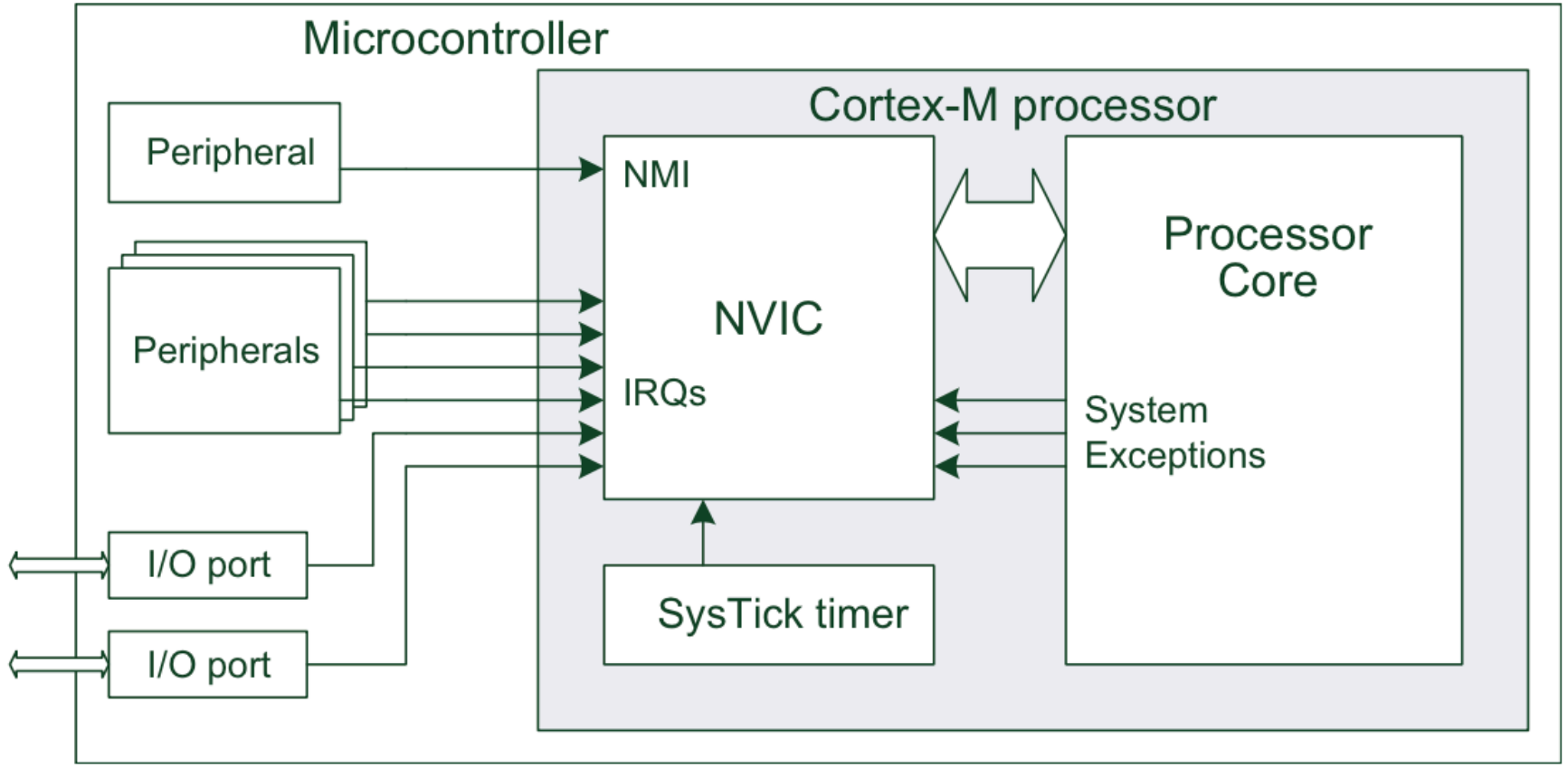


FIGURE 7.1

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 asserts interrupt request
2. Processor suspends currently operating task
3. Processor executes an Interrupt Service Routine (ISR) to service the peripheral and optionally clear the interrupt request
4. Resume previously suspended task.

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.
- "Nested" 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.
- "Vector" refers to the fact that the interrupt service handlers

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

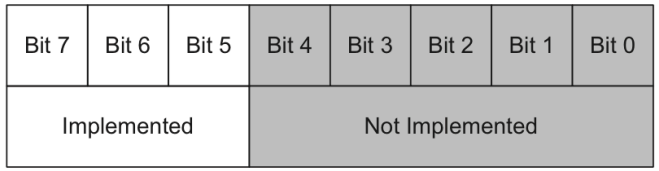


FIGURE 7.2

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*

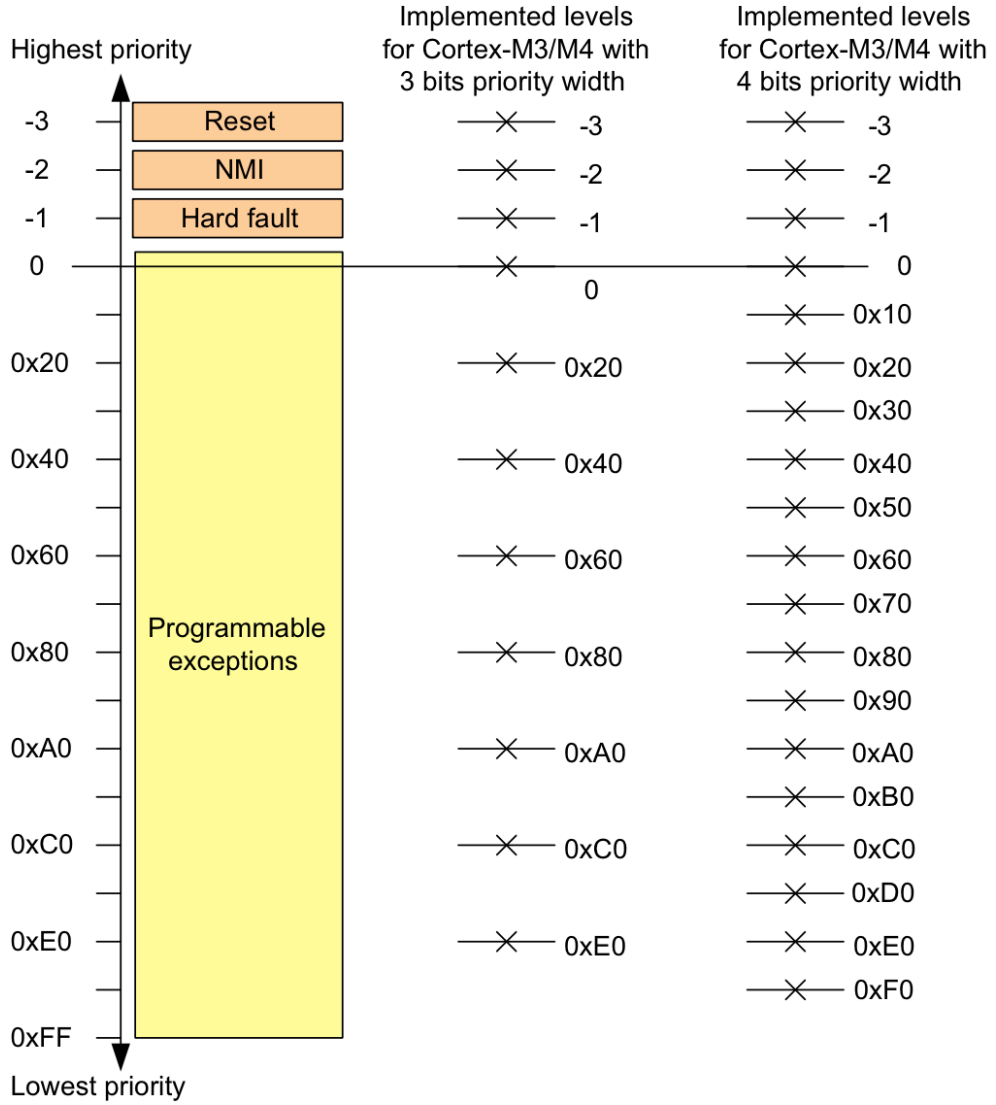


FIGURE 7.4

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 global interrupts
2. Set the priority level (optional)
3. Enable the interrupt generation control in the peripheral that triggers the interrupt
4. Enable the interrupt in the NVIC

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

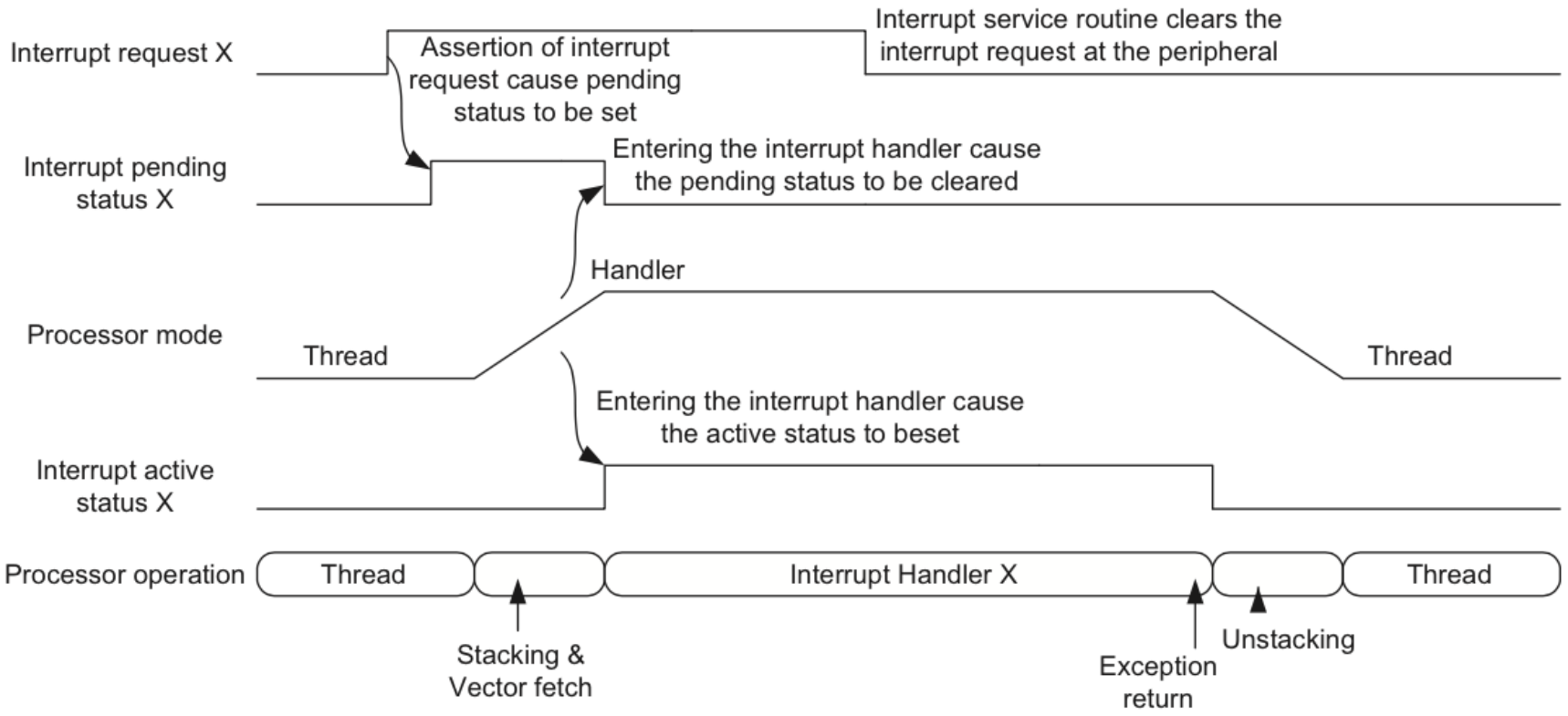


FIGURE 7.14

A simple case of interrupt pending and activation behavior

Relevant Files in 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 STM32F401RE 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.

`stm32f401xe.h` - Device specific configurations.

e.g., the number of NVIC priority bits

NVIC Memory Location

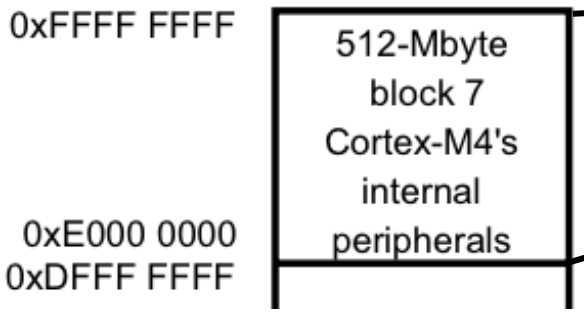
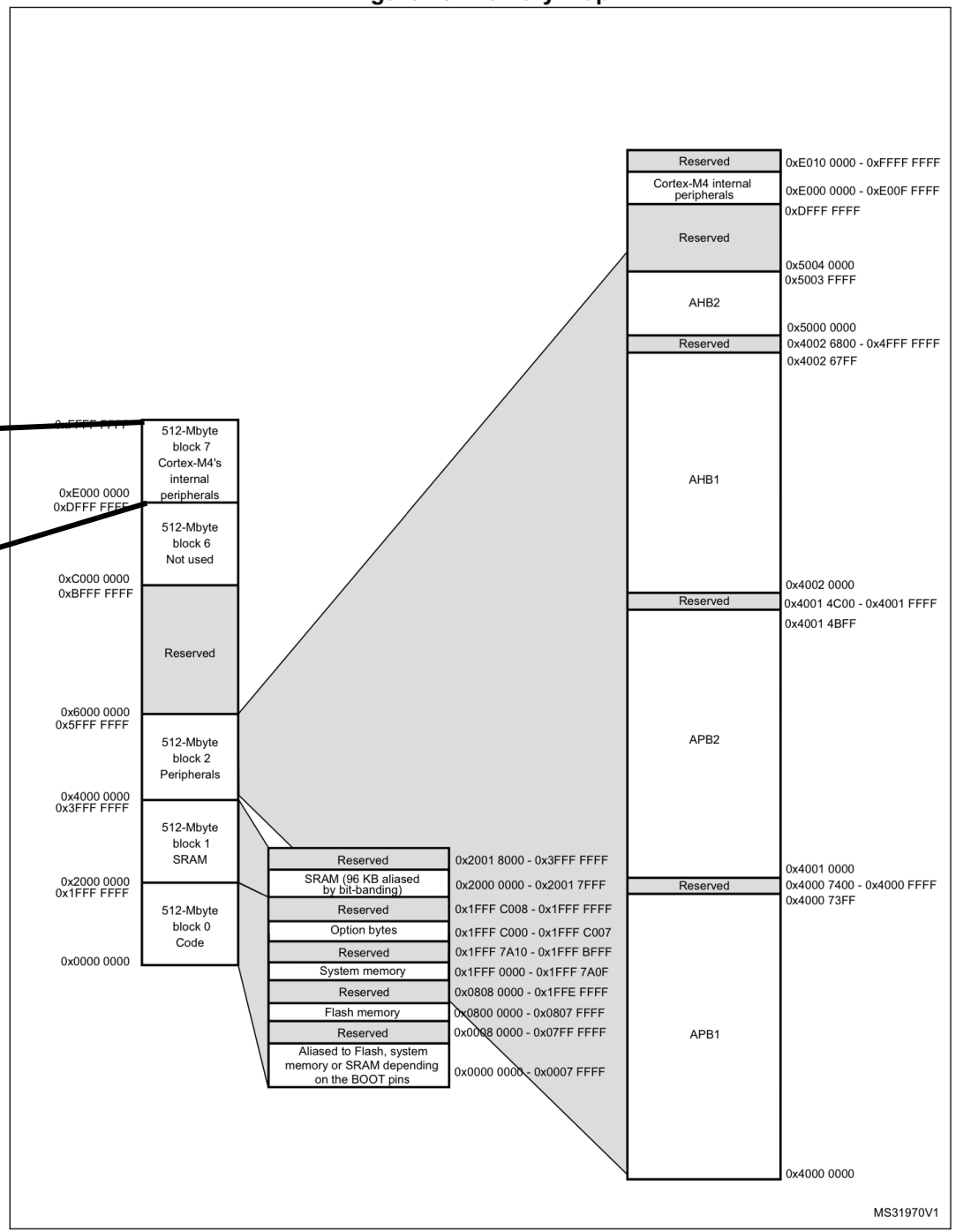


Figure 15. Memory map



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*.

Cortex-M4 Technical Reference Manual, Table 6-1 p. 6-4

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  }
```

```
39   /* CMSIS compiler specific defines */
40   #ifndef   __ASM
41     #define __ASM                                  __asm
42   #endif
43   #ifndef   __INLINE
44     #define __INLINE                               inline
45   #endif
46   #ifndef   __STATIC_INLINE
47     #define __STATIC_INLINE                        static inline
48   #endif
```

stm32f401xe.h

`IRQn_Type` enumerator (enum) is copied into `main.h`

Creates shorthand so we can refer to TIM2_IRQn to return 28 instead of needing to always look it up in the datasheet

```
/**
 * @brief STM32F4XX Interrupt Number Definition, according to the selected device
 *        in @ref Library_configuration_section
 */
typedef enum
{
/******  Cortex-M4 Processor Exceptions Numbers ***********************************************************/
  NonMaskableInt_IRQn        = -14,    /*!< 2 Non Maskable 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_IRQn                   = 1,      /*!< PVD through EXTI Line detection Interrupt               */
  TAMP_STAMP_IRQn            = 2,      /*!< Tamper and TimeStamp interrupts through the EXTI line   */
  RTC_WKUP_IRQn              = 3,      /*!< RTC Wakeup interrupt through the EXTI line              */
  FLASH_IRQn                 = 4,      /*!< FLASH global Interrupt                                  */
  RCC_IRQn                   = 5,      /*!< RCC global Interrupt                                    */
  EXTI0_IRQn                 = 6,      /*!< EXTI Line0 Interrupt                                    */
  EXTI1_IRQn                 = 7,      /*!< EXTI Line1 Interrupt                                    */
  EXTI2_IRQn                 = 8,      /*!< EXTI Line2 Interrupt                                    */
  EXTI3_IRQn                 = 9,      /*!< EXTI Line3 Interrupt                                    */
  EXTI4_IRQn                 = 10,     /*!< EXTI Line4 Interrupt                                    */
  DMA1_Stream0_IRQn          = 11,     /*!< DMA1 Stream 0 global Interrupt                          */
  DMA1_Stream1_IRQn          = 12,     /*!< DMA1 Stream 1 global Interrupt                          */
  DMA1_Stream2_IRQn          = 13,     /*!< DMA1 Stream 2 global Interrupt                          */
  DMA1_Stream3_IRQn          = 14,     /*!< DMA1 Stream 3 global Interrupt                          */
  DMA1_Stream4_IRQn          = 15,     /*!< DMA1 Stream 4 global Interrupt                          */
  DMA1_Stream5_IRQn          = 16,     /*!< DMA1 Stream 5 global Interrupt                          */
  DMA1_Stream6_IRQn          = 17,     /*!< DMA1 Stream 6 global Interrupt                          */
  ADC_IRQn                   = 18,     /*!< ADC1, ADC2 and ADC3 global Interrupts                   */
  EXTI9_5_IRQn               = 23,     /*!< External Line[9:5] Interrupts                           */
  TIM1_BRK_TIM9_IRQn         = 24,     /*!< TIM1 Break interrupt and TIM9 global interrupt          */
  TIM1_UP_TIM10_IRQn         = 25,     /*!< TIM1 Update Interrupt and TIM10 global interrupt        */
  TIM1_TRG_COM_TIM11_IRQn    = 26,     /*!< TIM1 Trigger and Commutation Interrupt and TIM11 global interrupt */
  TIM1_CC_IRQn               = 27,     /*!< TIM1 Capture Compare Interrupt                          */
  TIM2_IRQn                  = 28,     /*!< TIM2 global Interrupt                                   */
  TIM3_IRQn                  = 29,     /*!< TIM3 global Interrupt                                   */
  TIM4_IRQn                  = 30,     /*!< TIM4 global Interrupt                                   */
  I2C1_EV_IRQn               = 31,     /*!< I2C1 Event Interrupt                                    */
  I2C1_ER_IRQn               = 32,     /*!< I2C1 Error Interrupt                                    */
  I2C2_EV_IRQn               = 33,     /*!< I2C2 Event Interrupt                                    */
  I2C2_ER_IRQn               = 34,     /*!< I2C2 Error Interrupt                                    */
  SPI1_IRQn                  = 35,     /*!< SPI1 global Interrupt                                   */
  SPI2_IRQn                  = 36,     /*!< SPI2 global Interrupt                                   */
  USART1_IRQn                = 37,     /*!< USART1 global Interrupt                                 */
  USART2_IRQn                = 38,     /*!< USART2 global Interrupt                                 */
  EXTI15_10_IRQn             = 40,     /*!< External Line[15:10] Interrupts                         */
  RTC_Alarm_IRQn             = 41,     /*!< RTC Alarm (A and B) through EXTI Line Interrupt         */
  OTG_FS_WKUP_IRQn           = 42,     /*!< USB OTG FS Wakeup through EXTI line interrupt           */
  DMA1_Stream7_IRQn          = 47,     /*!< DMA1 Stream7 Interrupt                                  */
  SDIO_IRQn                  = 49,     /*!< SDIO global Interrupt                                   */
  TIM5_IRQn                  = 50,     /*!< TIM5 global Interrupt                                   */
  SPI3_IRQn                  = 51,     /*!< SPI3 global Interrupt                                   */
  DMA2_Stream0_IRQn          = 56,     /*!< DMA2 Stream 0 global Interrupt                          */
  DMA2_Stream1_IRQn          = 57,     /*!< DMA2 Stream 1 global Interrupt                          */
  DMA2_Stream2_IRQn          = 58,     /*!< DMA2 Stream 2 global Interrupt                          */
  DMA2_Stream3_IRQn          = 59,     /*!< DMA2 Stream 3 global Interrupt                          */
  DMA2_Stream4_IRQn          = 60,     /*!< DMA2 Stream 4 global Interrupt                          */
  OTG_FS_IRQn                = 67,     /*!< USB OTG FS global Interrupt                             */
  DMA2_Stream5_IRQn          = 68,     /*!< DMA2 Stream 5 global interrupt                          */
  DMA2_Stream6_IRQn          = 69,     /*!< DMA2 Stream 6 global interrupt                          */
  DMA2_Stream7_IRQn          = 70,     /*!< DMA2 Stream 7 global interrupt                          */
  USART6_IRQn                = 71,     /*!< USART6 global interrupt                                 */
  I2C3_EV_IRQn               = 72,     /*!< I2C3 event interrupt                                    */
  I2C3_ER_IRQn               = 73,     /*!< I2C3 error interrupt                                    */
  FPU_IRQn                   = 81,     /*!< FPU global interrupt                                    */
  SPI4_IRQn                  = 84      /*!< SPI4 global Interrupt                                    */
} IRQn_Type;
```

Activity: GPIO Pin Interrupts

- Download the code from the course Github (https://github.com/joshbrake/E155_FA2020/tree/master/L12)
- Create new PlatformIO project with CMSIS framework
  - Replace contents of platformio.ini with those from the Github repo platformio.ini
  - Move contents of lib into your PIO lib folder
  - Move demo_src/src into your PIO src folder
  - Move demo_src/demo into your PIO project folder
- Build upload `main_button_polling_solution.c`

Project Setup

platformio.ini

```
[env:genericSTM32F401RE]
platform = ststm32
board = genericSTM32F401RE
framework = cmsis
debug_tool = stlink
upload_protocol = stlink
lib_extra_dirs = ./lib/
    ; Put path to libraries here
```

STM32F401RE.h

```
// STM32F401RE.h
// Header to include all other STM32F401RE libraries.

#ifndef STM32F4_H
#define STM32F4_H

#include <stdint.h>

// Include other peripheral libraries

#include "STM32F401RE_GPIO.h"
#include "STM32F401RE_FLASH.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_SPI.h"
#include "STM32F401RE_TIM.h"


#endif
```

main.h

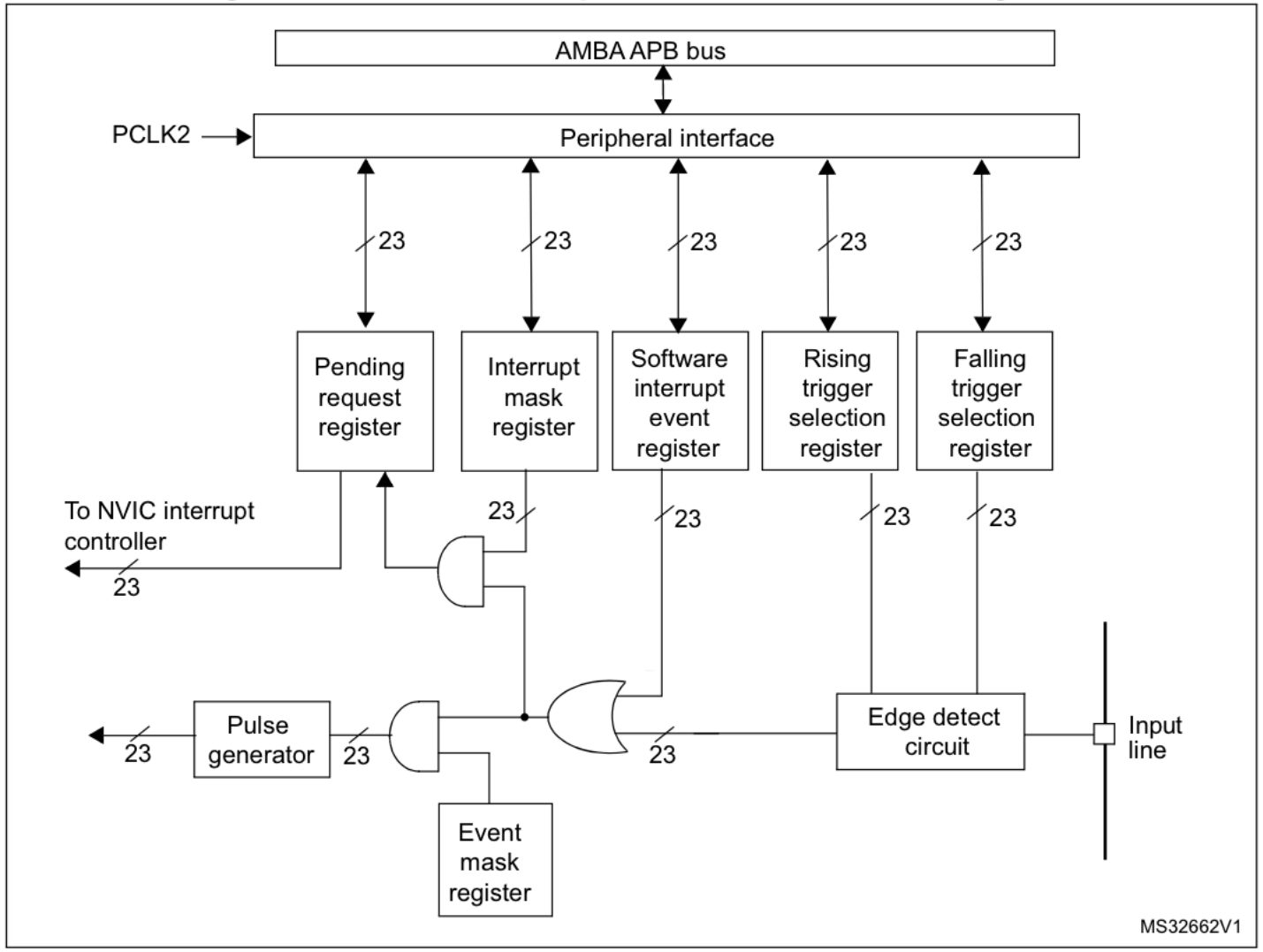
```
// main.h
// Josh Brake
// jbrake@hmc.edu
// 9/30/20

#ifndef MAIN_H
#define MAIN_H

#include "STM32F401RE.h"

//////////////////////////////////////////////////////////////////////////////////
// Custom defines
//////////////////////////////////////////////////////////////////////////////////

#define LED_PIN 5
#define BUTTON_PIN 13 // PC13
#define DELAY_TIM TIM2

#define NVIC_ISER0 ((uint32_t *) 0xE000E100UL)
#define NVIC_ISER1 ((uint32_t *) 0xE000E104UL)
#define SYSCFG_EXTICR4 ((uint32_t *) (0x40013800UL + 0x14UL))

typedef struct {
    volatile uint32_t IMR;
    volatile uint32_t EMR;
    volatile uint32_t RTSR;
    volatile uint32_t FTSR;
    volatile uint32_t SWIER;
    volatile uint32_t PR;
}EXTI_TypeDef;
```

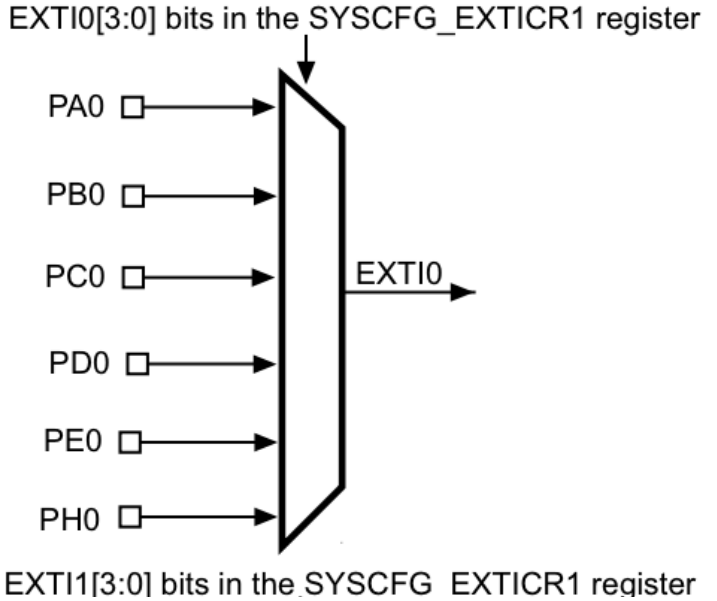
main_button_polling_solution.c

```c
// main_button_polling_solution.c
// Josh Brake
// jbrake@hmc.edu
// 9/30/20

#include "main.h"

int main(void) {
    configureFlash();
    configureClock();

    // Enable LED as output
    RCC->AHB1ENR.GPIOAEN = 1;
    pinMode(GPIOA, LED_PIN, GPIO_OUTPUT);

    // Enable button as input
    RCC->AHB1ENR.GPIOCEN = 1;
    pinMode(GPIOC, BUTTON_PIN, GPIO_INPUT);

    // Initialize timer
    RCC->APB1ENR |= (1 << 0); // TIM2EN
    initTIM(DELAY_TIM);

    uint8_t volatile cur_button_state = digitalRead(GPIOC, BUTTON_PIN);
    uint8_t volatile led_state = 0;
    uint8_t volatile prev_button_state = cur_button_state;

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

```

External Interrupt/Event Controller (EXTI)

Figure 29. External interrupt/event controller block diagram





Your Task

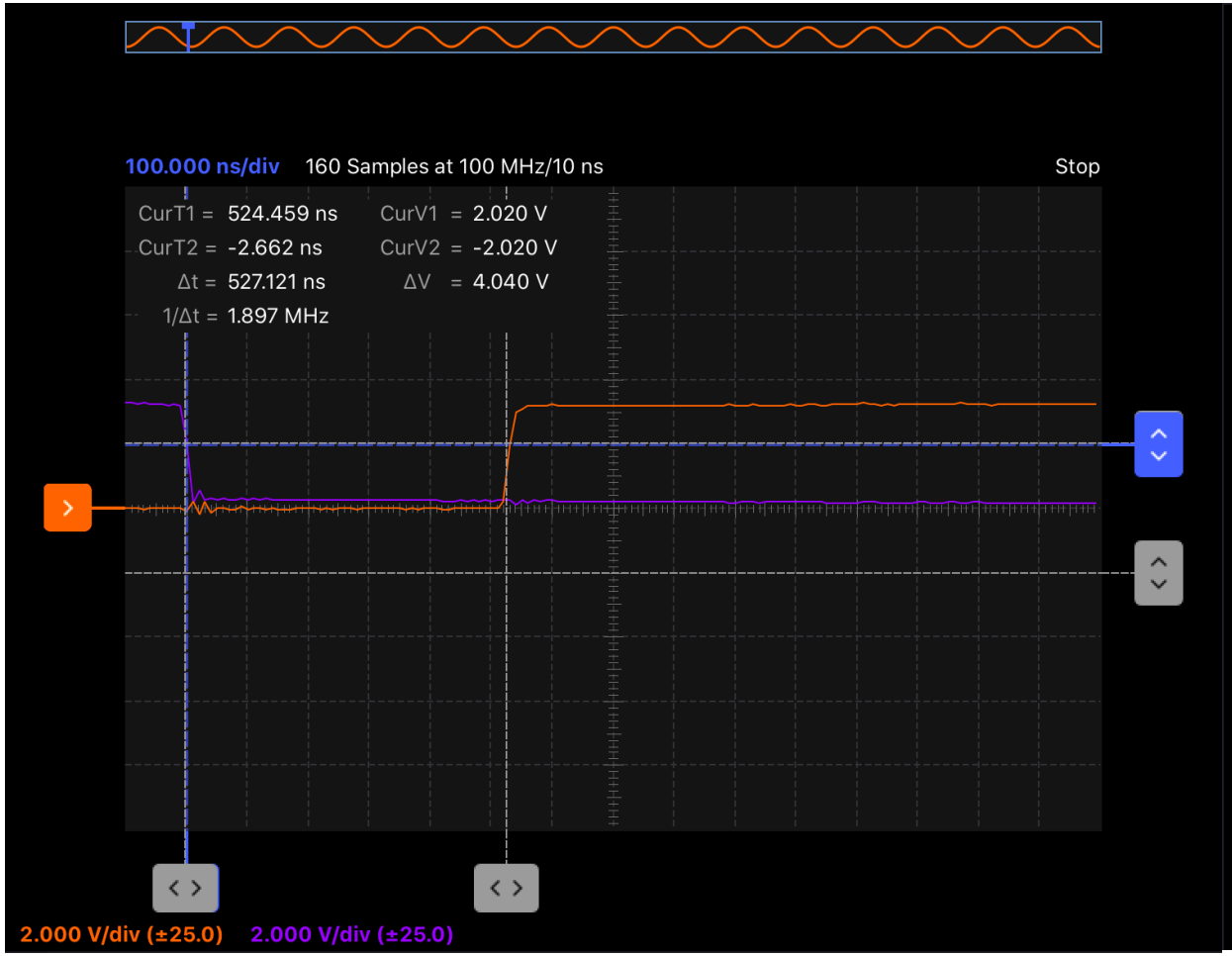
Configure button input as interrupt

- Configure EXTI controller
- Define IRQ handler name
- Upload test the response time compared to polling using Scopy

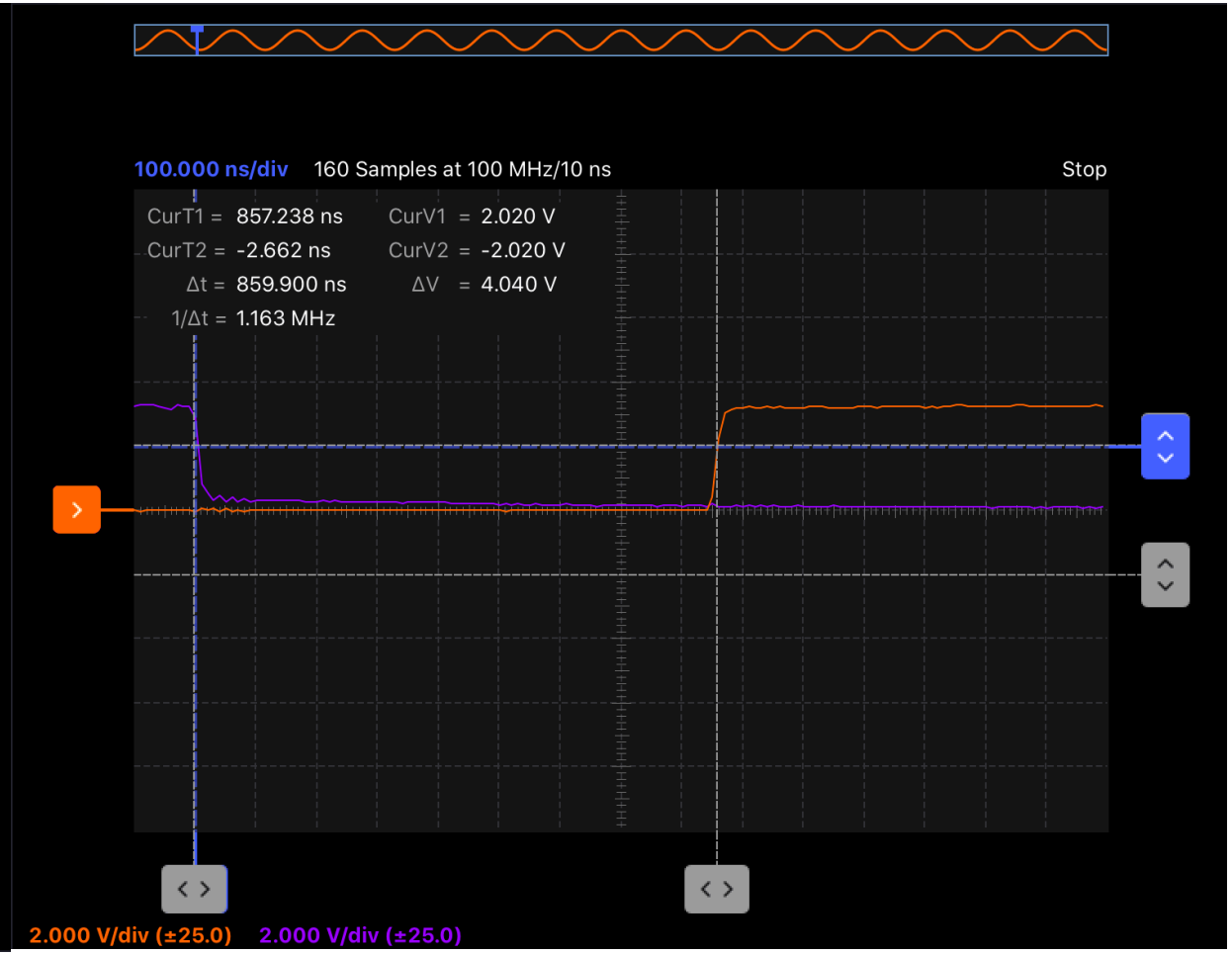
```c
// main_button_interrupt.c
// Josh Brake
// jbrake@hmc.edu
// 9/30/20

#include "main.h"

int main(void) {
    configureFlash();
    configureClock();

    // Enable LED as output
    RCC->AHB1ENR.GPIOAEN = 1;
    pinMode(GPIOA, LED_PIN, GPIO_OUTPUT);

    // Enable button as input
    RCC->AHB1ENR.GPIOCEN = 1;
    pinMode(GPIOC, BUTTON_PIN, GPIO_INPUT);

    // Initialize timer
    RCC->APB1ENR |= (1 << 0); // TIM2EN
    initTIM(DELAY_TIM);


    // TODO
    // 1. Enable SYSCFG clock domain in RCC
    // 2. Set EXTICR4 for PC13

    // Enable interrupts globally
    __enable_irq();

    // TODO: Configure interrupt for falling edge of GPIO PC13
    // 1. Configure mask bit
    // 2. Disable rising edge trigger
    // 3. Enable falling edge trigger
    // 4. Turn on EXTI interrupt in NVIC_ISER1

    while(1){
        delay_millis(TIM2, 200);
    }

}

// TODO: What is the right name for the IRQHandler for PC13?
void XXXXXX(void){
    // Check that the button EXTI_13 was what triggered our interrupt
    if (EXTI->PR & (1 << BUTTON_PIN)){
        // If so, clear the interrupt
        EXTI->PR |= (1 << BUTTON_PIN);

        // Then toggle the LED
        togglePin(GPIOA, LED_PIN);

    }
}
```

Results

Interrupt – 527 ns



Polling (with 0 delay in while (859 ns)



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

Up Next

- Monday: Project Kickoff!
- Wednesday: Digital Signal Processing
- Lab 6: Serial temperature sensor

Lecture Feedback

- What is the most important thing you learned in class today?
- What point was most unclear from lecture today?

https://forms.gle/Ay6MkpZ6x3xsW2Eb8