EE 14: Interrupts + SysTick

Steven Bell
February 2025

Tufts UNIVERSITY

By the end of class today, you should be able to:

- Explain what an "interrupt" is, and why it is useful.
- Describe the steps that happen when an interrupt event occurs

Our normal flow of activity

Our code usually looks something like this:

```
// set up stuff
while(true) {
    // Read inputs
    // Update state machine
    // Set outputs
}
```

This main loop might take a while - what if I need to respond to an event really fast (say, < 100µs)?

How to respond quickly

How can we have fast (low-latency) response to events?

Let's use a GPIO pin going high as an example

1) Use a really tight loop:

```
while(get_gpio_value() == 0) { }
respond_to_event()
```

This repeated checking is called **polling.**

The loop is called a **spin-wait**.

+ Latency is a couple of clock cycles (whatever it takes to read the GPIO)
- But the processor is spending 100% of its time on this!

But what if the processor can't spin-wait?

We have other things to tend to; how can we maintain a fast response?

Interrupts provide a way for events to interrupt the normal flow of execution, and run some code immediately in response.

How do interrupts work?

Dedicated hardware support in the processor

When an interrupt event occurs, the processor:

- Saves the current state (current PC, register values)
- Jumps to a special function in the code (the **interrupt handler**)
- When the interrupt handler returns, resumes where it left off

How do interrupts work?

the Nested Vectored Interrupt Controller (**NVIC**)

Dedicated hardware support in the processor

aka Interrupt Request (**IRQ**)

When an interrupt event occurs, the processor:

- Saves the current state (current PC, register values)
- Jumps to a special function in the code (the **interrupt handler**)
- When the interrupt handler returns, resumes where it left off

aka Interrupt Service Routine (**ISR**)

Time between event occuring and executing the ISR is the **interrupt latency.**

Interrupt vectors

How does the CPU know where this special function is?

We could just put it in a constant known location (e.g., 0x08000100)

But what if that location doesn't have enough space? (or too much?)

Instead, just put a **function pointer** in a known location.

Interrupt vector *table*

But why stop at one? We could have many interrupts, each with their own interrupt handler.

The whole collection of function pointers is the interrupt vector table.

STM32L4 vector table

Has 67 interrupt channels!

Table 51. STM32L41xxx/42xxx/43xxx/44xxx/45xxx/46xxx vector table

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-	-1	fixed	HardFault	All classes of fault	0x0000 000C
-	0	settable	MemManage	Memory management	0x0000 0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C - 0x0000 0028
-	3	settable	SVCall	System service call via SWI instruction	0x0000 002C
-	4	settable	Debug	Monitor	0x0000 0030
-	-	-	-	Reserved	0x0000 0034
-	5	settable	PendSV	Pendable request for system service	0x0000 0038
-	6	settable	SysTick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD_PVM	PVD/PVM1/PVM2(1)/PVM3/PVM4 through EXTI lines 16/35/36/37/38 interrupts	0x0000 0044
2	9	settable	RTC_TAMP_STAMP /CSS_LSE	RTC Tamper or TimeStamp /CSS on LSE through EXTI line 19 interrupts	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup timer through EXTI line 20 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

Some familiar stuff in here…

GPIO is bundled into EXTI 15:0,
- 0-4 each have their own IRQ
- 9:5 is one IRQ
- 15:10 is one IRQ

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

Multiple sources

If there are multiple potential sources, the ISR needs to check which one triggered the interrupt.

13.5.6 Pending register 1 (EXTI_PR1)

Address offset: 0x14

Reset value: 0x0000 0000

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.	PIF22	PIF21	PIF20	PIF19	PIF18	Res.	PIF16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1		rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIF15	PIF14	PIF13	PIF12	PIF11	PIF10	PIF9	PIF8	PIF7	PIF6	PIF5	PIF4	PIF3	PIF2	PIF1	PIF0
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 31:23 Reserved, must be kept at reset value.

Bits 22:18 **PIFx:** Pending interrupt flag on line x (x = 22 to 18)

0: No trigger request occurred

1: Selected trigger request occurred

This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a ‘1’ to the bit.

How to use an interrupt

1) Configure the peripheral to generate interrupts
2) Write an appropriate interrupt handler
3) Put a pointer to the interrupt handler into the interrupt vector table

   CMSIS / PlatformIO does this for you, if you name your function right
4) Enable the interrupt in the NVIC

SysTick Timer

Like general-purpose timers, but:

- It only counts down
- When it hits 0, it triggers an interrupt
- And starts over at the reload value

SysTick Timer

Like general-purpose timers, but:

- It only counts down
- When it hits 0, it triggers an interrupt
- And starts over at the reload value

Set this 24-bit value to
control rate of interrupts!