Using External Interrupts With The STM32F051C6 Microcontroller

1. The need for interrupts

Interrupts help improve code execution efficiency by running code dependent on events only when the event occurs rather than polling the microcontroller until the event occurs.

Using an interrupt will pause execution of the sequential code at any point to handle the interrupt code(Interrupt Service Routine or ISR) then resume the sequential code after handling the interrupt code, so, it will be considerate of the microcontroller's efficiency to make the interrupt code as short as possible.

2. How STM32F051C6 External Interrupts Work

All interrupts are prioritised and handled by the Nested-Vectored Interrupt Controller(NVIC). In an example of a push-button interrupt, when a button on pin PA3 is pressed an electrical signal called 'interrupt request EXTI3', is generated by the hardware. Each ISR is assigned an 8-bit unsigned integer(called IQRn), which is used to identify the ISR in the Interrupt Vector Table. When the NVIC receives interrupt request EXTI3, it forces the microprocessor(Cortex-M0) to jump to and execute the corresponding ISR, specifically a function called EXTI3_IQRHandler from the Interrupt Vector Table.
See Figure 1 below.

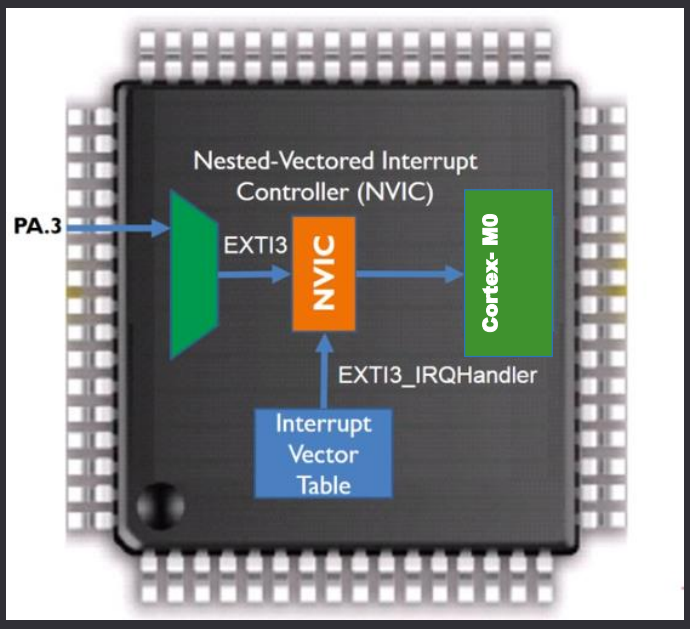

Figure 1 How External Interrupts are handled by the STM32F051C6 microcontroller
Adopted from Dr. Yifeng Zhu's YouTube video on External Interrupts

If two or more interrupts signals are fired at the same time, they can be assigned priority to establish which one is handled first. However, only a single pin on each extended interrupts and events controller(EXTI) multiplexer(mux) can be used to generate an external interrupt, see Figure 2 below.

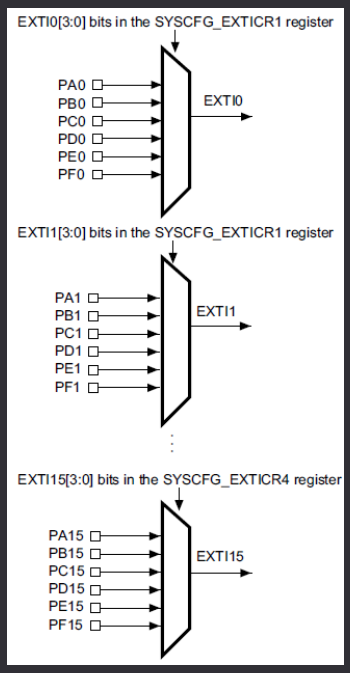

Figure 2a

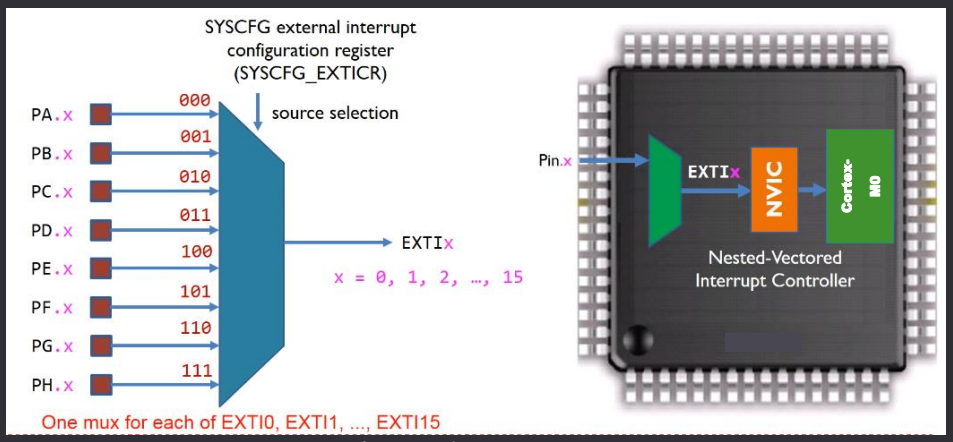

Figure 2b

Figure 2 (a) EXTI multiplexers (b) EXTI signal route from mux to the processor

Figure 2(b) adopted from Dr. Yifeng Zhu's YouTube video on External Interrupts

Interrupts assigned larger priority values will be handled later than interrupts assigned smaller priority values. Therefore, smaller priority values represent higher urgency.

priority_number = 4-bit unsigned integer

3. STM32F051C6 Interrupt Registers

NVIC_EnableIQR(EXTIx_IQRn or IQRn) NVIX_DisableIQR(EXTIx_IQRn or IQRn) XXX Interrupt Set Enable Register(ISER) XXX Interrupt Clear Enable Register(ICER)	To enable/disable peripheral or external interrupts NVIC -> ISER[EXTIx_IQRn or IQRn >>…] = … NVIC -> ICER[EXTIx_IQRn or IQRn > …] = …
Interrupt Priority Register(IPR)	To assign priority to interrupts NVIC_SetPriority(IQRn, priority_number) NVIC -> IPR[EXTIx_IQRn or IQRn] = …
APB2ENR	To enable clock for SYSCFG RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN
SYSCFG_EXTCR[x]	To select which pin to use on a particular mux to generate EXTI, i.e connect line to GPI of port X SYSCFG -> CR[x] |= SYSCFG_CR1_EXTIx_PX
Rising Trigger Selection Register(RTSR)	Enable/Disable rising edge trigger EXTI -> RTSR1 … EXTI_RTSR1_TRx
Interrupt Mask Register(IMR)	To mask/unmask interrupts EXTI -> IMR1 … EXTI_IMR1_

4. Interrupts code for STM32F051C6

4.1 Enable clock for GPIO port and SYSCFG

```
RCC -> AHBENR |= RCC_AHBEENR_GPIOxEN;
RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN;
```

4.2 EXTI source selection

```
/* Setup selected pin */
GPIO… -> MODER &= ~( GPIO_MODER_MODERx );
```

00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

Pull line LOW	GPIO… -> PUPDR |= GPIO_PUPDR_PUPDRx_1;
Pull line HIGH	GPIO… -> PUPDR |= GPIO_PUPDR_PUPDRx_0;
Line with external PU or PD resistor	GPIO… -> PUPDR &= ~GPIO_PUPDR_PUPDRx;

```
/* Select interrupt source for selected pin */
SYSCFG -> EXTICR[y-1] &= ~SYSCFG_EXTICRy_EXTIx;  // ensure no pin is selected
SYSCFG -> EXTICR[y-1] |= SYSCFG_EXTICRy_EXTIx_P(A,B,C,D,E,F);  // select pin of choice
```

PIN NUMBER (x)	CONTROL REGISTER NUMBER (y)	SYSCGF_EXTICR[y-1] value	Bitmask value
0 to 3	1	PAx : x000	SYSCFG_EXTICRy_EXTIx_PA
4 to 7	2	PBx : x001	SYSCFG_EXTICRy_EXTIx_PB
8 to 11	3	PCx : x010	SYSCFG_EXTICRy_EXTIx_PC
12 to 15	4	PDx : x011	SYSCFG_EXTICRy_EXTIx_PD
		PEx : x100	SYSCFG_EXTICRy_EXTIx_PE
		PFx : x101	SYSCFG_EXTICRy_EXTIx_PF

4.3 Unmask interrupt request on selected EXTI source line(pin x) so that interrupt can be generated

```
EXTI -> IMR |= EXTI_IMR_MRx;
```

4.4 Edge trigger selection

For HIGH lines(e.g normally closed button)	Use Falling Edge trigger
EXTI -> FTSR |= EXTI_FTSR_TRx;	
For LOW lines (e.g normally open button)	Use Rising Edge trigger
EXTI -> RTSR |= EXTI_RTSR_TRx;	

4.5 Enable EXTIx interrupt

```
NVIC_EnableIQR(EXTI…IQRn);
```

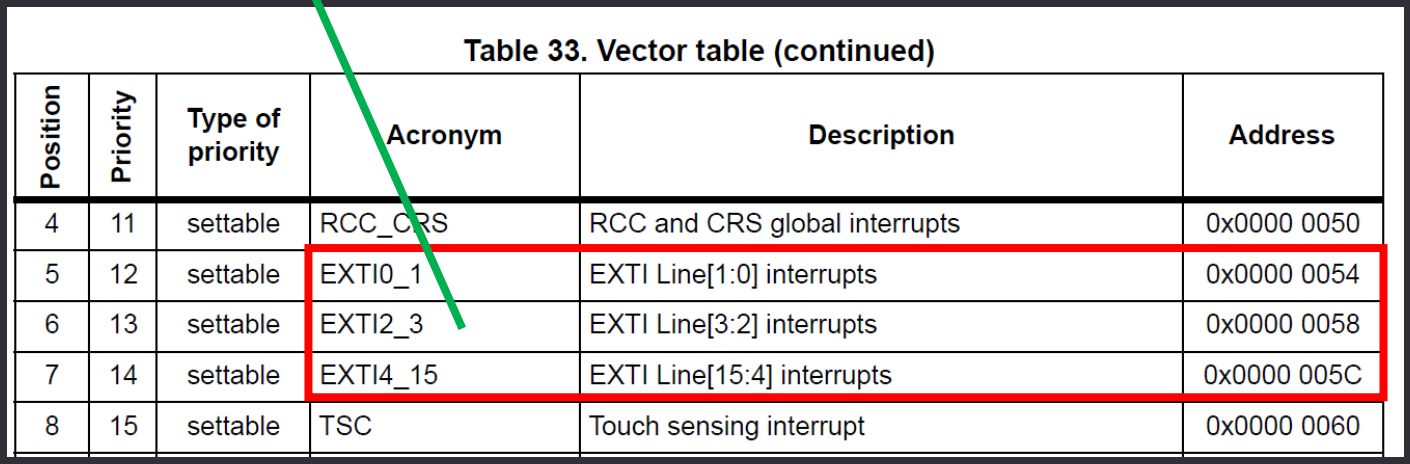

Table 33. Vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
4	11	settable	RCC_CRS	RCC and CRS global interrupts	0x0000 0050
5	12	settable	EXTI0_1	EXTI Line[1:0] interrupts	0x0000 0054
6	13	settable	EXTI2_3	EXTI Line[3:2] interrupts	0x0000 0058
7	14	settable	EXTI4_15	EXTI Line[15:4] interrupts	0x0000 005C
8	15	settable	TSC	Touch sensing interrupt	0x0000 0060

4.6 Write ISR

```
void EXTI…IQRHandler(void){
      //ISR code
      …
      // Clear interrupt pending bit to indicate interrupt request has occured
      EXTI -> PR |= EXTI_PR_PRx;
}
```