Обработка нажатия кнопки

Мы уже познакомились с понятием прерывания (и даже использовали таблицу векторов прерываний) для реализации задержки с SysTick. Теперь же нам предстоит работать с кнопкой, и если с точки зрения программной реализации нам всё понятно — действия аналогичны тому, как мы работали со светодиодом (только в этом случае мы не пишем в `ODR`, а читаем из `IDR` — если вы делаете это в основном цикле), — то с прерываниями не всё так просто. Как отмечалось раньше, они бывают двух видов: внутренние и внешние по отношению к ядру. Так как SysTick — составная часть ядра, то и прерывание от него реализовано по-другому (вы включаете его непосредственно из регистров SysTick). В случае с GPIO (или любой другой периферией) это происходит через модуль внешних прерываний EXTI (англ. external interrupt). Но перед тем, как мы начнем разбираться с ним, давайте настроим ножку, используя библиотеку StdPeriph.

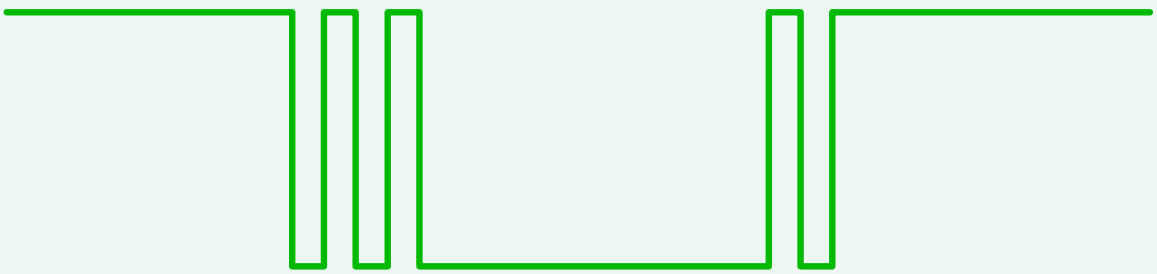
Исходя из принципиальной схемы устройства, ножка, к которой подсоединена кнопка — PA5. Добавим функцию инициализации в файл `button.c`

```
void button_init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    // PA3 - floating input
    GPIOA->CRL &= ~GPIO_CRL_MODE5;
    GPIOA->CRL |= GPIO_CRL_CNF5_1;
    GPIOA->CRL &= ~GPIO_CRL_CNF5_0;
    // ...
}
```

Настройка завершена. Наша цель — по нажатию кнопки менять состояние светодиода (выключать, если он был включен, и наоборот), который мы уже настраивали ранее. Теперь осталось написать код, который будет реагировать на нажатие. Конечно, мы можем проверять регистр `IDR` в главном цикле, но это не самое лучшее решение.

Замечание

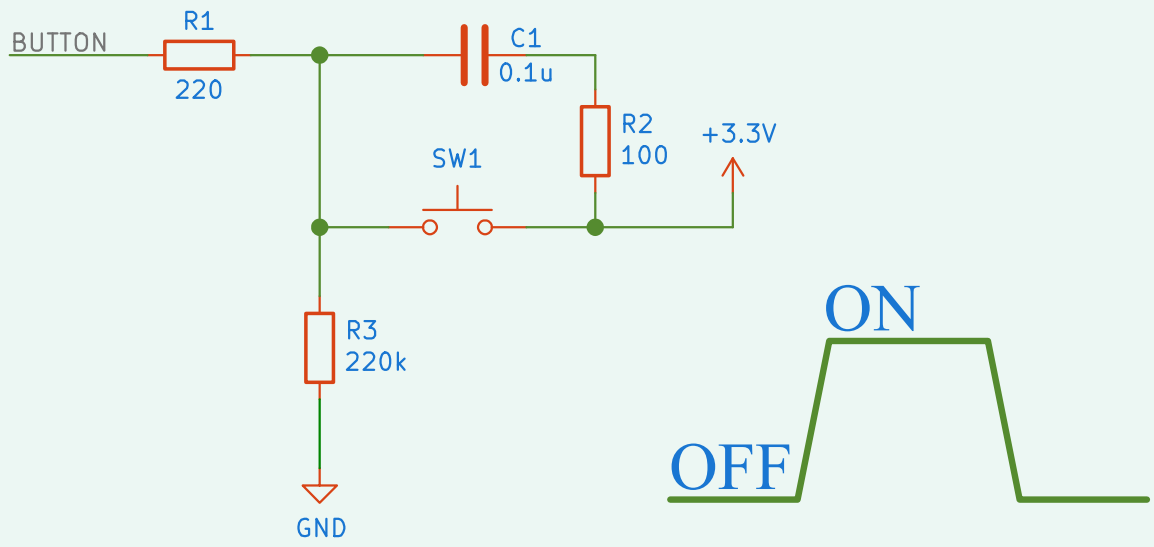
Посмотрите внимательно на принципиальную схему устройства, в частности, на то, как подключена кнопка. В случае подключения её без «обвеса» (RC-фильтра) будет происходить так называемый «дребезг» механического контакта.



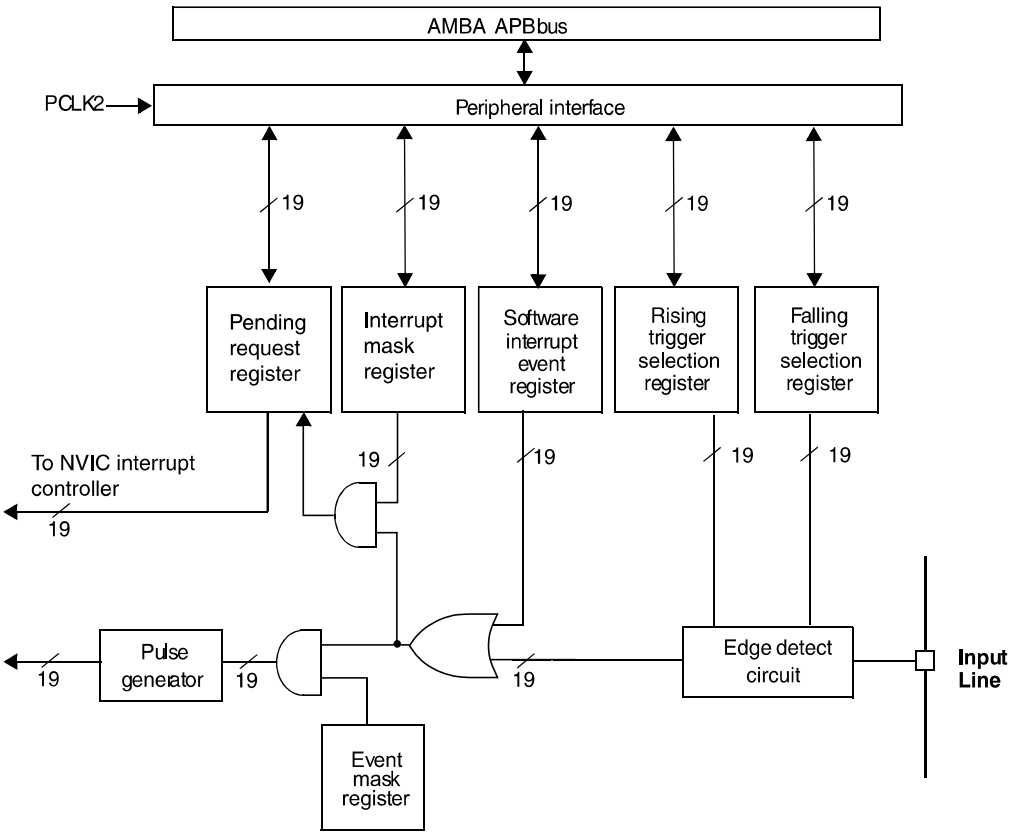
Это приведет к непредсказуемому результату, т. е. получится так, что кнопка нажимается несколько раз, и в каком состоянии окажется светодиод – неизвестно. Чтобы «отфильтровать» шум, можно поставить задержку на 30-40 мс (принято считать, что дребезг составляет именно столько времени). Далее мы еще раз проверим, нажата ли кнопка: если да, то, очевидно, это был не «шум», и нам действительно нужно переключить состояние светодиода. Для того чтобы избежать такого же дребезга, но уже на этапе отпускания кнопки, необходимо добавить цикл `while`, аргументом которого будет состояние регистра нужной ножки. Вот часть программы:

```
while(1) {
    if ((GPIOA->IDR & GPIO_IDR_IDR5) == GPIO_IDR_IDR5) {
        delay(40);
    if ((GPIOA->IDR & GPIO_IDR_IDR5) == GPIO_IDR_IDR5) {
        GPIOA->ODR ^= GPIO_ODR_ODR3;
        while(GPIOA->IDR & GPIO_IDR_IDR5);
    }
}
```

Так как RC-цепочка впаяна, то дребезг будет фильтроваться и никакой проблемы не возникнет.

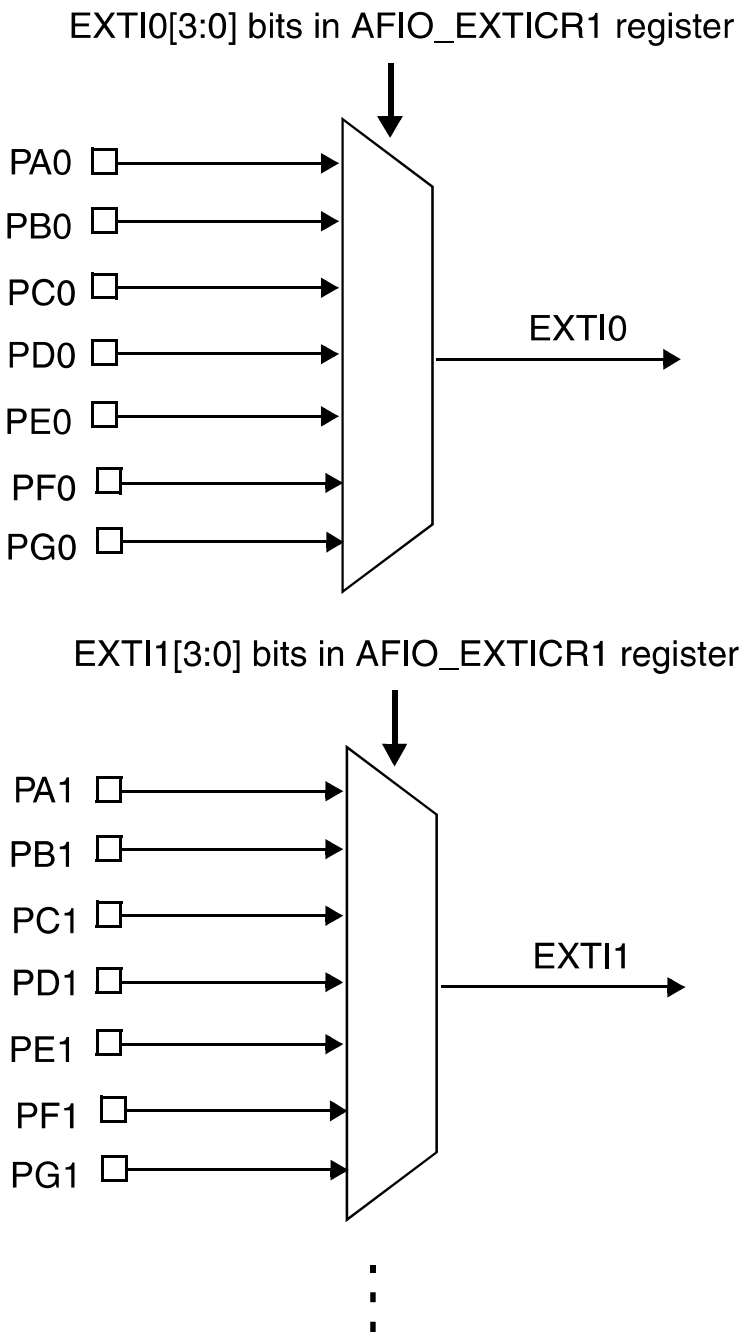


Гораздо лучше использовать для таких целей прерывания. Обратимся к документации (это делается для более глубокого понимания — эту информацию можно получить и из наименования макросов в библиотеке), к разделу «**8 Interrupts and events**» ⇒ «**External interrupt/event controller (EXTI)**». На странице 133 можно найти функциональную схему работы этого модуля.



Изображение из Reference Manual, *Figure 20. External interrupt/event controller block diagram*

Блок «**Edge detect circuit**» может реагировать как на передний фронт (импульса), так и на задний или на оба сразу. Это задается соответствующими битами в соответствующих регистрах. Всё в той же документации можно найти схему подключения ножек к модулю EXTI:



Изображение из Reference Manual, *Figure 21. External interrupt/event GPIO mapping*

Как видно, 3-я ножка порта A подключена к `EXTI5`. Чтобы глобально разрешить прерывания с `EXTI5`, необходимо произвести работу с регистром `ISER` (контроллера NVIC). К счастью, в файле `core_cm3.h` уже имеется готовая функция:

```
__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] =
  (1 << ((uint32_t)(IRQn) & 0x1F)); /* enable interrupt */
}
```

IRQn_Type объявлен в файле `stm32f10x.h`, следовательно:

```
NVIC_EnableIRQ(EXTI9_5_IRQn);
```

Перейдем к регистрам.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												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

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Interrupt Mask on line x
0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

Регистр `IMR` из Reference Manual

Регистр `IMR` отвечает за включение/отключение прерывания. Чтобы разрешить прерывание `EXTI5`, нужно записать «1» в `MR3`:

```
EXTI->IMR |= EXTI_IMR_MR5;
```

Последующие два регистра отвечают за прерывание по переднему фронту и заднему соответственно. Мы будем реагировать на передний, поэтому следует настроить `RTSR`:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												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

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Rising trigger event configuration bit of line x
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line.
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

Note: The external wakeup lines are edge triggered, no glitches must be generated on these lines. If a rising edge on external interrupt line occurs during writing of EXTI_RTSR register, the pending bit will not be set.

Rising and Falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

Регистр `RTSR` из Reference Manual

Так как мы работаем с 5-й ножкой, нас интересует бит `TR5`:

```
EXTI->RTSR |= EXTI_RTSR_TR5;
```

Допишем код.

```
void button_init() {
    // ...
    NVIC_EnableIRQ(EXTI9_5_IRQn);
    EXTI->IMR  |= EXTI_IMR_MR5;
    EXTI->RTSR |= EXTI_RTSR_TR5;
    EXTI->FTSR &= ~EXTI_FTSR_TR5;
}
```

Настройка прерывания закончена, осталось написать обработчик, название которого мы также возьмем в `startup_<mcu>.s` файле.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
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 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **PRx:** Pending bit
0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a ‘1’ into the bit.

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

Регистр `PR` из Reference Manual

Последний значимый для нас регистр именуется `PR` (англ. pending register). Дело в том, что прерывание может прийти от разных портов (что было видно на схеме выше), поэтому вам самостоятельно нужно проверять, с какого именно порта оно пришло, и сбрасывать «флаг», который записывается в соответствующий бит регистра `PR`.

Таким образом, код для обработчика примет следующий вид:

```
void EXTI9_5_IRQHandler(void) {
    static volatile uint8_t flag = 0;
    if (flag) {
        GPIOA->ODR |= GPIO_ODR_ODR3; // led on
        flag = 0;
    } else {
        GPIOA->ODR &= ~GPIO_ODR_ODR3; // led off
        flag = 1;
    }

    EXTI->PR |= EXTI_PR_PR5;
}
```

В стандартную библиотеку входят два файла для работы с модулем `EXTI`. Опять же, для повышения уровня абстракции мы предлагаем вам переписать работу с `EXTI` на функции стандартной библиотеки. В дальнейшем мы не будем рассматривать периферию так же подробно (на уровне регистров). Нашей целью было показать принцип — мы этой цели достигли.

Код урока можно найти на github: **CMSIS**.

Назад | Оглавление | Дальше