

## Глава 6. Прерывания. NVIC, EXTI

Ежелев Г.И.

29 Ноября.

# Основные принципы

## Применение

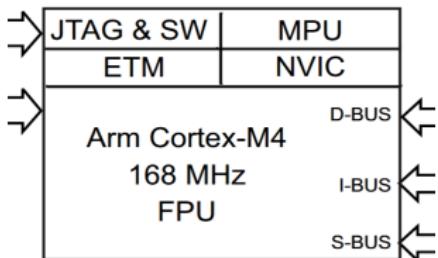
- ▶ Задачи, где важно **быстро** среагировать на какое-то событие (не обязательно внешнее) и его обработать. Т.е. **прервать** программу; **обработать** событие; **вернуться обратно** в ту же часть программы и **продолжить** выполнение.
- ▶ Например, **инкремент** энкодера. Сигнал от датчика настолько короткий, что его можно **упустить** при стандартной обработке:  
`data = GPIO_PIN_x & GPIOx -> IDR` (сигнал пройдет пока программа на других строках кода).
- ▶ Внутреннее событие так же может вызывать прерывание. На самом деле такое прерывание мы уже писали - это **подсчёт времени**.

# NVIC

**Nested vectored interrupt controller** (NVIC) - Вложенный векторный контроллер прерываний.

Что это значит?

- ▶ **Вложенный** (nested) - контроллер находится прямо внутри ядра процессора CORTEX-M4. Это обеспечивает наибольшую скорость обработки прерываний.



- ▶ **Векторный** (vectored) - программа содержит таблицу с адресами функций - обработчиков конкретного события. Каждый такой адрес называется **вектором**.

# Немного о самой таблице векторов (адресов прерываний)

- ▶ В начале памяти (по умолчанию во флеше) лежит массив слов: первое слово — начальный адрес стека, дальше идут адреса обработчиков **Reset**, **NMI**, **HardFault**, затем всех остальных прерываний.
- ▶ В самом ядре есть регистр **VTOR**: в нём хранится начальный адрес таблицы.
- ▶ **Номер прерывания = индекс в таблице.** Если придёт, например, IRQ №23, ядро возьмёт слово №23 от начала таблицы и прыгнет по адресу, который там лежит.
- ▶ То есть конкретное прерывание **жестко привязано** именно к номеру строки в таблице.

Эту таблицу можно найти в файле `startup_stm32` (в проекте в папке `main`)

```
; Vector Table Mapped to Address 0 at Reset
AREA    RESET, DATA, READONLY
EXPORT   __Vectors
EXPORT   __Vectors_End
EXPORT   __Vectors_Size

__Vectors DCD    __initial_sp           ; Top of Stack
          DCD    Reset_Handler        ; Reset Handler
          DCD    NMI_Handler         ; NMI Handler
          DCD    HardFault_Handler  ; Hard Fault Handler
          DCD    MemManage_Handler   ; MPU Fault Handler
          DCD    BusFault_Handler   ; Bus Fault Handler
          DCD    UsageFault_Handler ; Usage Fault Handler
          DCD    0                   ; Reserved
          DCD    0                   ; Reserved
          DCD    0                   ; Reserved
          DCD    0                   ; Reserved
          DCD    SVC_Handler        ; SVCall Handler
          DCD    DebugMon_Handler   ; Debug Monitor Handler
          DCD    0                   ; Reserved
          DCD    PendSV_Handler    ; PendSV Handler
          DCD    SysTick_Handler    ; SysTick Handler

; External Interrupts
          DCD    WWDG_IRQHandler ; Window WatchDog
```

# ISR

То, что мы называем прерыванием - функция обрабатывающая конкретное событие, вызванная по адресу из таблицы

- ▶ Строго говоря, перед тем, как **прыгнуть** в эту функцию **хорошо бы запомнить** откуда мы это **делаем**, чтобы после конца прерывания не забыть, куда надо **вернуться**. Также нужно **сохранить** все локальные переменные и необработанные значения
- ▶ **ISR** (Interrupt Service Routine) — это **процедура обслуживания прерывания**, то есть функция, которую процессор выполняет в ответ на конкретное прерывание.
- ▶ Состоит как раз из «запоминания», самой функции, возврата в исходную позицию.
- ▶ За нас, как и в ардуине, все делает либа!!!

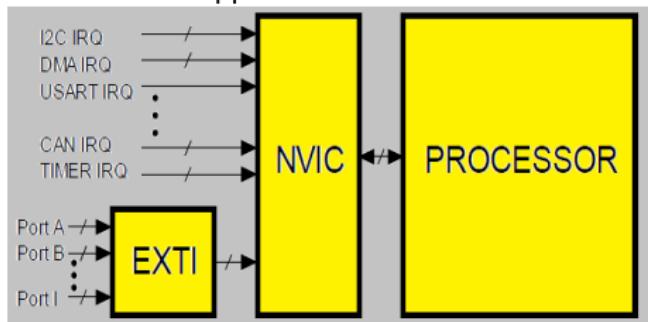
## Обычно ISR:

- ▶ **Быстро фиксирует событие:** читает/сбрасывает флаг прерывания, забирает данные из регистра периферии, кладёт в буфер, ставит флаги.
- ▶ **Минимизирует время выполнения:** без тяжёлых циклов, логики чтобы не блокировать остальные прерывания и основной код.
- ▶ На время исполнения может блокировать (откладывать) другие прерывания. **Если вы заблокируете другие прерывания и вызовите `delay(ms)` то программа зависнет.** Угадайте почему.

EXTI

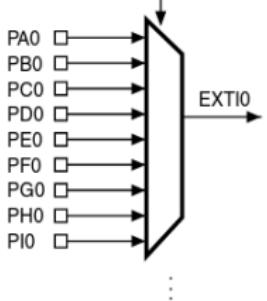
**EXTI** - external interrupt event controller. Контроллер внешних прерываний.

- ▶ Работа с GPIO самая сложная - т.к. это самая быстрая периферия.
  - ▶ Для неё выделен отдельный контроллер - EXTI. Он стоит на входе NVIC.



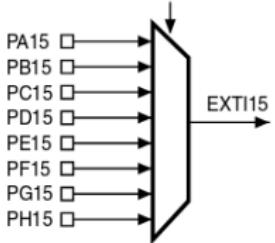
- Имеет 20 линий. EXTI0...EXTI15 ->PX0...PX15

EXTI0[3:0] bits in the SYSCFG\_EXTICR1 register



⋮

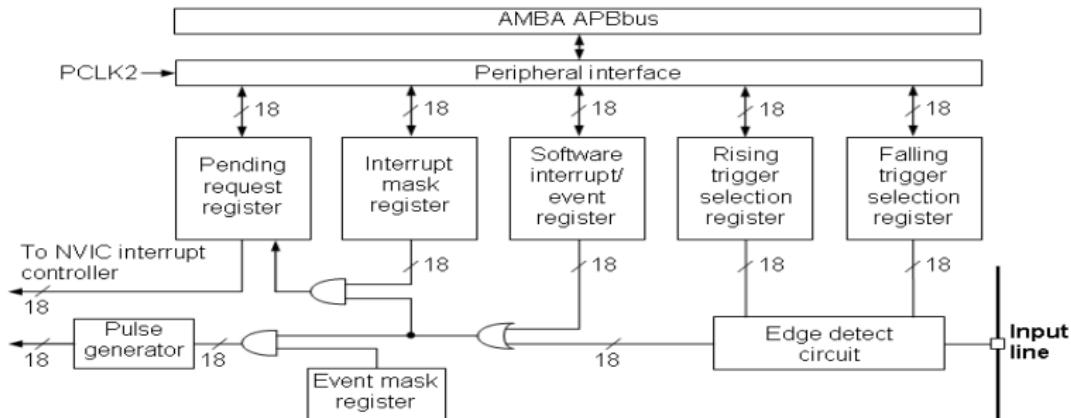
EXTI15[3:0] bits in the SYSCFG\_EXTICR4 register



EAX.ME

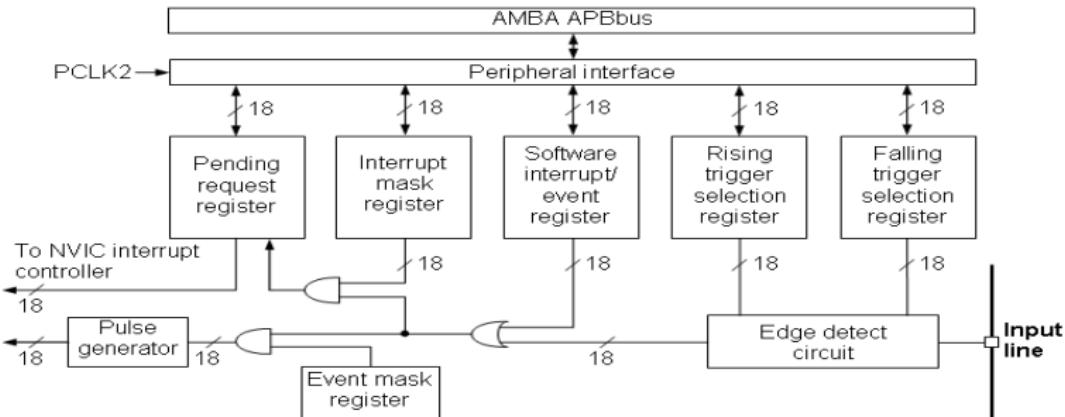
- EXTI16-20 RTC, датчик напряжения, USB, Ethernet

# Внутреннее устройство



- ▶ **Edge detector** - детектор "фронтов" уже встречался в главе 0. Управляется двумя регистрами выше.
- ▶ Выход с детектора проходит через логическое или с **регистром программного прерывания** - нужен для программного включения прерывания.
- ▶ Код на [github](#)

# Внутреннее устройство



- ▶ если interrupt mask register - если в нем лежит правильная маска пропустит через логическое И сигнал и установит **pending bit** (бит ожидания).
- ▶ Нижний путь через pulse generator - создает **событие** например wakeup или external trigger для ADC. Может использоваться без прямого задействования механизма прерываний для активации других блоков контроллера.

# Настройка

Сначала TIME -> GPIO -> EXTI -> NVIC

- ▶ настройка **GPIO** - стандартная, режим, очевидно, входа.
- ▶ Настройка **EXTI** идет перед настройкой **NVIC** (просто так принято)