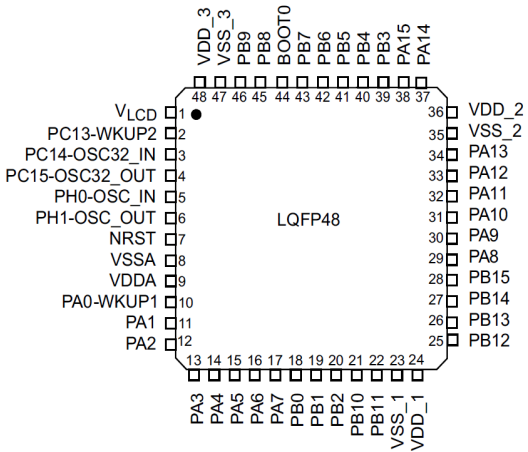
Аппаратное обеспечение IoT/CPS
Лекция 2

А. А. Подшивалов
apodshivalov@miem.hse.ru

GPIO

GPIO — самый «микроконтроллерный» интерфейс

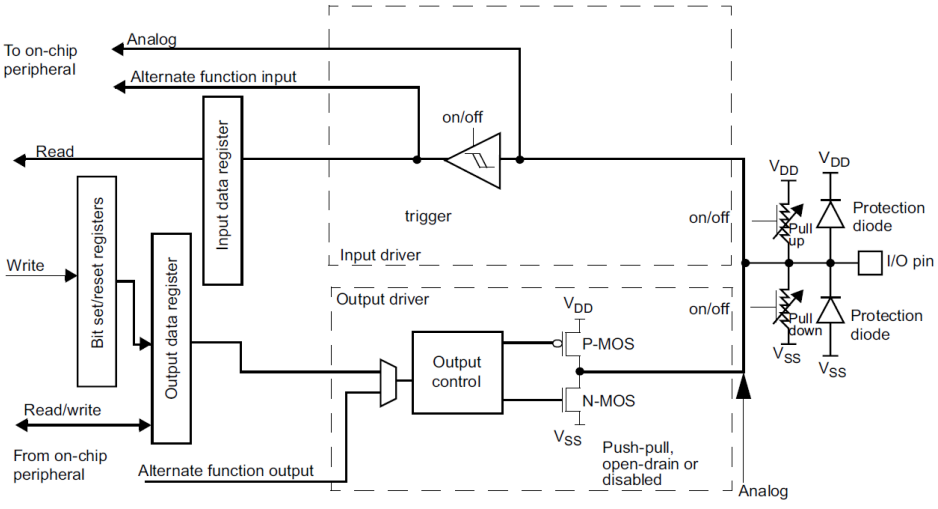
- General purpose input/output, ввод-вывод общего назначения
- Из 48 выводов STM32L151CC — 37 могут работать, как GPIO
- 8/**16**/32 вывода объединяются в порт
- Обозначение PA5 — 5 вывод порта A



Основные и дополнительные функции

- Режимы работы GPIO (в STM32L1 — регистр MODER; далее рассматриваем эту серию)
  - Цифровой вход
  - Цифровой выход
  - Аналоговый вход/выход (реализован не у всех выводов)
  - Альтернативные функции
- Альтернативные функции подключаются при помощи регистров AFRL и AFRH
- Иногда выводы совмещаются со специальными функциями микроконтроллера (выбор источника загрузки, сброс, тактирование)

Структура вывода GPIO



Альтернативные функции

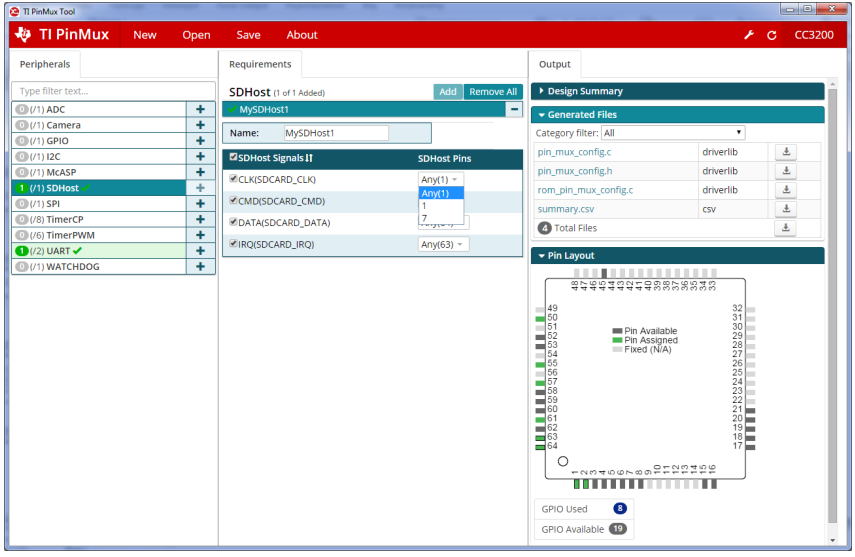
Port name	Digital alternate function number														
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFIO6	AFIO7	AFIO8	AFIO9	AFIO11	AFIO12	AFIO13	AFIO14	AFIO15
	Alternate function														
	SYSTEM	TIM2	TIM3/4	TIM9/10/11	I2C1/2	SPI1/2	N/A	USART1/2/3	N/A	N/A	LCD	N/A	N/A	RI	SYSTEM
BOOT0	BOOT0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
NRST	NRST	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PA0-WKUP1	-	TIM2_CH1_ETR	-	-	-	-	-	USART2_CTS	-	-	-	-	-	TIMx_IC1	EVENTOUT
PA1	-	TIM2_CH2	-	-	-	-	-	USART2_RTS	-	-	[SEG0]	-	-	TIMx_IC2	EVENTOUT
PA2	-	TIM2_CH3	-	TIM9_CH1	-	-	-	USART2_TX	-	-	[SEG1]	-	-	TIMx_IC3	EVENTOUT
PA3	-	TIM2_CH4	-	TIM9_CH2	-	-	-	USART2_RX	-	-	[SEG2]	-	-	TIMx_IC4	EVENTOUT
PA4	-	-	-	-	-	SPI1_NSS	-	USART2_CK	-	-	-	-	-	TIMx_IC1	EVENTOUT
PA5	-	TIM2_CH1_ETR	-	-	-	SPI1_SCK	-	-	-	-	-	-	-	TIMx_IC2	EVENTOUT
PA6	-	-	TIM3_CH1	TIM10_CH1	-	SPI1_MISO	-	-	-	-	[SEG3]	-	-	TIMx_IC3	EVENTOUT
PA7	-	-	TIM3_CH2	TIM11_CH1	-	SPI1_MOSI	-	-	-	-	[SEG4]	-	-	TIMx_IC4	EVENTOUT
PA8	MCO	-	-	-	-	-	-	USART1_CK	-	-	[COM0]	-	-	TIMx_IC1	EVENTOUT
PA9	-	-	-	-	-	-	-	USART1_TX	-	-	[COM1]	-	-	TIMx_IC2	EVENTOUT
PA10	-	-	-	-	-	-	-	USART1_RX	-	-	[COM2]	-	-	TIMx_IC3	EVENTOUT
PA11	-	-	-	-	-	SPI1_MISO	-	USART1_CTS	-	-	-	-	-	TIMx_IC4	EVENTOUT
PA12	-	-	-	-	-	SPI1_MOSI	-	USART1_RTS	-	-	-	-	-	TIMx_IC1	EVENTOUT
PA13	JTMS-SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	TIMx_IC2	EVENTOUT
PA14	JTCK-SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	TIMx_IC3	EVENTOUT
PA15	JTDI	TIM2_CH1_ETR	-	-	-	SPI1_NSS	-	-	-	-	SEG17	-	-	TIMx_IC4	EVENTOUT
PB0	-	-	TIM3_CH3	-	-	-	-	-	-	-	[SEG5]	-	-	-	EVENTOUT

Альтернативные функции

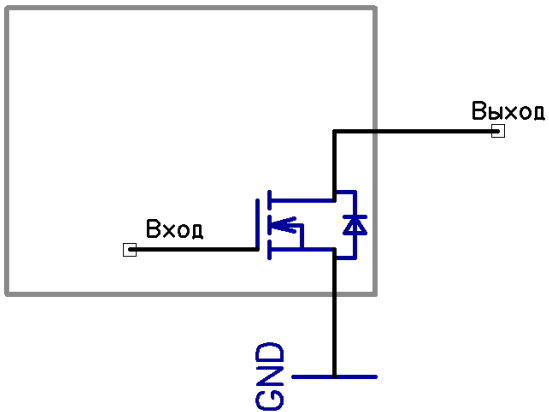
Как мы можем подключать различные встроенные периферийные устройства к выводам микроконтроллера?

- Полная матрица переключений
  - Любой периферийный блок может быть подключен к любому выводу микроконтроллера (за исключением «аналоговых» устройств)
  - TI CC13xx/CC26xx, Nordic Semiconductor nRF51/nRF52
- Ограниченная матрица переключений
  - Каждый периферийный блок может подключаться к 2-4 выводам GPIO
  - STM32 и большинство других микроконтроллеров
- Фиксированная матрица переключений
  - Каждый периферийный блок привязан к конкретному GPIO
  - PIC, AVR, большинство 8-битных микроконтроллеров

Альтернативные функции

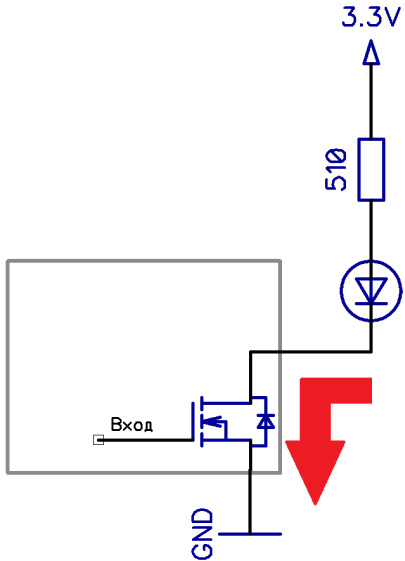


Выход с открытым стоком

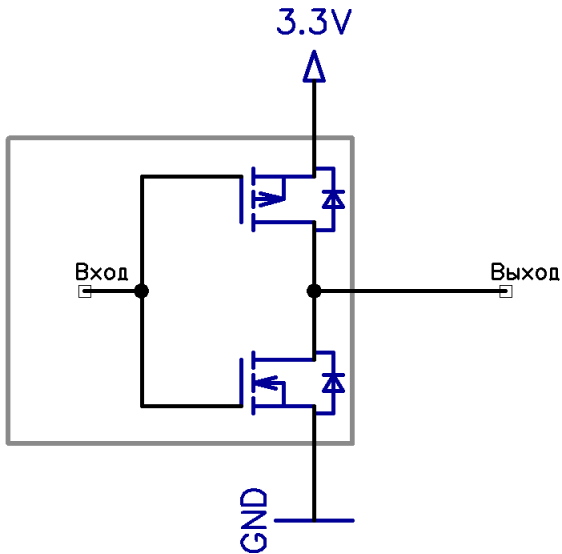


- Он же «открытый коллектор», он же «open drain»

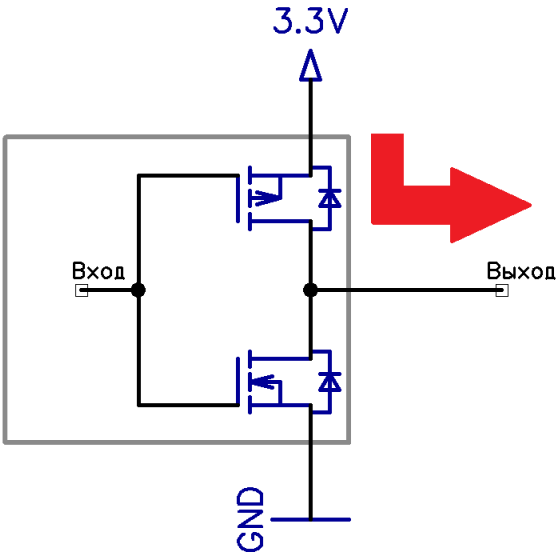
Выход с открытым стоком



Push-pull

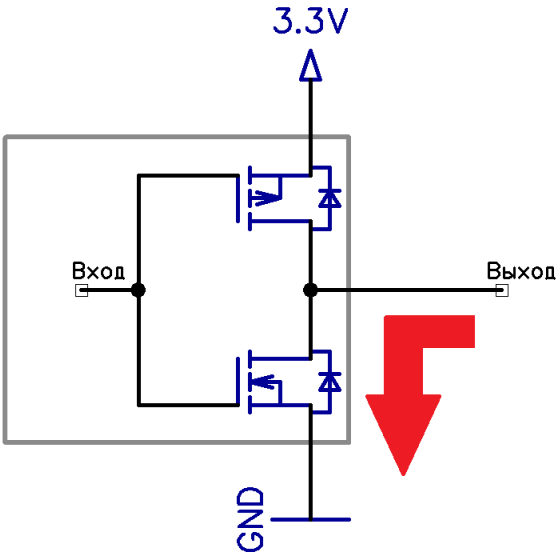


Push-pull



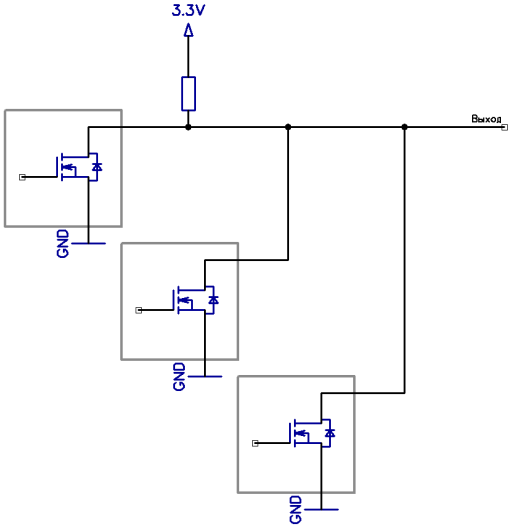
- На входе низкий уровень, верхний транзистор открыт, нижний закрыт
- Вытекающий ток (source current)

Push-pull



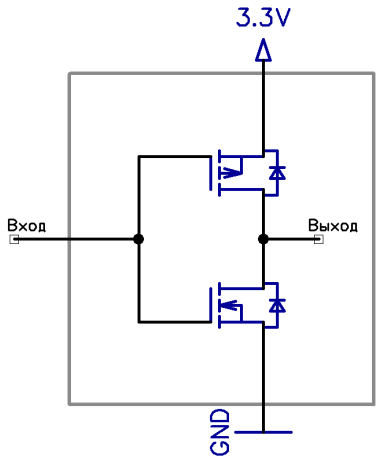
- На входе высокий уровень, верхний транзистор закрыт, нижний открыт
- Втекающий ток (sink current)
- Иногда втекающий ток существенно больше вытекающего — смотрите даташит!

Открытый сток еще раз

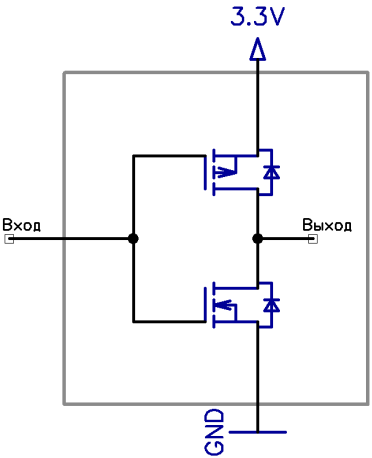


- Такое подключение применяется в шинах I^2C и 1-Wire
- Одновременное включение «0» любым количеством устройств не приводит к выходу из строя других устройств на шине

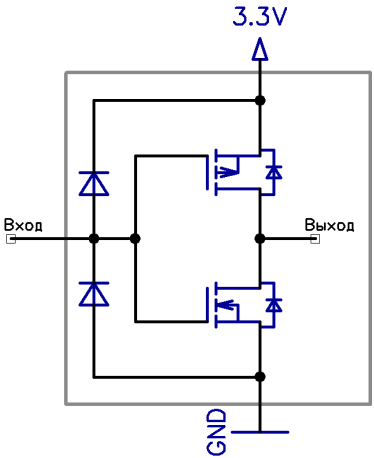
Цифровой вход



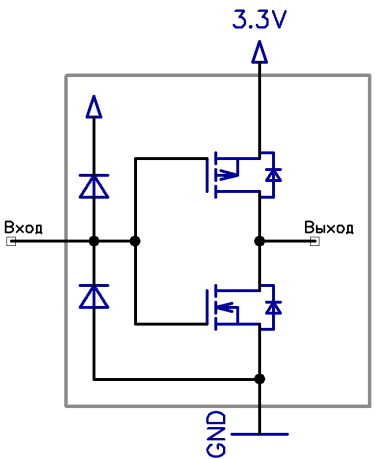
Цифровой вход с защитой



Без защиты

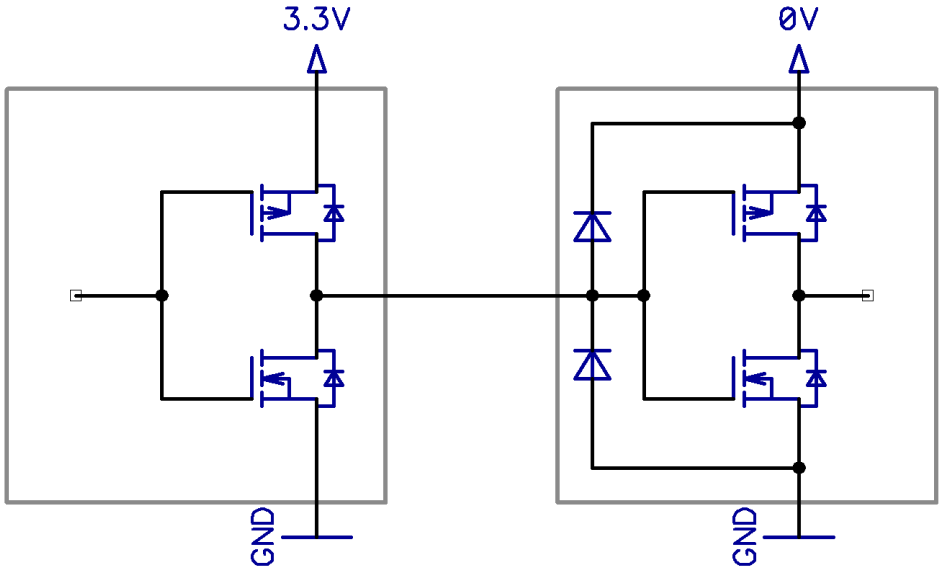


Защита от статики

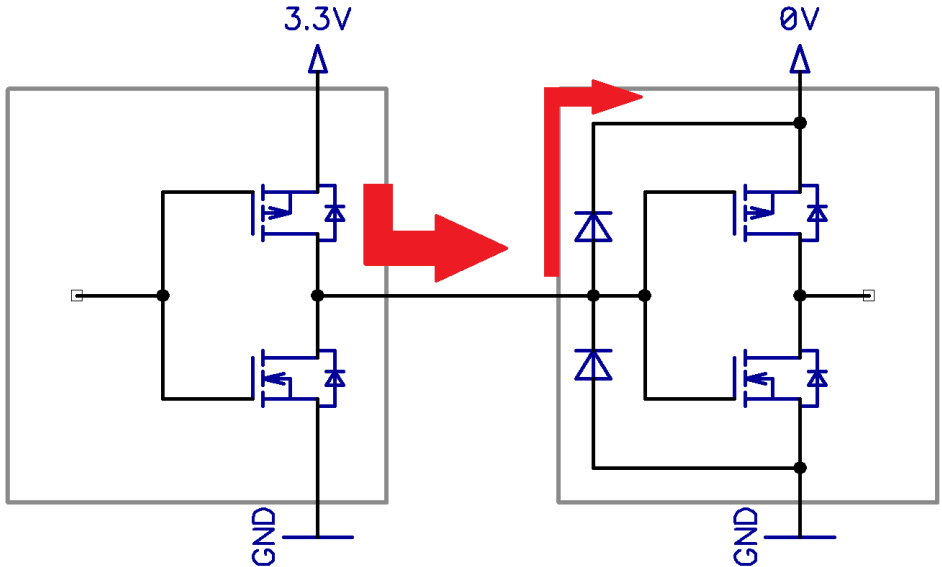


5V-tolerant

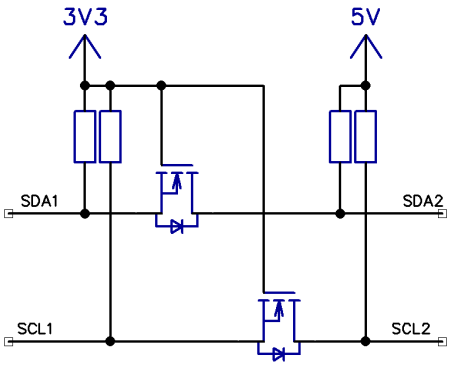
???


3.3V
0V
GND
GND

Паразитное питание

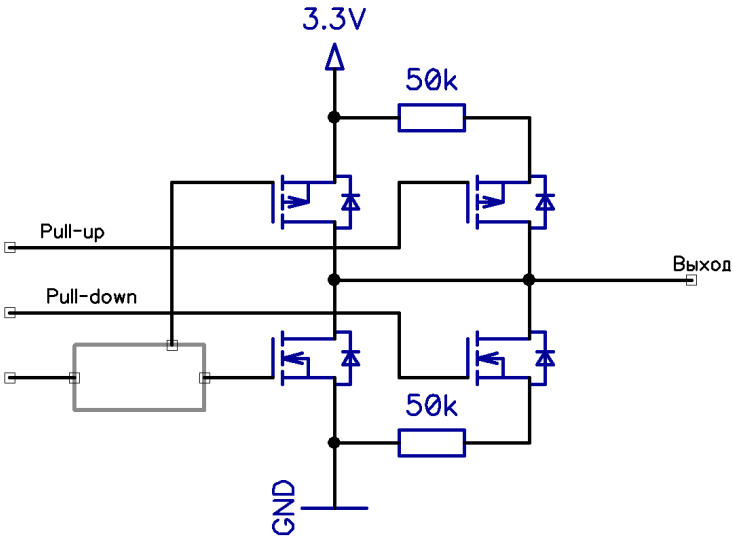


Согласование уровней



- Philips Semiconductor AN97055
- Согласование уровней и изоляция шин I^2C с разным напряжением
- Может быть использовано и как однонаправленный преобразователь уровней в случае push-pull, тогда можно обойтись без подтягивающих резисторов на стороне выхода
- А вообще используйте готовые преобразователи уровней (например, TI TXB0108), особенно в случае двунаправленных шин

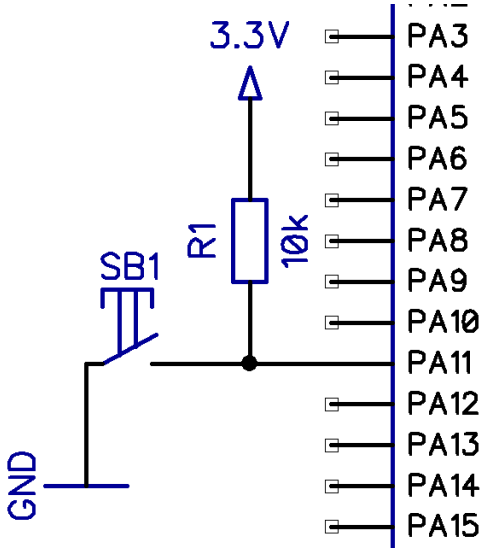
Встроенная подтяжка



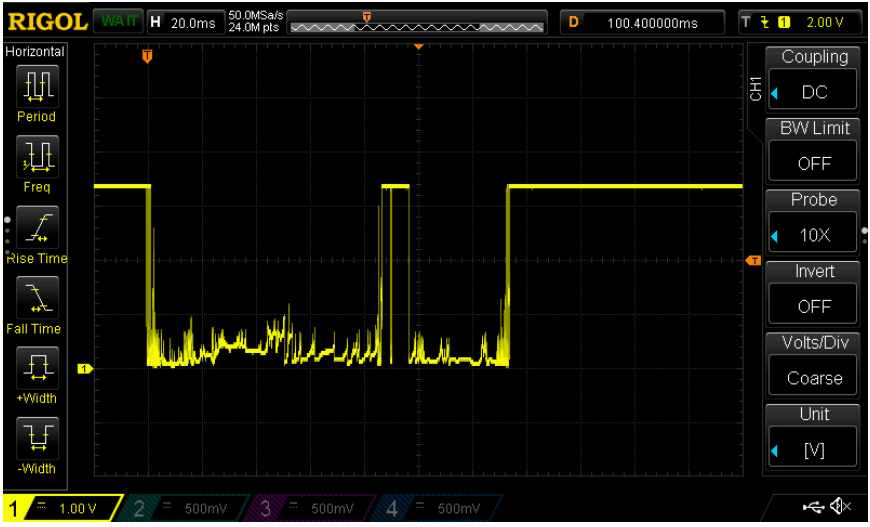
API GPIO в Riot

- Инициализация
  `gpio_init(GPIO_PIN(PORT_A, 5), GPIO_OUT);`
- Варианты:
  `GPIO_IN, GPIO_IN_PD, GPIO_IN_PU, GPIO_OUT, GPIO_OD, GPIO_OD_PU`
- Чтение
  `gpio_read(GPIO_PIN(PORT_A, 5));`
- Запись
  `gpio_set, gpio_clear, gpio_toggle, gpio_write`

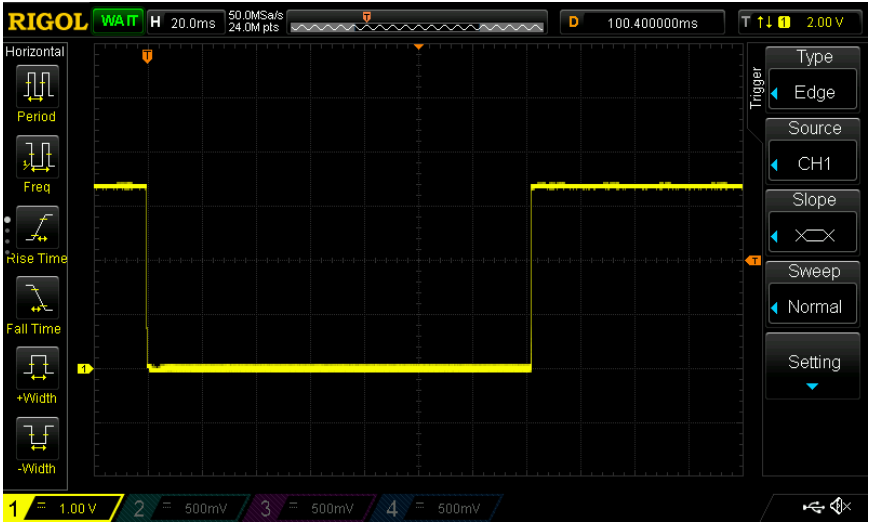
Давайте подключим кнопку



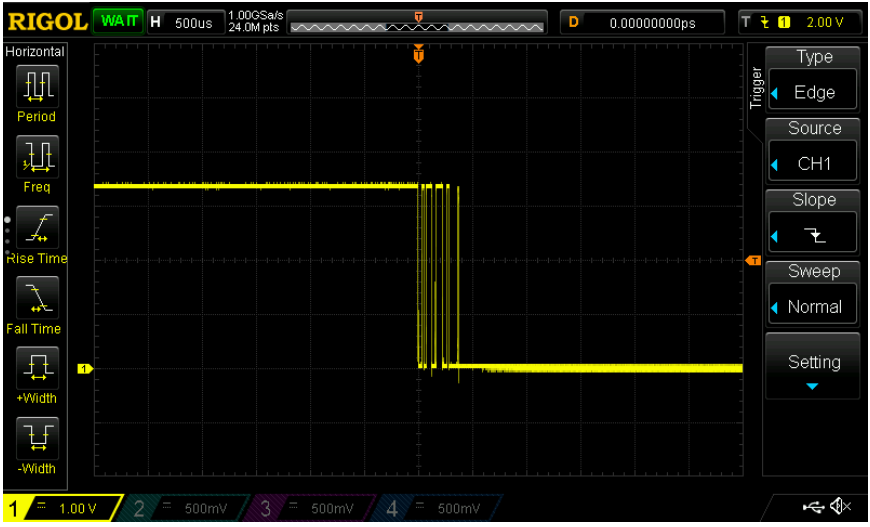
Давайте подключим кнопку



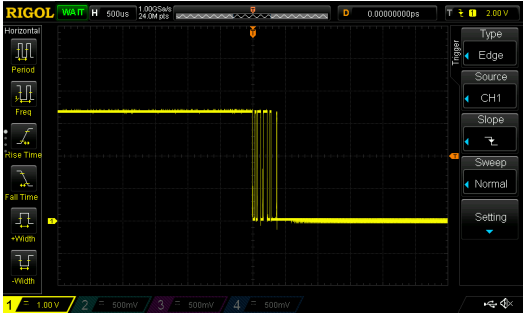
Давайте подключим кнопку



Давайте подключим кнопку



Дребезг контактов



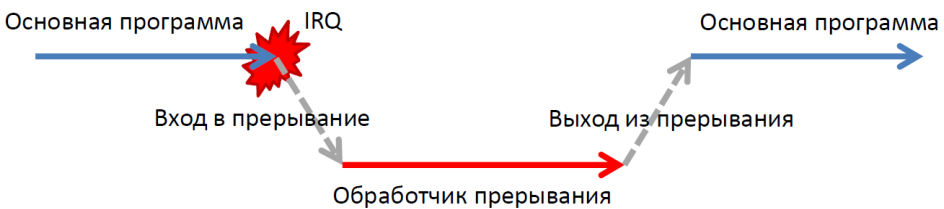
Два метода борьбы:

- Периодический опрос
- Отключение прерываний

Прерывания

События и прерывания

- Прерывание — событие, на которое процессор обязан немедленно отреагировать
  - Программные — вызываются выполнением специальной инструкции
  - Синхронные — вызывается процессором при сбое в выполнении инструкций
  - Асинхронные — от внешних источников
- Прерывание может быть отключено (замаскировано), за исключением NMI — Non-maskable Interrupt



Обработка прерываний на примере Cortex-M

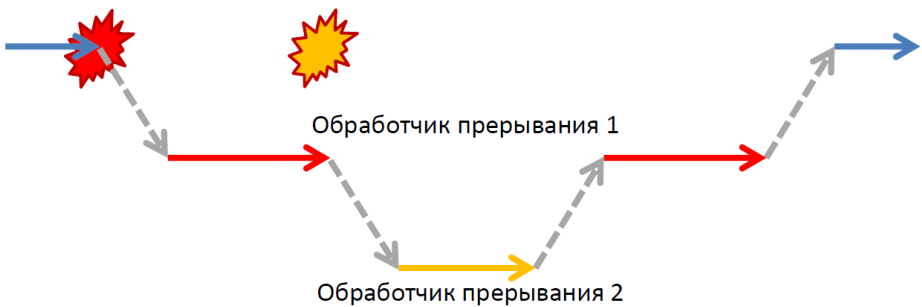
- NVIC — Nested Vectored Interrupt Controller
- Поддерживается таблица прерываний — набор указателей на функции-обработчики

```
/**
 * @brief   All ISR functions have this type
 */
typedef void (*isr_t)(void);

/**
 * @brief   Structure of Cortex-M basic vector table
 */
typedef struct {
  void* _estack;                /**< exception stack pointer */
  isr_t vectors[CPU_EXCEPTIONS];/**< shared Cortex-M vectors */
} cortexm_base_t;
```

Приоритеты и вложенность прерываний

- Что будет, если в ходе обработки прерывания произойдет еще одно событие?



- Прерывание 2 имеет больший приоритет, поэтому обработчик прерывания 1 приостанавливает свою работу

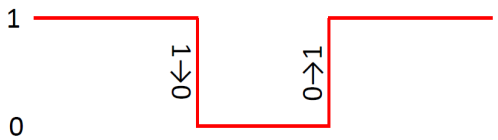
Приоритеты и вложенность прерываний

- Что будет, если в ходе обработки прерывания произойдет еще одно событие?

- Прерывание 2 имеет меньший либо равный приоритет, поэтому обработчик прерывания 1 продолжает выполняться

Прерывания GPIO

- Срабатывает по фронту сигнала
  - Передний фронт ($0 \rightarrow 1$)
  - Задний фронт ($1 \rightarrow 0$)
  - Оба фронта



Поддержка прерываний в Riot на примере Cortex-M

- Все прерывания имеют одинаковый приоритет
- Таблица прерываний находится в ROM, обработчики прерываний описаны в драйверах периферии, например:
  - В файле `cpu/stm32/vectors/STM32L151xC.c` (генерируется автоматически) объявлена и помещена в таблицу прерываний функция

    ```
    WEAK_DEFAULT void isr_exti(void);
    ```
  - Реализация этой функции находится в файле `cpu/stm32/periph/gpio_ll.c`
- В конце каждого прерывания вызывается функция

  ```
  cortexm_isr_end();
  ```

Поддержка прерываний в Riot на примере Cortex-M

```
void isr_exti(void) {
    /* read all pending interrupts wired to isr_exti */
    uint32_t pending_isr = (EXTI_REG_PR & EXTI_MASK);
    /* clear by writing a 1 */
    EXTI_REG_PR = pending_isr;
    /* only generate interrupts against lines which have IMR set */
    pending_isr &= EXTI_REG_IMR;
    /* iterate over all set bits */
    uint8_t pin = 0;
    while (pending_isr) {
        pending_isr = bitarithm_test_and_clear(pending_isr, &pin);
        isr_ctx[pin].cb(isr_ctx[pin].arg);
    }
    cortexm_isr_end();
}
```

API прерываний GPIO в Riot

- Определяем функцию для обработки прерывания, например:

```
void btn_handler(void *arg) {
    do_something_with( (int) arg );
}
```

- Регистрируем ее как обработчик soft interrupt, можно передать приведенный к типу `void*` аргумент:

```
gpio_init_int(GPIO_PIN(PORT_A, 11),
              GPIO_IN_PU,
              GPIO_FALLING,
              btn_handler,
              (void*) 123);
```

- Чтобы включить или выключить конкретное прерывание, используем функции

`gpio_irq_enable, gpio_irq_disable`

Таймеры

Аппаратные таймеры

- Таймер в микроконтроллере — счетчик с возможностью генерации событий
- Разрядность — 16, 24, 32 бита
- Настройки:
  - Источник тактирования (внутренний генератор или один из входов)
  - Делитель
  - Регистр сравнения («будильник»)
  - Направление счета, синхронизация между таймерами, ...
- Прерывания по переполнению или совпадению значения с регистром сравнения
- Иногда могут управлять одним из выводов GPIO (удобно для аппаратной генерации ШИМ) или периферийным устройством (обычно АЦП или ЦАП) без участия процессора

Программные таймеры в Riot

- Слой абстракции поверх аппаратных таймеров
- Позволяет завести любое количество программных «будильников», сложность добавления нового таймера — $O(n)$
- xtimer — частота 1 МГц (от одного из таймеров общего назначения), позволяет отмерять интервалы времени с точностью до нескольких микросекунд
- `lptimer` — частота около 1 кГц, работает от real-time counter, в том числе в «спящем» режиме
- `ztimer` — совмещает в одном программном модуле функциональность xtimer и lptimer