Программно-аппаратные платформы Интернета вещей и встраиваемые системы

Лекция 2

КОММЕНТАРИИ К ОПРОСУ

volatile

Что означает ключевое слово volatile в языке Си?

- ничего, компилятор может проигнорировать это ключевое слово
- переменная, объявленная с модификатором volatile, может быть изменена не предусмотренным в программе способом
- эту переменную можно безопасно использовать из разных потоков/процессов в многозадачной системе
- запрещает некоторые оптимизации компилятора при обращении к этой переменной

static

- Модификатор `static` для функций и внешних переменных (определенных снаружи функции) ограничивает область видимости текущим файлом

```
#define PRIVATE static
```

- Модификатор `static` для внутренних переменных функции определяет время их существования – оно не прекращается при выходе из функции (т. е. память выделяется не на стеке, а в сегменте `.bss`)

auto

```
auto a = 1.0 / 2.0;
if (a > 0){
        puts("Compiled as C++ code!");
} else {
        puts("Compiled as C code!");
}
```

Арифметика

- 16-битный `int` (и `INT_MAX`, равный 32767) – не редкость
- `char` может быть как знаковым, так и беззнаковым; используйте этот тип только для работы с «буковками», для вычислений есть `int8_t` и `uint8_t`
- По возможности пользуйтесь типами из `stdint.h`

Определения

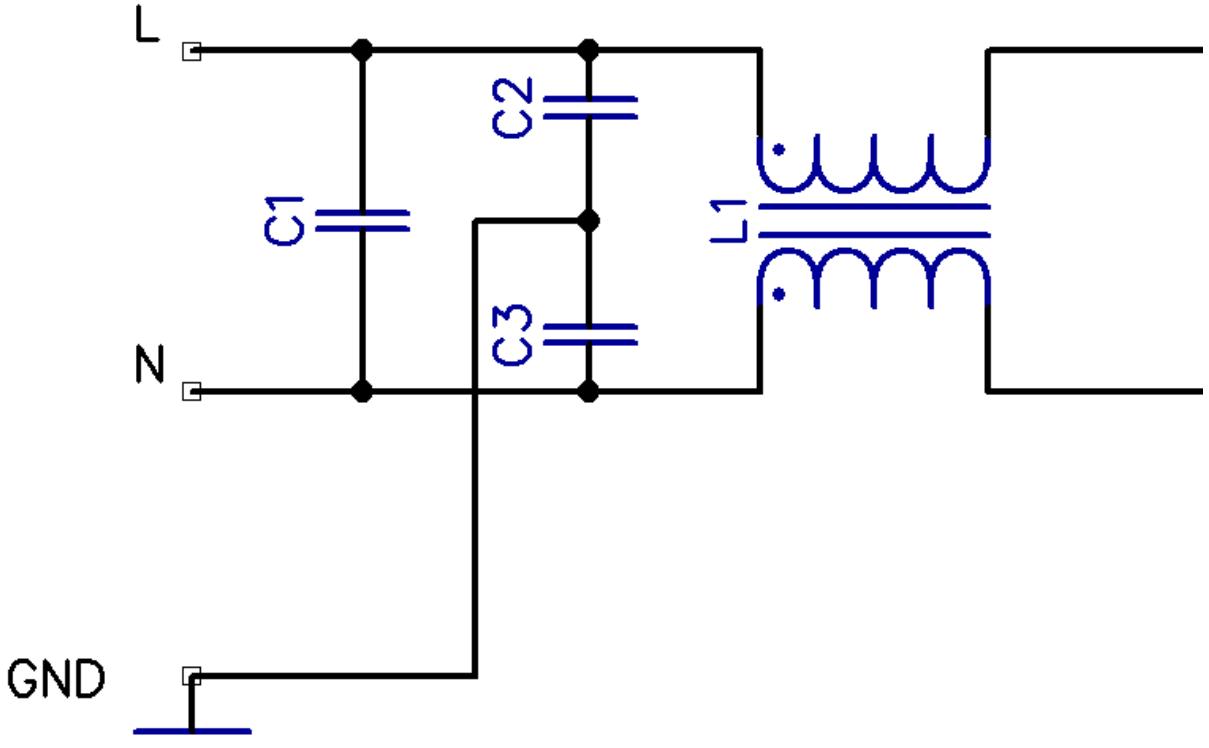
```
int a; // Целое число
int *a; // Указатель на целое число
int a[10]; // Массив из 10 целых чисел
int *a[10]; // Массив из 10 указателей на целое число
int (*a)[10]; // Указатель на массив из 10 целых чисел
int (*a)(int); // Указатель на функцию
int (*a[10])(int); // Массив из 10 указателей на функцию
```

malloc

```
char *ptr;
if ((ptr = (char*) malloc(0)) == NULL) {
      puts("Got a null pointer");
} else {
      puts("Got a valid pointer");
}
```

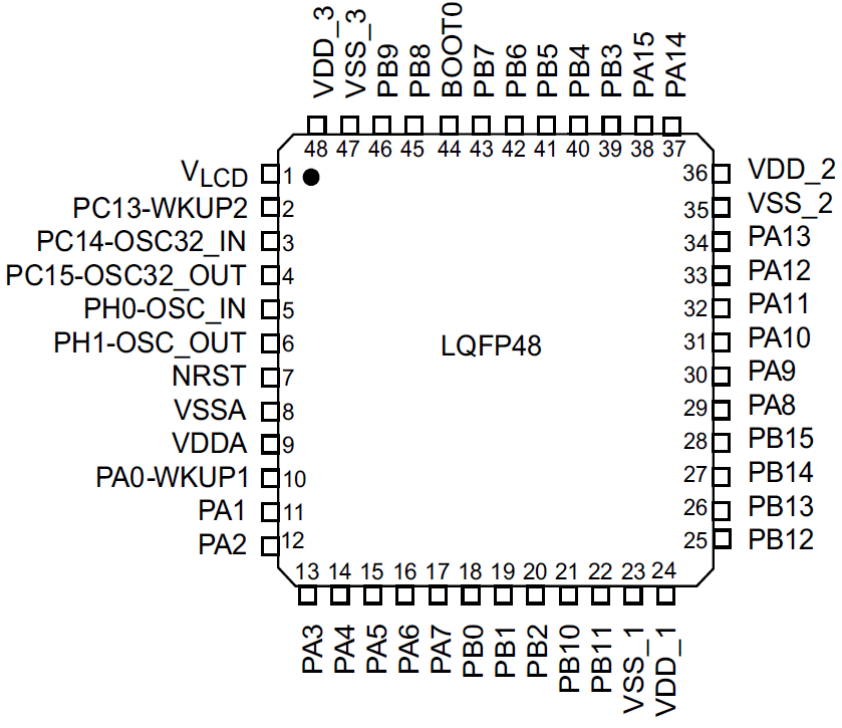
Правильный ответ – ~~«не знаю»~~ implementation defined

Электробезопасность



GPIO

General purpose input/output

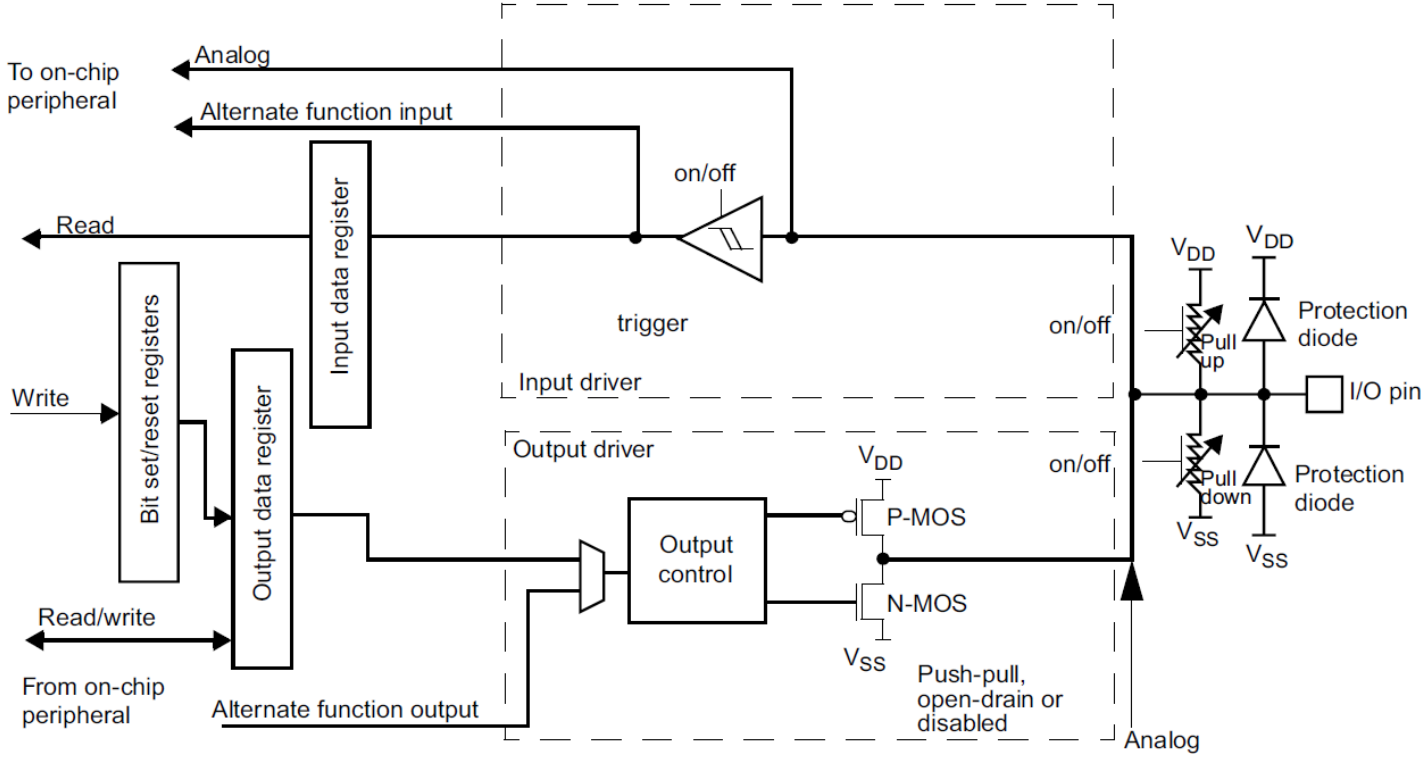


- Основной внешний интерфейс микроконтроллера, «ввод-вывод общего назначения»
- Далее будем рассматривать на примере STM32L151CC
- Из 48 выводов – 37 могут работать как GPIO

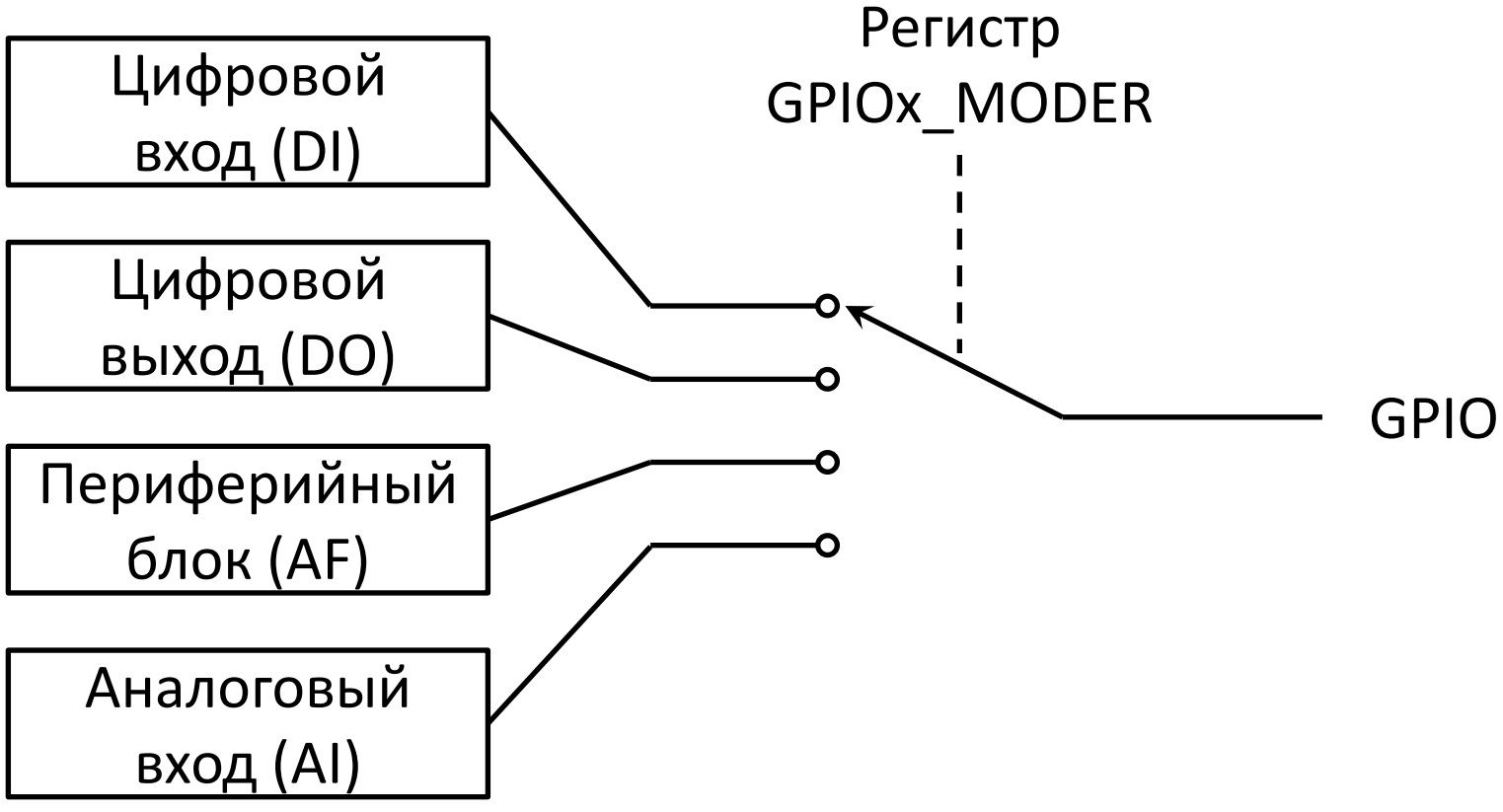
GPIO и порты

- Основные функции
  - Цифровой вход
  - Цифровой выход
- Дополнительные функции
  - Аналоговые входы (блоки АЦП/компаратора)
  - Цифровые входы и выходы внутренних периферийных блоков
  - Специальные функции микроконтроллера (выбор источника загрузки, сброс, тактирование)
- Порты
  - 8/**16**/32 вывода объединяются в **порт**
  - GPIO одного порта соответствуют биты одного и того же набора регистров
  - Обозначение: PA5 – 5 вывод порта A

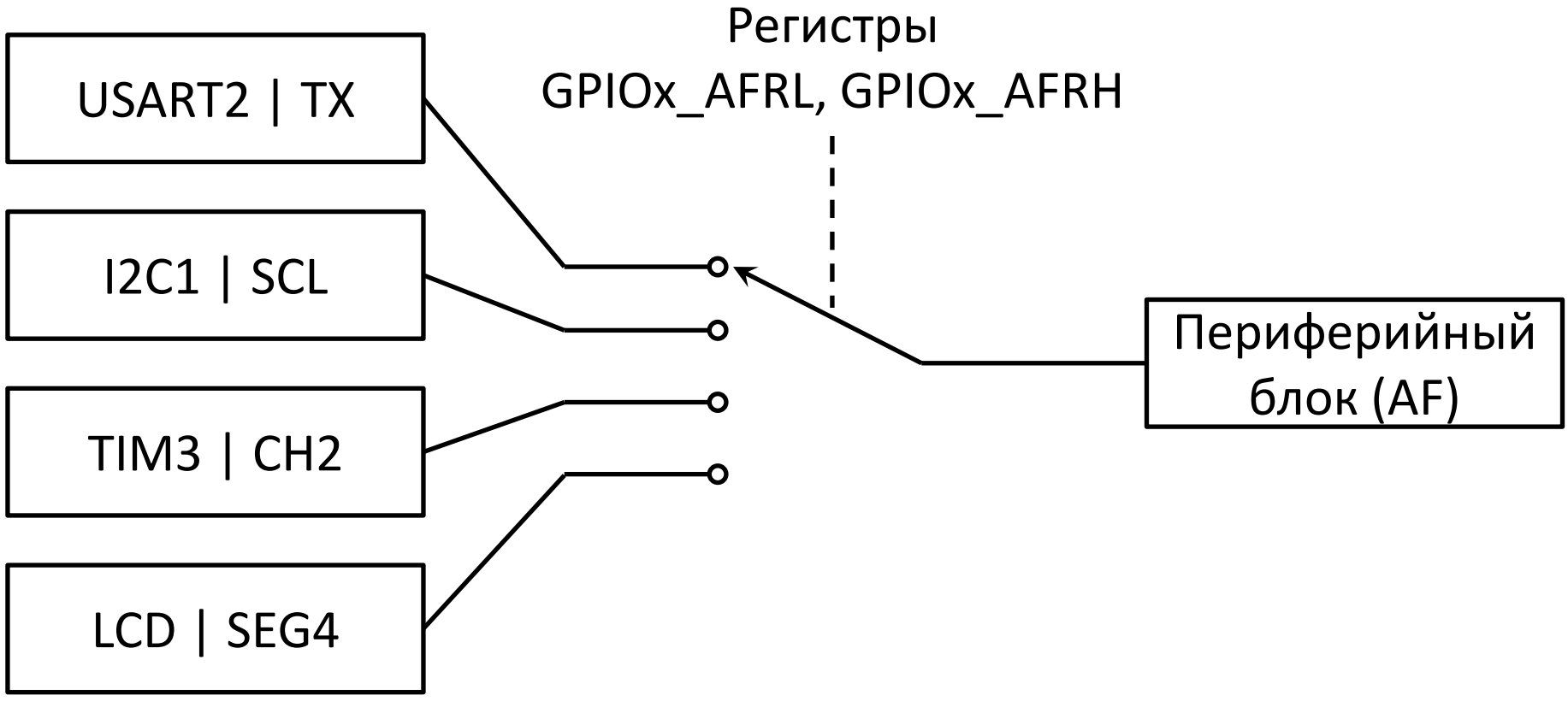
Структура вывода GPIO



Режимы работы GPIO



Альтернативные функции



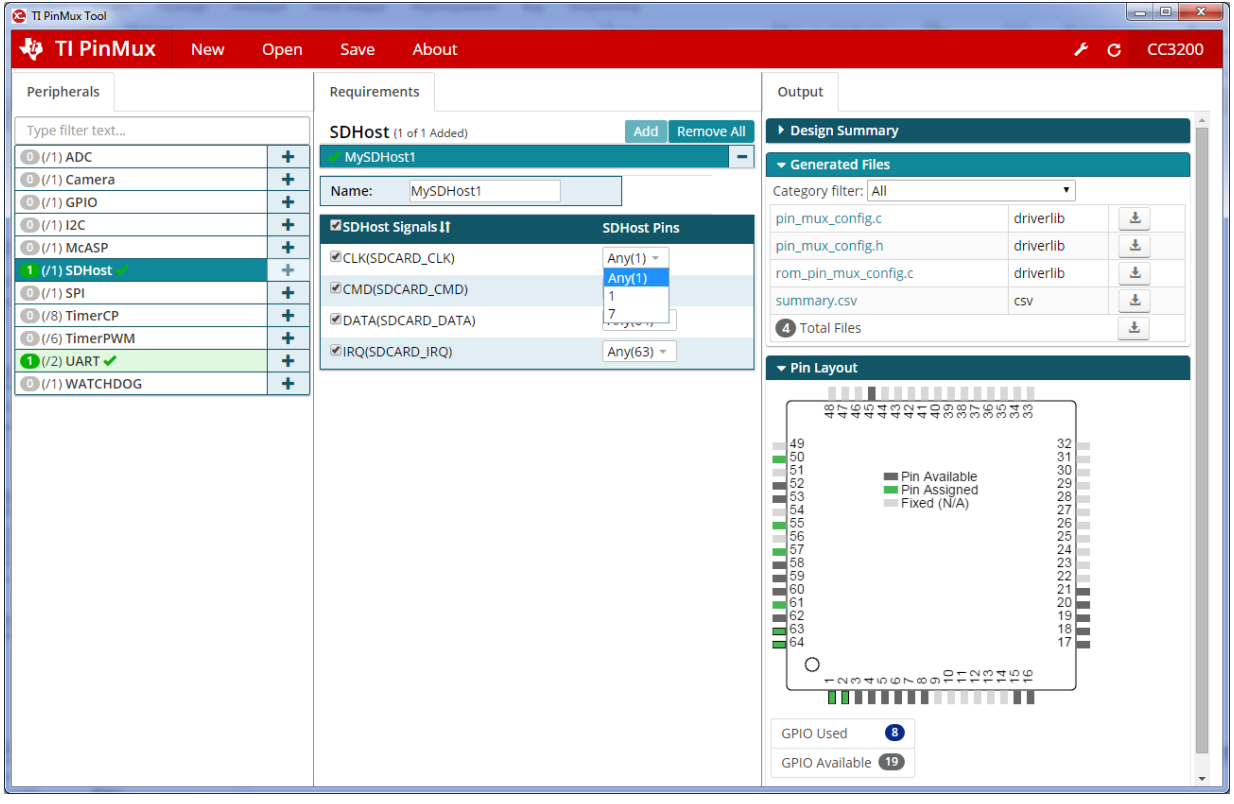
Альтернативные функции

Port name	Digital alternate function number														
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFOI6	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

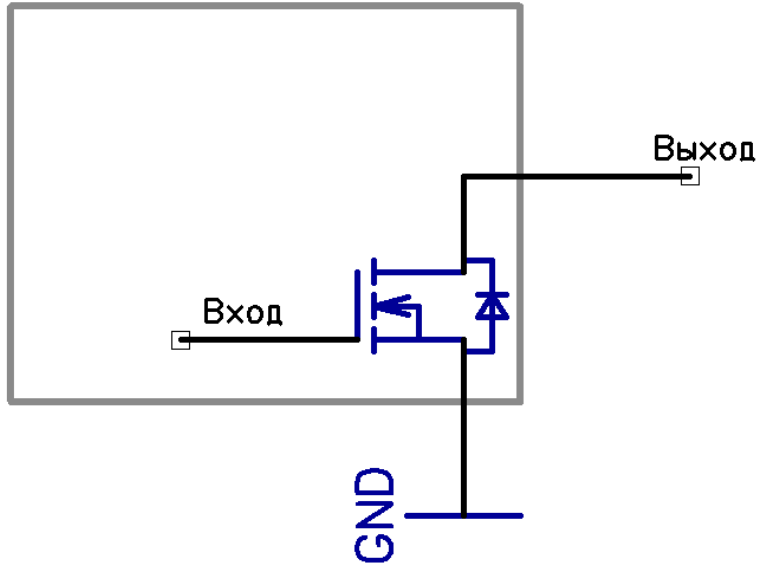
Альтернативные функции

- Полная матрица переключений
  - любой GPIO может выполнять любую функцию (кроме аналоговых входов)
  - Texas Instruments CC13xx, CC26xx, Nordic Semiconductor nRF52
- Ограниченная матрица переключений
  - каждый периферийный блок может подключаться к 2-4 разным GPIO
  - STM32 и большинство других микроконтроллеров
- Без матрицы переключений
  - каждый периферийный блок привязан к конкретному GPIO
  - AVR (ATTiny, ATMega), PIC, большинство восьмибитных МК

Альтернативные функции

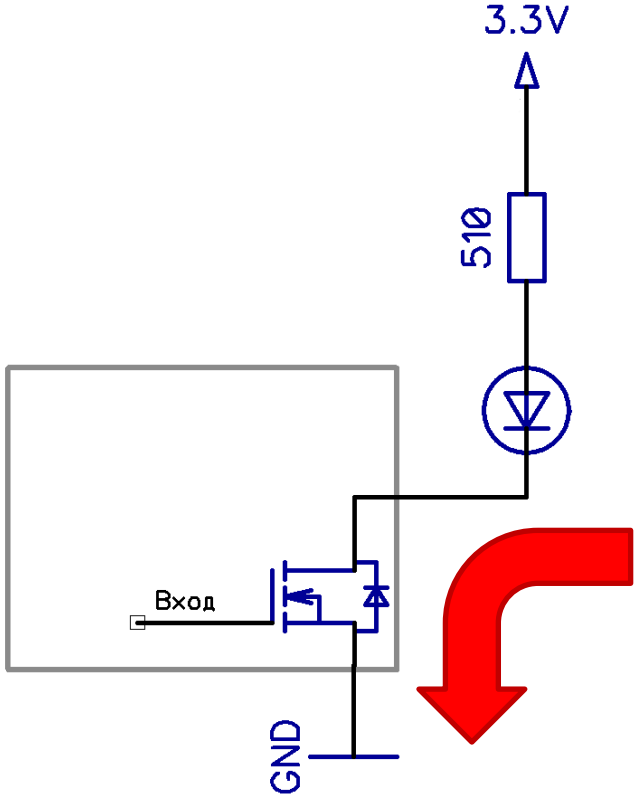


Выход с открытым стоком

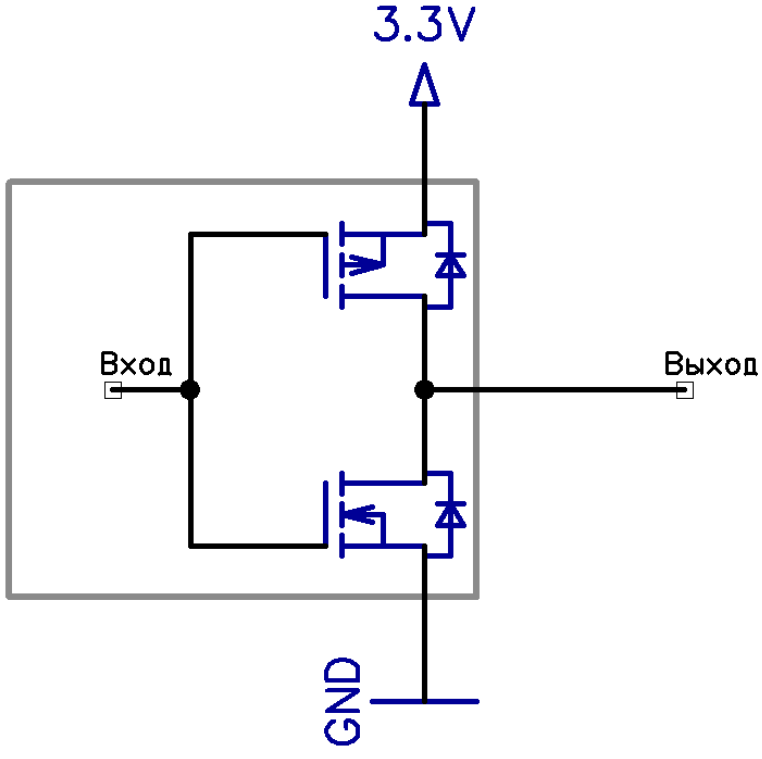


- Он же «открытый коллектор», он же «open drain»

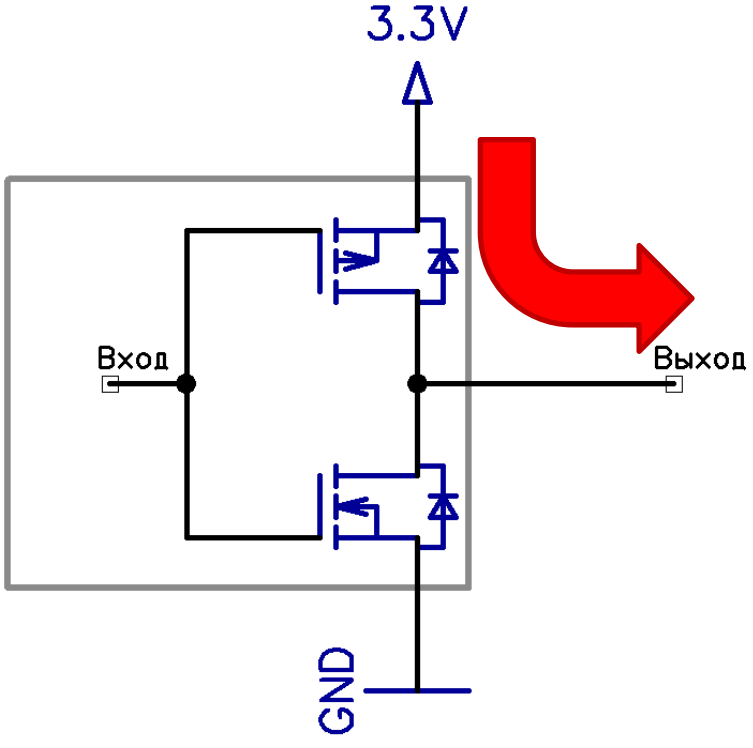
Выход с открытым стоком



Push-pull

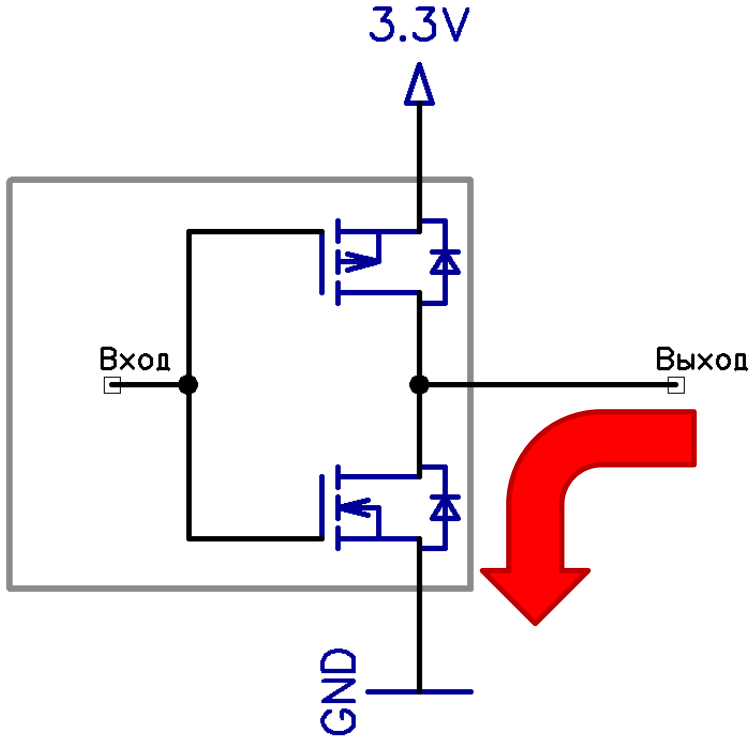


Push-pull



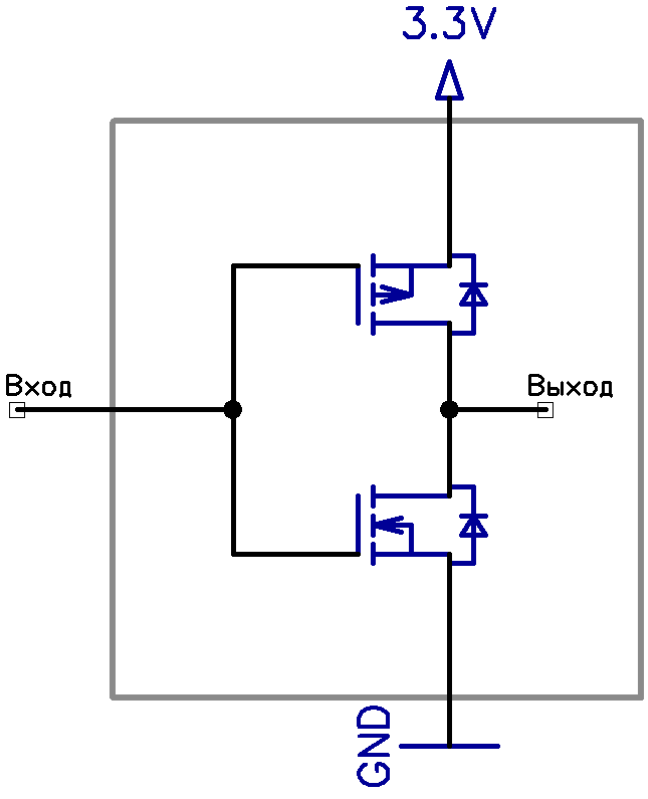
- На входе низкий уровень, верхний транзистор открыт, нижний закрыт
- Вытекающий ток (source current)

Push-pull

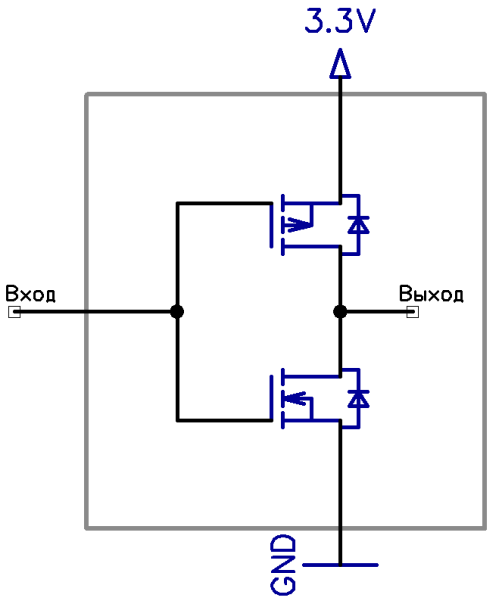


- На входе высокий уровень, верхний транзистор закрыт, нижний открыт
- Втекающий ток (sink current)
- Часто втекающий ток может быть больше вытекающего – смотрите даташит!

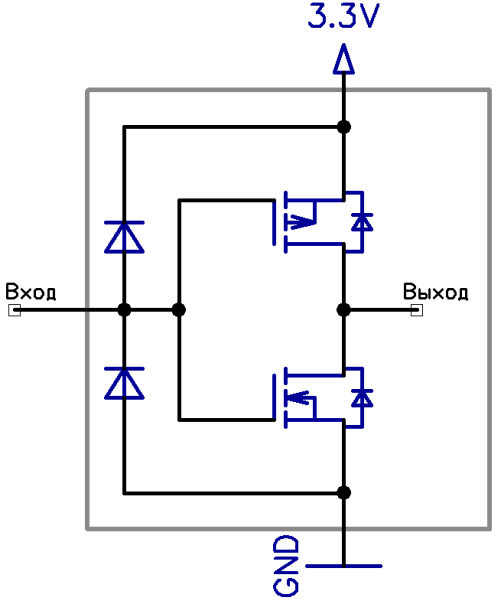
Цифровой вход



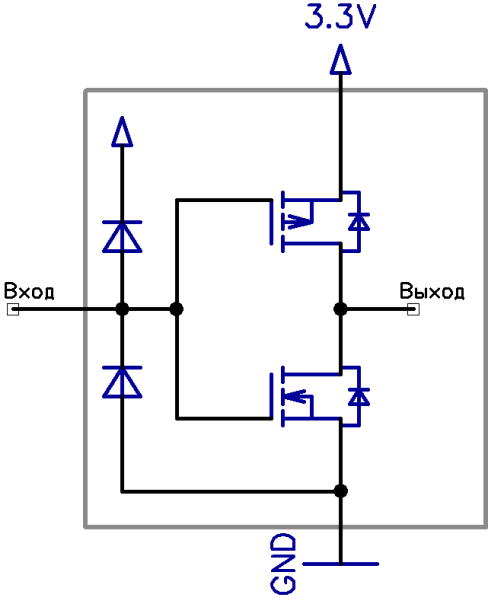
Цифровой вход с защитой



Без защиты

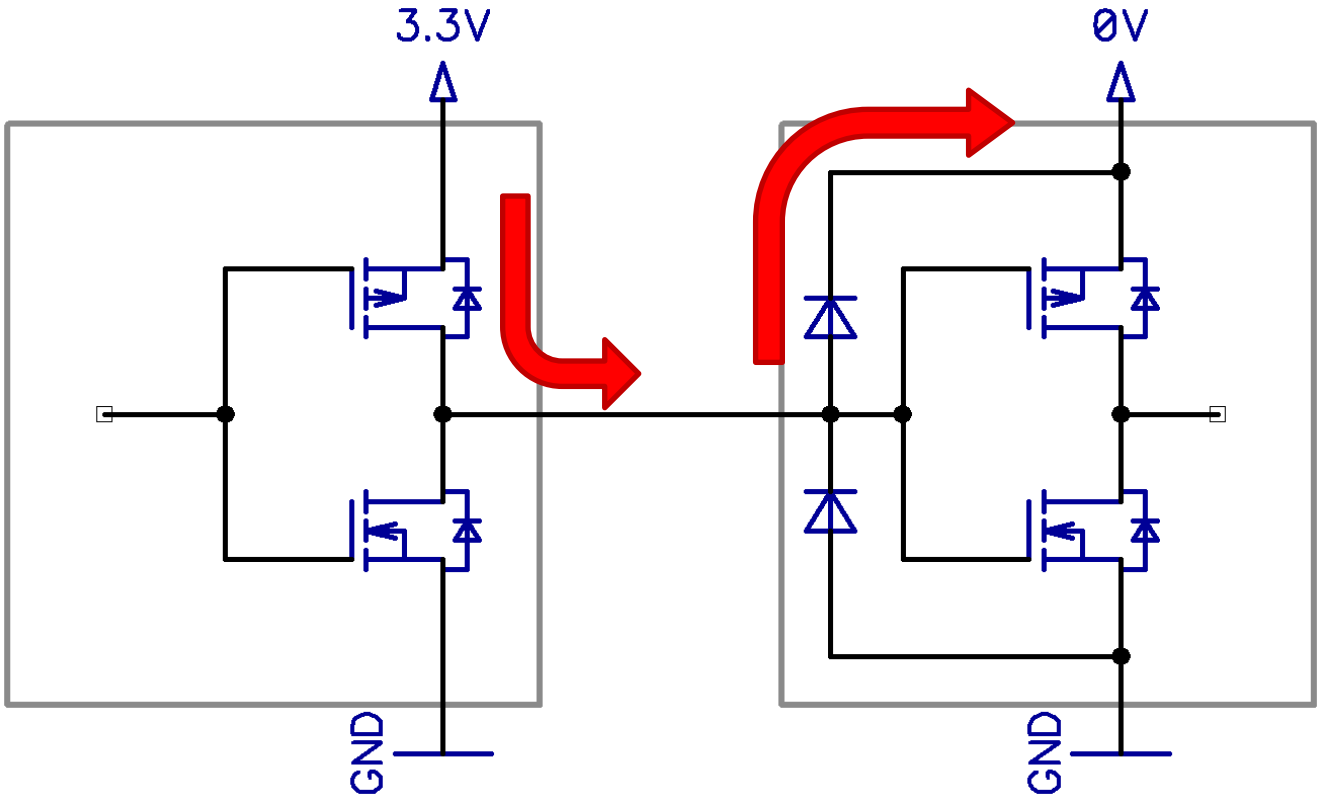


Защита от статики

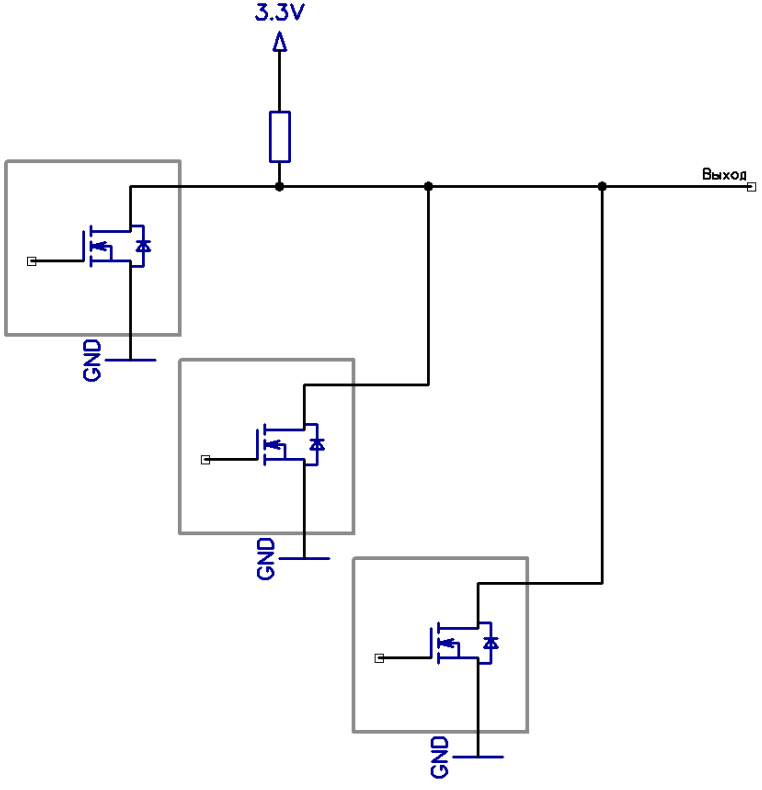


5V-tolerant

Паразитное питание

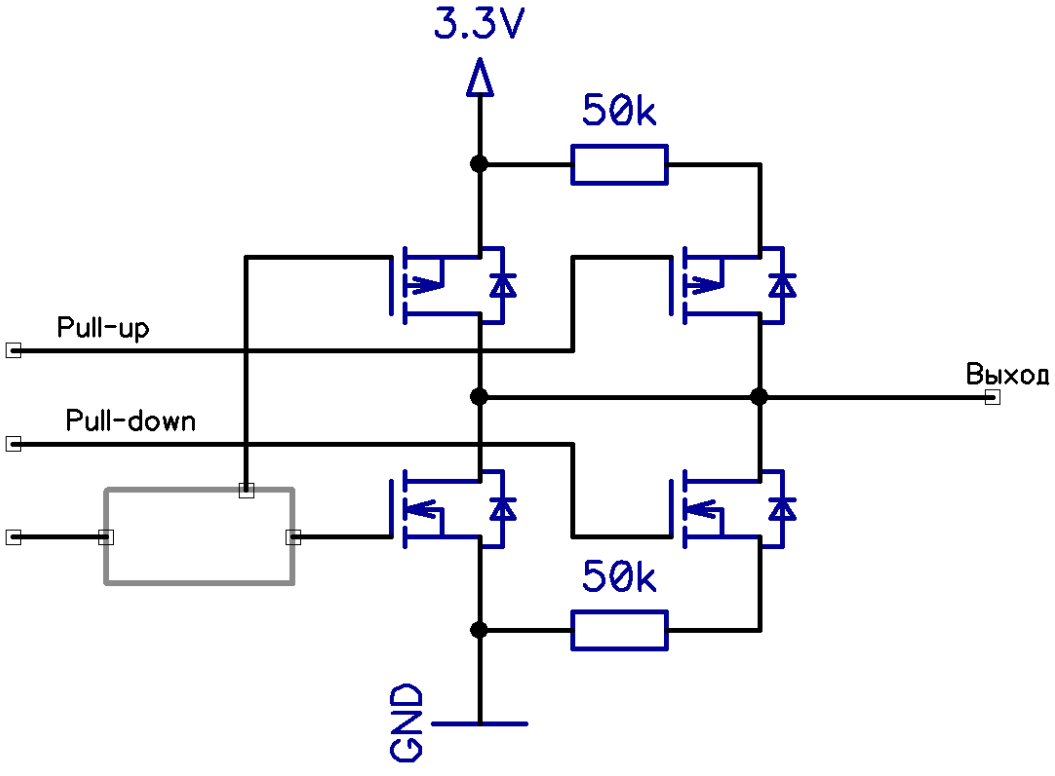


Открытый сток еще раз

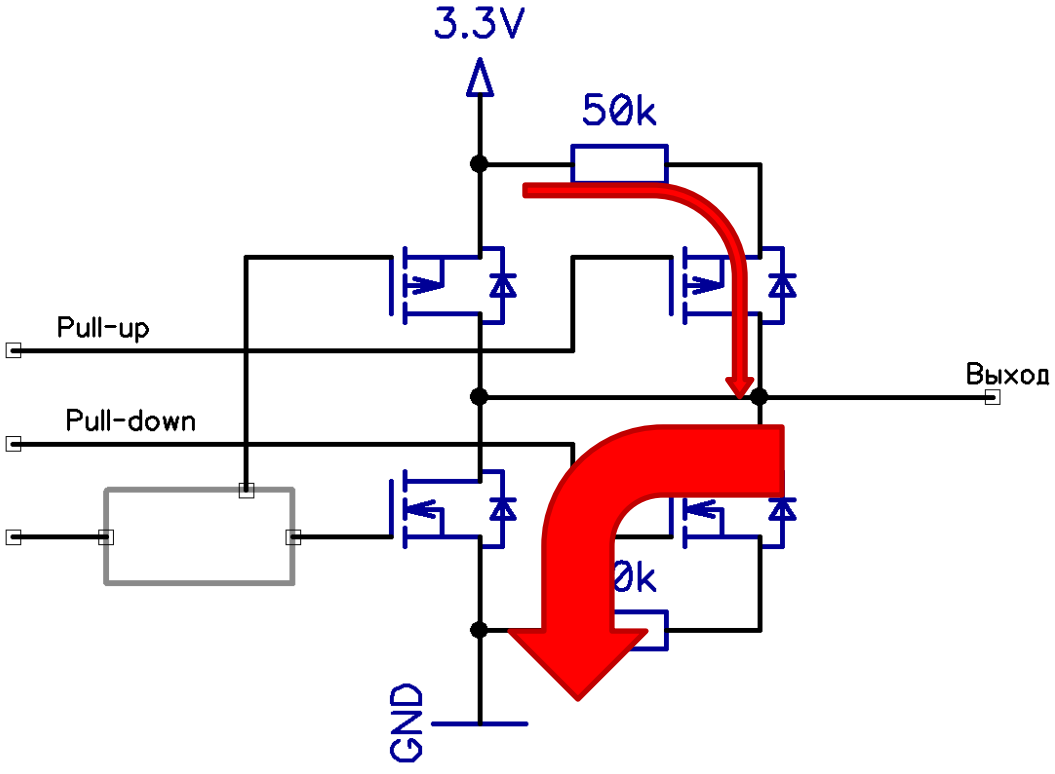


- Такое подключение применяется в шинах I2C и 1-Wire
- Одновременное включение “0” любым количеством устройств не приводит к выходу из строя других устройств на шине

Встроенная подтяжка



Встроенная подтяжка



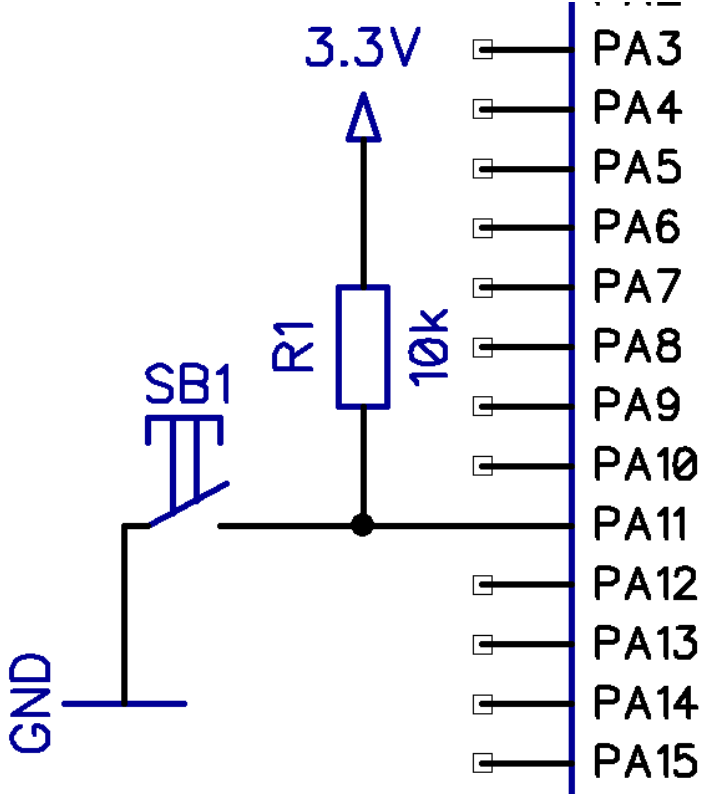
Подведем итоги

- Конфигурация GPIO

```
gpio_init(GPIO_PIN(PORT_A, 5), GPIO_OUT);
```

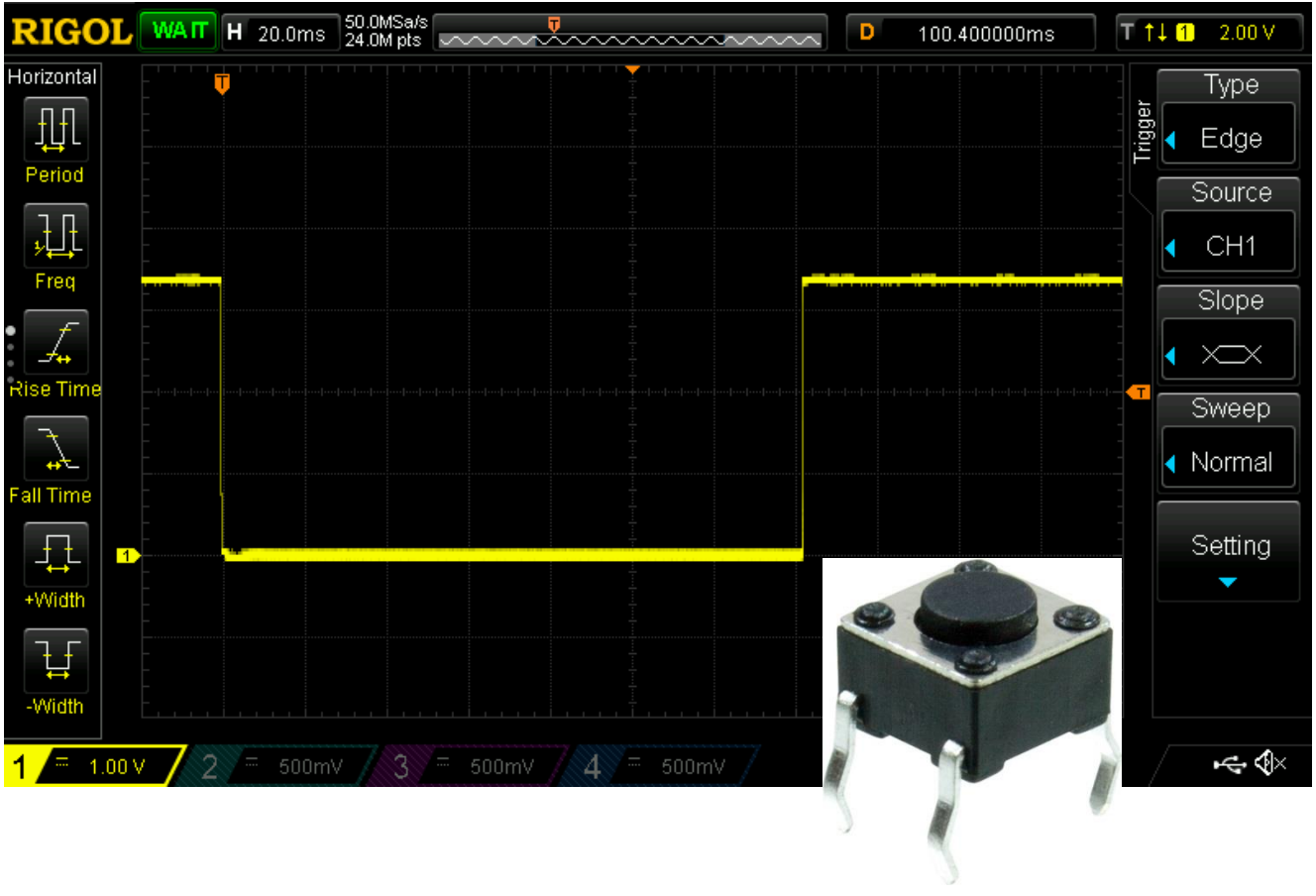
- Варианты: `GPIO_IN`, `GPIO_IN_PD`, `GPIO_IN_PU`, `GPIO_OUT`, `GPIO_OD`, `GPIO_OD_PU`
- Чтение – функция `gpio_read`
- Запись: `gpio_set`, `gpio_clear`, `gpio_toggle`, `gpio_write`

Давайте подключим кнопку

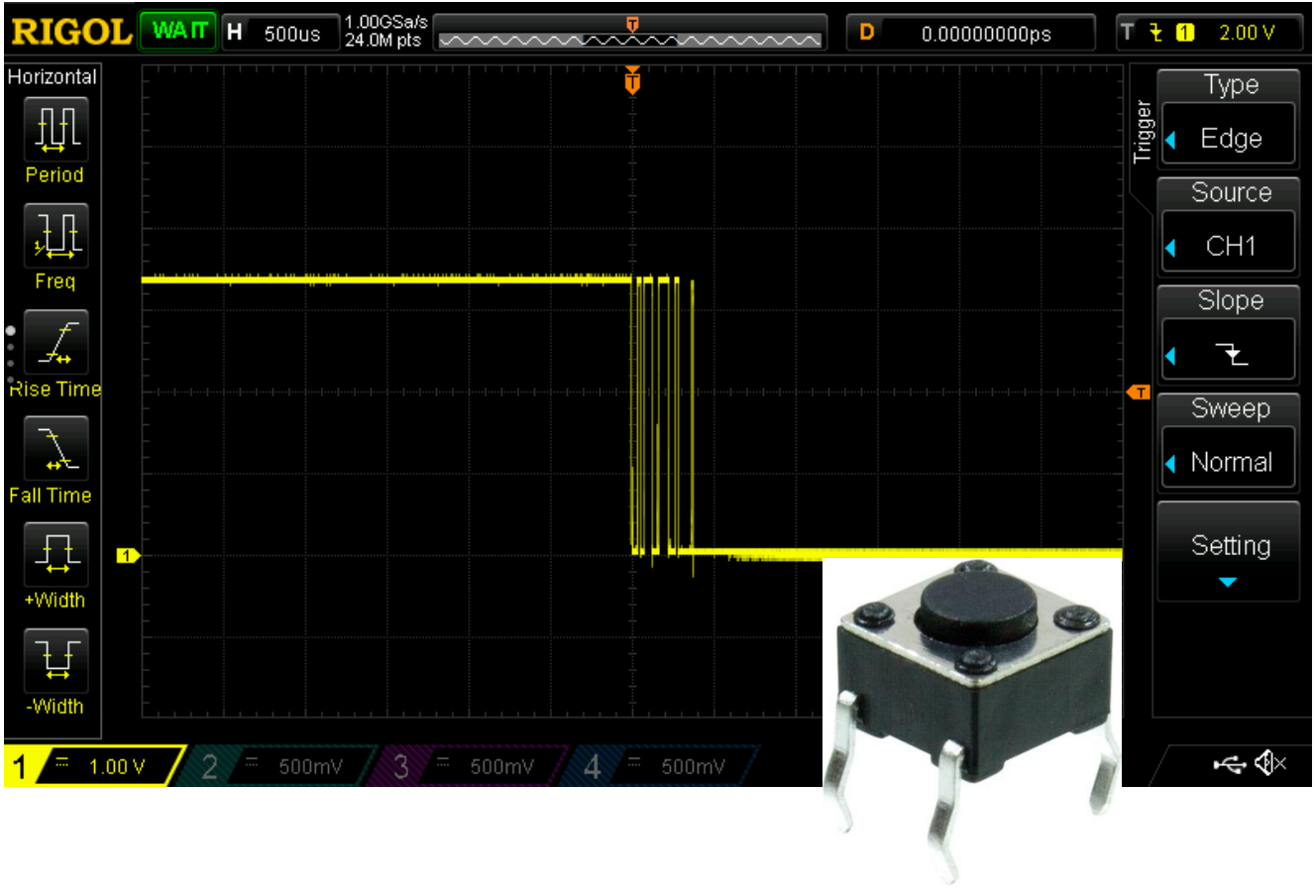


Давайте подключим кнопку

Давайте подключим кнопку



Давайте подключим кнопку



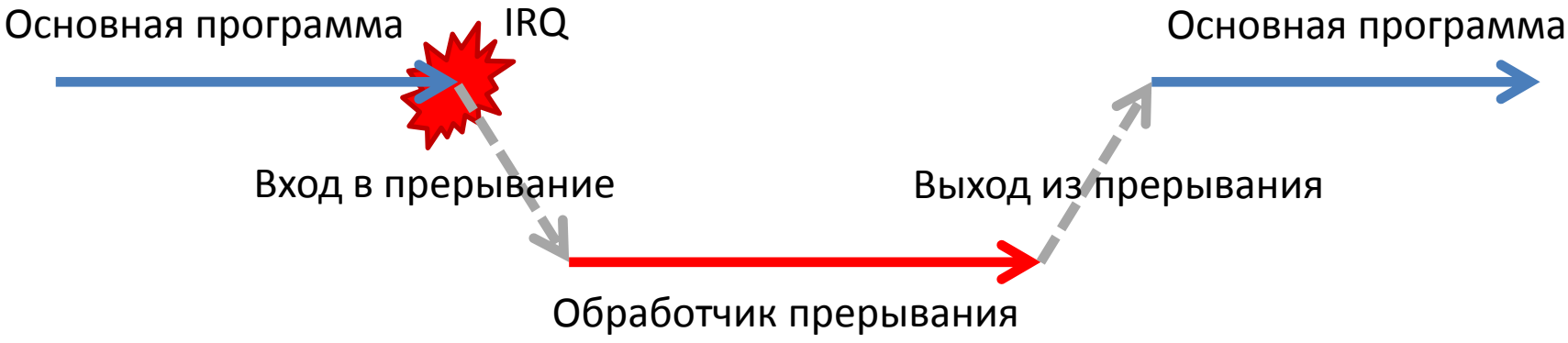
Дребезг контактов

- Два метода борьбы:
  - Периодический опрос
  - Отключение прерываний

ПРЕРЫВАНИЯ

Прерывания

- Прерывание – аппаратный сигнал, на который процессор обязан немедленно отреагировать
- Прерывание может быть отключено (замаскировано), за исключением NMI



Обработка прерываний

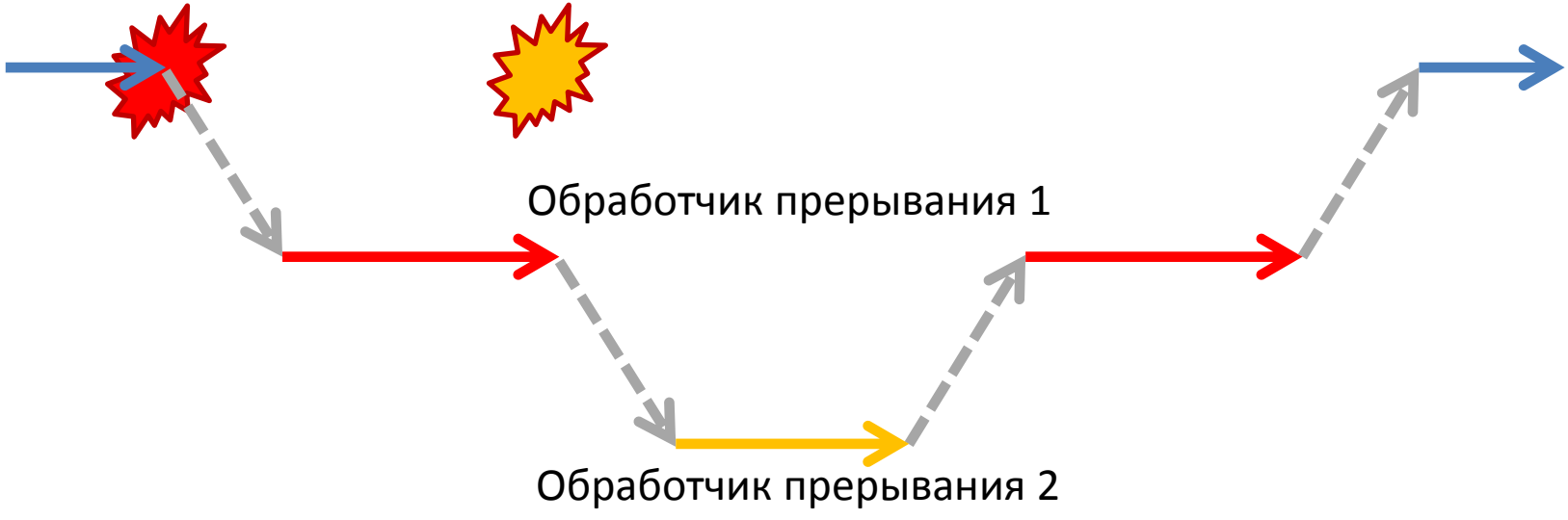
- NVIC – Nested Vectored Interrupt Controller
- Поддерживается таблица функций-обработчиков прерывания

```
typedef void (*isr_t)(void);
```

- Вход в прерывание = вызов функции со всеми полагающимися атрибутами (сохранение регистров на стеке и т. д.)
- Приоритеты прерываний, вложенные прерывания

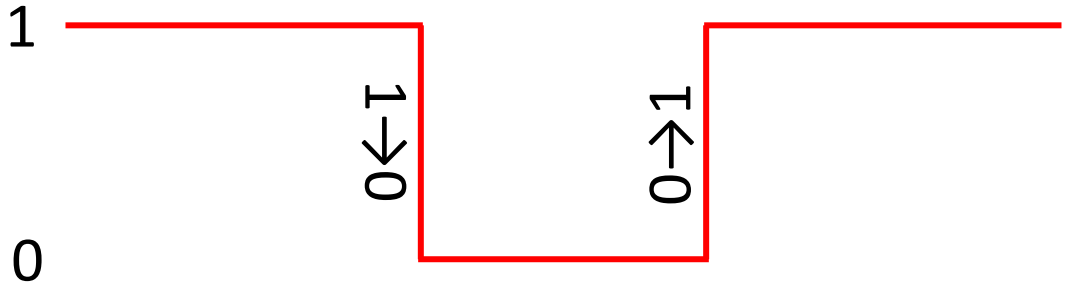
Вложенные прерывания

- Приоритеты прерываний – что будет, если в ходе обработки прерывания произойдет еще одно внешнее событие?



Прерывание GPIO

- Срабатывает по фронту сигнала
  - Передний фронт (0→1)
  - Задний фронт (1→0)
  - Оба фронта



Прерывания GPIO в RIOT

- Регистрируем функцию для обработки прерывания, например:

```
void btn_handler(void *arg);
gpio_init(GPIO_PIN(PORT_A, 11),
          GPIO_IN_PU,
          GPIO_FALLING,
          btn_handler,
          (void*) 211);
```

- Чтобы включить или выключить конкретное прерывание, используем функции `gpio_irq_disable`, `gpio_irq_enable`

Прерывания в RIOT (для Cortex-M)

- Поддержка вложенных прерываний отсутствует, все прерывания имеют одинаковый приоритет
- Таблица прерываний в ROM, обработчики прерываний описаны в драйверах периферии, например:
  - в файле `cpu/stm32l1/vectors.c` объявлена функция `isr_exti` (с модификатором `weak`)
  - ее реализация – в файле `cpu/stm32_common/periph/gpio.c`
- `В конце обработчика прерывания всегда вызывается функция cortexm_isr_end()`

Прерывания в RIOT

```
void isr_exti(void){
    /* only generate interrupts against lines which have their IMR set */
    uint32_t pending_isr = (EXTI->PR & EXTI->IMR);
    for (size_t i = 0; i < EXTI_NUMOF; i++) {
        if (pending_isr & (1 << i)) {
            EXTI->PR = (1 << i);        /* clear by writing a 1 */
            isr_ctx[i].cb(isr_ctx[i].arg);
        }
    }
    cortexm_isr_end();
}
```

Для других архитектур все может быть устроено немного по-другому

ТАЙМЕРЫ

Аппаратные таймеры

- Таймер – счетчик с возможностью вызывать прерывания
- Разрядность – 8, 16, 24 бита
- Настройки:
  - Источник тактирования (внутренний генератор или один из входов)
  - Делитель
  - Регистр сравнения («будильник»); может быть несколько
- Прерывания по переполнению или по совпадению значения с регистром сравнения

Программные таймеры в RIOT

- Слой абстракции поверх аппаратных таймеров
- Позволяет завести любое количество программных «будильников»
- `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

МНОГОЗАДАЧНОСТЬ

Зачем нужна многозадачность?

```
int main(void){
    puts("Hello World!");
    while (1){
        task1();
        task2();
        task3();
    }
    return 0;
}
```

- Бесконечный цикл, он же loop() в Arduino
- Очень быстро сталкиваемся с тем, что задания «мешают» друг другу
- Можно вынести логику в прерывание одного из таймеров (SYSTICK)
- Пример – отладчик UMDK-EMB

Виды многозадачности

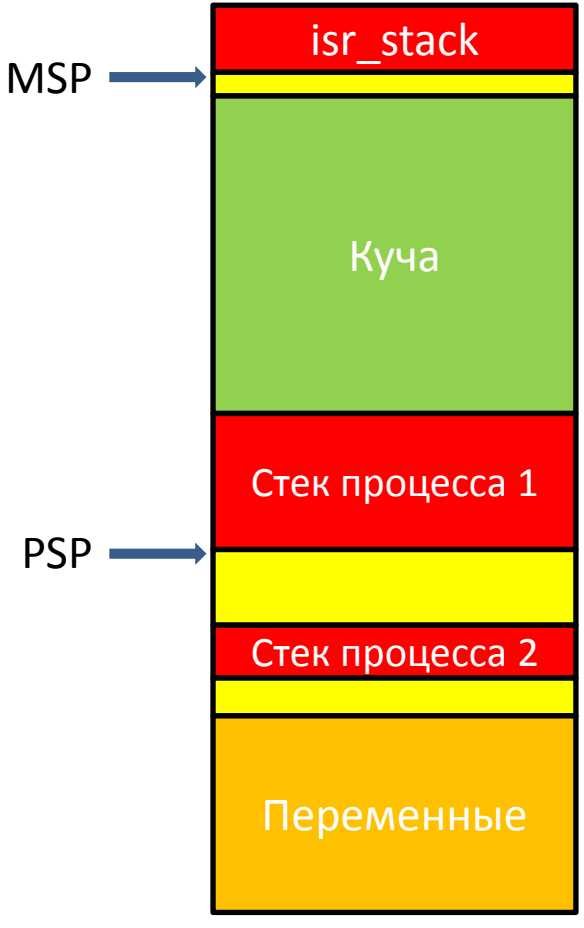
- Вытесняющая многозадачность
  - Планировщик вызывается по прерыванию таймера
  - «Повисшая» задача не останавливает остальные
  - Повышенный расход энергии
- Кооперативная (невытесняющая, tickless) многозадачность
  - Планировщик вызывается при переходе очередной задачи в режим ожидания
  - «Повисшая» задача останавливает работу системы
  - Когда все задачи находятся в режиме ожидания – можно включить энергосберегающий режим

Аппаратная поддержка многозадачности в Cortex-M

- Таймер SYSTICK (24 бита, тактирование от основного тактового генератора)
- Два указателя стека

  MSP – main stack pointer

  PSP – process stack pointer
- В большинстве ОС стек процесса – это просто достаточно большой массив char



Многозадачность в RIOT

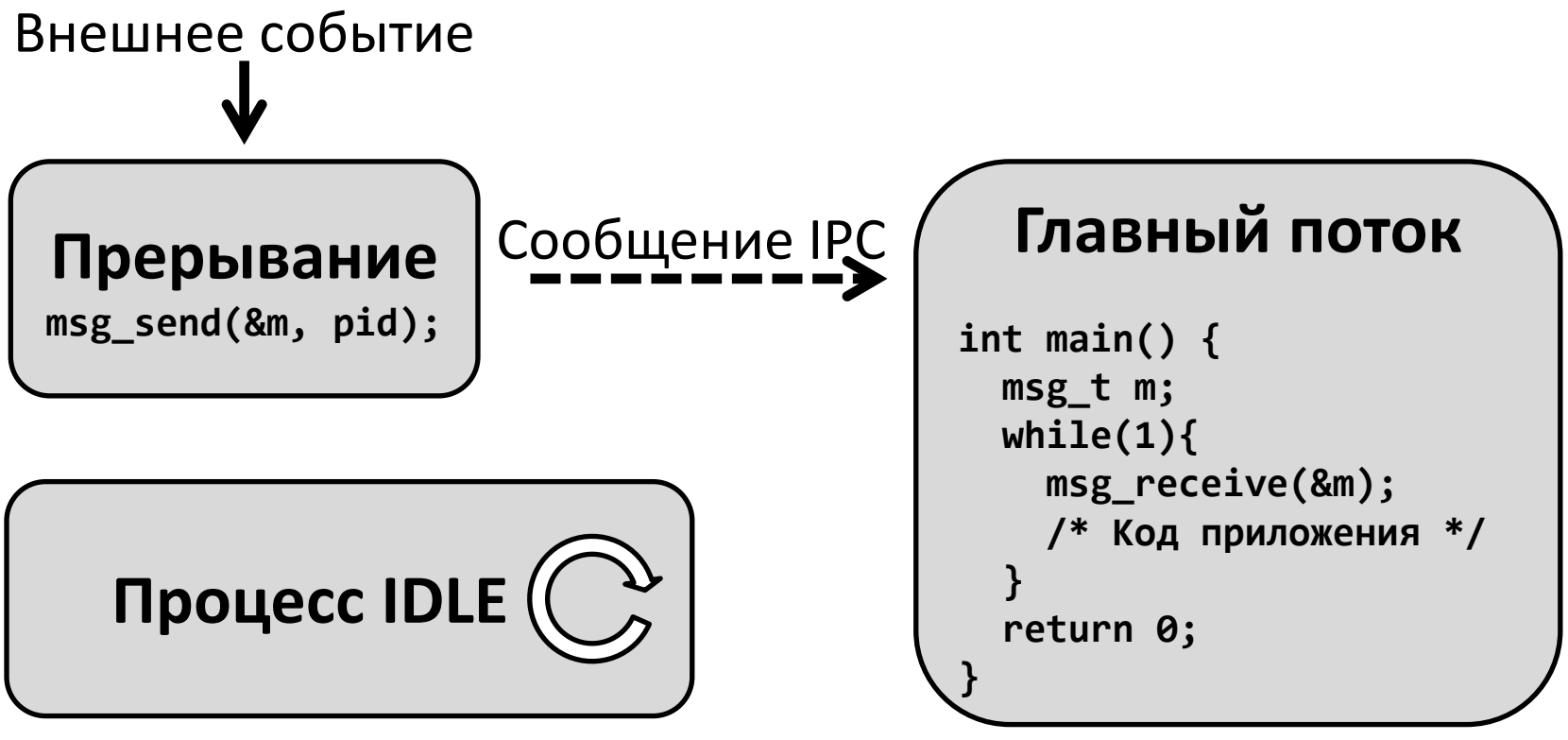
- Tickless-планировщик, сложность O(1)
- 16 приоритетов процессов (процесс/поток/задача – это одно и то же)
  - 0 – высший приоритет, 15 – низший (для процесса IDLE)
  - Процесс main имеет приоритет 7
- Прерывание может прервать работу любого процесса
- По завершении прерывания может быть вызван планировщик

Создание процесса

```
static char my_stack[THREAD_STACKSIZE_DEFAULT];
void* my_thread(void* arg){
    while(1){
        lptimer_sleep((int) arg);
        gpio_toggle(LED0_PIN);
    }
}

int main(void) {
    thread_create(my_stack, sizeof(my_stack),
                  THREAD_PRIORITY_MAIN-1,
                  THREAD_CREATE_STACKTEST,
                  my_thread,
                  (void*) 500,
                  "My own thread");
    return 0;
}
```

Межпроцессное взаимодействие



На практическом занятии

- Примеры многопоточных программ
- Межпроцессное взаимодействие
- API таймеров