Глава 1. Основы

Ежелев Г. И.

30 января 2026 г.

Что вы знаете об устройстве микроконтроллеров?

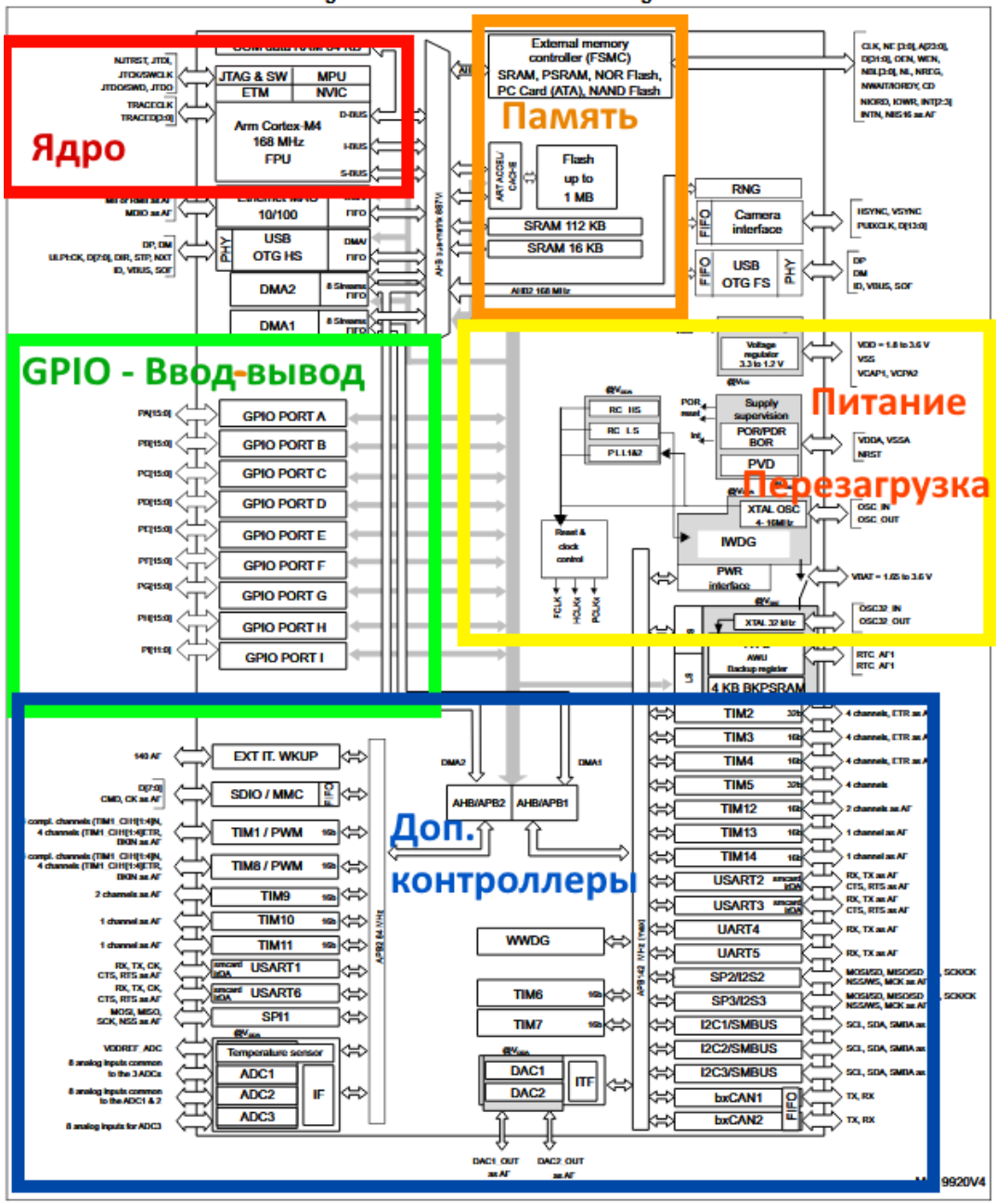
Основные элементы

- Ядро - Отвечает за выполнение вашей программы, управляет всеми устройствами МК. Самое быстрое устройство в составе контроллера.
- Ножки микроконтроллера - Pins
- Дополнительные контроллеры ввода-вывода, отвечающие за отдельные функции ножек, например контроллеры цифровых протоколов передачи данных UART, IIC или контроллер чтения аналогового сигнала
- Внутренние обработчики событий - подсчет времени, прерывания и т.д.
- Шины - Соединяют все устройства воедино
- Тактовый генератор - Генерирует импульсы заданной частоты. Все вычисления происходят строго в момент этих импульсов. К концу такта все вычисления должны быть завершены.

Структурная схема - Datasheet стр. 19

2.2 Functional overview

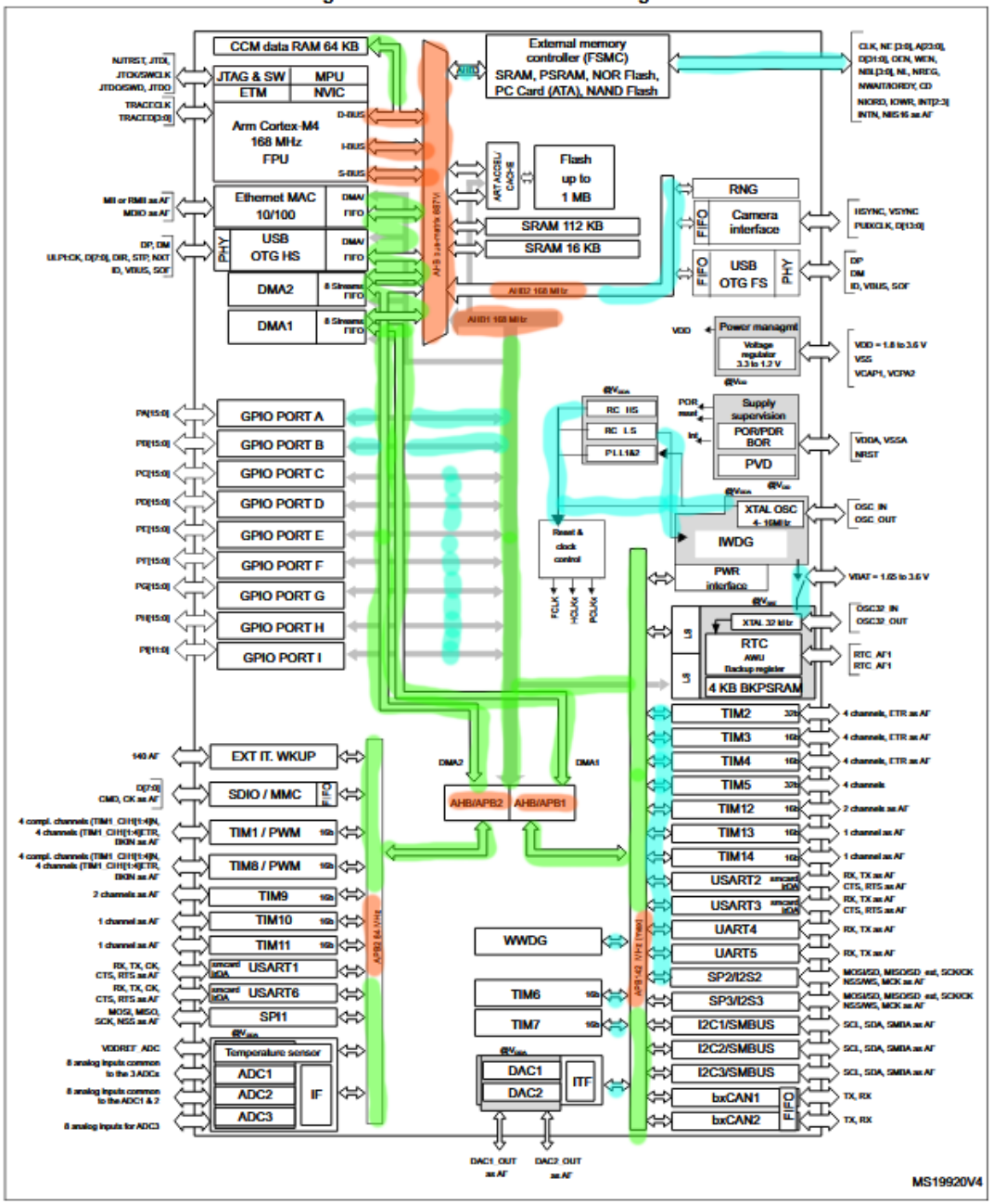
Figure 5. STM32F40xxx block diagram



А где же шины?

2.2 Functional overview

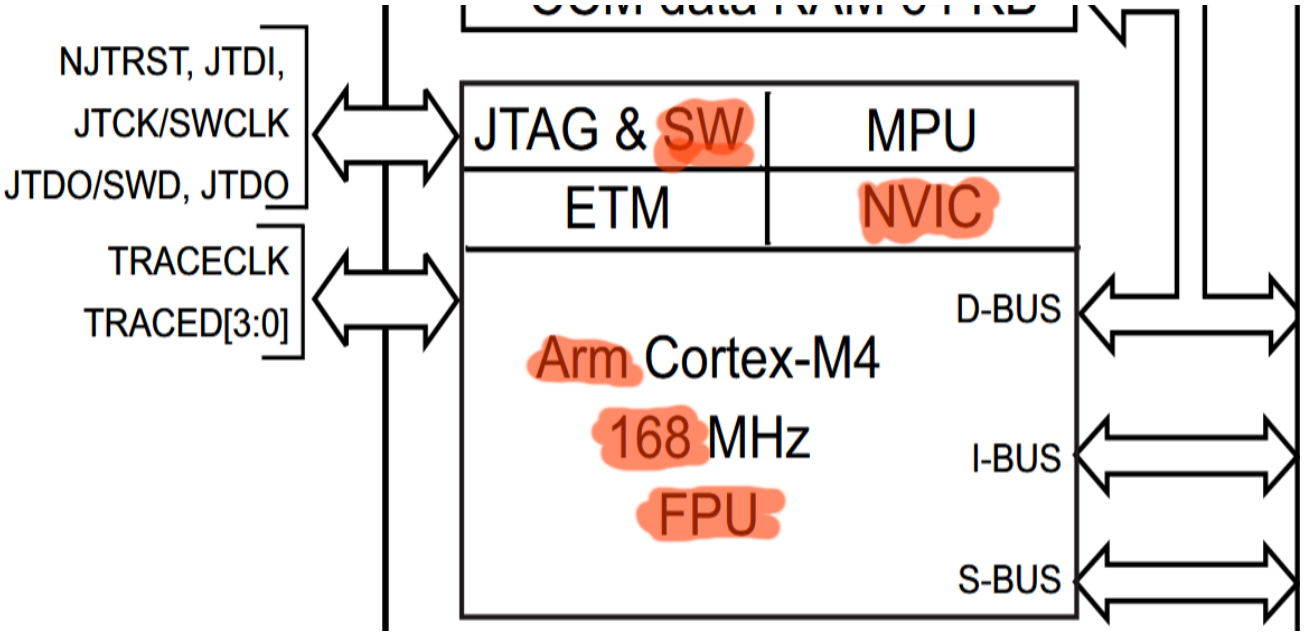
Figure 5. STM32F40xxx block diagram



Ежелев Г. И.

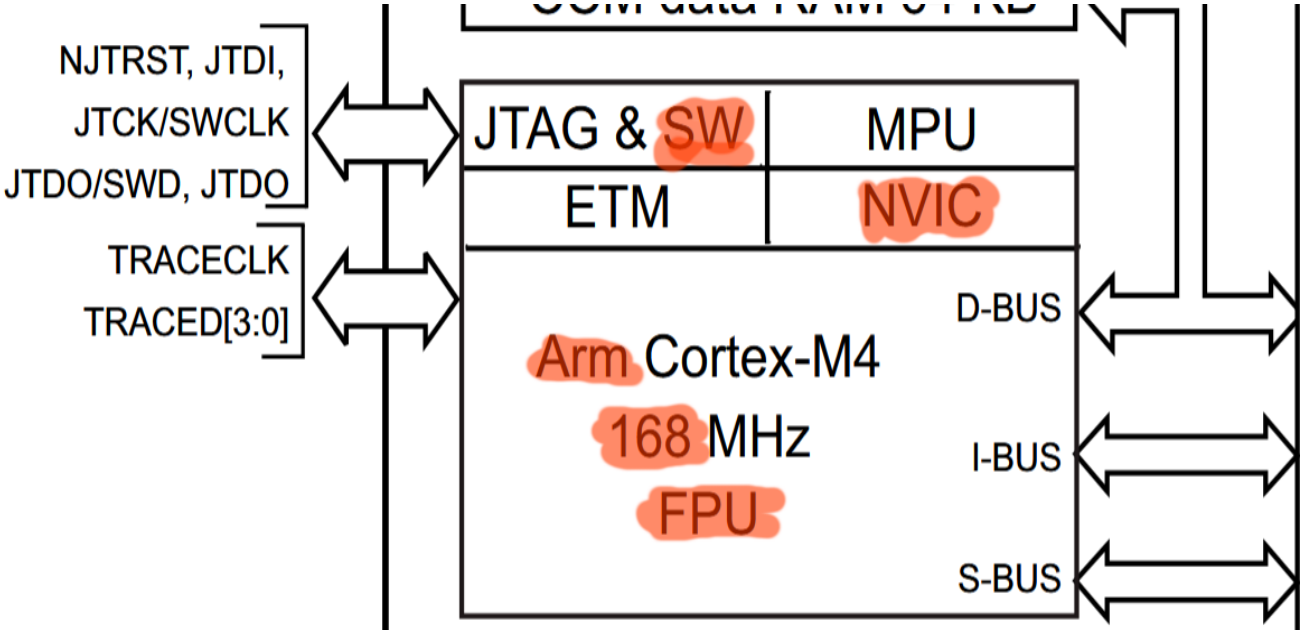
Советую открыть этот даташит (он есть в репозитории!) и самостоятельно вглядеться - стр. 19. На второй картинке названия шин, отмеченные оранжевым, нам пригодяться, запомните их.

Ядро



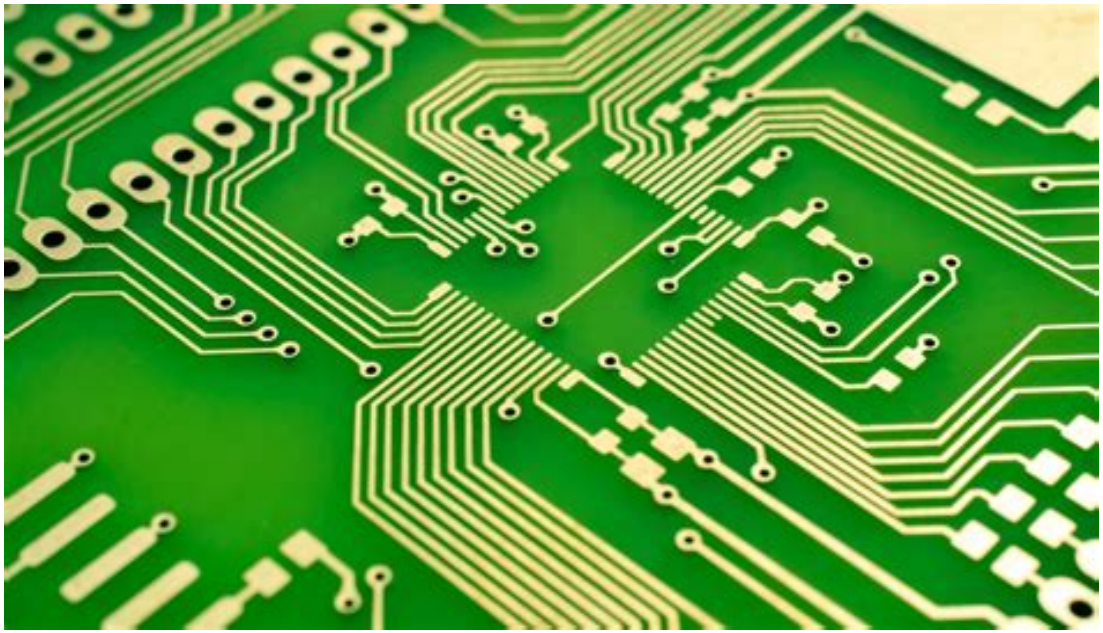
- Архитектура ядра - ARM - энергоэффективная. Как в ваших телефонах. В ПК часто ставят более производительную, но прожорливую X86. Архитектура - это то, какие базовые команды реализованы внутри ядра транзисторами. Другими словами, какие команды ядро умеет исполнять сразу. Все что ядро не умеет делать - разбивают на команды поменьше и выполняют.

Ядро



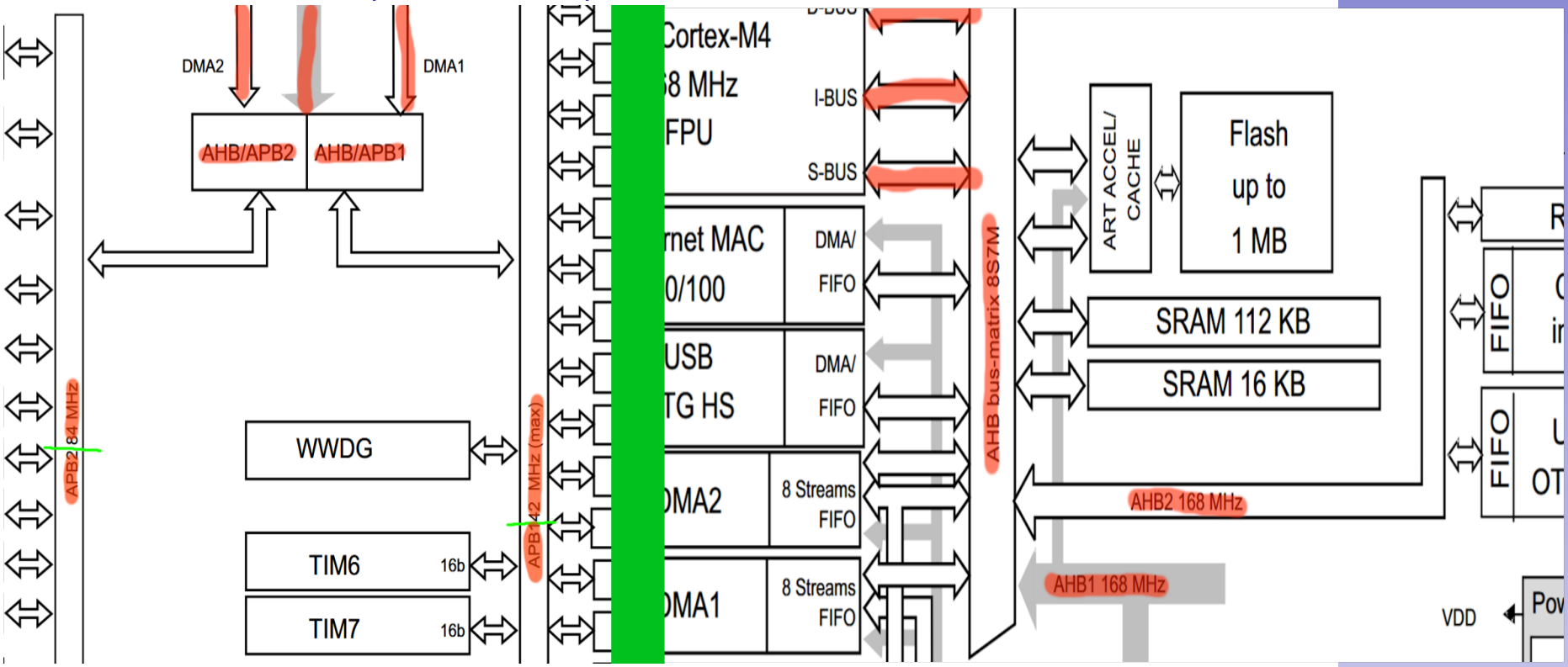
- Частота ядра - 168 мегагерц. Частота - сколько таких простых операций в секунду оно умеет делать. В нашем случае 168 миллионов.
- FPU - Floating Point Unit - ускоритель вычисления операций с дробными числами. Микроконтроллерам тоже тяжело складывать дроби!
- В Arduino нет FPU и дробные числа там складываются на порядки дольше.

Шины



- Шины - Много-много проводов / дорожек / проводников собранных вместе, чтобы передавать много много данных. Каждому проводу - своя (далеко не обязательно универсальная) функция
- Обычно, процесс передачи данных по шине выглядит так: либо сначала передается адрес, а следующим тактом на тех же проводах значение, или есть отдельные провода под адрес и под значение - передача мгновенная (быстрее передача, но на плате более громоздко)

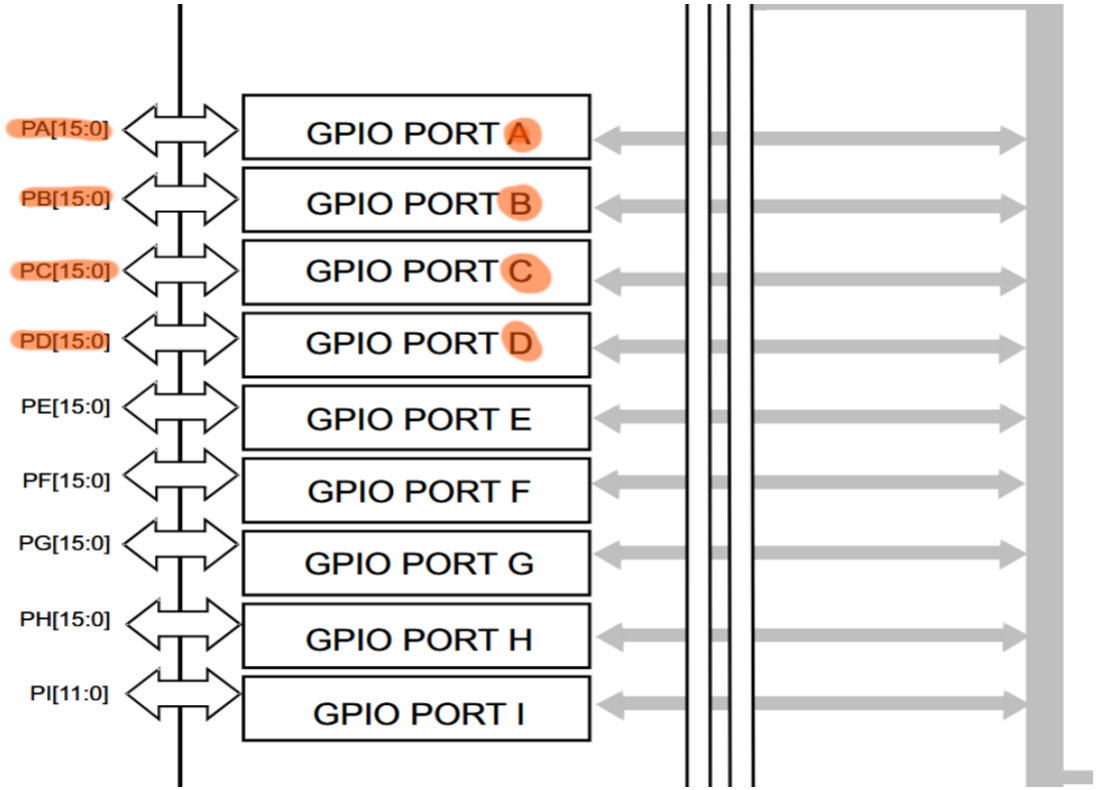
Шины - AHB, APB 1,2



- Ядро подключено ко всем устройствам через шины. В программе обращение к устройству происходит через его шину. Скорость шины - скорость устройства.
- AHB - высокоскоростная шина. На ней висят самые важные устройства - память, ввод вывод, питание и т.д.
- APB1,2 - Все остальное.

GPIO

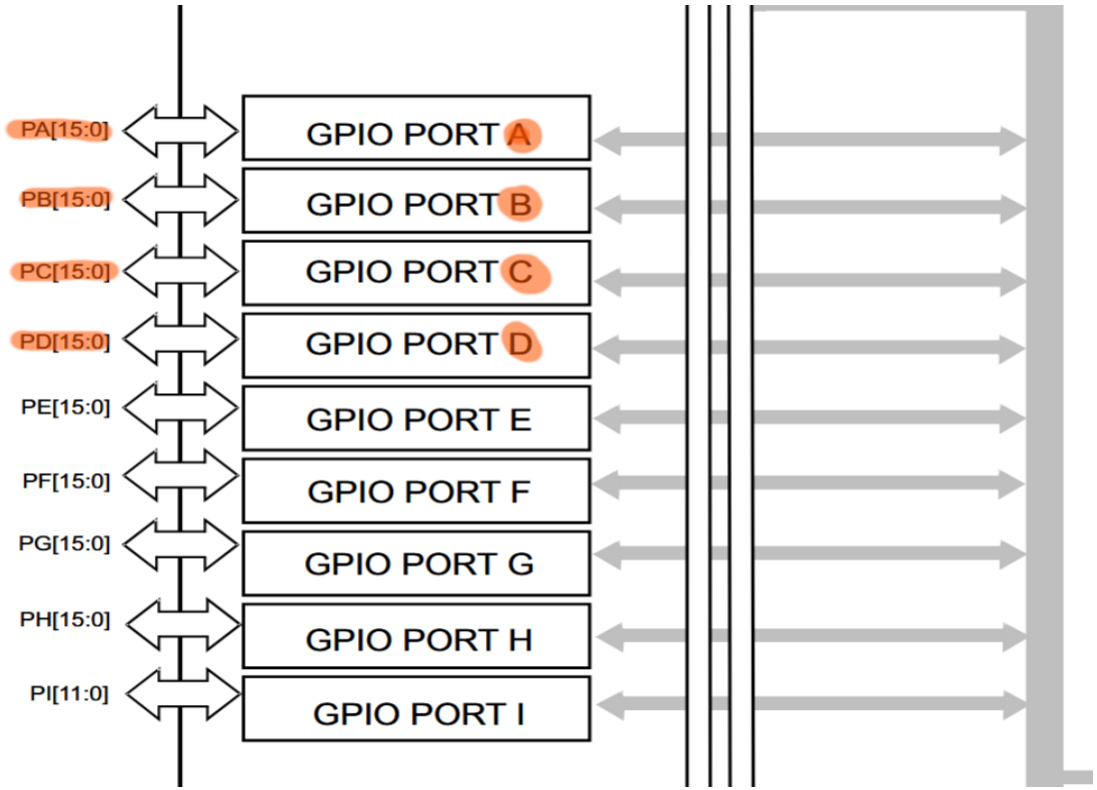
Последняя порция теории перед кодом!



- GPIO - General Purpose Input-Output - Вход-выход основного назначения.
- Контроллер GPIO - GPIO Port обрабатывает поведение каждого пина (вывода, ножки) микроконтроллера. В каком он режиме, чтение сигнала и т.д.

GPIO

Последняя порция теории перед кодом!



- Портов несколько A,B,C... . На каждом 16 пинов - от 0 до 15. Например 13 пин контроллера C - это GPIO Port C Pin 13 или сокращенно PC13. В Ардуино 2(3) порта - A и D(поделен под капотом на 2).

Код

Взаимодействие ядра со всеми устройствами происходит через регистры - микроячейки памяти в 32 бита.

Контроллер 32 битный - потому что за одну операцию он может обработать максимум 32-битное число.

Запись в регистр - управление устройством, чтение регистра - получение от него данных

Каждый бит в регистре имеет свое назначение. Все регистры описаны в Reference manual (не поверите, тоже есть в репозитории).

Регистр указывается так: устройство->регистр. Если устройств несколько мы будем заменять индексы на X для универсальности, например: GPIOX->IDR (что будет значить для конкретного устройства GPIOA GPIOA->IDR, а для GPIOB: GPIOB->IDR)

Рассмотрим регистр GPIOX->ODR

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)
These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..I/J/K).

Port Output data register - регистр, отвечающий за то, какой цифровой сигнал будет на каждой из 16 ножек конкретного порта GPIOX(GPIOA, GPIOD...).

Если на i-м месте будет 1 <=> на i-м пине будет высокий сигнал(3.3В).

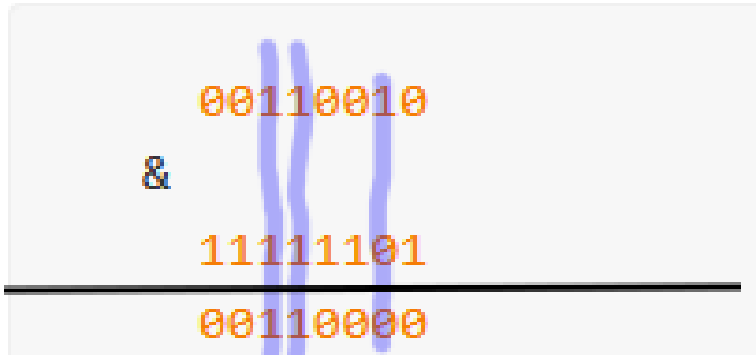
Если на i-м месте будет 0 <=> на i-м пине будет низкий сигнал.

Пример 000...0010. Все пины кроме первого - низкий сигнал (нумерация с 0).

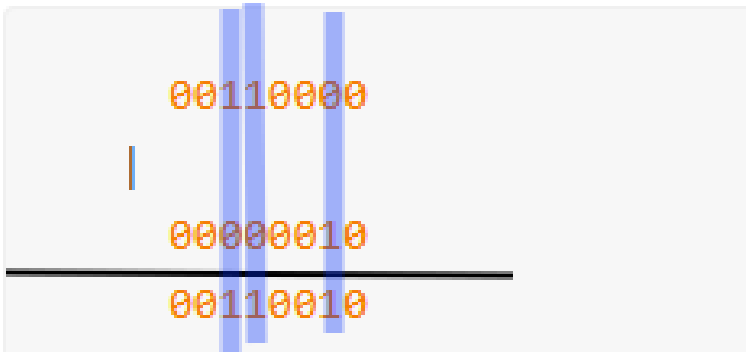
Как же записать?

Сначала нужно очистить конкретный бит. Потом поставить его значение. Будем использовать побитовые операции чтобы не изменить соседние биты.

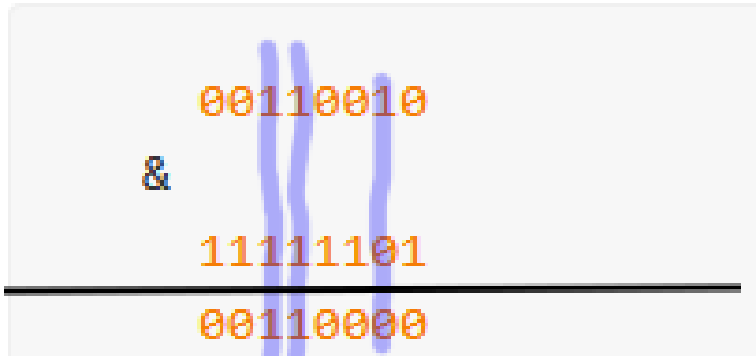
1. Установка бита в ноль



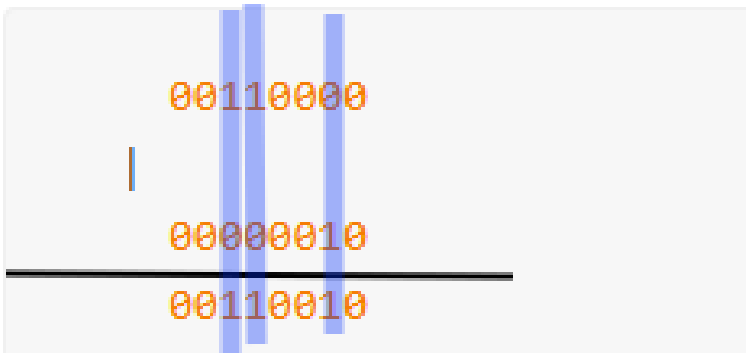
2. Установка бита в единицу



1. Установка бита в ноль



2. Установка бита в единицу



Тем самым мы сначала очистили бит 1, потом записали туда значение

В коде это будет выглядеть так (пример для GPIOC и пина PC7):

```
GPIOC->ODR &= ~GPIO_ODR_ODR_7;
GPIOC->ODR |= GPIO_ODR_ODR_7;
```

& - побитовое И
GPIO_ODR_ODR_7 - обозначение(define)
000...010000000 - 7-я единица, т.е. адрес бита, отвечающего за 7 пин. Таких дефайнов будет много - Сначала название устройства, далее название регистра, затем конкретный бит или группа битов в регистре
~ - тильда - побитовое отрицание, превращает 000...010000000 в 111...10111111. Применяя это число через логическое И к значению регистра как раз занулим 7-ой бит, остальные не тронем.
| - побитовое ИЛИ - выставляет эту 7 единицу в регистр, не трогая остальные.
Если нужно выставит на пине логический 0, то достаточно сделать только первую операцию (без побитового или).

Материалы курса можно найти по ссылке:
https://github.com/haaroner/ezh239

Настройка пина

Итак, теперь мы умеем подавать логическую единицу на пин. Давайте его настроим.

Запись каждого бита в регистры будет происходить аналогичным способом, так что разберем только суть.

```
1  #include "project_config.h"
2
3  int main()
4  {
5    SystemInit();
6
7    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
8
9    GPIOA->MODER &= ~GPIO_MODER_MODER1;
10   GPIOA->MODER |= GPIO_MODER_MODER1_0;
11
12   GPIOA->PUPDR &= ~GPIO_PUPDR_PUPDR0;
13   GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;
14
15   GPIOA->ODR &= ~GPIO_ODR_ODR_1;
16   GPIOA->ODR |= GPIO_ODR_ODR_1;
17 }
```

- Строчка 1 - подключение файла со всеми необходимыми библиотеками
- Строчка 5 - сброс всех регистров
- Строчка 7 - Настройка тактирования - тактовые импульсы, необходимые для работы любого устройства, подаются от устройства RCC - Reset and Clock Control. Поскольку порт GPIO подключен к шине AHB1, то настраиваем мы регистр AHB1ENR (ENR - Enable Register). Такую операцию необходимо проделывать для включения любого устройства.

- Строчки 9, 10 - настройка регистра MODER - Mode Register. Он разбит на пары битов, каждая пара соответствует какому-то пину. Разные последовательности чисел в этих парах кодируют разные режимы работы.
  1 цифра в дефайне обозначает номер пары <=> номер пина
  2 цифра обозначает номер бита в паре: нулевой бит, первый бит. Если второй цифры нет, то в маске, которая соответствует этому дефайну выделены оба бита. Тем самым мы сначала инверсируем оба бита, а потом выставляем первый бит, что означает режим цифрового выхода
- Далее идет настройка регистра PUPDR - Pull Up Pull Down Register - Регистр, управляющий подтяжкой пина. Потренируйтесь, и посмотрите куда в данном случае подтянут пин. В Reference Manual описание всех регистров GPIO находиться после 281 страницы.
- Далее идет аналогичная подача единицы на пин PA1, это было разобрано выше.

Мигание светодиодом. Регистр BSRR

Такое переключение светодиода несовсем удобное. Для переключения сигнала на пине сделан регистр BSRR - Bit Set Reset Register. Он состоит из верхней (биты 16-31) и нижней (биты 0 - 15) частей. Запись единицы в нижний пин переключает пин в логический 0, запись единицы в верхнюю половину переключает пин в логическую 1. После переключения ячейка обнуляется.
Такие операции называютсяатомарными.

Мигание светодиодом. Регистр BSRR

В библиотеке сами регистры разделены BSRRH (BSRR High) и BSRRL (BSRR Low). Сами дефайны одинаковые GPIO_BSRR_BS_X (Вместо X - номер пина). Таким образом имеем:

```
1  #include "project_config.h"
2
3  int main()
4  {
5    SystemInit();
6
7    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
8
9    GPIOA->MODER &= ~GPIO_MODER_MODER1;
10   GPIOA->MODER |= GPIO_MODER_MODER1_0;
11
12   GPIOA->PUPDR &= ~GPIO_PUPDR_PUPDR0;
13   GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;
14
15   while(true)
16   {
17     GPIOA->BSRRH = GPIO_BSRR_BS_1;
18     for(int i = 0; i < 1000000; i++);
19     GPIOA->BSRRL = GPIO_BSRR_BS_1;
20     for(int i = 0; i < 1000000; i++);
21   }
22 }
```

Такой цикл for выступает в роли примитивной задержки. Обычную задержку мы научимся делать позже

Чтение значения

- Для чтения значения на пине сначала нужно перевести с помощью регистра MODER пин в режим INPUT.
- Далее нужно достать значение на всех пинах порта GPIO через регистр IDR - Input Data Register и вычленить конкретный бит, соответствующий нужному пину.

```
uint32_t PA0_data = GPIOA->IDR & GPIO_IDR_IDR_0;
```

Материалы курса можно найти по ссылке:
https://github.com/haaroner/ezh239