

STM32

Глава 1.
Основы

Ежелев Г. И.

Глава 1. Основы

Ежелев Г. И.

21 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

Структурная схема - Datasheet стр. 19

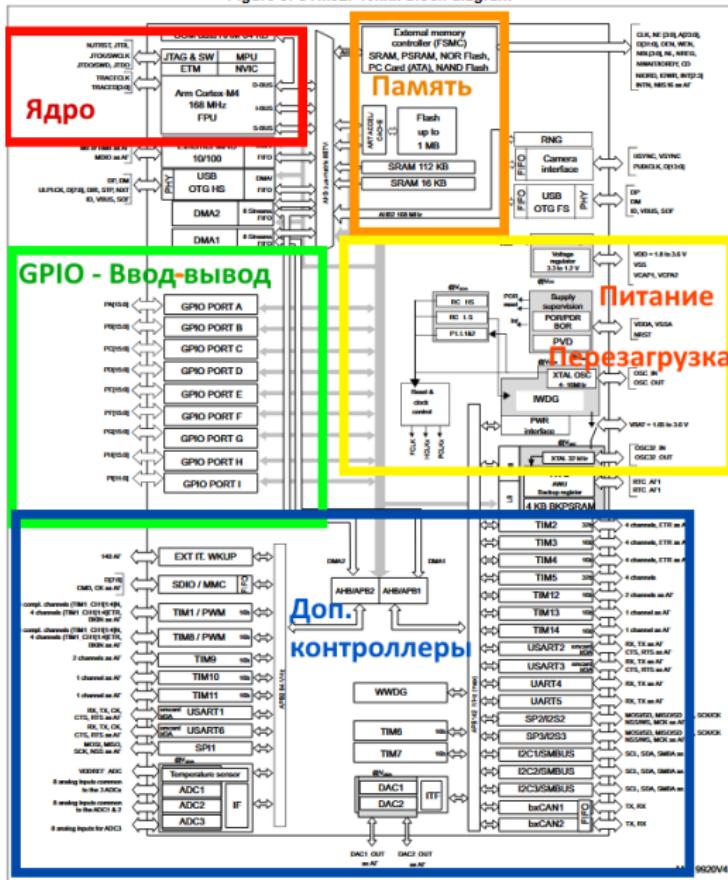
STM32

Глава 1.
Основы

Ежелев Г. И.

2.2 Functional overview

Figure 5. STM32F40xxx block diagram



А где же шины?

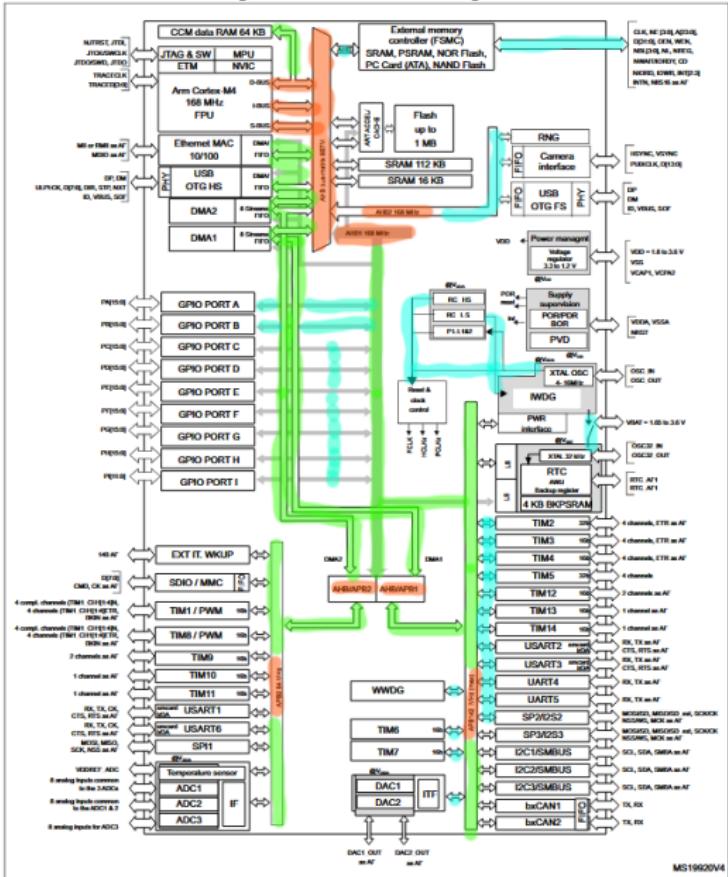
STM32

Глава 1.

Основы

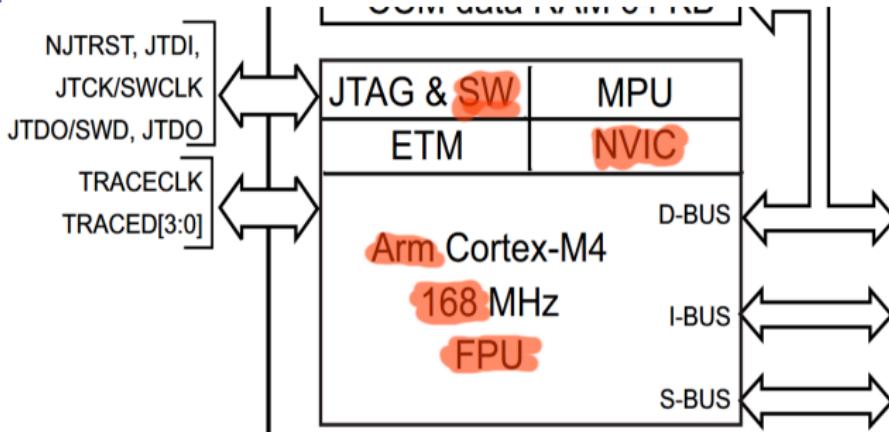
Ежелев Г. И.

Figure 5. STM32F40xxx block diagram



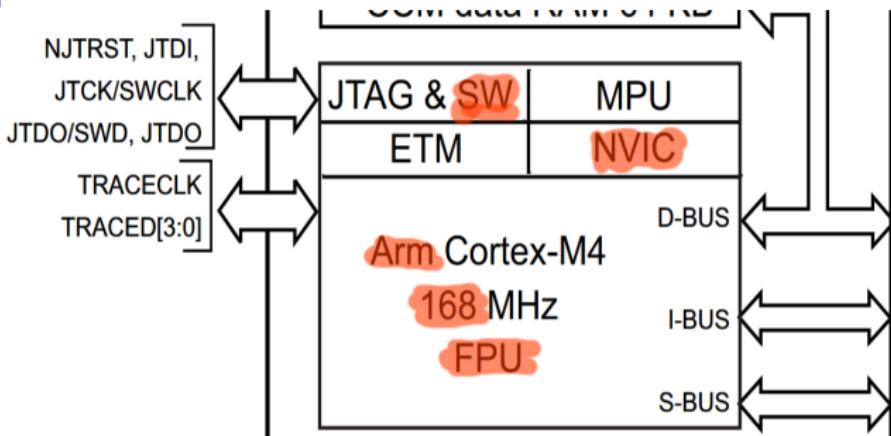
Советую открыть этот даташит (он есть в репозитории!) и самостоятельно взглянуться. На второй картинке названия шин, отмеченные оранжевым, нам пригодятся, запомните их

Ядро

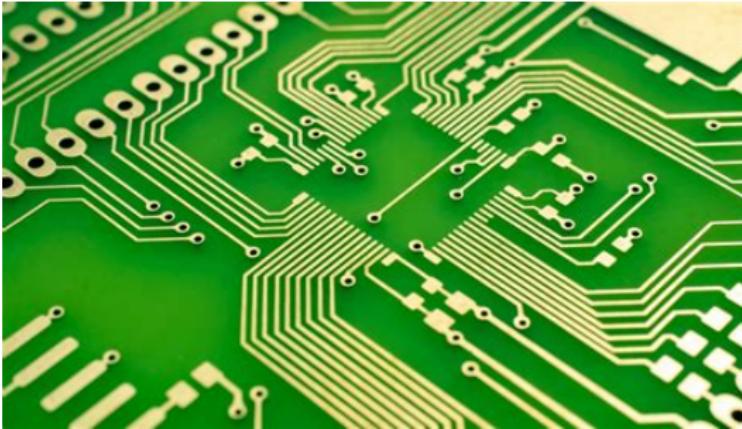


- Архитектура ядра - **ARM** - энергоэффективная. Как в ваших телефонах. В ПК часто ставят более производительную, но прожорливую X86. Архитектура - это то, какие базовые команды реализованы внутри ядра транзисторами. Другими словами, какие команды ядро умеет выполнять сразу. Все что ядро не умеет делать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

Ядро



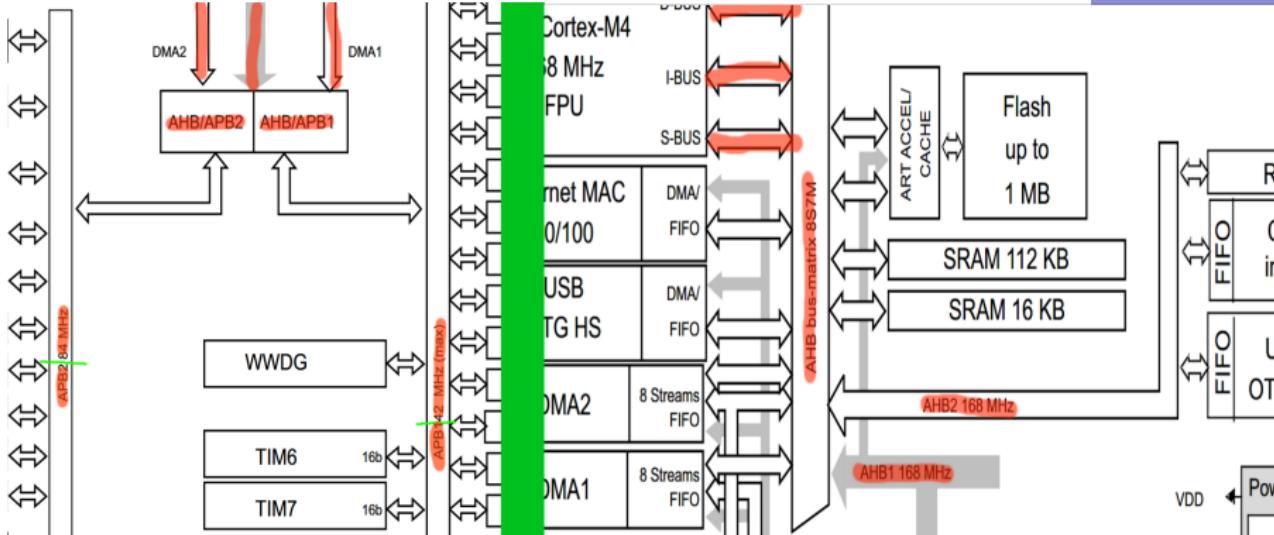
- ▶ Частота ядра - 168 мегагерц. Частота - сколько таких простых операций в секунду оно умеет делать. В нашем случае 168 миллионов.
- ▶ FPU - **Floating Point Unit** - ускоритель вычисления операций с дробными числами. Микроконтроллерам тоже тяжело складывать дроби!
- ▶ В Arduino нет **FPU** и дробные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



- ▶ Шины - 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)

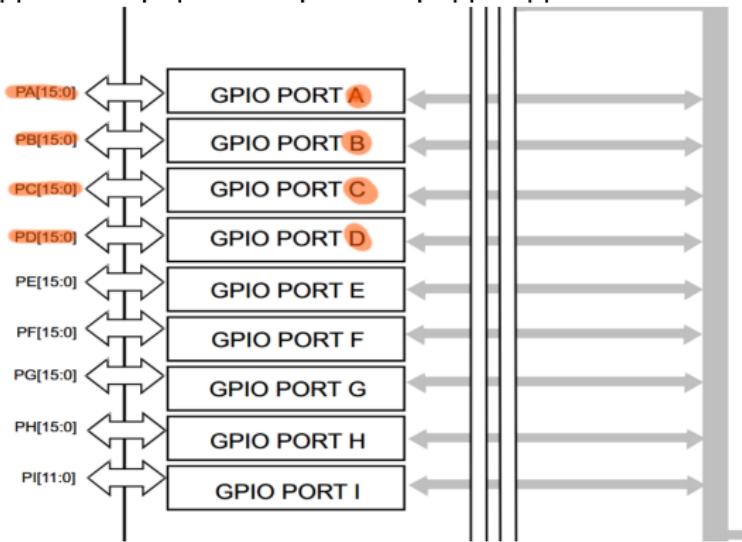
Шины - АНВ, АРВ 1,2

STM32



- ▶ Ядро подключено ко всем устройствам через шины. В программе обращение к устройству происходит через его шину. Скорость шины - скорость устройства.
- ▶ АНВ - высокоскоростная шина. На ней висят самые важные устройства - память, ввод вывод, питание и т.д.
- ▶ АРВ1,2 - Все остальное.

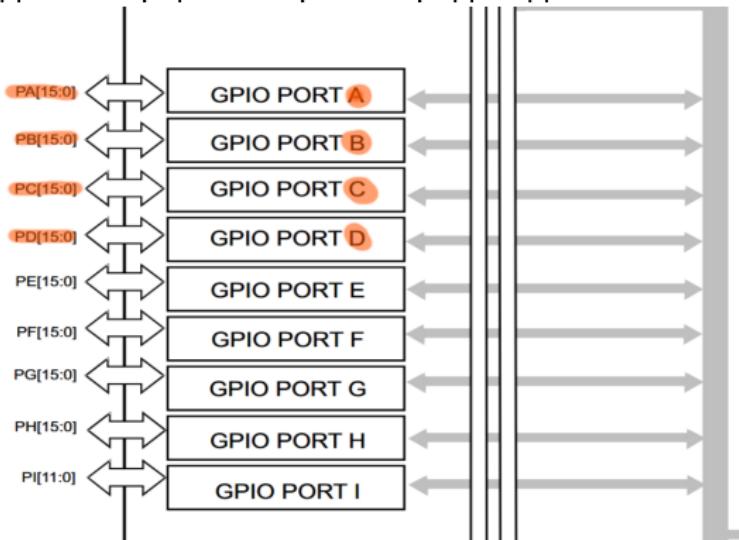
Последняя порция теории перед кодом!



- ▶ **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 А, В, С... . На каждом 16 пинов - от 0 до 15. Например 13 пин контроллера С - это GPIO Port C Pin 13 или сокращенно PC13. В Ардуино 2(3) порта - А 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)

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** \Leftrightarrow на i-м пине будет **высокий** сигнал(3.3В).

Если на i-м месте **будет 0** \Leftrightarrow 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

Diagram illustrating binary subtraction:

$$\begin{array}{r} 00110010 \\ - 11111101 \\ \hline 00110000 \end{array}$$

The result is 00110000. Blue vertical bars highlight the 5th bit from the left (bit 4) in both the minuend and the subtrahend, and the result shows that this bit has been set to 0.

2. Установка бита в единицу

Diagram illustrating binary addition:

$$\begin{array}{r} 00110000 \\ + 00000010 \\ \hline 00110010 \end{array}$$

The result is 00110010. A blue vertical bar highlights the 5th bit from the left (bit 4) in the first operand, and the result shows that this bit has been set to 1.

1. Установка бита в ноль

Diagram illustrating binary subtraction:

$$\begin{array}{r} 00110010 \\ & \& \\ 11111101 \\ \hline 00110000 \end{array}$$

The result is 00110000, where the 5th bit from the left is set to 0.

2. Установка бита в единицу

Diagram illustrating binary addition:

$$\begin{array}{r} 00110000 \\ | \\ 00000010 \\ \hline 00110010 \end{array}$$

The result is 00110010, where the 5th bit from the left is set to 1.

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 000...010000000 - 7-я единица, т.е. адрес бита, отвечающего за 7 пин.

~ - 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.



Это немного, но это честная работа

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