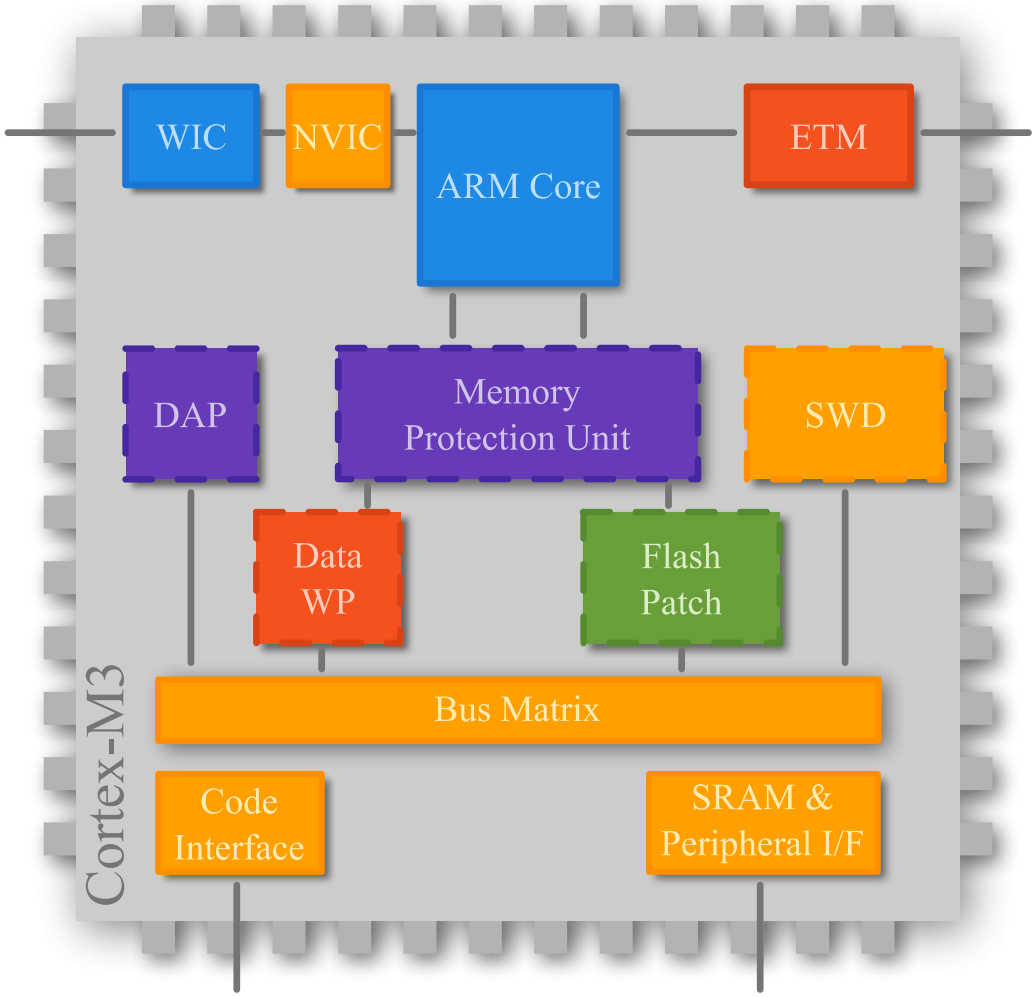
Микроконтроллер под микроскопом

Наверняка вы уже знаете, как устроен компьютер. Даже на бытовом уровне вы представляете, что такое память RAM и ROM. Также знаете, что информация в компьютере представляется в двоичной форме. Однако этих знаний нам недостаточно для того, чтобы начать работать с микроконтроллером.

Помимо уже упомянутых видов памяти и непосредственно самого процессорного ядра в микроконтроллер входит регулятор напряжения, генератор тактового сигнала, системная шина, различные контроллеры, блок системы прерываний, периферийные устройства и т.д.



Однако, первое, с чем нам придется ознакомиться – с особым видом памяти под названием «регистр».

Что такое регистр?

Регистр (англ. register) – это устройство, расположенное внутри ядра микроконтроллера (или процессора), для хранения n-разрядных двоичных данных и выполнения преобразований над ними. Скорость их работы очень высока. Регистр представляет собой упорядоченный набор триггеров, или переключателей (англ. trigger), способных принимать значение «0» (потенциал равен 0 вольт) или «1»

(потенциал равен рабочему напряжению для внутренностей МК, в нашем случае это 1,8 вольта). Число триггеров n соответствует числу разрядов в слове. В свою очередь, машинное слово — платформозависимая величина, измеряемая в битах (или байтах) и равная разрядности регистров процессора и/или разрядности шины данных. Так, например, наш микроконтроллер является 32-разрядным, а это значит, что слово состоит из 32 бит (или из 4 байт). Специальные регистры — это часть RAM-памяти. Их функции определены производителем и не могут быть изменены. Все биты (триггеры) такого регистра подсоединены к определенным цепям внутри микроконтроллера, т. е. изменение состояния триггера напрямую влияет на работу микроконтроллера (отдельных его частей/цепей). Например, записав «0» в ячейку памяти регистра, отвечающего за порт ввода/вывода, вы определяете его функцию: будет он работать на вход (т. е. принимать сигналы) или на выход (т. е. посылать сигналы).

Так как мы будем использовать язык программирования Си, то будет правильно напомнить некоторые операции, которые позволят нам быстро и удобно настраивать необходимые биты в регистре. Первое, что стоит вспомнить – это операции побитового смещения. Допустим, нам нужно получить число 8. Мы уже знаем что 8_{10} равно 1000_2. Таким образом, чтобы получить 8, нам всего-то нужно передвинуть в памяти «1» на три позиции влево. Сделать это можно так:

```
mask = 1 << 3; // mask = 8
```

Бит, который хотим изменить, мы будем называть маской. Маска позволит нам обращаться к нужному биту регистра непосредственно.

Однако нам также потребуются побитовые логические операции «И», «ИЛИ» и «НЕ». Для того, чтобы записать «1» в нужный бит регистра, мы можем побитово воздействовать на регистр маской, применяя операцию «ИЛИ». Вспомним таблицу истинности:

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Как видно, если в бите регистра была «1», то мы не затрем её своей маской и изменим только интересующий нас бит. Сделать это можно так:

```
uint32_t mask = 1 << 3;
// reg = 0000 0000 0000 0000 0000 0000 0000 0100;
reg = reg | mask;
// reg = 0000 0000 0000 0000 0000 0000 0000 1100;
```

или так:

```
reg |= mask;
```

Для того чтобы записать «0» в необходимый бит, нам потребуется операция «И», таблица истинности которой изображена ниже:

A	B	A & ¬B
0	0	0
0	1	0
1	0	0
1	1	1

Как видно, мы сотрем все биты, где не содержится «1». Перед тем как мы применим эту операцию, нужно инвертировать маску (т. е. применить операцию «НЕ»):

```
mask = mask;
//  mask = 0000 0000 0000 0000 0000 0000 0000 1000
// ~mask = 1111 1111 1111 1111 1111 1111 1111 0111
reg = reg & (~mask);
```

Теперь применим операцию «И» к регистру, используя модифицированную маску:

```
// reg = 0000 0000 0000 0000 0000 0000 0000 1100;
reg = reg & (~mask);
```

Короткая запись выглядит так:

```
reg &= ~mask;
```

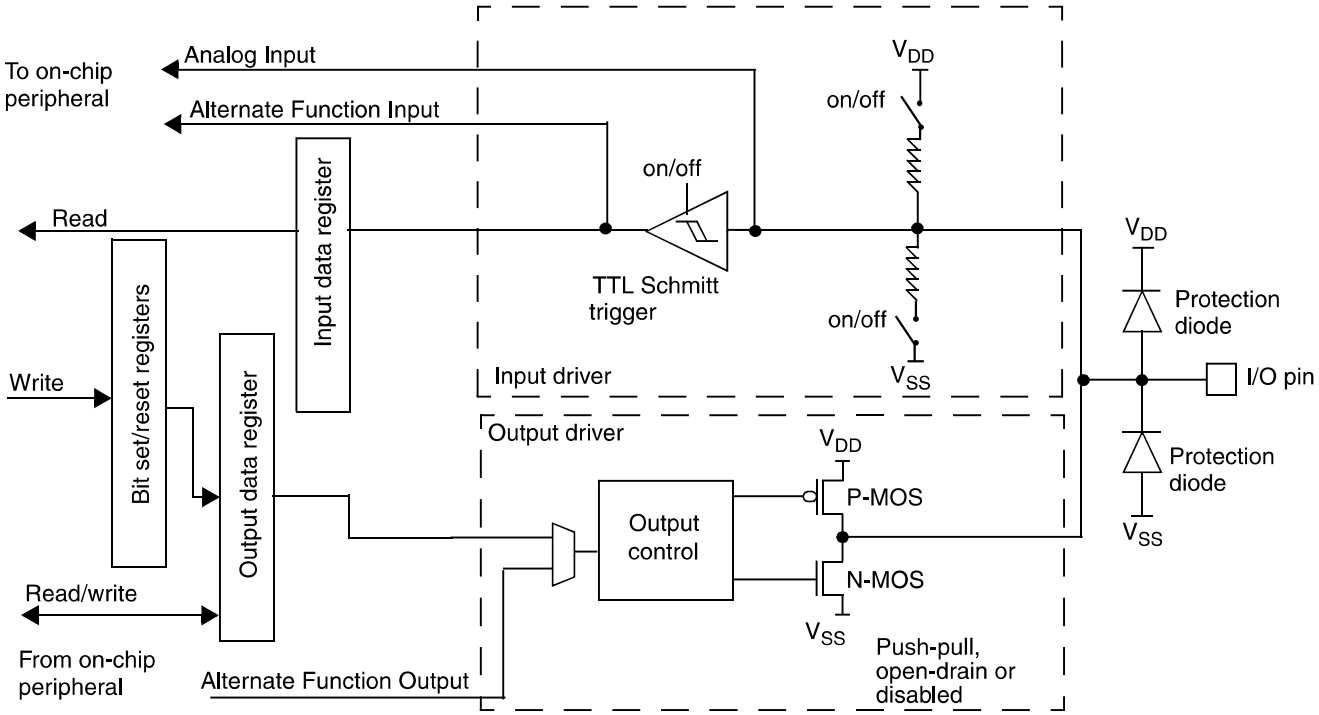
Теперь мы готовы работать с регистрами, но перед этим пару слов о других важных частях микроконтроллера.

Что такое порты ввода-вывода?

Пожалуй, основное, что делает микроконтроллер таким интересным — это порты ввода-вывода (англ. general-purpose input/output). Почему они так называются? Всё очень просто — они либо принимают, либо отправляют сигнал.

Порт – это не просто одна ножка микроконтроллера: в нашем случае на одном порте располагаются до 16 ножек (нумерация идет с 0 и до 15). В свою очередь, порты маркируются буквами — A, B, C и т. д. Например, светодиод может быть подключён к ножке PD8 (порт D, ножка 8).

Режимов работы у ножки может быть несколько, и настраиваются они посредством соответствующих регистров. Как найти эти регистры, как они называются и что с ними нужно сделать, чтобы включить тот или иной режим, мы поговорим когда приступим к программированию, а сейчас рассмотрим эти самые режимы. Ниже приведена схема устройства одной ножки нашего МК.



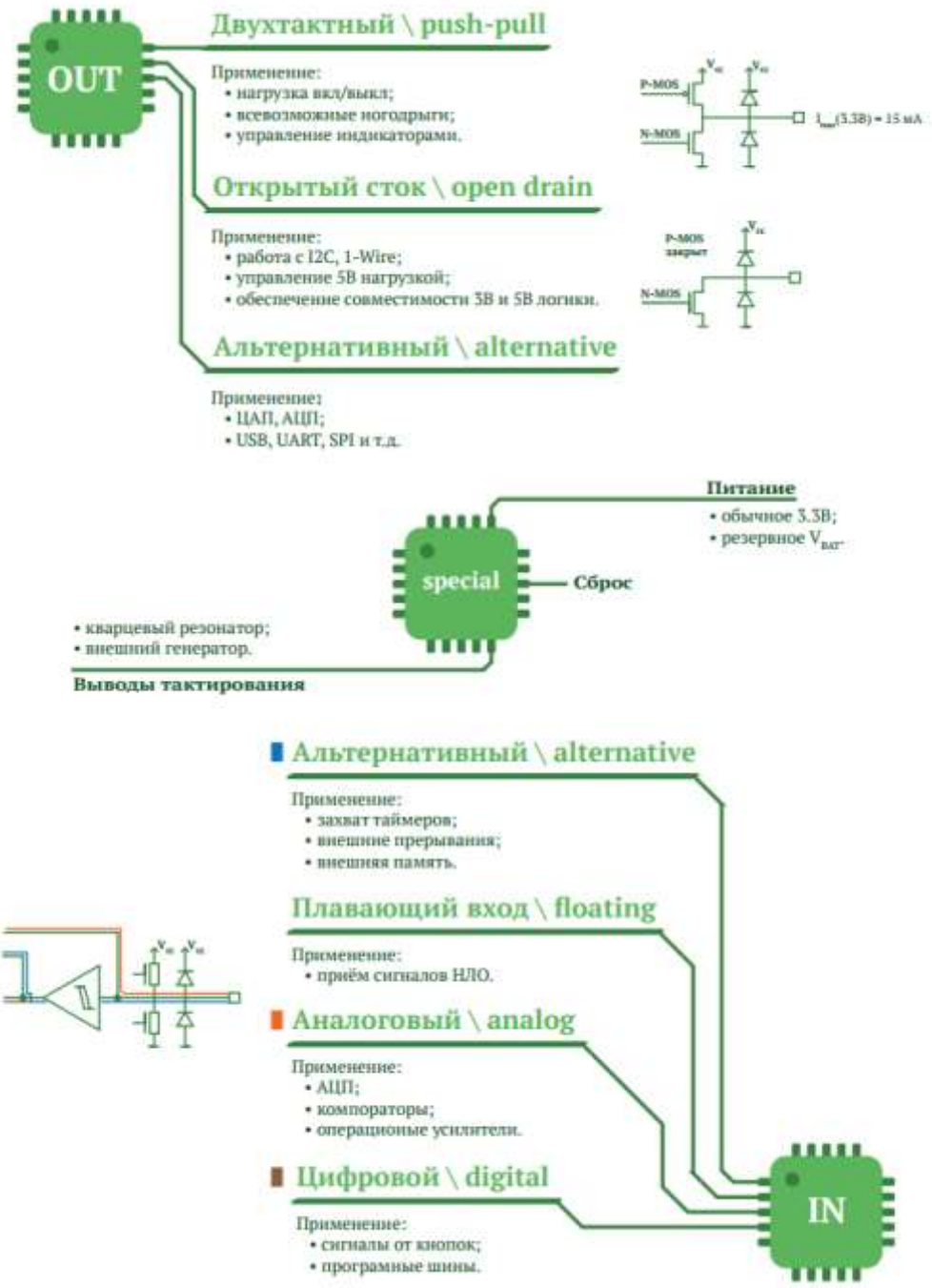
Изображение из Reference Manual, *Figure 13. Basic structure of a standard I/O port bit*

Первый отличительный признак, который можно заметить (да и просто вывести из названия), это то, что ножка может работать как вход и выход. Выход имеет четыре режима работы:

- выход с открытым стоком (англ. output open-drain) — в этом случае при записи «0» в выходной регистр (англ. output register) активируется N-MOS, а при записи «1» порт переходит в высокоимпедансное состояние (также этот режим называют Hi-Z, P-MOS никогда не активируется);
- выход с подтяжкой «тяни-толкай» [1] (англ. output push-pull) – в русской литературе называется «двухтактный выход», а принцип его прост – запись «0» в Output register активирует N-MOS, запись «1» активирует P-MOS;
- альтернативная функция с подтяжкой «тяни-толкай» (англ. alternate function push-pull) – уже описанный ранее двухтактный выход, только для альтернативной функции;
- альтернативная функция с открытым стоком/коллектором (англ. alternate function open-drain).

В свою очередь, у входа имеются разные режимы работы:

- плавающий вход (англ. input floating) – подтяжка отключена (без подтяжки ножка находится в Hi-Z состоянии, а это означает, что сопротивление входа велико, и любая электрическая наводка (помеха) может вызвать появление на таком входе «1» или «0»);
- вход, подтянутый к верху (англ. input pull-up) — подтяжка к питанию (для STM32 обычно это 3,3 вольт);
- вход, подтянутый к низу (англ. Input-pull-down) — подтяжка к земле (0 вольт);
- аналоговый вход (англ. analog) — если ножка настроена как аналоговый вход, то используется модуль АЦП, если он подключен к этой ножке.

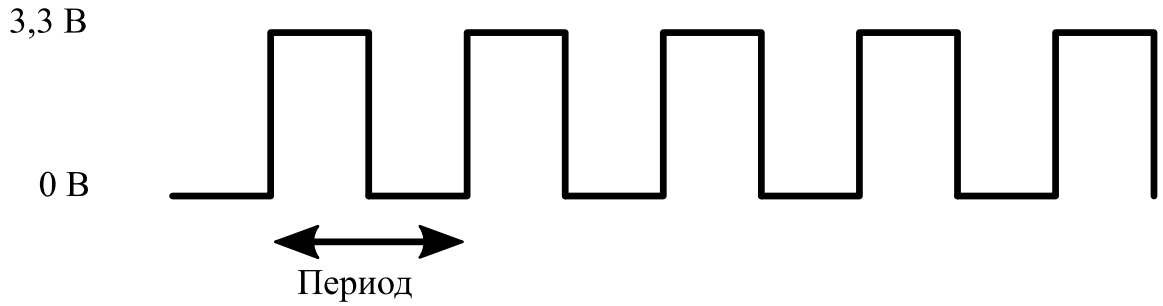


Чтобы воспользоваться GPIO, нужно еще включить тактирование порта... а для этого нужно понимать, что такое тактирование!

Что такое система тактирования?

Что же является сердцем микроконтроллера, что заставляет его работать? Выбирая компьютер в магазине, вы наверняка обращали внимание на такой показатель, как «тактовая частота».

> **Тактовый сигнал** или **синхросигнал** — сигнал, использующийся для согласования операций одной или более цифровых схем. Синхросигнал обычно имеет форму меандра (прим. — прямоугольник) и колеблется между высоким и низким логическими уровнями. // Wikipedia



В первом приближении частота характеризует производительность (память, ядро и т.д.), и коррелирует с количеством операций в секунду. Однако, системы с одинаковой частотой могут иметь различную производительность: выполнение одной и той же операции может занимать разное количество тактов, это зависит от архитектуры и инструкций, реализованных в ядре.

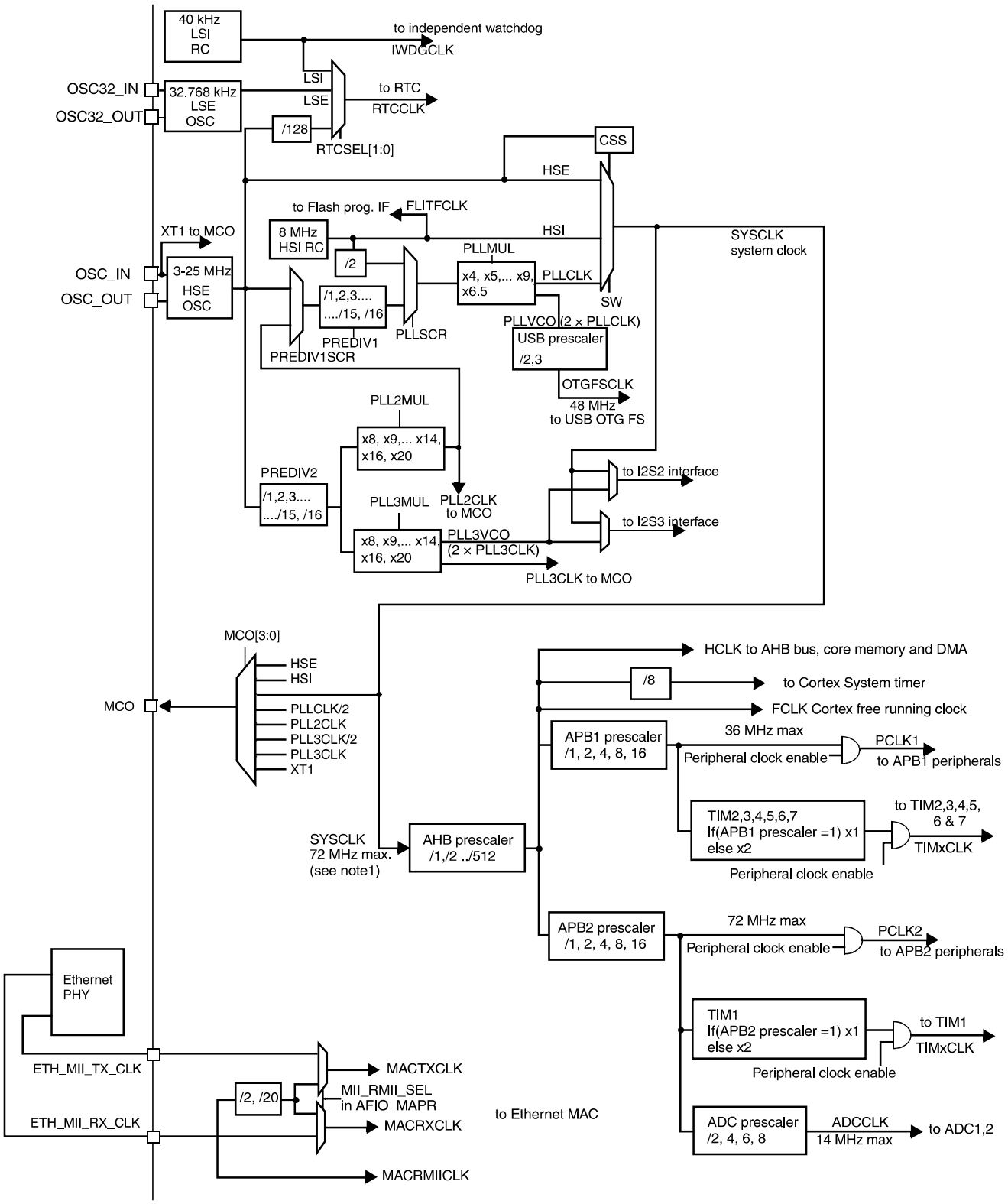
Работа всех подсистем микроконтроллера (блоков периферии) зависит от того, получают они тактовые сигналы или нет. Нет тактового сигнала — периферия не работает. При этом некоторые блоки не могут работать на той же частоте, что и ядро. Все эти тонкости описываются в документации.

По умолчанию тактирование любой периферии в STM32 отключено, и необходимо включать её вручную. Для этого используется модуль RCC (англ. Reset and Clock Control), о котором будет рассказано в другом разделе.

Источником тактового синхросигнала служит генератор тактовых импульсов. Он может быть реализован разными способами, с использованием разных физических эффектов.

- **RC-генератор** — встроенный источник тактового сигнала, основан на RC-цепочке. Минусом такого генератора является небольшая точность (для часов такой генератор не рекомендуется использовать).
- **Кварцевый** или **керамический резонатор** — более точный источник тактовых импульсов, основан на пьезоэлектрическом эффекте.
- Внешний сигнал — можно использовать любой другой источник, хоть **сиси-одоси**, который при опрокидывании будет замыкать проводник на питание и тем самым генерировать сигнал (заполнение должно быть порядка 50%). В некоторых случаях для тактирования часов используют частоту питающей сети (в России это **50 Гц**, в США 60 Гц), но точность будет не большой.

Ниже приведена схема тактирования микроконтроллера **stm32f103c8**, поясняющая, как происходит тактирование.



Изображение из Reference Manual, Figure 8. Clock tree

Большая часть системы тактируется от линии `SYSCLK` (англ. System Clock), за исключением блоков USB, RTC и т.д. Источником сигнала для неё могут выступать три источника HSI, HSE и PLL.

- **HSI** (сокр. High Speed Internal) — встроенный RC-генератор тактового сигнала работающий на частоте 8 МГц. Его калибруют на заводе, но при изменении температуры частота может колебаться от 7,3 МГц до 8,7 МГц. При подаче питания он является источником тактового сигнала, далее программа может переключаться на другие источники.

- **HSE** (сокр. High Speed External) — внешний генератор (обычно кварцевый или керамический резонатор), который подключаются к соответствующим ножкам МК.
- **PLL** (сокр. Phase-Locked Loop) — система **фазовой автоподстройки частоты** (ФАПЧ), позволяет умножать частоту HSI или HSE на множитель от 2 до 16. Частота поступающая на ФАПЧ делится пополам.

При работе внешний резонатор может перестать работать. Причины могут быть разными [2]: сбои в питании, температурный режим, плохая разводка, наводки или его может оторвать в ходе эксплуатации (почему нет?). Для этих случаев предусмотрен блок аварийного переключения CSS (англ. Clock Security System): он способен автоматически переключить систему на внутренний источник (PLL, если он использовался, будет остановлен). Рассматривать мы его не будем.

Максимальная частота **stm2f103c8** 72 МГц, однако она не достижима при работе от внутреннего генератора (4 МГц · 16 = 64 МГц).

> Обратите внимание, что не все периферийные устройства могут работать на частоте `SYSCLK`, это видно на диаграмме. Например максимальная частота шины `APB1` 36 МГц, т.е. елси `SYSCLK` равно 64 МГц, то шине `APB1` нужно выставить предделитель (англ. prescaler) минимум 2 (тогда частота будет равна 32 МГц).

При настройке системы тактирования также нужно брать во внимание, что в STM32 flash-память не всегда может работать на частоте `SYSCLK`, поэтому на предусмотрена система задержки. Например при частотах от 0 до 24 МГц (включительно) чтение происходит сразу, при частотах от 24 до 48 МГц (включительно) пропускается один такт, а при частотах от 48 до 72 (включительно) два такта.

> В STM32 один из выводов, его называют MCO (англ. Master Clock Output) можно использовать для вывода тактового сигнала от шины `SYSCLK` или источников HSI, HSE и PLL (делённое на 2). На плате данный вывод выведен в виде медного пяточка; если у вас есть осциллограф или логический анализатор, то вы можете сконфигурировать данную ножку и посмотреть сигнал на выходе.

Для сторожевого таймера и часов реального времени (англ. Real Time Clock, RTC) предусмотрен низкочастотный внутренний RC-генератор LSI. RTC так же может работать от HSE (частота делится на 128) или собственного внешнего источника LSE.

- **LSI** (сокр. Low Speed Internal) — внутренний RC-генератор на 40 кГц.
- **LSE** (сокр. Low Speed External) — внешний осциллятор для часов реального времени на 32,768 кГц.

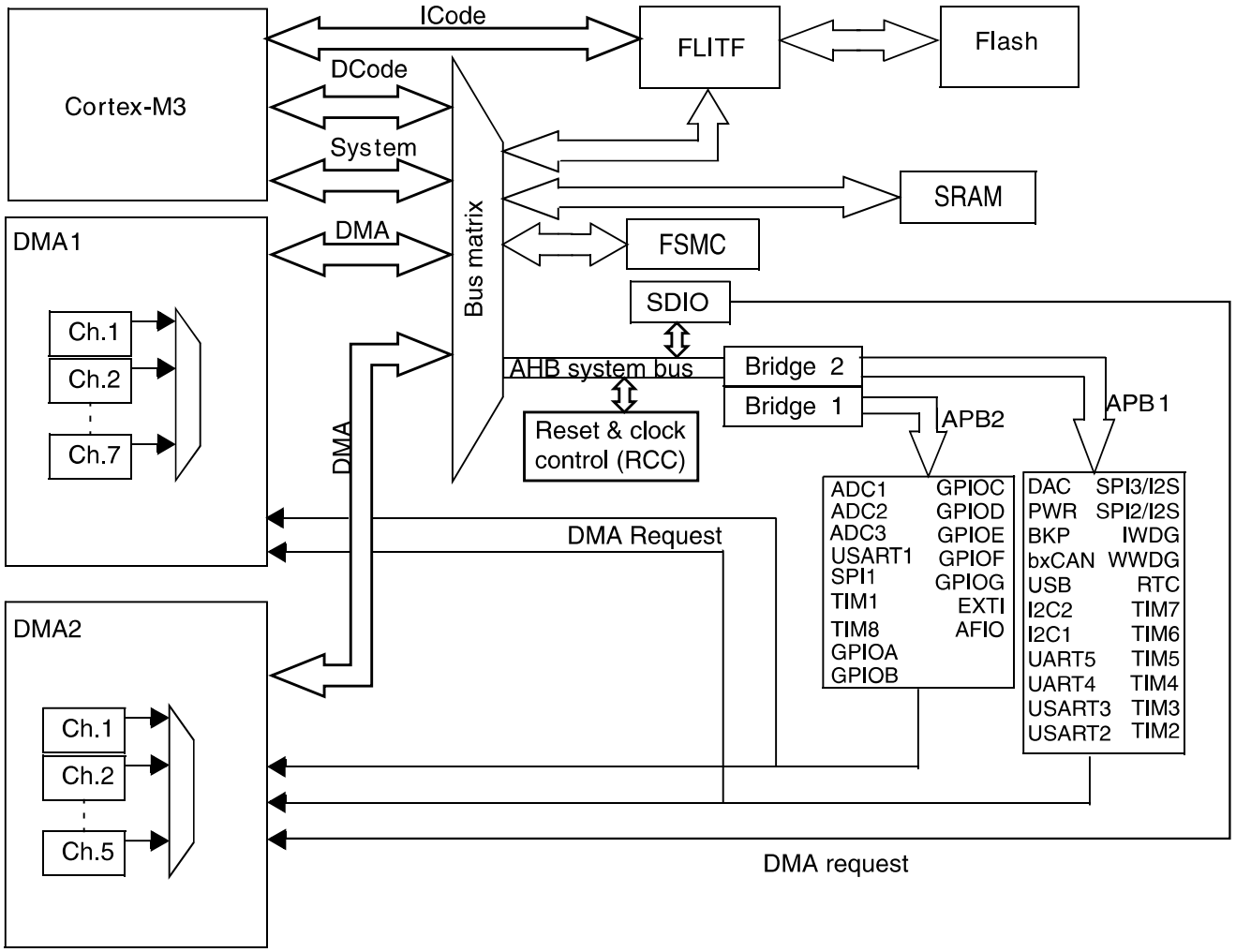
В устройстве `SYSCLK` будет брать сигнал от PLL (HSI), а часы реального времени от LSE.

Архитектура ARM

Перед нами не стоит цели дать исчерпывающее описание ядра, напротив, наша цель — дать поверхностное представление, чтобы можно было начать что-то делать. Подробнее с архитектурой ядра можно ознакомиться в **специальных изданиях** или в **документации**.

В состав Cortex входит не только ядро, но и различные компоненты управления и отладки, которые соединены между собой шинами AHB и APB. Еще два компонента, которые нам потребуются в дальнейшем, это:

- NVIC — встроенный контроллер прерываний, о котором мы еще поговорим. Число возможных прерываний определяется изготовителем микросхемы. Этот контроллер тесно связан с ядром и содержит регистры управления системой.
- SysTick — системный таймер, представляющий собой вычитающий счетчик, который может быть использован для генерации прерываний через равные промежутки времени (даже если МК находится в спящем режиме)

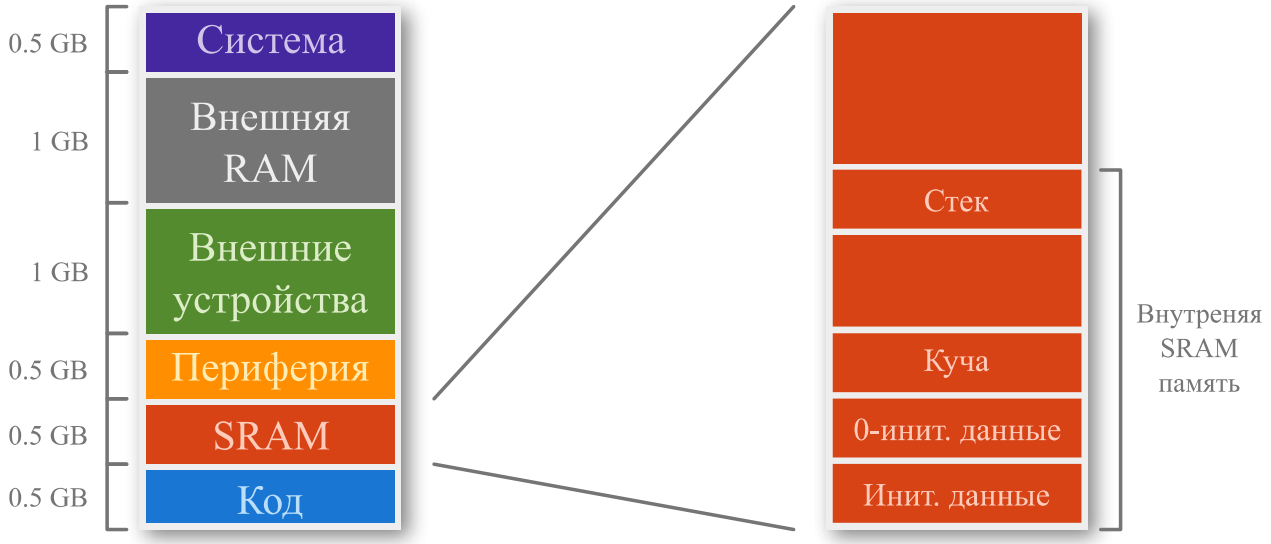


Изображение из Reference Manual, *Figure 1. Low and medium density value line system architecture)*

Устройство памяти

Любое устройство воспринимается микроконтроллером как модуль памяти, хотя физически таковым может и не являться. Память программы, оперативная память, регистры устройства ввода\вывода – все они находятся в едином «адресном пространстве».

Как уже говорилось, Cortex-M3 является стандартизованным ядром, а значит, структура его адресного пространства закреплена стандартом. Оно имеет размер 2^{32} = 4 гигабайта. Первый гигабайт памяти распределен между областью кода и статического ОЗУ. Следующие полгигабайта памяти отведены для встроенных устройств ввода-вывода. Следующие два гигабайта отведены для внешнего статического ОЗУ и внешних устройств ввода-вывода. Последние полгигабайта зарезервированы для системных ресурсов процессора Cortex. Диаграмму карты памяти можно найти в документации.



Вот, пожалуй, и всё, что требуется сейчас знать о микроконтроллере, чтобы начать его программировать. Однако... перед этим немного о стандартных библиотеках.

Назад | Оглавление | Дальше

---

1. В русскоязычной литературе такой режим называют **двухтактным** . ↩

2. Можете изучить документ от компании NXP « **AN3208. Crystal Oscillator Troubleshooting Guide** » в котором рассматриваются распространённые проблемы с кварцевыми резонаторами и методами их решения. ↩