II Погружение в HAL

# 6. Управление портами GPIO

Решение полностью обновить HAL (Hardware Abstraction Level, уровень абстрагирования от аппаратуры) возникло в ST вместе с появлением STCube. До релиза STCube HAL официальной библиотекой для разработки под STM32 была SPL (Standart Peripheral Library, стандартная библиотека для работы с периферией). Несмотря на то, что разработчики до сих пор достаточно широко используют SPL и вы можете найти примеры в сети, в которых используется эта библиотека, STCube HAL – это мощный скачок вперёд по сравнению со старой SPL. Конечно, SPL, будучи первой библиотекой, разработанной ST, не была идеальной: не всё её части были совместимы для разных семейств STM32, а ранние версии содержали ошибки. Это стало причиной появления разных альтернативных библиотек, и официальный софт от ST многие люди до сих пор считают скудным.

Таким образом, ST полностью переработала HAL и, поскольку она всё ещё требует небольшой «подстройки», ST обеспечивает ей официальную поддержку в будущем. Более того, новая версия HAL упрощает портирование кода между разными подсемействами STM32 (F0, F1 и т.п.), снижая затраты усилий, необходимых для адаптации ваших приложений под разные микроконтроллеры (совместимость выводов без хорошего слоя абстракции – это всего лишь маркетинговое преимущество). По этой и некоторым другим причинам эта книга базируется исключительно на STCube HAL.

Текущая глава начинает наше путешествие внутрь HAL с рассмотрения простейшего модуля: HAL\_GPIO. Мы уже использовали многие функции этого модуля в предыдущих примерах этой книги, но сейчас настало время узнать обо всех возможностях, которые предлагает такая простая и часто используемая периферия. Однако, прежде чем мы начнём описание всех особенностей HAL, лучше всего будет кратко пробежаться по тому, как именно периферия STM32 отображается (или, как ещё говорят, маппится) на логические адреса и как она представлена в HAL.

## 6.1 Отображение периферии STM32 и хэндлеры HAL

Любая периферия STM32 соединяется с ядром микроконтроллера (МК) через некоторый набор шин, как показано на рисунке 1[[1]](#footnote-0).

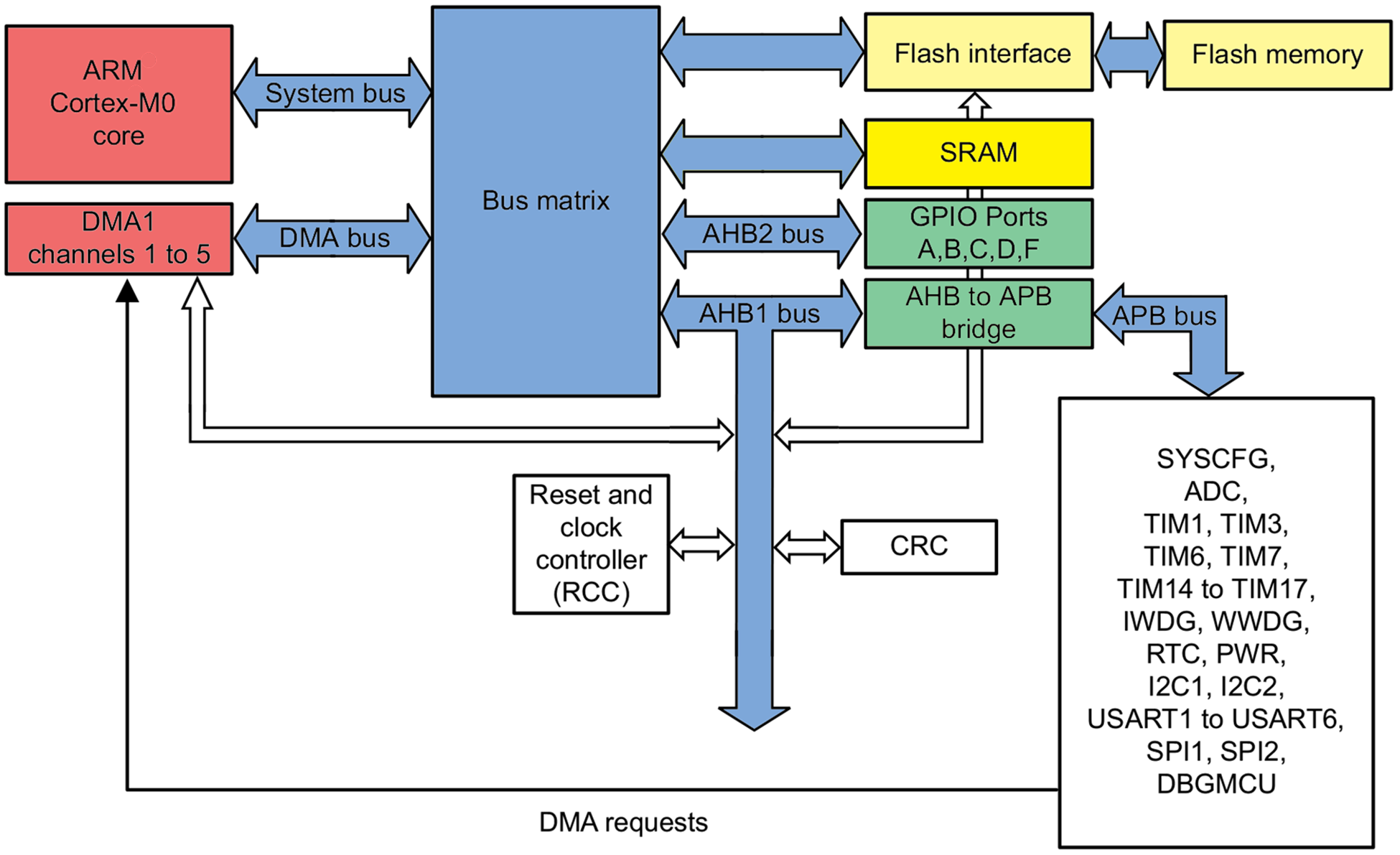


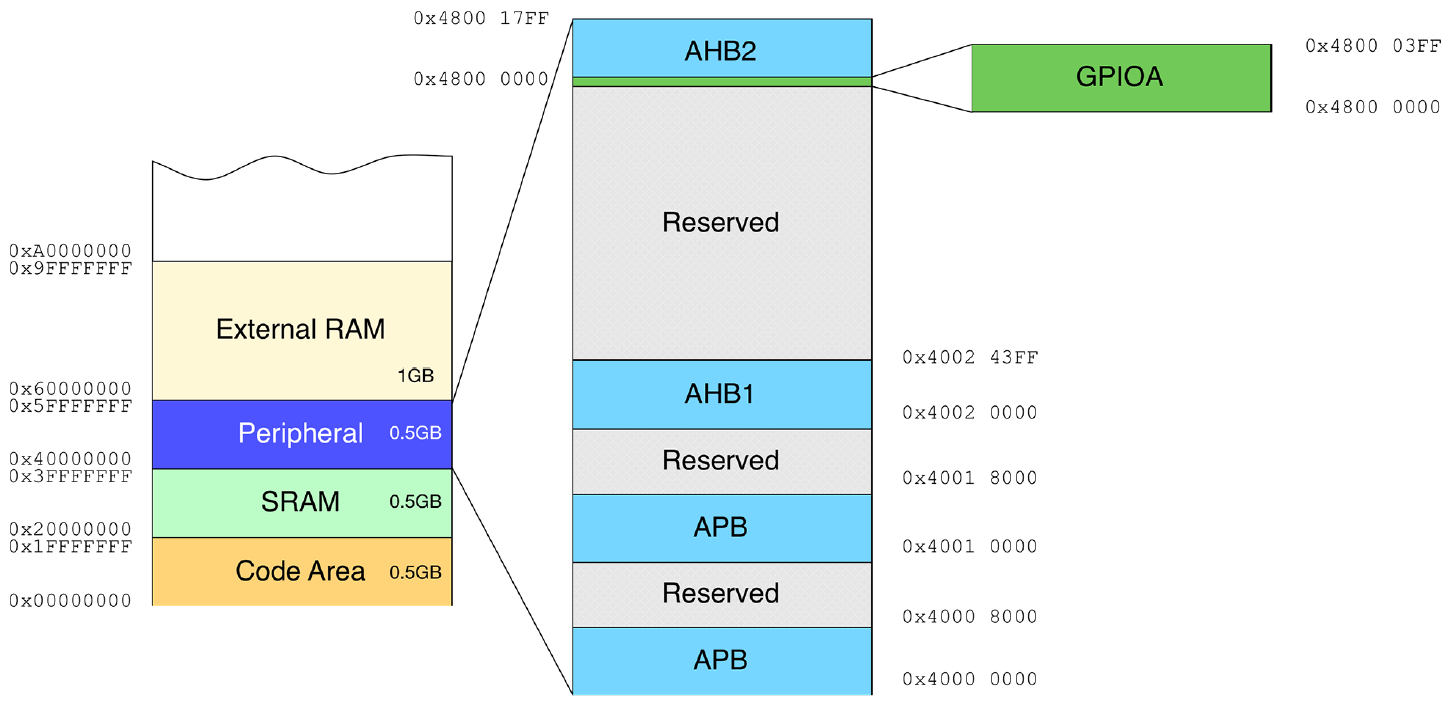
Рисунок 1. Архитектура шины STM32F030

* System bus (системная шина) соединяет ядро Cortex-M c Bus Matrix (матрицей шин), которая выполняет роль арбитра между ядром и DMA (Direct Memory Access, прямой доступ в память). И ядро, и DMA действуют как мастер (в терминологии мастер-слэйв, master-slave).
* DMA bus (шина прямого доступа в память) соединяет мастер интерфейс DMA шины AHB (Advanced High-performance Bus, продвинутой высокопроизводительной шины) с Bus Matrix, которая управляет доступом CPU и DMA к SRAM, флэш-памяти и периферии.
* Bus Matrix управляет последовательностью доступа между системной шиной ядра и мастер-шиной DMA. Арбитраж производится по алгоритму round-robin. Bus Matrix состоит из двух мастеров (CPU, DMA) и четырёх слэйвов:
  + Интерфейс флэш-памяти
  + SRAM
  + AHB1 с мостом AHB-APB (Advanced Peripheral Bus, продвинутая шина периферии)
  + AHB2
* Мост AHB-APB предоставляет полностью синхронные соединения между шиной AHB и шиной APB, куда подключена большая часть периферии.

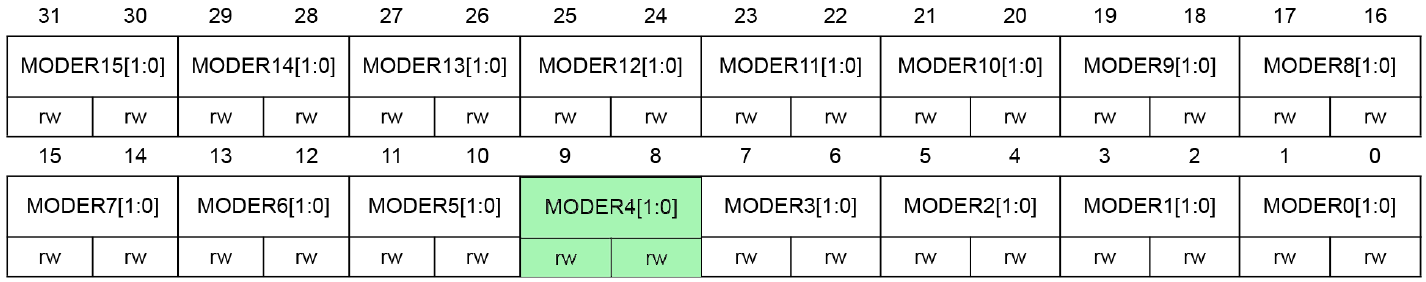
Как мы увидим позже, каждая из этих шин соединена с разным источником тактирования, который определяет максимальную скорость периферии, подключённой к этой шине[[2]](#footnote-1).

В главе 1 мы узнали, что периферия отображается на особый сегмент 4 гигабайтного адресного пространства, начиная с адреса 0x4000 0000 и заканчивая адресом 0x5FFF FFFF. Этот сегмент в дальнейшем разделяется на несколько подсегментов, на каждый из которых отображается свой периферийный модуль, как показано на рисунке 2.

Способ организации этого пространства и, следовательно, способ отображения периферии специфичен для каждого МК STM32. Например, в STM32F030 шина AHB2 отображается на сегмент, расположенный между адресами 0x4800 0000 и 0x4800 17FF, т.е. занимает 6 кБ или 6144 байт. Он разбит на подсегменты, к примеру, периферийный модуль GPIOA (который управляет всеми пинами порта A) занимает пространство между адресами 0x4800 0000 и 0x4800 03FF, т.е. занимает 1 КБ или 1024 байт.

Рисунок 2. Карта памяти сегментов периферийных модулей для STM32F030

Как именно, в свою очередь, организовано это размеченное пространство, зависит от конкретного типа периферии. Таблица 1[[3]](#footnote-2) показывает представление памяти для GPIO.

Bits 2(y+1):2y MODERy[1:0]: Конфигурационные биты порта x (y = 0..15). Эти биты можно записывать программно для конфигурации режима I/O (Input/Output, ввод/вывод).

00: режим входа (по умолчанию)  
01: режим выхода общего назначения  
10: режим альтернативной функциональности  
11: аналоговый режим

Рисунок 3. Разметка памяти для регистра GPIO MODER

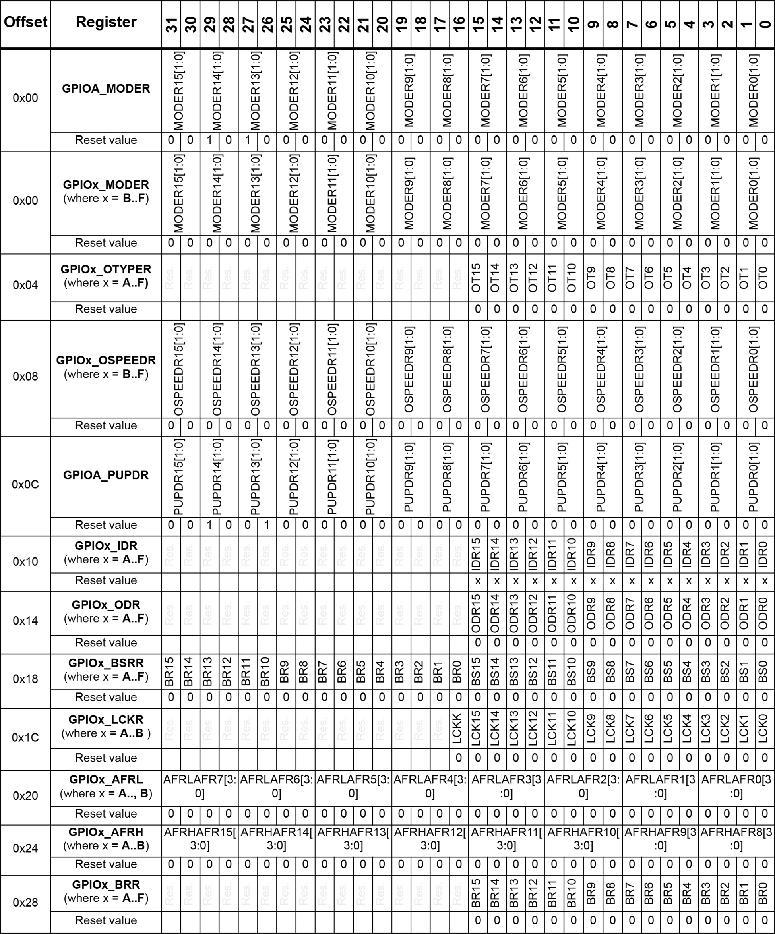


Таблица 1. Карта памяти GPIO для STM32F030

Периферия контролируется изменением и чтением каждого регистра сегмента, на который она отображается. Например, в продолжение разговора о периферийном модуле GPIOA: для переключения пина PA5 в режим выхода мы должны установить регистр MODER так, чтобы биты [11:10] принимали значение 01 (что означает Режим выхода общего назначения), как показано на рисунке 3. Далее, чтобы на выводе появился высокий уровень напряжения, мы должны установить бит 5 в регистре ODR (Output Data Register, регистр выходных данных), который, согласно таблице 1, отображается на адрес GPIOA + 0x14, т.е на 0x4800 0000 + 0x14 = 0x4800 0014.

Следующий небольшой пример покажет, как использовать указатели для доступа к памяти, на которую отображается регистр GPIOA.

int main(void)

{

// Включаем тактирование периферии и подключаем её к AHB1

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

volatile uint32\_t \*GPIOA\_MODER = 0x0, \*GPIOA\_ODR = 0x0;

// Задаём адрес регистра GPIOA->MODER

GPIOA\_MODER = (uint32\_t \*)0x48000000;

// Задаём адрес регистра GPIOA->ODR

GPIOA\_ODR = (uint32\_t \*)(0x48000000 + 0x14);

// Записываем 0x1 в MODER[11:10]

\*GPIOA\_MODER = \*GPIOA\_MODER | 0x400;

// Записываем 0x1 в ODR[5], это устанавливает высокий уровень на PA5

\*GPIOA\_ODR = \*GPIOA\_ODR | 0x20;

while(1);

}

Важно ещё раз прояснить, что каждое семейство STM32 (F0, F1 и т.п.) и каждый конкретный представитель семейства (STM32F030, STM32F103 и т.п.) предоставляет свой набор периферии, которая отображается на определённые адреса. Более того, способ, которым периферия реализована, отличается между сериями STM32.

Одно из назначений HAL -- абстрагироваться от конкретной разметки памяти периферии. Это сделано с помощью определения нескольких *хэндлеров* для каждой периферии. Хэндлер -- это просто структура в языке C, ссылки на которую используются для доступа к реальным адресам периферии. Давайте рассмотрим один из них.

В предыдущих главах мы настраивали вывод PA5 с помощью такого кода:

/\*Настройка вывода GPIO : PA5 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_5;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

Переменная GPIOA -- это указатель на тип GPIO\_TypeDef, определённый так:

typedef struct

{

volatile uint32\_t MODER;

volatile uint32\_t OTYPER;

volatile uint32\_t OSPEEDR;

volatile uint32\_t PUPDR;

volatile uint32\_t IDR;

volatile uint32\_t ODR;

volatile uint32\_t BSRR;

volatile uint32\_t LCKR;

volatile uint32\_t AFR[2];

volatile uint32\_t BRR;

} GPIO\_TypeDef;

Указатель GPIOA определён так, что ссылается[[4]](#footnote-3) на адрес 0x4800 0000:

GPIO\_TypeDef \*GPIOA = 0x48000000;

GPIOA->MODER |= 0x400;

GPIOA->ODR |= 0x20;

## 6.2 Настройка портов GPIO

Разные МК STM32 имеют разное число доступных для программного управления выводов. Точно их число зависит от:

* типа корпуса (LQFP48, BGA176 и т.д.)
* семейства (F0, F1 и т.д.)
* используется ли внешний кристалл для HSE и LSE (см. главу о схеме тактирования)

Выводы GPIO -- это способ связываться с внешним миром для МК. Любая плата использует некоторое число выводов для управления внешней периферией (например, светодиодами) или для передачи данных посредством различных интерфейсов (UART, USB, SPI и др.). Каждый раз, когда нам нужно настраивать периферию, использующую ножки МК, нам нужно настраивать и эти выводы, используя модуль HAL\_GPIO.

Как было сказано выше, HAL разработана так, чтобы можно было абстрагироваться от конкретной разметки памяти. В то же время она предоставляет общий и удобный для пользователя способ настраивать периферию без необходимости программисту вникать в детали конфигурации регистров.

Для настройки GPIO мы используем функцию HAL\_GPIO\_Init(GPIO\_TypeDef \*GPIOx, GPIO\_InitTypeDef \*GPIO\_Init). GPIO\_InitTypeDef -- это C-структура, используемая для настройки GPIO, и она определена так:

typedef struct

{

uint32\_t Pin;

uint32\_t Mode;

uint32\_t Pull;

uint32\_t Speed;

uint32\_t Alternate;

} GPIO\_InitTypeDef;

Назначение каждого из полей структуры:

* Pin -- номер вывода(начиная с 0), который мы собираемся настраивать. Эти номера обёрнуты в макросы типа GPIO\_PIN\_x[[5]](#footnote-4), к примеру, для пина PA5 или PB5 соответствующий макрос будет выглядеть как GPIO\_PIN\_5. Если необходимо настроить одинаково сразу несколько пинов, нужно использовать оператор побитовое ИЛИ: GPIO\_PIN\_0 | GPIO\_PIN\_1 | GPIO\_PIN\_9.
* Mode -- это режим работы вывода, он может принимать одно из значений из таблицы 2. Больше информации об этом поле ниже.
* Pull -- определяет, включать ли подтяжку вверх (Pull-up) или вниз (Pull-down) для выбранных пинов. См. таблицу 3.
* Speed -- определяет скорость пина. Больше информации далее по книге.
* Alternate -- определяет, какая периферия “захватывает” пин. Больше информации далее по книге.

|  |  |
| --- | --- |
| **Режим вывода** | **Описание** |
| GPIO\_MODE\_INPUT | Режим плавающего входа[[6]](#footnote-5) |
| GPIO\_MODE\_OUTPUT\_PP | Режим выхода типа пуш-пулл |
| GPIO\_MODE\_OUTPUT\_OD | Режим выхода типа открытый сток |
| GPIO\_MODE\_AF\_PP | Режим выхода типа пуш-пулл для альтернативной функции |
| GPIO\_MODE\_AF\_OD | Режим выхода типа открытый сток для альтернативной функции |
| GPIO\_MODE\_ANALOG | Аналоговый режим |
| GPIO\_MODE\_IT\_RISING | Режим внешнего прерывания с триггером на фронт сигнала |
| GPIO\_MODE\_IT\_FALLING | Режим внешнего прерывания с триггером на спад сигнала |
| GPIO\_MODE\_IT\_RISING\_FALLING | Режим внешнего прерывания с триггером на фронт и спад сигнала |
| GPIO\_MODE\_EVT\_RISING | Режим события с триггером на фронт сигнала |
| GPIO\_MODE\_EVT\_FALLING | Режим события с триггером на спад сигнала |
| GPIO\_MODE\_EVT\_RISING\_FALLING | Режим события с триггером на фронт и спад сигнала |
| Таблица 2. Макросы, определённые для поля GPIO\_InitTypeDef.Mode | |
| **Режим вывода** | **Описание** |
| GPIO\_PULLUP | Включить подтяжку вверх |
| GPIO\_PULLDOWN | Включить подтяжку вниз |
| GPIO\_NOPULL | Отключить подтяжку |
| Таблица 2. Макросы, определённые для поля GPIO\_InitTypeDef.Pull | |

### 6.2.1 Режимы работы GPIO

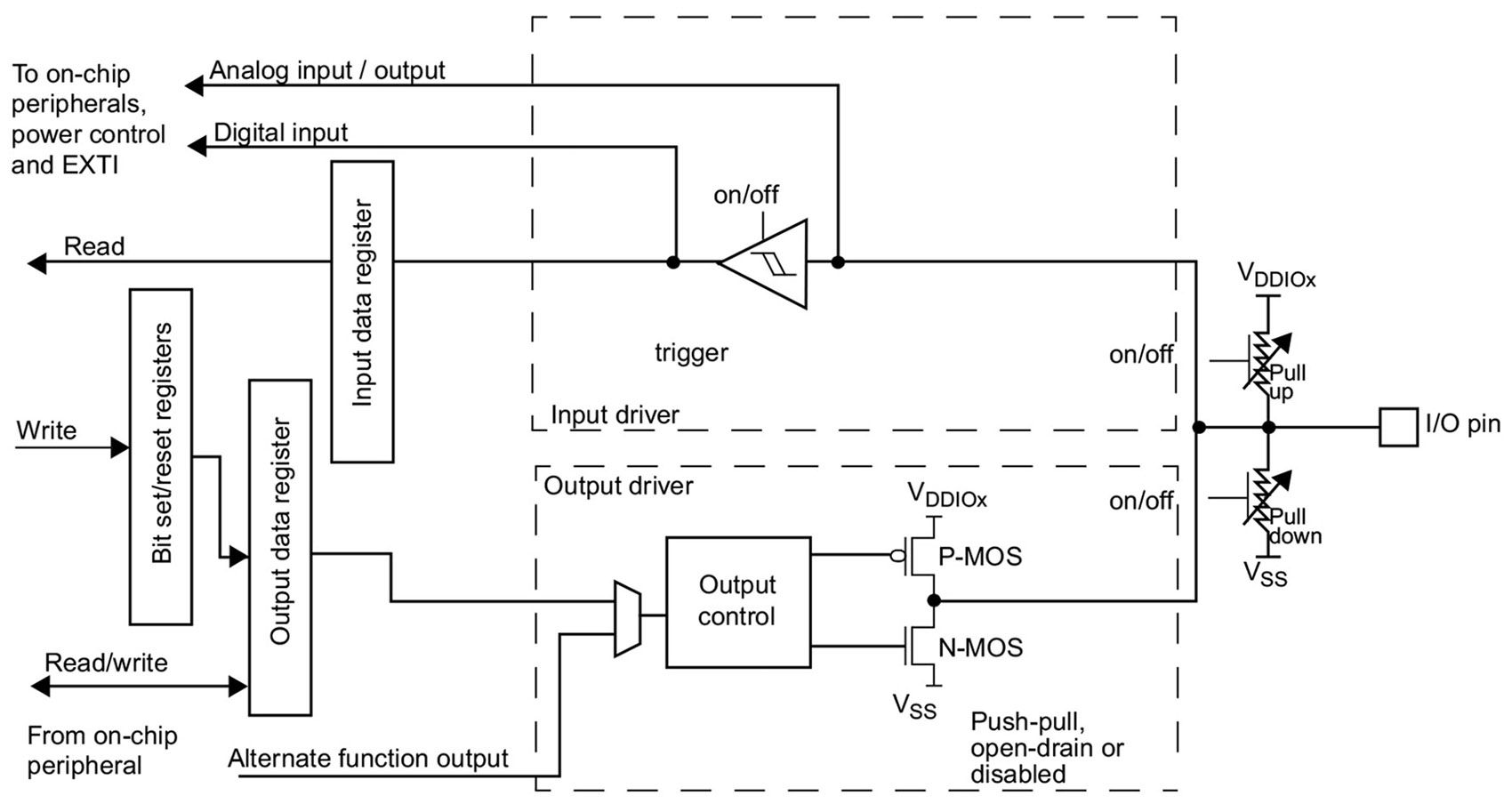
МК STM32 предоставляют по-настоящему гибкое управление GPIO. Рисунок 4[[7]](#footnote-6) показывает аппаратное устройство одиночного вывода контроллера STM32F030.

Рисунок 4. Упрощённая структура вывода.

В зависимости от значения поля GPIO\_InitTypeDef.Mode МК изменяет аппаратную функцию вывода. Давайте взглянем поближе на основные режимы.

Если вывод сконфигурирован в режиме плавающего входа (GPIO\_MODE\_INPUT):

* Выходной буфер отключён.
* Входной триггер Шмитта включён.
* Резисторы подтяжки (pull-up и pull-down) активированы в зависимости от значения поля Pull.
* Данные, поступающие на вывод, защёлкиваются в регистр входных данных на каждом такте AHB.
* Чтение регистра входных данных отображает состояние выводов.

Если порт сконфигурирован в режиме аналогового входа (GPIO\_MODE\_ANALOG):

* Выходной буфер отключён.
* Входной триггер Шмитта отключён и не потребляет входной ток.
* Резисторы подтяжки аппаратно отключены.
* Чтение регистра входных данных возвращает 0.

Если вывод сконфигурирован как выход:

* Выходной буфер находится в одном из следующих режимов:
  + GPIO\_MODE\_OUTPUT\_OD: 0 в регистре ODR активирует N-канальный полевой транзистор, в то время как 1 переводит вывод в состояние HiZ (высокого импеданса). P-канальный транзистор всегда бездействует.
  + GPIO\_MODE\_OUTPUT\_PP: 0 в регистре ODR активирует N-канальный полевой транзистор, 1 активирует P-канальный полевой транзистор.
* Триггер Шмитта на входе активирован.
* Резисторы подтяжки активированы согласно настройке поля Pull.
* Данные, выведенные на пин, каждый такт AHB защёлкиваются в регистр IDR.
* Чтение IDR возвращает текущее состояние вывода
* Чтение ODR возвращает последнее записанное в него значение.

Если вывод сконфигурирован как альтернативная функция:

* Выходной буфер можно настроить в режиме открытого стока или в режиме пуш-пула.
* Выходной буфер управляется сигналами, пришедшими от периферийных модулей.
* Триггер Шмитта на входе активирован.
* Наличие подтяжки зависит от настройки поля Pull.
* Данные, выведенные на пин, каждый такт AHB защёлкиваются в IDR.
* Чтение IDR возвращает текущее состояние вывода.

Режимы GPIO\_MODE\_EVT\_\* относятся к спящим режимам. Когда вывод настроен на работу в одном из этих режимов, CPU просыпается (если находился в спящем режиме после инструкции WFE) как только соответствующий вывод ловит нужное изменение состояния, не генерируя при этом прерывание (больше информации в следующей главе). Режимы GPIO\_MODE\_IT\_\* относятся к управлению прерываниями и будут проанализированы в следующей главе.

Однако, нужно понимать, что детали реализации могут различаться между разными семейства контроллеров STM32, особенно для серий с малым потреблением энергии (low-power). Всегда читайте руководство reference manual на ваш МК, который точно описывает все режимы IO и их влияние на работу МК и потребление энергии.

Также важно отметить, что эта гибкость представляет собой настоящий прорыв и в плане проектирования аппаратной части тоже. Например, теперь не нужно использовать внешние резисторы подтяжки для управления I2C устройствами с тех пор как соответствующие выводы могут быть настроены так: GPIO\_InitTypeDef.Mode = GPIO\_MODE\_OUTPUT\_PP and GPIO\_InitTypeDef.Pull = GPIO\_PULLUP. Это экономит место на плате и упрощает перечень элементов.

Режим I/O может быть настроен через приложение CubeMX, как показано на рисунке 5. Диалог Pin Configuration можно найти в контекстном меню Configuration после клика ПКМ по кнопке, соответствующей GPIO на картинке.

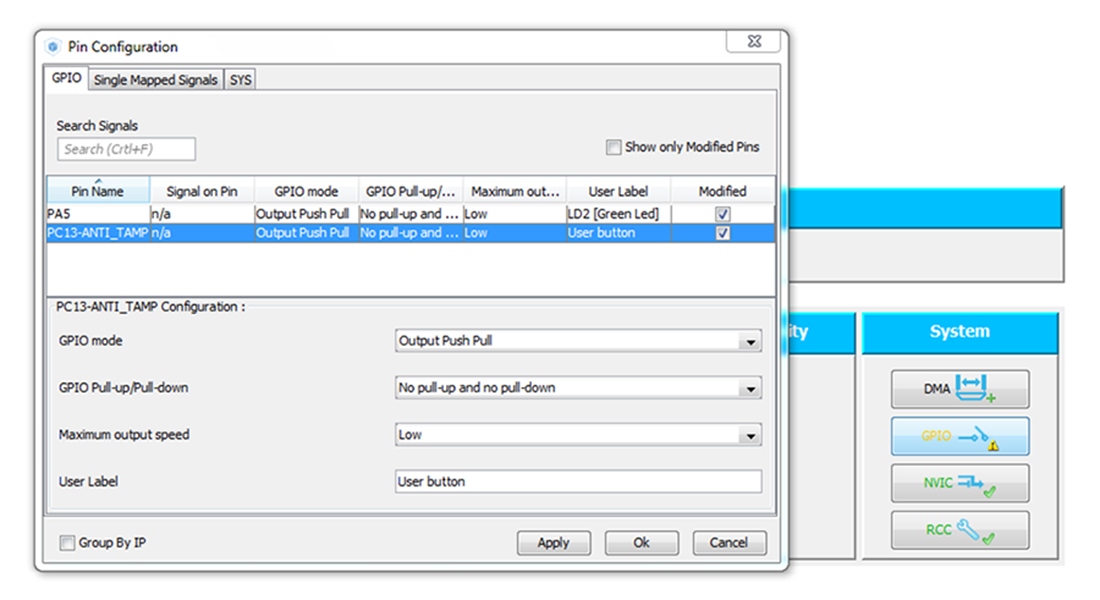


Рисунок 5. Диалог Pin Configuration может быть использован для конфигурирования режима вывода.

### 6.2.2 Режим альтернативной функции

Большая часть выводов имеет так называемые “альтернативные функции”, что означает, что эти выводы могут быть использованы как минимум одним внутренним периферийным модулем. Однако, помните, что любой вывод I/O может быть соединён только с одной периферией одновременно.

Чтобы узнать, какая конкретно периферия может быть привязана к выводу, можно воспользоваться документацией на МК, но проще всего воспользоваться CubeMX. Клик по выводу в окне Pin View вызывает всплывающее меню. В этом меню мы можем назначить желаемую альтернативную функцию. Например, на рисунке 6 вы можете видеть, что вывод PA3 может быть использован как USART2\_RX (что означает, что этот вывод может использован как RX для периферийного модуля USART/UART2, и это справедливо для любого МК STM32 в корпусе LQFP48). CubeMX автоматически сгенерирует для нас код с правильной инициализацией, который показан ниже:

*/\* Configure GPIO pins : PA2 PA3 \*/*

GPIO\_InitStruct.Pin = GPIO\_PIN\_2 | GPIO\_PIN\_3;

GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_LOW;

GPIO\_InitStruct.Alternate = GPIO\_AF1\_USART2;

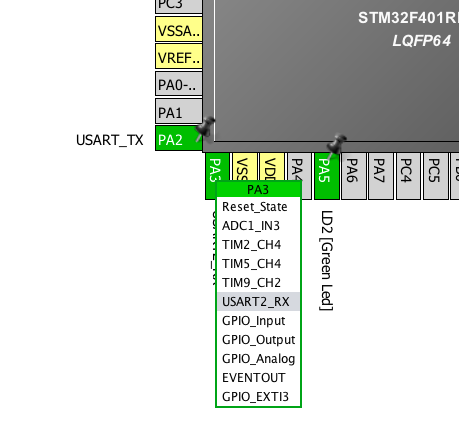
HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

Рисунок 6. CubeMX проще использовать, чтобы узнавать альтернативные функции вывода.

Те, кто работает с МК серии STM32F1, увидят уведомление о том, что поле GPIO\_InitTypeDef.Alternate отсутствует в CubeF1 HAL. Это происходит потому, что что МК серии STM32F1 предоставляют менее гибкий метод для назначения альтернативных функций выводу. В то время как остальные МК STM32 назначают возможные альтернативные функции на уровне GPIO (с помощью настройки специальных регистров GPIOx\_AFRL и GPIOx\_ARFH), что позволяет ассоциировать до 16 различных альтернативных функций с одним выводом (это возможно только для корпусов с большим числом выводов), выводы МК серии STM32F1 имеют сильно ограниченные возможности переназначения, или, как говорят, “ремаппинга”. Например, в МК STM32F103RB у модуля USART3 есть только две пары выводов, на которые можно назначить альтернативную функцию, связанную с этим периферийным модулем. Lва специальных регистра, AFIO\_MAPR и AFIO\_MAPR2, отвечают за “ремаппинг” выводов к соответствующему периферийному модулю. Это и есть причина, по которой поле Alternate недоступно в CubeF1 HAL.

### 6.2.3 Понятие скорости GPIO

Одна из наиболее часто вводящих в заблуждение вещей в МК STM32 -- это параметр GPIO\_InitTypeDef.Speed. Это поле может принимать значения, показанные в таблице 4, и это имеет значение только в том случае, если вывод настроен на выход. К сожалению, ST не подобрала более подходящего имени для этих констант в разных Cube HAL.

|  |  |
| --- | --- |
| CubeF0/1/3/L0/L1 | CubeF4/L4 |
| GPIO\_SPEED\_LOW | GPIO\_SPEED\_FREQ\_LOW |
| GPIO\_SPEED\_MEDIUM | GPIO\_SPEED\_FREQ\_MEDIUM |
| GPIO\_SPEED\_FAST | GPIO\_SPEED\_FREQ\_HIGH |
| GPIO\_SPEED\_HIGH[[8]](#footnote-7) | GPIO\_SPEED\_FREQ\_VERY\_HIGH |

Таблица 4. Доступные режимы скорости для GPIO.

Скорость. Такое манящее слово для любого, кто бредит производительностью. Но что на самом деле оно значит тогда, когда мы говорим о GPIO? Скорость GPIO не имеет отношения к частоте переключений, т.е. сколько раз пин переходит из состояния ON в состояние OFF и обратно за единицу времени. Вместо этого, параметр GPIO\_InitTypeDef.Speed определяет длительность фронта при изменении сигнала GPIO, которая отвечает за то, как быстро сигнал на выводе перейдёт от уровня 0 В до VDD и наоборот.

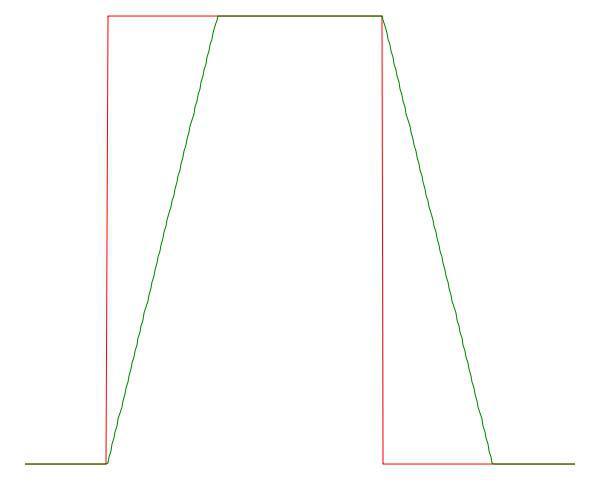


Рисунок 7. Эффект длительности фронта для прямоугольного сигнала. Красным показан идеальный сигнал, зелёным -- реальный.

Рисунок 7 разъясняет суть этого явления. Красный график -- это сигнал, который мы получили бы, будь длительность фронта бесконечно малой, поэтому запаздывание отсутствовало бы. На практике же мы получаем то, что изображено на зелёном графике.

Но насколько большое влияние имеет этот параметр на длительность фронта выводов STM32? Прежде всего, мы должны помнить, что каждое семейство STM32 обладает своими характеристиками управления выводами. Поэтому Вам необходимо сверяться с даташитом на Ваш микроконтроллер в разделе Absolute Maximum Ratings (максимально допустимые/достижимые значения). Затем мы можем провести простой тест для измерения длительности фронтов (тест проводился на отладочной плате Nucleo-F446RE).

int main(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct;

HAL\_Init();

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

/\* Configure GPIO pin : PC4 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_4;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\* Configure GPIO pin : PC8 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_8;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_VERY\_HIGH;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

while(1)

{

GPIOC->ODR = 0x110;

GPIOC->ODR = 0;

}

}

Этот код говорит сам за себя. Мы сконфигурировали два вывода в режим выхода. Один из них, PC4, сконфигурирован с параметром GPIO\_SPEED\_FREQ\_LOW. Второй -- с параметром GPIO\_SPEED\_FREQ\_VERY\_HIGH. На рисунке 8 показана разница в работе этих двух выводов. Как мы можем видеть, фронт сигнала на выводе PC4 соответствует примерно 25 МГц, а на выводе PC8 -- примерно 50 МГц[[9]](#footnote-8).

ВСТАВИТЬ РИСУНОК 8

Однако, всегда помните, что слишком высокая скорость переключения вывода приводит к повышенным электромагнитным помехам, излучаемым вашей платой. В наше время профессиональная разработка -- это стремление к минимизации электромагнитного излучения. Используйте повышенную скорость переключения только тогда, когда это необходимо, во всех остальных случаях используйте минимальную.

Что можно сказать об эффективной частоте переключений? ST в своих даташитах утверждает, что выводы могут переключаться на каждый второй такт шины. Шина AHB1, к которой подключены все GPIO, в МК STM32F446 работает на частоте 42 МГц, т.е. вывод должен переключаться с частотой около 20 МГц. Однако мы внесли дополнительные накладные расходы (оверхед), связанные с переносом в памяти между регистром GPIO->ODR и значением (0x110), которое мы сохраняем в нём, что стоит ЦПУ ещё 1 такт. Поэтому ожидаемая максимальная скорость переключения оказывается около 14 МГц. Осциллограф только подтверждает наши расчёты, как видно на рисунке 9[[10]](#footnote-9).

ВСТАВИТЬ РИСУНОК 9

Странно, но управление выводами через bit-banding-регион, использующее то же самое число ассемблерных инструкций, снижает частоту переключений до драматичных 4 МГц, что показано на рисунке 10.

ВСТАВИТЬ РИСУНОК 10

Код теста выглядит так:

#define BITBAND\_PERI\_BASE 0x40000000

#define ALIAS\_PERI\_BASE 0x42000000

#define BITBAND\_PERI(a,b)((ALIAS\_PERI\_BASE+((uint32\_t)a-BITBAND\_PERI\_BASE)\*32+(b\*4)))

...

volatile uint32\_t \*GPIOC\_ODR = (((((uint32\_t)0x40000000) + 0x00020000) + 0x0800) + 0x14);

volatile uint32\_t \*GPIOC\_PIN8 = (uint32\_t)BITBAND\_PERI(GPIOC\_ODR, 8);

...

while(1)

{

\*GPIOC\_PIN8 = 0x1;

\*GPIOC\_PIN8 = 0;

}

## 6.3 Управление GPIO

CubeHAL предоставляет четыре операции для следующих манипуляций с GPIO: чтение, изменение и заморозка состояние вывода.

Для чтения состояния вывода мы можем использовать функцию:

GPIO\_PinState HAL\_GPIO\_ReadPin(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin)

которая принимает дескриптор порта GPIO и номер вывода. Она возвращает GPIO\_PIN\_RESET, если на выводе низкий уровень сигнала, и GPIO\_PIN\_SET, если высокий.

Чтобы, наоборот, изменить состояние вывода, воспользуемся функцией:

void HAL\_GPIO\_WritePin(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin, GPIO\_PinState PinState)

которая принимает дескриптор порта GPIO, номер вывода и желаемое состояние.

Если же на просто нужно инвертировать состояние вывода, то мы можем использовать эту удобную процедуру:

void HAL\_GPIO\_TogglePin(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin)

И, наконец, одной из особенностей GPIO портов является возможность заморозки состояния вывода. Любая последующая попытка изменить его состояние будет проваливаться до тех пор, пока не произойдёт сброс. Чтобы заморозить состояние вывода, можно использовать процедуру:

HAL\_StatusTypeDef HAL\_GPIO\_LockPin(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin).

## 6.4 Деинициализация GPIO

Также существует возможность привести выводы в исходное состояние (режим плавающего входа, input floating mode). Функция:

void HAL\_GPIO\_DeInit(GPIO\_TypeDef \*GPIOx, uint32\_t GPIO\_Pin)

проделает эту работу за нас автоматически. Эта функция очень удобна, если нам больше не нужна данная периферия или если нам нужно уменьшить потери заряда батареи, когда МК переходит в спящий режим.

1. Здесь, чтобы не нагружать тему лишней информацией, мы покажем организацию шин одного из простейших МК STM32: STM32F030. STM32F4 и STM32F7, к примеру, обладают гораздо более продвинутой системой шинных межсоединений (интерконнектов), обсуждение которой выходит за рамки этой книги. Дополнительную информацию можно всегда найти в reference manual на ваш МК. [↑](#footnote-ref-0)
2. Для некоторых читателей описание выше может быть неясным и слишком сложным для понимания. Не беспокойтесь насчёт этого, просто продолжайте читать главу дальше. Всё станет понятнее, когда вы достигнете главы, посвящённой DMA. [↑](#footnote-ref-1)
3. И таблица 1, и рисунок 1 взяты из STM32F030 reference manual. [↑](#footnote-ref-2)
4. Это не совсем верно, т.к. для экономии RAM, реальная HAL определяет GPIOA как макрос (#define GPIOA ((GPIO\_TypeDef \*) GPIOA\_BASE)). [↑](#footnote-ref-3)
5. Имейте в виду, что GPIO\_PIN\_x это битовая маска, где i-й пин соответствует i-му биту 16битного типа данных (uint16\_t). Например, GPIO\_PIN\_5 имеет значение 0x0020, т.к. 0000 0000 0001 0000 = 32. [↑](#footnote-ref-4)
6. Во время и сразу после сброса, альтернативные функции неактивны и все пины настроены в режим плавающего входа. [↑](#footnote-ref-5)
7. Рисунок взят из STM32F030 reference manual [↑](#footnote-ref-6)
8. Эти режимы доступны только для некоторых высокопроизводительных МК. Сверяйтесь с reference manual на ваш МК. [↑](#footnote-ref-7)
9. К сожалению, щупы моего осциллографа имеют вдвое большую нагрузочную ёмкость, чем необходимо для точных измерений. В соответствии с даташитом на STM32F446RE, максимальная частота переключения его выводов составляет 90 МГц, когда ёмкости нагрузки составляет 30 пФ, напряжение питания -- 2,7 В и активирована компенсационная ячейка. Но у меня нет возможности наблюдать такие результаты из-за слабенького осциллографа и, вероятно, из-за длины дорожек, соединяющих гребёнку Nucleo и выводы МК [↑](#footnote-ref-8)
10. Тест проводился при максимальном уровне оптимизации GCC(-O3), включёнными предвыборкой инструкций и всеми внутренними кэшами. Это обуславливает тот факт, что измеренная скорость немного больше 14 МГц. [↑](#footnote-ref-9)