

# ОПИСАНИЕ БИБЛИОТЕКИ STM32F2\_API

Автор: Дерябкин Вадим (Vadimatorik)

2017

# Глава 1

## ВВЕДЕНИЕ

В данном документе приводится исчерпывающее описание:

- философии библиотеки (логики построения и использования)
- соглашения о написании библиотеки (допустимые синтаксические приемы языка и общие правила написания кода)
- примеров использования библиотеки в реальных задачах

# Оглавление

<b>1</b>	<b>ВВЕДЕНИЕ</b>	<b>1</b>
<b>2</b>	<b>ФИЛОСОФИЯ БИБЛИОТЕКИ</b>	<b>3</b>
2.1	Общие сведения . . . . .	3
2.2	Краткий обзор реализации . . . . .	3
<b>3</b>	<b>СОГЛАШЕНИЕ О НАПИСАНИИ БИБЛИОТЕКИ</b>	<b>5</b>
3.1	Общие положения . . . . .	5
3.2	Средства сборки . . . . .	5
3.3	Дерево проекта и именование файлов . . . . .	5
3.4	Принятые сокращения . . . . .	7
3.5	Общие правила оформления имен . . . . .	8
3.6	Оформление .h файлов библиотеки . . . . .	8

## Глава 2

# ФИЛОСОФИЯ БИБЛИОТЕКИ

### 2.1 Общие сведения

В основу библиотеки легли следующие постулаты:

1. Все, что можно вычислить на этапе компиляции - не должно вычисляться в реальном времени.
2. Между производительностью и расходом памяти выбор должен быть в сторону производительности.
3. Все, что может быть выполнено с помощью аппаратной периферии - не должно выполняться программно.
4. Библиотека должна иметь как можно больше средств гибкой настройки на этапе компиляции и по минимуму - в реальном времени (в угоду производительности).
5. Работа программы должна быть по максимуму предсказуема еще на этапе компиляции. Отсюда следует, что все режимы работы периферии должны быть заданы статически.

### 2.2 Краткий обзор реализации

1. Библиотека написана на C++14.
2. Большую часть библиотеки составляют constexpr функции, которые обрабатывают заполненные пользователем структуры инициализации периферии на этапе компиляции и создают маски регистров для всевозможных, указанных в структуре инициализации, режимов. В реальном времени созданные из const constexpr структур инициализации глобальные объекты в коде пользователя оперируют созданными на этапе компиляции масками регистров для настройки и работы с периферийными блоками.

Этим достигается высокая производительность. Поскольку программе не нужно «собирать» маски регистров в реальном времени, как это сделано в HAL или SPL. Достаточно только применить маску.

3. Тот факт, что для инициализации глобальных объектов используются глобальные const constexpr структуры вовсе не означает, что данные структуры войдут в состав прошивки контроллера.

Яркий тому пример, объект класса `global_port` (который будет рассмотрен в разделе 2.2). Он принимает в себя массив `const constexpr pin_config_t` структур, после чего `private constexpr` методы объекта класса `global_port` их (структуры) анализируют и возвращают `private global_port_msk_reg_struct` структуру, которая будет `private` структурой глобального объекта класса `global_port`.

Структуры `pin_config_t`, использовавшиеся для инициализации `private global_port_msk_reg_struct`, во flash загружены не будут, потому что в ходе работы программы обращений к ним не будет.

4. Для работы с аппаратной частью контроллера используются объявленные в коде пользователя глобальные объекты. Все они должны быть объявлены как `const constexpr`. В качестве параметра(-ов) конструктора передается(-ются) указатель(-и) на `const constexpr` глобальную(-ые) структуру(-ы). Важно отметить следующее:

- В случае, если после анализа структур(-ы) инициализации они(-на) больше не требуется - линкер не включит эти(-у) структуры(-у) в состав выходного файла программы (о чем было сказано в пункте 3). Однако в случае, если используемая структура инициализации, возможно, будет использована во время выполнения программы, как, например, в классе `pin`, описанного в разделе 2.2, то она обязательно пойдет в состав выходной программы.

- Так как конструкторы классов используемых в коде пользователя объектов объявлены внутри класса как `constexpr`, то создание этих объектов, по сути, заключается в простом копировании в оперативную память их изменяемых данных. Никаких действий в реальном времени (за исключением копирования в оперативную память изменяемых в процессе работы данных объекта) не производится.

Объекты, классы которых имеют не `constexpr` конструктор (требующий вызова функции инициализации (конструктора) объекта перед вызовом `main` в реальном времени), **не поддерживаются намеренно**. Подробнее об этом рассказано в разделе, посвященному описанию логики работы `startup` файла (раздел 2.2).

- Из того, что все объекты объявлены как `const constexpr` следует, что у каждого глобального объекта, работающего с периферией в реальном времени, имеется метод начальной инициализации (и/или пере инициализации), вызов которого необходимо произвести из кода пользователя.

Это очень оправданно, когда требуется инициализировать объекты в определенном порядке в ходе выполнения программы, чего сложно достигнуть, когда объекты вызываются автоматически перед вызовом функции `main`. Именно это является причиной отказа от поддержки не `constexpr` конструкторов классов (вызов функций инициализации (конструкторов) которых, без применения дополнительных директив, производится в случайном порядке (нельзя гарантировать, инициализация какого объекта будет произведена раньше)).

- В случае, если `const constexpr` объект был объявлен глобально в коде пользователя, но обращений к нему не было на протяжении всей программы, он не будет добавлен в итоговый файл программы. Ситуация здесь аналогична ситуации с глобальными `const constexpr` структурами.

## Глава 3

# СОГЛАШЕНИЕ О НАПИСАНИИ БИБЛИОТЕКИ

### 3.1 Общие положения

В данной главе будет изложен некоторого рода стандарт, которого следует придерживаться на протяжении всего времени написания кода библиотеки (в идеале, и пользовательского кода тоже).

Стандарт распространяется на:

1. Средства сборки (раздел [3.2](#)).
2. Дерево проекта и именование файлов (раздел [3.3](#)).
3. Принятые сокращения (подраздел [3.4](#)).
4. Общие правила оформления имен (раздел [3.5](#)).
5. Оформление .h файлов библиотеки (раздел [3.6](#)).

### 3.2 Средства сборки

В основе библиотеки лежат constexpr функции, полноценная поддержка которых появилась в C++14. Отсюда следует вывод, что минимально возможная версия используемого языка - C++14. В случае, если в более поздних версиях будет несовместимость с C++14, следует внести изменения в библиотеку, решающие вопросы несовместимости по средствам проверки версии используемого стандарта языка и выбора совместимого с ним участка кода.

Для компиляции библиотеки следует использовать arm-none-eabi-g++ не старше (GNU Tools for ARM Embedded Processors 6-2017-q1-update) 6.3.1 20170215 (release) [ARM/embedded-6-branch revision 245512].

### 3.3 Дерево проекта и именование файлов

Правила, касающиеся оформления библиотеки:

1. Для файлов, относящихся к работе с блоками периферии, должна существовать своя папка на каждый модуль.

Пример: rcc, port, pwm и т.д.

Имя папки должно содержать только название аппаратного модуля, написанного с маленькой буквы.

2. Каждая папка, посвященная определенному блоку аппаратной периферии, должна содержать следующие файлы:

- **prefix\_moduleName.h**

В данном файле должны находиться классы, относящиеся к определенному блоку периферии. Объекты этих классов можно использовать в коде пользователя.

- **prefix\_moduleName.cpp**

Если в prefix\_moduleName.h всего один класс, то в данном файле находятся методы класса из файла prefix\_moduleName.h, вызов которых производится в реальном времени.

В случае, если классов несколько и у них нет static общих методов (используемые двумя и более классами) - данный файл создавать не следует. Вместо этого для уникальных методов каждого класса должен быть свой файл с соответствующим постфиксом (именем класса). Об этом ниже.

- **prefix\_moduleName\_class\_className.cpp**

В случае, если в файле prefix\_moduleName.h более одного класса и какой-то из этих классов имеет методы, доступные только ему - их следует вынести в отдельный файл с постфиксом, соответствующим имени класса, к которому он (метод) относится.

В случае, если в файле prefix\_moduleName.h один класс, методы, относящиеся к этому классу, должны быть размещены в файле prefix\_moduleName.cpp.

- **prefix\_moduleName\_constexpr\_func.h**

В данном файле содержатся все constexpr методы, которые используются классом(-и) из файла prefix\_moduleName.h. Эти методы, как правило, являются private методами класса(-ов).

В случае, если в файле prefix\_moduleName.h более одного класса, в данном файле должны находиться лишь те методы, которые используются всеми классами файла prefix\_moduleName.h.

В случае, если каждый класс файла prefix\_moduleName.h использует лишь свой определенный набор методов, никак не пересекающийся с остальными классами, данный файл создавать не следует.

- **prefix\_moduleName\_constexpr\_func\_class\_className.h**

В случае, если в файле prefix\_moduleName.h более одного класса и у какого-то из классов имеются constexpr методы, никак не связанные с остальными (используются только им), их следует вынести в отдельный файл.

В случае, если таких классов несколько (каждый из которых использует свои определенные constexpr методы), то для каждого такого класса следует создать отдельный файл.

- **prefix\_moduleName\_struct.h**

В данном файле содержатся все структуры и enum class-ы, используемые всеми классами файла prefix\_moduleName.h.

В случае, если классы не имеют общих структур или enum class-ов, данный файл создавать не следует.

В случае, если в `prefix_moduleName.h` всего один класс, его структуры и `enum class`-ы должны располагаться здесь без создания конкретного файла под конкретный класс (из пункта ниже).

- **`prefix_moduleName_struct_class_className.h`**

В случае, если классов в файле `prefix_moduleName.h` более одного и у какого-то из классов имеются структуры или `enum class`-ы, которые используются только им одним, данные структуры и/или `enum class`-ы требуется вынести в отдельный файл с постфиксом имени класса, к которому они относятся.

Имена всех файлов должны быть написаны строчными латинскими символами (маленькие английские буквы). В том числе и сокращения по типу «`rwg`».

Все слова в имени должны разделяться символами нижнего подчеркивания.

В качестве примера рассмотрим дерево папки `port` библиотеки `stm32_f20x_f21x` (название библиотеки выступает в качестве префикса).

`stm32_f20x_f21x_port.h` содержит 2 класса (`global_port` и `pin`). У них есть общие структуры, `enum class`-ы и методы. Однако есть и личные (используемые только ими) структуры, `enum class`-ы и `constexpr` методы. При этом у них нет общих `static` методов.

- `stm32_f20x_f21x_port_class_global_port.cpp`
- `stm32_f20x_f21x_port_class_pin.cpp`
- `stm32_f20x_f21x_port_constexpr_func_class_global_port.h`
- `stm32_f20x_f21x_port_constexpr_func_class_pin.h`
- `stm32_f20x_f21x_port_constexpr_func.h`
- `stm32_f20x_f21x_port_struct_class_global_port.h`
- `stm32_f20x_f21x_port_struct_class_pin.h`
- `stm32_f20x_f21x_port_struct.h`
- `stm32_f20x_f21x_port.h`

## 3.4 Принятые сокращения

1. Если `uint32_t` переменная содержит внутри себя адрес в памяти (является указателем), то перед ее именем должен быть префикс «`p_`».

**Пример:** «`p_target_port`».

2. «`bit_banding_`» == «`bb_`»

Только в тексте (не применимо к коду).

3. «`point_bit_banding_bit_address`» == «`bb_p_`»

Когда `uint32_t` переменная содержит адрес бита в `bit banding` области (является указателем).



## 3.5 Общие правила оформления имен

1. Все имена переменных, структур, объектов, функций должны быть написаны строчными латинскими символами (маленькие английские буквы).

**Пример:** «pwr», «port», «value».

2. Директивы препроцессора (define, макросы, ifndef и т.д.) должны писаться заглавными латинскими символами (большие английские буквы).

**Пример:** «ADD(A,B)»

3. Слова в именах должны быть разделены нижним подчеркиванием.

**Пример:** «buf\_speed», «STM32F2\_API\_PORT\_STM32\_F20X\_F21X\_PORT\_STRUCT\_CLASS\_PIN\_H\_», «PORT\_PIN\_0»

4. Макросы должны начинаться с префикса «M\_», после чего идет действие, которое он совершает («GET»/«SET»).

В именах так же следует использовать принятые сокращения.

**Пример:** «M\_GET\_BB\_P\_PER(ADDRESS,BIT)»

5. **Рекомендуется воздержаться от использования enum-ов.**

Вместо них следует использовать **enum class**.

6. Имя прототипа enum class должно начинаться с префикса «EC\_». К нему можно обращаться только через «::».

Прямое обращение к значению enum class-а без указания пространства имен - запрещено.

**Пример:** «EC\_PORT\_NAME::A»

## 3.6 Оформление .h файлов библиотеки

- 1.
- 2.
- 3.
- 4.