ВВЕДЕНИЕ В БИБЛИОТЕКУ STM32F2_API

Автор: Дерябкин Вадим (Vadimatorik)

2017

ВВЕДЕНИЕ

Данный документ создан с целью донести до пользователя:

- основные сведения о библиотеке (глава 1);
-
-

Оглавление

Глава 1

ЗНАКОМСТВО С БИБЛИОТЕКОЙ

1.1 Введение

Как известно, для микроконтроллеров существует множество различных библиотек, решающих широкий спектр задач. Однако большинство из них используют ресурсы микроконтроллера не экономно. Связано это с тем, что зачастую библиотеки пишутся под абстрактные микроконтроллеры и не учитывают индивидуальные особенности, имеющиеся у конкретных моделей. В своей библиотеке я поставил цель максимально эффективно использовать все доступные ресурсы конкретных серий микроконтроллеров.

1.2 Идеи, лежащие в основе библиотеки

На этапе проектирования библиотеки были отобраны следующие идеи, которые впоследствии легли в ее основу:

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 ещё на этапе компиляции. Отсюда следует, что все режимы работы периферии должны быть заданы статически.
6. Все используемые блоки периферии (как связанные с аппаратной периферией, так и являющиеся логической надстройкой) должны быть объявлены в коде пользователя в виде глобальных const constexpr объектов.

1.3 Краткий обзор реализации библиотеки

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 const constexpr объекты в коде пользователя оперируют созданными на этапе компиляции масками регистров для 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 1.3). 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 const constexpr объекты. В качестве параметра(-ов) конструктора передаётся(-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 - компоновщик 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 1.3, то она обязательно пойдёт 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 объекта (конструктора) перед вызовом main в реальном времени), **не поддерживаются намеренно**.
   - Из того, что все объекты объявлены как const constexpr следует, что у каждого глобального объекта, работающего в реальном времени, имеется метод начальной инициализации (и/или переинициализации), вызов которого необходимо произвести из кода пользователя.

     Это очень оправданно, когда требуется инициализировать объекты в определённом 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 2

СОГЛАШЕНИЕ О НАПИСАНИИ БИБЛИОТЕКИ

2.1 Введение

В данной главе изложен стандарт, которого следует придерживаться на протяжении всего времени написания кода библиотеки, а так же рекомендации по ее (библиотеки) использованию в коде пользователя.

Стандарт распространяется на:

1. средства сборки (раздел 2.2);
2. дерево проекта и именование файлов (раздел 2.3);
3. принятые сокращения (подраздел 2.4);
4. общие правила оформления имён (раздел 2.5);
5. оформление .h файлов библиотеки (раздел 2.6);
6. объявление классов в .h файлах (раздел 2.7).

2.2 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 старее (GNU Tools for ARM Embedded Processors 6-2017-q1-update) 6.3.1 20170215 (release) [ARM/embedded-6-branch revision 245512].

2.3 Дерево проекта и именование файлов

Правила, касающиеся оформления библиотеки:

1. Для файлов, относящихся к работе с блоками аппаратной и программной (абстрактные) периферии, должна существовать своя папка на каждый модуль.
   Пример: *rcc*, *port*, *pwr* и т.д.
   Имя папки должно содержать только название аппаратного модуля, написанного строчными буквами латинского алфавита.

2. Каждая папка, посвящённая определённому блоку периферии (аппаратной или программной), должна содержать следующие файлы:

   - **perfix_moduleName.h**
     В данном файле должны находится классы, относящиеся к определённому блоку периферии. Объекты этих классов можно использовать в коде пользователя.
   - **perfix_moduleName.cpp**
     Если в perfix_moduleName.h всего один класс, то в данном файле находятся методы класса из файла perfix_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.
   - **perfix_moduleName_class_className.cpp**
     В случае, если в файле perfix_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 perfix_moduleName.h один класс, методы, относящиеся к этому классу, должны быть размещены в файле perfix_moduleName.cpp.
   - **perfix_moduleName_constexpr_func.h**
     В данном файле содержатся все constexpr методы, которые используются классом(-и) из файла perfix_moduleName.h. Эти методы, как правило, являются private методами класса(-ов).
     В случае, если в файле perfix_moduleName.h более одного класса, в данном файле должны находятся лишь те методы, которые используются всеми классами файла perfix_moduleName.h.
     В случае, если каждый класс файла perfix_moduleName.h использует лишь свой определенный набор методов, никак не пересекающийся с остальными классами, данный файл создавать не следует.
   - **perfix_moduleName_constexpr_func_class_className.h**
     В случае, если в файле perfix_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.
   - **perfix_moduleName_struct.h**
     В данном файле содержатся все структуры и enum class-ы, используемые всеми классами файла perfix_moduleName.h.
     В случае, если классы не имеют общих структур или enum class-ов, данный файл

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

- **perfix_moduleName_struct_class_className.h**
В случае, если классов в файле perfix_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 «pwr».

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
```

2.4 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).

2.5 Правила оформления имён

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**.

   Заместо 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»

2.6 Оформление .h файлов библиотеки

2.6.1 Общее оформление

1. Файл должен включать в себя защиту от повторного включения в процесс компиляции по типу *ifndef-define-endif*, оканчивающуюся пустой строкой.
   Пример:

```
#ifndef STM32F2_API_STM32_F20X_F21X_PORT_H_
#define STM32F2_API_STM32_F20X_F21X_PORT_H_

#endif
```

2. После *define* строки защиты следует пустая строка, за которой располагается *include* на файл конфигурации библиотеки, имеющий имя **perfix_conf.h** (название библиотеки выступает в качестве префикса).
   Пример:

```
#define STM32F2_API_STM32_F20X_F21X_PORT_H_

#include "stm32_f20x_f21x_conf.h"
```

3. В случае, если файл стоит включать в процесс компиляции только при каком-то условии, это условие (обернутое в *ifdef*) необходимо указать через одну пустую строку после *include* файла конфигурации библиотеки. Блок *endif*, закрывающий тело блока условной компиляции должен быть написан без пустой строки перед *endif*, закрывающим блок защиты повторной компиляции.
   Пример:

```
#ifndef STM32F2_API_STM32_F20X_F21X_PORT_H_
#define STM32F2_API_STM32_F20X_F21X_PORT_H_

#include "stm32_f20x_f21x_conf.h"

#ifdef MODULE_PORT

CODE

#endif
#endif
```

2.7 Объявление классов в .h файлах

1. Общие сведения об оформлении class-ов в .h файлах (подраздел 2.7.1).
2. В public области должны располагаться (с соблюдением последовательности сверху вниз):
   - конструктор(-ы) класса (подраздел 2.7.2);
   - constexpr конструктор(-ы) класса (подраздел 2.7.3);
   - constexpr методы класса (подраздел 2.7.4);
   - доступные пользователю нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных (подраздел 2.7.5);
   - доступные пользователю нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 2.7.6;
   - доступные пользователю статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип (подраздел 2.7.7);
   - доступные пользователю статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип (подраздел 2.7.8);
   - открытие переменные и константы класса, доступные пользователю напрямую (подраздел 2.7.9).
3. В private область должны располагаться (с соблюдением последовательности сверху вниз):
   - внутренние constexpr методы класса, возвращающие значение стандартного типа или указатель на стандартный тип данных (подраздел 2.7.10);
   - внутренние constexpr методы класса, возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 2.7.11);
   - закрытые нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных (подраздел 2.7.12);
   - закрытые нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 2.7.13);

- закрытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип (подраздел 2.7.14);
- закрытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип (подраздел 2.7.15);
- закрытые константы класса стандартных типов (подраздел 2.7.16).
- закрытые константы класса нестандартных типов (подраздел 2.7.17).
- закрытые переменные класса стандартных типов (подраздел 2.7.18).
- закрытые переменные класса нестандартных типов (подраздел 2.7.19).

2.7.1 Общие сведения об оформлении class-ов в .h файлах

- Между зарезервированным словом class и именем класса ставится один (1) пробел.
- Между последним символом имени класса и открывающейся фигурной скобкой ставится один (1) пробел.
- Сначала идет public, а за ним private область.
- «}» (скобка закрывающая тело класса) должна находится на новой строке.

```
class name_class {
public:
private:
};
```

2.7.2 Конструктор(-ы) класса

Использование не constexpr конструкторов классов запрещено. Это связано с неочевидной последовательностью вызова конструкторов глобальных объектов, которая может привести к неверной инициализации объекта (если явно не указывать последовательность вызовов с помощью специальных директив компоновщика). Например, сначала будет предпринята попытка инициализировать внешнюю переферию (за пределами микроконтроллера), не инициализировав интерфейс, по которому она подключена.

В случае если пользователь все же создаст объект, конструктор которого будет требовать выполнения кода функции конструктора во время инициализации, вызов его метода инициализации произведен не будет (объект останется не инициализированным).

2.7.3 Constexpr конструктор(-ы) класса

- В случае, если конструкторов несколько, они должны быть расположены от большего количества входных параметров к меньшему.
- Реализация самого конструктора не должна находится в теле класса. Ее (реализацию конструктора) следует вынести в отдельный файл.
- Перед словом constexpr должен быть выполнен отступ в 1 tab.

- Между словом constexpr и именем конструктора(-ов) ставится один (1) пробел.
- После имени конструктора должен быть выполнен один (1) пробел.
- Аргументы конструктора(-ов) в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов конструктора должен быть отступ в 1 пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».

Пример:

```
    constexpr pin ( const pin_config_t *pin_cfg_array, const uint32_t pin_cout );
    constexpr pin ( const pin_config_t *pin_cfg_array );
```

2.7.4 Constexpr методы класса

Размещение constexpr методов в разделе public запрещено и не имеет смысла.

2.7.5 Доступные пользователю нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных

- Перед типом возвращаемого значения должен быть выполнен отступ в один (1) tab.
- Имена методов должны быть выравнены с помощью tab с остальными методами этого типа. Выравнивание методов других типов производится по иной сетке.
- Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов метода должен быть отступ в один (1) пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».
- В случае, если метод не изменяет данные класса, после параметров в скобках следует поставить один (1) пробел, после чего слово «const;». «;» закрывает заголовок функции.

Пример:

```
    void    set             ( void ) const;
    void    reset           ( void ) const;
    void    invert          ( void ) const;
    int     read            ( void ) const;
```

2.7.6 Доступные пользователю нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных

- В качестве нестандартного типа может выступать enum class или структура.
- В качестве указателя на нестандартный тип может выступать указатель на enum class переменную или структуру.

- Перед возвращаемым типом метода должен быть отступ в один (1) tab.
- Имена методов должны быть выравнены с помощью tab с остальными методами этого типа. Выравнивание методов других типов производится по иной сетке.
- Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов метода должен быть отступ в 1 пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».

Пример:

```
        E_ANSWER_GP     met_g   ( void );
```

2.7.7 Доступные пользователю статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип

- Перед зарезервированным словом «static» должен быть отступ в один (1) tab.
- Между «static» и типом возвращаемого значения должен быть выполнен отступ в один (1) пробел.
- Имена методов должны быть выравнены с помощью tab с остальными методами этого типа. Выравнивание методов других типов производится по иной сетке.
- Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов метода должен быть отступ в 1 пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».
- Так как предполагается, что данный метод будет работать с объектом(-ами) класса, в котором(-ых) описан его заголовок, то первым аргументом метода должен быть указатель на void, который внутри класса будет разыменован в указатель на объект класса, в котором был объявлен.
  Используется указатель на void, а не на объект класса с целью совместимости с FreeRTOS, написанной на C.
- Первый аргумент метода следует называть «void *obj».

Пример:

```
        static void     task_1  ( void *obj );
        static int      m_1     ( void *obj );
```

2.7.8 Доступные пользователю статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип

- В качестве нестандартного типа может выступать enum class или структура.
- В качестве указателя на нестандартный тип может выступать указатель на enum class переменную или структуру.
- Перед зарезервированным словом «static» должен быть отступ в один (1) tab.
- Между «static» и типом возвращаемого значения должен быть выполнен отступ в один (1) пробел.
- Имена методов должны быть выравнены с помощью tab с остальными методами этого типа. Выравнивание методов других типов производится по иной сетке.
- Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов метода должен быть отступ в 1 пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».
- Так как предполагается, что данный метод будет работать с объектом(-ами) класса, в котором(-ых) описан его заголовок, то первым аргументом метода должен быть указатель на void, который внутри класса будет разыменован в указатель на объект класса, в котором был объявлен.
  Используется указатель на void, а не на объект класса с целью совместимости с FreeRTOS, написанной на C.
- Первый аргумент метода следует называть «void *obj».

Пример:

```
        static E_ANSWER_GP      met_a   ( void *obj );
```

2.7.9 Открытие переменные и константы класса, доступные пользователю напрямую

Размещение переменных (изменяемых или заданных как const) в public области запрещено. Даже в случае, если требуется просто читать/записывать одну переменную, следует сделать отдельный метод(-ы) для этого. Так как прямое чтение данных из класса является нарушением ООП.

2.7.10 Внутренние constexpr методы класса, возвращающие значение стандартного типа или указатель на стандартный тип данных

- Реализация тела функции не должна находится в теле класса. Она (реализация тела функции) должна быть вынесена в отдельный файл. Допускаются только заголовки функций.

- Перед словом constexpr должен быть выполнен отступ в один (1) tab.
- Между зарезервированным словом constexpr и типом возвращаемого значения требуется поставить один (1) пробел.
- Имена методов должны быть выравнены с помощью tab с остальными методами этого типа. Выравнивание методов других типов производится по иной сетке.
- Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов метода должен быть отступ в 1 пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».

Пример:

```
        constexpr uint32_t      moder_reg_reset_init_msk_get    ( EC_PORT_NAME port_name );
```

2.7.11 Внутренние constexpr методы класса, возвращающие значение нестандартного типа или указатель на нестандартный тип данных

- В качестве нестандартного типа может выступать enum class или структура.
- В качестве указателя на нестандартный тип может выступать указатель на enum class переменную или структуру.
- Реализация тела функции не должна находится в теле класса. Она (реализация тела функции) должна быть вынесена в отдельный файл. Допускаются только заголовки функций.
- Перед словом constexpr должен быть выполнен отступ в один (1) tab.
- Между зарезервированным словом constexpr и типом возвращаемого значения требуется поставить один (1) пробел.
- Имена методов должны быть выравнены с помощью tab с остальными методами этого типа. Выравнивание методов других типов производится по иной сетке.
- Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
- Внутри скобок перечисления аргументов метода должен быть отступ в 1 пробел с каждой стороны.
  Пример: «(uint32_t a, uint8_t b)».

```
        constexpr EC_PORT_NAME  port_name_get   ( uint32_t value_reg );
```

2.7.12 Закрытые нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных

Оформляются так же, как и открытые нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных (подраздел 2.7.5).

2.7.13 Закрытые нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных

Оформляются так же, как и открытые нестатические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 2.7.6).

2.7.14 Закрытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип

Оформляются так же, как и открытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип (подраздел 2.7.7).

2.7.15 Закрытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип

Оформляются так же, как и открытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип (подраздел 2.7.8).

2.7.16 Закрытые константы класса стандартных типов

- На одной строке допустимо объявлять лишь одну константу.
- Перед зарезервированным словом «const» должен быть поставлен один (1) tab.
- Имена констант должны быть выравнены с помощью tab с остальными константами этого типа. Выравнивание констант других типов производится по иной сетке.

```
        const uint32_t  count;
```

2.7.17 Закрытые константы класса нестандартных типов

Оформляются так же, как и закрытые константы класса стандартных типов (подраздел 2.7.16)

```
        const global_port_msk_reg_struct        gb_msk_struct;
```

2.7.18 Закрытые переменные класса стандартных типов

- На одной строке допустимо объявлять лишь одну переменную.
- Перед типом должен быть поставлен один (1) tab.
- Имена переменных должны быть выравнены с помощью tab с остальными переменными этого типа. Выравнивание переменных других типов производится по иной сетке.

```
        uint32_t        flag;
```

2.7.19 Закрытые переменные класса нестандартных типов

Оформляются так же, как и закрытые переменные класса стандартных типов (подраздел 2.7.18)

```
        EC_FL    mb_msk_struct;
```