ВВЕДЕНИЕ В БИБЛИОТЕКУ STM32F2_API

Автор: Дерябкин Вадим (Vadimatorik)

2017

ВВЕДЕНИЕ

Данный документ создан с целью донести до пользователя:

- основные сведения о библиотеке (часть I);
- соглашения о написании и оформлении библиотеки (часть II);
- структуру абстрактных аппаратных блоков периферии и примеры их использования (часть III);

Оглавление

Часть I

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 указатель на const constexpr глобальную(-ые) структуру(-ы) (может передаваться указатель как на одну структуру, так и на массива структур). 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 требуют 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 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.

Часть II

СОГЛАШЕНИЕ О НАПИСАНИИ И ОФОРМЛЕНИИ БИБЛИОТЕКИ

1.4 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:

- средства сборки (раздел 1.5);
- дерево проекта и именование файлов (раздел 1.6);
- принятые сокращения (подраздел 1.7);
- общие правила оформления имён (раздел 1.8);
- оформление .h файлов библиотеки (раздел 1.9);
- объявление классов в .h файлах (раздел 1.10);
- объявление упакованных структур (packed struct) (раздел 1.11);
- объявление структур (раздел 1.12);
- объявление enum class-ов (раздел 1.13);
- объявление макросов (раздел 1.14).

1.5 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].

1.6 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(-ами) из файла *perfix_ moduleName.h*. Эти методы 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 определённый 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 определённые 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
```

1.7 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 на бит).

1.8 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»

1.9 Оформление .h файлов библиотеки

1.9.1 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
```

4. Внутри всех необходимых обёрток, описанных выше (на месте слова «CODE» примера из предыдущего пункта), располагается основное содержимое, уникальное для каждого типа .h файла:

   - *perfix_moduleName.h* (дел 1.9.2);
   - *perfix_moduleName_constexpr_func.h* (подраздел 1.9.3);
   - *perfix_moduleName_constexpr_func_class_className.h* (подраздел 1.9.4);
   - *perfix_moduleName_struct.h* и *perfix_moduleName_struct_class_className.h* (подраздел 1.9.5);

1.9.2 Содержимое perfix_moduleName.h файла

1. include *perfix_moduleName_struct.h* и *perfix_moduleName_constexpr_func.h* файлов, если таковые имеются;

2. пустая строка;

3. краткое описание всего модуля, включающее в себя описание всех классов модуля, обернутое в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук.
4. пустая строка;
5. краткое описание класса, обернутое в много строчный комментарий;
6. пустая строка;
7. include *perfix_ moduleName_ struct_ class_ className.h*, если имеется;
8. пустая строка;
9. тело класса;
10. пустая строка;
11. include файла *perfix_ moduleName_ constexpr_ func_ class_ className.h*, если имеется;
12. пустая строка;
13. пункты 5-12 повторить для всех требуемых классов;
14. пустая строка.

Пример всего файла:

```
#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

#include "stm32_f20x_f21x_port_struct.h"
#include "stm32_f20x_f21x_port_constexpr_func.h"

/**********************************************************************
 * Краткое описание модуля...
 **********************************************************************/

/*
 * Краткое описание класса pin...
 */

#include "stm32_f20x_f21x_port_struct_class_pin.h"

class pin {
public:
  ...

private:
  ...
};

#include "stm32_f20x_f21x_port_constexpr_func_class_pin.h"

/*
 * Краткое описание класса global_port...
 */

#include "stm32_f20x_f21x_port_struct_class_global_port.h"

class global_port {
public:
  ...
private:
  ...
};
```

```
#include "stm32_f20x_f21x_port_constexpr_func_class_global_port.h"

#endif
#endif
```

1.9.3 Содержимое perfix_moduleName_constexpr_func.h файла

1. include *perfix_ moduleName_ struct.h* файла, если таковой имеются;
2. пустая строка;
3. краткое описание constexpr функции, обернутое в много строчный комментарий;
4. тело функции;
5. пустая строка;
6. пункты 3-5 повторить для всех имеющихся методов;

Пример всего файла:

```
#ifndef STM32F2_API_PORT_STM32_F20X_F21X_PORT_CONSTEXPR_FUNC_H_
#define STM32F2_API_PORT_STM32_F20X_F21X_PORT_CONSTEXPR_FUNC_H_

#include "stm32_f20x_f21x_conf.h"

#ifdef MODULE_PORT

#include "stm32_f20x_f21x_port_struct.h"

/*
 * Краткое описание constexpr функции p_base_port_address_get...
 */

constexpr uint32_t p_base_port_address_get( EC_PORT_NAME port_name ) {
  CODE;
}

/*
 * Краткое описание constexpr функции bb_p_port_look_key_get...
 */
constexpr uint32_t bb_p_port_look_key_get ( EC_PORT_NAME port_name ) {
  CODE;
}

#endif
#endif
```

1.9.4 Содержимое perfix_moduleName_constexpr_func_class_className.h файла

1. include *perfix_ moduleName_ struct.h* и *perfix_ moduleName_ struct_ class_ className.h* файлов, если таковые имеются;
2. пустая строка;
3. комментарий о начале области с constexpr конструктором(-ами), обернутый в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук. После комментария должна следовать пустая строка;
4. тело constexpr конструктора;

5. пустая строка;
6. пункты 3-5 повторяются для всех имеющихся конструкторов;
7. комментарий о начале области с constexpr методами, , обернутый в много строчный комментарий с явно прописанными границами, бросающимися в газа;
8. пустая строка;
9. краткое описание constexpr функции, обернутое в много строчный комментарий;
10. тело функции;
11. пустая строка;
12. пункты 9-11 повторить для всех имеющихся методов;

Пример всего файла:

```
#ifndef STM32F2_API_PORT_STM32_F20X_F21X_PORT_CONSTEXPR_FUNC_CLASS_PIN_H_
#define STM32F2_API_PORT_STM32_F20X_F21X_PORT_CONSTEXPR_FUNC_CLASS_PIN_H_

#include "stm32_f20x_f21x_conf.h"

#ifdef MODULE_PORT

#include "stm32_f20x_f21x_port_struct.h"
#include "stm32_f20x_f21x_port_struct_class_pin.h"

/*******************************************************************
 * Область constexpr конструкторов.
 *******************************************************************/
constexpr pin::pin ( const pin_config_t* pin_cfg_array ):
  КОД ИНИЦИАЛИЗАЦИИ ПЕРЕМЕННЫХ КЛАССА {};

/*******************************************************************
 * Область constexpr функций.
 *******************************************************************/

/*
 * Краткое описание constexpr функции set_msk_get...
 */
constexpr uint32_t pin::set_msk_get ( const pin_config_t* const pin_cfg_array ) {
  CODE;
}

/*
 * Краткое описание constexpr функции p_base_port_address_get...
 */
constexpr uint32_t pin::p_base_port_address_get ( const pin_config_t* const
                                                  pin_cfg_array ) {
  CODE;
}

#endif
#endif
```

1.9.5 Содержимое perfix_moduleName_struct.h и erfix_moduleName_struct_class_className.h файлов

В файлах располагаются следующие конструкции (сверху вниз):

1. packed структуры;
2. структуры;
3. enum class-ы;

4. макросы.

Правила оформления следующие:

1. перед каждым пунктом размещается комментарий о начале области этого пункта, обернутое в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук;

2. перед каждым элементом пункта размещается краткое описание элемента, обернутое в много строчный комментарий. После краткого описания пустая строка не ставится;

3. между предыдущим элементом пункта и комментарием следующего элемента этого же пункта вставляется пустая строка.

4. между предыдущим элементом пункта и комментария области следующего пункта вставляется пустая строка.

Пример всего файла:

```
#ifndef STM32F2_API_PORT_STM32_F20X_F21X_PORT_STRUCT_CLASS_PIN_H_
#define STM32F2_API_PORT_STM32_F20X_F21X_PORT_STRUCT_CLASS_PIN_H_

#include "stm32_f20x_f21x_conf.h"

#ifdef MODULE_PORT

/*****************************************************************
 * Область упакованных структур.
 *****************************************************************/

/*
 * Краткое описание упакованной структуры.
 */
struct __attribute__((packed)) port_registers_struct {
  ПОЛЯ СТРУКТУРЫ;
};

/*****************************************************************
 * Область структур.
 *****************************************************************/

/*
 * Краткое описание структуры.
 */
struct st_struct {
  ПОЛЯ СТРУКТУРЫ;
};

/*****************************************************************
 * Область enum class.
 *****************************************************************/

/*
 * Краткое описание num classa-.
 */
enum class EC_PORT_PIN_NAME {
  VALUE_FIELDS;
};


/*****************************************************************
 * Область макросов.
 *****************************************************************/

/*
 * Краткое описание макроса.
 */
#define M_PIN_CFG_ADC(PORT,PIN) {
  VALUE_FIELDS;
}
```

```
#endif
#endif
```

1.10 Объявление классов в .h файлах

1. Общие сведения об оформлении class-ов в .h файлах (подраздел 1.10.1).
2. В public области должны располагаться (с соблюдением последовательности сверху вниз):
   - конструктор(-ы) класса (подраздел 1.10.2);
   - constexpr конструктор(-ы) класса (подраздел 1.10.3);
   - constexpr методы класса (подраздел 1.10.4);
   - доступные пользователю не статические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных (подраздел 1.10.5);
   - доступные пользователю не статические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 1.10.6;
   - доступные пользователю статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип (подраздел 1.10.7);
   - доступные пользователю статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип (подраздел 1.10.8);
   - открытие переменные и константы класса, доступные пользователю напрямую (подраздел 1.10.9).
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 1.10.10);
   - внутренние constexpr методы класса, возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 1.10.11);
   - закрытые не статические методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип данных (подраздел 1.10.12);
   - закрытые не статические методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип данных (подраздел 1.10.13);
   - закрытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение стандартного типа или указатель на стандартный тип (подраздел 1.10.14);
   - закрытые статические (static) методы класса, выполняющиеся в реальном времени и возвращающие значение нестандартного типа или указатель на нестандартный тип (подраздел 1.10.15);
   - закрытые константы класса стандартных типов (подраздел 1.10.16).
   - закрытые константы класса нестандартных типов (подраздел 1.10.17).
   - закрытые переменные класса стандартных типов (подраздел 1.10.18).
   - закрытые переменные класса нестандартных типов (подраздел 1.10.19).

1.10.1 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 идёт public, а за ним private область.
- «}» (скобка закрывающая тело класса) должна находится на новой строке.

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

1.10.2 Конструктор(-ы) класса

Использование не constexpr конструкторов классов запрещено. Это связано с не очевидной 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).

1.10.3 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 );
```

1.10.4 Constexpr методы класса

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

1.10.5 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;
```

1.10.6 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 );
```

1.10.7 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 );
```

1.10.8 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 );
```

1.10.9 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.

1.10.10 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 );
```

1.10.11 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 );
```

1.10.12 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 1.10.5).

1.10.13 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 1.10.6).

1.10.14 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 1.10.7).

1.10.15 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 1.10.8).

1.10.16 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;
```

1.10.17 Закрытые константы класса нестандартных типов

Оформляются так же, как и закрытые константы класса стандартных типов (подраздел 1.10.16)

```
const global_port_msk_reg_struct  gb_msk_struct;
```

1.10.18 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;
```

1.10.19 Закрытые переменные класса нестандартных типов

Оформляются так же, как и закрытые переменные класса стандартных типов (подраздел 1.10.18).

```
EC_FL mb_msk_struct;
```

1.11 Объявление упакованных структур (packed struct)

1.11.1 Когда стоит объявлять структуру как упакованную (packed struct)?

Структуру следует объявить как упакованную (packed struct), если она:

- описывает совокупность регистров аппаратного блока периферии (у которого, как известно, регистры имеют четко фиксированный размер и порядок следования);
- описывает образ памяти регистров аппаратного блока;
- описывает структуру какого-либо пакета со строго фиксированными полями.

1.11.2 Размещение упакованных структур (packed struct)

Прототипы упакованных структур должны быть размещены только в **.h** файлах. Экземпляры - в .cpp.

1.11.3 Оформление упакованных структур (packed struct)

- перед первой объявленной упакованной структурой размещается комментарий о начале соответствующей области (области упакованных структур), обернутый в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук. После комментария должна следовать пустая строка;
- перед каждой packed структурой размещается ее краткое описание, обернутое в много строчный комментарий. После краткого описания пустая строка не ставится;
- заголовок структуры следует оформить следующим образом:
  1. ключевое слово struct без отступов в начале строки;
  2. отступ в один (1) пробел;
  3. директива препроцессора «__attribute__((packed))»;
  4. пробел;
  5. имя структуры;
  6. пробел;
  7. открывающая тело packed структуры скобка «{»;

- поля структуры следует оформлять следующим образом:
  1. каждая строка начинается с отступа в один (1) tab;
  2. ключевое слово volatile;
  3. один (1) пробел;
  4. тип поля;
  5. требуемое количество отступов, выполненных с помощью tab;
  6. имя поля;
  7. «;»;
  8. требуемое количество tab;
  9. «// » (// + пробел) + одно строчный комментарий.
- все имена полей структуры должны быть выравнены с помощью tab между собой;
- после последнего поля структуры следует скобка закрытия тела структуры («}»);
- после последней packed структуры вставляется пуста строка.

Пример области упакованных структур:

```
/*****************************************************************
 * Область упакованных структур.
 *****************************************************************/

/*
 * Перечень регистров физического порта ввода - вывода.
 */
struct __attribute__((packed)) port_registers_struct {
  volatile uint32_t mode;    // Регистр выбора режима работы выводов.
  volatile uint32_t otype;   // Регистр выбора режима выхода
                             // ( в случае, если вывод настроен как выход ).
  volatile uint32_t ospeede;// Регистр выбора скорости выводов.
  volatile uint32_t pupd;    // Регистр включения подтяжки выводов.
  volatile uint32_t id;      // Регистр с текущими данными на входе вывода.
  volatile uint32_t od;      // Регистр с выставленными пользователем на выход данными
                             // ( в случае, если вывод настроен как выход ).
  volatile uint32_t bsr;     // Регистр быстрой установки состояния выводов
                             // ( когда вывод настроен как выход ).
  volatile uint32_t lck;     // Регистр блокировки настроек.
  volatile uint32_t afl;     // Младший регистр настройки альтернативных функций выводов.
  volatile uint32_t afh;     // Старший регистр настройки альтернативных функций выводов.
};
```

1.12 Объявление структур

1.12.1 Когда стоит оборачивать данные в структуру?

Данные следует обернуть в структуру, если:

- требуется передавать более одного параметра в конструктор класса;
- требуется вернуть из функции более одного параметра;

Замечание: перед тем, как оборачивать данные в структуру проверьте, не имеются ли у вас условий, согласно которым данные должны быть обернуты в упакованную структуру (раздел 1.11)

1.12.2 Размещение структур

Прототипы структур должны быть размещены только в **.h** файлах. Экземпляры - в **.cpp**.

1.12.3 Оформление структур

- перед первой объявленной структурой размещается комментарий о начале соответствующей области (области структур), обернутый в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук. После комментария должна следовать пустая строка;
- перед каждой структурой размещается ее краткое описание, обернутое в многострочный комментарий. После краткого описания пустая строка не ставится;
- заголовок структуры следует оформить следующим образом:
  1. ключевое слово struct без отступов в начале строки;
  2. отступ в один (1) пробел;
  3. имя структуры;
  4. пробел;
  5. открывающая тело packed структуры скобка «{»;
- поля структуры следует оформлять следующим образом:
  1. каждая строка начинается с отступа в один (1) tab;
  2. тип поля;
  3. требуемое количество отступов, выполненных с помощью tab;
  4. имя поля;
  5. «;»;
  6. требуемое количество tab;
  7. «// » (// + пробел) + одно строчный комментарий.
- все имена полей структуры должны быть выравнены с помощью tab между собой;
- после последнего поля структуры следует скобка закрытия тела структуры («}»);
- после последней структуры вставляется пуста строка.

Пример области структур:

```
/*****************************************************************
 * Область структур.
 *****************************************************************/

/*
 * Краткое описание структуры...
 */
struct a {
  uint32_t  b;     // Пояснение к полю b.
  uint32_t  c;     // Пояснение к полю c.
};
```

1.13 Объявление enum class-ов

1.13.1 Когда стоит объявлять enum class?

Enum class-ы стоит объявлять, если какому-то полю структуры/переменной требуется присвоить какое-то одно значение из заранее известного ряда, каждому из которых можно присвоить уникальное имя.

1.13.2 Размещение enum class-ов

Прототипы enum class-ов должны быть размещены только в **.h** файлах.

1.13.3 Оформление enum class-ов

- перед первым объявленным enum class-ом размещается комментарий о начале соответствующей области (области enum class-ов), обернутый в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук. После комментария должна следовать пустая строка;
- перед каждым enum class-ом размещается его краткое описание, обернутое в многострочный комментарий. После краткого описания пустая строка не ставится;
- заголовок enum class-а следует оформить следующим образом:
  1. ключевое словосочетание «enum class» без отступов в начале строки;
  2. отступ в один (1) пробел;
  3. имя enum class-а (в соответствии с правилами написания имен enum class-ов);
  4. пробел;
  5. открывающая тело enum class-а скобка «{»;
- значения enum class-ов следует оформлять следующим образом:
  1. каждое значение начинается с отступа в один (1) tab;
  2. имя значения;
  3. требуемое количество отступов, выполненных с помощью tab;
  4. «= » («=» + один (1) tab);
  5. непосредственное цифровое значение;
  6. «,» (в случае, если элемент последний, запятая не ставится).
  7. требуемое количество tab;
  8. «// » (// + пробел) + однострочный комментарий.
- выравнивание производится по знаку «=» с левой стороны;
- после последнего значения enum class-а следует скобка закрытия его тела и точка с запятой («};»), расположенная на новой строке;
- после последнего enum class - а вставляется пуста строка.

Пример области структур:

```
/*****************************************************************
 * Область enum class- ов.
 *****************************************************************/

enum class EC_PORT_NAME {
  A = 0,
  B = 1,
  C = 2,
  D = 3,
  H = 4
};
```

1.14 Объявление макросов

1.14.1 Когда стоит объявлять макрос?

Макрос следует объявить, если имеется фрагмент кода, многократно повторяющийся в коде программы с незначительными изменениями в каждом конкретном случае. При этом использовать отдельную constexpr функцию нерационально или невозможно.
Пример: при заполнении массива структур инициализации выводов некоторое количество выводов инициализируются как входы ADC. Для того, чтобы не писать каждый раз все параметры каждого пина, достаточно будет воспользоваться макросом, в котором нужно указать изменяющиеся параметры: имя порта и вывода.

1.14.2 Размещение макросов

Прототипы макросов должны быть размещены в **.h** файлах.

1.14.3 Оформление макросов

- перед первым объявленным макросом размещается комментарий о начале соответствующей области (области макросов), обернутый в многострочный комментарий с явно обозначенными границами символами «*» в количестве 70 штук. После комментария должна следовать пустая строка;
- перед каждым макросом размещается его краткое описание, обернутое в многострочный комментарий. После краткого описания пустая строка не ставится;
- заголовок макроса следует оформить следующим образом:
  1. ключевое слово #define без отступов в начале строки;
  2. отступ в один (1) пробел;
  3. имя макроса (в соответствии с правилами написания макросов);
  4. открывающаяся скобка («(»);
  5. параметры макроса через запятую без пробелов в соответствии с правилами написания макросов;
  6. закрывающаяся скобка («)»);
  7. тело макроса;

Пример:

```
/*****************************************************************
 * Область макросов.
 *****************************************************************/

/*
 * Возвращает структуру конфигурации вывода,
 * настроенного на вход, подключенный к ADC.
 */
#define M_PIN_CFG_ADC(PORT,PIN) {                          \
   .port             = PORT,                               \
   .pin_name         = PIN,                                \
   .mode             = EC_PIN_MODE::ANALOG,                \
   .output_config    = EC_PIN_OUTPUT_CFG::NOT_USE,         \
   .speed            = EC_PIN_SPEED::LOW,                  \
   .pull             = EC_PIN_PULL::NO,                    \
   .af               = EC_PIN_AF::NOT_USE,                 \
   .locked           = EC_LOCKED::LOCKED,                  \
   .state_after_init = EC_PIN_STATE_AFTER_INIT::NO_USE     \
}
```

Часть III

СТРУКТУРА АБСТРАКЦИЙ АППАРАТНЫХ БЛОКОВ ПЕРИФЕРИИ И ПРИМЕРЫ ИХ ИСПОЛЬЗОВАНИЯ

1.15 Введение

Для работы с аппаратными блоками периферии в библиотеке присутствуют абстрактные блоки, которые используются пользователем или драйверами более высокого уровня. Они обеспечивают высокую скорость взаимодействия пользователя или драйверов высокого уровня с аппаратной периферией, сохраняя при этом удобный для восприятия и использования уровень абстракции.

Модули библиотеки позволяют комфортно взаимодействовать со следующими аппаратными блоками:

- порты ввода-вывода общего назначения (раздел 1.16);

Каждый абстрактный блок представляет из себя класс/группу классов, экземпляр(-ы) которого(-ых) должен(-ы) быть объявлен(-ы) в коде пользователя/драйвере более высокого уровня как глобальный(-е) const constexpr объект(-ы). Более подробная информация об объявлении и использовании блоков приведена в соответствующих разделах.

1.16 Модуль работы с портами ввода-вывода общего назначения (PORT)

Данный модуль состоит из следующих классов:

- **pin** (раздел 1.16.1);
- **global_port** (раздел ??);

1.16.1 Класс pin

Общие сведения о классе

Объект данного класса позволяет работать с конкретным выводом порта и предоставляет следующие методы:

- **constexpr pin (const pin_config_t* const pin_cfg_array);**
  Конструктор класса принимает указатель на структуру/массив структур конфигурации режима работы вывода.
  Замечание 1: в случае, если метод **reinit** объекта будет использован в коде пользователя или драйверах высокого уровня, структура/массив структур, на которую(-ый) ссылается указатель, полностью попадает во flash. В связи с тем, что метод **reinit** будет использовать структуру/элемент массива.
  Замечание 2: важно понимать, что все структуры конфигурации вывода (в случае, если передаётся указатель на массив структур) должны относится к одному физическому выводу (изменяя лишь режим его работы).

- **void set (void) const;**
  Устанавливает 1 на выводе, если вывод сконфигурирован как выход.

- **void reset (void) const;**
  Устанавливает 0 на выводе, если вывод сконфигурирован как выход.

- **void set (uint8_t state) const;**
  Устанавливает заданное состояние на выводе (1 или 0), если вывод сконфигурирован как выход.

- **void set (bool state) const;**
  Устанавливает заданное состояние на выводе (true == 1 или false == 0), если вывод сконфигурирован как выход.

- **void set (int state) const;**
  Устанавливает заданное состояние на выводе (1 или 0), если вывод сконфигурирован как выход.

- **void invert (void) const;**
  Логическое **НЕ** состояния на выходе вывода, если вывод сконфигурирован как выход.

- **int read (void) const;**
  Возвращает логическое состояние на входе вывода, если вывод сконфигурирован как вход.

- **EC_ANSWER_PIN_REINIT reinit (uint8_t number_config) const;**
  Переинициализирует вывод в ходе выполнения программы в выбранную конфигурацию.
  Замечание: первичная инициализация вывода по нулевой (0) структуре производится объектом *global_port*. Что избавляет от надобности начальной инициализации вывода. Данный метод следует вызывать, если требуется изменить конфигурацию вывода на иную в ходе выполнения программы (при условии, что он не был заблокирован объектом *global_port*).