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

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 конструктов 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 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 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 старее (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, pwr 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:

   - **perfix_moduleName.h**
     В данном файле должны находится классы, относящиеся к определенному 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
```

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**.

   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»

3.6 Оформление .h файлов библиотеки

3.6.1 Общее оформление

1. «.h» файл должен включать в себя защиту от повторного включения файла в процесс компиляции.

```
#ifndef STM32F2_API_STM32_F20X_F21X_PORT_H_
#define STM32F2_API_STM32_F20X_F21X_PORT_H_

#endif
```

3.6.2 Классы

1. Типичный образец класса выглядит следующим образом:

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

   Из данного примера следует:

   - Между зарезервированным словом class и именем класса один (1) пробел.
   - Между последним символом имени класса и открывающейся фигурной скобкой один (1) пробел.

- Сначала идет public область, а затем private.
- «}» (скобка закрывающая тело класса) должна находится на новой строке.

2. В public области должны располагаться (с соблюдением последовательности сверху вниз):

   - Constexpr конструктор класса.
     - Перед словом constexpr должен быть выполнен отступ в 1 tab.
     - 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.
     - После слова constexpr и до первого символа имени конструктора, ставится один (1) пробел.
     - После имени конструктора (класса) должен быть выполнен один (1) пробел.
     - Аргументы конструктора в скобках должны быть разделены «, » (запятая + пробел).
       По бокам (от каждой скобки) должен быть отступ в 1 пробел.
       Пример: «(uint32_t a, uint8_t b)».
   - В данной области запрещено размещать constexpr методы (за исключением конструктора(-ов)).
   - Доступные пользователю методы, выполняющиеся в реальном времени.
     - Перед типом возвращаемого значения должен быть выполнен отступ в 1 tab.
     - Имена методов должны быть выравнены с помощью tab с остальными методами public области (кроме конструкторов).
       Следует отметить, что речь идет лишь об одинаковых по структуре методах. В случае, если имеются и static методы, они должны будут выравнены между собой по другой сетке (об этом далее).
     - В случае, если имеется(-ются) static метод(-ы) - он(они) должен(-ы) быть расположен(-ы) самым(ыми) последним(-и) со смещением в выравнивании в одно поле (вместо типа возвращаемого значения - static, вместо имени тип - возвращаемого значения).
       После чего выравнивание идет уже по другой сетки (уникальной для static методов).
     - В случае, если часть (не все) функций возвращают enum class, то такие функции следует так же разместить после остальных (но до static методов) и создать им отдельную сетку с учетом ширины имени enum class-а.
     - Аргументы методов в скобках должны быть разделены «, » (запятая + пробел).
       По бокам (от каждой скобки) должен быть отступ в 1 пробел.
       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.

   После конструктора(-ов) и методов разных типов (обычные и с префиксом static) следует оставлять пустую строку.

3. В private области должны располагаться (с соблюдением последовательности сверху вниз):

   - Скрытые constexpr методы класса.
     - Перед словом constexpr должен быть выполнен отступ в 1 tab.
     - Реализация тела функции не должно находится в теле класса (должна быть вынесена в отдельный файл).
     - Имя функции должно быть выравнено с помощью tab с остальными методами private области.
     - После имени конструктора (класса) должен быть выполнен один (1) пробел.
     - Аргументы в скобках должны быть разделены «, » (запятая + пробел).
       По бокам (от каждой скобки) должен быть отступ в 1 пробел.
       Пример: «(uint32_t a, uint8_t b)».
     - Параметры методов должны быть выровнены с помощью tab.
   - Внутренние методы класса, выполняющиеся в реальном времени.
     - Перед типом возвращаемого значения должен быть выполнен отступ в 1 tab.
     - Имена методов должны быть выравнены с помощью tab с остальными методами private области (только своего типа).
     - В случае, если имеется(-ются) static метод(-ы) - он(они) должен(-ы) быть расположен(-ы) самым(ыми) последним(-и) со смещением в выравнивании в одно поле (вместо типа возвращаемого значения - static, вместо имени тип - возвращаемого значения).
       После чего выравнивание идет уже по другой сетки (уникальной для static методов).
     - В случае, если часть (не все) функций возвращают enum class, то такие функции следует так же разместить после остальных (но до static методов) и создать им отдельную сетку с учетом ширины имени enum class-а.
     - Параметры в скобках должны быть разделены «, » (запятая + пробел), в по бокам (от каждой скобки) должен быть отступ в 1 пробел. 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. Реализации самих функций должны быть в «.cpp» файлах.

     После constexpr методов, обычных методов, методов с префиксом static, следует оставлять пустую строку.
   - Данные класса (изменяемые и не изменяемые).
     - Переменные должны быть сгруппированы на const (идут первыми) и изменяемые.
     - В случае, если переменная const, то перед ключевым словом «const» требуется поставить один (1) tab. После чего один (1) пробел. Далее идет тип переменной. После типа переменной с помощью tab выравниваются все имена между собой (только между const переменными). В случае. ели переменная является указателем, символ указателя («*») воспринимается 1-м символом переменной и выравнивается как часть имени.

– В случае, если имеются изменяемые данные. Перед типом переменной следует выполнить один tab. После типа переменной с помощью tab выравниваются все имена между собой (только между const переменными). В случае. если переменная является указателем, символ указателя («*») воспринимается 1-м символом переменной и выравнивается как часть имени.

3.6.3 Enum class-ы

3.6.4 Комментарии/разделения