Стандартная библиотека периферии

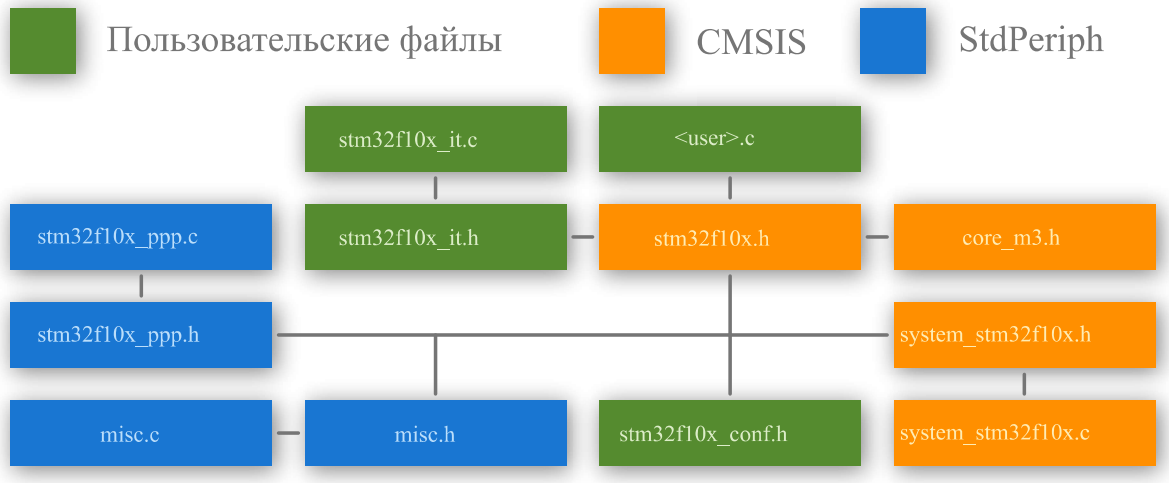
Библиотека CMSIS абстрагирует программиста от карты памяти микроконтроллера. Код получается эффективным, так как программист просит компилятор сделать только нужные вещи (записать какое-нибудь значение в нужное место). Проблема такого подхода в том, что нужно заглядывать в документацию, чтобы определить, в какой регистр и что нужно записать. У одного и того же производителя регистры на разных МК могут отличаться, как названием, так и количеством. Это неудобно.

Абстракция — мощный инструмент, которую легко реализовать. Вместо обращения к регистрам можно просто вызывать функции. И в CMSIS такая абстракция уже присутствует (совсем чуть-чуть).

```
NVIC_Enable(ADC1_2_IRQn);
// Вместо
NVIC->ISER[((uint32_t)(18) >> 5)] = (1 << ((uint32_t)(18) & 0x1F));
```

Если модуль несложный (те же порты ввода-вывода), то больших усилий для его инициализации прикладывать не нужно. Но вот если нужно настроить какой-нибудь таймер в нестандартный режим работы, то рутина по выставлению нужных битов в памяти принимает устрашающий характер. Стандартная библиотека периферии помогает заглядывать в документацию реже[^lib_error]. Всё, что должен сделать программист (в общем случае) — это заполнить структуру с читаемыми параметрами и выполнить функцию.

Стандартный проект будет включать в себя библиотеку CMSIS (она используется внутри StdPeriph), пользовательские файлы и файлы самой библиотеки.



Архив с библиотекой и примерами ее использования можно найти на странице целевого МК в разделе **Design Resources**. Каждому модулю периферии соответствует два файла: заголовочный (`stm32f10x_ppp.h`) и исходного кода (`stm32f10x_ppp.c`). Здесь `ppp` — название периферии. К примеру, для работы с аналого-цифровым преобразователем нужны файлы `stm32f10x_adc.h` и `stm32f10x_adc.c` . Файлы `misc.h` и `misc.c` реализуют работу с контроллером прерываний NVIC и системным таймером SysTick (эти функции есть в CMSIS).

Чтобы подключить стандартную библиотеку, нужно в файле `stm32f10x.h` определить макрос `USE_STDPERIPH_DRIVER` [^dont_change].

```
// stm32f10x.h
#ifdef USE_STDPERIPH_DRIVER
  #include "stm32f10x_conf.h"
#endif
```

Заголовочный файл `stm32f10x_conf.h` не является частью библиотеки, он пользовательский. С его помощью можно подключать или отключать части библиотеки.

```
#ifndef __STM32F10x_CONF_H
#define __STM32F10x_CONF_H

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x_adc.h"
#include "stm32f10x_bkp.h"
#include "stm32f10x_can.h"
// ...
```

Оставшиеся два файла (`stm32f10x_it.h` и `stm32f10x_it.c`) выделены для реализации обработчиков прерываний, именно туда следует помещать данные функции.

В стандартной библиотеке периферии есть соглашение о наименовании функций и обозначений.

- `PPP` — акроним для периферии, например, `ADC` .
- Системные, заголовочные файлы и файлы исходного кода начинаются с префикса `stm32f10x_` .
- Константы, используемые в одном файле, определены в этом файле.
- Константы, используемые в более чем одном файле, определены в заголовочных файлах. Все константы в библиотеке периферии чаще всего написаны в *ВЕРХНЕМ* регистре.
- Регистры рассматриваются как константы и именуются также *БОЛЬШИМИ* буквами.
- Имена функций, относящихся к определенной периферии, имеют префикс с ее названием, например, `USART_SendData()` .
- Для настройки каждого периферийного устройства используется структура `PPP_InitTypeDef` , которая передается в функцию `PPP_Init()` .
- Для деинициализации (установки значения по умолчанию) можно использовать функцию `PPP_DeInit()` .
- Функция, позволяющая включить или отключить периферию, именуется `PPP_Cmd()` .
- Функция включения/отключения прерывания именуется `PPP_ITConfig` .

С полным списком вы можете ознакомиться в файле поддержки библиотеки.

Перепишем инициализацию порта ввода-вывода из предыдущего раздела. Во-первых, нужно включить тактирование модуля (подать питание) — делается это через функцию, объявленную в `stm32f10x_rcc.h` :

```
void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph,
                            FunctionalState NewState);
```

Возможные варианты первого аргумента можно найти в комментарии к функции или в заголовочном файле. Так как мы работаем с портом `A` , нам нужен `RCC_APB2Periph_GPIOA` . Перечисление `FunctionalState` определено в `stm32f10x.h` :

```
typedef enum {DISABLE = 0, ENABLE = !DISABLE} FunctionalState;
```

Далее нужно обратиться к структуре порта из `stm32f10x_gpio.h`:

```
typedef struct {
  uint16_t GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
} GPIO_InitTypeDef;
```

Параметры структуры можно найти заголовочном файле.

```
// Включаем тактирование
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
// Объявляем структуру и заполняем её
GPIO_InitTypeDef gpio;
gpio.GPIO_Pin = GPIO_Pin_0;
gpio.GPIO_Speed = GPIO_Speed_2MHz;
gpio.GPIO_Mode = GPIO_Mode_Out_PP;
// Инициализируем порт
GPIO_Init(GPIOA, &gpio); // передаём указатель на структуру!
```

Главное — запомнить порядок инициализации: включаем тактирование периферии, объявляем структуру, заполняем структуру, вызываем функцию инициализации. Другие периферийные устройства обычно настраиваются по подобной схеме.

Назад | Оглавление | Дальше