Внутренняя flash-память

Встроенная flash-память может быть запрограммирована через ICP (in-circuit programming) или IAP (in-circuit programming). Первый предоставляется через разъём программирования JTAG/SWD, а второй, как не сложно догадаться, через приложение зашитое на МК. Цели программировать flash-память изнутри могут быть разными: от сохранения каких-либо настроек для последующих запусков, до перепрошивки устройства.

В данном случае, нас интересует именно сохранение настроек, например яркости экрана в дневное и ночное время [1].

Детально процесс работы с памятью описан в документе «**PM0075 Programming manual**». В нём, а также в Reference Manual, можно найти таблицу с описанием размера страницы и их количество.

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 Kbyte
	Page 1	0x0800 0400 - 0x0800 07FF	1 Kbyte
	Page 2	0x0800 0800 - 0x0800 0BFF	1 Kbyte
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 Kbyte
	Page 4	0x0800 1000 - 0x0800 13FF	1 Kbyte
	. . .	. . .	. . .
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 Kbyte
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

В зависимости от МК, размер страницы может быть разным и занимать, например, 2 Кб. Кроме того, все страницы объединены в блоки. В нашем случае он один, но в других моделях блоков может быть несколько.

Память разбита на три части.

- **Main memory block** (1 блок из 16 страниц по 1 КБайту для **stm32f103c8**) – память программ, та самая где хранится прошивка и куда можно записывать пользовательские данные.
- **Information block** — состоит из двух частей

  - *System memory* — компания ST прошивает в эту область загрузчик (англ. bootloader), позволяющий прошивать устройство через внешний интерфейс — USART1 (см. **AN2606**). Данная область защищена и пользователь не может получить к ней доступ.
  - *Option bytes* — область памяти, отвечающая за защиту памяти от чтения\записи. Можно защищать как отдельные страницы, так и всю память целиком. Мы поговорим об этом в следующем разделе.
- **Flash memory interface registers** — регистры отвечающие за работу с flash-памятью.

> Всеми операциями, будь то чтение, запись или блокировки управляет специальный контроллер Flash Program/Erase Controller (FPEC). Он отвечает за корректную работу с памятью. Особенность flash в stm32 заключается в том, что она работает не на той же частоте, что и ядро (у NXP, например, flash разгоняется до скорости ядра). Это позволяет делать МК немного дешевле, так как такая память дешевле/проще в производстве. В разделе про тактирование при настройке PLL мы использовали следующая строчка:
>
> ```
> FLASH_SetLatency(FLASH_Latency_2);
> FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
> ```
>
> Согласно документации, встроенная память может работать корректно без каких либо ухищрений вплоть до 24 МГц (`FLASH_Latency_0`). При более высоких частотах контроллер должен вставлять пустую операцию, что бы операция чтения\записи успела завершиться. Для частот от 24 до 48 МГц вставляется один такт задержки (`FLASH_Latency_1`), для частот от 48 до 72 МГц два такта (`FLASH_Latency_2`).

Как же определить свободную страницу? Это зависит от размера прошивки. Проще всего использовать последнюю страницу в памяти (вы вряд ли умудритесь заполнить все 64 Кб). Мы, однако, для примера возьмём 16-тую страницу. Её адрес можно вычислить как [начальный адрес] + [номер страницы - 1] * [размер страницы].

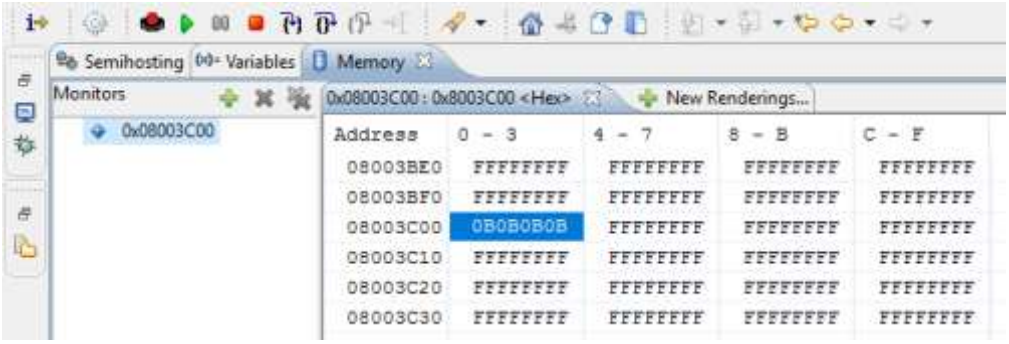
```
0x0800 0000 + 15 * 0x0000 0400 = 0x0800 3C00
```

Определим её в коде:

```
#define SETTINGS_PAGE 0x08003C00
```

Особенность flash такова, что для перезаписи данных **необходимо стереть всю страницу**. Поэтому перед операцией записи скопируйте все ваши данные в буфер, затем внесите в него изменения, а только потом приступайте к записи непосредственно на flash. Пример с использованием стандартной библиотеки периферии.

```
uint32_t data = 0x0B0B0B0B;
// ...
FLASH_Unlock();
FLASH_ErasePage(SETTINGS_PAGE);
FLASH_ProgramWord(SETTINGS_PAGE, data);
FLASH_Lock();
```



Для удобства работы с настройками можно создать структуру. Сделайте это самостоятельно.

Считать данные можно написав вот такую функцию:

```
uint32_t flash_read(u32 address) {
    return *((u32 *)address);
}
```

У flash-памяти есть лимит по количеству циклов перезаписи, 100 000 для **stm32f1**, после которого производитель не гарантирует сохранность данных. Вам правда придётся стирать и записывать в память примерно по 275 раз каждый день в течении года, чтобы достигнуть такого состояния. Тем не менее, в качестве тренировки вы можете попробовать написать модуль, который будите использовать в дальнейших проектах, где "распределение" нагрузки производится равномерно. Предположим у нас имеется следующая структура:

```
typedef struct {
    uint32_t a;
    uint32_t b;
    uint32_t c;
    uint32_t d;
} SETTINGS_t;
```

В памяти она занимает 16 байт (4 раза по 4 байта). По описанному выше принципу, все их можно сложить в памяти последовательно. При внесении изменений новые данные можно положить в следующие 16 байт, а не стирать каждый раз страницу. Далее нужно только определить, по какому адресу лежат актуальные настройки.

Назад | **Оглавление** | **Дальше**

---

1. В некоторых микроконтроллерах, например, AVR, или даже STM32L1-серии, присутствует модуль памяти EEPROM, которую можно использовать для сохранения пользовательских данных. Однако, в STM32F1-серии такого модуля нет. ↩