**Алгоритм работы загрузчика STM32\_I2C\_BootLoader**

Переход в main()  
(address = 0x0800 0004)

Reset\_Handler  
(0x0800 0000)

Запрет прерываний  
(\_\_disable\_irq())

Разрешение прерываний  
(\_\_enable\_irq())

Индикация запуска загрузчика (мигнем три раза светодиодом)

Подсчет   
размера приложения  
(в байтах)

Инициализация:  
Clock\_Init();  
GPIO\_Init();  
DELAY\_Init();  
BOOT\_Init();

Размер приложения > 0?

Нет

Да

CRC верно?

Расчет CRC   
приложения

Да

Нет

По I2C принят BOOT\_I2C\_ADDR?

Нет

Да

Истек таймаут  
(1 сек.)?

Да

Нет

Переход на Reset\_Handler  
 приложения  
(address = 0x0800 2800+4)

Инициализация сторожевого таймера (период 3 сек.)

Сброс сторожевого  
 таймера

Основной цикл загрузчика BOOT\_Loop()

**Параметры загрузчика:**

- интерфейс - I2C (I2C1 или I2C зависит от устройства);

- скорость - 400кГц;

- BOOT\_I2C\_ADDR - адрес на шине I2C - зависит от устройства (пока адрес 0b0111000 (0x38));

- переход в режим загрузчика - при старте МК первым запускается загрузчик;

- начальный адрес - 0х0800 0000;

- размер - 10КБ (10240 байт = 0х2800) (из них 9КБ — это сам загрузчик, 1КБ — это область хранения метаданных приложения)

**Параметры приложения:**

- начальный адрес = 0х0800 0000 + 0х2800 = 0х0800 2800

- размер = размер флэш-памяти МК минус размер загрузчика (10240 байт = 0х2800)

**Условие перехода из загрузчика в приложение** (если оно есть)**:**

- таймаут (1 сек.);

- по команде (команда Go загрузчика).

**Условие перехода из приложения в загрузчик:**- установка флага DFU\_REQUESTED\_FLAG в разделяемой памяти ОЗУ и системный сброс микроконтроллера функцией NVIC\_SystemReset().

**Порядок прошивки микроконтроллера**

1) Очистить ВСЮ флэш-память МК с помощью программатора и утилиты ST-Link (команда Erase Chip);

2) Прошить HEX-файл загрузчика с помощью программатора и утилиты ST-Link;   
3) Прошить HEX-файл основного приложения с помощью программатора и утилиты ST-Link.

После этих манипуляций и после сброса МК, первым запустится загрузчик. Он увидит, что есть основное приложение и передаст управление ему. Так же будет доступна прошивка внутренней флэш-памяти МК через загрузчик.

Так же можно пропустить шаг №3 при прошивке МК. В этом случае после сброса МК первым запустится загрузчик, он увидит, что основного приложения нет и будет ожидать прихода команд по шине I2C. Подробное описание команд смотреть в файле AN4221\_I2C protocol used in the STM32 bootloader\_Перевод.pdf

**Общие мысли**

У нас есть один целевой микроконтроллер и в его флеш-память записывается ДВЕ прошивки. Одна прошивка – это загрузчик, втора – это основное приложение. Т.е. создаются два независимых проекта. При создании этих независимых проектов необходимо настроить линковщики для каждого из проектов.

**Настройка линковщика для сборки проекта загрузчика**

При включении питания МК всегда стартует с начала флэш памяти, а именно с адреса 0х08000000. По этому адресу у нас расположен загрузчик. Размер загрузчика 10КБ (10240 байт = 0х2800) (из них 9КБ — это сам загрузчик, 1КБ — это область хранения состояния приложения).

Настройка линковщика сводится к указанию размера области флеш-памяти, в которую будет записан загрузчик. Делается это следующим образом:  
  
**Все что описывается ниже, справедливо для Cube IDE 1.6.1**  
- открываем файл линковщика STM32F103C8TX\_FLASH.ld  
- в нем находим следующие строки:  
  
**/\* Memories definition \*/**

**MEMORY**

**{**

**RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 20K**

**FLASH (rx) : ORIGIN = 0x8000000, LENGTH = 64K**

**}**

- модифицируем данные строчки следующим образом:

**/\* Memories definition \*/**

**MEMORY**

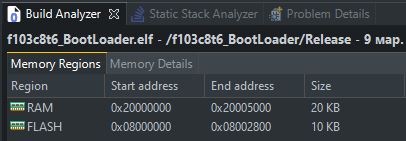
**{**

**RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 20K**

**FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 10K /\* 64K \*/**

**}**

После чего сохраняем файл линковщика и собираем проект. После сборки проекта во вкладке «Build Analyzer» Cube IDE можно будет увидеть результат наших манипуляций с файлом STM32F103C8TX\_FLASH.ld. Конечный адрес (End address) FLASH должен измениться с 0x08010000 на 0x08002800, а размер (Size) FLASH измениться с 64KB на 10KB. На рисунке ниже представлен пример вкладки «Build Analyzer».



**Метаданные приложения**

Образ основного приложения состоит из метаданных и, собственно, самого основного приложения.

Подробнее почитать о концепции метаданных приложения можно тут **https://interrupt.memfault.com/blog/device-firmware-update-cookbook#design-patterns--recipes**  
 Что из себя представляют метаданные? Это структура данных, которая располагается во флэш-памяти микроконтроллера, начиная с адреса 0x08002400 и имеет размер 1024 байта. Размер метаданных выбран равным размеру стираемой страницы флэш-памяти микроконтроллера для того, чтобы из приложения можно было производить модификацию метаданных (возможно это понадобится в будущем).   
Структура метаданных приложения:

**typedef** **struct** **\_\_attribute\_\_**((packed)){

uint16\_t metadataVersion; //версия методанных

//"магическое" число обычно представляет собой константу, которая используется

//для идентификации формата файла или структуры данных.

//В этом случае каждое изображение должно иметь байты 0xcafe в первых двух байтах.

//Если число отличается - образ недействителен!

uint16\_t appMagic; //"магическое" число

uint8\_t appType; //???

uint32\_t appVectorAddr; //стартовый адрес приложения

uint32\_t appSize; //размер bin-файла приложения в байтах

uint32\_t appCrc; //контрольная сумма bin-файла приложения

uint8\_t appVersion\_major; //

uint8\_t appVersion\_minor; //

uint8\_t appVersion\_patch; //

**char** git\_sha[8]; //хеш гита

uint32\_t reserved; //

}appMetadata\_t;

В проекте приложения нужно заполнить часть полей структуры метаданных вручную. К таким полям относятся: metadataVersion, appMagic, appType, appVectorAddr, appVersion.

Оставшиеся поля должны заполняться автоматически. Это не реализовано!!!

Пример заполнения метаданных приложения представлен ниже:

appMetadata\_t appMetadata **\_\_attribute\_\_**((section(".app\_metadata"))) = {

.metadataVersion = 0x1234,

.appMagic = METADATA\_MAGIC, //константа 0xcafe

.appType = 0x56,

.appVectorAddr = 0x08002800, //вектор сброса приложения

.appVersion\_major = 0x07,

.appVersion\_minor = 0x08,

.appVersion\_patch = 0x09,

// GIT\_SHA is generated by Makefile

.git\_sha = "12345678",

// populated as part of a post compilation step

.appSize = 0x12345678,

.appCrc = 0xabcdefde,

};

Для размещения метаданных в нужном месте флеш-памяти микроконтроллера используется атрибут .section для помещение структуры appMetadata в секцию app\_metadata. Адрес и размер секции определен в скрипте линковщика приложения stm32f103\_memory\_map\_for\_boot.ld.

**Разделяемая память ОЗУ**

Подробнее почитать о разделяемой между загрузчиком и приложением памяти ОЗУ можно тут [**https://interrupt.memfault.com/blog/device-firmware-update-cookbook#design-patterns--recipes**](https://interrupt.memfault.com/blog/device-firmware-update-cookbook#design-patterns--recipes)

Общая идея такая: приложение может захотеть сообщить загрузчику, что оно должно перейти в режим DFU (device firmware update) при следующей перезагрузке, а загрузчик может захотеть передать аргументы приложению. Для обмена информацией между приложением и загрузчиком используется область памяти в ОЗУ. ОЗУ сохраняет свое состояние пока микроконтроллер остается включенным.

**Настройка линковщика для сборки проекта основного приложения**

Со стартовым адресом и размером загрузчика разобрались. Далее нужно вычислить начальный адрес основного приложения, который зависит от размера загрузчика. Для того чтобы вычислить начальный адрес основного приложение надо к 0х08000000 прибавить размер загрузчика. В нашем случае размер загрузчика равен 10КБ (или 10 \* 1024 = 10240 байт = 0х2800). Следовательно, основное приложение должно начинаться с адреса 0x08002800.

Вот именно этот адрес 0x08002800 нужно указать линкеру, как адрес начала основного приложения, а также указать размер флеш-памяти доступное для основного приложения (флеш-память МК минус размер загрузчика). Делается это следующим образом:  
  
**Все что описывается ниже, справедливо для Cube IDE 1.6.1**  
- открываем файл линковщика STM32F103C8TX\_FLASH.ld  
- в нем находим следующие строки:  
  
**/\* Memories definition \*/**

**MEMORY**

**{**

**RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 20K**

**FLASH (rx) : ORIGIN = 0x8000000, LENGTH = 64K**

**}**

- модифицируем данные строчки следующим образом:

**/\* Memories definition \*/**

**MEMORY**

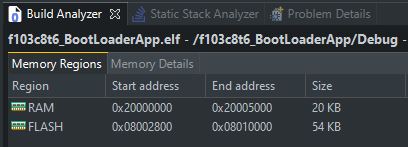
**{**

**RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 20K**

**FLASH (rx) : ORIGIN =(0x08000000 + 1024 \* 10), LENGTH = 64K-10K**

**}**

После чего сохраняем файл линковщика и собираем проект. После сборки проекта во вкладке «Build Analyzer» Cube IDE можно будет увидеть результат наших манипуляций с файлом STM32F103C8TX\_FLASH.ld. Начальный адрес (Start address) FLASH должен измениться с 0x08000000 на 0x08002800, а размер (Size) FLASH измениться с 64KB на 54KB. На рисунке ниже представлен пример вкладки «Build Analyzer».



**Перенос векторов прерываний при переходе в основную программу**

Дело в том, что в загрузчике таблица векторов прерываний располагается по одному адресу (0x08000000), а в основной программе по-другому (0x08002800). У них не может быть общей таблицы векторов прерываний, так как это две отдельные программы, и они отдельно компилируются. Для того, чтобы в основном приложении мы могли использовать прерывания, необходимо сразу же в начале функции main() приложения вписать две команды:

\_\_disable\_irq(); //Запрещаем прерывания

SCB->VTOR = 0x08000000 + 0x2800; //переносим таблицу векторов прерываний

**Команды загрузчика**

Подробное описание команд смотреть в документе

AN4221\_STM32\_Загрузчик\_I2C\_Перевод\_Основной.pdf

//Реализованные команды

CMD\_BOOT\_Get 0x00

CMD\_BOOT\_GetVersion 0x01

CMD\_BOOT\_GetID 0x02

CMD\_BOOT\_RM 0x11

CMD\_BOOT\_GO 0x21

CMD\_BOOT\_WM 0x31

CMD\_BOOT\_ERASE 0x44

CMD\_BOOT\_RP 0x82

CMD\_BOOT\_RUP 0x92

CMD\_BOOT\_NS\_GetCheckSum 0xA1

//Нереализованные команды

CMD\_BOOT\_NS\_WM 0x32

CMD\_BOOT\_NS\_Erase 0x45

CMD\_BOOT\_Special 0x50

CMD\_BOOT\_ExtSpecial 0x51

CMD\_BOOT\_WP 0x63

CMD\_BOOT\_NS\_WP 0x64

CMD\_BOOT\_WUP 0x73

CMD\_BOOT\_NS\_WUP 0x74

CMD\_BOOT\_NS\_RP 0x83

CMD\_BOOT\_NS\_RUP 0x93