BOT

Реализация загрузчиков кода (Bootloader).

MICROCHIP MASTERs 2012

Темы

- **Концепция загрузчиков**
- **Управление размещением в памяти**
  - Работа с Linker Script
  - Особенности PIC32
- **Особенности разработки загрузчиков**
  - Обработка прерываний
  - Каналы связи
  - Работа с Flash
  - Работа с регистрами конфигурации
- **Связь между загрузчиком и приложением**
  - Механизм динамической загрузки
- **Управление доступом**
  - Особенности системы разграничения доступа CodeGuard
- **С чего начать ?**

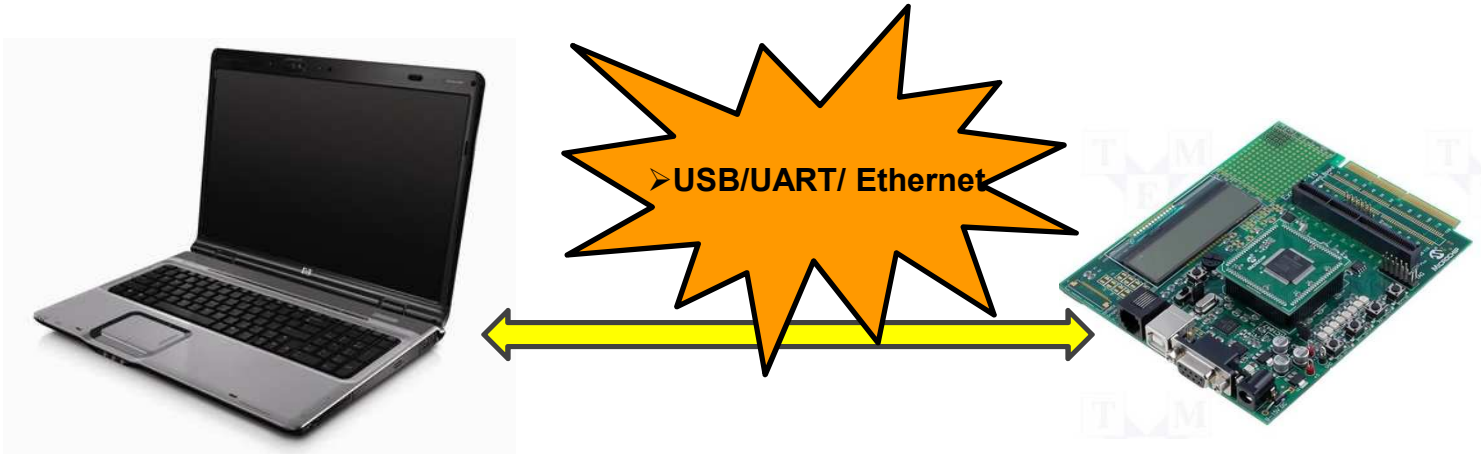
Зачем нужен загрузчик ?

- **Загрузчик** – это часть кода, выполняемая сразу же после сброса и предназначенная, при необходимости, изменять память микроконтроллера
- **Выполняемые задачи:**
  - Загрузка/обновление ПО
  - Проверка целостности приложения
  - Запуск приложения

Программирование без загрузчика

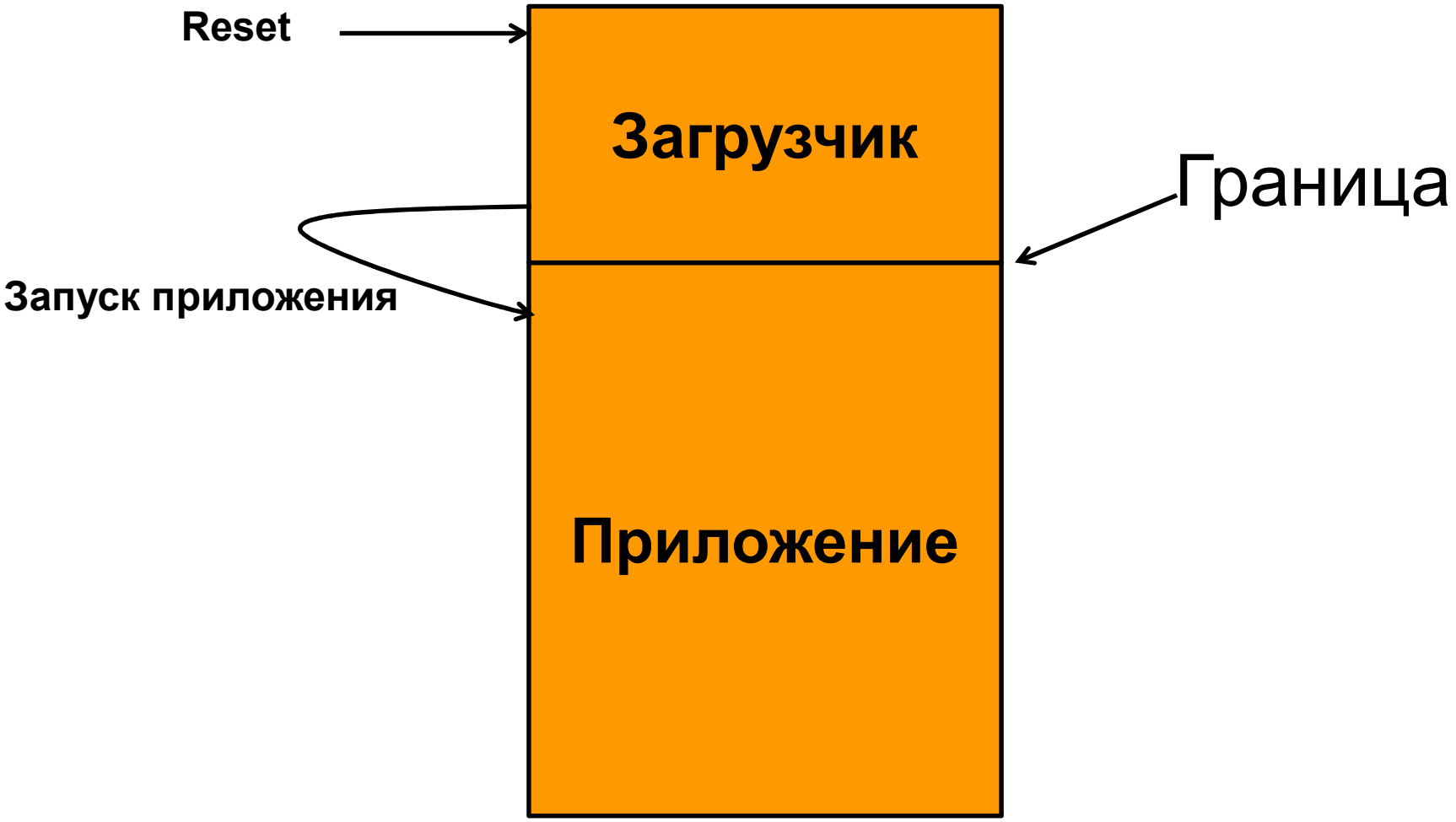
- Требуется квалифицированный персонал
- Требуется дорогой программатор
- Требуется дополнительное ПО
- Файл программы доступен оператору в открытом виде

➢Программирование с загрузчиком

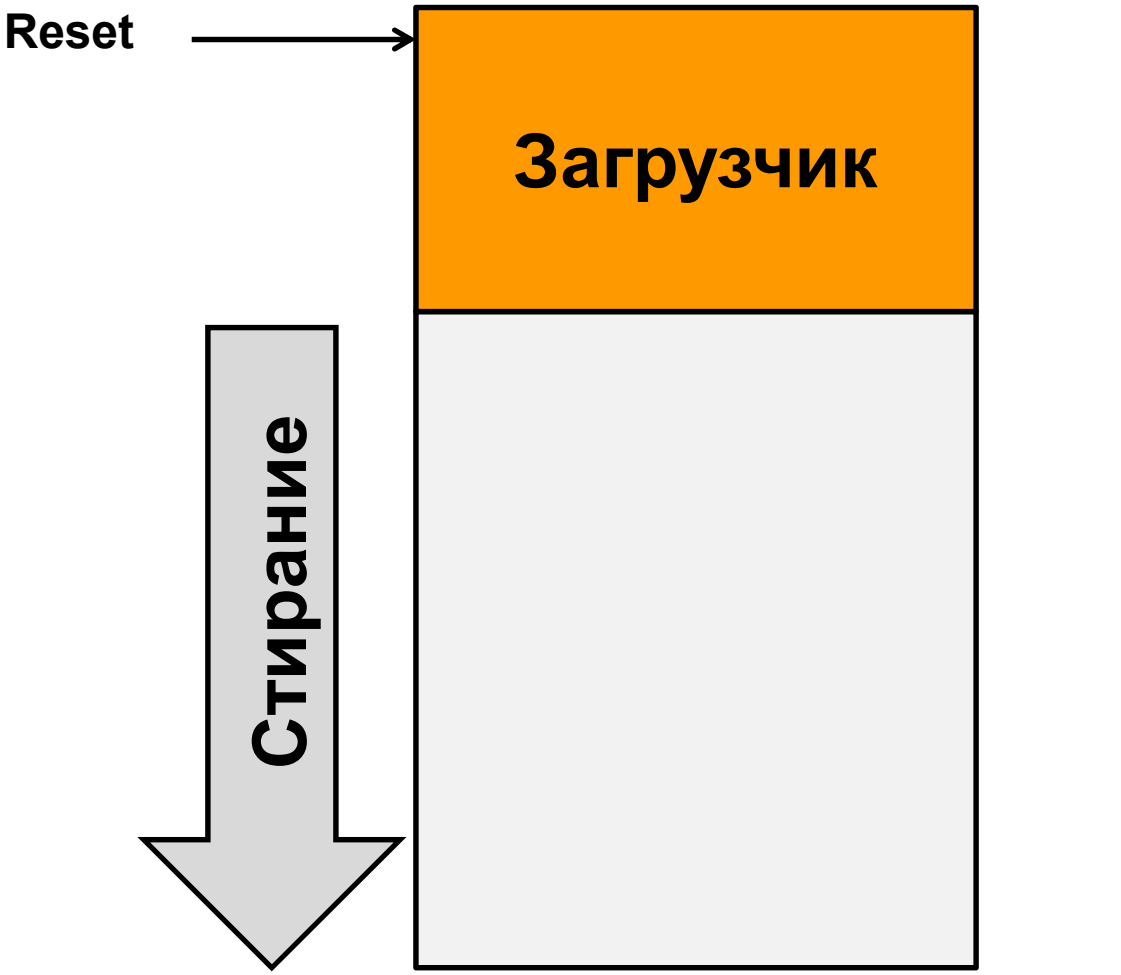


- Обновление прошивки самостоятельно
- Не требуется дополнительного оборудования
- Простое приложения для PC
- Упрощает обслуживание проданных изделий
- Файл программы может быть закодирован

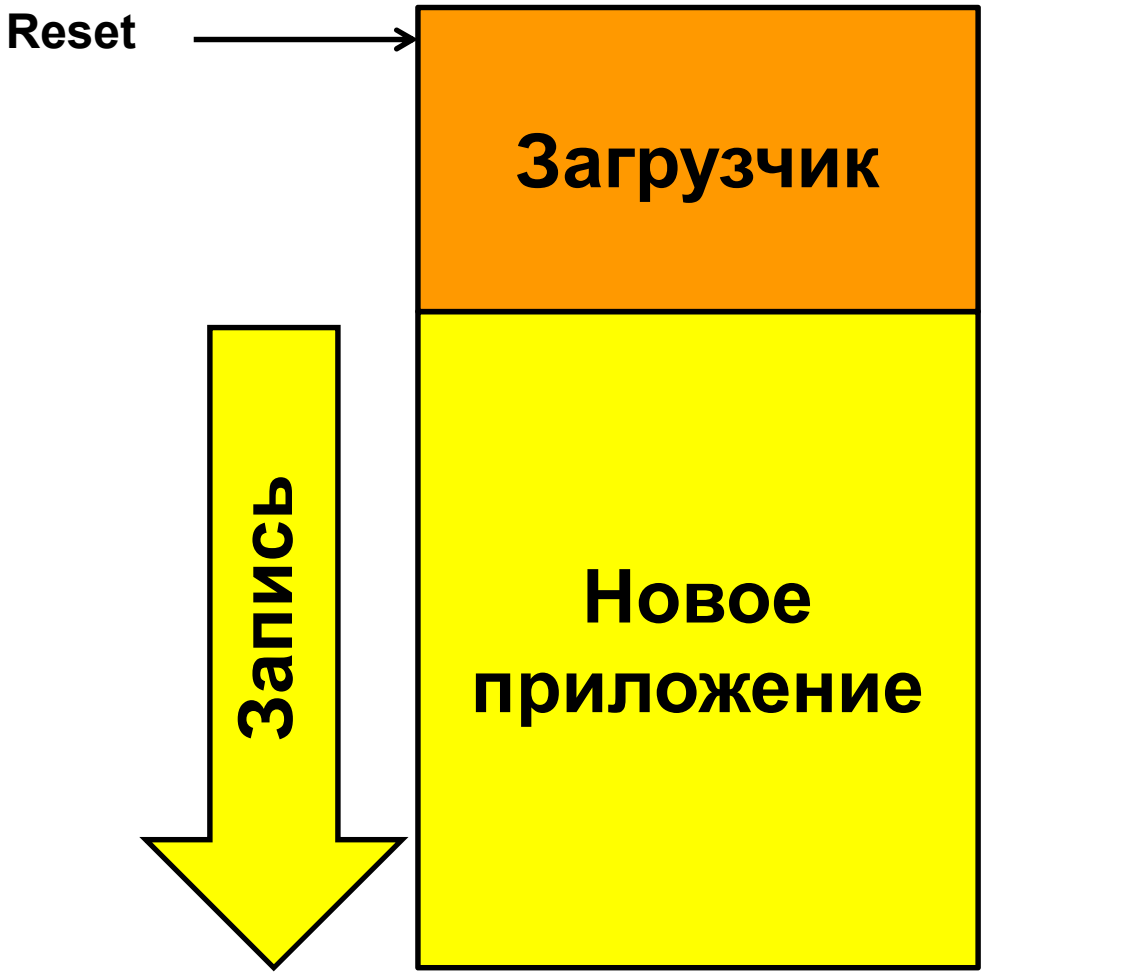
Концепция


Reset
Загрузчик
Граница
Запуск приложения
Приложение
Flash Program Memory

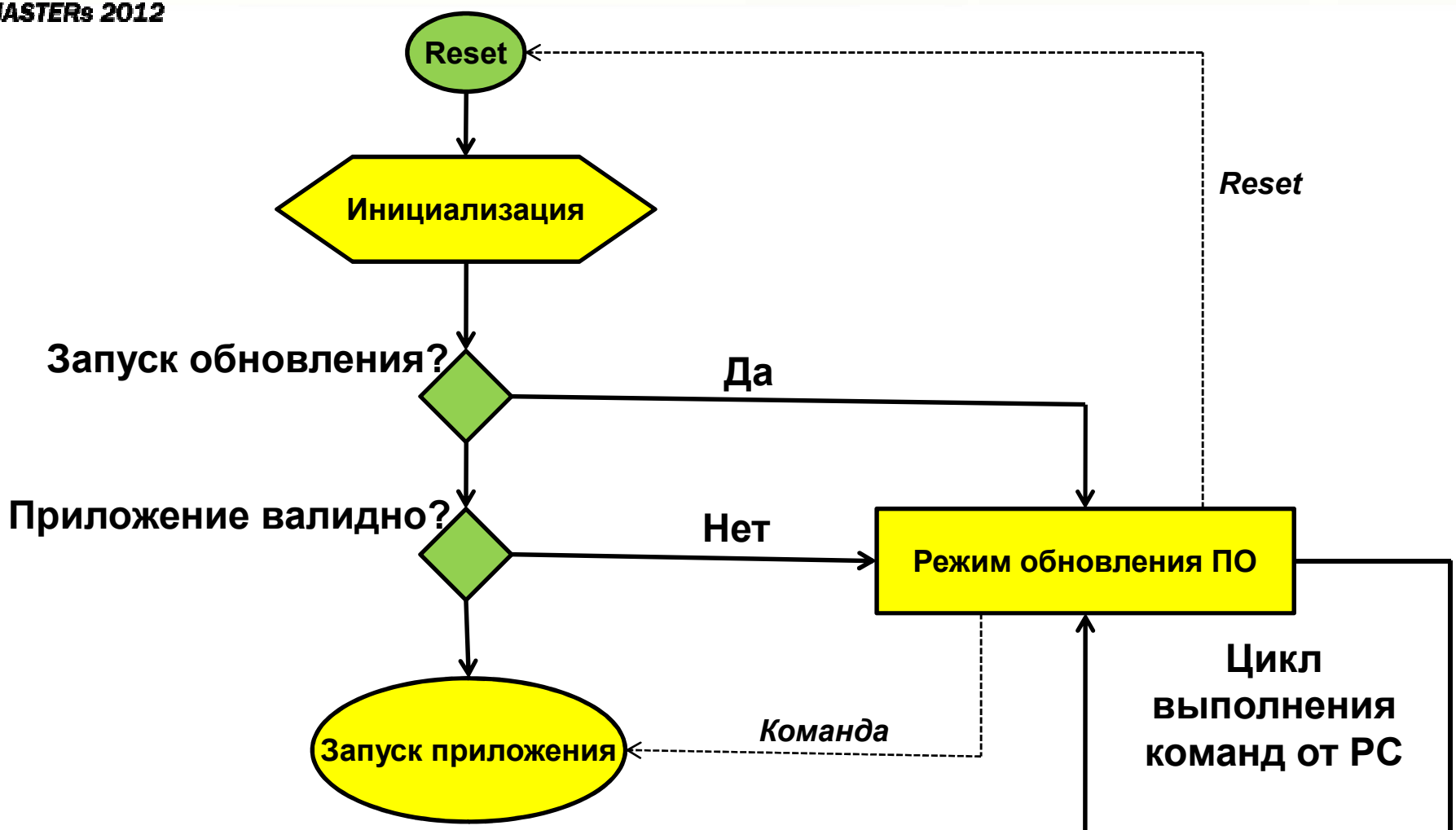
Концепция



Концепция


Reset
Загрузчик
Запись
Новое приложение
Flash Program Memory

Блок-схема


Reset
Инициализация
Запуск обновления?
Да
Приложение валидно?
Нет
Режим обновления ПО
Цикл выполнения команд от PC
Запуск приложения
Команда
Reset

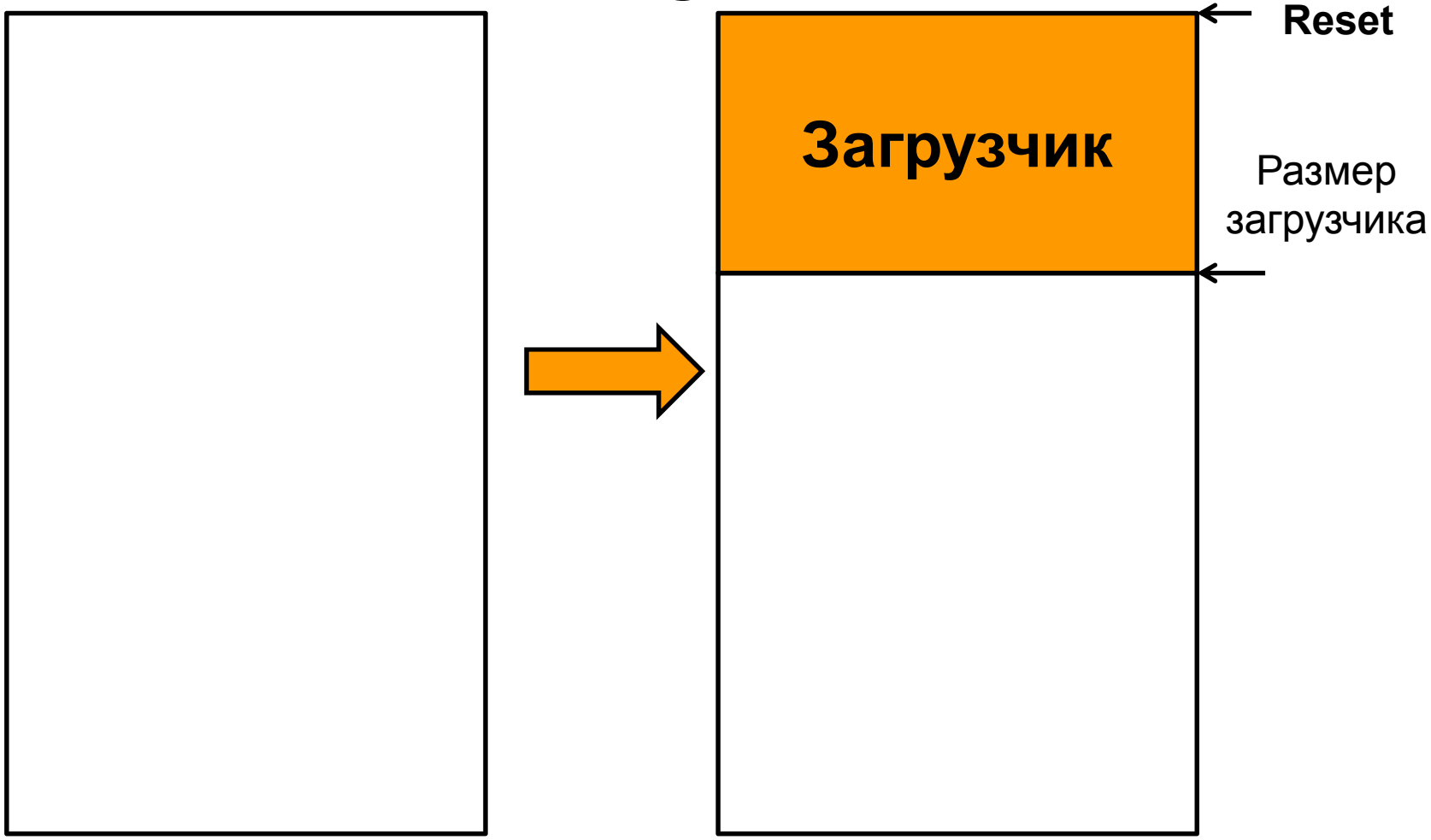
Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

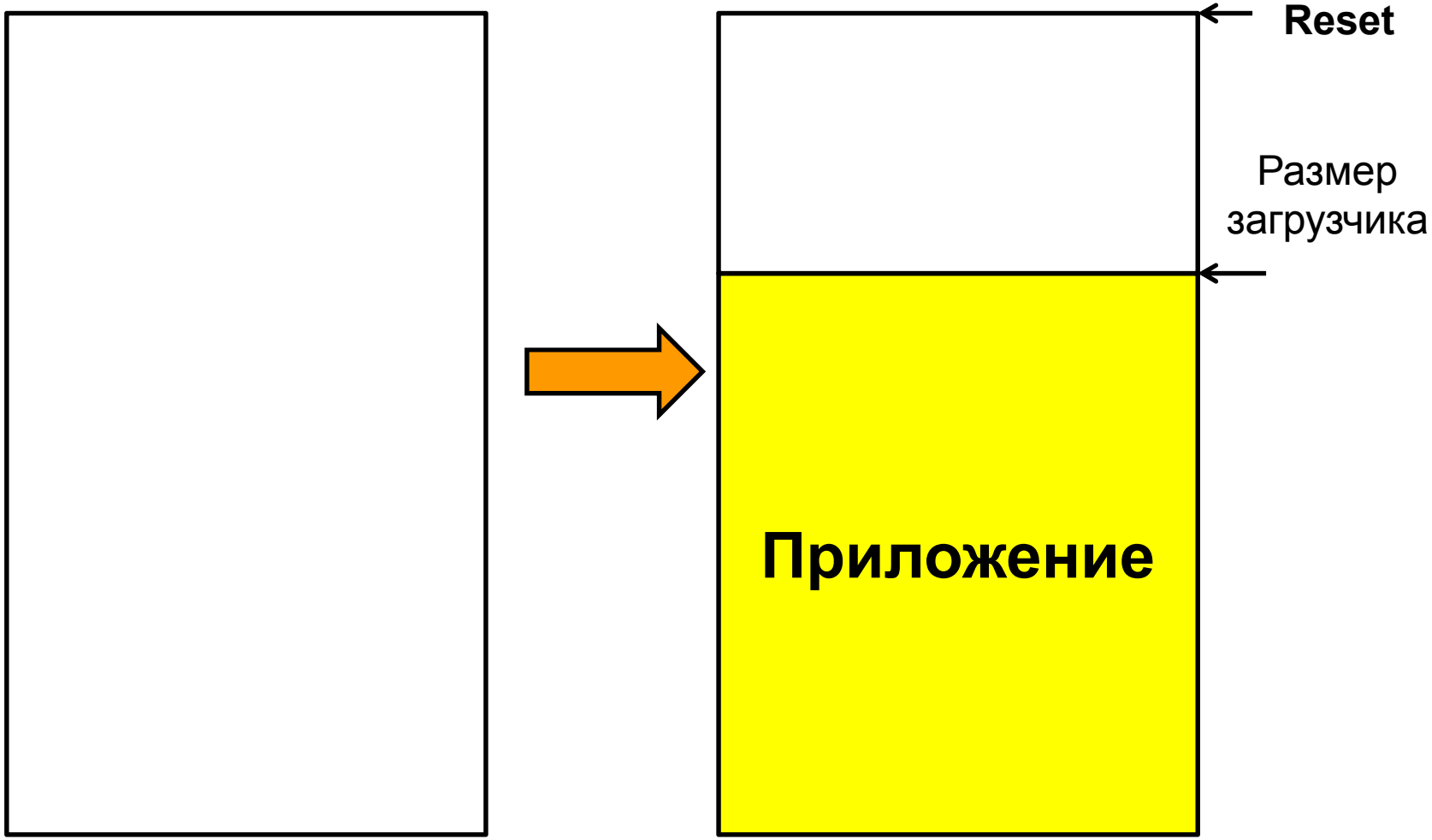
Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- **Сегментировать память для обоих проектов**
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Сегмент Flash в проекте загрузчика

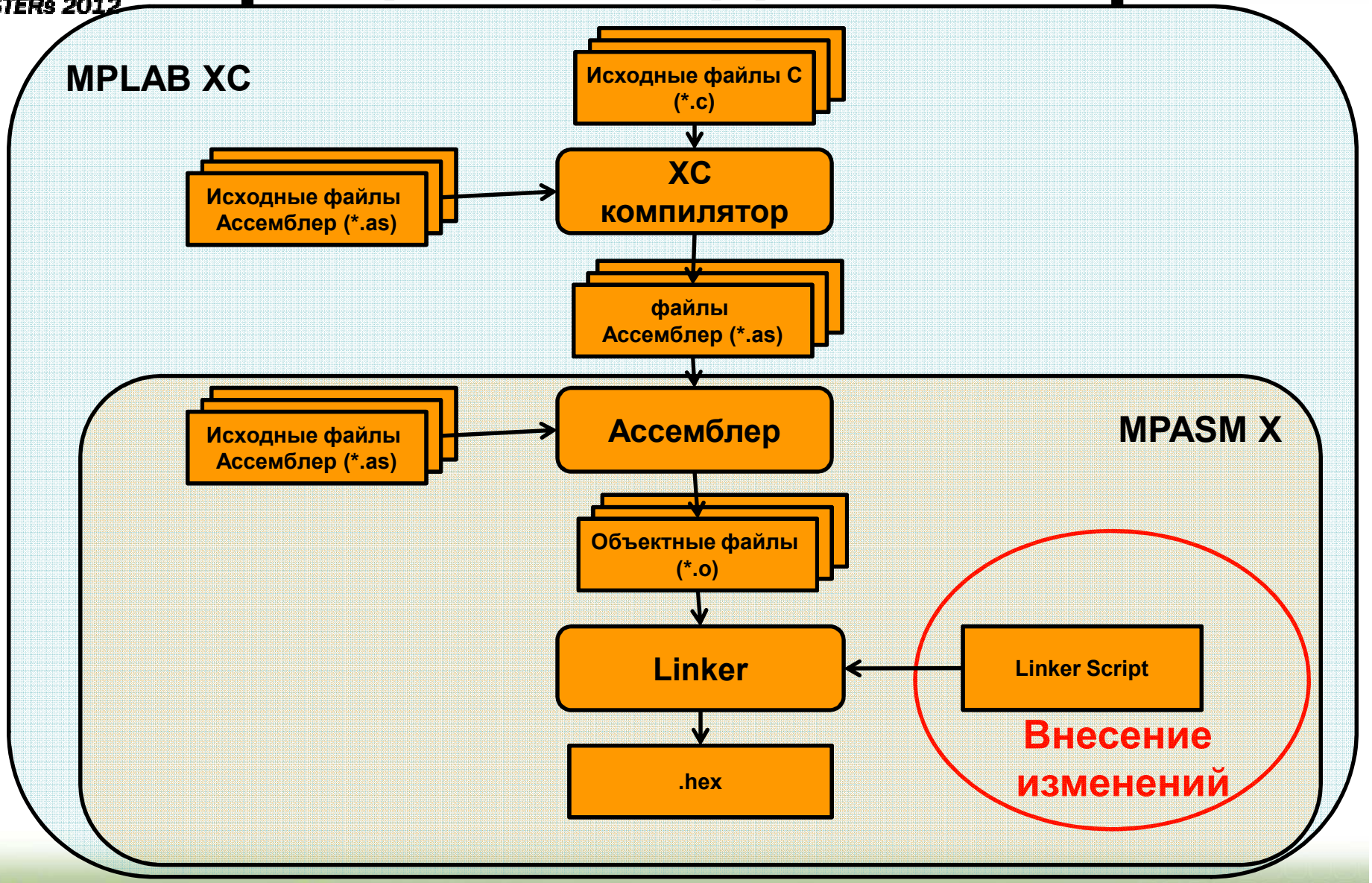


Сегмент Flash в проекте приложения



Flash Program Memory

Процесс создания образа



Как работает Linker

- Linker размещает элементы программы в памяти:
  - **Вся память микроконтроллера делится на классы**
  - **Все элементы программы распределяются по секциям**
  - **Linker размещает секции по классам**
  - **Управляя Linker-ом можно размещать нужные элементы программ в нужных областях памяти**

Классы памяти

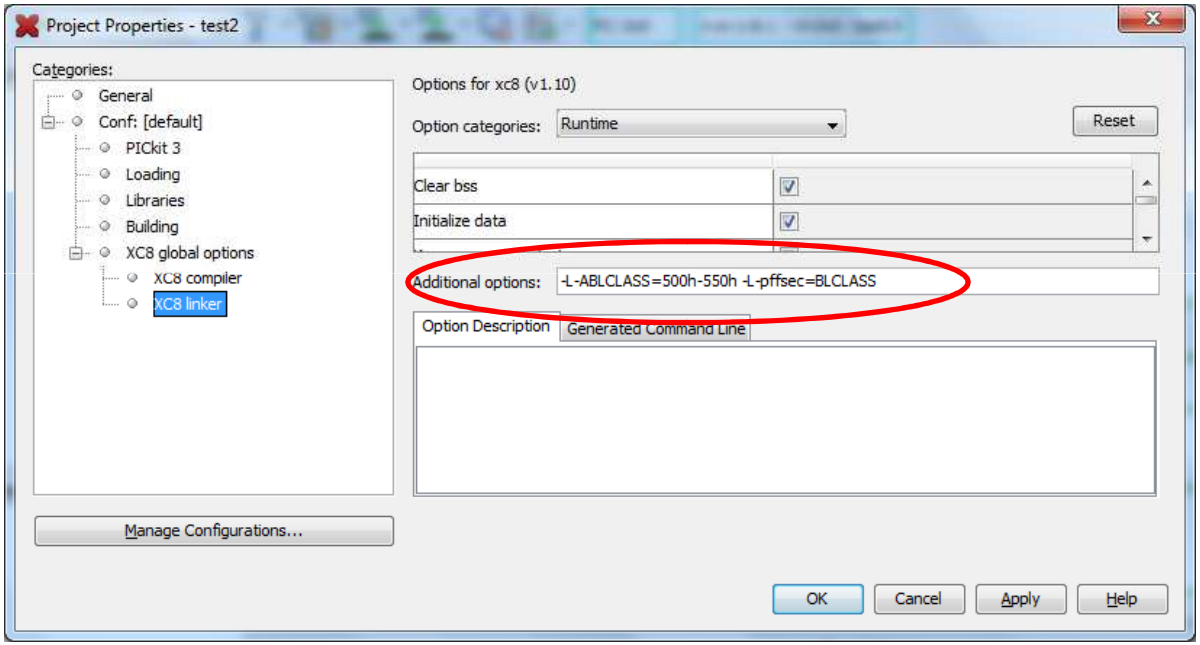
- Память микроконтроллера делиться на области (классы), характеризуемые следующими параметрами:
  - **Имя класса**
  - **Тип памяти (ROM, RAM, EEPROM)**
  - **Диапазон адресов**
  - **Отличительные признаки**

Классы памяти

- Классы памяти специфичны для каждого микроконтроллера и задаются:
  - **XC8 – в опциях командной строки linker-a**
  - **XC16 – в linker-script файле .gld**
  - **XC32 – в linker-script файле .ld**
  - **MPASMX - в linker-script файле .lkr**

Классы памяти в XC8

- Задается в опциях командной строки linker-а




```
-L-AMY_CLASS=0100h-01ffh
```

Классы памяти в XC16

- Задается в linker-script файле .gld (включается в проект)

```
MEMORY
{
  data  (a!xr) : ORIGIN = 0x800,  LENGTH = 0x7800
  reset        : ORIGIN = 0x0,    LENGTH = 0x4
  ivt          : ORIGIN = 0x4,    LENGTH = 0xFC
  aivt         : ORIGIN = 0x104,  LENGTH = 0xFC
  MY_CLASS (xr): ORIGIN = 0xc00,  LENGTH = 0x100
  program (xr) : ORIGIN = 0xd00,  LENGTH = 0x2000
  CONFIG1      : ORIGIN = 0xF80000, LENGTH = 0x2
                 . . .
}
```

Классы памяти в XC32

- Задается в linker-script файле .ld (включается в проект)

```
MEMORY
{
MY_CLASS   (rx) : ORIGIN = 0x9D000000, LENGTH = 0x100
kseg0_p_m  (rx) : ORIGIN = 0x9D000100, LENGTH = 0x7FF00
kseg0_b_m       : ORIGIN = 0x9FC00490, LENGTH = 0x970
exception_m     : ORIGIN = 0x9FC01000, LENGTH = 0x1000
sfrs            : ORIGIN = 0xBF800000, LENGTH = 0x100000
kseg1_b_m       : ORIGIN = 0xBFC00000, LENGTH = 0x490
debug_e_m       : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
            . . .
}
```

Секции

- При компиляции все элементы программы (переменные, константы, функции) распределяются по секциям.
- Названия секций компилятор задает по умолчанию (разные компиляторы по разному)
- Можно указать компилятору, чтобы он разместил любой элементы программы в секции с заданным именем

Указание имени секции

Компиляторы XC

- При CCI используется модификатор __section

```
int __section(“my_v_section”) myVar;
void __section("my_f_section")  myFunc( void)
{
     . . .
}
```

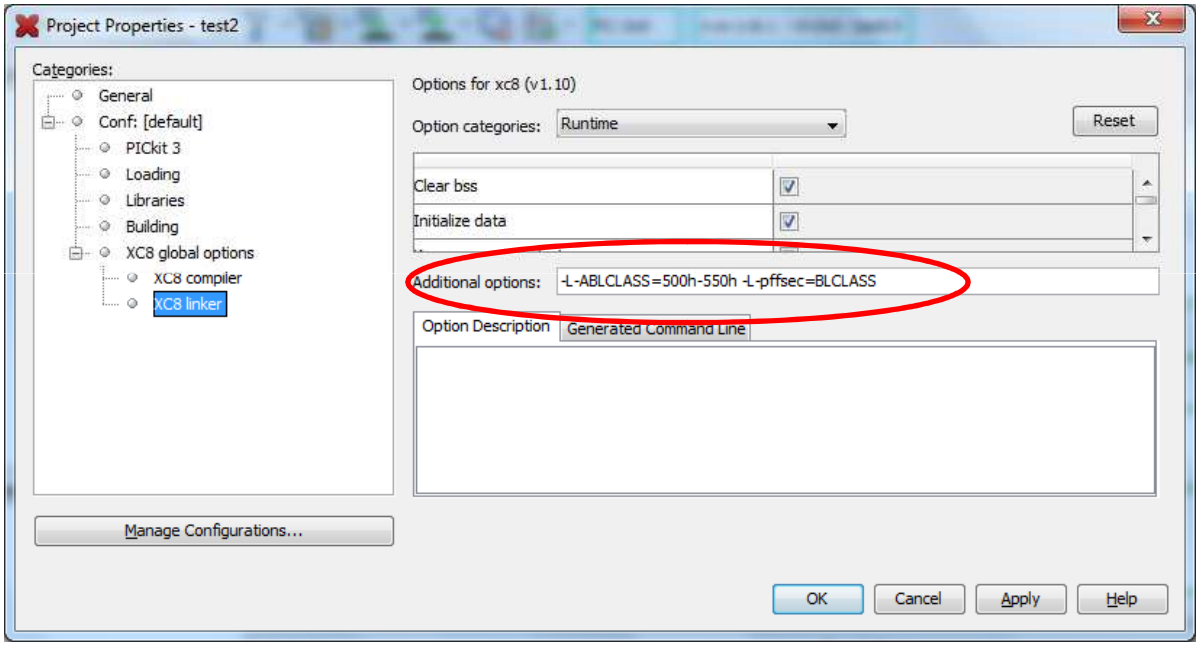
Ассемблер MPASMX

- Используется директива code

```
MY_SECTION_NAME  code
     . . .
```

Размещение секции XC8

- Задается в опциях командной строки linker-а




```
-L-Pmy_f_section=MY_CLASS
```

Размещение секции XC16/XC32

- Задается в linker-script файле .gld/.ld (включается в проект)

```
SECTIONS
{

 my_f_section :
  {
    *(my_f_section);
  } > MY_CLASS

      . . .
```

Способы задания

Компиляторы XC

- При CCI используется модификатор __section

```
int __section(“my_v_section”) myVar;
void __section("my_f_section")  myFunc( void)
{
     . . .
}
```

Ассемблер MPASMX

- Используется директива code

```
MY_SECTION_NAME  code
     . . .
```

Размещение элементов

- Способы размещения элементов памяти по абсолютному адресу:
  - **Задание адреса элемента в исходном тексте**
  - **Размещение секции элемента в специальном классе памяти**
  - **Задание расположения класса по умолчанию**

Задание адреса элемента в исходном тексте

- Для XC компиляторов с CCI используется модификатор __at(address)

```
void __at(0x1ff)  myFunc( void) {
     . . .
}
```

- Для MPASMX используется директива code с указанием адреса

```
MY_SECTION_NAME  code    1ffh
     . . .
```

- Линкер автоматически создает секцию и размещает ее по заданному адресу

Размещение секции в специальном классе памяти

- Задается специальный класс памяти по заданному адресу

```
MY_CLASS (xr): ORIGIN = 0xc00,  LENGTH = 0x100
```

- Задается специальная секция, которая

```
my_sect :
  {
    *(my_sect);
  } > MY_CLASS
```

- Все функции, которые требуется разместить в сегменте, привязываются к специальной секции

```
void __section(“my_sect") myFunc(void)
```

Задание расположения класса по умолчанию

- При отсутствии директив линкер размещает элементы программ в известных по умолчанию секциях. (.text, .idata , …)
- При отсутствии в проекте Linker script-а, используется скрипт по умолчанию. Этот скрипт размещает секции в определенных классах памяти.
- Для изменения положения элементов программ достаточно изменить положение класса памяти по умолчанию через Linker Script

```
kseg0_p_m (rx) : ORIGIN = 0x9D000000, LENGTH = 0x80000
```

```
kseg0_p_m (rx) : ORIGIN = 0x9D001000, LENGTH = 0x7F000
```

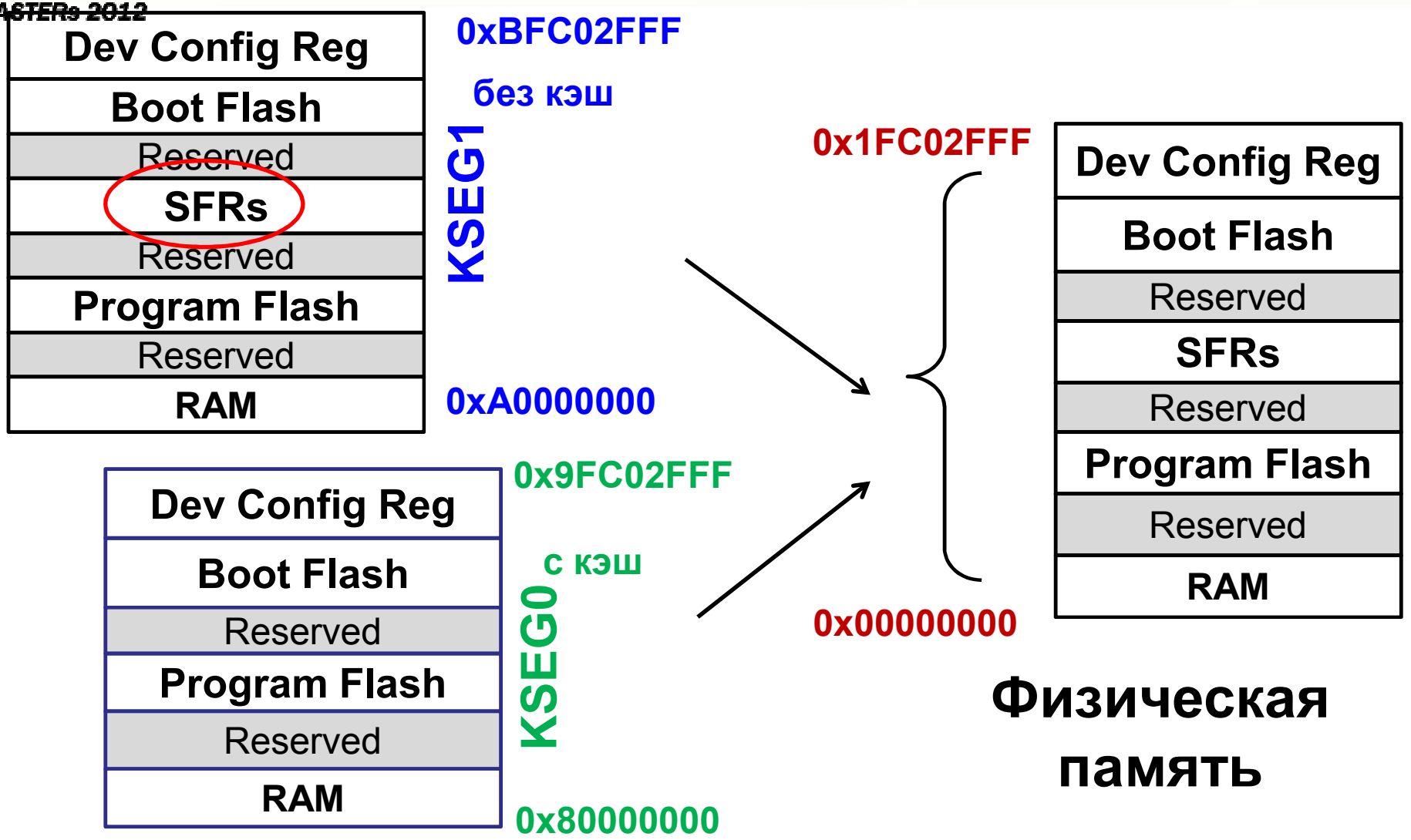
Задание расположения класса по умолчанию

- Наиболее удобный способ для загрузчиков и для приложений, работающих с загрузчиками
- Позволяет изменять приложения, для автономной работы на приложение для загрузчика изменением Linker Script

Особенности PIC32

- Наличие специальной области Boot Flash
- Отображение физической памяти на виртуальную
- Кэшированное/не кэшированное выполнение операций
- Разграничение доступа Ядро/Пользователь

Архитектура памяти PIC32

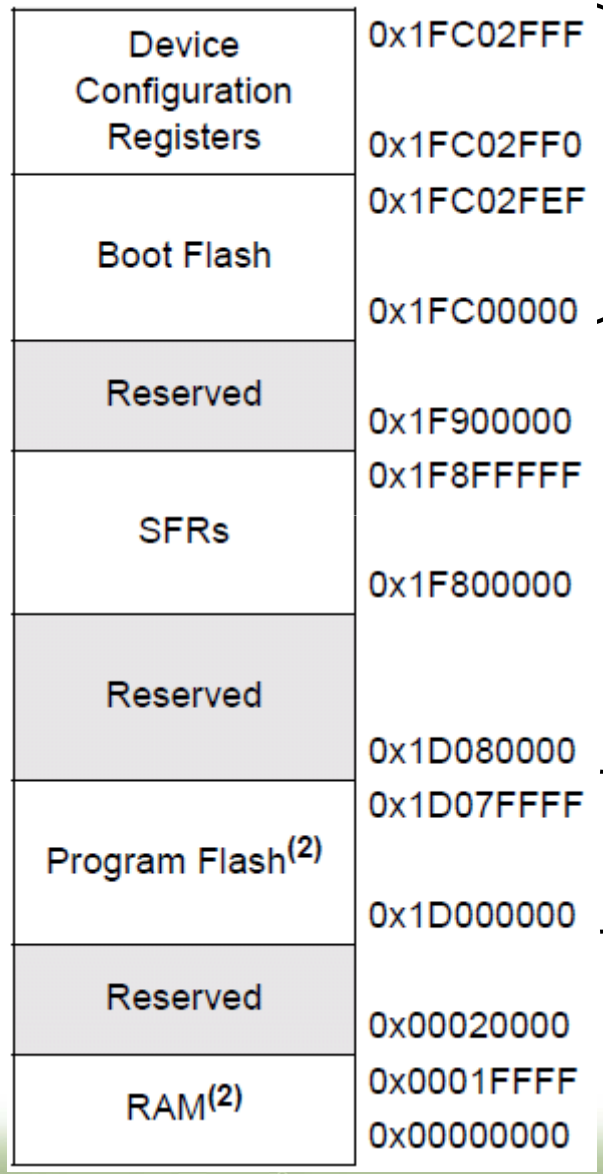


Использование Boot Flash

Вариант 1

Загрузчик →

Приложение →



Boot Flash

Application Flash

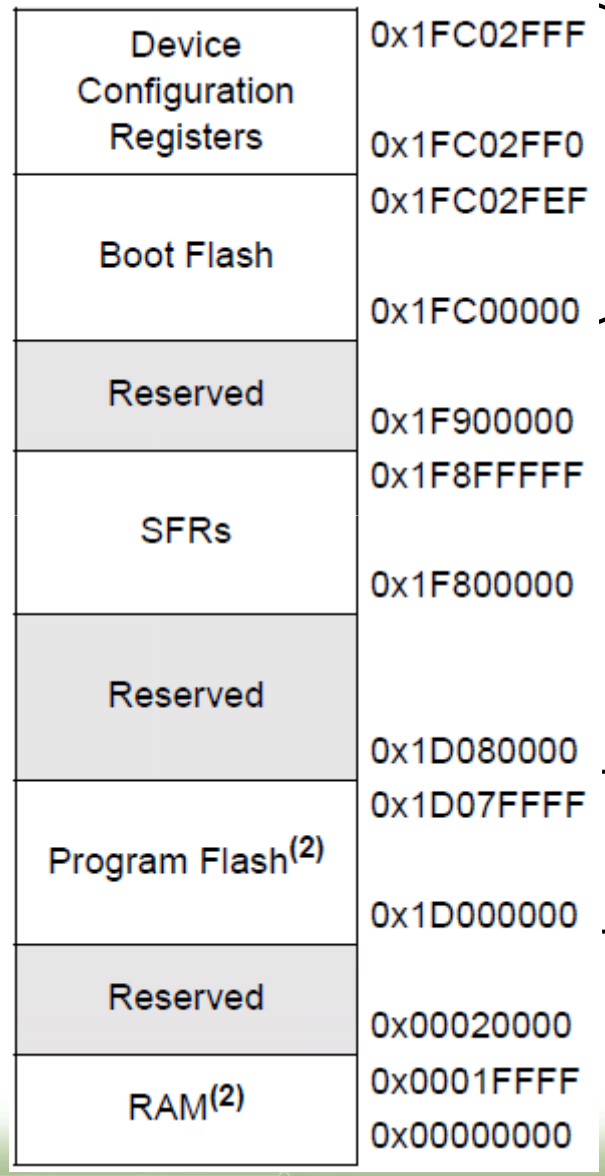
Использование Boot Flash

Вариант 2

Загрузчик. Часть-1

Загрузчик. Часть-2

Приложение



Boot Flash

Application Flash

Стандартные адреса

Linker-Script файл *procdefs.ld*

```
/*************************************************************************
 * For interrupt vector handling
 ************************************************************************/
PROVIDE(_vector_spacing = 0x00000001);
_ebase_address = 0x9FC01000;
/*************************************************************************
 * Memory Address Equates
*************************************************************************/
_RESET_ADDR                 = 0xBFC00000;
_BEV_EXCPT_ADDR             = 0xBFC00380;
_DBG_EXCPT_ADDR             = 0xBFC00480;
_DBG_CODE_ADDR              = 0xBFC02000;
_DBG_CODE_SIZE              = 0xFF0     ;
_GEN_EXCPT_ADDR             = _ebase_address + 0x180;
```

MICROCHIP
MASTERs 2012

адреса

Задает адрес таблицы векторов прерываний (IVT).

```
/***                                                    **********
 * For in
 **********                                            **********************/
PROVIDE(_v    _spacing = 0x00000001);
_ebase_address = 0x9FC01000;
/*****************************************************************************
 * Memory Address Equates
*****************************************************************************/
_RESET_ADDR                    = 0xBFC00000;
_BEV_EXCPT_ADDR                = 0xBFC00380;
_DBG_EXCPT_ADDR                = 0xBFC00480;
_DBG_CODE_ADDR                 = 0xBFC02000;
_DBG_CODE_SIZE                 = 0xFF0     ;
_GEN_EXCPT_ADDR                = _ebase_address + 0x180;
```

Стандартные адреса

procdefs.ld

Определяет адрес перехода при сбросе.

```
/*                                                    **********
                                                           **********/
_e
/*                                                    **********
 * Memor
**********                                                          /
_RESET_ADDR                 = 0xBFC00000;
_BEV_EXCPT_ADDR             = 0xBFC00380;
_DBG_EXCPT_ADDR             = 0xBFC00480;
_DBG_CODE_ADDR              = 0xBFC02000;
_DBG_CODE_SIZE              = 0xFF0     ;
_GEN_EXCPT_ADDR             = _ebase_address + 0x180;
```

Стандартные адреса

Linker-Script файл *procdefs.ld*

При любом сбросе микроконтроллер начинает работу режиме загрузчика (BOOTSTRAP). В этом режиме все Прерывания запрещены, а все исключения направляются на один вектор - _BEV_EXCPT_ADDR

```
/******                                               *********************
 *
                                                       ************/

/*                                                     *************
 *  Me
********                                          ***********************/
_RESET_A                        = 0xBFC00000;
_BEV_EXCPT_ADDR                 = 0xBFC00380;
_DBG_EXCPT_ADDR                 = 0xBFC00480;
_DBG_CODE_ADDR                  = 0xBFC02000;
_DBG_CODE_SIZE                  = 0xFF0     ;
_GEN_EXCPT_ADDR                 = _ebase_address + 0x180;
```

MICROCHIP MASTERs 2012

Стандартные адреса

Linker-Script файл *procdefs.ld*

Адрес прерывания от отладчика

```
/*******                                   *********************
 *
                                                   ************/

/*                                               *************
 * Mem
**********                          ***********************/
_RESET_AD                     = 0xBFC00000;
_BEV_EXC     _ADDR            = 0xBFC00380;
_DBG_EXCPT_ADDR               = 0xBFC00480;
_DBG_CODE_ADDR                = 0xBFC02000;
_DBG_CODE_SIZE                = 0xFF0     ;
_GEN_EXCPT_ADDR               = _ebase_address + 0x180;
```

MICROCHIP MASTERs 2012

Стандартные адреса

Linker-Script файл *procdefs.ld*

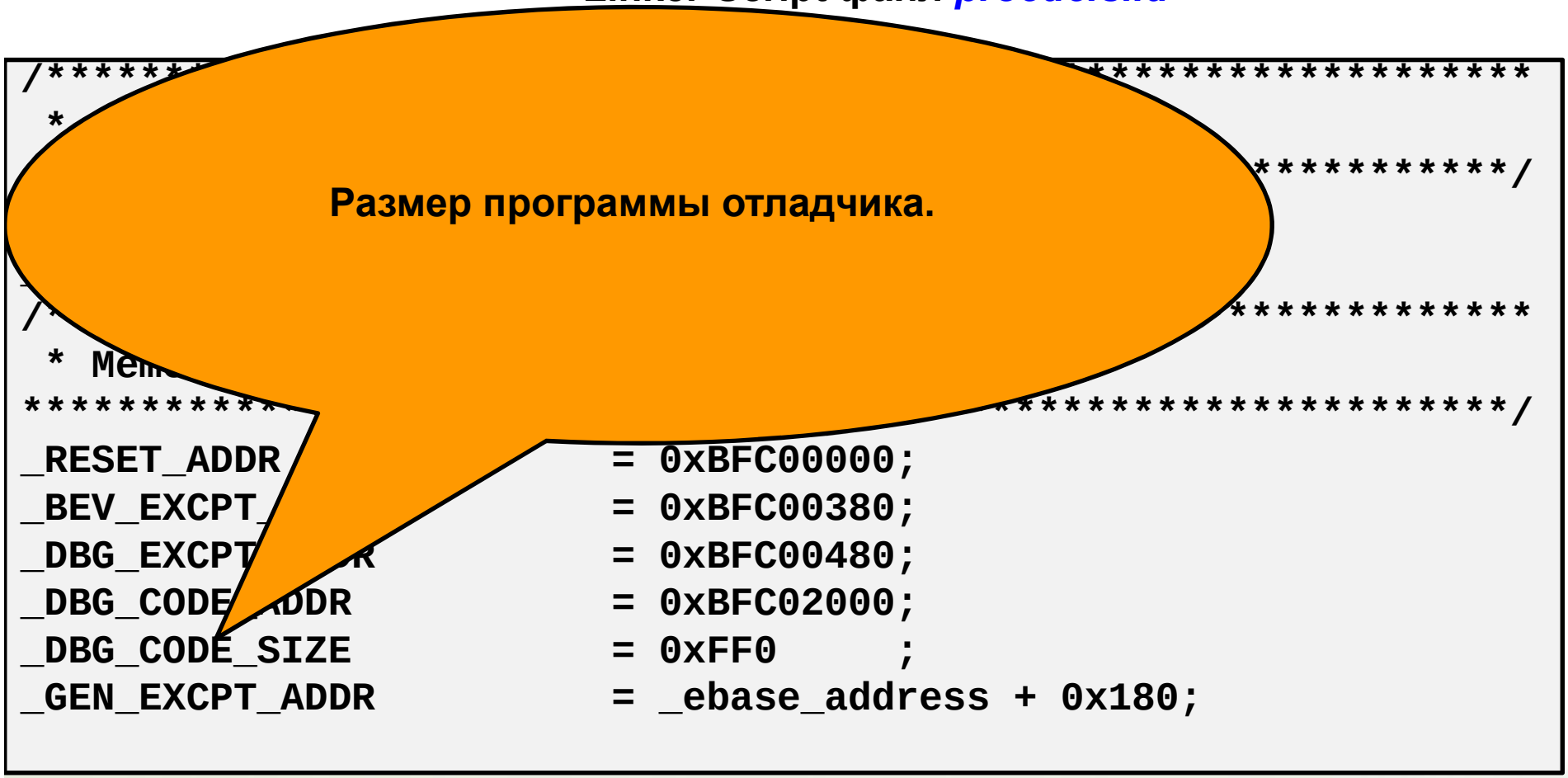
Адрес программы отладчика. При отладке отладчик загружает в микроконтроллер небольшую программу, отвечающую за отладку

```
/*****                                       ********************
 *
                                                    ***********/

_
/*                                                  *************
 * Me
***********                           **********************/
_RESET_ADD                  = 0xBFC00000;
_BEV_EXCPT                  = 0xBFC00380;
_DBG_EXC   _ADDR            = 0xBFC00480;
_DBG_CODE_ADDR              = 0xBFC02000;
_DBG_CODE_SIZE              = 0xFF0     ;
_GEN_EXCPT_ADDR             = _ebase_address + 0x180;
```

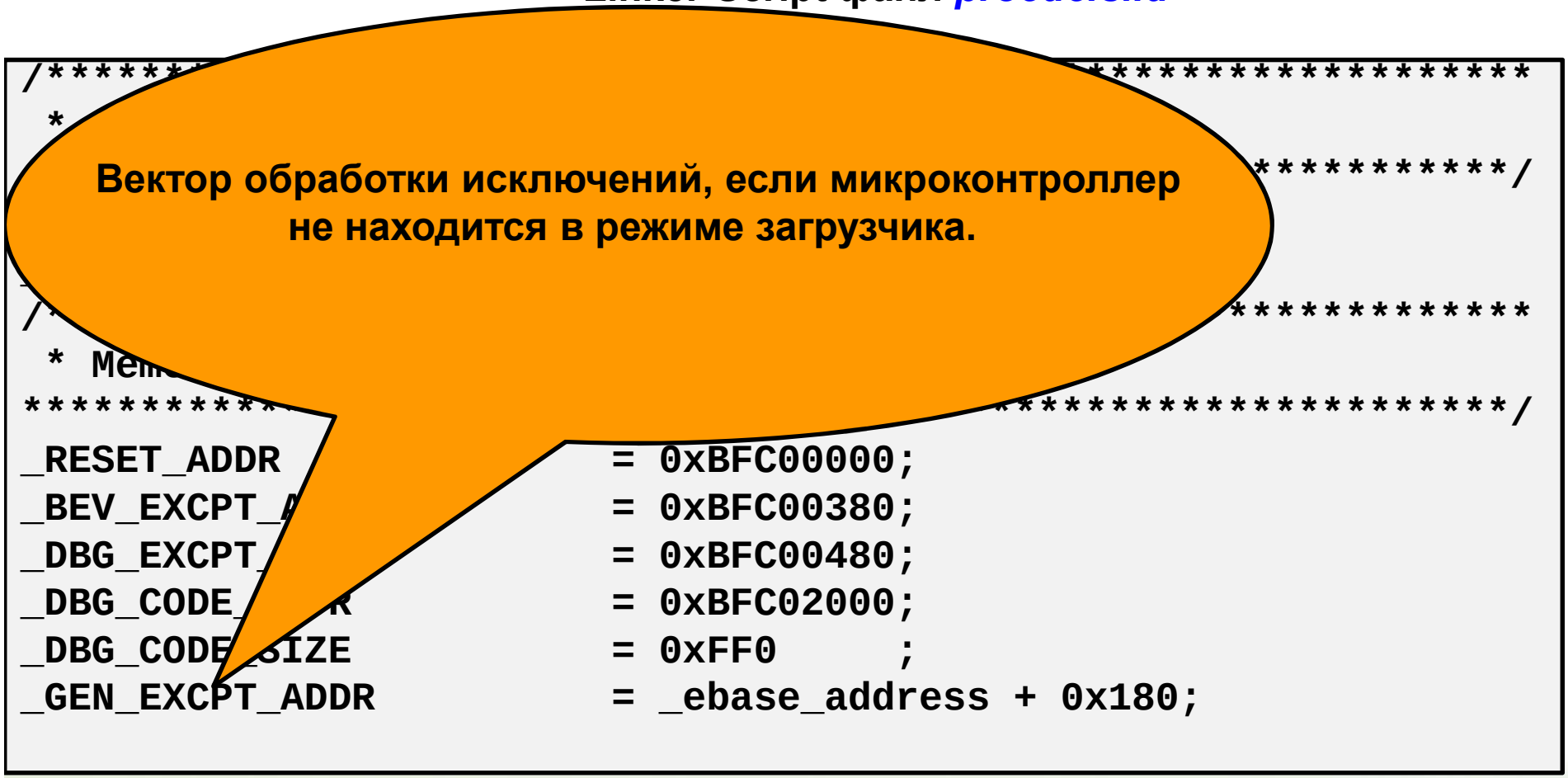
Стандартные адреса

Linker-Script файл *procdefs.ld*



Стандартные адреса

Linker-Script файл *procdefs.ld*

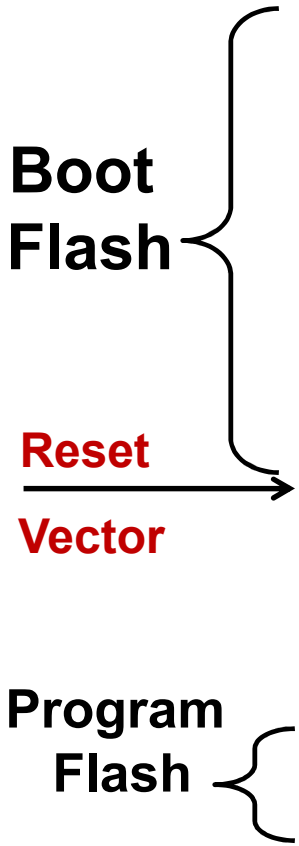


Стандартные классы памяти

Linker-Script файл *procdefs.ld*

```
MEMORY
{
    kseg0_program_mem (rx) :ORIGIN = 0x9D000000, LENGTH = 0x80000
    kseg0_boot_mem         : ORIGIN = 0x9FC00490, LENGTH = 0x970
    exception_mem          : ORIGIN = 0x9FC01000, LENGTH = 0x1000
    kseg1_boot_mem         : ORIGIN = 0xBFC00000, LENGTH = 0x490
    debug_exec_mem         : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
    config3                : ORIGIN = 0xBFC02FF0, LENGTH = 0x4
    config2                : ORIGIN = 0xBFC02FF4, LENGTH = 0x4
    config1                : ORIGIN = 0xBFC02FF8, LENGTH = 0x4
    config0                : ORIGIN = 0xBFC02FFC, LENGTH = 0x4
    kseg1_data_mem (w!x)   : ORIGIN = 0xA0000000, LENGTH = 0x2000
    sfrs                   : ORIGIN = 0xBF800000, LENGTH = 0x10000
}
```

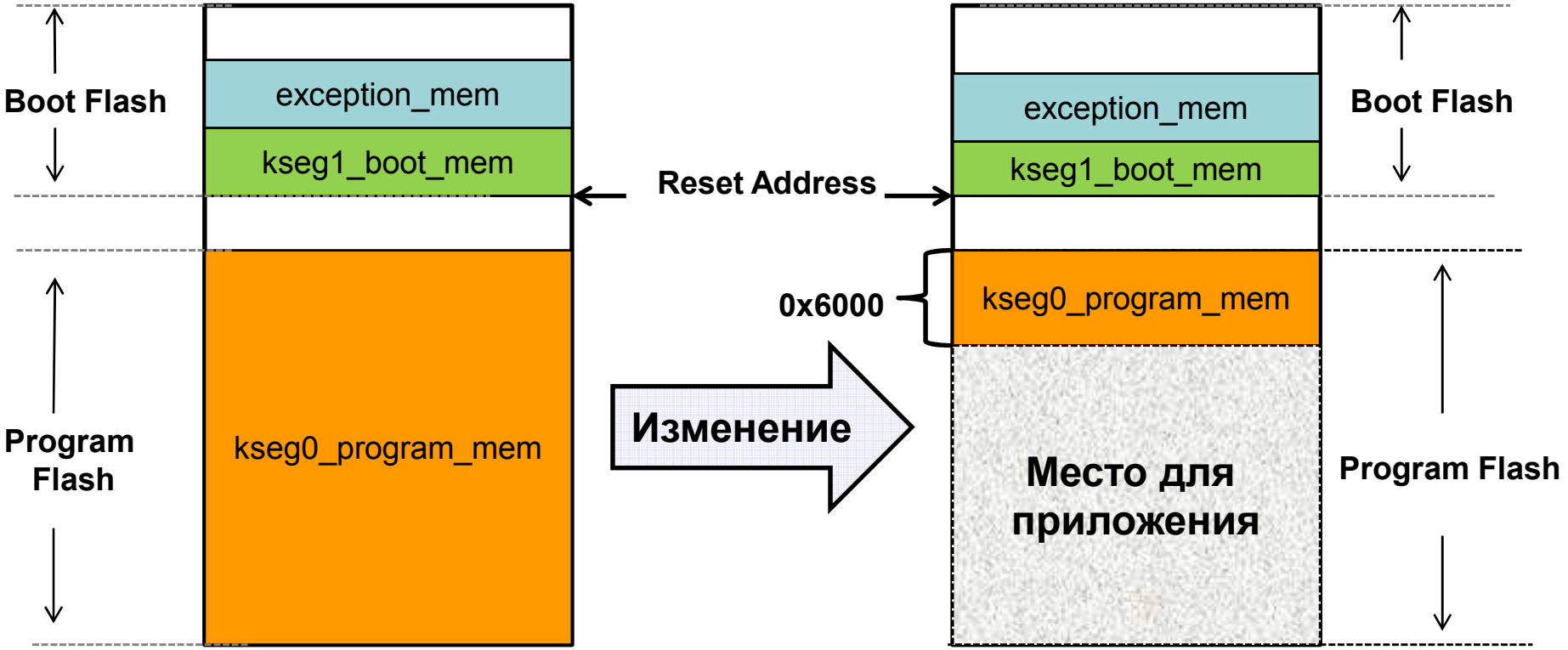
Размещение классов памяти



	Kseg1 address	Kseg0 address	Physical address
Config 0, Config 1, Config2, Config3 (Device Config Reg)	0xBFC02FFF 0xBFC02FF0		0x1FC02FFF 0x1FC02FF0
debug_exec_mem (Debugger Code)	0xBFC02FEF 0xBFC02000		0x1FC02FEF 0x1FC02000
exception_mem (IVT)		0x9FC01FFF 0x9FC01000	0x1FC01FFF 0x1FC01000
kseg0_boot_mem (Not Used)		0x9FC00E00 0x9FC00490	0x1FC00E00 0x1FC00490
kseg1_boot_mem (C Startup Code)	0xBFC0048F 0xBFC00000		0x1FC0048F 0x1FC00000
Reserved			
Sfrs (Peripheral Registers)	0xBF8FFFFF 0xBF800000		0x1F8FFFFF 0x1F800000
Reserved			
kseg0_program_mem (All C Files)		0x9D07FFFF 0x9D000000	0x1D07FFFF 0x1D000000
Reserved			
kseg1_data_mem (RAM)	0xA001FFFF 0xA0000000		0x0001FFFF 0x00000000

Изменения для загрузчика

Пример изменения расположения классов памяти в файле *procdefs.ld* для загрузчика



Расположение по умолчанию

Новое расположение для загрузчика

Изменение размера класса памяти kseg0_program_mem для загрузчика

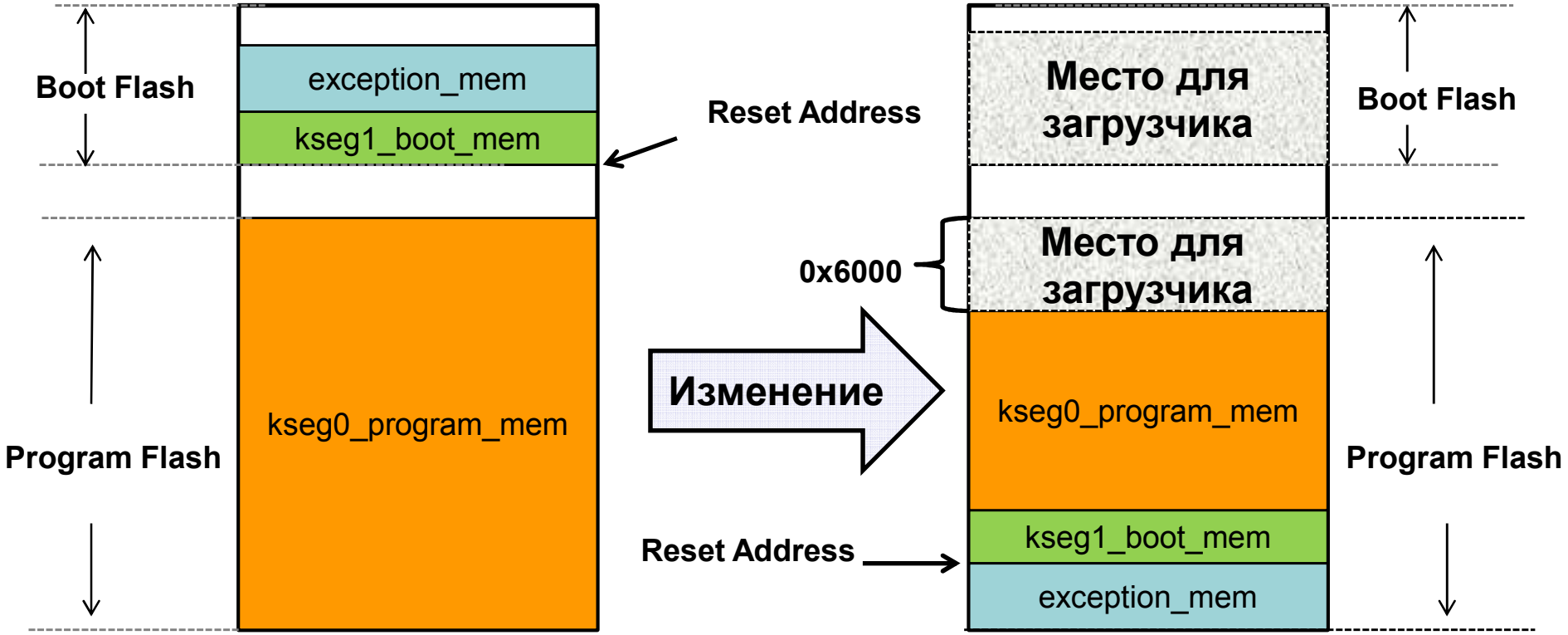
Linker-Script файл *procdefs.ld*

```
MEMORY
{
    kseg0_program_mem (rx) :ORIGIN = 0x9D000000, LENGTH = 0x6000
    kseg0_boot_mem         : ORIGIN = 0x9FC00490, LENGTH = 0x970
    exception_mem          : ORIGIN = 0x9FC01000, LENGTH = 0x1000
    kseg1_boot_mem         : ORIGIN = 0xA0000000, LENGTH = 0x490
    debug_exec             : ORIGIN = 0xA0000490, LENGTH = 0xFF0
    config3                : ORIGIN = 0xA0001480, LENGTH = 0x4
    co                     : ORIGIN = 0xA0001484, LENGTH = 0x4
    co                     : ORIGIN = 0xA0001488, LENGTH = 0x4
    con                    : ORIGIN = 0xA000148C, LENGTH = 0x4
    kseg1_dat              : ORIGIN = 0xA0001490, LENGTH = 0x2000
    sfrs                   : ORIGIN = 0xBF800000, LENGTH = 0x10000
}
```

Изменение длины класса памяти kseg0_program_mem

Изменения для приложения

Изменение расположения классов памяти в файле *procdefs.ld* для приложения



Расположение по умолчанию

Новое расположение для приложеия

Перенос таблицы прерываний для приложения

Linker-Script файл *procdefs.ld*

Изменение положения класса памяти exception_mem

```
MEM
{
                              IN = 0x9D000000, LENGTH = 0x80000
    kseg0_boot_mem            : ORIG   = 0x9FC00490, LENGTH = 0x970
    exception_mem             : ORIGIN = 0x9D006000, LENGTH = 0x1000
    kseg1_boot_mem            : ORIGIN = 0xBFC00000, LENGTH = 0x490
    debug_exec_mem            : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
```

```
PROVIDE(_vector_spacing = 0x00000001);
_ebase_address = 0x9D006000;
```

Изменение адреса таблицы векторов прерываний

Перенос вектора сброса и области старта программы для приложения

Linker-Script файл *procdefs.ld*

```
MEM
{
                                 = 0x9D000000, LENGTH = 0x80000
                                 IN = 0x9FC00490, LENGTH = 0x970
    exception_                   N = 0x9D006000, LENGTH = 0x1000
    kseg1_boot_mem           : ORIGIN = 0xBD007000, LENGTH = 0x490
    debug_exec_mem           : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
```

Изменение положения класса памяти kseg1_boot_mem (размещается C-startup)

```
_RESET_ADDR                  = 0xBD007000;
_BEV_EXCPT_ADDR              = 0xBFC0038
_DBG_EXCPT_ADDR              = 0xBFC
_DBG_CODE_ADDR               = 0xBFC
```

Изменение адреса сброса

Перенос векторов исключений и изменение размера памяти программм для приложения

Linker-Script файл *procdefs.ld*

```
MEMORY
{
    kseg0_program_mem (rx) :ORIGIN = 0xBD007490, LENGTH = 0x78B70
    kseg0_boot_mem         : ORIGIN = 0x9FC00490, LENGTH = 0x970
    ram                    : ORIGIN = 0x9D006000, LENGTH = 0x1000
    boot_mem               : ORIGIN = 0xBD007000, LENGTH = 0x490
    config3                : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
```

Изменение положения и размера памяти для программы приложения

```
_RESET_ADDR                     = 0xBD007000;
_BEV_EXCPT_ADDR                 = 0xBD007380;
_DBG_EXCPT_ADDR                 = 0xBD007480;
_DBG_CODE_ADDR                  = 0xBFC02000;
```

Изменение адресов векторов исключений

ВАЖНО!

- При размещении *exception_mem* всегда помещайте его на 4KB границу
- Не изменяйте размеры следующих секций:
  - exception_mem (IVT)
  - kseg1_boot_mem

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- **Создать механизм входа в режим загрузчика в проекте загрузчика**
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Вход в режим обновления ПО

- **Признаки:**
- Состояние вывода (выводов)
- Ожидание команды от PC с таймаутом
- Отсутствие ошибок в имеющемся ПО
- Состояние ячейки энергонезависимой памяти
- Другие

- Если признака не обнаружено, управление передается приложению

Что требуется для создания загрузчика

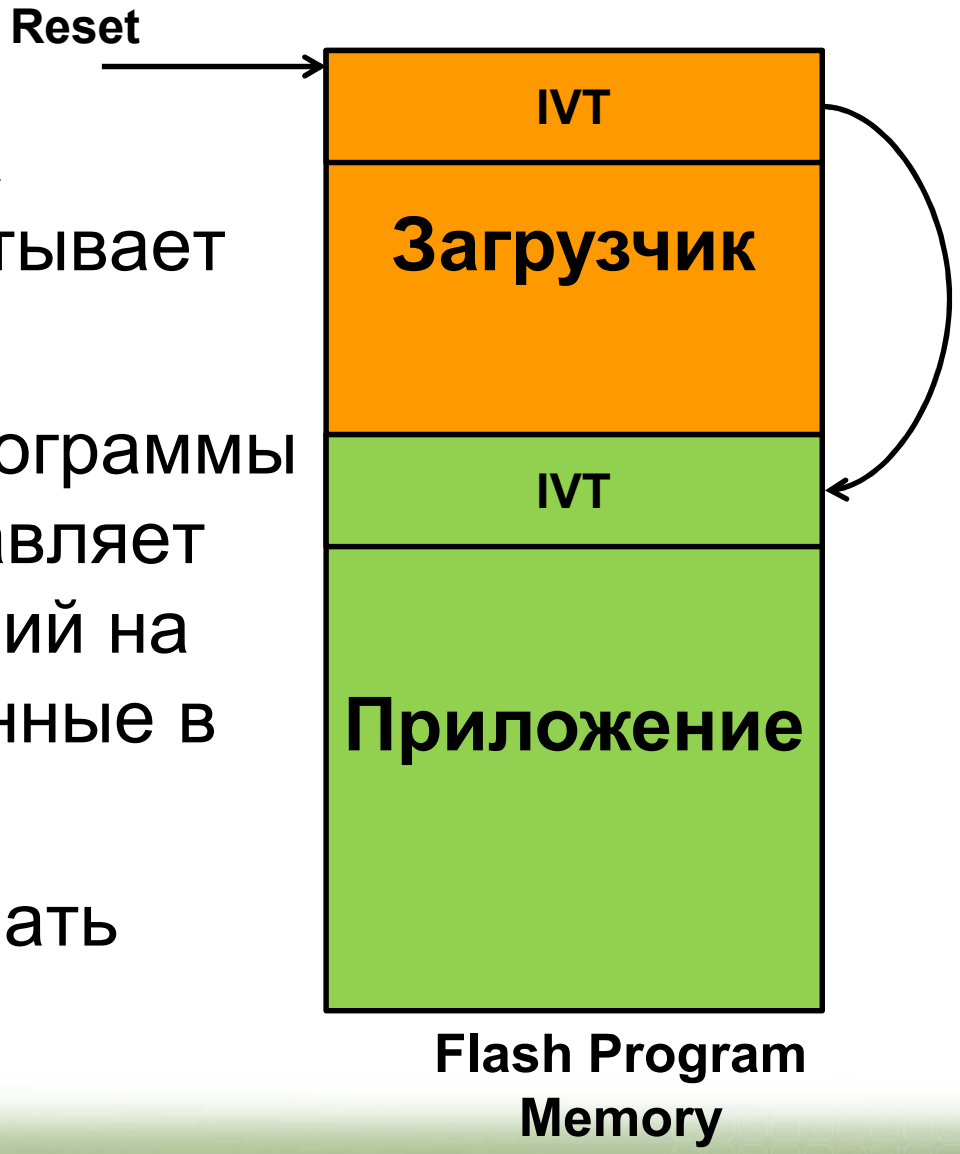
- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- **Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.**
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

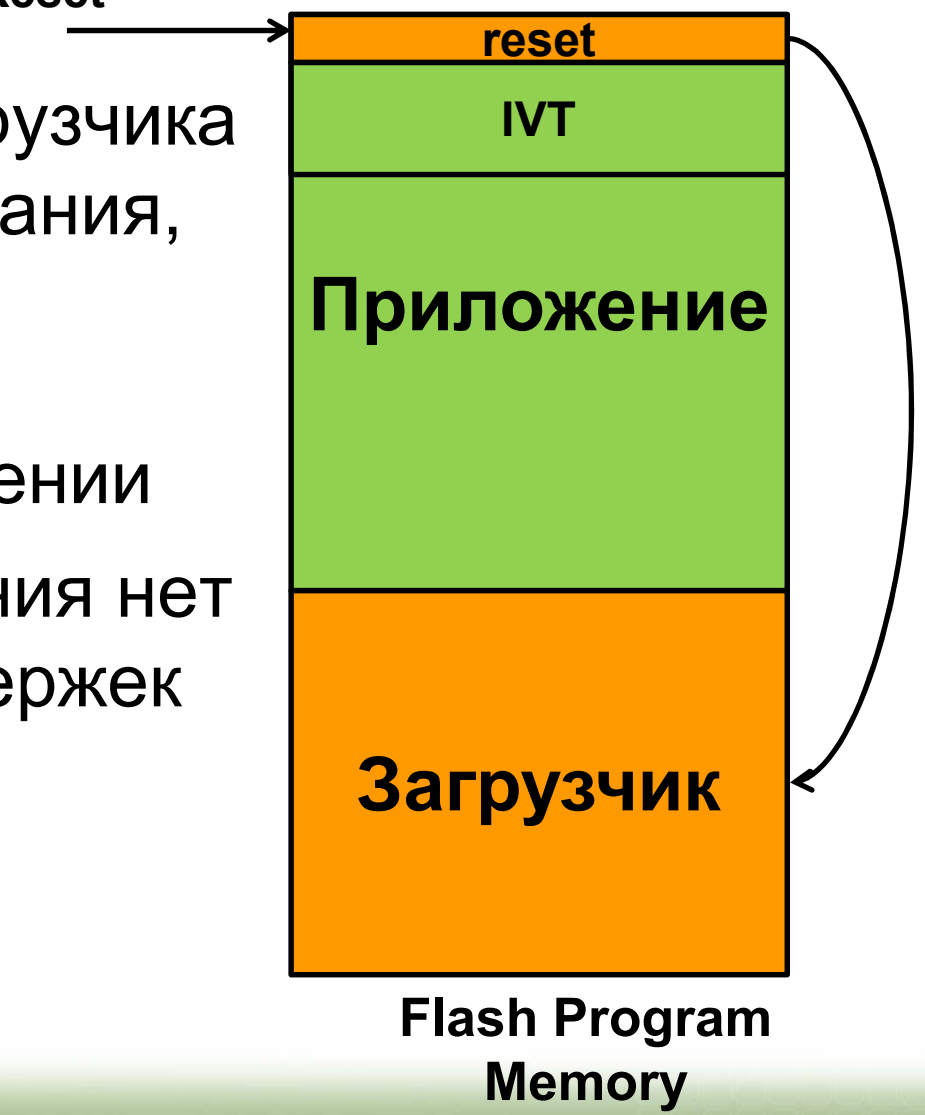
Обработка прерываний

- Аппаратные вектора прерываний обрабатывает загрузчик
- Во время работы программы загрузчик перенаправляет обработку прерываний на вектора, расположенные в приложении
- Загрузчик должен знать вектора прерываний приложения



Ускоренная обработка прерываний

- Если для работы загрузчика не требуются прерывания, то таблицу векторов прерывания можно разместить в приложении
- При работе приложения нет дополнительных задержек при прерывании



Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- **Создать механизм приема и обработки данных с ПО в проекте загрузчика.**
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Канал связи

- **Передача нового ПО может происходить по следующим каналам связи:**
  - UART
  - I^2C, SPI
  - USB (device HID)
  - Ethernet (Wi-Fi)
  - Радио (802.15.4, subGHz)
  - SD карта
  - USB Flash накопитель
  - Другие
- Для организации канала связи можно использовать библиотеки Microchip (USB, TCP/IP, MiWi и т.д.)

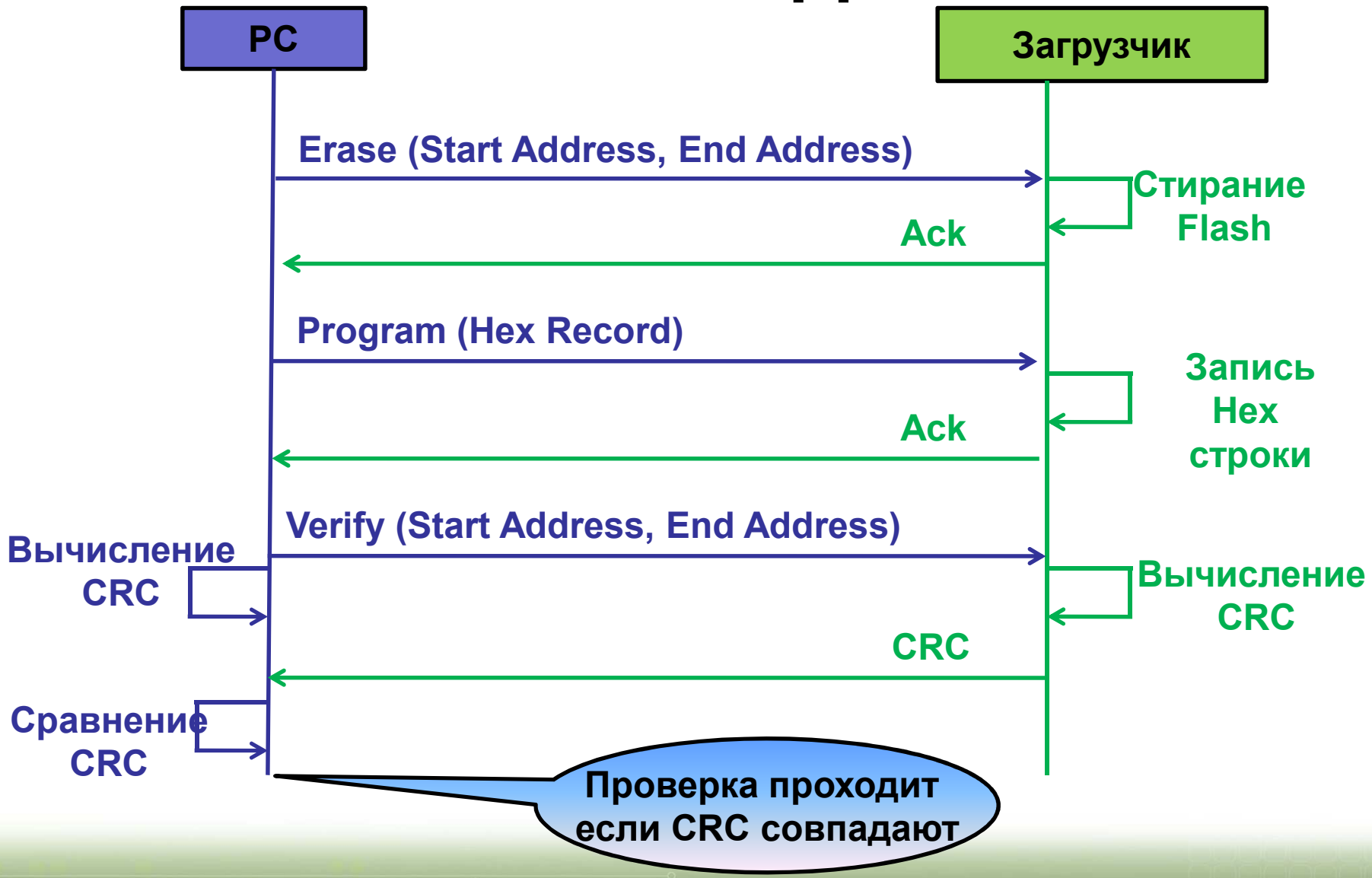
Протокол связи

- Формат (протокол) обмена данными по выбранному каналу связи может быть любым и должен быть согласован с программой на PC, которая эти данные передает
- При использовании протоколов обмена Microchip можно воспользоваться готовыми программами для PC для загрузки данных.

Команды загрузчика

- **По каналу связи загрузчик принимает команды и данные**
- **Необходимые команды для обновления ПО:**
  - Erase Flash
  - Program Flash
  - Verify Flash

Последовательность команд



Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

➢Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- **Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH**
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Работа с Flash

- Загрузка через загрузчик возможна для микроконтроллеров с поддержкой самопрограммирования
- Стирание данных производится секторами.
- Запись данных производится страницами
- Обычно сектор больше страницы
- Низкоуровневые библиотеки для работы с Flash обычно встроены в компиляторы

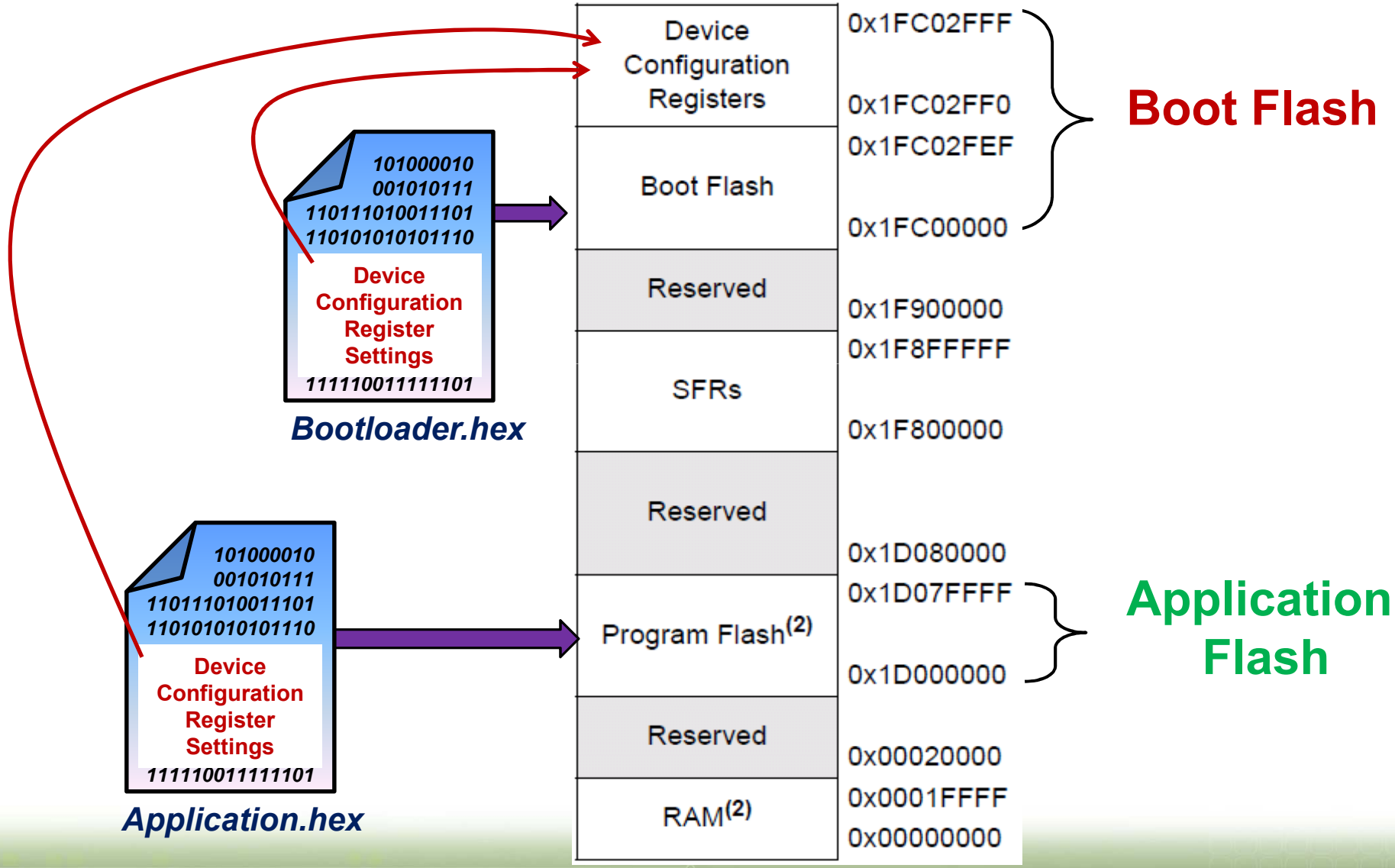
Запись во Flash

- После подачи команды на запись работа микроконтроллера прекращается до окончания записи. Прерывания не обрабатываются
- В PIC32 есть возможность выполнять программы из RAM. В этом случае, при записи во Flash работа микроконтроллера продолжается.
- В PIC32 есть возможность стирания всего массива Flash : Program Flash или Boot Flash. Программа не должна выполняться из того массива, который стирается.

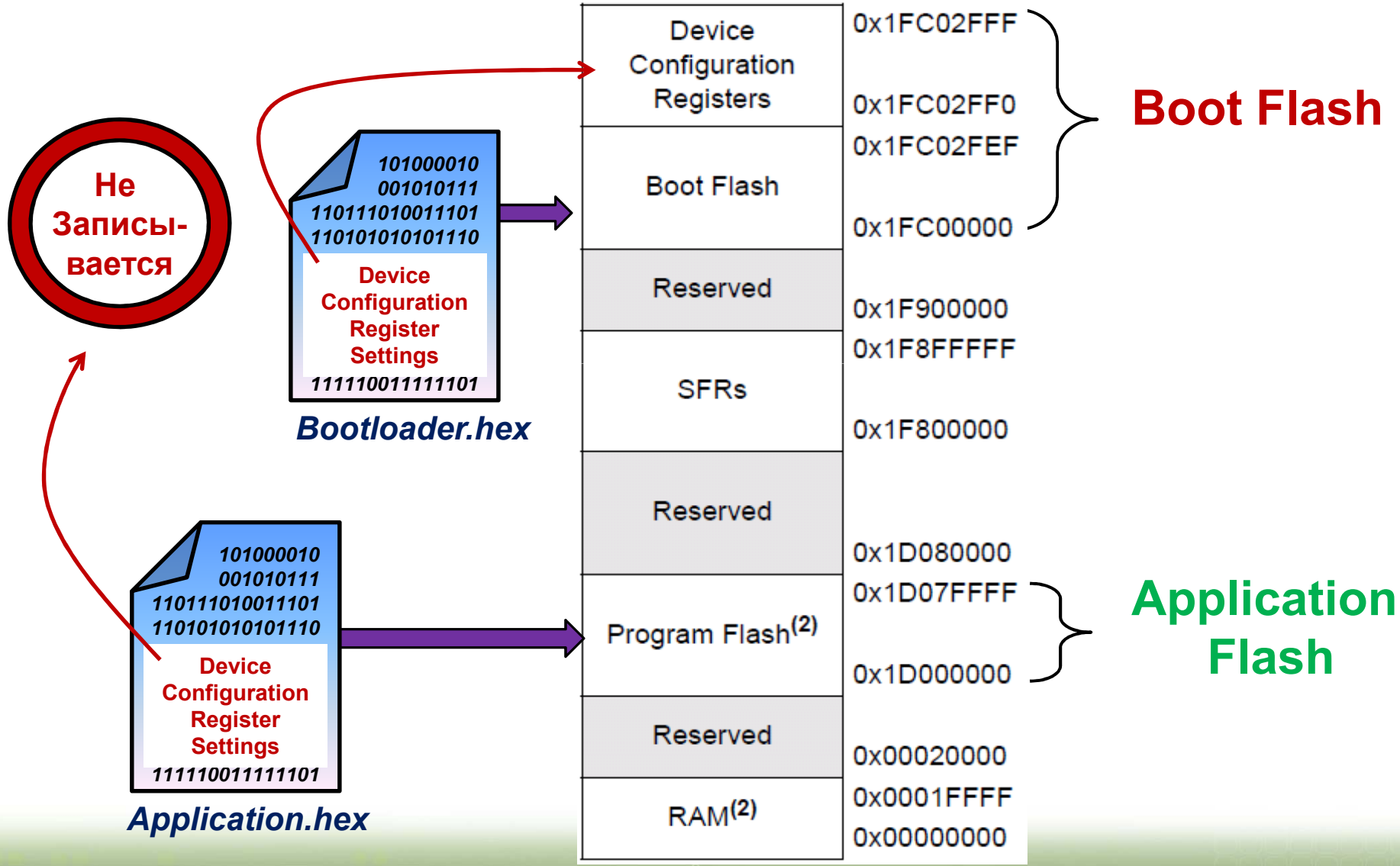
Работа с регистрами конфигурации

- Обычно проекты загрузчика и приложения используют одинаковые настройки конфигурации
- Сбой во время записи регистров конфигурации может привести к прекращению работы изделия
- В некоторых семействах PIC регистры конфигурации расположены в массиве общей памяти, поэтому необходимо защищать/сохранять настройки при стирании памяти при загрузке ПО.

Регистры конфигурации PIC32



Регистры конфигурации PIC32



Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- **Опционально создать механизм взаимодействия загрузчика и приложения**
- Опционально определить степень защиты загрузчика и приложения

Взаимодействие загрузчика и приложения

- Приложение может перейти к режиму загрузки во время своей работы, например, по событию
- Приложение может использовать функции загрузчика в своей работе
- Приложение, как и загрузчик, при необходимости, может быть запрограммировано программатором

Переход от приложения к загрузчику

- Приложение, при необходимости, может прекратить свою работу и передать управление загрузчику
- Для передачи управления загрузчику необходимо создать признак перехода и перейти по адресу сброса
  - **Включить Watchdog и уйти в бесконечный цикл**
  - **Для 16-битных PIC – выполнение команды RESET**
  - **Для 32-битных PIC – установка бита SWR**
- Альтернативный вариант перехода – знать адрес места программы загрузчика после проверки признака перехода

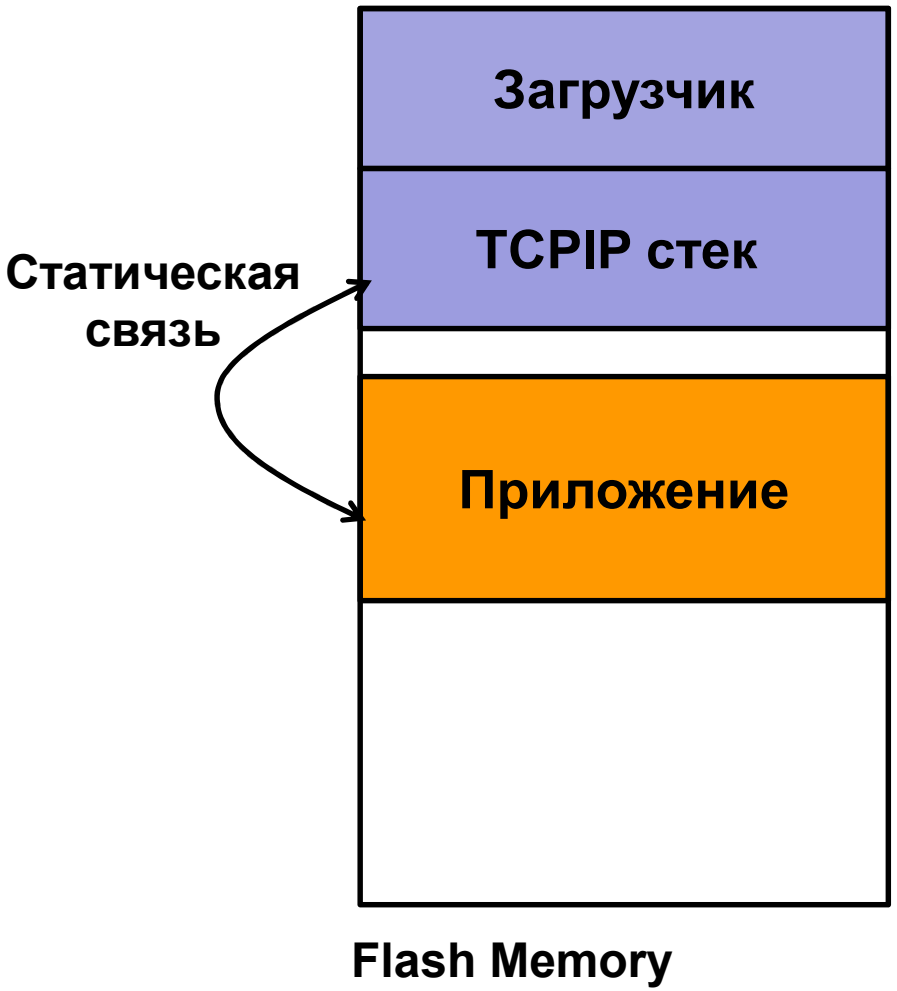
Переход от приложения к загрузчику

- Приложение, при необходимости, может прекратить свою работу и передать управление загрузчику
- Для передачи управления загрузчику необходимо создать признак перехода и перейти по адресу сброса
  - **Включить Watchdog и уйти в бесконечный цикл**
  - **Для 16-битных PIC – выполнение команды RESET**
  - **Для 32-битных PIC – установка бита SWR**
- Альтернативный вариант перехода – знать адрес места программы загрузчика после проверки признака перехода

Независимые проекты

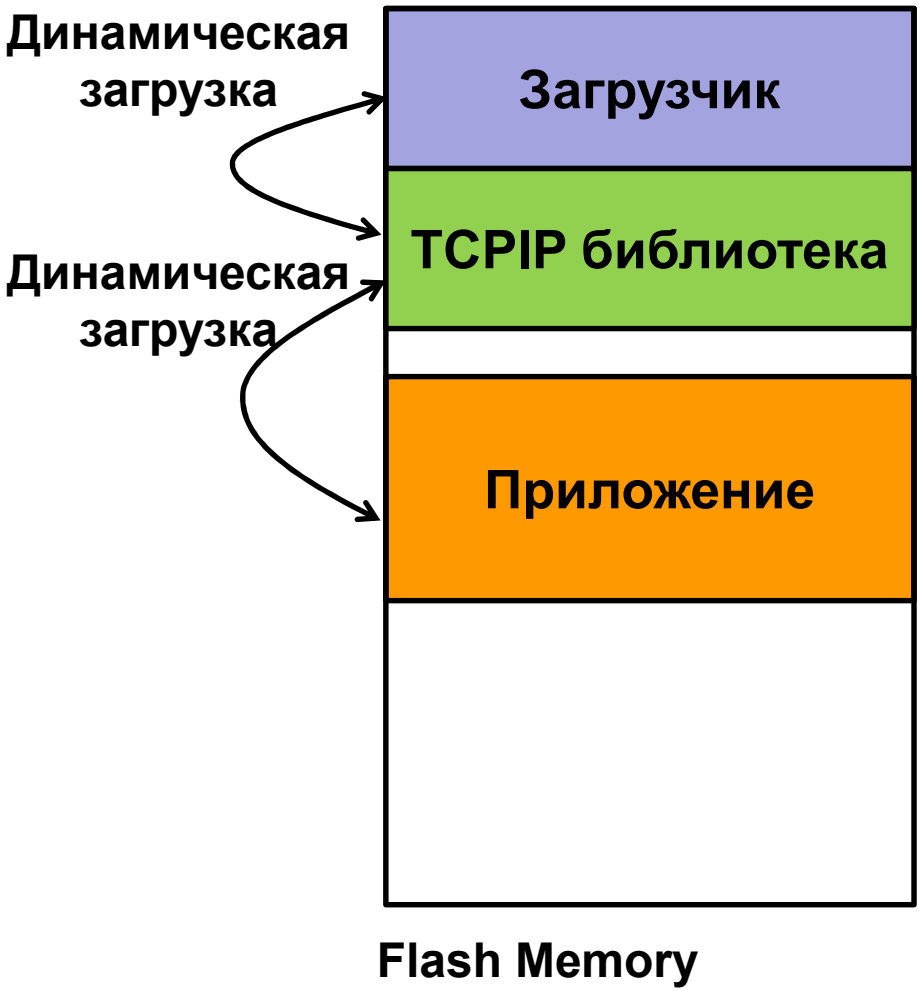
- Независимые проекты
- Память содержит две копии стека
- Нерациональное использование памяти

Статическое связывание



- Загрузчик и стек в одном проекте
- Приложению в другом проекте известны адреса функций API
- Одна копия стека
- Затруднены внесение изменений, параллельная разработка проектов

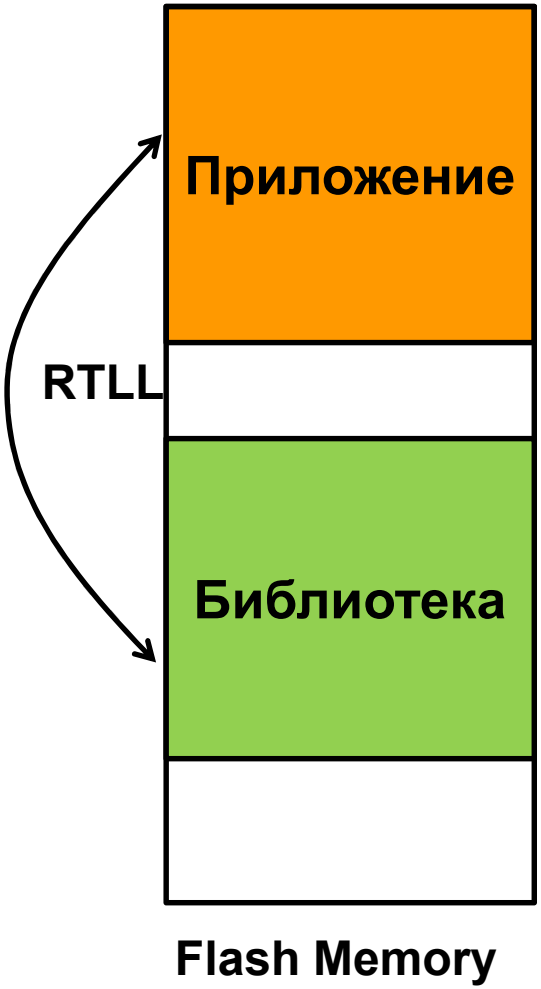
RTLL – динамическая загрузка



- Три независимо разрабатываемых проекта
- Одна копия стека
- Эффективное использование памяти
- Проектам необходимо знать только одну точку входа
- Все функции задаются по имени

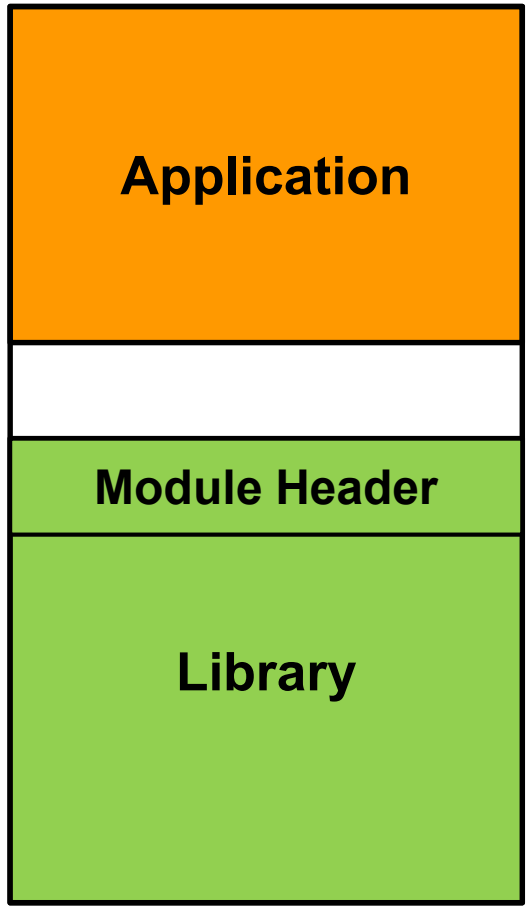
Цель RTLL

- Определить адрес библиотечной программы во время выполнения программы
- Адрес неизвестен на этапе компиляции

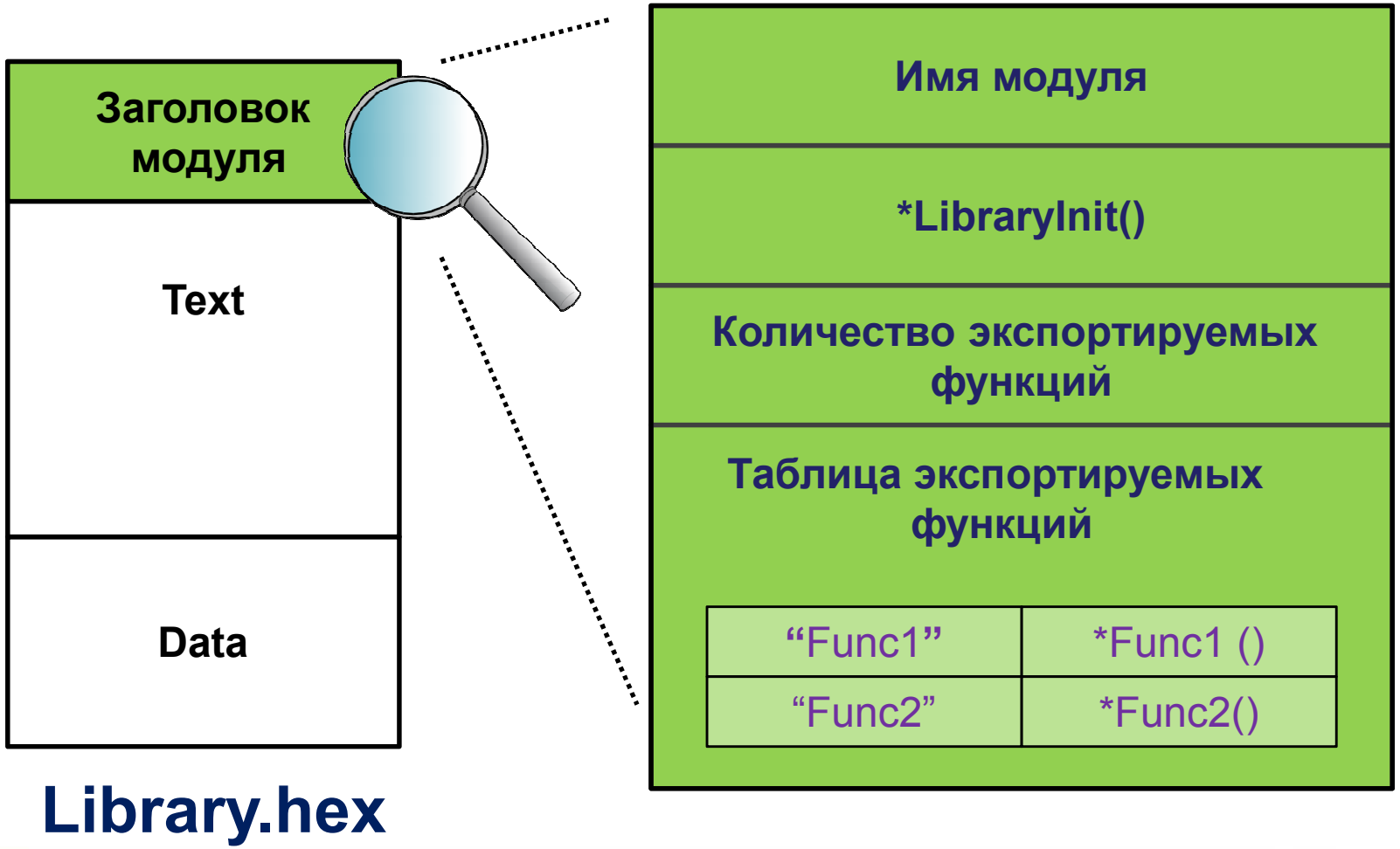


Как это работает ?

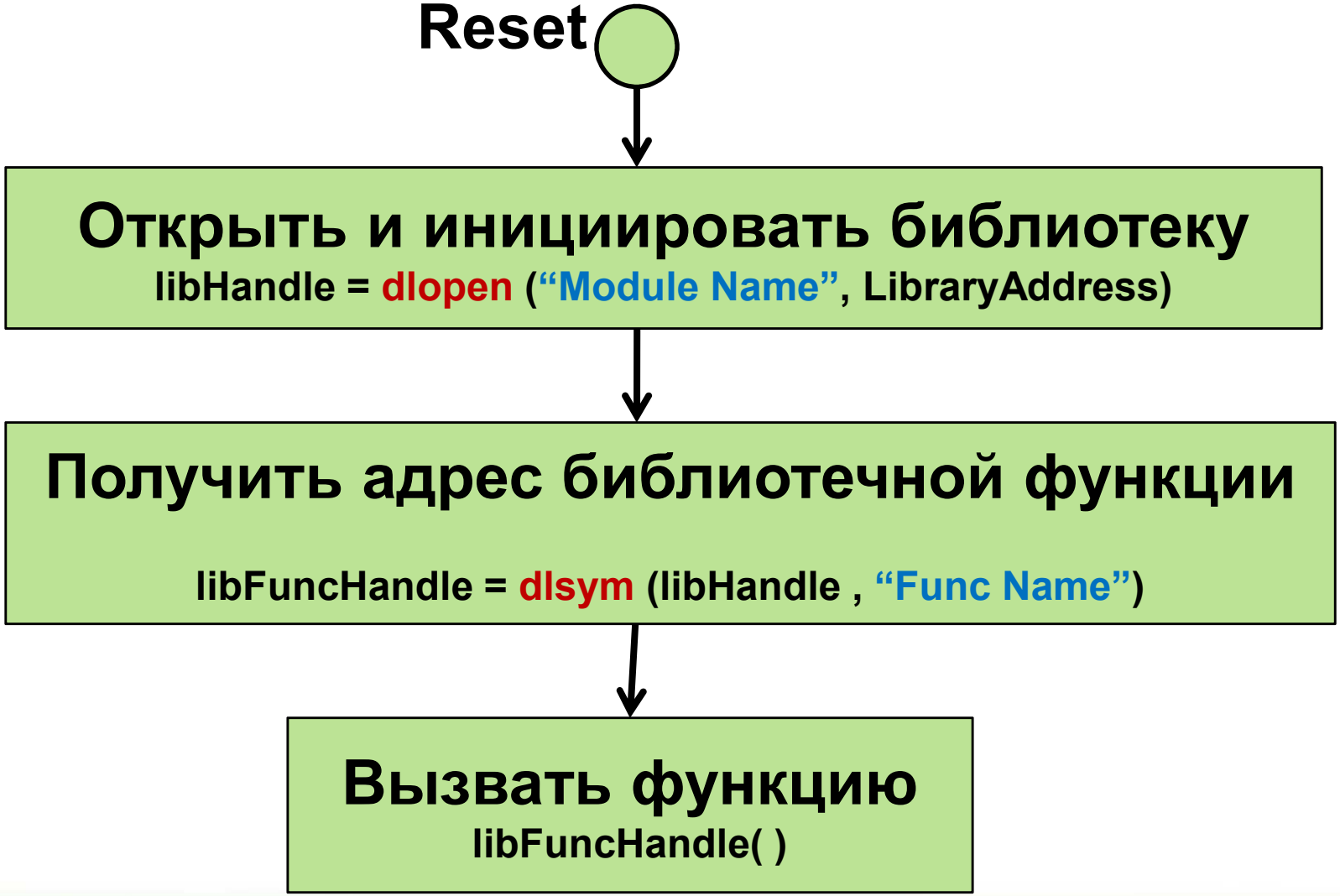
- Библиотека размещает адрес подпрограммы в *стандартной структуре*
- Приложение знает адрес *стандартной структуры*
- Во время работы приложение извлекает адрес подпрограммы из *стандартной структуры* по имени подпрограммы



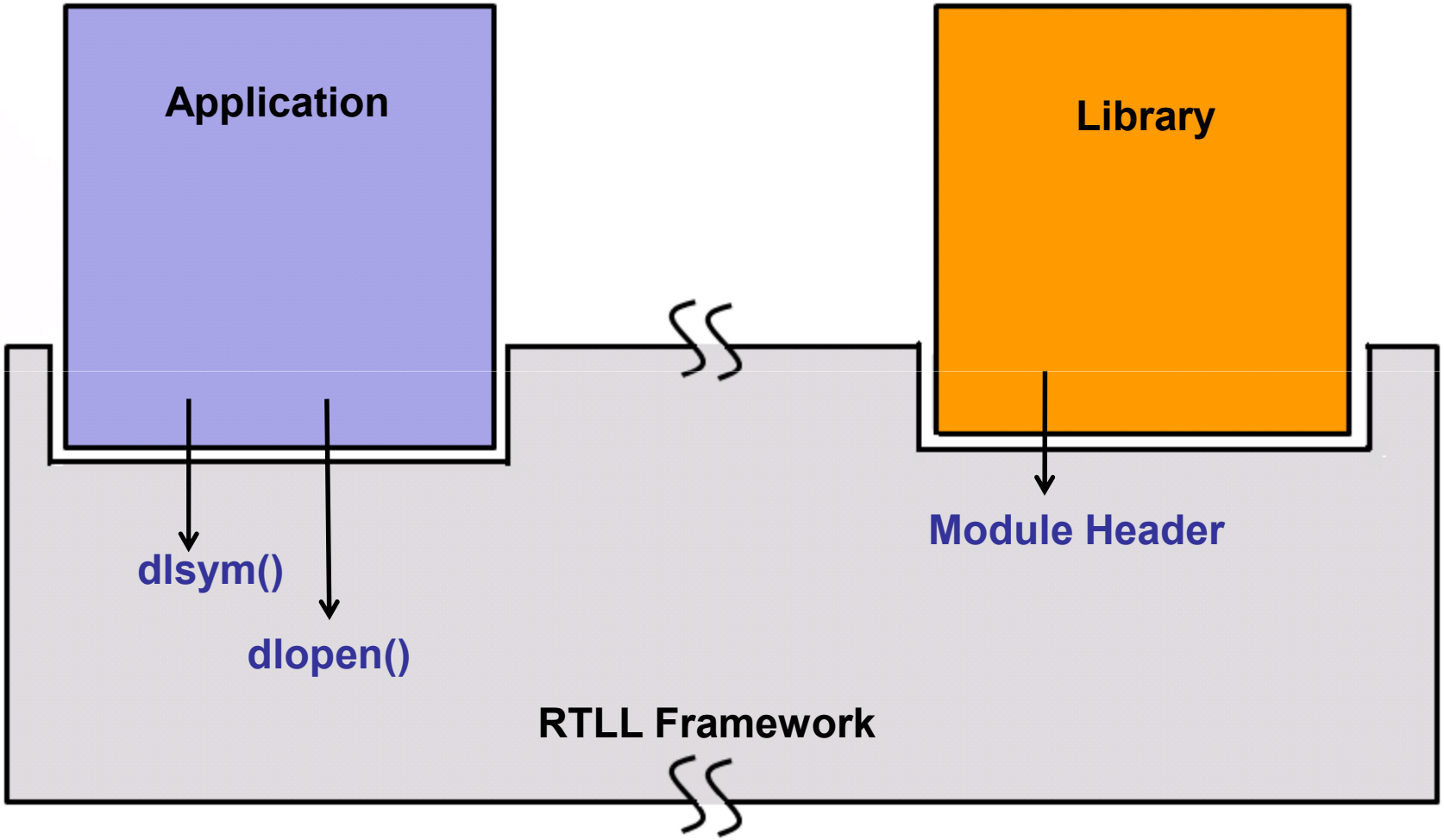
Структура библиотеки



Блок схема приложения



RTLL - Framework



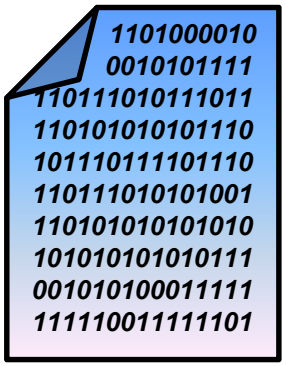
RTLL -Сохранение контекста

- **Для упрощения работы механизм сохранения контекста не предусмотрен**
  - Библиотека использует стек приложения
  - Библиотека использует «кучу» приложения
  - GP регистры использует только приложение. Относительная адресация GP регистров в библиотеке отключена (опция компилятора “-G0”)

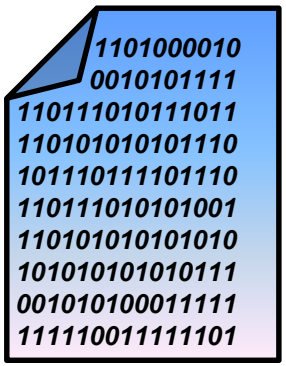
Важные замечания

- В библиотеке не должно быть обработчиков прерываний
- В библиотеке не должно быть установки регистров конфигурации
- В библиотеке не должно быть стартового кода Си (C-Startup) и функции main()
- Необходимо использовать опцию линкера “*--no-gc-sections*” чтобы запретить «сбор мусора»

Соединение Hex файлов



- При производстве изделия бывает необходимо запрограммировать загрузчик и приложение одновременно

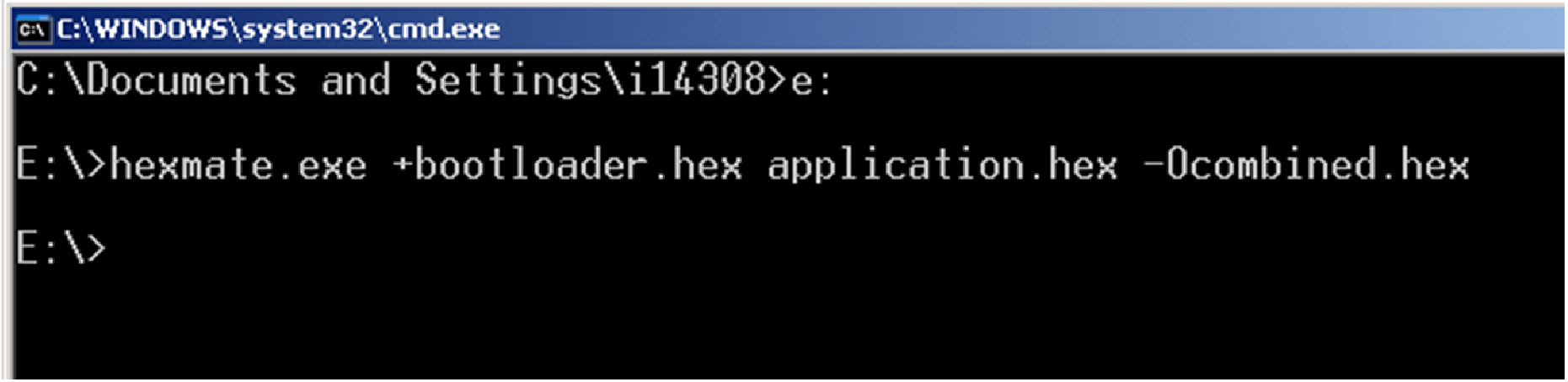


HexMate

- **Возможности**
  - В составе компилятора XC8 имеется утилита HexMate
  - Позволяет сформировать единый HEX файл из двух других HEX файлов
  - Автоматически удаляет перекрывающиеся области

HexMate

Использование



- “+” означает, что в случае перекрывающихся областей будут использоваться данные из bootloader

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- Опционально определить степень защиты загрузчика и приложения

Что требуется для создания загрузчика

- Два типа проекта – загрузчик и приложение
- Сегментировать память для обоих проектов
- Создать механизм входа в режим загрузчика в проекте загрузчика
- Создать механизм передачи обработки прерываний в приложение в проекте загрузчика. Учесть механизм обработки прерываний в проекте приложения.
- Создать механизм приема и обработки данных с ПО в проекте загрузчика.
- Создать механизм работы с FLASH в проекте загрузчика. Учесть особенности различных областей FLASH
- Опционально создать механизм взаимодействия загрузчика и приложения
- **Опционально определить степень защиты загрузчика и приложения**

Зачем защищать ?

- Уверенность, что не произойдет непредвиденного изменения конфигурации
- Уверенность, что не произойдет непредвиденного изменения программы/констант
- Защита интеллектуальной собственности

Какие бывают защиты ?

- **Code protect** – защита от считывания программатором
  - Всей памяти
  - Области памяти (Conf, Boot, Flash, EEPROM)
  - Отдельных сегментов
- **Write protect** – защита от программного стирания/записи
  - Всей памяти
  - Области памяти (Conf, Boot, Flash, EEPROM)
  - Отдельных сегментов
- **External Read protect** - защита от программного чтения отдельных сегментов извне сегмента
- **CodeGuard** – расширенная система защиты интеллектуальной собственности

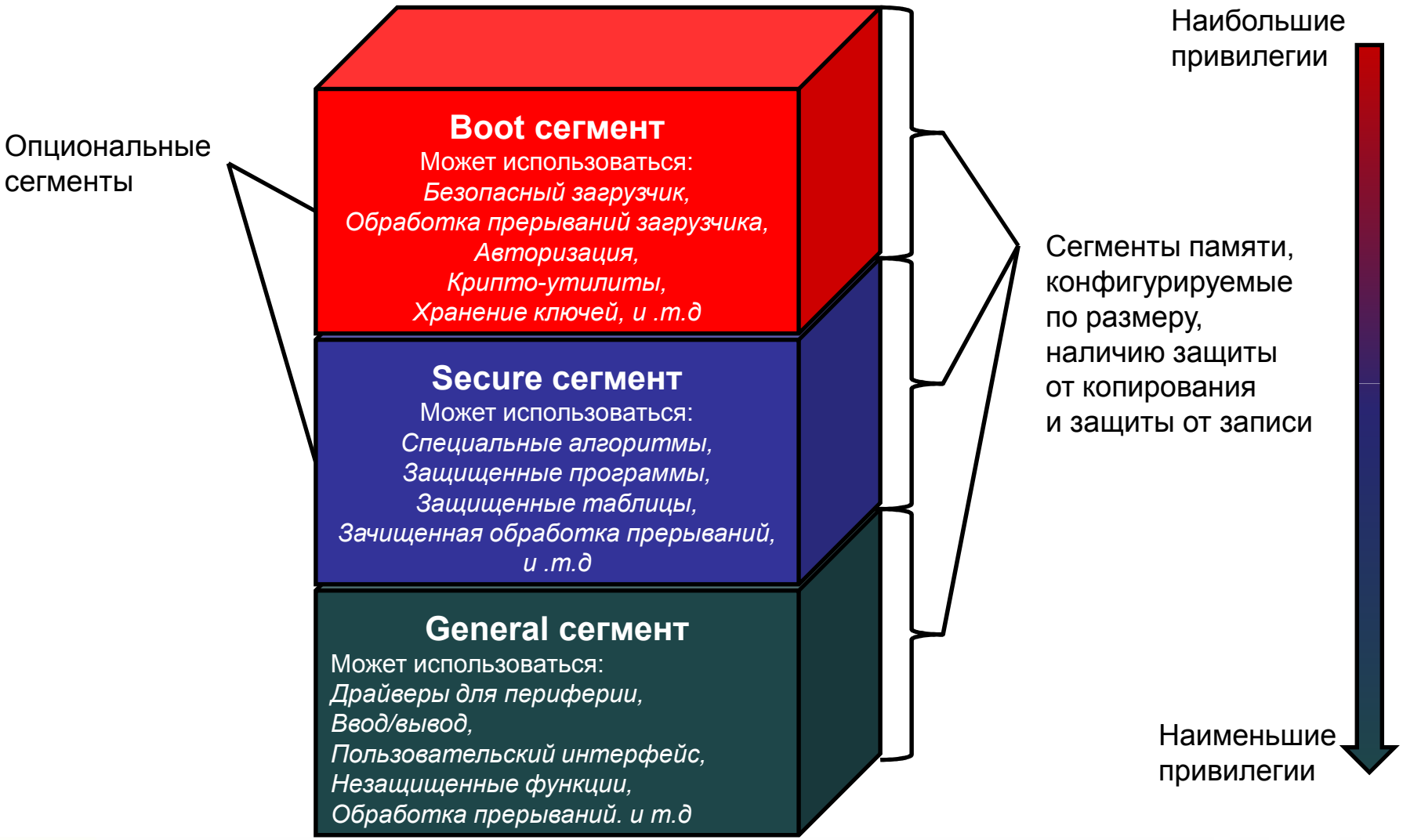
Особенности защиты

- Степень защиты устанавливается в регистрах конфигурации
- Различные микроконтроллеры реализуют различные механизмы защиты
  - Уточняйте по документации
- Code protect и Write protect не защищают от возможности считывания
  - Загруженная программа может считывать память, в т.ч. загрузчик !!!
- Загрузчик не может изменять память, защищенную Write protect
- Загрузчик может изменять регистры конфигурации (если они не защищены от записи), в т.ч изменить степень защиты
  - Применять с осторожностью !!!

Code Guard

- **Система CodeGuard обеспечивает расширенный механизм защиты кода и дает возможность нескольким независимым разработчикам безопасно совместно использовать ресурсы микроконтроллера**:
  - Сегментация памяти с различными уровнями доступа
  - Посегментное стирание/программирование для реализации безопасных загрузчиков
  - Безопасная обработка прерываний
  - Безопасная разработка и отладка

Code Guard


Наибольшие привилегии
Опциональные сегменты
Boot сегмент
Может использоваться:
Безопасный загрузчик,
Обработка прерываний загрузчика,
Авторизация,
Крипто-утилиты,
Хранение ключей, и .т.д
Secure сегмент
Может использоваться:
Специальные алгоритмы,
Защищенные программы,
Защищенные таблицы,
Зачищенная обработка прерываний,
и .т.д
General сегмент
Может использоваться:
Драйверы для периферии,
Ввод/вывод,
Пользовательский интерфейс,
Незащищенные функции,
Обработка прерываний. и т.д
Сегменты памяти, конфигурируемые по размеру, наличию защиты от копирования и защиты от записи
Наименьшие привилегии

Характеристики сегмента

- **Для каждого сегмента в регистрах конфигурации можно установить**:
  - Размер Flash сегмента
  - Размер ОЗУ, выделенной сегменту
  - Степень защиты сегмента
    - Standard security
    - High security
    - Без защиты (для General)
  - Защита от стирания/записи
- **Фиксированный приоритет привилегий устанавливается в следующем порядке:**
  - Boot (наивысшие привилегии)
  - Secure
  - General (наименьшие привилегии)

Правила безопасности

- **При выполнении кода в сегментах должны соблюдаться следующие правила:**
  - Чтение/запись ОЗУ/Flash допускается внутри сегмента или в сегмент с более низкими привилегиями, если его степень защиты - Standard
  - Переход на любой адрес допускается внутри сегмента, в сегмент General, или в любой другой сегмент, если его степень защиты - Standard
  - Переход на специальные выделенные адреса допускается в сегменты Boot и Security, если их степень защиты – High
- **Все остальные операции – запрещены. При попытке их выполнения возникает системное исключение.**

Выделенные адреса

- Если Boot или General сегмент имеет степень защиты High Security, то первые несколько ячеек сегмента зарезервированы для специальных адресов
- По специальным адресам должны располагаться вектора вызова функций, находящихся внутри сегмента
- Переход из других сегментов возможен только по этим выделенным адресам.

Обработка прерываний

- Если программа выполняется в Boot или General сегменте и возникает любое прерывание, то управление передается по фиксированному адресу внутри этого сегмента
- При необходимости, обработчик прерывания может определить источник прерывания и передать управление обработчику данного прерывание в этом же, или другом сегменте.
- При передаче управления правила безопасности должны соблюдаться

Программирование и отладка

- Программирование и отладка проводятся для каждого сегмента индивидуально
- При использовании программатора или отладчика необходимо указать с каким сегментом идет работа
- При программировании сегмента, все сегменты с более низкими привилегиями стираются.
- При отладке сегмента, доступ к коду и ОЗУ более привилегированных сегментов запрещен

Поддержка компилятора

- **В компиляторе XC16 имеется поддержка работы с CodeGuard**
  - Для размещения функции в Boot или Security сегментах используются атрибуты:

    __attribute__((boot)), __attribute__((security))

```
void __attribute__((security)) security_func(void);
```

  - Для размещения функции в Boot или Security сегментах, переход на которую возможен через специальный адрес используются атрибуты:

    __attribute__((boot(n))), __attribute__((security(n)))

```
void __attribute__((boot(3))) boot_func(void);
```

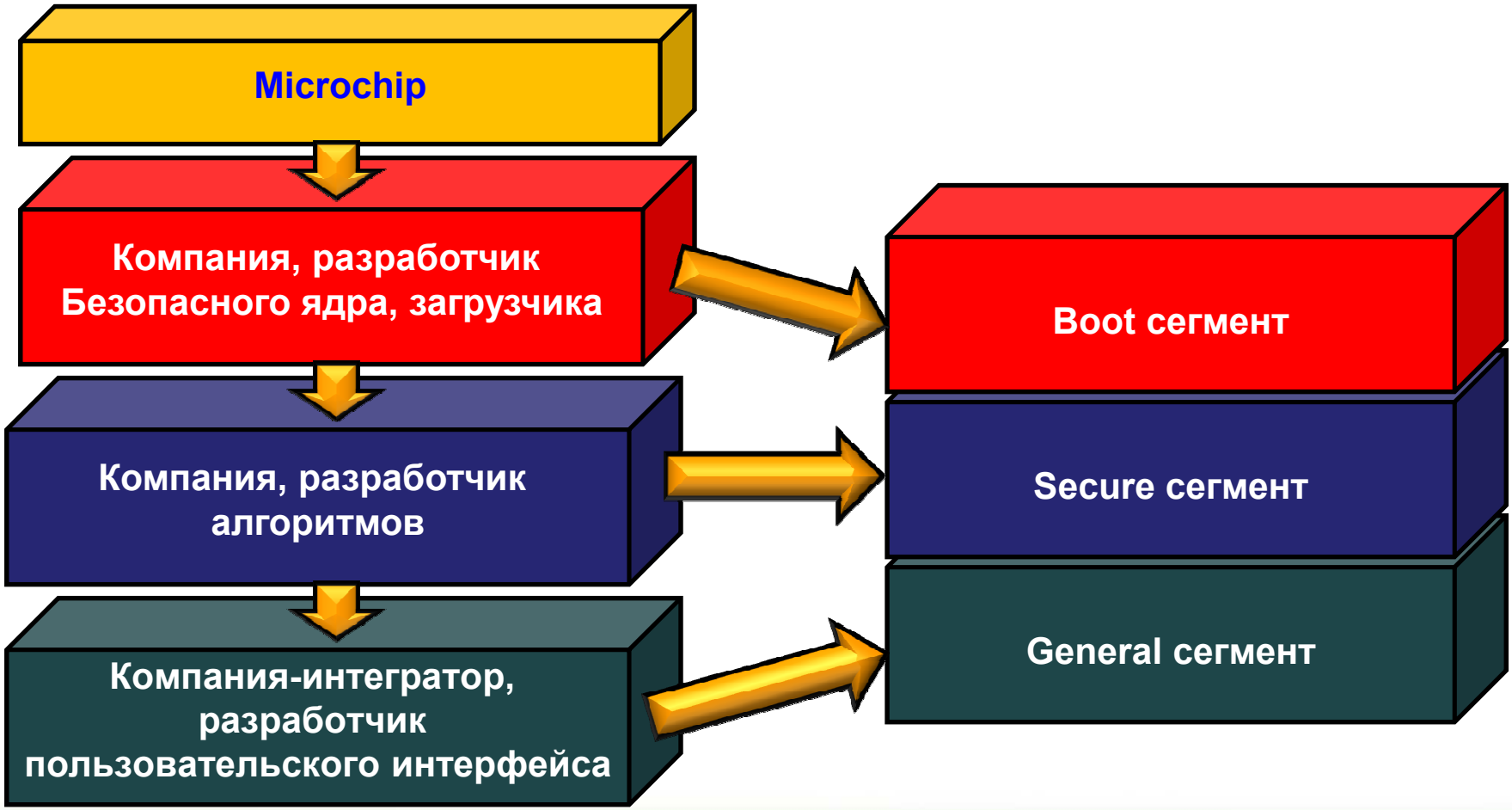
  - Для размещения обработчика сегментного прерывания в Boot или Security сегментах используются атрибуты:

    __attribute__((boot,interrupt)),
    __attribute__((security,interrupt))

```
void __attribute__((boot,interrupt)) bint(void);
```

Пример использования

Путь микроконтроллера



Готовые решения

Готовые загрузчики от Microchip

- Microchip предлагает большое количество разнообразных готовых решений для загрузчиков
- Решения Microchip могут использоваться «как есть» или стать основой для начала разработки собственного загрузчика
- Все решения содержат:
  - Проект загрузчика
  - Проект тестового приложения
  - Программа для PC для загрузки программы

Примеры готовых загрузчиков от Microchip

- AN 851 (PIC16, PIC18 по UART)
- AN 1302 (PIC16 по I^2C)
- AN 1310 (PIC12, PIC16, PIC18 по UART)
- AN 1094 (dsPIC30/33F, PIC24F/24H по UART)
- AN 1157 (PIC24F по UART)
- AN 1388 (PIC32 по UART, USB HID, Ethernet, SD card, USB flash)
- MAL USB Device-Bootloaders (PIC18, PIC24F по USB HID, USB MCHPUSB)
- MAL USB Host-Bootloaders (PIC24F, PIC32 с USB Flash)
- MAL InternetBootloader (PIC18 по Ethernet)
- MAL Android Web Bootloader (PIC24, PIC32 через Андроид- из файловой системы или Web)

Спасибо за внимание !!!