# **Code Analysis for the Flash API Drivers**

## 1. 综述

g\_bootloaderTree 是Kinetis bootloader中一个非常重要的全局结构体变量，它里面包含了很多内容，比如Flash API Driver，AES Driver等，它的地址通常被保存在某一个固定的地方，比如ROM的0x1C00001C处，Flash的0x1C处， 用户可以通过该地址来直接访问g\_bootloaderTree 里面的driver，从而减少了重新编写driver的工作量。

当前ROM 中有大约21个Flash API Driver，本文旨在对当前ROM 里的Flash API Driver 进行源码分析，并给出如何使用的例子。

## 2. Flash API Driver 源码分析

2.1 flash\_init

2.1.1 函数的定义及注解

status\_t flash\_init (flash\_driver\_t \*config)

{

uint32\_t flashDensity;

if (config == NULL)

{

return kStatus\_FLASH\_InvalidArgument;

}

/\* calculate the flash density from SIM\_FCFG1.PFSIZE \*/

uint8\_t pfsize = (SIM->FCFG1 & SIM\_FCFG1\_PFSIZE\_MASK) >> SIM\_FCFG1\_PFSIZE\_SHIFT;

if (pfsize == 0xf)

{

flashDensity=FSL\_FEATURE\_FLASH\_PFLASH\_BLOCK\_COUNT\* FSL\_FEATURE\_FLASH\_PFLASH\_BLOCK\_SIZE;

}

else

{

flashDensity = ((uint32\_t)kPFlashDensities[pfsize]) << 10;

}

/\* fill out a few of the structure members \*/

config->PFlashBlockBase = FSL\_FEATURE\_FLASH\_PFLASH\_START\_ADDRESS;

config->PFlashTotalSize = flashDensity;

config->PFlashBlockCount = FSL\_FEATURE\_FLASH\_PFLASH\_BLOCK\_COUNT;

config->PFlashSectorSize = FSL\_FEATURE\_FLASH\_PFLASH\_BLOCK\_SECTOR\_SIZE;

#if defined(FSL\_FEATURE\_FLASH\_HAS\_ACCESS\_CONTROL) && FSL\_FEATURE\_FLASH\_HAS\_ACCESS\_CONTROL

config->PFlashAccessSegmentSize = kFLASH\_AccessSegmentBase << FTFx->FACSS;

config->PFlashAccessSegmentCount = FTFx->FACSN;

#else

config->PFlashAccessSegmentSize = 0;

config->PFlashAccessSegmentCount = 0;

#endif /\* FSL\_FEATURE\_FLASH\_HAS\_ACCESS\_CONTROL \*/

config->PFlashCallback = NULL;

/\* copy required flash commands to RAM \*/

#if (FLASH\_DRIVER\_IS\_FLASH\_RESIDENT && !FLASH\_DRIVER\_IS\_EXPORTED)

if (kStatus\_FLASH\_Success != flash\_check\_execute\_in\_ram\_function\_info(config))

{

s\_flashExecuteInRamFunctionInfo.activeFunctionCount = 0;

s\_flashExecuteInRamFunctionInfo.flashRunCommand = s\_flashRunCommand;

s\_flashExecuteInRamFunctionInfo.flashCacheClearCommand = s\_flashCacheClearCommand;

config->flashExecuteInRamFunctionInfo = &s\_flashExecuteInRamFunctionInfo.activeFunctionCount;

FLASH\_PrepareExecuteInRamFunctions(config);

}

#endif

config->FlexRAMBlockBase = FSL\_FEATURE\_FLASH\_FLEX\_RAM\_START\_ADDRESS;

config->FlexRAMTotalSize = FSL\_FEATURE\_FLASH\_FLEX\_RAM\_SIZE;

#if FLASH\_SSD\_IS\_FLEXNVM\_ENABLED

{

status\_t returnCode;

config->DFlashBlockBase = FSL\_FEATURE\_FLASH\_FLEX\_NVM\_START\_ADDRESS;

returnCode = flash\_update\_flexnvm\_memory\_partition\_status(config);

if (returnCode != kStatus\_FLASH\_Success)

{

return returnCode;

}

}

#endif

return kStatus\_FLASH\_Success;

}

2.1.2 函数的功能

根据flash特性填充flash属性参数，即填充flash\_driver\_t结构体

@参数 config：指向flash\_driver\_t结构体的指针。

@返回：错误代码或kStatus\_FLASH\_Success。

2.1.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

printf(“flash init success!”);

}

2.2 flash\_erase\_all\_unsecure

2.2.1 函数的定义及注解

#if defined(FSL\_FEATURE\_FLASH\_HAS\_ERASE\_ALL\_BLOCKS\_UNSECURE\_CMD) && FSL\_FEATURE\_FLASH\_HAS\_ERASE\_ALL\_BLOCKS\_UNSECURE\_CMD

status\_t flash\_erase\_all\_unsecure(flash\_driver\_t\*config, uint32\_t key)

{

status\_t returnCode;

if (config == NULL)

{

return kStatus\_FLASH\_InvalidArgument;

}

//传送erase all 命令0x49到FCCOB0寄存器

kFCCOBx[0] = BYTES\_JOIN\_TO\_WORD\_1\_3(FTFx\_ERASE\_ALL\_BLOCK\_UNSECURE, 0xFFFFFFU);

//验证erase key是否为”k f e k”

returnCode = flash\_check\_user\_key(key);

if (returnCode)

{

return returnCode;

}

//FSTAT 中CCIF位置1执行命令

returnCode = flash\_command\_sequence(config);

flash\_cache\_clear(config);

#if FLASH\_SSD\_IS\_FLEXNVM\_ENABLED

/\* Data flash IFR will be erased by erase all unsecure command, so we need to

\* update FlexNVM memory partition status synchronously \*/

if (returnCode == kStatus\_FLASH\_Success)

{

returnCode = flash\_update\_flexnvm\_memory\_partition\_status(config);

}

#endif

return returnCode;

}

#endif /\* FSL\_FEATURE\_FLASH\_HAS\_ERASE\_ALL\_BLOCKS\_UNSECURE\_CMD \*/

2.2.2 函数的功能

擦除flash 跟erasable IFR同时让flash处于unsecure状态

@参数 config：指向flash\_driver\_t结构体的指针。

@参数 key：flash erase key为 “k f e k”

@返回：错误代码或kStatus\_FLASH\_Success。

2.2.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

#define FOUR\_CHAR\_CODE(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))

enum \_flash\_driver\_api\_keys

{

kFLASH\_ApiEraseKey = FOUR\_CHAR\_CODE('k', 'f', 'e', 'k')

};

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

If([s\_flashInterface](#s_flashInterface)->flash\_erase\_all\_unsecure(&driver,kFLASH\_ApiEraseKey)== kStatus\_Succcess)

{

printf(“flash erase all unsecure success”);

}

}

2.3 flash\_erase\_all

2.3.1 函数的定义及注解

status\_t flash\_erase\_all (flash\_driver\_t \*config, uint32\_t key)

{

status\_t returnCode;

if (config == NULL)

{

return kStatus\_FLASH\_InvalidArgument;

}

//传递flash erase all命令0x44到 FCCOB0

kFCCOBx[0] = BYTES\_JOIN\_TO\_WORD\_1\_3(FTFx\_ERASE\_ALL\_BLOCK, 0xFFFFFFU);

//验证erase key是否为”k f e k”

returnCode = flash\_check\_user\_key(key);

if (returnCode)

{

return returnCode;

}

//FSTAT 中CCIF位置1执行命令

returnCode = flash\_command\_sequence(config);

flash\_cache\_clear(config);

#if FLASH\_SSD\_IS\_FLEXNVM\_ENABLED

/\* Data flash IFR will be erased by erase all command, so we need to

\* update FlexNVM memory partition status synchronously \*/

if (returnCode == kStatus\_FLASH\_Success)

{

returnCode = flash\_update\_flexnvm\_memory\_partition\_status(config);

}

#endif

return returnCode;

}

2.3.2 函数的功能

擦除flash跟erasable IFR

@参数 config：指向flash\_driver\_t结构体的指针。

@参数 key：flash erase key为 “k f e k”

@返回：错误代码或kStatus\_FLASH\_Success。

2.3.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

#define FOUR\_CHAR\_CODE(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))

enum \_flash\_driver\_api\_keys

{

kFLASH\_ApiEraseKey = FOUR\_CHAR\_CODE('k', 'f', 'e', 'k')

};

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

if([s\_flashInterface](#s_flashInterface)->flash\_erase\_all (&driver,kFLASH\_ApiEraseKey)== kStatus\_Succcess)

{

printf(“flash erase all success”);

}

}

2.4 flash\_verify\_erase\_all

2.5 flash\_erase

2.5.1 函数的定义及注解

status\_t flash\_erase(flash\_driver\_t \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)

{

uint32\_t sectorSize;

flash\_operation\_config\_t flashInfo;

uint32\_t endAddress; /\* storing end address \*/

uint32\_t numberOfSectors; /\* number of sectors calculated by endAddress \*/

status\_t returnCode;

//将block size 、segment size保存到结构体flashInfo中

flash\_get\_matched\_operation\_info(config, start, &flashInfo);

//检查start address是否符合要求

returnCode = flash\_check\_range(config, start, lengthInBytes, flashInfo.sectorCmdAddressAligment);

if (returnCode)

{

return returnCode;

}

start = flashInfo.convertedAddress;

sectorSize = flashInfo.activeSectorSize;

//重新计算end address，因为擦除以sector为单位

endAddress = start + lengthInBytes - 1;

if (endAddress % sectorSize)

{

numberOfSectors = endAddress / sectorSize + 1;

endAddress = numberOfSectors \* sectorSize - 1;

}

/\* the start address will increment to the next sector address

\* until it reaches the endAdddress \*/

while (start <= endAddress)

{

//输入命令0x09跟地址到FCCOB寄存器

kFCCOBx[0] = BYTES\_JOIN\_TO\_WORD\_1\_3(FTFx\_ERASE\_SECTOR, start);

/\* Validate the user key \*/

returnCode = flash\_check\_user\_key(key);

if (returnCode)

{

return returnCode;

}

//执行命令

returnCode = flash\_command\_sequence(config);

//代码里面这个函数为NULL，所以不能执行

if (config->PFlashCallback)

{

config->PFlashCallback();

}

/\* checking the success of command execution \*/

if (kStatus\_FLASH\_Success != returnCode)

{

break;

}

else

{

/\* Increment to the next sector \*/

start += sectorSize;

}

}

flash\_cache\_clear(config);

return (returnCode);

}

2.5.2 函数的功能

以sector为单位擦除flash

@参数 config：指向flash\_driver\_t结构体的指针。

@参数start ：erase的起始地址

@参数lengthInBytes ：erase的长度，擦除以sector为单位

@参数key ： 必须为“k f e k”

@返回：错误代码或kStatus\_FLASH\_Success。

2.5.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

if([s\_flashInterface](#s_flashInterface)->flash\_erase(&driver, 0x0, 0x1000, kFLASH\_ApiEraseKey)== kStatus\_Succcess)

{

printf(“flash erase success”);

}

}

2.6 flash\_verify\_erase

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.7 flash\_program

2.2.1 函数的定义及注解

status\_t flash\_program(flash\_config\_t \*config, uint32\_t start, uint32\_t \*src, uint32\_t lengthInBytes)

{

status\_t returnCode;

flash\_operation\_config\_t flashInfo;

if (src == NULL)

{

return kStatus\_FLASH\_InvalidArgument;

}

flash\_get\_matched\_operation\_info(config, start, &flashInfo);

/\* Check the supplied address range. \*/

returnCode = flash\_check\_range(config, start, lengthInBytes, flashInfo.blockWriteUnitSize);

if (returnCode)

{

return returnCode;

}

start = flashInfo.convertedAddress;

while (lengthInBytes > 0)

{

//FCCOB4-B里面写入待操作的数据

kFCCOBx[1] = \*src++;

if (4 == flashInfo.blockWriteUnitSize)

{

//FCCOB0-3里面写入命令跟地址

kFCCOBx[0] = BYTES\_JOIN\_TO\_WORD\_1\_3(FTFx\_PROGRAM\_LONGWORD, start);

}

else if (8 == flashInfo.blockWriteUnitSize)

{

kFCCOBx[2] = \*src++;

kFCCOBx[0] = BYTES\_JOIN\_TO\_WORD\_1\_3(FTFx\_PROGRAM\_PHRASE, start);

}

else

{

}

//执行命令

returnCode = flash\_command\_sequence(config);

/\* calling flash callback function if it is available \*/

if (config->PFlashCallback)

{

config->PFlashCallback();

}

/\* checking for the success of command execution \*/

if (kStatus\_FLASH\_Success != returnCode)

{

break;

}

else

{

/\* update start address for next iteration \*/

start += flashInfo.blockWriteUnitSize;

/\* update lengthInBytes for next iteration \*/

lengthInBytes -= flashInfo.blockWriteUnitSize;

}

}

flash\_cache\_clear(config);

return (returnCode);

}

2.2.2 函数的功能

向flash写入数据

@参数 config：指向flash\_driver\_t结构体的指针。

@参数start： 待写入数据的起始地址

@参数 src ：指向uint32\_t类型的指针

@参数lengthbyte ：待写入数据的长度

@返回：错误代码或者kStatus\_FLASH\_Success。

2.2.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

const uint32\_t buffer[2] = {0x01234567, 0x89abcdef};

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

If([s\_flashInterface](#s_flashInterface)-> flash\_program (&driver, 0x0, &buffer[0], 0x16)== kStatus\_Succcess)

{

printf(“flash program success”);

}

}

2.8 flash\_verify\_program

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.9 flash\_get\_security\_state

2.9.1 函数的定义及注解

status\_t FLASH\_GetSecurityState(flash\_driver\_t \*config, [flash\_security\_state\_t](#flash_security_state_t) \*state)

{

/\* store data read from flash register \*/

uint8\_t registerValue;

if ((config == NULL) || (state == NULL))

{

return kStatus\_FLASH\_InvalidArgument;

}

//获取flash的状态

registerValue = FTFx->FSEC;

/\* check the status of the flash security bits in the security register \*/

if (FLASH\_SECURITY\_STATE\_UNSECURED == (registerValue & FTFx\_FSEC\_SEC\_MASK))

{

/\* Flash in unsecured state \*/

\*state = kFLASH\_SecurityStateNotSecure;

}

else

{

/\* Flash in secured state

\* check for backdoor key security enable bit \*/

if (FLASH\_SECURITY\_STATE\_KEYEN == (registerValue & FTFx\_FSEC\_KEYEN\_MASK))

{

/\* Backdoor key security enabled \*/

\*state = kFLASH\_SecurityStateBackdoorEnabled;

}

else

{

/\* Backdoor key security disabled \*/

\*state = kFLASH\_SecurityStateBackdoorDisabled;

}

}

return (kStatus\_FLASH\_Success);

}

2.9.2 函数的功能

返回flash的状态，secure、unsecure、backdoorkeyEnabled、backdoorkeyDisabled

@参数 config：指向flash\_driver\_t结构体的指针。

@参数state ：指向枚举类型[flash\_security\_state\_t](#flash_security_state_t) 的指针

@返回：kStatus\_FLASH\_Success。

2.9.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

[flash\_security\_state\_t](#flash_security_state_t) state;

uint32\_t flashState;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

flashState = [s\_flashInterface](#s_flashInterface)->flash\_get\_security\_state(&driver, &state);

if(flashState == kStatus\_Succcess)

{

printf(“flash erase success”);

}

switch(state)

{

case (kFLASH\_SecurityStateNotSecure):

printf(“flash is not secure”);

break;

case (kFLASH\_SecurityStateBackdoorEnabled):

printf(“backdoor key enabled”);

break;

case (kFLASH\_SecurityStateBackdoorEnabled):

printf(“backdoor key disabled”);

break;

default:

break;

}

}

2.10 flash\_security\_bypass

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.11 flash\_get\_property

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.12 flash\_register\_callback

2.12.1 函数的定义及注解

status\_t flash\_register\_callback ([flash\_driver\_t](#flash_driver_t) \*driver, [flash\_callback\_t](#flash_driver_t) callback)

{

if (driver == NULL)

{

return kStatus\_InvalidArgument;

}

driver->PFlashCallback = callback;

return kStatus\_Success;

}

2.12.2 函数的功能

为flash API 注册一个不带参数返回值为空类型的callback 函数。

@参数 driver：包含flash 相关参数的结构体指针。

@参数 callback：不带参数返回值为空类型的函数指针。

@返回：错误代码或 kStatus\_Success。

2.12.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

[s\_flashInterface](#s_flashInterface)-> flash\_register\_callback(&driver, callback);

driver.PFlashCallback();

}

void callback(void)

{

printf("Flash API callback function executed!\r\n");

}

2.13 flash\_program\_once

2.13.1 函数的定义及注解

status\_t flash\_program\_once([flash\_driver\_t](#flash_driver_t)\* driver, uint32\_t index, uint32\_t \*src, uint32\_t lengthInBytes)

{

if (src == NULL)

{

return kStatus\_InvalidArgument;

}

//检查输入的index和LengthInBytes是否有效并匹配，当前IFR支持4字节和8字节。

status\_t returnCode = flash\_check\_access\_ifr\_range(driver, index, lengthInBytes);

if (kStatus\_Success != returnCode)

{

return returnCode;

}

//将program once的指令码写入FCCOB0寄存器，通知flash controller接下来要执行program once指令。

FTFx\_WR\_FCCOBx(FTFx, 0, FTFx\_PROGRAM\_ONCE);

//当执行program once指令时，FCCOB1寄存器用来保存program once record index。因此此处需要将program once record index写入该寄存器。

FTFx\_WR\_FCCOBx(FTFx, 1, index);

//FCCOB4~FCCOBB共八个寄存器用来存放要写入的数据，此处先写入4字节到FCCOB4~FCCOB7 四个寄存器中，FCCOB7存放低字节，FCCOB4存放高字节。

kFCCOBx[1] = \*src;

// Note: Have to separate the first index from the rest if it equals 0 to avoid a pointless comparison of unsigned int to 0 compiler warning

if (((index == FLASH\_PROGRAM\_ONCE\_MIN\_ID\_8BYTES) ||

//Range check

((index >= FLASH\_PROGRAM\_ONCE\_MIN\_ID\_8BYTES + 1) &&

(index <= FLASH\_PROGRAM\_ONCE\_MAX\_ID\_8BYTES))) &&

(lengthInBytes == 8))

{

//如果是8字节的IFR则再写入4字节到FCCOB8~FCCOBB 四个寄存器中。FCCOB7存放低字节，FCCOB4存放高字节。

kFCCOBx[2] = \*(src + 1);

}

//清CCIF位触发执行program once指令

returnCode = flash\_command\_sequence(driver);

// 清除flash缓存

flash\_cache\_clear(driver);

return returnCode;

}

2.13.2 函数的功能

写IFR寄存器。

@参数 driver：包含flash 相关参数的结构体指针。

@参数 index：IFR寄存器的索引号。

@参数src：指向4字节的地址，该地址中存放要写入IFR寄存器的值。比如src可以是一个uint32\_t类型的数组的首地址。

@参数 lengthInBytes：IFR寄存器对应的字节数。

@返回：错误代码 或 kStatus\_Success。

2.13.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

uint32\_t src[] = {0x12345678, 0xabcdefef};

[s\_flashInterface](#s_flashInterface)-> flash\_program\_once(&driver, 0x30, src, 4);//将0x12345678写入IFR 0x30。

}

2.14 flash\_read\_once

2.14.1 函数的定义及注解

status\_t flash\_read\_once ([flash\_driver\_t](#flash_driver_t)\* driver, uint32\_t index, uint32\_t \*dst, uint32\_t lengthInBytes)

{

if (dst == NULL)

{

return kStatus\_InvalidArgument;

}

//检查输入的index和LengthInBytes是否有效并匹配，当前IFR支持4字节和8字节。

status\_t returnCode = flash\_check\_access\_ifr\_range(driver, index, lengthInBytes);

if (kStatus\_Success != returnCode)

{

return returnCode;

}

// 将program once的指令码写入FCCOB0寄存器，通知flash controller接下来要执行read once指令。

FTFx\_WR\_FCCOBx(FTFx, 0, FTFx\_READ\_ONCE);

// 将index 写入FCCOB1寄存器，通知flash controller接下来要读取的IFR寄存器。

FTFx\_WR\_FCCOBx(FTFx, 1, index);

//清CCIF位触发执行read once指令

returnCode = flash\_command\_sequence(driver);

if (kStatus\_Success == returnCode)

{

//read once 执行成功后先从FCCOB4~FCCOB7 四个寄存器中读取4字节数据。FCCOB7存放低字节，FCCOB4存放高字节。

\*dst = kFCCOBx[1];

// Note: Have to seperate the first index from the rest if it equals 0

// to avoid a pointless comparison of unsigned int to 0 compiler warning

if (((index == FLASH\_PROGRAM\_ONCE\_MIN\_ID\_8BYTES) ||

//Range check

((index >= FLASH\_PROGRAM\_ONCE\_MIN\_ID\_8BYTES + 1) &&

(index <= FLASH\_PROGRAM\_ONCE\_MAX\_ID\_8BYTES))) &&

(lengthInBytes == 8))

{

//如果是8字节IFR，则再从FCCOB8~FCCOBB 四个寄存器中读取4字节数据。FCCOBB存放低字节，FCCOB8存放高字节。

\*(dst + 1) = kFCCOBx[2];

}

}

return returnCode;

}

2.14.2 函数的功能

读IFR寄存器。

@参数 driver：包含flash 相关参数的结构体指针。

@参数 index：IFR寄存器的索引号。

@参数dst：指向4字节的地址，该地址中存放从IFR寄存器中读出来的值。比如dst可以是一个uint32\_t类型的数组的首地址。

@参数 lengthInBytes：IFR寄存器对应的字节数。

@返回：错误代码 或 kStatus\_Success。

2.14.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

uint32\_t src[] = {0x12345678, 0xabcdefef};

[s\_flashInterface](#s_flashInterface)-> flash\_program\_once(&driver, 0x30, src, 4);//将0x12345678写入IFR 0x30。

uint32\_t dst[] = {0, 0};

[s\_flashInterface](#s_flashInterface)-> flash\_read\_once(&driver, 0x30, dst, 4); //读4字节IFR寄存器

printf("The result of flash read once: 0x%x, 0x%x\r\n", \*dst, \*(dst+1));

[s\_flashInterface](#s_flashInterface)-> flash\_read\_once(&driver, 0x10, dst, 8); //读8字节IFR寄存器

printf("The result of flash read once: 0x%x, 0x%x\r\n", \*dst, \*(dst+1));

}

2.15 flash\_read\_resource

2.15.1 函数的定义及注解

status\_t flash\_read\_resource([flash\_driver\_t](#flash_driver_t) \*driver, uint32\_t start, uint32\_t \*dst, uint32\_t lengthInBytes, [flash\_read\_resource\_option\_t](#flash_read_resource_option_t) option)

{

if (driver == NULL || dst == NULL)

{

return kStatus\_InvalidArgument;

}

//检查输入的三个参数是否有效，如果无效返回kStatus\_InvalidArgument，否则返回成功。

status\_t returnCode = flash\_check\_resource\_range(start, lengthInBytes, option);

if (returnCode != kStatus\_Success)

{

return returnCode;

}

//循环读取多个字节数据

while(lengthInBytes > 0)

{

//将起始地址写入FCCOB3~FCCOB1三个寄存器中。FCCOB3存放低字节，FCCOB1存放次高字节。注意，下一条指令不能放在该条指令的前面，因为这里的赋值会覆盖FCCOB0寄存器。

kFCCOBx[0] = start;

//将read resource 的指令码写入FCCOB0寄存器中，通知flash控制器下面要执行read resource 指令。

FTFx\_WR\_FCCOBx(FTFx, 0, FTFx\_READ\_RESOURCE);

//read resource 可读取两块地方的数据，具体读取哪个取决与option参数，0 表示读取PFlash 0 IFR， 1 表示读取Version ID，详见芯片数据手册 ”Read Resource Select Codes” 表格。

#if (FSL\_FEATURE\_FLASH\_PFLASH\_RESOURCE\_CMD\_ADDRESS\_ALIGMENT == 8)

//option参数要写入FCCOB4 或 FCCOB8 寄存器，具体写入哪个可参见芯片收据手册 “Read Resource Command FCCOB Requirements” 表格，且该处的宏值也可从该表中找到。

FTFx\_WR\_FCCOBx(FTFx, 4, option);

#else

FTFx\_WR\_FCCOBx(FTFx, 8, option);

#endif

//清CCIF位触发并执行read resource指令

returnCode = flash\_command\_sequence(driver);

if (kStatus\_Success != returnCode)

{

break;

}

//如果指令执行失败则退出，否则从FCCOB7~FCCOB4读取4字节数据

\*dst++ = kFCCOBx[1];

#if (FSL\_FEATURE\_FLASH\_PFLASH\_RESOURCE\_CMD\_ADDRESS\_ALIGMENT > 4)

//如果该宏的值大于4，则再从FCCOBB~FCCOB8读取另外4字节数据，即一次循环读取8字节数据，否则一次循环读取4字节数据。

\*dst++ = kFCCOBx[2];

#endif

//更新起始地址，此处的步长值与上面的宏值相等。

start += kFlashReadResource\_UnitInBytes;

//更新要读取的字节长度参数，以便于下次循环。

lengthInBytes -= kFlashReadResource\_UnitInBytes;

}

return (returnCode);

}

2.15.2 函数的功能

读取芯片的PFlash 0 IFR或Version ID 数据。

@参数 driver：包含flash 相关参数的结构体指针。

@参数 start：要读取的数据的起始地址。该参数有对齐要求，具体参见数据手册，如果不对齐则函数直接返回kStatus\_InvalidArgument。

@参数dst：指向4字节类型数据的地址，从该地址开始处依次存放最终读取出来的数据。比如dst可以是一个uint32\_t类型的数组的首地址。

@参数 lengthInBytes：要读取的数据长度。该参数也有对齐要求，对齐情况跟start参数一样，如果不对齐则函数直接返回kStatus\_InvalidArgument。此外，如果该参数值大于实际数据的长度，也会直接返回kStatus\_InvalidArgument，例如，实际芯片的PFlash 0 IFR 大小只有256 字节，如果该参数取512 （>256），则函数返回kStatus\_InvalidArgument。

@返回：错误代码 或 kStatus\_Success。

2.15.3 例子

[flash\_driver\_t](#flash_driver_t) driver;

if ([s\_flashInterface](#s_flashInterface)->flash\_init(&driver) == kStatus\_Success)

{

uint32\_t dst[256];

uint32\_t start, lengthInBytes, i;

start = 128;

lengthInBytes = 128;

// 读取芯片PFlash 0 IFR数据

[s\_flashInterface](#s_flashInterface)-> flash\_read\_resource(&driver, start, dst, lengthInBytes, 0);

for (i = 0; i < lengthInBytes/sizeof(uint32\_t); i++)

{

printf ("IFR data in address 0x%x: 0x%x\r\n", start+i\*sizeof(uint32\_t), \*(dst+i));

}

// 读取芯片的Version ID

start = 0;

lengthInBytes = 8;

// 读取芯片PFlash 0 IFR数据

[s\_flashInterface](#s_flashInterface)-> flash\_read\_resource(&driver, start, dst, lengthInBytes, 1);

for (i = 0; i < lengthInBytes/sizeof(uint32\_t); i++)

{

printf ("IFR data in address 0x%x: 0x%x\r\n", start+i\*sizeof(uint32\_t), \*(dst+i));

}

}

2.16 flash\_prepare\_execute\_in\_ram\_functions

2.16.1 函数的定义及注解

status\_t flash\_prepare\_execute\_in\_ram\_functions(flash\_driver\_t \* driver)

{

if (driver == NULL)

{

return kStatus\_InvalidArgument;

}

//将一个uint32\_t类型的指针转强制化成flash\_execute\_in\_ram\_function\_info\_t 结构体类型指针。

[flash\_execute\_in\_ram\_function\_info\_t](#flash_execute_in_ram_function_config_t) \*flashExecuteInRamFunctionInfo = \

([flash\_execute\_in\_ram\_function\_info\_t](#flash_execute_in_ram_function_config_t) \*)driver->flashExecuteInRamFunctionInfo;

//将flash\_run\_command 函数的可执行二进制代码拷贝到flashExecuteInRamFunctionInfo-> flashRunCommand 所指地址的开始处，然后将该地址强制转化成一个函数指针，其类型和flash\_run\_command函数相同。

//注意：flash\_run\_command 函数主要有两个操作，一是清CCIF标志位以便触发相应的flash 指令执行，二是等待CCIF位置位，即等待flash指令执行完成。

copy\_flash\_run\_command(flashExecuteInRamFunctionInfo->flashRunCommand);

//将flash\_cache\_clear\_command 函数的可执行码拷贝到flashExecuteInRamFunctionInfo-> flashCacheClearCommand 所指地址的开始处，然后将该地址强制转化成一个函数指针，其类型和flash\_cache\_clear\_command函数相同。

copy\_flash\_cache\_clear\_command (

flashExecuteInRamFunctionInfo->flashCacheClearCommand);

//给 flashExecuteInRamFunctionInfo->activeFunctionCount 赋值，表示将要在RAM中执行的函数个数。

flashExecuteInRamFunctionInfo->activeFunctionCount = \

FlashExecuteInRamFunction\_TotalNum;

return kStatus\_Success;

}

2.16.2 函数的功能

将flash\_run\_command和flash\_cache\_clear\_command的函数可执行码分别拷贝到RAM中。注意，该函数在ROM bootloader中不存在，只在flash-resident bootloader中存在，在flash-resident bootloader中这两个函数被拷贝到RAM 运行。

@参数 driver：包含flash 相关参数的结构体指针。

@返回：错误代码 或 kStatus\_Success。

2.16.3 例子

该函数在ROM中不存在，暂无例子。

2.17 flash\_is\_execute\_only

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.18 flash\_erase\_all\_execute\_only\_segments

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.19 flash\_verify\_ erase\_all\_execute\_only\_segments

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.20 flash\_set\_flexram\_function

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

2.21 flash\_program\_section

本发明涉及一种升级软件的方法，特别涉及嵌入式领域bootloader的软件升级方法。

## 3. 附录

3.1 结构体flash\_driver\_t的定义

typedef struct \_flash\_driver {

uint32\_t PFlashBlockBase; /\*!< Base address of the first PFlash block \*/

uint32\_t PFlashTotalSize; /\*!< Size of all combined PFlash block. \*/

uint32\_t PFlashBlockCount; /\*!< Number of PFlash blocks. \*/

uint32\_t PFlashSectorSize; /\*!< Size in bytes of a sector of PFlash. \*/

[flash\_callback\_t](#flash_callback_t) PFlashCallback; /\*!< Callback function for flash API. \*/

uint32\_t PFlashAccessSegmentSize; /\*!< Size in bytes of a access segment of PFlash. \*/

uint32\_t PFlashAccessSegmentCount; /\*!< Number of PFlash access segments. \*/

uint32\_t \*flashExecuteInRamFunctionInfo; /\*!< Info struct of flash execute-in-ram function. \*/

} flash\_driver\_t;

3.2 函数指针flash\_callback\_t的定义

typedef void (\*flash\_callback\_t) (void);

3.3 结构体bootloader\_tree\_t的定义

typedef struct BootloaderTree {

void (\*runBootloader)(void \*arg); //!< Function to start the bootloader executing.

standard\_version\_t version; //!< Bootloader version number.

const char \* copyright; //!< Copyright string.

const bootloader\_context\_t \* runtimeContext; //!< Pointer to the bootloader's runtime context.

const [flash\_driver\_interface\_t](#flash_driver_interface_t) \*flashDriver; //!< Flash driver API.

const aes\_driver\_interface\_t \*aesDriver; //!< AES driver API.

} bootloader\_tree\_t;

3.4 结构体flash\_driver\_interface\_t的定义

typedef struct FlashDriverInterface {

standard\_version\_t version; //!< flash driver API version number.

status\_t (\*flash\_init) ([flash\_driver\_t](#flash_driver_t) \*driver);

status\_t (\*flash\_erase\_all) ([flash\_driver\_t](#flash_driver_t) \*driver, uint32\_t key);

status\_t (\*flash\_erase\_all\_unsecure) ([flash\_driver\_t](#flash_driver_t) \*driver, uint32\_t key);

status\_t (\*flash\_erase) ([flash\_driver\_t](#flash_driver_t) \*driver, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key);

status\_t (\*flash\_program) ([flash\_driver\_t](#flash_driver_t) \*driver,

uint32\_t start,

uint32\_t \* src,

uint32\_t lengthInBytes);

status\_t (\*flash\_get\_security\_state) ([flash\_driver\_t](#flash_driver_t) \*driver, flash\_security\_state\_t \* state);

status\_t (\*flash\_security\_bypass) ([flash\_driver\_t](#flash_driver_t) \*driver, const uint8\_t \* backdoorKey);

status\_t (\*flash\_verify\_erase\_all) ([flash\_driver\_t](#flash_driver_t) \*driver, flash\_margin\_value\_t margin);

status\_t (\*flash\_verify\_erase) ([flash\_driver\_t](#flash_driver_t) \*driver,

uint32\_t start,

uint32\_t lengthInBytes,

flash\_margin\_value\_t margin);

status\_t (\*flash\_verify\_program) ([flash\_driver\_t](#flash_driver_t) \*driver,

uint32\_t start,

uint32\_t lengthInBytes,

const uint8\_t \* expectedData,

flash\_margin\_value\_t margin,

uint32\_t \* failedAddress,

uint32\_t \* failedData);

status\_t (\*flash\_get\_property) ([flash\_driver\_t](#flash_driver_t) \*driver,

flash\_property\_t whichProperty,

uint32\_t \* value);

status\_t (\*flash\_register\_callback) ([flash\_driver\_t](#flash_driver_t) \*driver, [flash\_callback\_t](#flash_callback_t) callback);

status\_t (\*flash\_program\_once) ([flash\_driver\_t](#flash_driver_t) \* driver,

uint32\_t index,

uint32\_t \* src,

uint32\_t lengthInBytes);

status\_t (\*flash\_read\_once) ([flash\_driver\_t](#flash_driver_t) \*driver,

uint32\_t index,

uint32\_t \* dst,

uint32\_t lengthInBytes);

status\_t (\*flash\_read\_resource) ([flash\_driver\_t](#flash_driver_t) \*driver,

uint32\_t start,

uint32\_t \*dst,

uint32\_t lengthInBytes,

flash\_read\_resource\_option\_t option);

status\_t (\*flash\_prepare\_execute\_in\_ram\_functions) ([flash\_driver\_t](#flash_driver_t) \* driver);

} flash\_driver\_interface\_t;

3.5 枚举类型flash\_security\_state\_t的定义

typedef enum \_flash\_security\_state

{

kFLASH\_SecurityStateNotSecure, /\*!< Flash is not secure.\*/

kFLASH\_SecurityStateBackdoorEnabled, /\*!< Flash backdoor is enabled.\*/

kFLASH\_SecurityStateBackdoorDisabled /\*!< Flash backdoor is disabled.\*/

} flash\_security\_state\_t;

3.6 第二部分例子中要用到的变量的定义

#define BOOTLOADER\_TREE\_LOCATION (0x1c00001cul)

#define BOOTLOADER\_API\_TREE\_POINTER \

(\*(bootloader\_tree\_t \*\*) BOOTLOADER\_TREE\_LOCATION)

typedef enum \_flash\_read\_resource\_option

{

kFlashResource\_ProgramIFR = 0,

kFlashResource\_VersionID = 1

} flash\_read\_resource\_option\_t;

static const [flash\_driver\_interface\_t](#flash_driver_interface_t) \*s\_flashInterface;

s\_flashInterface = BOOTLOADER\_API\_TREE\_POINTER->flashDriver;

typedef struct \_flash\_execute\_in\_ram\_function\_config

{

uint32\_t activeFunctionCount; /\*!< Number of available execute-in-ram functions.\*/

uint32\_t \*flashRunCommand; /\*!< execute-in-ram function: flash\_run\_command.\*/

uint32\_t \*flashCacheClearCommand; /\*!< execute-in-ram function: flash\_cache\_clear\_command.\*/

} flash\_execute\_in\_ram\_function\_config\_t;