



STM32 FLASH 模拟 EEPROM 使用和优化

原始文件 ST 官方有例子和文档: AN2594

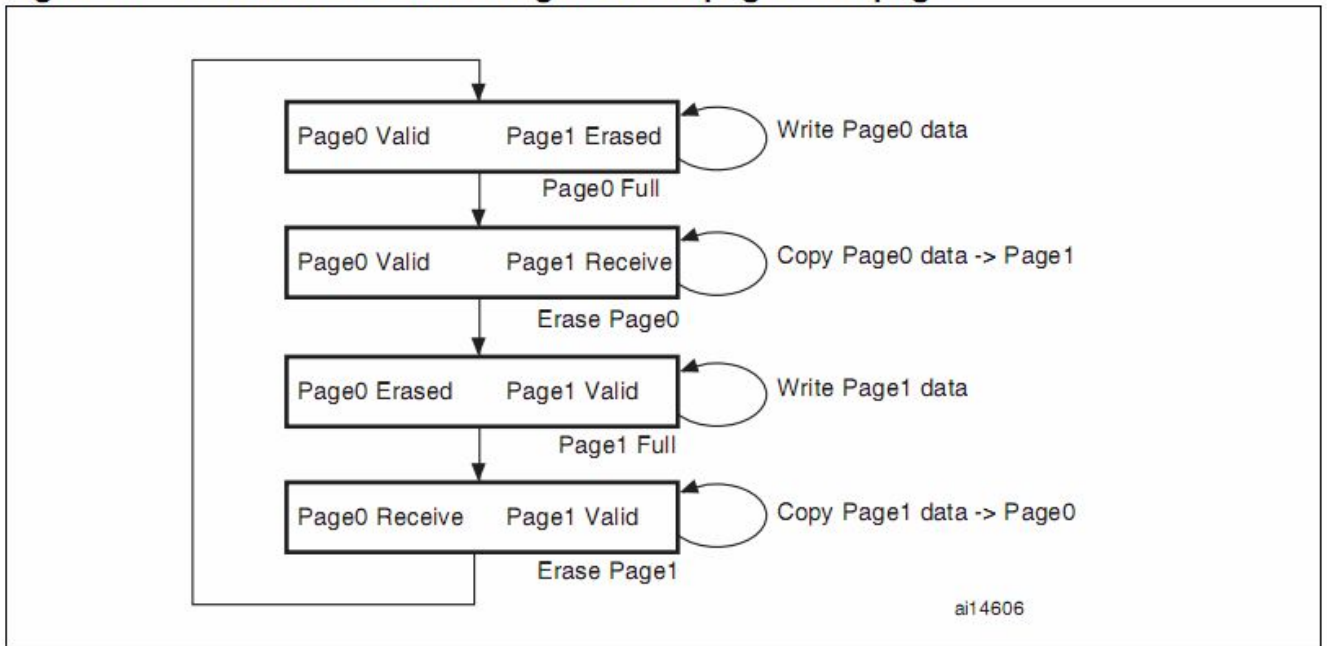
<http://www.st.com/mcu/familiesdocs-110.html>

AN2594	EEPROM emulation in STM32F10x microcontrollers	3	Aug-2009	 
--------	------------------------------------------------	---	----------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

看到不少网上使用官方例子程序不成功的问题, 我估计大概是没有仔细阅读官方文档的原因吧, 也许很多人没理解官方例子的原理。那么下面就详细说明一下原理再说如何优化。

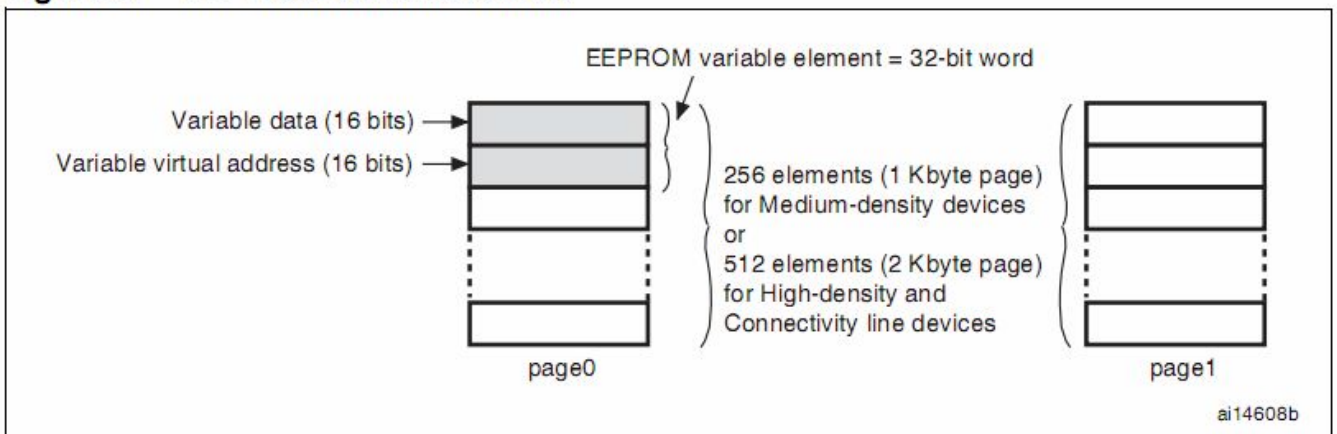
原理如下:

Figure 1. Header status switching between page0 and page1



首先使用 2 页 FLASH 空间, 如果 0 页空间写满数据, 那么把 0 页空间里面的【有效数据】复制到 1 页, 如果 1 页数据满那么把 1 页空间里面的【有效数据】复制到 0 页, 这样循环使用, 当然如果你想增加使用寿命可以增加多页循环, 官方例子只是按 2 页实现的例子。每页前面 4 字节保留, 其中前 2 字节是该页状态标志
下面的图显示数据在 FLASH 中的保存格式:

Figure 2. EEPROM variable format



保存数据是 16 位的, 后面 16 位是该数据的虚拟地址, 注意: 1 个数据有唯 1 个虚拟地址, 地址必须为: 0~0xffff 范围内 (每页将按 4 字节分块, 1 块保存 1 个 16 位数据)。下面继续说明 16 位虚拟地址的作用。

Figure 3. Data update flow

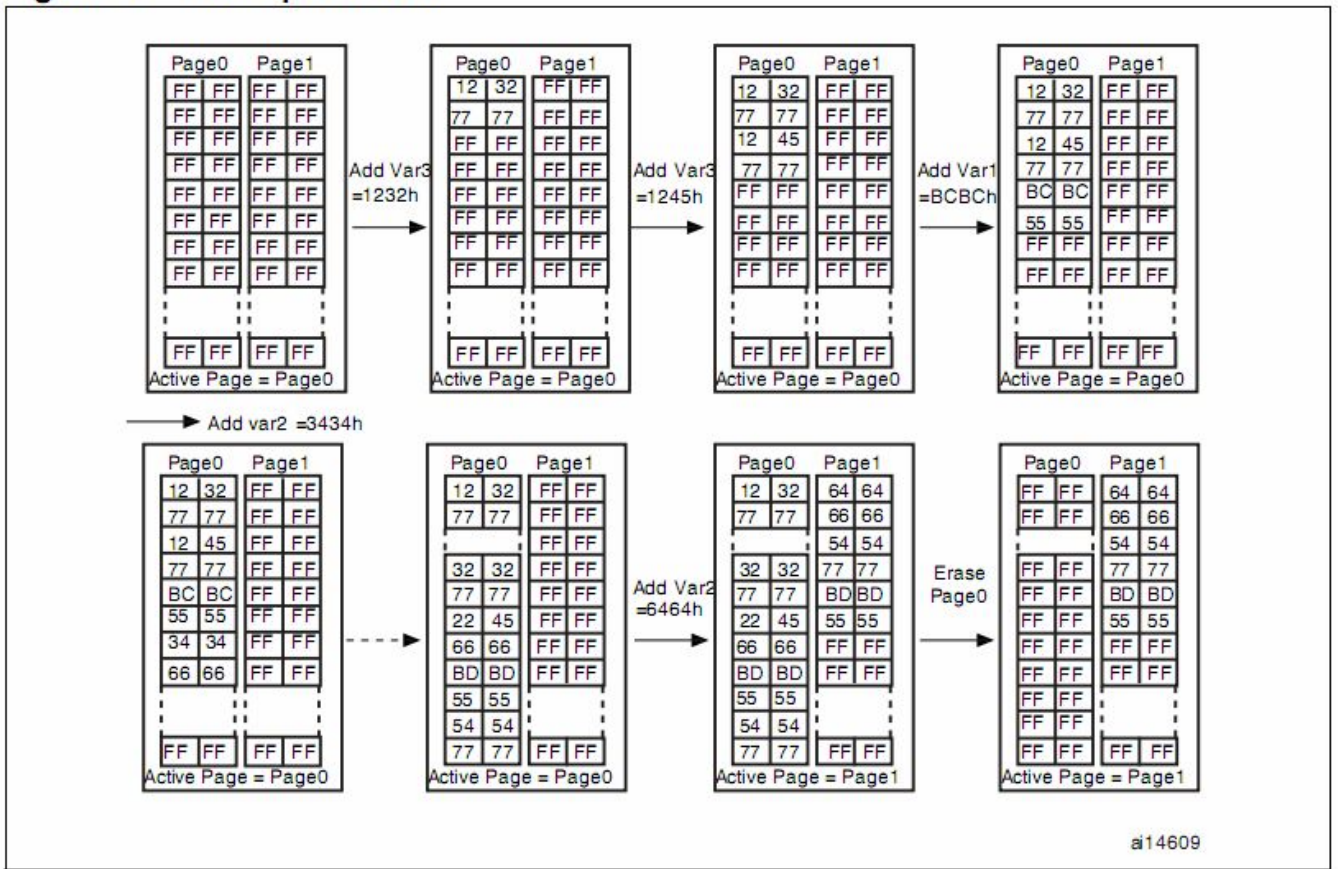


Figure 3 显示了数据更新的过程:

1. 写数据

假设保存的数据虚拟地址是 0X7777, 那么程序写数据是从当前有效页首地址开始查询虚拟地址位置为 0XFFFF 的空间, 如果是 0XFFFF 那么该位置可以保存数据; 如果不是, 那么继续找下 1 个位置, 如果本页无 0XFFFF 的空间那么表示本页已满, 那么将本页【有效数据】复制到另外 1 页继续保存数据。

当两次保存同一虚拟地址的数据时如右图所示: 从上到下, 第 2 个虚拟地址是 0X7777 对应的数据 1245 才是有效的。清楚了这点, 那么读数据要怎么处理基本就明白了。

2. 读数据

读数据时是从有效页的末尾地址开始检测是否是有效数据, 如果是那么立即返回, 程序是通过虚拟地址判断有效数据的, 第 1 个匹配的虚拟地址的数据才是有效的

3. 页满时处理数据

说到这里, 看到不少使用例子程序不成功的问题, 那么就请注意下面了, 他们的错误估计是下面的原因造成的。当 1 页写满时其实里面有很多无效数据, 你只需要将【有效数据】复制到另外 1 页就成。如何复制有效数据呢? 我想很多人估计忽略了【#define NumOfVar ((uint8_t)0x03)/* Variables' number */】, NumOfVar 就是你程序中实际要保存的数据量, 这个必须与实际保持一致, 不能多也不能少, 这个如果不一致, 那么在换页时将出错, 没换页之前倒是没问题的, 原因在于: 程序在换页时将根据 NumOfVar 的值复制有效数据的个数, 如果比实际少, 那么换页时将丢失数据, 如果比实际多那么将出现旧数据覆盖最新数据



错误的例子:

```
/* Variables' number */
#define NumOfVar                ((uint8_t)0x05)

uint16_t VirtAddVarTab[NumOfVar] = {0, 1, 2};
//NumOfVar 定义的比用的多实际是{0, 1, 2, 0, 0},虚拟地址 0 的数据换页后将出现旧数据覆盖最新数据
```

```
int main(void)
{
uint16_t temp;
for (VarValue = 0; VarValue < 100; VarValue++)
{
EE_WriteVariable(VirtAddVarTab[0], VarValue+10);
}
for (VarValue = 0; VarValue < 500; VarValue++)
{
EE_WriteVariable(VirtAddVarTab[1], VarValue);
temp=0;
EE_ReadVariable(0, &temp);//不换页读出数据是对的, 换页后读出数据错误
}
}
```

//=====

```
/* Variables' number */
#define NumOfVar                ((uint8_t)0x03)

uint16_t VirtAddVarTab[NumOfVar] = {0, 1, 2};
//NumOfVar 定义为 3, 下面用到虚拟地址超过 VirtAddVarTab 表里面的值
```

```
int main(void)
{
uint16_t temp;

for (VarValue = 0; VarValue < 100; VarValue++)
{
EE_WriteVariable(VirtAddVarTab[0], VarValue);
}
for (VarValue = 0; VarValue < 50; VarValue++)
{
EE_WriteVariable(3, VarValue+2);
}

for (VarValue = 0; VarValue < 200; VarValue++)
```

```
{  
    EE_WriteVariable(2, VarValue);  
    temp=0;  
    EE_ReadVariable(3, &temp); //不翻页读出数据是对的, 翻页后读出数据错误  
}  
}
```

STM32 FLASH 模拟 EEPROM 使用注意:

不少人问该程序的 FLASH 保存数据多少和使用寿命

保存数据多少跟 FLASH 页大小有关, 如果页大小是 1K 那么只能保存 $1024/4-1=256-1$ 个 16 位数据, 如果你保存 8 位数, 你可以 2 个 8 位数据组合后保存或者直接保存, 如果保存 32 位数据那就拆成 2 个 16 位保存, 当然关于寿命

现在 STM32 的 FLASH 寿命是 10000 次,

如果你保存 255 个数据那么每次修改 1 个数据 FLASH 就要擦写 1 次, 如果你保存 1 个数据, 那么你修改 255 次该页才擦 1 次, 继续用另外 1 页, 建议保存数据个数不要超过 50%, 当然如果你的数据基本都不修改你保存 255 个也是没有任何问题(你的数据都不修改根本不用关心寿命问题了:)。

STM32 FLASH 模拟 EEPROM 优化

官方例程中读写数据每次要查询读写位置, 写数据是从页首地址开始查询, 读地址是从页末地址查询。

假如只有 1 个数据, 读数据时效率是很低的, 要查到最后才能找到有效数据,

如果页快满了写数据效率也很低, 读效率反而好一些了。

实际程序中记录下一个可以写数据的位置将提高数据的读写效率, 这样的话: 写数据就是立即写不用查询, 读数据不从页末地址查询, 而是从最后 1 个写入数据处查询, 这样特别在页数据少时效率提高不少。优化过的例子代码只需要增加很少部分就能实现。

增加关键代码

```
uint32_t CurWrAddress;
```

//初始化写地址, 减少每次读写时查询时间

```
uint16_t InitCurrWrAddress(void)
```

详细请看修改后的例子, 读写函数也做了相应更改

STM32 FLASH 模拟 EEPROM 进一步优化

上面优化过的例子在写数据无须查询直接写入就可, 但是读数据在页数据少是效率提升明显, 在页数据多时效率不明显, 特别是页数据快满时就跟原来一样的。

说明: 不管是官方还是优化过的例子在页交换时这个模拟 EEPROM 程序都将耗费不少时间的如果你对时间要求不高完全不用考虑下面的了。

下面就进一步提升它的效率, 方法如下:

为每 1 个保存的变量定义 1 个映射地址, 就是在写数据时将写数据的地址偏移保存起来。比如第 1 次的数据映射地址是 0, 第 2 次的数据映射地址是 1, 那么读数据时就可以立即计算出地址。此方法对于 1K 页大小的每个数据将增加 1 个 8 位映射地址, 对于 2K 页大小的每个数据将增加 1 个 16 位映射地址。

Page0		Page1	
12	32	FF	FF
77	77	FF	FF
12	45	FF	FF
77	77	FF	FF
FF	FF	FF	FF
FF	FF	FF	FF
FF	FF	FF	FF

这里只提供方法, 当然方法不是唯一的, 有兴趣的自己去玩。