三星 Full Flash 型 MCU 的应用之

—— 在线存储用户数据

在很多应用场合都需要断电时仍能保存部分用户数据。因此有些客户在电路设计时外加了 E2PROM。

三星最近推出了几款 Full-flash 型 ROM 的单片机。所谓 Full Flash，即无需外部施加任何编程电压，仅用单片机的供电电压就可以完成 Flash 的擦写操作。这是因为芯片内部集成了一个 pumping 模块，可以实现升压。这对于单片机的使用者来说，意味着所有的片上 ROM 都可以在程序的运行过程中对其内容进行修改。显然，一旦使用了这样的芯片就不再因为需要断电保持数据而外扩其他程序存储器了。这很大程度的降低了产品成本。

本文介绍了利用 Full flash 芯片的部分空间存储用户数据的相关应用。

1 基本原理

1.1 Full Flash 的 tool mode 和 user mode

1.1.1 Tool mode

Full Flash 单片机的 tool mode 和以往（称之为 half flash）的芯片基本相同，只是原来需要 6 根编程线（分别为 RESET，SDA，SCLK，VDD，VSS，VPP）与编程器相连，现在 VPP 可以不需要编程器提供，所以 5 根线就可以实现编程。无论是那种类型的单片机，注意编程时 VDD 电压的选择必须在芯片的工作电压范围内，一般设为典型值。

1.1.2 User mode

Full Flash 单片机具有 user mode，即在 MCU 的工作电压下，可以用存储在 Flash 中的指令改变程序存储区的内容。具体的操作可分为读（Read），擦除（ERASE），写（Program），锁定（Hard Lock Protection）。下面分别做一介绍。

1.1.2.1 Read

Flash 的读操作相信大多数用户都非常熟悉，一般是用来完成查表操作的。

示例代码：实现 R1 = @ 1784

```
LD      R3,     #17H
LD      R4,     #84H
LDC     R1,     @ RR3
```

1.1.2.2 Erase

编程步骤：

1. 使能 user mode (FMUSR = 0xA5)
2. 定位需擦除扇区（设置 FMSECH/FMSECL）
3. 定义此次操作为擦除并开始擦除（FMCON = 0xA1）
4. 查询擦除操作是否成功（FMCON.0），不成功则返回 3 再次擦除
5. 关闭 user mode

示例代码：实现擦除起始地址为 1000H 的一个 sector

```
        …
LD  FMUSR,      #0A5H       ; User program mode enable
LD  FMSECH,     #10H
LD  FMSECL,     #00H        ; Set sector address (1000H – 107FH)
REDO:
LD  FMCON,      #A1H    ; Start sector erase
NOP                     ; Dummy instruction, this instruction must be needed
NOP                     ; Dummy instruction, this instruction must be needed
LD  FMUSR,      #0          ; User program mode disable
JR  NZ,         REDO        ; Jump to reErase if fail
TM  FMCON,      #00000001B ; Check "sector erase status bit"
        …
```

注意事项：

1. 注意与 Flash 操作相关的寄存器所在 RAM 空间，对其操作前必须有相应的 bank 或者 page 选择。
2. 因为 flash 的结构特征，任何字节在写之前必须有擦除操作。
3. flash erase 操作的最小单位为一个 sector，128 bytes，具体的 sector 起始地址请参照相应的数据手册。

1.1.2.3 Program

一个字节的编程步骤：

1. 使能 user mode (FMUSR = 0xA5)
2. 定位需编程地址所在的扇区（设置 FMSECH/FMSECL）
3. 定义此次操作为写（FMCON = 0x 50）
4. 用工作寄存器存储需要传输的数据
5. 用工作寄存器组定位 16 位地址
6. 用 LDC 指令的间接寻址方式将工作寄存器的数据送到工作寄存器组指向的地址空间
7. 关闭 user mode（FMUSR = 0x0）

示例代码：实现@1784 = 78H

```
…
LD  FMSECH,     #17H
LD  FMSECL,     #80H    ; Set sector address (1780H–17FFH)
LD  R2,         #17H    ; Set ROM address in the same sector 1780H–17FFH
LD  R3,         #84H
LD  R4,         #78H    ; Temporary data
LD  FMUSR,      #0A5H   ; User program mode enable
LD  FMCON,      #50H    ; Start program
LDC @RR2,       R4      ; Write the data to a address of same sector (1784H)
NOP                     ; Dummy instruction must be needed
LD  FMUSR,      #0      ; User program mode disable
…
```

注意事项：

1. 注意与 Flash 操作相关的寄存器所在 RAM 空间，对其操作前必须有相应的 bank 或者 page 选择
2. 写之前应该先定位目标地址所在的 sector（通过写寄存器 FMSECH 和 FMSECL），然后具体定位目标地址
3. 每次写一个 byte，当写的数据总数超过一个 sector 的大小 128byte 时，注意更改 FMSECH 和 FMSECL 寄存器的值

1.1.2.4 Hard Lock Protection

编程步骤：

1. 使能 user mode (FMUSR = 0xA5)
2. 定义此次操作为 Hard Lock Protection （FMCON = 0x61）
4. 关闭 user mode（FMUSR = 0x0）

示例代码：

```
…
LD   FMUSR,      #0A5H        ; User program mode enable
LD   FMCON,      #01100001B ; Hard Lock mode set & start
NOP                           ; Dummy instruction, this instruction must be needed
LD   FMUSR,      #0           ; User program mode disable
…
```

注意事项：

1. 使能 Hard Lock Protection 将使 flash 的擦除和写操作无法进行
2. 一旦使能，只能通过 tool mode 擦除 flash 内容解除保护

1.2 存储算法

从前面的描述可以看到，修改同一存储地址的数据，必须先擦除这个地址所在的整个 sector 的内容。这在一定程度上增加了编程的难度，因为不可能为一个字节分配一个 byte 的存储空间。又，考虑到 flash 的可擦写次数不是无限的，一般保证 10000 次。所以需要设计一种算法既提高 flash 空间的使用率又最大限度的增加 flash 的使用寿命。

在介绍算法前，先做如下假设：

1. 存储用户数据的扇区的起止地址定为 FF80H – FFFFH
2. 为方便陈述，假设用户数据大小为 8byte
3. 假设用户数据中没有 FFH（这一点在实际应用中是比较容易满足的，如果实在必须有 FFH 的内容，可以将此假设改为用户数据中没有连续 2 个以上的 FFH，如有需要，条件可以更紧）。这是因为 flash 擦除后的状态就是 1，这个假设可以帮助找到已编程部分和未编程部分的分界点

整个算法的基本思想可以用下图的存储模型来概括。 对于 8 bytes 用户数据的场合，只有第 1+16(n-1)次的修改在 FF80H –FF87H 进行，2+16(n-1)次的修改在 FF88H –FF8FH 进行以此类推。n（1→10000）表示 flash 的擦除次数。 16 次写操作以后擦除整个 sector 的内容后继续写第 17 次数据。

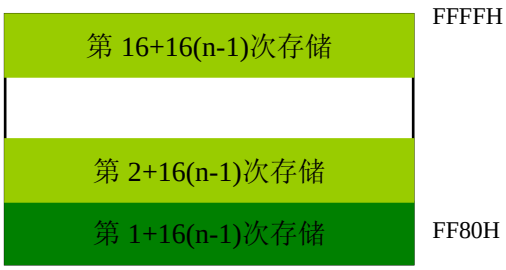

图 1 存储模型

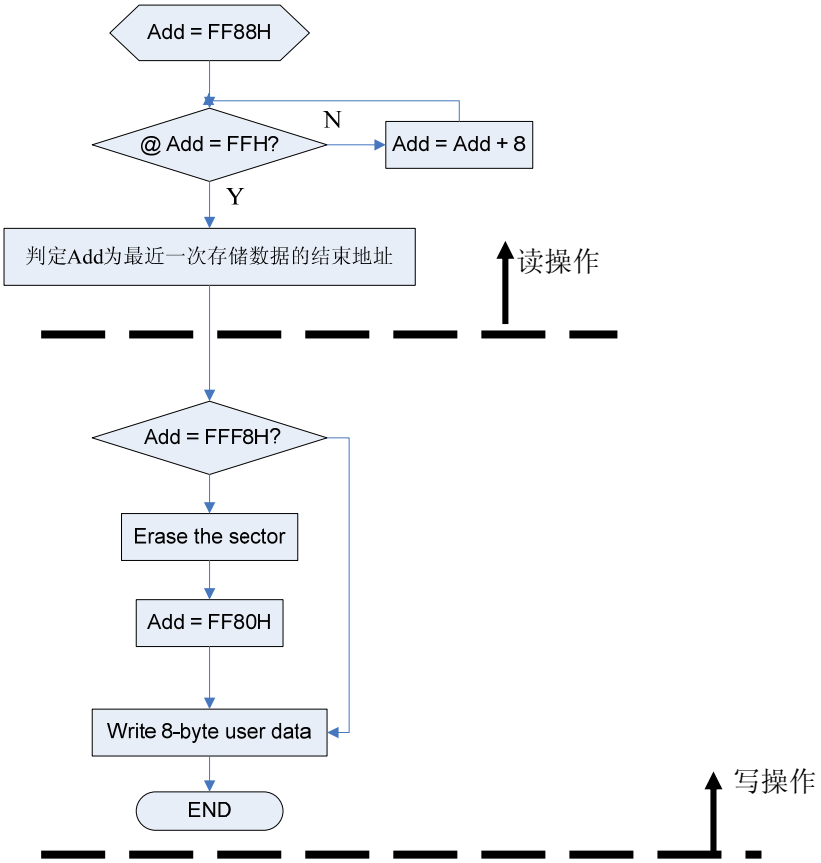

图 2 存储算法流程图

显然对最后一次存储数据结束地址的判断并不需要每次读写前都进行，只要在程序上电前判断一次就可以了。因为只要没有断电，就可以用通用寄存器存储每次修改数据的位置信息。

2 应用实例

未完待续… ^_^