SRAM 中的数据丢失

问题：

该问题由某客户提出，发生在 STM32F103VDT6 器件上。据其工程师讲述：在该公司的某型号产品的设 计中用到了 STM32F103VDT6 器件，而其软件的设计采用了 IAP+APP 的架构。IAP 是一段 BOOT 程序， 负责对硬件进行初始化以及在接到相关指令的情况下更新 APP 程序，而 APP 程序则负责对常规业务 处理。在 STM32 启动后，IAP 首先运行。在初始化硬件之后，检查是否有更新 APP 的指令，如果 有，则更新 APP，如果没有，则跳转到 APP。APP 对常规业务进行处理时，如果接到更新指令则复位 STM32，以使程序的流程回到 IAP。在 IAP 和 APP 进行切换的过程中，需要相互传递一些数据。其软 件的实现的方法是，发送方将数据写到 SRAM 的某个区域，接收方则到同一区域将数据读回。然而实 测中发现，相关数据并未被正确的传递，而是丢失了。在整个切换过程中，STM32 并未掉电。

调研：

检查其软件设计，了解到其所使用的开发工具为 Keil MDK，而 IAP 和 APP 分别由两个独立的软件工 程所生成。进一步检查相关的代码，发现在 IAP 和 APP 中，用于数据传递的缓冲区定义相同，如表（一），而其 IAP 和 APP 的内存分配也相同，如表（二）

```
//IapToApp.c
char IapToAppBuf[100];  //
```

表（一）

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
LR_IROM1 0x08000000 0x00080000 {      ; load region size_region
  ER_IROM1 0x08000000 0x00080000 {    ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  IRAM1 0x20000000 0x00000064 {
    IapToApp.o(+RW +ZI)
  }
  IRAM2 +0 0x0000FF9C {                ; RW data
    .ANY (+RW +ZI)
    startup*.o (HEAP, +Last)           ; HEAP base define
  }
  STACKS 0x20010000 UNINIT {
    startup*.o (STACK)                 ; STACK base define
  }
}
```

表（二）

对其软件加以分析：

1. 编译两个工程，通过检查 map 文件确认 IAP 中定义的 IapToAppBuf 与 APP 中定义的 IapToAppBuf 所分配的地址相同，都是 0x20000000 开始的内存区域。
2. 修改 IAP 的代码，使其在缓冲区内填满 0x55。修改 APP 的代码，使其在进入 main()函数后打印整个缓冲区。编译两个工程后，全部加载到 STM32。运行结果表明，APP 从缓冲区内读出的数据匀为零，而不是 0x55。
3. 修改 APP 的代码，使其在缓冲区内填满 0x55，然后复位 STM32。修改 IAP 的代码，使其进入 main()函数后，打印缓冲区内的数据。结果表明，IAP 从缓冲区内读出的数据匀为零，而不是 0x55。 修改 APP 的内存分配，如（三），然后重复“分析”中的测试：

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
LR_IROM1 0x08000000 0x00080000 {     ; load region size_region
  ER_IROM1 0x08000000 0x00080000 {   ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  IRAM1 0x20000000 UNINIT 0x00000064 {
    IapToApp.o(+RW +ZI)
  }
  IRAM2 +0 0x0000FF9C {              ; RW data
    .ANY (+RW +ZI)
    startup*.o (HEAP, +Last)         ; HEAP base define
  }
  STACKS 0x20010000 UNINIT {
    startup*.o (STACK)               ; STACK base define
  }
}
```

表（三）

1. 重复测试（2），结果表明 APP 读出的数据为 0x55，是 IAP 放在缓冲区内的数据。
2. 重复测试（3），结果表明 IAP 读出的数据匀为零，不是 APP 放在缓冲区内的 0x55。 修改 IAP 的内存分配，如表（三），然后重复“分析”中的测试：

1. 重复测试（2），结果表明 APP 读出的数据为 0x55，是 IAP 放在缓冲区内的数据。
2. 重复测试（3），结果表明 IAP 读出的数据为 0x55，是 APP 放在缓冲区内的数据。

结论：

为缓冲区分配的内存未加 UNINIT 属性控制，导致 C 语言初始化程序对该区域进行了清零处理，造成 存放在其中的数据丢失。

处理：

修改内存分配，将该内存区域加上 UNINIT 属性。

建议：

C 语言初始化程序是由 C 语言开发工具中的链接器（Linker）加到应用程序当中的一段程序，它早于 main()函数得以执行，完成对 C 语言运行环境的初始化工作。这段程序会对所有用到的且没有 UNINIT 属性的内存做清零处理。所以，要使得某段内存区域的数据保留给 main()函数而不被清除， 需要对这段内存加上 UNINIT 属性。当然，也可以把一段内存区域隐藏起来，不交给 C 语言初始化程 序管理，如表（四）所示。其中 0x20000000 – 0x20000063 这段内存没有被分配，于是 C 语言初始 化程序不知道它的存在，也就不会对它做任何的处理。在应用程序中，可以通过指针来访问这一区 域，从而实现通过这段内存向 main()函数传递数据功能。

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
LR_IROM1 0x08000000 0x00080000 {     ; load region size_region
  ER_IROM1 0x08000000 0x00080000 {   ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  IRAM1 0x20000064 0x00010000 {       ; RW data
    .ANY (+RW +ZI)
    startup*.o (HEAP, +Last)          ; HEAP base define
  }
  STACKS 0x20010000 UNINIT {
    startup*.o (STACK)                ; STACK base define
  }
}
```

表（四）

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。