*Application Note*

# *UCD3138 器件上的栈溢出检测*

*Xuemei Lu*

**摘要**

由于栈的大小随运行的代码而异，因此很难准确判断栈静态使用的大小。RAM 中还存储了其他内容，例如全局变量。如果发生栈溢出，它会意外修改其他栈，从而导致不可预测的问题。因此，必须为栈保留足够的空间。本应用手册介绍了两个用以检查栈是否发生溢出的过程。

## 内容



## 插图清单



## 商标

# 1 引言

## 1.1 检查每个栈的大小

演示代码中通常使用四种栈：用户栈、IRQ 栈、FIQ 栈和监控器栈。用户栈适用于后台例程，IRQ 栈适用于标准中断例程，FIQ 栈适用于快速中断例程，监控器栈适用于软件中断 (SWI) 例程。有些可能在例外情况下具有未定义的栈和中止栈，但在正常情况下很少使用。栈在 load.asm 文件的顶部声明。考虑下面的示例，了解如何分配每个栈。

```
SUP_STACK_TOP  .equ    0x6bffc ;Supervisor mode (SWI stack) starts at top of memory
FIQ_STACK_TOP    .equ    0x6be00 ;allocate 256 bytes to supervisor stack, then do FIQ stack
IRQ_STACK_TOP    .equ    0x6bd00 ;allocate 256 bytes to fiq stack, then start irq stack
USER_STACK_TOP    .equ    0x6bb00 ;Allocate 512 bytes to irq stack, regular stack gets rest, down
to variables
        .global _c_int00
        .global $c_int00
```

上述定义表明用户栈的顶部位于地址 0x6bb00，再往下是变量，IRQ 栈的分配范围从地址 0x6bb00（底部）到 0x6bd00（顶部），FIQ 栈的分配范围从地址 0x6bd00（底部）到 0x6be00（顶部），监控器栈的分配范围从地址 0x6be00（底部）到 0x6bffc（顶部）。

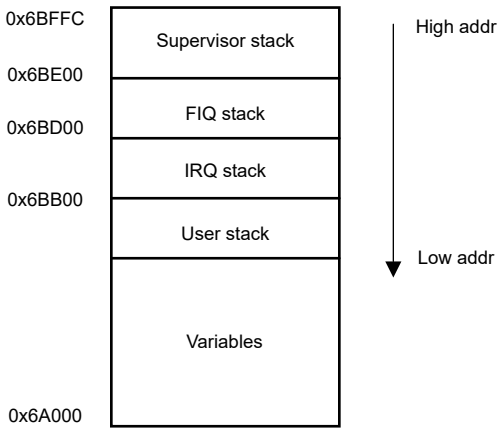


**图 1-1. 栈分配**

至于用户栈的底部，请查看 .map 文件，其中显示了 RAM 存储器分配。在构建固件工程时，.map 文件由 CCS 生成，应位于 .x0 文件所在的目录中。下面是从 .map 文件复制的示例，其中的文本显示变量从地址 0x6a000 开始，到地址 0x6a80d 结束。因此，用户栈可向下到达 0x6a80e。

```
        name            origin    length      used     unused   attr    fill
----------------------  --------  ---------  --------  --------  ----  --------
  FLASHVECS             00000000  00000020  00000020  00000000  R  X
  PFLASH                00000020  00007f34  00003d1e  00004216  R  X
  DEVICEID              00007f54  00000020  0000001f  00000001  R  X
  FIXTFA                00007f74  00000004  00000000  00000004  R  X
  FIXCONST              00007f78  00000080  00000000  00000080  R  X
  FLASHSUM              00007ff8  00000008  00000000  00000008  R  X
  ROMVECS               00020000  00000020  00000000  00000020  RWIX
  ROM                   00020020  00001d5e  00000000  00001d5e  RWIX
  SINE                  00021d7e  00000282  00000000  00000282  RWIX
  DFLASH                00069800  00000800  00000398  00000468  R  X
  RAM                   0006a000  00001dd0  0000080d  000015c3  RW
  RAM_PGM_AREA          0006bdd0  00000080  00000000  00000080  RW
  STACKS                0006be50  000001b0  00000000  000001b0  RW
  LOOP_MUX              00120000  00000070  0000006c  00000004  RWIX
```

在一些演示代码中，.cmd 文件中有栈分配，如下所示。栈分配不会生效，栈分配实际上是在 load.asm 中完成的。

```
STACKS    (RW) : org = 0x0006BE50, len = 0x000001B0
.stack         : {                              /* total = 400 = 0x190              */
                        _StackUSER_  = .            + 184;   /* USER                     */
                        _StackFIQ_   = _StackUSER_  + 112;   /* FIQ                      */
                        _StackIRQ_   = _StackFIQ_   + 84;    /* IRQ                      */
                        _StackABORT_ = _StackIRQ_   + 4;     /* ABORT                    */
                        _StackUND_   = _StackABORT_ + 4;     /* UND                      */
                        _StackSUPER_ = _StackUND_   + 12;    /* SUPER                    */
                    } > STACKS                          /* Software System stack     */.
```

## 2 检查是否发生溢出

栈是 RAM 的一部分，在 load.asm 中被完全初始化为零。若要检查每个栈的使用情况，一种选择是从栈位置读取，然后查看栈底部是否有全零空间。有必要在代码运行时检查栈，这可以通过适配器和嵌入在 UCD3xxx 器件 GUI 中的存储器 Peek/Poke 工具来完成。

连接适配器，启动 UCD3xxx 器件 GUI，转到 Debug > Memory Peek/Poke。

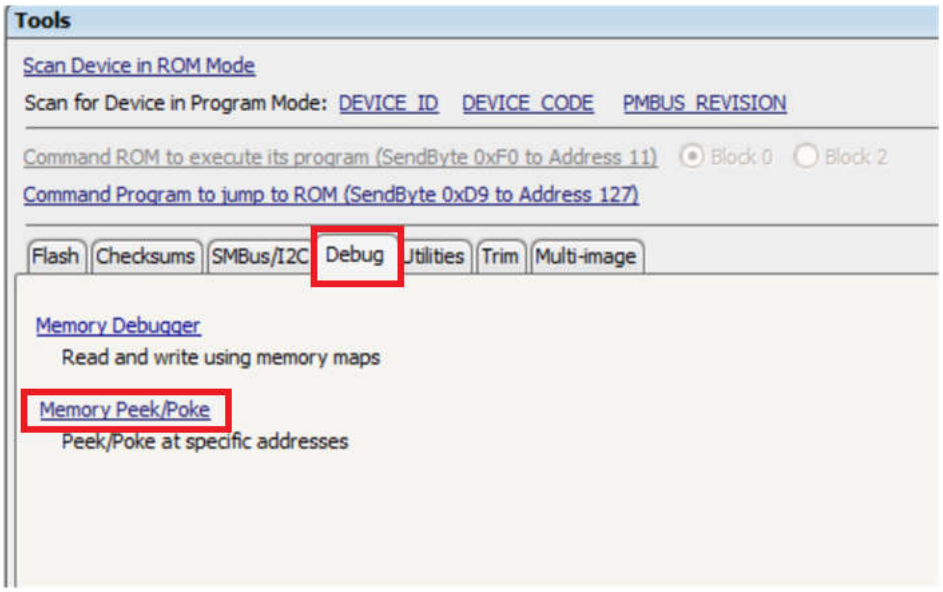


**图 2-1. 存储器 Peek/Poke**

以用户栈为例。读取存储器的地址 0x6a80e 至 0x6bb00，如下所示。它显示大约使用了 60 个字节，4754 个字节仍然可用。当然，这很难捕捉到最坏的情况，因为代码运行期间栈会不时发生变化。但是，通过检查栈的边距大小，我们应该可以获得一些线索，以了解是否可能发生溢出。

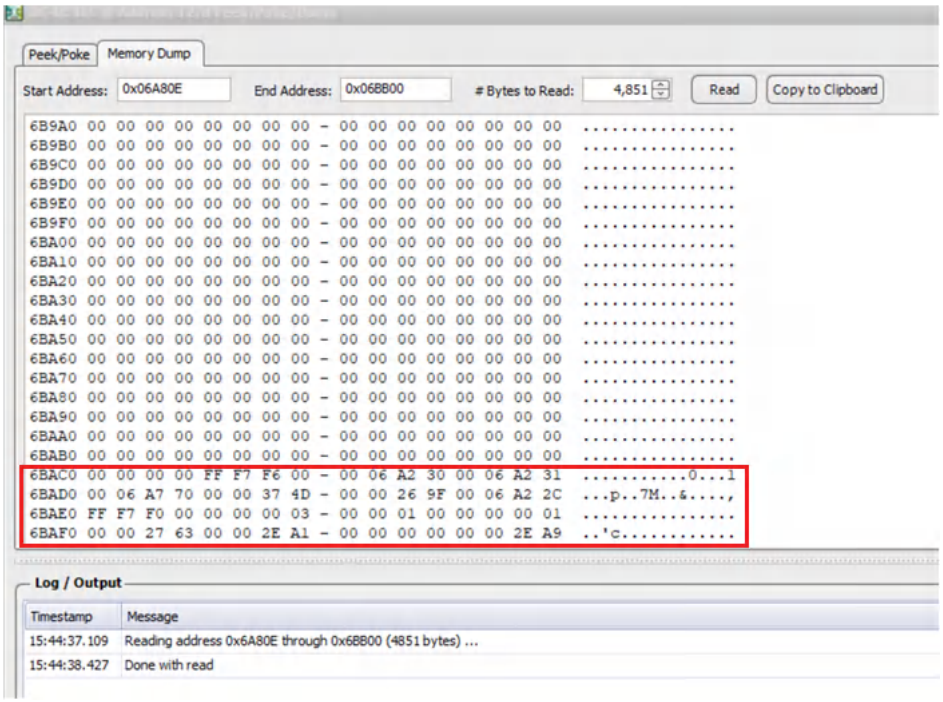


**图 2-2. 通过存储器 Peek/Poke 检测栈使用情况**

另一种选择是在固件中添加测试代码，以持续检查栈的底部（或某些边距的底部附近）是否为非零。检测到非零后，它会显示发生溢出，切换 IO 进行报告。此选项的好处是，它会在代码运行期间持续进行检查。

以监控器栈为例，检测代码可能与以下代码类似，其中 stack_mon.ptr 最初是指向监控器栈底部的指针。它必须从底部开始。

```
                if (((Uint32)stack_mon.ptr == (Uint32)SUP_STACK_TOP) || (Uint32)(*(stack_mon.ptr) !
= 0))
                {
                    // reached top of stack so the stack is all zeros (so stack is empty), or else
encountered the stack (as encountered a non-zero word)
                    stack_sup_headroom = (int32)stack_mon.ptr - (int32)SUP_STACK_BOT;
                    if (stack_sup_headroom < STACK_MON_HEADROOM_ALERT)
                    {
                        LoopMuxRegs.DTCIOCTRL.bit.DTC_B_GPIO_VAL = 1;
                    }
                }
                else
                {
                    // move onto the next address in the stack
                    stack_mon.ptr++;
                }
```

## 3 总结

发生栈溢出时，可能会出现意外问题。本应用手册介绍了检测栈溢出的两种选项。选项 1 使用存储器 peek/poke 完成。它易于实现，但无法检测最坏的情况，因为栈使用情况因运行的代码而异。选项 2 是在溢出发生后持续检测并发出警报。

## 4 参考文献

- 德州仪器 (TI)，*Fusion Digital Power Studio*。

# 重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265