Xradio SDK 系统异常调试

使用指南

版本号：1.0

发布时间：2022-01-07

版本历史

版本	日期	责任人	版本描述
1.0	2022-01-07	AWA 0498	创建文档。

目录

表格目录

1 前言

1.1 文档简介

本文档介绍 Xradio SDK 在遇到系统问题时，如何通过系统的异常打印分析定位出异常的原因和出错点，加快问题的解决。

1.2 目标读者

使用 Xradio SDK 开发的用户。

1.3 适用范围

本文档适用于 XR872/XR808/XR806 SDK，支持 Xradio 系列芯片。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示(寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

2 概述

为了方便用户使用 Xradio SDK，Xradio SDK 中加入了系统出现异常时的现场打印，打印信息中包含了引起系统异常的原因、异常产生时的 CPU 寄存器值、线程栈、系统栈、程序计数器（PC）前后的代码段内容以及各线程状态，可通过以上信息对系统异常的原因及现场进行分析。然而，系统出现异常时，有可能不是错误产生的第一现场，所以打印的异常信息不一定能定位出异常根因。

3 系统异常信息

系统异常信息是指当 Xradio SDK 系统异常出错时(比如访问非法地址)打印出来的信息，从中能分析出一些引起异常的原因，异常位置等信息。

图 3-1 系统异常信息

```
exception:6 happen!!
appos pstack:0x4b930 msp:0x7ffe0 psp:0x4b950
usage fault happen, UFSR:0x200
CPU registers:
R00:[0x4b950]: 0x00000000
R01:[0x4b954]: 0x00000011
R02:[0x4b958]: 0x00000022
R03:[0x4b95c]: 0x00000210
R04:[0x4b930]: 0x0004b590
R05:[0x4b934]: 0xa5a5a5a5
R06:[0x4b938]: 0xa5a5a5a5
R07:[0x4b93c]: 0xa5a5a5a5
R08:[0x4b940]: 0xa5a5a5a5
R09:[0x4b944]: 0xa5a5a5a5
R10:[0x4b948]: 0xa5a5a5a5
R11:[0x4b94c]: 0xa5a5a5a5
R12:[0x4b960]: 0x00043b0e
R14(LR):[0x4b964]: 0x00010645
R15(PC):[0x4b968]: 0x00010506
xPSR:[0x4b96c]: 0x21000000
SHCSR:0x00070008 step:0
FPSCR:0x00000000

stack info:
[0x4b970]: 0x00000000 0x00000011 0x00000022 0x0004b590
[0x4b980]: 0xa5a5a5a5 0x00010645 0xa5a5a5a5 0x00036531
```

4 异常信息包含的内容

4.1 异常类型

第一行指出的信息，它指明了发生的是哪类异常，类型范围从 2~15，定义见异常类型章节。

示例中的：“exception:6 happen!!”，其中“6”就是发生异常的类型，是“用法错误”导致了此次异常。

4.2 异常栈信息

第二行指出的信息，它指明了发生异常时正在使用的栈。

示例中的：“appos pstack:0x4b930 msp:0x7ffe0 psp:0x4b950”，说明发生异常时正在使用的栈位置是：“pstack:0x4b930”，此时系统主栈位置是：“msp:0x7ffe0”，发生异常时刻的栈位置是：“psp:0x4b950”。pstack 是在当前栈的基础上又保存了一些信息供异常打印，所以和发生异常时刻的栈位置稍有区别，通过 pstack 接近 psp 说明发生异常时正在使用 psp，是线程中发生是异常。系统初始化完后，msp 只有中断在使用，因此通过 pstack 可判断出发送异常时是在中断上下文中还是线程上下文中。

4.3 异常 CPU 信息

```
CPU registers:
R00:[0x4b950]: 0x00000000
R01:[0x4b954]: 0x00000011
…
```

示例中的：“CPU registers:”部分。即发生异常时 CPU 各个寄存器值如下：“R0:0x00000000, R1:0x00000011, …”，通过这些值可以还原现场，辅助分析异常原因。

4.4 栈及指令信息

4.4.1 栈内容

```
stack info:
[0x4b970]: 0x00000000 0x00000011 0x00000022 0x0004b590
[0x4b980]: 0xa5a5a5a5 0x00010645 0xa5a5a5a5 0x00036531
…
```

示例中的：“stack info:”部分。即发生异常时栈的内容，栈是出异常时值向后打印一些 Word，通过这些值可以还原现场，辅助分析异常原因。

4.4.2 返回值附近的机器码

```
[LR]:0x10645
```

```
[0x105c0]: 0xf0274850 0xf002fac7 0x4601fca5 0xf027484e
[0x105d0]: 0xf002fac1 0x4601fcad 0xf027484c 0xf004fabb
...
```

示例中的：“[LR]:”部分。即发生异常时返回值附件的内容，通常返回值是此函数执行完后要返回执行的下一条指令，通过返回值的前一个函数调用，可以知道发生异常的函数，通过对比此段内存值，可以看出此段内存是否被修改过。

4.4.3 异常指令附近的机器码

```
[PC]:0x10506
[0x10480]: 0xf0272204 0xb118fdf5 0x4621480c 0xfb62f027
[0x10490]: 0x490b4620 0xf000220d 0x2864ff81 0xd0084604
...
```

示例中的：“[PC]:”部分。即发生异常时 PC 前后的内容，通过 PC 值可以直接定位到发生的异常的那条指令，但有时因为发生异常时已经不是第一现场，因此 PC 值不一定能和源码对应上，当 PC 值无效时，可以通过 LR 值进一步定位。

4.5 异常各任务信息

```
tasks state:
Name            State   Pri     HWM     Idx     StkCur  StkBot
main            R       3       148     1       0x4b8a8 0x4b598
IDLE            R       0       98      2       0x4bba0 0x4ba08
...
```

示例中的：“tasks state:”部分。即发生异常时系统各线程任务的信息，包括当前任务名称、状态、优先级、历史最小栈余量、任务编号、当前栈位置、栈底位置。

任务状态：“R”代表 ready，“B”代表 block。

栈信息：Xradio 的栈为向下生长模式，StkCur 要大于 StkBot，栈顶位置为 StkBot 加上初始化时配置的栈大小。通过对比 pstack 落在哪个任务的栈中，可以定位是哪个任务线程发送的异常。

5 使用异常信息定位异常点实例

5.1 获得异常信息

本节故意修改一段程序产生异常：main_task 结尾增加下面的除零错误代码：

```
static int div_zero(int a, int b, int c)
{
	int res;

	res = b/a;
	res = res/c;
	printf("resule:%d %d %d %d\n", res, a, b, c);

	return res;
}

static void main_task(void *arg)
{
	...

	writel(readl(0xe000ed14)|(1<<4), 0xe000ed14);
	div = div_zero(0x00, 0x11, 0x22);
	printf("%d\n", div);
	div = div_zero(0x02, 0x1, 3);
	printf("%d\n", div);
	...
}
```

重新编译代码和烧录，启动后会打印出如下异常信息：

```
exception:6 happen!!
appos pstack:0x4b930 msp:0x7ffe0 psp:0x4b950
usage fault happen, UFSR:0x200
CPU registers:
R00:[0x4b950]: 0x00000000
R01:[0x4b954]: 0x00000011
R02:[0x4b958]: 0x00000022
R03:[0x4b95c]: 0x00000210
R04:[0x4b930]: 0x0004b590
R05:[0x4b934]: 0xa5a5a5a5
R06:[0x4b938]: 0xa5a5a5a5
```

R07:[0x4b93c]: 0xa5a5a5a5
R08:[0x4b940]: 0xa5a5a5a5
R09:[0x4b944]: 0xa5a5a5a5
R10:[0x4b948]: 0xa5a5a5a5
R11:[0x4b94c]: 0xa5a5a5a5
R12:[0x4b960]: 0x00043b0e
R14(LR):[0x4b964]: 0x00010645
R15(PC):[0x4b968]: 0x00010506
xPSR:[0x4b96c]: 0x21000000
SHCSR:0x00070008 step:0
FPSCR:0x00000000

stack info:
[0x4b970]: 0x00000000 0x00000011 0x00000022 0x0004b590
[0x4b980]: 0xa5a5a5a5 0x00010645 0xa5a5a5a5 0x00036531
[0x4b990]: 0xa5a5a5a5 0xa5a5a5a5 0x00000068 0xfffffffc
[0x4b9a0]: 0x0004b8a8 0x7ce1ec9b 0x0004b484 0x0004b484
[0x4b9b0]: 0x0004b9a0 0x0004b47c 0x00000004 0x0004b4ec
[0x4b9c0]: 0x0004b4ec 0x0004b9a0 0x00000000 0x00000003
[0x4b9d0]: 0x0004b598 0x6e69616d 0x05d67f00 0xe680c650
[0x4b9e0]: 0x008cebfc 0x00000001 0xdbe4ab13 0x00000003
[0x4b9f0]: 0x00000000 0x00000000 0x00000000 0x7be4c100
[0x4ba00]: 0x00000208 0xfffffffc 0xa5a5a5a5 0xa5a5a5a5
[0x4ba10]: 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5
[0x4ba20]: 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5
[0x4ba30]: 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5
[0x4ba40]: 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5
[0x4ba50]: 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5
[0x4ba60]: 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5

[LR]:0x10645
[0x105c0]: 0xf0274850 0xf002fac7 0x4601fca5 0xf027484e
[0x105d0]: 0xf002fac1 0x4601fcad 0xf027484c 0xf004fabb
[0x105e0]: 0x4601fb11 0xf027484a 0xf44ffab5 0x48494100
[0x105f0]: 0xfab0f027 0xf027200a 0x4947fac5 0xf0274847
[0x10600]: 0xf000faa9 0xf000fb5d 0x2108f8df 0xf0234844
[0x10610]: 0x2106fbed 0xf0014843 0x2100feaf 0xf0052006
[0x10620]: 0x2001fd77 0xfb88f023 0x483f2118 0xf8b4f023
[0x10630]: 0x21114a3e 0x20006813 0x0310f043 0x22226013
[0x10640]: 0xff60f7ff 0x483a4601 0xfa84f027 0x21012203
[0x10650]: 0xf7ff2002 0x4601ff57 0xf0274835 0x4835fa7b
[0x10660]: 0xff04f7ff 0xf7ff4834 0x4834ff01 0xfefef7ff
[0x10670]: 0x7010f242 0xfa90f025 0xf9a2f025 0x48304601
[0x10680]: 0xfa68f027 0xbf00e7f4 0x00010000 0x0003cb8b

```
[0x10690]: 0x00046fcc 0x0003cb9f 0x00046fd4 0x0003cbb1
[0x106a0]: 0x000475c0 0x0003cbbe 0x000475c0 0x0003cbd0
[0x106b0]: 0x0004b590 0x000475c0 0x0003cbe2 0x0004b590

[PC]:0x10506
[0x10480]: 0xf0272204 0xb118fdf5 0x4621480c 0xfb62f027
[0x10490]: 0x490b4620 0xf000220d 0x2864ff81 0xd0084604
[0x104a0]: 0xf806f001 0xe8bd4621 0x46024010 0xf0012000
[0x104b0]: 0xbd10b815 0x0003cae5 0x0003cae7 0x0003caec
[0x104c0]: 0x00046fd4 0x4604b510 0x4b0a4a09 0x2800490a
[0x104d0]: 0x461abf18 0xf0274809 0xb124fb3d 0x48082106
[0x104e0]: 0xff4af001 0xf001e001 0x2000ff7b 0xbf00bd10
[0x104f0]: 0x0003d55d 0x00040477 0x0003cb52 0x0003cb60
[0x10500]: 0x0003cdfa 0xfb91b537 0x4605f4f0 0xf4f2fb94
[0x10510]: 0x9200460b 0x462a4621 0xf0274802 0x4620fb1b
[0x10520]: 0xbd30b003 0x0003cb77 0x4957b510 0xf0274857
[0x10530]: 0x4957fb11 0xf0274857 0x4957fb0d 0xf0274857
[0x10540]: 0x4957fb09 0xf0274857 0x4957fb05 0xf0274857
[0x10550]: 0x4c57fb01 0x48584957 0xfafcf027 0x48584957
[0x10560]: 0xfaf8f027 0x48574621 0xfaf4f027 0x48574956
[0x10570]: 0xfaf0f027 0x48574956 0xfaecf027 0x48574956

tasks state:
Name            State   Pri     HWM     Idx     StkCur  StkBot
main            R       3       148     1       0x4b8a8 0x4b598
IDLE            R       0       98      2       0x4bba0 0x4ba08
tcpip           B       3       432     8       0x4ff38 0x4f7f8
                B       4       472     4       0x4d788 0x4d028
                B       4       472     5       0x4e0d8 0x4d978
                B       3       470     7       0x4f570 0x4ee18
                B       4       426     6       0x4ed08 0x4e5a8
Tmr Svc         B       6       480     3       0x4c4f8 0x4bd78
```

5.2 分析异常信息

5.2.1 明确出错原因

由出错信息：

```
exception:6 happen!!
appos pstack:0x4b930 msp:0x7ffe0 psp:0x4b950
usage fault happen, UFSR:0x200
```

可知是 exception:6 也即 usage fault。UFSR:0x200 指明异常原因是“表示除法运算时除数为零（只有在 DIV_0_TRP 置位时才会发生）”。

5.2.2 根据异常信息找出出错函数

用 objdump 反汇编出原代码：make objdump 生成 xx.objdump 文件，文件位于 xx/gcc/ 目录下。由出错信息：

```
R14(LR):[0x4b964]: 0x00010645
R15(PC):[0x4b968]: 0x00010506
```

可知出错 PC 位置为 0x00010506，在汇编文件内：

```
int div_zero(int a, int b, int c)
{
    10504:      b537            push{r0, r1, r2, r4, r5, lr}
     int res;

     res = b/a;
    10506:      fb91 f4f0       sdiv  r4, r1, r0
     return 0;
}

int div_zero(int a, int b, int c)
{
    1050a:      4605            mov r5, r0
     int res;

     res = b/a;
     res = res/c;
    1050c:      fb94 f4f2       sdiv  r4, r4, r2
     return 0;
}

int div_zero(int a, int b, int c)
{
    10510:      460b            mov r3, r1

     res = b/a;
     res = res/c;
     printf("resule:%d %d %d %d\n", res, a, b, c);
    10512:      9200            str   r2, [sp, #0]
    10514:      4621            mov r1, r4
    10516:      462a            mov r2, r5
    10518:      4802            ldr   r0, [pc, #8]      ; (10524 <div_zero+0x20>)
```

```
    1051a:      f027 fb1b       bl    37b54 <iprintf>

     return res;
}
    1051e:      4620            mov r0, r4
    10520:      b003            add  sp, #12
    10522:      bd30            pop  {r4, r5, pc}
    10524:      0003cb77        .word      0x0003cb77
```

sdiv r4, r1, r0 为除法指令，除数是 R0，需要定位 R0 的值，根据打印 R0 为 0，因此是除零异常。

5.2.3 解析返回值信息

```
R14(LR):[0x4b964]: 0x00010645
```

通过 PC 值已经能定位到出错位置，当 PC 值无效时，需要通过 LR 进一步定位，方法如下： R14(LR)值的最后 1bit 指示当前代码为 thumb 模式，在汇编文件内，用 0x00010644 定位函数位置：

```
#ifdef __CONFIG_XIP_ENABLE
      board_xip_init();
      CTRL_DBG("XIP enable\n");
#endif
      writel(readl(0xe000ed14)|(1<<4), 0xe000ed14);
    10634:      6813            ldr    r3, [r2, #0]
      int div = div_zero(0x00, 0x11, 0x22);
    10636:      2000            movs       r0, #0
#endif
#ifdef __CONFIG_XIP_ENABLE
      board_xip_init();
      CTRL_DBG("XIP enable\n");
#endif
      writel(readl(0xe000ed14)|(1<<4), 0xe000ed14);
    10638:      f043 0310       orr.w      r3, r3, #16
    1063c:      6013            str    r3, [r2, #0]
      int div = div_zero(0x00, 0x11, 0x22);
    1063e:      2222            movs       r2, #34    ; 0x22
    10640:      f7ff ff60   bl     10504 <div_zero>
    10644:      4601            mov r1, r0
      printf("%d\n", div);
    10646:      483a            ldr    r0, [pc, #232]   ; (10730 <main_task+0x208>)
    10648:      f027 fa84       bl     37b54 <iprintf>
      div = div_zero(0x02, 0x1, 3);
    1064c:      2203            movs       r2, #3
    1064e:      2101            movs       r1, #1
```

```
10650:      2002            movs    r0, #2
10652:      f7ff ff57   bl      10504 <div_zero>
10656:      4601            mov r1, r0
  printf("%d\n", div);
10658:      4835            ldr     r0, [pc, #212]  ; (10730 <main_task+0x208>)
1065a:      f027 fa7b       bl      37b54 <iprintf>

  extern void main_cmd_exec(char *cmd);
```

10644 是函数 div_zero 被调用的下一条指令。

5.2.4 解析栈信息

```
stack info:
[0x4b970]: 0x00000000 0x00000011 0x00000022 0x0004b590
[0x4b980]: 0xa5a5a5a5 0x00010645 0xa5a5a5a5 0x00036531
[0x4b990]: 0xa5a5a5a5 0xa5a5a5a5 0x00000068 0xfffffffc
[0x4b9a0]: 0x0004b8a8 0x7ce1ec9b 0x0004b484 0x0004b484
[0x4b9b0]: 0x0004b9a0 0x0004b47c 0x00000004 0x0004b4ec
```

由出错地方的汇编知，函数调用时，首先执行了：

```
10504:    b537              push{r0, r1, r2, r4, r5, lr}
```

即入栈了 R0, R1, R2, R4, R5, LR，因此对应关系如下：

```
R0 <-> 0x00000000
R1 <-> 0x00000011
R2 <-> 0x00000022
R4 <-> 0x0004b590
R5 <-> 0xa5a5a5a5
LR <-> 0x00010645
```

5.2.5 解析 CPU 寄存器

```
CPU registers:
R00:[0x4b950]: 0x00000000
R01:[0x4b954]: 0x00000011
R02:[0x4b958]: 0x00000022
R03:[0x4b95c]: 0x00000210
R04:[0x4b930]: 0x0004b590
R05:[0x4b934]: 0xa5a5a5a5
R06:[0x4b938]: 0xa5a5a5a5
```

```
R07:[0x4b93c]: 0xa5a5a5a5
R08:[0x4b940]: 0xa5a5a5a5
R09:[0x4b944]: 0xa5a5a5a5
R10:[0x4b948]: 0xa5a5a5a5
R11:[0x4b94c]: 0xa5a5a5a5
R12:[0x4b960]: 0x00043b0e
R14(LR):[0x4b964]: 0x00010645
R15(PC):[0x4b968]: 0x00010506
xPSR:[0x4b96c]: 0x21000000
SHCSR:0x00070008 step:0
```

比如分析 R3，在出错时，并未用到 R3，可知 R3 的值是调用之前的值，通过查找 div_zero 函数之前的汇编：

```
10638:   f043 0310       orr.w     r3, r3, #16
    10634:    6813          ldr    r3, [r2, #0]
```

知：R3 应是 writel(readl(0xe000ed14)|(1<<4), 0xe000ed14); 中的 readl(0xe000ed14)|(1<<4)，即 readl(0xe000ed14)|(1<<4) 为 0x0210。

5.2.6 解析线程信息

```
tasks state:
Name            State   Pri     HWM     Idx     StkCur  StkBot
main            R       3       148     1       0x4b8a8 0x4b598
IDLE            R       0       98      2       0x4bba0 0x4ba08
tcpip           B       3       432     8       0x4ff38 0x4f7f8
                B       4       472     4       0x4d788 0x4d028
                B       4       472     5       0x4e0d8 0x4d978
                B       3       470     7       0x4f570 0x4ee18
                B       4       426     6       0x4ed08 0x4e5a8
Tmr Svc         B       6       480     3       0x4c4f8 0x4bd78
```

可知，当前有 8 个线程，其中 Idx 为 4,5,6,7 的没有 Name，main 线程状态为 Ready，优先级为 3，栈剩余最少时，有 148 个字节，栈底为 0x4b598，最近一次调度是，使用栈位置为 0x4b8a8， main 线程创建时，栈大小为 1K，因此栈空间为： [0x4b598 0x4b998)。出异常时，栈在 0x4b930，因此落在 main 线程栈中，也能说明是 man 这个任务发生的异常。

5.2.7 取指异常

如果指令部分被修改了，导致取指异常，通常 PC 是无效的，如果 LR 仍有效的话，会打出 LR 前后的指令，对比原指令，可确定指令是否又被修改。

附录 A： 异常类型

表 A-1 异常类型

编号	类型	优先级	简介
0	N/A	N/A	没有异常在运行
1	复位	-3（最高）	复位
2	NMI	-2	不可屏蔽中断（来自外部 NMI 输入脚）
3	硬（hard）fault	-1	所有被除能的 fault，都将“上访”成硬 fault。除能的原因包括当前被禁用，或者被 PRIMASK 或 BASPRI 掩蔽
4	MemManage fault	可编程	存储器管理 fault，MPU 访问犯规以及访问非法位置均可引发。企图在“非执行区”取指也会引发此 fault
5	总线 fault	可编程	从总线系统收到了错误响应，原因可以是预取流产（Abort）或数据流产，或者企图访问协处理器
6	用法（usage）fault	可编程	由于程序错误导致的异常，通常是使用了一条无效指令，或者是非法的状态转换，例如尝试切换到 ARM 状态
7-10	保留	N/A	N/A
11	SVCall	可编程	执行系统服务调用指令（SVC）引发的异常
12	调试监视器	可编程	调试监视器（断点，数据观察点，或者是外部调试请求）
13	保留	N/A	N/A
14	PendSV	可编程	为系统设备而设的“可悬挂请求”（pendable request）
15	SysTick	可编程	系统滴答定时器（也就是周期性溢出的时基定时器）——译注
16	IRQ #0	可编程	外中断#0
17	IRQ #1	可编程	外中断#1
...	...	...	...
255	IRQ #239	可编程	外中断#239

附录 B：SHCSR 信息

表 B-1 SHCSR 信息

位	名称	描述
[31:19]	-	保留
[18]	USGFAULTENA	用法 Fault 使能位，设为 1 时使能
[17]	BUSFAULTENA	总线 Fault 使能位，设为 1 时使能
[16]	MEMFAULTENA	存储器管理 Fault 使能位，设为 1 使能
[15]	SVCALLPENDED	SVC 调用挂起位，如果异常挂起，该位读为 1
[14]	BUSFAULTPENDED	总线 Fault 异常挂起位，如果异常挂起，该位读为 1
[13]	MEMFAULTPENDED	存储器 Fault 故障异常挂起位，如果异常挂起，该位读为 1
[12]	USGFAULTPENDED	用法 Fault 异常挂起位，如果异常挂起，该位读为 1
[11]	SYSTICKACT	SysTick 异常有效位，如果异常有效，该位读为 1
[10]	PENDSVACT	PendSV 异常有效位，如果异常有效，该位读为 1
[9]	-	保留
[8]	MONITORACT	调试监控有效位，如果调试监控有效，该位读为 1
[7]	SVCALLACT	SVC 调用有效位，如果 SVC 调用有效，该位读为 1
[6:4]	-	保留
[3]	USGFAULTACT	用法 Fault 异常有效位，如果异常有效，该位读为 1
[2]	-	保留
[1]	BUSFAULTACT	总线 Fault 异常有效位，如果异常有效，该位读为 1
[0]	MEMFAULTACT	存储器管理 Fault 异常有效位，如果异常有效，该位读为 1

附录 C：MFSR 信息

表 C-1 MFSR 信息

位	名称	描述
[7]	MMARVALID	存储器管理 Fault 地址寄存器（MMAR）有效标志： 0：MMAR 中的值不是一个有效 Fault 地址 1：MMAR 中保留一个有效 Fault 地址。 如果发生了一个存储器管理 Fault，并由于优先级的原因升级成一个硬 Fault，那么硬 Fault 处理程序必须将该位设为 0。
[6:5]	-	保留
[4]	MSTKERR	进入异常时的入栈操作引起的存储器管理 Fault： 0：无入栈 Fault 1：进入异常时的入栈操作引起了一个或一个以上的访问违犯。 当该位设为 1 时，依然要对 SP 进行调节，并且堆栈的上下文区域的值可能不正确。处理器没有向 MMAR 中写入 Fault 地址。
[3]	MUNSTKERR	异常返回时的出栈操作引起的存储器管理 Fault： 0：无出栈 Fault 1：异常返回时的出栈操作已引起一个或一个以上的访问违犯. 该 Fault 与处理程序相连，这意味着当该位为 1 时，原始的返回堆栈仍然存在。 处理器不能对返回失败的 SP 进行调节，并且不会执行新的存储操作。处理器没有向 MMAR 中写入 Fault 地址。
[2]	-	保留
[1]	DACCVIOL	数据访问违犯标志： 0：无数据访问违犯 Fault 1：处理器试图在不允许执行操作的位置上进行加载和存储。 当该位为 1 时，异常返回的压入堆栈的 PC 值指向出错指令。处理器已在 MMAR 中加载了目标访问的地址。
[0]	IACCVIOL	指令访问违犯标志： 0：无指令访问违犯错误 1：处理器试图从不允许执行操作的位置上进行指令获取。 即使 MPU 被禁能，这一故障也会在 XN(CM3 内核的 0xE0000000~0xFFFFFFFF 区域）区寻址时发生。 当该位为 1 时，异常返回的压入堆栈的 PC 值指向出错指令。处理器没有向 MMAR 中写入故障地址。

附录 D： BFSR 信息

表 D-1 BFSR 信息

位	名称	描述
[7]	BFARVALID	总线 Fault 地址寄存器（BFAR）有效标志： 0：BFAR 中的值不是有效故障地址 1：BFAR 中保留一个有效故障地址。 在地址已知的总线故障发生后处理器将该位设为 1。该位可以被其他 Fault 清零，例如之后发生的存储器管理 Fault。 如果发生总线 Fault，并由于优先级原因升级为一个硬 Fault，那么硬 Fault 处理程序必须将该位设为 0。
[6:5]	-	保留
[4]	STKERR	进入异常时的入栈操作引起的总线 Fault： 0：无入栈故障 1：进入异常时的入栈操作已引起一个或一个以上的总线故障。 当处理器将该位设为 1 时，依然要对 SP 进行调节，并且堆栈的上下文区域的值可能不正确。处理器没有向 BFAR 中写入 Fault 地址。
[3]	UNSTKERR	异常返回时的出栈操作引起的总线 Fault： 0：无出栈 Fault 1：异常返回时的出栈操作已引起一个或一个以上的总线 Fault。 该 Fault 与处理程序相连， 这意味着当处理器将该位设为 1 时，原始的返回堆栈仍然存在。处理器不能对返回失败的 SP 进行调节，并且不会执行新的存储操作，也未向 BFAR 中写入 Fault 地址。
[2]	IMPRECISERR	非精确数据总线错误： 0：无非精确数据总线错误 1：已发生一个数据总线错误，但是堆栈帧中的返回地址与引起错误的指令无关。 当处理器将该位设为 1 时，不向 BFAR 中写入 Fault 地址。 这是一个异步 Fault。因此，如果在当前进程的优先级高于总线 Fault 优先级时检测到该 Fault，总线 Fault 被挂起并仅在处理器从所有更高优先级进程中返回时开始变为有效。如果在处理器进入非精确总线 Fault 的处理程序前发生一个精确 Fault，那么处理程序同时对 IMPRECISERR 和其中一个精确 Fault 状态位进行检测，判断它们是否置位为 1。
[1]	PRECISERR	精确数据总线错误： 0：非精确数据总线错误 1：已发生一个数据总线错误，且异常返回的压入堆栈的 PC 值指向引起 Fault 的指令。 当处理器将该位设为 1 时，向 BFAR 中写入 Fault 地址。

位	名称	描述
[0]	IBUSERR	指令总线错误： 0：无指令总线错误 1：指令总线错误。 处理器检测到预取指令时的指令总线错误，但仅在其试图签发 Fault 指令时才将 IBUSERR 标志设为 1。 当处理器将该位设为 1 时，不向 BFAR 中写入 Fault 地址。

附录 E： UFSR 信息

表 E-1 UFSR 信息

位	名称	描述
[15:10]	-	保留
[9]	DIVBYZERO	0：无除以零 Fault 或除以零捕获未使能 1：处理器已执行 SDIV 或 UDIV 指令（除以零）。 当处理器将该位设为 1 时，异常返回的压入堆栈的 PC 值指向执行除以零的指令。 注：通过将 CCR 中的 DIV_0_TRP 位设为 1 使能除以零捕获，默认是不使能的。
[8]	UNALIGNED	0：无非对齐访问 Fault，或非对齐访问捕获未使能 1：处理器已进行了一次非对齐的存储器访问。 注：通过将 CCR 中的 UNALIGN_TRP 位设为 1 来使能非对齐访问捕获，默认是不使能的。非对齐的 LDM、STM、LDRD 和 STRD 指令总是出错，与 UNALIGN_TRP 的设置无关。
[7:4]	-	保留
[3]	NOCP	无协处理器用法 Fault。处理器不支持协处理器指令： 0：试图访问一个协处理器未引起用法 Fault 1：处理器已试图访问一个协处理器。
[2]	INVPC	EXC_RETURN 的无效 PC 加载引起的无效 PC 加载用法 Fault： 0：没有发生无效 PC 加载用法 Fault 1：处理器已试图将 EXC_RETURN 非法载入 PC，作为一个无效的上下文或一个无效的 EXC_RETURN 值。 当该位被设为 1 时，异常返回的压入堆栈的 PC 值指向尝试执行非法 PC 加载的指令。
[1]	INVSTATE	无效状态用法 Fault： 0：未发生无效状态用法 Fault 1：处理器已试图执行一个非法使用 EPSR 的指令。 当该位设为 1 时，异常返回的压入堆栈的 PC 值指向一个尝试非法使用 EPSR 的指令。 如果一个未定义的指令使用了 EPSR，则该位不被置位为 1。
[0]	UNDEFINSTR	未定义的指令用法 Fault: 0：无未定义的指令用法 Fault 1：处理器已试图执行一个未定义的指令。当该位设为 1 时，异常返回的压入堆栈的 PC 值指向未定义的指令。 未定义的指令是一条不能被处理器译码的指令。

附录 F： HFSR 信息

表 F-1 HFSR 信息

位	名称	描述
[31]	DEBUGEVT	硬 Fault 因调试事件产生，保留供调试使用。对寄存器执行写操作时，必须向该位写入 0；否则，该行为不可预知。
[30]	FORCED	指示硬 Fault 是否由上访产生，非硬 Fault 的处理程序无法执行时，会上访成硬 Fault。 0：硬 Fault 不是因为非硬 Fault 上访产生的 1：硬 Fault 是通过非硬 Fault 上访产生的。 当该位设为 1 时，硬 Fault 处理程序必须读其他 Fault 状态寄存器以找出 Fault 原因。
[29:2]	-	保留
[1]	VECTTBL	指示一个在异常处理过程中读向量表而引起的总线 Fault： 0：读向量表未引起总线 Fault 1：读向量表引起了总线 Fault。 这一错误通常情况下都由硬 Fault 处理程序来处理。
[0]	-	保留

附录 G： MMAR 信息

图 G-1 存储管理地址寄存器（MMAR）信息

位段	名称	类型	复位值	描述
31：0	MMAR	R	-	触发存储管理 fault 的地址

附录 H： BFAR 信息

图 H-1 总线 fault 地址寄存器（BFAR）信息

位段	名称	类型	复位值	描述
31：0	BFAR	R	-	触发总线 fault 的地址

附录 I： FPSCR 信息

表 I-1 FPSCR 信息

位	名称	描述
[31-6]	xxx	xxx
[5]	IDC	Input Denormal cumulative exception bit; value is 1 when floating point exception occurred.
[4]	IXC	Inexact cumulative exception bit; value is 1 wen floating point exception occurred.
[3]	UFC	Underflow cumulative exception bit; value is 1 when floating point exception occurred.
[2]	OFC	Overflow cumulative exception bit; value is 1 when floating point exception occurred.
[1]	DZC	Division by Zero cumulative exception bit; value is 1 when floating point exception occurred.
[0]	IOC	Invalid Operation cumulative exception bit; value is 1 when floating point exception occurred.

著作权声明

商标声明

免责声明