Application Report
ZHCA072– November 2009

TI C64x+ DSP 内核异常处理机制的应用

崔晶 TI 通用数字信号处理系统技术支持

摘要

很多 DSP 工程师都会遇到程序跑飞，系统死机等一系列复杂棘手的问题。特别是一些软件问题很难复现，毫无规律，甚至需要系统运行几天或者在特定的场景下才会出现。而问题出来后要么连不上仿真器，要么连上仿真器后 PC 指针早已跑飞，很难找到线索解决问题。德州仪器（TI）的 C64x+ DSP 内核是 C64x DSP 内核的扩展升级版，其中显著的两个特点是增加了异常（Exception）和内存保护（Memory Protection）机制。这个两个机制可以配合使用，用来检测、报告和处理一些错误。比如在运行过程中如果 CPU 读写了片内的程序段或者 ROM 空间就可以产生一个异常中断，这个时候就可以查看系统的调用堆栈，分析错误原因。本文将介绍异常处理和内存保护机制的原理和典型应用，旨在对 C64x+ DSP 工程师有所帮助，在实际工作中借助异常处理和内存保护机制定位调试并最终解决问题。

内容

图

1 异常（Exception）处理机制

在 DSP 算法比较复杂的系统，比如包括多路音视频编解码、网络以及和主机通过 PCI 通信的 DSP 系统中，我们常常会遇到系统稳定性的问题。例如系统运行一段时间以后发生死机问题，主机无法通过 PCI 和 DSP 通信，也不知道具体是哪个线程出问题？而且问题很难复现，不是每次必现，甚至要跑一周才会出问题。或者说对特殊的码流进行编码时系统会死机，但又不知道问题具体出在编码算法的什么地方？这个时候工程师最希望的是能捕捉到出错时的状态，最好能有地方记录错误原因和检查出错前的函数调用堆栈。在发生死机之后，我们往往连不上仿真器不能通过 JTAG 查看 DSP 的相关寄存器来分析出错的原因。如果连上了仿真器，又发现 DSP 常常停在 ROM 或一些无效的地址空间，很难找到线索来判断到底是在哪个函数或环节出了问题。而有了异常处理机制，我们就可以在程序中注册一个异常中断服务程序， 当 DSP 运行出错或内存保护机制检测到错误时会跳转到该异常中断服务程序，同时相关的寄存器会记录和报告错误的原因。C64x+ DSP 异常处理机制可以捕捉到的异常包括 CPU 内部的异常（如访问非法地址、执行非法指令）和 CPU 之外的异常等等。

在产品的开发和测试阶段，我们可以在异常中断服务程序里加上调试打印信息和死循环。通过调试打印信息我们可以知道有异常发生。死循环可以保证程序不跑飞，并可以成功连上仿真器。连上仿真器后，我们就可以通过检查函数调用堆栈和具体的异常相关寄存器很快的找到问题产生的原因。在最终的产品软件中，我们也可以通过异常处理机制做一些系统的自恢复处理，比如在异常中断服务程序里复位 DSP，或者发信号给主机报告 DSP 异常请求主机做整个系统复位等等。

传统的 C64x DSP 的中断控制器可以输出三种中断信号给 CPU：复位中断、不可屏蔽中断（NMI）和可屏蔽中断（INT4~INT15）（注：不是所有的 C6000 DSP 都支持 NMI）。如图 1 所示 C64x+ DSP 的中断控制器增加了一个可屏蔽异常中断信号（EXCEP）的输出。如果使能了异常处理机制，不可屏蔽中断 NMI 就被当做是一个不可屏蔽异常信号（EXCEP）；如果没使能 EXCEP，不可屏蔽中断 NMI 就和 C64x DSP 的 NMI 一样被当做一个中断信号直接输出给 CPU。EXCEP 和 NMI 中断（不使能异常时）共享一个中断矢量，也就是说它们在中断矢量表的位置是相同的。

异常处理机制用来支持错误检测，并可以产生异常中断（EXCEP），这样就可以用相应的中断服务程序处理具体的异常错误。

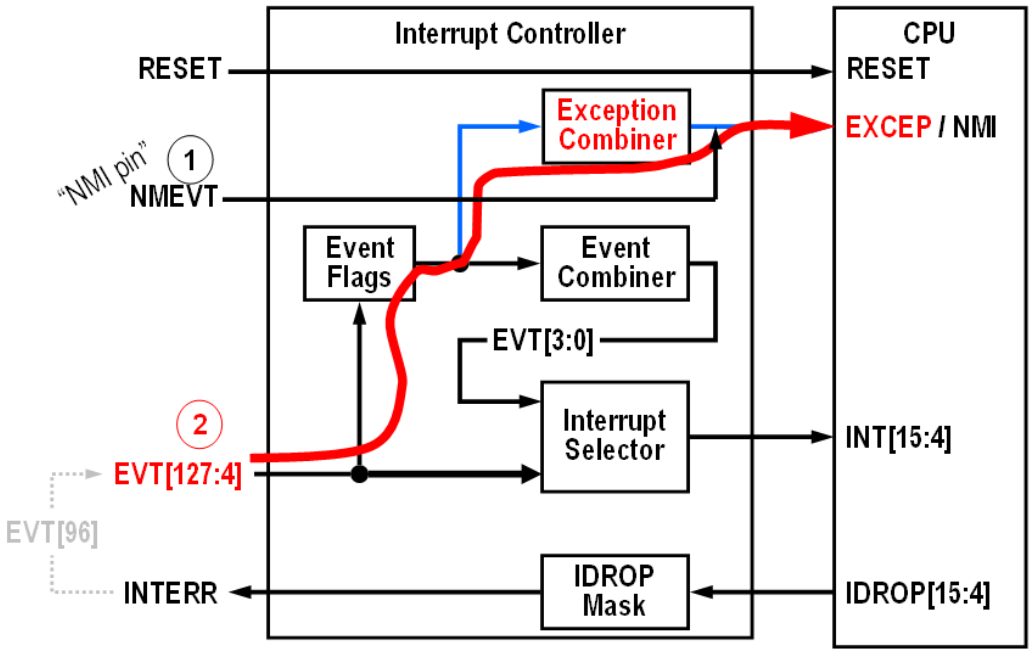

图 1. C64x+ DSP 中断控制器示意图

2 C64x+ DSP 异常中断的四种类型

C64x+ DSP 内核的异常包括以下四种类型：

- 外部（内核之外）产生的可屏蔽异常中断（EXF）。例如系统外设事件和内存保护触发的中断。
- 外部（内核之外）产生的不可屏蔽异常中断（NXF）。就是 DSP 的 NMI 引脚中断。
- 内核产生的内部不可屏蔽异常中断（IXF）。
- 软件（SWE 和 SWENR 指令）产生的软件异常中断（SXF）。

当异常产生时，我们可以通过读异常标志寄存器（EFR）来知道产生的异常中断的具体类型，并执行相应的中断服务程序。关于异常标志寄存器的具体信息，请参考 spru732.pdf。

2.1 外部不可屏蔽异常中断和外部可屏蔽异常中断

图 1 中所示的①就是传统 C64x DSP 的外部不可屏蔽中断 NMI，是由 DSP 的 NMI 引脚触发。我们前面提到在 C64x+ DSP 中如果使能了异常处理机制，NMI 就被当做是异常中断，就是我们这里所说的外部不可屏蔽异常中断（NXF）。

如图 1 所示，通常我们可以把 DSP 的 4 号到 127 号中断事件通过中断选择器路由到 CPU 4 至 15 号中断。同时我们也可以根据自己的实际系统应用把 4 号到 127 号中断事件通过异常组合器（Exception Combiner）路由到 CPU 的异常中断。在这种情况下，EVT[127:4]产生的中断就是 C64x+ DSP 的外部可屏蔽异常中断（EXF）。比如实际的系统中根本没有用到 DSP 外设 McBSP，我们就可以在中断初始化的时候把 McBSP 的接收或发送中断路由到异常中断，如果系统运行过程中有 McBSP 的接收或发送中断产生，这个中断就会被当做异常中断处理。图 1 的②就是这种应用的示例。通常我们会使能 L1P/L1D/L2 CPU 内存保护事件来触发异常中断，具体将在典型应用中介绍。

2.2 内部不可屏蔽异常中断

内部不可屏蔽异常中断（IXF）是 C64x+ DSP 内核产生的不可屏蔽异常中断，典型原因有：

- CPU 跳到 32 位指令的中间。
- CPU 跳到取址包的头。
- 执行不合法或者保留的指令。
- 试图执行保留的操作数。
- 用户模式下试图访问管理员模式才可访问的内存（特权违反）。
- SPLOOP buffer 异常。

当 IXF 发生时，可以从内部异常报告寄存器（IERR）找到具体的内部异常原因。关于内部异常报告寄存器的具体信息，请参考 spru732.pdf。

2.3 软件异常中断

软件异常中断（SXF）是指执行软件异常指令（SWE）或软件异常无返回指令（SWENR）触发的中断。当 CPU 执行了 SWE 指令，异常检测逻辑会检测到信号，EFR 的 SXF 位会被置位。如果异常中断被使能，CPU 会执行相应的中断服务程序，返回地址被存入 NRP 寄存器，并且操作模式也被改为管理员模式（Supervisor mode）(注：管理员模式是 C64x+ CPU 权利模式的一种，具体参考 spru732.pdf 的 C64x+ CPU Privilege 章节)。用户模式的任务可以通过调用这条指令来请求系统服务，这个服务可以是管理员模式下才可以做的 memory、寄存器或其它资源的访问。SWENR 指令和 SWE 指令类似，也会产生一个软件异常。不同的是调用 SWENR 后任务状态寄存器（TSR）的内容不会被拷贝到不可屏蔽中断/异常任务状态寄存器（NTSR），并且返回地址不会被放在不可屏蔽中断返回指针寄存器（NRP）中。此外，SWE 指令产生的软件异常中断会让 CPU 跳转到 NMI 中断矢量表入口处，而 SWENR 指令产生的软件异常中断会让 CPU 跳转到 REP(Restricted entry point)寄存器中的地址。假如我们在管理员模式的任务程序中调用了一个用户模式的任务，就可以用 SWENR 指令中止这个用户模式的任务并返回调用它的管理员模式下的任务程序。

3 异常处理机制是如何工作的

和通常的 DSP 中断机制类似，我们可以把异常处理机制分为五个步骤：使能异常中断→异常发生→相应标志位置位→响应异常中断→执行异常中断服务程序。

3.1 使能异常中断

如图 2 所示要使能异常中断，必须设置寄存器 TSR 的 XEN 和 GEE 位及 IER 的 NMIE 位为 1。要注意的是，如果使用了 DSP/BIOS，那么 BIOS_init()程序中会设置 IER.NMIE。

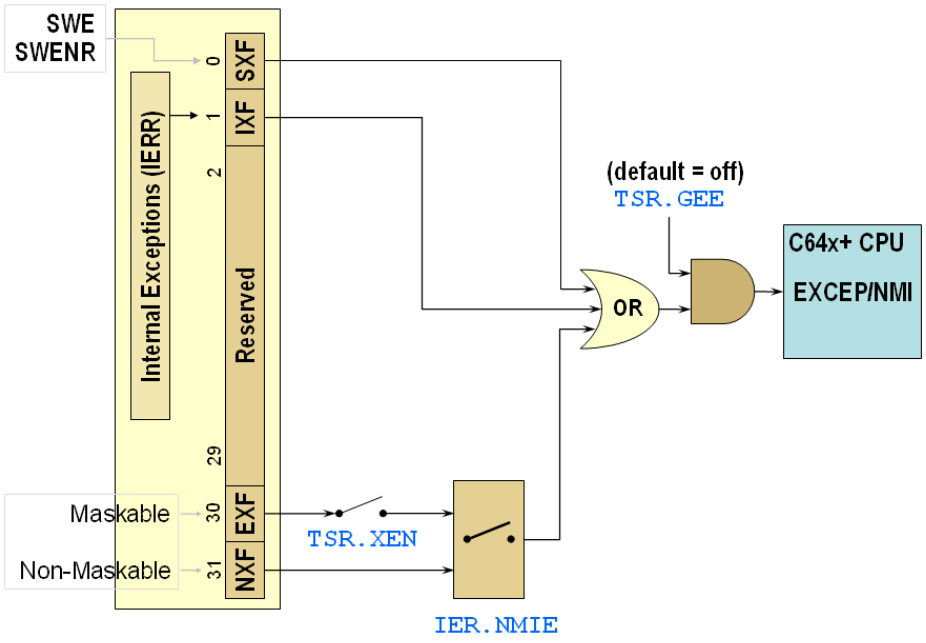

图 2. 异常中断使能示意图

3.2 异常中断产生和标志位置位

正确使能异常中断后，第 2 章开始提到的四种类型的异常事件都可以产生异常中断， EFR 寄存器的相应位也会被置位。

3.3 响应异常中断

CPU 会自动应答异常中断。以下操作将被执行：

1）CPU 停止正在做的工作。
2）保持 EFR 的标志位。
3）关闭 IER.NMIE。
4）TSR 的内容被拷贝到 NTSR。
5）TSR 的变化：EXC 位置位（表示异常处理中），CXM 被清零（进入管理员模式），GIE/SGIE/XEN/DBGM/INT 位都被清零。
6）保存返回地址到寄存器 NRP（SWENR 产生的异常除外）。
7）跳转到 NMI 中断矢量表入口处（SWENR 产生的异常导致 CPU 跳转到寄存器 REP 中的地址）。

3.4 异常中断服务程序

异常中断服务程序需要执行以下操作：

1）保存系统的现场信息（异常中断服务程序需要用到的所有寄存器的内容）。注意，如果用 DSP/BIOS 硬件中断调度器，不需要这个操作。
2）检查触发异常中断的原因：
A．检查 EFR 寄存器决定中断源。

B．如果 EFR.IXF 置位，读内部异常报告寄存器 IERR。
C．如果 EFR.EXF 置位，读寄存器 MEXPFLAG[3:0]。
D．清除 EFR、IERR、EVTFLAG 和 MEXPFLAG 中的所有置位的标志位。

3）执行自己的异常中断处理程序。
4）我们知道调用一条指令包括取址、译码和执行等阶段，而异常中断可能在这些阶段的任何时候发生。因此，在异常中断返回之前我们需要检查中断是否可以正常返回。NTSR 寄存器的 IB、GIE 和 SPLX 位都为 0，表示异常中断可以正常返回。否则，表示异常中断不能返回，需要执行不能返回对应的程序。
5）恢复系统的现场信息。注意如果用 DSP/BIOS 硬件中断调度器（dispatcher），不需要这个操作。
6）CPU 跳转到寄存器 NRP 中的地址。（把 TSR 的内容从 NTSR 中恢复回来）。

4 异常处理机制的典型应用

除了应用第 1 节中提到的内部不可屏蔽中断之外，异常处理机制的典型应用更多的是外部可屏蔽中断。我们前面提到系统根本没用到的 DSP 外设产生的中断可以完全被我们当做一个异常中断来处理，这里我们着重介绍 C64x+DSP 内核内存保护触发异常处理机制的典型应用。以 TI 的 DSP DM648 为例，它的中断事件 EVT[120~126]都是内存保护产生的事件。比如我们可以根据自己系统的应用，设置在 CPU 试图去写或读 L2 里的程序段时产生一个异常中断。因为这个程序段的内容都是被 CPU 执行的，如果 CPU 试图去写或读这个空间就表明系统已经异常，可以通过异常中断做相应的处理。

4.1 C64x+的内存保护

内存保护允许系统设置谁被授权访问 L1D、L1P 和 L2。为了实现内存保护机制，L1D、L1P 和 L2 的 memory 又被分为页(pages)。以 DM648 为例，L1D 有 16 个页（2K 字节每页），L1P 有 16 个页（2K 字节每页），L2 有 32 页（128K 字节每页）。需要注意的是 L2 的所有页是连续的 4M 字节地址范围 0x00800000~0x00BFFFFF，覆盖了 ROM、L2RAM 和保留(Reserved)地址范围。C64x+的 DMC（L1D Memory Controller）、PMC（L1P Memory Controller）和 UMC（L2 Memory Controller）有各自的一套内存保护页属性寄存器（MPPA），可以用来定义每一个内存页的许可权利（读、写或者执行）。L1DMPPA[31:16]对应 L1D 的 16 个内存页，L1PMPPA[31:16]对应 L1P 的 16 个内存页，L2MPPA[31:0]对应 L2 的 32 个内存页。其中 L1DMPPA16、L1PMPPA16 和 L2MPPA0 控制相应内存范围的起始地址最小的页。换句话说，L1DMPPA16 控制地址从 0x00F00000 开始的页，L1PMPPA16 控制地址从 0x00E00000 开始的页，L2MPPA0 控制地址从 0x00800000 开始的页。（注：具体信息参考 DM648 DSP 子系统用户手册 sprueu6.pdf）

每一页可以设置完全不相关的用户和管理员的权利（读、写或执行），比如允许用户读、管理员写等等。此外，每一页可以被设为是本地或全局（或两者都可以）可访问的。本地访问是指 CPU 直接访问 L1D、L1P 或 L2。全局访问是指 DMA（IDMA 或 EDMA）或系统其他 master（可以主动发起访问的 DSP 外设，比如 PCI 或 EMAC）发起的访问。CPU 的特权 ID 为 0，所有系统其他 master 的特权 ID 为 1。因此，图 3 内存保护页属性寄存器中所示 AID0 和 AID1 位分别可以被用来控制来自 CPU 和系统其他 master（包括 DMA）的访问操作。而 LOCAL 位指 CPU 访问它本地的 L1D、L1P 和 L2。访问的类型包括：管理员模式读、管理员模式写、管理员模式执行、用户模式读、用户模式写和用户模式执行。如 SR 位设为 1，表示对应的内存页允许管理员模式读，为 0 表示不允许管理员模式读。

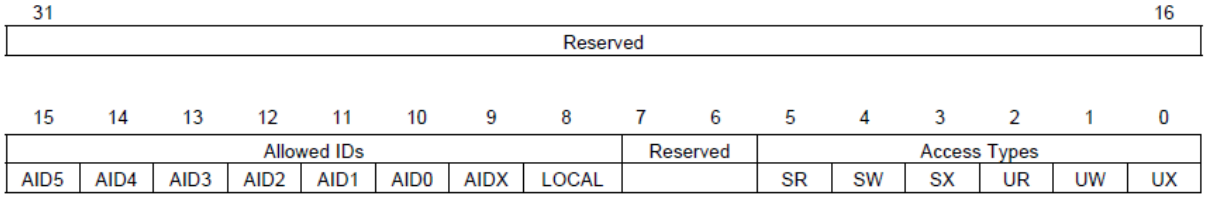

图 3. 内存保护页属性寄存器 MPPA

内存保护硬件会根据每一个内存保护页属性寄存器的设置执行两个任务：阻止未经许可的访问操作发生；根据检测到的无效访问产生相应的异常中断。C64x+内存保护机制有三类寄存器：内存保护故障地址寄存器（MPFAR）、内存保护故障状态寄存器（MPFSR）和内存保护故障命令寄存器（MPFCR）。当内存保护硬件检测

到未经许可的操作，除了执行刚才提到的两个任务之外，还会把具体发生故障的地址保存到 MPFAR、具体的故障类型（比如 CPU 写、用户模式读等）保存到 MPFSR。我们可以通过软件对 MPFCR 的第 0 位写 1 清除这个故障及对应 MPFAR 和 MPFSR 的内容。（注：关于内存保护的具体信息参考 spru871.pdf。）

4.2 内存保护触发异常中断的具体实现

TI 的 DSP/BIOS 通过 EXC 模块提供 C64x+ DSP 异常处理支持。用户在.tcf 中使能异常中断，并且在 DSP 的应用程序中调用 EXC 模块的 API 就可实现。具体信息可以参考 spru403.pdf。我们还可以参考 DSP/BIOS 例程，如 bios_5_31_07\packages\ti\bios\examples\advanced\目录下的 mpc_static 就是内存保护触发异常中断的例程。同一目录下的 mpc_priv 例子演示了一个使用用户/管理员模式的系统，所有的 DSP/BIOS 的 data 段被设置成只有管理员模式才能访问，其中一个任务被切换到用户模式访问 DSP/BIOS 的 data 段产生异常中断。同一目录下的 mpc_tsk 例子演示了任务级的动态内存保护。

下面我们介绍不使用 DSP/BIOS 的情况下，如何实现内存保护触发异常中断。我们需要在程序中添加几个函数：内存保护初始化、异常中断初始化、设置相应内存保护页属性寄存器和中断服务程序，并且还要修改中断矢量表。具体可以参考我们的示例程序 DM648_EXC.zip。

1） 内存保护初始化

```
#define EXC_EVTPMCCMPA  120             /* L1P CPU Memory Protection fault */
#define EXC_EVTDMCCMPA  122             /* L1D CPU Memory Protection fault */
#define EXC_EVTUMCCMPA  124             /* L2 CPU Memory Protection fault */
void MPC_init (void)
{
    /*写 1 到 MPFCR，清除 BOOT ROM 可能触发的内存保护异常*/
    L1PMPFCR = 1;
    L1DMPFCR = 1;
    L2MPFCR = 1;
    /*清除内存保护故障事件对应的 EVTFLAG 和 MEXPFLAG 标志位*/
    EVTCLR3 | = 0x01000000;
    EVTCLR3 | = 0x04000000;
    EVTCLR3 | = 0x10000000;
    /*使能内存保护故障事件*/
    EXPMASK3 & = ~0x01000000;
    EXPMASK3 & = ~0x04000000;
    EXPMASK3 & = ~0x10000000;
}
```

2） 异常中断初始化

```
#define EXC_TSRGEE      0x00000004
#define EXC_TSRXEN      0x00000008
void EXC_init(void)
{
    extern volatile cregister unsigned int TSR;
    extern volatile cregister unsigned int ISTP;
    extern volatile cregister unsigned int IER;
    extern volatile cregister unsigned ECR;
    extern volatile cregister unsigned NRP;
    extern volatile cregister unsigned NTSR;
    extern volatile cregister unsigned IERR;
    /*把中断矢量表的地址赋给 ISTP 寄存器*/
    ISTP = (unsigned int) & vectorTab;
    /*清除异常中断相关寄存器的状态*/
    ECR = 0xFFFFFFFF;
    NRP = 0;
    NTSR = 0;
    IERR = 0;
   /* 使能异常中断*/
    TSR |= EXC_TSRXEN | EXC_TSRGEE;
    IER = IER |2;                                          //IER.NMIE=1
```

```
}
```

3） 根据自己的应用设置相应内存保护页属性寄存器

举例，使能 16KB L1D cache，配置 L1D 的低 16KB 为 CPU 用户模式/管理员模式读。配置 L1P/L2 内存保护页属性寄存器的方法是类似的，需要注意的是自己要保护的地址范围和相应内存保护页属性寄存器的对应关系（参考本文 3.1 节和具体 DSP 的数据手册）。

```
#define MPC_MPPA_UX     0x00000001      /* User execute */
#define MPC_MPPA_UW     0x00000002      /* User Write */
#define MPC_MPPA_UR     0x00000004      /* User Read */
#define MPC_MPPA_SX     0x00000008      /* Supervisor execute */
#define MPC_MPPA_SW     0x00000010      /* Supervisor Write */
#define MPC_MPPA_SR     0x00000020      /* Supervisor Read */
#define MPC_MPPA_LOCAL  0x00000100      /* LOCAL CPU access */
void MPC_setBufferPA(void)
{
        unsigned int * p1;
        unsigned int i;
        p1 = (unsigned int *) 0x0184AE40;             //L1DMPPA16 地址
        for(i=0; i<8; i++)
              *p1++ = MPC_MPPA_UR | MPC_MPPA_SR | MPC_MPPA_LOCAL;
        for(i=0; i<8; i++)
              *p1++ = 0;                              //16KB cache
}
```

4） 中断矢量表

如下所示，把自己的 NMI 的中断服务程序 excInt( )添加到中断矢量表中。

```
.sect ".vectors"
              .align  32
              .ref _c_int00
              .def _vectorTab
_vectorTab:
_resetVec:
              mvkl _c_int00, b0
              mvkh _c_int00, b0
              b b0
              nop 5
              .align 32
              .ref _excInt
_nmi:
             stw b0, *b15--(8)
              mvkl _excInt,b0
              mvkh _excInt,b0
              b b0
              ldw *b15++(8),b0
              nop  4
              .align 32
_rsvd1:
              b _rsvd1
              nop 5
              .align 32
_rsvd2:
              b _rsvd2
              nop 5
```

5） 中断服务程序

根据自己的实际应用写相应的中断服务程序，以下仅供参考。

```
void interrupt excInt(void)
{
        extern volatile cregister unsigned EFR;
        extern volatile cregister unsigned ECR;
```

```
        extern volatile cregister unsigned NRP;
     extern volatile cregister unsigned NTSR;
     unsigned efr, nrp, ntsr;
     efr = EFR;   /*记录 EFR */
     nrp = NRP;   /*记录 NRP */
     ntsr = NTSR; /*记录 NTSR */
    /* 处理产生异常的所有可能事件 */
    if (efr & EXC_EFRSXF) {
        /* S/W generated exception */
        efr ^= EXC_EFRSXF;
        EXC_swgen();                    //软件产生的异常中断服务程序
    }
    if (efr & EXC_EFRIXF) {
        /* internal exception */
        efr ^= EXC_EFRIXF;
        EXC_internal();                 //内核产生的内部不可屏蔽异常中断
    }
    if (efr & EXC_EFREXF) {
        /* external exception */
        efr ^= EXC_EFREXF;
        EXC_external();                 //外部（内核之外）产生的可屏蔽异常中断
    }
    if (efr & EXC_EFRNXF) {
        /* legacy NMI exception */
        efr ^= EXC_EFRNXF;
        EXC_nmi();                      //外部（内核之外）产生的不可屏蔽异常中断，传统的 NMI 中断
    }
    /* 清除 EFR 的标志位 */
    ECR = efr;
    if(ntsr & 0xC001)      //NTSR 的 IB、GIE 和 SPLX.IB 不全为 0，异常中断不能安全返回
              while(1);                 //根据自己的应用修改
              }
void EXC_swgen(void)
{
              while(1);                 //根据自己的应用修改
}

void EXC_internal(void)
{
    extern volatile cregister unsigned IERR;
    unsigned int ierr;
    /*记录 IERR */
    ierr = IERR;
    if (ierr & EXC_IERRIFX) {
        printf("  Instruction fetch exception\n");
    }
    if (ierr & EXC_IERRFPX) {
        printf("  Fetch packet exception\n");
    }
    if (ierr & EXC_IERREPX) {
        printf("  Execute patcket exception\n");
    }
    if (ierr & EXC_IERROPX) {
        printf("  Opcode exception\n");
    }
    if (ierr & EXC_IERRRCX) {
        printf("  Resource conflict exception\n");
    }
    if (ierr & EXC_IERRRAX) {
```

```
        printf("  Resource access exception\n");
    }
    if (ierr & EXC_IERRPRX) {
        printf("  Privilege exception\n");
    }
    if (ierr & EXC_IERRLBX) {
        printf("  Loop buffer exception\n");
    }
    if (ierr & EXC_IERRMS) {
        printf("  Missed stall\n");
    }
    //根据自己的应用添加相应的处理程序
    /* 清除内部异常使得它们可以再次被检测到*/
    IERR = 0;
}

void EXC_external(void)
{
    unsigned int mpfsr,mpfar,flag;
    //printf("External exception\n");
    mpfsr=0;
    flag = MEXPFLAG3;
    if (flag & 0x01000000)              //L1P 内存保护故障
    {         mpfar = L1PMPFAR;
              mpfsr = L1PMPFSR;
              //根据自己的应用添加相应的处理程序
              /* 写 1 到 MPFCR，清除寄存器 MPFSR & MPFAR */
              L1PMPFCR =1;
              EVTCLR3 | = 0x01000000;
     }
    if (flag & 0x04000000)              //L1D 内存保护故障
    {         mpfar = L1DMPFAR;
              mpfsr = L1DMPFSR;
              //根据自己的应用添加相应的处理程序
              /* 写 1 到 MPFCR，清除寄存器 MPFSR & MPFAR */
              L1DMPFCR =1;
              EVTCLR3 | = 0x04000000;
     }
    if (flag & 0x10000000)              //L2 内存保护故障
    {         mpfar = L2MPFAR;
              mpfsr = L2MPFSR;
              //根据自己的应用添加相应的处理程序
              /* 写 1 到 MPFCR，清除寄存器 MPFSR & MPFAR */
              L2MPFCR =1;
              EVTCLR3 | = 0x10000000;
    }
    if (mpfsr & _MPC_MPFSR_SECE) {
                printf("  Security violation\n");
            }
    if (mpfsr & _MPC_MPFSR_UXE) {
                printf("  User Execute violation\n");
            }
    if (mpfsr & _MPC_MPFSR_UWE) {
                printf("  User Write violation\n");
            }
    if (mpfsr & _MPC_MPFSR_URE) {
                printf("  User Read violation\n");
            }
    if (mpfsr & _MPC_MPFSR_SXE) {
```

```
                printf("  Supervisor Execute violation\n");
            }
    if (mpfsr & _MPC_MPFSR_SWE) {
                printf("  Supervisor Write violation\n");
            }
    if (mpfsr & _MPC_MPFSR_SRE) {
                printf("  Supervisor Read violation\n");
            }
}

void EXC_nmi(void)
{
    printf("Legacy NMI exception\n");
    //根据自己的应用添加相应的处理程序
}
```

6） 应用程序

如下所示，如果 CPU 去写 L1D cache 的起始地址会产生异常中断，L1DMPFAR 寄存器记录了内存保护故障地址 0x00F04000，L1DMPFSR 寄存器内容为 0x110 表示是管理员模式写操作。执行中断服务程序，可以看到打印信息“Supervisor Write violation”。我们可以根据自己的实际应用修改程序 MPC_setBufferPA( )实现自己需要的内存保护处理。

```
#define L1DSRAM_CACHE_BASE 0x00F04000
void main(void)
{   volatile unsigned int * p;
    /* L1DCFG L1D Configuration. 16k cache, 16k addressable */
    L1DCFG = 3;
    MPC_init();
    EXC_init();
    MPC_setBufferPA();
    p= (unsigned int *) L1DSRAM_CACHE_BASE;
    *p=123;
}
```

5 总结

如 4.1 节提到的除了 L1P/L1D/L2 CPU 内存保护故障可产生异常中断之外，C64x+ DSP 还支持 DMA（IDMA、EDMA 或系统其他 master）内存保护故障产生异常中断，其原理和处理方式和 CPU 内存保护故障是类似的，这里不再详细描述。总之，我们可以利用异常中断和内存保护来调试自己的系统，帮助我们排查 CPU 跑飞或系统死机的原因。关于异常处理和内存保护的具体信息，还请参考 spru871.pdf 和 spru732.pdf（这两个文档可以在 TI 网页上下载）。

参考文档

1. *TMS320C64x+ DSP Megamodule Reference Guide* (SPRU871)
2. *TMS320C64x/C64x+ DSP CPU and Instruction Set* (SPRU732)

重要声明

德州仪器 (TI) 及其下属子公司有权在不事先通知的情况下，随时对所提供的产品和服务进行更正、修改、增强、改进或其它更改，并有权随时中止提供任何产品和服务。 客户在下订单前应获取最新的相关信息，并验证这些信息是否完整且是最新的。 所有产品的销售都遵循在订单确认时所提供的 TI 销售条款与条件。

TI 保证其所销售的硬件产品的性能符合 TI 标准保修的适用规范。 仅在 TI 保修的范围内，且 TI 认为有必要时才会使用测试或其它质量控制技术。 除非政府做出了硬性规定，否则没有必要对每种产品的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。 客户应对其使用 TI 组件的产品和应用自行负责。 为尽量减小与客户产品和应用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 产品或服务的组合设备、机器、流程相关的 TI 知识产权中授予的直接或隐含权限作出任何保证或解释。 TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服务的许可、授权、或认可。 使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它知识产权方面的许可。

对于 TI 的数据手册或数据表，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况下才允许进行复制。 在复制信息的过程中对内容的篡改属于非法的、欺诈性商业行为。 TI 对此类篡改过的文件不承担任何责任。

在转售 TI 产品或服务时，如果存在对产品或服务参数的虚假陈述，则会失去相关 TI 产品或服务的明示或暗示授权，且这是非法的、欺诈性商业行为。 TI 对此类虚假陈述不承担任何责任。

可访问以下 URL 地址以获取有关其它 TI 产品和应用解决方案的信息：

产品	
放大器	http://www.ti.com.cn/amplifiers
数据转换器	http://www.ti.com.cn/dataconverters
DSP	http://www.ti.com.cn/dsp
接口	http://www.ti.com.cn/interface
逻辑	http://www.ti.com.cn/logic
电源管理	http://www.ti.com.cn/power
微控制器	http://www.ti.com.cn/microcontrollers
应用	
音频	http://www.ti.com.cn/audio
汽车	http://www.ti.com.cn/automotive
宽带	http://www.ti.com.cn/broadband
数字控制	http://www.ti.com.cn/control
光纤网络	http://www.ti.com.cn/opticalnetwork
安全	http://www.ti.com.cn/security
电话	http://www.ti.com.cn/telecom
视频与成像	http://www.ti.com.cn/video
无线	http://www.ti.com.cn/wireless

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