Application Notes
ZHCA899 – 3 / 2019

AM57XX 处理器平台ARM和DSP通信的实现

DennyYang
TI 嵌入式系统技术支持工程师

摘要

TI的新一代处理器平台AM57XX是多核异构结构的SOC，片上有一到两个ARM核(ARM CORTEX-A15)和一到两个DSP(C66x)核。AM57xx处理器是高度集成的器件，可用于实现高性能和多媒体应用。板载加速器提供加速视觉和深度学习功能，支持多个工业以太网协议和视频处理。多核 SOC的软件相对单核系统比较复杂，TI 的 AM57XX 的软件包是 processor sdk（http://www.ti.com/tool/processor-sdk-am57x）。它有多个版本，LINUX 版本 ARM 运行 LINUX 系统，DSP 运行 SYSBIOS。RTOS 版本的 SDK，ARM 和 DSP 都运行 SYSBIOS 系统。

当前有很多工业或其他行业用户在 ARM 上运行 Linux，DSP 上不运行操作系统。需要有一套简单方便的方法实现 ARM 和 DSP 的通信。本文详细介绍了这种应用下 ARM 和 DSP 的通信机制并分析了具体例程。

目录

1 ARM 和 DSP通信机制 2
2 ARM 和 DSP通信例程 4
2.1 使用方法 4
2.2 ARM LINUX测试例程分析 5
2.3 DSP测试例程分析 6
2.4 ARM LINUX驱动分析 7
2.5 DSP代码加载例程分析 8
3 结束语 9
4 参考文献 9

图

图 1 系统 MAILBOX 构架 2
图 2 MAILBOX 内部构架 2
图 3 OPENCL 通信机制 3
图 4 IPC 通信机制 4

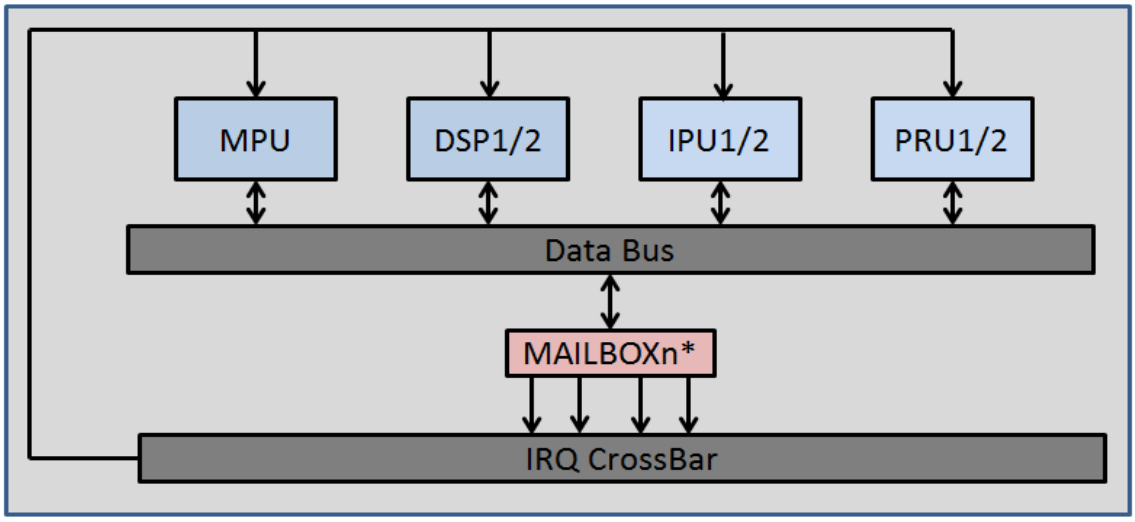
表

表 1 Mailbox Registers Mapping Summary 3

1 ARM 和 DSP通信机制

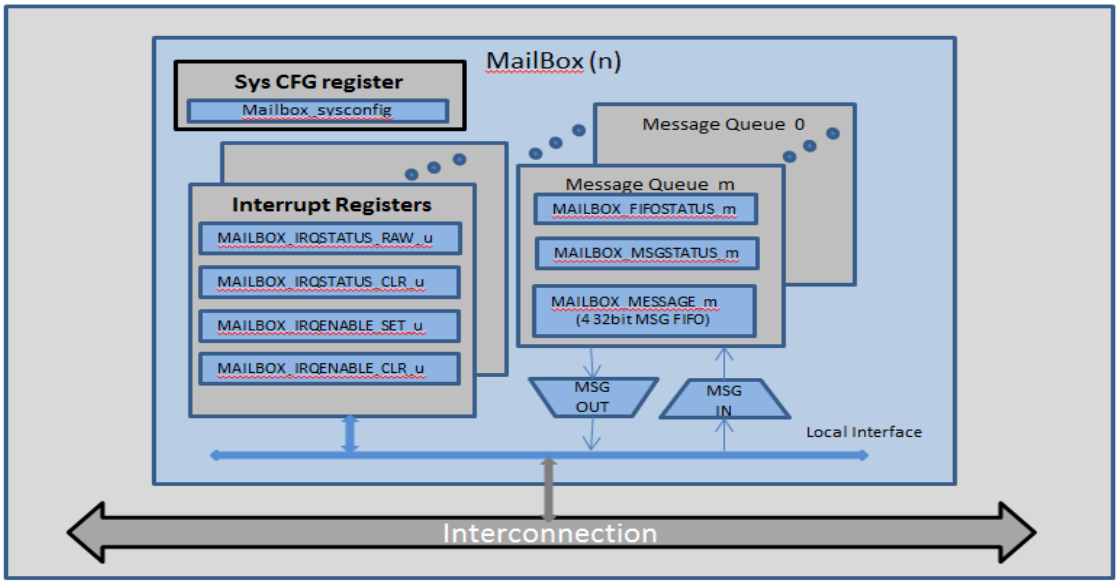
AM57XX系列芯片是目前 TI 推出的高端 SOC，片内包含 ARM A15 和 C66 DSP。主频从 500MHZ 到 1.5GHZ。目前广泛应用于电力，工业控制，高端 HMI,PLC，物联网智能网关等领域。片内包含硬件 MAILBOX 模块，SOC芯片内部各核之间可以通过MAILBOX实现互相触发中断和交换消息。SOC 内部各核通过数据总线来配置 MAILBOX 并读写消息。MAILBOX 通过系统 IRQ CROSSBAR 向系统各核心发送中断事件。如下图所示：

图 1 系统 MAILBOX 构架



设置 MAILBOX 相关寄存器后往 MAILBOX 的消息寄存器里写一个 32BIT 的消息可以让 MAILBOX 触发对应核的中断。然后相应的核就可以在中断处理函数里通过读 MAILBOX 的消息寄存器来获取对方发来的消息。

图 2 MAILBOX 内部构架



系统有多个 MAILBOX，每个都可以发出四个中断，每个中断都有一个专有的中断信号线。通过数据手册可以获知每个中断的中断事件号。
下表是 MAILBOX 的寄存器定义，其中 m 代表使用哪个 MAILBOX。u 代表四根中断线分别连的四个核。

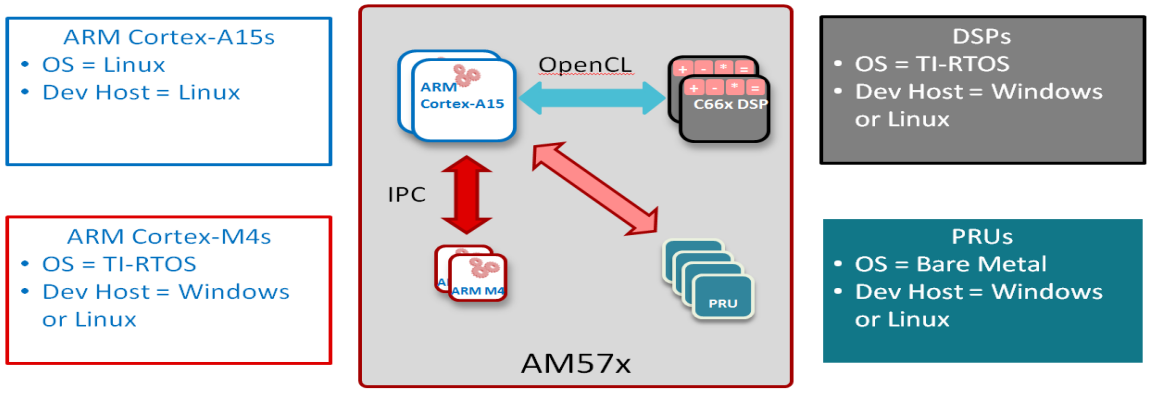
表 1 Mailbox Registers Mapping Summary

Offset	Acronym	Register Description
0000h	MAILBOX_REVISION	Mailbox Revision Register
0010h	MAILBOX_SYSCONFIG	Mailbox System Configuration Register
0040h+(4h*m)	MAILBOX_MESSAGE_m	Mailbox Message Register
0080h+(4h*m)	MAILBOX_FIFOSTATUS_m	Mailbox FIFO Status Register
00C0h+(4h*m)	MAILBOX_MSGSTATUS_m	Mailbox Message Status Register
0100h+(10h*u)	MAILBOX_IRQSTATUS_RAW_u	Mailbox IRQ RAW Register
0104h+(10h*u)	MAILBOX_ IRQSTATUS_CLR_u	Mailbox IRQ Clear Status Register
0108h+(10h*u)	MAILBOX_ IRQENABLE_SET_u	Mailbox IRQ Enable Set Register
010Ch+(10h*u)	MAILBOX_ IRQENABLE_CLR_u	Mailbox IRQ Enable Clear Register

基于 MAILBOX 的硬件构架，，目前有三种方法实现 ARM 和 DSP 通信：

（1），TI 的 LINUX SDK，此软件默认 ARM 使用 OPENCL 接口来调用 DSP。

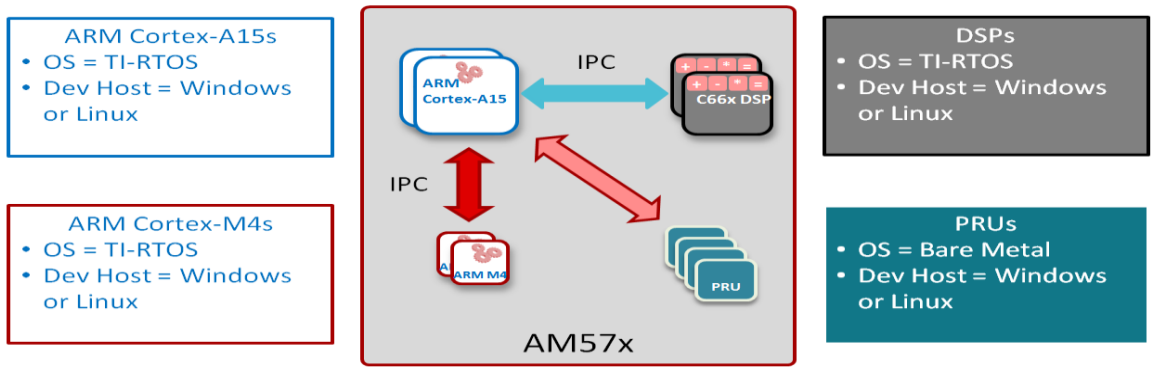
图 3 OPENCL 通信机制



OPENCL 把 DSP 当作一个加速器来使用的，DSP 跑的是 OPENCL 组件，因此开发者只需要了解 OPENCL 的 API 就可以对 DSP 进行编程了。不需要对 DSP 进行学习。以下链接是 OPENCL 的用户指导：
http://downloads.ti.com/mctools/esd/docs/opencl/index.html

（2），使用 TI 的 RTOS SDK，DSP 可以运行 SYSBIOS 系统，ARM 和 DSP 之间可以通过 IPC API 消息机制来进行通信。

图 4 IPC 通信机制



如果希望运行 IPC 通信机制，需要安装两个版本的 SDK，Processor SDK Linux 和 Processor SDK RTOS。LINUX 版本的 SDK 为 A15 提供 LINUX 支持，RTOS SDK 提供 DSP 运行软件。IPC 软件编译方法可以参考：

https://e2echina.ti.com/question_answer/dsp_arm/sitara_arm/f/25/t/139873

（3），本文实现的在 LINUX SDK 通过对 MAILBOX 寄存器直接操作的方式来实现 ARM 和 DSP 互相通信的机制。
DSP给ARM发消息为例，按照下面的步骤就可以实现ARM和DSP通过MAILBOX进行通讯：

- 选择空闲的 MAILBOX，假设选择 MAILBOX5（本例为了继承原有的中断事件号使用 IPC 预留的 MAILBOX5 U2 来触发 ARM 中断，MAILBOX13 来触发 DSP1 和 DSP2）；
- 对 MAILBOX_IRQENABLE_SET_U2 寄存器操作，使能 MAILBOX5 对应的 bit；
- 在 ARM 侧准备好中断处理函数，准备接收 MAILBOX 中断；
- 在 DSP 侧向 MAILBOX5 对应的 message 寄存器写个消息；
- ARM 被触发中断，在中断处理函数里读出消息。

三种方法小结：以上三种方法开发难度由难到易 3>2>1，软件的运行效率由高到低 3>2>1。

2 ARM 和 DSP 通信例程

2.1 使用方法

本例程软件包括以下几个部分：

1) ARM侧user space测试程序；
2) ARM侧硬件驱动；
3) DSP侧测试程序；
4) ARM侧Linux下加载DSP代码的程序。

对于不关心底层的读者可以仅阅读 1)和 3)部分的详解，参考 1)和 3)可以写出自己的代码。
解压缩例程软件包，可以看到四个目录：mailbox_test 是软件 1)；mailbox_driver 是软件 2）；AM57xx_DSP1DEMO 和 AM57xx_DSP2DEMO 目录下各有一个 CCS 的工程，是软件 3)；dsploader 是软件 4）。
这套 demo 的 ARM 侧软件是基于 LINUX 操作系统开发的，笔者用 ti-processor-sdk-linux-am57xx-evm-05.01.00.11 进行测试，可以正常工作。用户参考如下步骤可以顺利完成测试。

- 重新编译驱动。修改 mailbox_driver\Makefile 文件，把 KERNELDIR 改成自己的内核路径。KERNELDIR 需要指向一个编译过的内核源码树。
- 板子启动后，在 uboot 的控制台下把 bootargs 参数加上“mem=169M@0x80000000”，这样 Linux 内核只会管理从 0x80000000 开始的 169M 字节的内存。我们的 DEMO 用了 0x90000000~0xa0000000 地址共 256M 字节的内存来做测试。所以 Linux 只能使用 0x80000000~0x90000000 地址共 256M 的内存，这里选择 169M 只是示范，程序默认并没有做内存 COPY 测试。
- 把本例程的四个软件包编译好的文件放入测试板的 Linux 文件系统中。并且删除测试板 /lib/firmware 下相关文件。
- 系统启动后通过“insmod dsp_mailbox.ko”命令手动插入驱动，然后查看”/proc/devices”获取主设备号。然后通过命令”mknod /dev/dsp_mailbox c 239 0”创建设备节点。现有主设备号是 239，这是笔者的实验环境获取的主设备号，不同环境下是不一样的。
- 在 dsploader 目录运行”./dsploader AM57xx_DSP1DEMO.out dsp1” ”./dsploader AM57xx_DSP2DEMO.out dsp2”，加载 DSP 的程序，并启动 DSP。
- 在 mailbox_test 目录运行”./mailboxtest”。如果串口持续有大量打印“recv msg xxxx”说明程序成功运行起来了。

2.2 ARM LINUX测试例程分析

软件 1)包括一个文件 mailboxtest.c。它创建两个线程，一个发数据另一个收数据。这个程序打开 /dev/dsp_mailbox 设备节点。通过 ioctl 系统调用的 MAILBOX_RECVMSG 和 MAILBOX_SENDMSG 就可以实现接受和发送消息。当调用 MAILBOX_RECVMSG 时会阻塞等待。通常我们在 ARM 和 DSP 通信的时候会发一个指针过去，这个指针指向二者交换数据的物理地址。这个例程是向 DSP1 发送一个消息，DSP1 把消息转发给 DSP2，DSP2 在把消息返回 ARM。以下是软件 1)的接收和发送代码。

```
void send_message()
{
     if((fd_dsp = open("/dev/dsp_mailbox", O_RDWR | O_SYNC)) == -1) FATAL;//打开设备
     do{
          printf("send msg 0x%x\n",argv)
          ret = ioctl(fd_dsp,MAILBOX_SENDMSG, &argv)                    //发送消息
          if(ret == -1) FATAL;
              sem_wait(&sem_id);                //等待接受消息
       }while(1)
     close(fd_dsp);
}

void recv_message()
{
     if((fd_dsp = open("/dev/dsp_mailbox", O_RDWR | O_SYNC)) == -1) FATAL;//打开设备
     do{
          ret = ioctl(fd_dsp,MAILBOX_RECVMSG, &argv);        //接受消息
          if(ret == -1) FATAL;
          printf("recv msg 0x%x\n",argv);
          sem_post(&sem_id);                                 //释放信号
       }while(1);
     close(fd_dsp);
}
```

2.3 DSP 测试例程分析

软件 3)是一个不带操作系统的程序。main 函数做一些初始化后进入 while 循环，使用中断处理函数来处理相关事件。

main函数如下：

```
void main (void)
{
   IntDSPINTCInit();//初始化DSP中断控制器
   IntRegister(C674X_MASK_INT5, MailBoxIsr);//注册中断处理函数到中断向量表中
   IntEventMap(C674X_MASK_INT5, SYS_INT_MAILBOX);//让mailbox事件与中断向量表挂钩
   IntEnable(C674X_MASK_INT5);//使能中断
   IntGlobalEnable();//使能全局中断
   while(1);//程序循环
}
```

AM57XX 的 DSP 使用 C66x 的核， DSP 中断控制部分的操作方法和物理地址与其他 C66x 核是一样的。由于 ARM 的系统已经运行起来，PLL、DDR 等寄存器已经初始化完毕，这里只需对 DSP 私有的中断寄存器进行初始化就行了。本例中 MAILBOX 的中断服务程序是 MailBoxIsr()。

```
static void MailBoxIsr(void)
{
    unsigned int temp;
    IntDisable(C674X_MASK_INT5);//关中断
    IntEventClear(SYS_INT_MAILBOX);//清中断
    if(RD_MEM_32(MAILBOX_IRQSTATUS_CLR_U1)&(0x1<<20))//判断是否是自己的中断，并处理
    {
         while(RD_MEM_32(MAILBOX_MSGSTATUS))
         {
              temp = RD_MEM_32(MAILBOX_MESSAGE);//读消息
         }
         WR_MEM_32(MAILBOX_IRQSTATUS_CLR_U1,(0x1<<20));//清断
    }
    IntEnable(C674X_MASK_INT5);//开中断
    if(check_memory(temp))
    sendmsg(i++);
}
```

用户可以参考上面 1）和3）的代码分析，开发自己的代码。

2.4 ARM LINUX驱动分析

程序 2）是 ARM 侧运行的 LINUX DRIVER，它实现了 ARM 和 DSP 通信的底层。这个工程包含一个代码文件 dsp_mailbox.c，一个头文件 mailbox.h，还包含 Makefile 和驱动加载的脚本文件。驱动主要实现以下功能：

- 通过 MAILBOX 发送消息；
- 通过 MAILBOX 接受消息
- 启动 DSP。

驱动加载时会执行mailbox_init()函数，这个函数前半部分注册字符设备，函数接口等。后半部分是通过ioremap_nocache()的API进行物理地址到虚拟地址转换，驱动操作的都是转换后的虚拟地址。本文通过”cat /proc/interrupts”可知 ARM核中断号49对应MAILBOX5U2，本例通过

```
result = request_irq(MPU_INTERRUPT_NO, mbox_interrupt, IRQF_SHARED,"dsp_mailbox",
mailbox_devp);
```

申请中断，注册中断服务函数mbox_interrupt()。MPU_INTERRUPT_NO号中断是共享的，系统中还存在IPC驱动，它也会申请这个中断号，所以使用IRQF_SHARED作为参数。

卸载驱动的时候调用mailbox_exit()，对资源进行释放。

这个驱动的三个功能是通过ioctl()接口对外实现的。本例设计了三个命令字来实现三个系统调用参数。

```
#define MAILBOX_IOC_MAGIC  'k'
#define MAILBOX_DSP_START    _IOWR(MAILBOX_IOC_MAGIC, 0,int)
#define MAILBOX_SENDMSG      _IOWR(MAILBOX_IOC_MAGIC,1,int)
#define MAILBOX_RECVMSG      _IOWR(MAILBOX_IOC_MAGIC,2,int)
```

当用户程序使用MAILBOX_DSP_START参数来调用ioctl()时，会调到驱动代码ioctl里面的dsp_start()，这个函数会把DSP的启动地址写到DSPBOOTADDR寄存器内，然后启动DSP。DSP核有两个复位状态：

Module Reset，控制 DSPL2RAM 的访问。Module Reset 释放前，DSP 无法访问 L2RAM。

LocalReset，控制 DSP 的运行。LocalReset 释放后，DSP 会从指定的入口地址处运行代码。系统上电复位后，两个 RESET 都被 Assert，此时释放 ModuleReset，DSP 才能访问 L2RAM；释放 LocalReset 后，DSP 开始从 DSPBOOTADDR 处执行 DSP 代码。

以 AM57XX 芯片为例，复位状态可以通过 PRCM 模块->PRM_ACTIVE 域->RM_ACTIVE_RSTCTRL 进行控制。

当 DSP 核从 LocalReset 状态中释放出来，从 DSPBOOTADDR 地址处开始运行。DSP 核上电复位后，DSPBOOTADDR 默认值是 0x00800000(L2)。用户需要将 DSPBOOTADDR 配置成用户 DSP 程序的入口地址，要求为 1K 字节对齐，可以在 DSP 工程的 link.cmd 文件里进行设置。

对于DSPBOOTADDR的描述在：Control Module->DSPBOOTADDR Register。

当用户程序通过参数 MAILBOX_SENDMSG 调用 ioctl()时，底层驱动会通过写 MAILBOX 寄存器的方式触发 DSP 的中断，从而把消息发出去。

```
WR_MEM_32(MAILBOX_IRQENABLE_SET_U_DSPx,RD_MEM_32(MAILBOX_IRQENABLE_SET_U_DSPx) |
(0x1<<y));
//使能mailbox对DSP的中断。
WR_MEM_32(MAILBOX_MESSAGE,msg);
//写消息，触发DSP中断。
```

当用户程序通过参数MAILBOX_RECVMSG调用ioctl()时，会阻塞等待MAILBOX的消息。首先驱动通过wait_event_interruptible()函数阻塞等待在这个地方。当 ARM 的中断被触发的时候，程序进入中断处理函数 mbox_interrupt()，在此函数中通过 wake_up_interruptible()唤醒读消息的进程，并把收到的消息复制出去。这个驱动里面的中断处理函数和 DSP 侧的类似，先判断中断，然后读消息再清中断，唤醒阻塞进程，退出。

强调说明：这个驱动是使用的动态主设备号，加载后应该通过命令“cat /proc/devices”查看一下系统分配的主设备号是多少，然后使用命令” mknod /dev/dsp_mailbox c xxx 0”创建设备节点。

2.5 DSP 代码加载例程分析

程序4）可以在Linux下面加载DSP的可执行文件，并启动DSP。DSP运行的.out文件是ELF或者COFF格式的文件，关于这两种格式详细介绍可以参考：http://processors.wiki.ti.com/index.php/A_Brief_History_of_TI_Object_File_Formats
文件头包含有文件属性等信息，后面是若干sector，每个sector对应不同物理地址。本程序实现以下几个功能：

- 解析.out 文件，得到文件头以及各个 sector 的信息；
- 拷贝各个 sector 中的数据到各 sector 指定的物理内存中；
- 解析得到程序运行入口地址 (EntryPoint) ；
- 启动 DSP

DSP 可执行程序的地址信息对应 DSP 核的内存空间。但是 DSP 和 ARM 对于 L2RAM，OCMC SRAM 的内存空间映射不一样，而数据拷贝是发生在 ARM 侧的 LINUX 中，因此在 ARM 侧的程序中使用解析生成的地址时需要进行地址变换。

比如在AM57XX中，DSP1核L2RAM的地址空间为：0x00800000～0x0083FFFF，而ARM核访问 DSP L2RAM的地址空间为：0x40800000～0x4083FFFF，因此解析DSP 执行文件得到的sector地址，在ARM核使用时需要加上0x4000000的偏移量。

ARM LINUX 下操作的地址是虚拟地址，必须要把物理地址转换成虚拟地址才能对物理地址进行操作。通常做法是在驱动程序里通过 ioremap()来实现物理/虚拟地址转换。本例为了简化操作步骤，使用了系统的/dev/mem 设备，这个 LINUX 自带设备驱动可以方便我们访问物理地址。

打开/dev/mem节点后就可以通过mmap()这个接口来把物理地址映射成虚拟的地址。

```
        #include <sys/mman.h>
        void *mmap(void *start, size_t length, int prot, int flags,int fd, off_t
offset);
        int munmap(void *start, size_t length);
        start：映射区的开始地址，设置为0时表示由系统决定映射区的起始地址。
        length：映射区的长度
        prot：期望的内存保护标志
        fd：有效的文件描述符。一般是由open()函数返回，其值也可以设置为-
1，此时需要指定flags参数中的MAP_ANON,表明进行的是匿名映射。
        ffset：被映射对象内容的起点。
```

这个操作的地址和长度是按页对齐的。所以在代码里有很多地址对齐的算法。本例会先尝试通过elf的格式加载，如果失败了就会再试coff格式。

3 结束语

本文以AM57XX 为例分析新一代处理器平台 DSP 和 ARM 底层硬件通信原理，并实现了相关的软件例程。如果用户在 ARM 上使用 Linux 操作系统，可以直接使用本例程。在理解了底层原理的基础上可以很方便的把本例程移植到不同的操作系统下面。经过测试，少量数据经过 ARM->DSP1->DSP2->ARM 传输平均时间 17uS 左右，使用机制 IPC 软件做相同工作平均时间在 180uS 左右。此程序运行稳定，效率更高。

4 参考文献

1. *AM572x Sitara™ Processors DataSheet (SPRS953)*
2. AM572x Sitara™ Processors Technical Reference Manual (SPRUHZ6H)

重要声明和免责声明

TI 均以“原样”提供技术性及可靠性数据（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，不保证其中不含任何瑕疵，且不做任何明示或暗示的担保，包括但不限于对适销性、适合某特定用途或不侵犯任何第三方知识产权的暗示担保。

所述资源可供专业开发人员应用TI 产品进行设计使用。您将对以下行为独自承担全部责任：(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安全、安保或其他要求。所述资源如有变更，恕不另行通知。TI 对您使用所述资源的授权仅限于开发资源所涉及TI 产品的相关应用。除此之外不得复制或展示所述资源，也不提供其它TI或任何第三方的知识产权授权许可。如因使用所述资源而产生任何索赔、赔偿、成本、损失及债务等，TI对此概不负责，并且您须赔偿由此对TI 及其代表造成的损害。

TI 所提供产品均受TI 的销售条款 (http://www.ti.com.cn/zh-cn/legal/termsofsale.html) 以及ti.com.cn上或随附TI产品提供的其他可适用条款的约束。TI提供所述资源并不扩展或以其他方式更改TI 针对TI 产品所发布的可适用的担保范围或担保免责声明。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122