



Hi3093 混合部署方案

使用指南

文档版本 00B01

发布日期 2023-10-31

版权所有 © 海思技术有限公司2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

海思技术有限公司

地址：上海市青浦区虹桥港路2号101室 邮编：201721

网址：<https://www.hisilicon.com/cn/>

客户服务邮箱：support@hisilicon.com



前言

概述

本文档为使用海思Hi3093芯片的进行混合部署开发的程序员而写，目的是供您在开发过程中查阅混合部署时，所需环境以及编译步骤。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3093	-

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。



符号	说明
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修订说明
2023-10-31	00B01	第1次临时版本发布。



目录

前言.....	i
1 概述.....	1
1.1 概述.....	1
2 指南参考.....	2
2.1 混合部署非实时侧接口说明.....	2
2.2 混合部署实时侧接口说明.....	4
2.2.1 UniProton 混合部署接口说明.....	5
2.2.2 Baremetal 混合部署接口说明.....	5
2.2.3 Libck 混合部署接口说明.....	7
2.3 混合部署 IPC 通信 Buffer 大小配置.....	8
2.4 混合部署驱动隔离指南.....	14
2.5 混合部署下电指南.....	14
2.6 混合部署通信流程.....	15
2.7 混合部署注意事项.....	15



插图目录

图 2-1 单条 buffer 数据大小配置.....	8
图 2-2 脚本注释函数示意图.....	10
图 2-3 build_uniproton 函数注释示意图.....	10
图 2-4 UniProton 中 build_openamp.sh 脚本注释示意图.....	11
图 2-5 代码修改示意图.....	12
图 2-6 UniProton 侧代码修改示意图.....	12
图 2-7 Libck 侧代码修改示意图.....	13
图 2-8 Baremetal 侧代码修改示意图.....	13



表格目录

表 2-1 函数 openamp_init 描述表..... 2

表 2-2 函数 openamp_create_endpoint 描述表..... 2

表 2-3 函数 openamp_create_pty_endpoint 描述表..... 2

表 2-4 函数 openamp_destroy_endpoint 描述表..... 3

表 2-5 函数 openamp_destroy_pty_endpoint 描述表..... 3

表 2-6 函数 openamp_deinit 描述表..... 3

表 2-7 函数 get_zero_cpy_data_addr 描述表..... 4

表 2-8 函数 openamp_send_message 描述表..... 4

表 2-9 函数 openamp_endpoint_ready 描述表..... 4

表 2-10 函数 rpmsg_send_offchannel_nocopy_hwi 描述表..... 5

表 2-11 函数 rpmsg_get_tx_payload_buffer 描述表..... 5

表 2-12 函数 bm_mcs_create_ept 描述表..... 5

表 2-13 函数 bm_mcs_destroy_ept 描述表..... 6

表 2-14 函数 bm_mcs_get_zero_cpy_data_addr 描述表..... 6

表 2-15 函数 bm_mcs_send 描述表..... 6

表 2-16 函数 bm_mcs_shutdown 描述表..... 6

表 2-17 函数 sre_mcs_create_ept 描述表..... 7

表 2-18 函数 sre_mcs_destroy_ept 描述表..... 7

表 2-19 函数 sre_mcs_get_zero_cpy_data_addr 描述表..... 7

表 2-20 函数 sre_mcs_send 描述表..... 7

表 2-21 函数 sre_mcs_shutdown 描述表..... 8



1 概述

1.1 概述

该文档提供混合部署方案中驱动隔离方案、支持openAMP中buffer大小配置、混合部署非实时侧接口说明以及混合部署下电的使用指南等说明。

多系统混合部署框架请参考如下链接：<https://openeuler.gitee.io/yocto-meta-openeuler/openEuler-22.03-LTS-SP1/features/mcs.html>。



2 指南参考

2.1 混合部署非实时侧接口说明

表 2-1 函数 openamp_init 描述表

函数名	openamp_init
函数原型	int openamp_init(struct client_os_inst *client);
输入	client_os_inst：结构体。
输出	返回值为int类型，返回值小于0，代表初始化失败。
备注	该函数实现openamp初始化，包含从核处理器生命周期的初始化、把从核镜像加载到指定内存地址，对从核处理器进行上下电等操作。

表 2-2 函数 openamp_create_endpoint 描述表

函数名	openamp_create_endpoint
函数原型	int openamp_create_endpoint(const char *ep_name, rpmsg_ept_cb cb);
输入	ep_name：字符串类型，代表主从核通信通道的别名。 cb：代表ep_name通道的回调函数，处理从核发送过来的消息。
输出	返回值为int类型，返回值小于0，代表注册失败。
备注	该函数实现对主从核通道进行注册。

表 2-3 函数 openamp_create_pty_endpoint 描述表

函数名	openamp_create_pty_endpoint
-----	-----------------------------



函数原型	int openamp_create_pty_endpoint (const char *ep_name, rpmsg_ept_cb cb);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。 cb: 代表ep_name通道的回调函数, 处理从核发送过来的消息。
输出	返回值为int类型, 返回值小于0, 代表注册失败。
备注	该函数实现对主从核通道进行注册, 区别在该函数带有虚拟串口打印输入功能; 相同通道名字的通信endpoint只能够调用一次, 不能够重复创建相同名字的通道。

表 2-4 函数 openamp_destroy_endpoint 描述表

函数名	openamp_destroy_endpoint
函数原型	int openamp_destroy_endpoint(const char *ep_name);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。
输出	返回值int类型, 返回值等于0, 代表销毁成功。
备注	该函数实现对已经存在的通道进行销毁; 相同通道名字的通信endpoint只能够调用一次, 不能够重复创建相同名字的通道。

表 2-5 函数 openamp_destroy_pty_endpoint 描述表

函数名	openamp_destroy_pty_endpoint
函数原型	int openamp_destroy_pty_endpoint(struct pty_ep_data *pty_ep);
输入	pty_ep: pty_ep_data结构体类型, 代表销毁的通道的结构体。
输出	返回值int类型, 返回值等于0, 代表销毁成功。
备注	该函数实现对已经存在带有虚拟串口的通道进行销毁。

表 2-6 函数 openamp_deinit 描述表

函数名	openamp_deinit
函数原型	int openamp_deinit(struct client_os_inst *client);
输入	client_os_inst: 结构体。
输出	返回值int类型, 返回值等于-1, 代表销毁openamp生命周期失败。
备注	该函数实现对openamp进行销毁, 包含对从核生命周期以及申请的内存资源进行释放。



表 2-7 函数 get_zero_cpy_data_addr 描述表

函数名	get_zero_cpy_data_addr
函数原型	void* get_zero_cpy_data_addr(const char *ep_name, int *buffer_len);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。 buffer_len: 输入的buffer_len的值 (函数内部会赋值), 返回当前buffer可填充数据的最大值。
输出	void *: 返回可用buffer的首地址。 buffer_len: 返回当前buffer可填充数据的最大值。
备注	该函数返回可用buffer的首地址。

表 2-8 函数 openamp_send_message 描述表

函数名	openamp_send_message
函数原型	int openamp_send_message(const char *ep_name, void *data, int data_len);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。 data: 代表buffer的首地址。 data_len: 代表buffer中已经实际填充数据的长度。
输出	返回值为int类型, 小于0代表发送失败。
备注	调用该函数时, 由客户自行保证参数len与发送data数据的长度一致, 并且len值要小于等于buffer的长度减去消息头(16字节)。

表 2-9 函数 openamp_endpoint_ready 描述表

函数名	openamp_endpoint_ready
函数原型	int openamp_endpoint_ready(const char *ep_name);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。
输出	返回值为int类型, 等于1代表通道创建完成。
备注	该函数一直调用, 如果超时15us, 返回创建通道失败。

2.2 混合部署实时侧接口说明



2.2.1 UniProton 混合部署接口说明

表 2-10 函数 rpmsg_send_offchannel_nocopy_hwi 描述表

函数名	rpmsg_send_offchannel_nocopy_hwi
函数原型	rpmsg_send_offchannel_nocopy_hwi(struct rpmsg_endpoint *ept, const void *buffer_addr, int data_len)
输入	ept: rpmsg_endpoint: 代表通信通道的别名结构体。 buffer_addr: 代表待发送数据的首地址。 data_len: 代表buffer中已经填充数据的长度。
输出	返回值为int类型，小于0代表发送失败。
备注	-

表 2-11 函数 rpmsg_get_tx_payload_buffer 描述表

函数名	rpmsg_get_tx_payload_buffer
函数原型	void *rpmsg_get_tx_payload_buffer(struct rpmsg_endpoint *ept, uint32_t *len, int wait);
输入	ept: rpmsg_endpoint，代表通信通道的别名结构体。 len: 代表buffer返回的可写的最大值。 wait: 代表如果当前没有空闲buffer，是否等待空闲buffer；wait = 1代表等待。
输出	返回值为void *，返回buffer的首地址。
备注	-

2.2.2 Baremetal 混合部署接口说明

表 2-12 函数 bm_mcs_create_ept 描述表

函数名	bm_mcs_create_ept
函数原型	int bm_mcs_create_ept(const char *ep_name, rpmsg_ept_cb cb);
输入	ep_name: 字符串类型，代表主从核通信通道的别名。 cb: 代表ep_name通道的回调函数，处理从核发送过来的消息。
输出	返回值为int类型，返回值小于0，代表注册失败。
备注	该函数实现对主从核通道进行注册。



表 2-13 函数 bm_mcs_destroy_ept 描述表

函数名	bm_mcs_destroy_ept
函数原型	int bm_mcs_destroy_ept(const char *ep_name);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。
输出	返回值为int类型, 返回值小于0, 代表销毁失败。
备注	该函数实现对已经存在的通道进行销毁, 相同通道名字的通信endpoint 只能够创建和销毁一次。

表 2-14 函数 bm_mcs_get_zero_cpy_data_addr 描述表

函数名	bm_mcs_get_zero_cpy_data_addr
函数原型	void* bm_mcs_get_zero_cpy_data_addr(const char *ep_name, int *buffer_len);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。
输出	buffer_len: 返回当前buffer可填充数据的最大值。 void *: 返回可用buffer的首地址。
备注	该函数返回可用buffer的首地址。

表 2-15 函数 bm_mcs_send 描述表

函数名	bm_mcs_send
函数原型	int bm_mcs_send(const char* ep_name, const void *data, int len);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。 data: 代表buffer的首地址。 len: 代表buffer中已经填充数据的长度。
输出	返回值为int类型, 小于0代表发送失败。
备注	调用该函数时, 由客户自行保证参数len与发送data数据的长度一致, 并且len值要小于等于buffer的长度减去消息头(16字节)。

表 2-16 函数 bm_mcs_shutdown 描述表

函数名	bm_mcs_shutdown
函数原型	void bm_mcs_shutdown(void);
输入	void
输出	void



备注	该函数用于baremetal下电。
----	-------------------

2.2.3 Libck 混合部署接口说明

表 2-17 函数 sre_mcs_create_ept 描述表

函数名	sre_mcs_create_ept
函数原型	int sre_mcs_create_ept(const char *ep_name, rpmsg_ept_cb cb);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。 cb: 代表ep_name通道的回调函数, 处理从核发送过来的消息。
输出	返回值为int类型, 返回值小于0, 代表注册失败。
备注	该函数实现对主从核通道进行注册。

表 2-18 函数 sre_mcs_destroy_ept 描述表

函数名	sre_mcs_destroy_ept
函数原型	int sre_mcs_destroy_ept(const char *ep_name);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。
输出	返回值为int类型, 返回值小于0, 代表销毁失败。
备注	该函数实现对已经存在的通道进行销毁, 相同通道名字的通信endpoint只能够创建和销毁一次。

表 2-19 函数 sre_mcs_get_zero_cpy_data_addr 描述表

函数名	sre_mcs_get_zero_cpy_data_addr
函数原型	void* sre_mcs_get_zero_cpy_data_addr(const char *ep_name, int *buffer_len);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。
输出	buffer_len: 返回当前buffer可填充数据的最大值。 void *: 返回可用buffer的首地址。
备注	该函数返回可用buffer的首地址。

表 2-20 函数 sre_mcs_send 描述表

函数名	sre_mcs_send
-----	--------------



函数原型	int sre_mcs_send(const char* ep_name, const void *data, int len);
输入	ep_name: 字符串类型, 代表主从核通信通道的别名。 data: 代表buffer的首地址。 len: 代表buffer中已经填充数据的长度。
输出	返回值为int类型, 小于0代表发送失败。
备注	调用该函数时, 由客户自行保证参数len与发送data数据的长度一致, 并且len值要小于等于buffer的长度减去消息头(16字节)。

表 2-21 函数 sre_mcs_shutdown 描述表

函数名	sre_mcs_shutdown
函数原型	void sre_mcs_shutdown(void);
输入	void
输出	void
备注	该函数用于libck下电。

2.3 混合部署 IPC 通信 Buffer 大小配置

步骤如下:

步骤1 修改mpu_solution/open_source/open-amp/lib/include/openamp/rpmsg_virtio.h
中: 范围最大为1K, 修改地方如图2-1所示。

图 2-1 单条 buffer 数据大小配置

```
/*Configurable parameters*/
#ifndef RPMSG_BUFFER_SIZE
#define RPMSG_BUFFER_SIZE (512)
#endif
```

步骤2 在混合部署libck或者Baremetal下, 进行Buffer大小可配置时, 重新执行《openEuler+libck混合部署方案编译运行指南》中2.2 统一构建镜像章节内容; 或者执行《openEuler+Baremetal混合部署方案编译运行指南》中2.2 制作openEuler+Baremetal镜像章节内容。

步骤3 如果针对于UniProton的混合部署, 还需要进行如下修改。

完成步骤1后, 执行如下步骤。

1. 重新执行《openEuler+UniProton混合部署方案编译运行指南》中2.2 统一构建镜像章节内容。



2. 针对于UniProton中buffer大小可配置，修改地方:mpu_solution/src/real_time/UniProton/UniProton-v1.0.1/demos/hi3093/component/open-amp/lib/include/openamp/rpmsg_virtio.h中RPMSG_BUFFER_SIZE；修改完成后，执行如下命令。

```
cd mpu_solution/build/build_uniproton  
./build_uniproton.sh
```

须知

- mpu_solution/src/real_time/UniProton/UniProton-v1.0.1/demos/hi3093/component/open-amp/lib/include/openamp/rpmsg_virtio.h中RPMSG_BUFFER_SIZE大小要与mpu_solution/open_source/open-amp/lib/include/openamp/rpmsg_virtio.h中RPMSG_BUFFER_SIZE大小保持一致。
- 执行build_uniproton.sh之前，需要注释build_uniproton.sh脚本中如图2-2画红色方框的地方。需要注释build_uniproton.sh脚本中如图2-3画红色方框的地方。需要注释mpu_solution/src/real_time/UniProton/UniProton-v1.0.1/demos/hi3093/build中build_openamp.sh脚本中如图2-4。

图 2-2 脚本注释函数示意图

```
prepare_env  
build_uniproton  
./build_mcs.sh
```

图 2-3 build_uniproton 函数注释示意图

```
build_uniproton()  
{  
    echo "##### build UniProton #####"  
    cd ${UNIPROTON_PATH}  
    wget https://gitee.com/openeuler/UniProton/repository/archive/v1.0.1.zip --no-check-certificate  
    unzip v1.0.1.zip  
    rm -rf v1.0.1.zip  
  
    cd ${UNIPROTON_PATH}/UniProton-v1.0.1  
    patch -p1 < ${UNIPROTON_PATCH}/UniProton_update.patch  
    cd ${UNIPROTON_PATH}/UniProton-v1.0.1/demos/hi3093/build  
    sh ./build_app.sh  
  
    cd ${BUILD_PATH}  
    rm -rf output  
    mkdir -p output  
    cp ${UNIPROTON_PATH}/UniProton-v1.0.1/demos/hi3093/build/hi3093.bin output  
}
```

图 2-4 UniProton 中 build_openamp.sh 脚本注释示意图

```
#·echo·"#####git·clone·libmetal·#####"
#·pushd·../component
#·rm·-rf·./libmetal*
#·wget·https://github.com/OpenAMP/libmetal/archive/refs/tags/v2022.10.0.tar.gz·--no-check-certificate
#·tar·-zxvf·v2022.10.0.tar.gz
#·mv·./libmetal-2022.10.0·./libmetal
#·rm·v2022.10.0.tar.gz
#·cp·UniProton-patch-for-libmetal.patch·./libmetal
#·cd·libmetal
#·patch·-p1·-d·..·<·UniProton-patch-for-libmetal.patch
#·popd

#·echo·"#####git·clone·openamp·#####"
#·pushd·../component
#·rm·-rf·./open-amp*
#·wget·https://github.com/OpenAMP/open-amp/archive/refs/tags/v2022.10.1.tar.gz·--no-check-certificate
#·tar·-zxvf·v2022.10.1.tar.gz
#·mv·./open-amp-2022.10.1·./open-amp
#·rm·-rf·v2022.10.1.tar.gz
#·cp·UniProton-patch-for-openamp.patch·./open-amp
#·cd·open-amp
#·patch·-p1·-d·..·<·UniProton-patch-for-openamp.patch
#·cd·${BUILD_DIR}/UniProton-v1.0.1
#·patch·-p1·<·${BUILD_DIR}/../patches/uniproton/uniproton_patch_for_update.patch
#·cd·${BUILD_DIR}/UniProton-v1.0.1/demos/hi3093/component/open-amp
#·popd
```

3. 重新执行《openEuler+UniProton混合部署方案编译运行指南》中3.1混合部署升级指南 章节内容。

须知

- 生成组件位于build/build_uniproton/output目录下。
需要修改rpmsg_main与hi3093.bin执行权限，如果不修改，导致执行失败。
./rpmsg_main -c 3 -t hi3093.bin -a 0x93000000。
其他系统生成组件如libck和Baremetal请参考对应混合部署运行指南中混合部署升级指南小节。
libck参考《openEuler+libck混合部署方案编译运行指南》3.1 混合部署升级指南章节内容。
Baremetal参考《openEuler+Baremetal混合部署方案编译运行指南》3.1 混合部署升级指南章节内容。
- 如果需要验证buffer是否修改正确，需要修改如下地方。
图2-5显示mpu_solution/open_source/mcs/rpmsg_pty_uniproton/rpmsg_main.c中画红色方框：（其他系统如libck和baremetal同理修改mcs目录下对应的rpmsg_main.c）。

图 2-5 代码修改示意图

```
void*buffer_addr;
uint32_t buffer_len;
char*send_message_chars="aaaaaaaaaaa\n";
buffer_addr=get_zero_cpy_data_addr("console",&buffer_len);
if((int)buffer_len<strlen(send_message_chars)){
    printf("Send more data than the maximum buffer\n");
    return -1;
}

/*Being of write data for buffer*/
memcpy(buffer_addr,send_message_chars,strlen(send_message_chars));
int ret=openamp_send_message("console",(void*)buffer_addr,strlen(send_message_chars));
if(ret<0){
    printf("Failed to send data\n");
    return ret;
}
```

- 各个系统代码需要修改处如下。
 - UniProton侧修改如**图2-6**画红色方框所示。
mpu_solution/src/real_time/UniProton/UniProton-v1.0.1/demos/hi3093/apps/openamp/rpmsg_service.c

图 2-6 UniProton 侧代码修改示意图

```
int rpmsg_endpoint_cb(struct rpmsg_endpoint*ept,void*data,size_t len,uint32_t src,void*priv)
{
    g_openampFlag=true;
    g_rpmsg_dst_ept=ept;
    int data_len=16;
    write_message_send(data_len);
    return OS_OK;
}
```

- libck侧修改如**图2-7**画红色方框所示。
mpu_solution/src/samples/real_time/libck/mcs/sample_mcs.c

图 2-7 Libck 侧代码修改示意图

```
static volatile unsigned char g_mcs_rcv_times = 0;
static unsigned int g_received_msg;
static char *g_send_msg = "Hello, Libck!\r\n";

static int sample_mcs_cb(struct rpmsg_endpoint *ept, void *data, size_t len, unsigned int src, void *priv)
{
    ... (void)ept;
    ... (void)src;
    ... (void)priv;
    ... g_mcs_rcv_times++;

    ... SRE_Printf("data: %s\n", (char *)data);

    ... int buffer_len;
    ... void *buffer_addr = sre_mcs_get_zero_cpy_data_addr(ENDPOINT_DATA, &buffer_len);
    ... if (buffer_len < strlen(g_send_msg)) {
    ... | ... SRE_Printf("sample_mcs: sre_mcs_get_zero_cpy_data_addr error\n");
    ... | ... return OS_ERROR;
    ... }

    ... if (memcpy_s(buffer_addr, buffer_len, g_send_msg, strlen(g_send_msg)) != EOK) {
    ... | ... return OS_ERROR;
    ... }

    ... int ret = sre_mcs_send(ENDPOINT_DATA, buffer_addr, strlen(g_send_msg) * sizeof(char));
    ... if (ret < 0) {
    ... | ... SRE_Printf("sre_mcs_send console error, ret = %d\n", ret);
    ... | ... return OS_ERROR;
    ... }

    ... return OS_OK;
}
```

- Baremetal侧如图2-8画红色方框所示。
mpu_solution/src/samples/real_time/baremetal/mcs/sample_mcs.c

图 2-8 Baremetal 侧代码修改示意图

```
static volatile unsigned char g_mcs_rcv_times = 0;
static const char *g_send_msg = "Hello, Baremetal!\n";
#define SHUTDOWN_CMD "stop"

static int sample_mcs_cb(struct rpmsg_endpoint *ept, void *data, size_t len, unsigned int src, void *priv)
{
    ... (void)data;
    ... (void)len;
    ... (void)ept;
    ... (void)src;
    ... (void)priv;
    ... g_mcs_rcv_times++;

    ... bm_log("data: %s\n", (char *)data);

    ... int buffer_len;

    ... void *buffer_addr = bm_mcs_get_zero_cpy_data_addr(ENDPOINT_DATA, &buffer_len);
    ... if (buffer_len < (int)strlen(g_send_msg)) {
    ... | ... bm_log("sample_mcs: bm_mcs_create_ept error\n");
    ... | ... return BM_FAIL;
    ... }

    ... if (memcpy_s(buffer_addr, buffer_len, g_send_msg, strlen(g_send_msg)) != EOK) {
    ... | ... return BM_FAIL;
    ... }

    ... bm_mcs_send(ENDPOINT_DATA, buffer_addr, strlen(g_send_msg) * sizeof(char));
    ... return BM_OK;
}
```



----结束

2.4 混合部署驱动隔离指南

前置条件：

1. 系统安装python3。
2. 下载release代码包mpu_solution.tgz并解压。

隔离步骤如下：

步骤1 修改mpu_solution/build/build_rtos现有的*.dts文件。

📖 说明

*.dts仅支持hi3093_mcs_2with2.dts与hi3093_mcs_3with1.dts

步骤2 打开现有*.dts文件，增加或者修改驱动外设中属性（驱动外设支持参考特别说明中），属性支持如下说明；支持属性：domain、channel。

1. domain参数值说明：0x0：非实时侧使用 0x1：代表实时侧使用。
2. channel：参数值：<0x0 0x7>：代表通道0到通道7赋予实时侧使用。
3. 仅支持数值小于10的数字。

步骤3 修改完成后，进入mpu_solution/build/build_dts目录下。

如果修改hi3093_mcs_2with2.dts，则执行如下命令。

```
./convert_dts.sh oe_bm
```

如果修改hi3093_mcs_3with1.dts，则执行如下命令。

```
./convert_dts.sh oe_libck
```

----结束

须知

现有支持uart、gmac、timer、gpio、i2c、spi、watchdog、dmac、localbus，只有dmac支持channel。其他驱动仅支持domain。

2.5 混合部署下电指南

在混合部署方案中，执行混合部署升级指南操作后，需要进行等待10s，然后在执行rpmsg_main的串口中，执行ctrl+c操作，就可以正常下从核cpu的电。

须知

不要在执行rpmsg_main文件的串口之外，或者正在进行发送接收数据时，执行ctrl+c操作，进行从核下电。



2.6 混合部署通信流程

主核调用流程如下步骤所示。

步骤1 openamp_init

功能：实现openamp初始化，包含从核处理器生命周期的初始化、把从核镜像加载到指定内存地址，对从核处理器进行上下电等操作。

步骤2 openamp_create_endpoint或者openamp_create_pty_endpoint

功能：注册通道端点。

步骤3 openamp_endpoint_ready

功能：实现等待两侧端点创建完成。

步骤4 在回调函数endpoint_cb中调用get_zero_cpy_data_addr与openamp_send_message。

1. 调用get_zero_cpy_data_addr获取buffer的首地址；
2. 向[步骤1](#)中获取的buffer首地址填写数据；
3. 调用openamp_send_message发送[步骤2](#)中的数据。

步骤5 openamp_deinit

功能：该函数实现对openamp进行销毁，包含对从核生命周期以及申请的内存资源进行释放；同时调用openamp_destroy_endpoint销毁端点。

----结束

从核调用流程如下步骤所示(libck为sre_xxx，baremetal为bm_xxx)：以libck为例。

步骤1 sre_mcs_create_ept

功能：注册通道端点，传入通道名和接收回调cb函数。

步骤2 发送数据。

1. 调用sre_mcs_get_zero_cpy_data_addr获取buffer的首地址；
2. 向[步骤1](#)中获取的buffer首地址填写数据；
3. 调用sre_mcs_send发送[步骤2](#)中的数据。

步骤3 接收数据。

在接收回调cb函数内，接收数据。

----结束

2.7 混合部署注意事项

在混合部署框架中，注意如下事项：

- 下电过程中，出现No buffer is available for shutdown现象时，说明openAMP中共享buffer已经被使用完，无法发送消息下从核cpu的电；因此需要对单板重新上下电(硬重启)。



- 出现Send more data than the maximum buffer此现象时，说明openAMP中共享buffer已经被使用完，openEuler侧需要重新调get_zero_cpy_data_addr函数获取buffer首地址，直到出现返回首地址不为NULL，后续才可能正常发消息。
- 下电功能仅支持核间通信mcs。