



Tina Linux E907 开发指南

版本号: 1.3
发布日期: 2022.5.19

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.1.18	AWA1809	初始版本
1.1	2022.3.7	AWA1809	迭代版本
1.2	2022.4.11	AWA1809	添加大数据传输章节
1.3	2022.5.19	AWA1809	调整格式和内容

目 录

1 编写目的	1
2 使用范围	2
3 环境	3
4 SDK 快捷命令说明	4
5 E907 启动环境	5
5.1 预先工作	5
5.2 配置 boot0 启动 e907	5
5.2.1 关闭 RISCv 的 IOMMU	5
5.3 配置打包 e907 固件	6
5.4 Linux 配置	7
5.5 编译打包	8
6 AMP 环境搭建	9
6.1 Linux 配置	9
6.1.1 remoteproc 驱动	9
6.1.2 rpmsg 驱动	10
6.2 melis 配置	10
6.2.1 msgbox 配置	10
6.2.2 openamp 配置	11
6.3 打包	12
6.4 测试	12
6.4.1 E907 控制	12
6.4.2 rpmsg 通信测试	13
6.4.2.1 名字监听	13
6.4.2.2 节点创建	13
6.4.2.3 节点通信	15
6.4.2.4 节点关闭	16
7 开发使用	18
7.1 rpmsg 内核开发	18
7.2 rpmsg 用户层接口	18
7.3 amp 控制台	19
7.4 大数据传输	22
7.4.1 配置	22
7.4.2 测试	22
7.4.3 使用	24
7.4.4 Note	26
8 其他	27

8.1	rpmsg 需知	27
8.2	rpbuf 简介	27
8.3	修改 e907 地址	27
8.3.1	修改设备树 (Linux)	28
8.3.2	修改配置项 (melis)	28
8.3.3	修改链接脚本 (melis)	28
8.4	添加新板级注意事项	29
8.5	melis 系统	29
8.5.1	常用命令	29
8.5.2	自定义命令	30



插图

5-1 配置 1	5
5-2 关闭 IOMMU	6
5-3 打包配置	7
5-4 补丁下载	7
5-5 e907-standby 配置	8
6-1 rproc config	9
6-2 rpmsg config	10
6-3 msgbox-melis config	11
6-4 openamp config	11
6-5 rpmsg client config	12
6-6 rproc test	13
6-7 rproc test	13
6-8 rpmsg test	14
6-9 rpmsg test	14
6-10 rpmsg test	15
6-11 rpmsg test	15
6-12 rpmsg test	16
6-13 test	16
6-14 rpmsg test	16
6-15 rpmsg test	16
6-16 rpmsg test	17
6-17 rpmsg test	17
6-18 rpmsg test	17
6-19 rpmsg test	17
7-1 rpmsg config	20
7-2 amp_shell config	20
7-3 amp_shell config	20
7-4 amp_shell test	21
7-5 amp_shell test	21
7-6 Linux 端 log	23
7-7 e907 端 log	23
7-8 Linux 端 log	24
7-9 e907 端 log	24
8-1 e907 dram config	28
8-2 e907 lds config	29
8-3 新板级配置	29
8-4 添加自定义命令	30
8-5 执行自定义命令	30

1 编写目的

介绍 v85X 上 E907 的启动环境和 AMP 的环境搭建。

2 使用范围

全志 V85X 系列芯片

3 环境

A7 SDK: Tina

E907 SDK: melis

4 SDK 快捷命令说明

这里主要介绍几个下文会用到的命令，并不会介绍全部命令，如果想了解全部命令，可以在lunch方案后使用hmm打印出所有tina提供的快捷命令。

1. ckernel, m kernel_menuconfig, mkernel: 分别对应进入到内核目录，配置内核，单独编译内核
2. cboot0, mboot0: 进入 boot0 目录，单独编译 boot0
3. cmelis, mmelis, mmelis menuconfig: 分别对应进入 melis 根目录，编译 melis，配置 melis
4. make: 编译整个 tina 除了 melis 外的所有东西，如 boot0, uboot, 内核，跟文件系统等
5. cconfigs: 进入板级配置目录，这里主要存放板级的设备树，分区等配置文件
6. p: 打包命令，将编译后的东西打包成固件

5 E907 启动环境

5.1 预先工作

选择方案

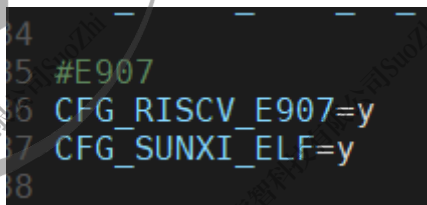
```
cd tina
source build/envsetup.sh
lunch
选择对应的V85x方案
```

5.2 配置 boot0 启动 e907

e907 在 boot0 阶段启动，需要对 boot0 进行一些配置

```
cboot0
vim board/sun8iw21p1/common.mk

# 如下图 取消注释
保存退出
mboot0 #编译
```



```
4 #E907
5 CFG_RISCV_E907=y
6 CFG_SUNXI_ELF=y
7
8
```

图 5-1: 配置 1

5.2.1 关闭 RISCv 的 IOMMU

本步骤只有需要在 boot0 阶段启动 E907 的需要配置。打开设备树，注释掉下面 2 条属性，因为 e907 在 boot0 阶段就启动了，不能打开其 IOMMU。

```
cconfigs
vim ../board.dts
```

```
49
50 >---e907_rproc: e907_rproc@0 {
51 >--->---compatible = "allwinner,sun8iw21p1-e907-rproc";
52 >--->---clock-frequency = <600000000>;
53 >--->---memory-region = <&e907_dram>, <&vdev0buffer>,
54 >--->--->--->--->--->--->---<&vdev0vring0>, <&vdev0vring1>;
55
56 >--->---mboxes = <&msgbox 0>;
57 >--->---mbox-names = "mbox-chan";
58 >--->---//iommu = <&mmu aw 5 1>;
59
60 >--->---memory-mappings =
61 >--->--->---/* DA >- len PA */
62 >--->--->---/* DDR for e907 */
63 >--->--->---< 0x48000000 0x00400000 0x48000000 >;
64 >--->---core-name = "sun8iw21p1-e907";
65 >--->---firmware-name = "melis-elf";
66 >--->---status = "okay";
67 >---};
68
69 >---rpbuff_controller0: rpbuff_controller@0 {
70 >--->---compatible = "allwinner,rpbuff-controller";
71 >--->---remoteproc = <&e907_rproc>;
72 >--->---ctrl_id = <0>;-/* index of /dev/rpbuff_ctrl */
73 >--->---//iommu = <&mmu aw 5 1>;
74 >--->---status = "okay";
75 >---};
76
```

图 5-2: 关闭 IOMMU

5.3 配置打包 e907 固件

```
cconfigs
cd ../../default/
vim boot_package_nor.cfg # 取消melis-elf选项的注释,如下图
vim boot_package.cfg # 取消melis-elf选项的注释,如下图
保存退出
```

```
1: boot_package.cfg
1 [package]
2 ;item=Item_TOC_name,      Item_filename,
3 ;item=scp,                scp.fex
4 item=optee,               optee.fex
5 item=u-boot,              u-boot.fex
6 item=dth,                 sunxi.fex
7 item=melis-elf,           riscv.fex
8 ;item=logo,               bootlogo.bmp.lzma
9 ;item=shutdowncharge,     bempty.bmp.lzma
10 ;item=androidcharge,     battery_charge.bmp.lzma
```

图 5-3: 打包配置

5.4 Linux 配置

```
ckernel
m kernel_menuconfig
# 如下图选中2个驱动
mkernel -j
```

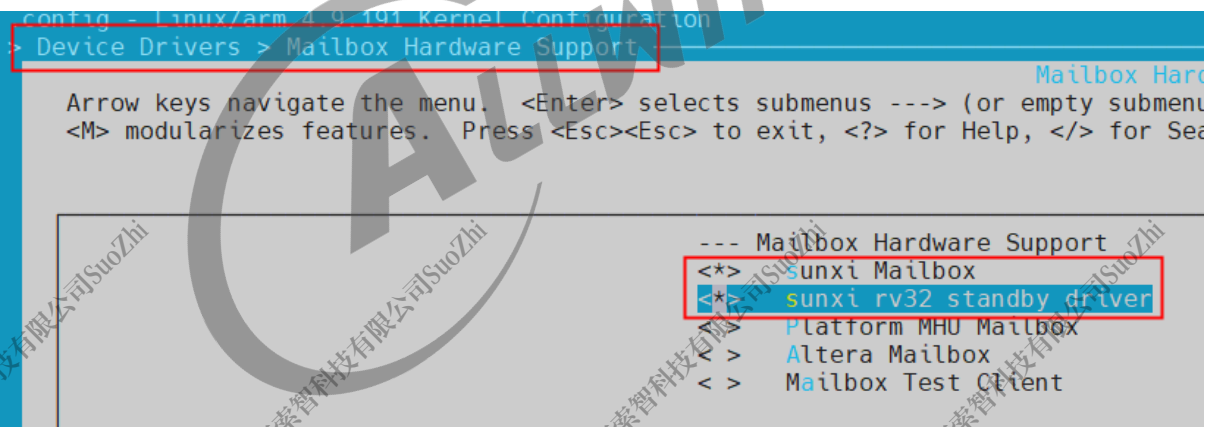


图 5-4: 补丁下载

```
mmelis menuconfig      # 如下图选中standby支持
```

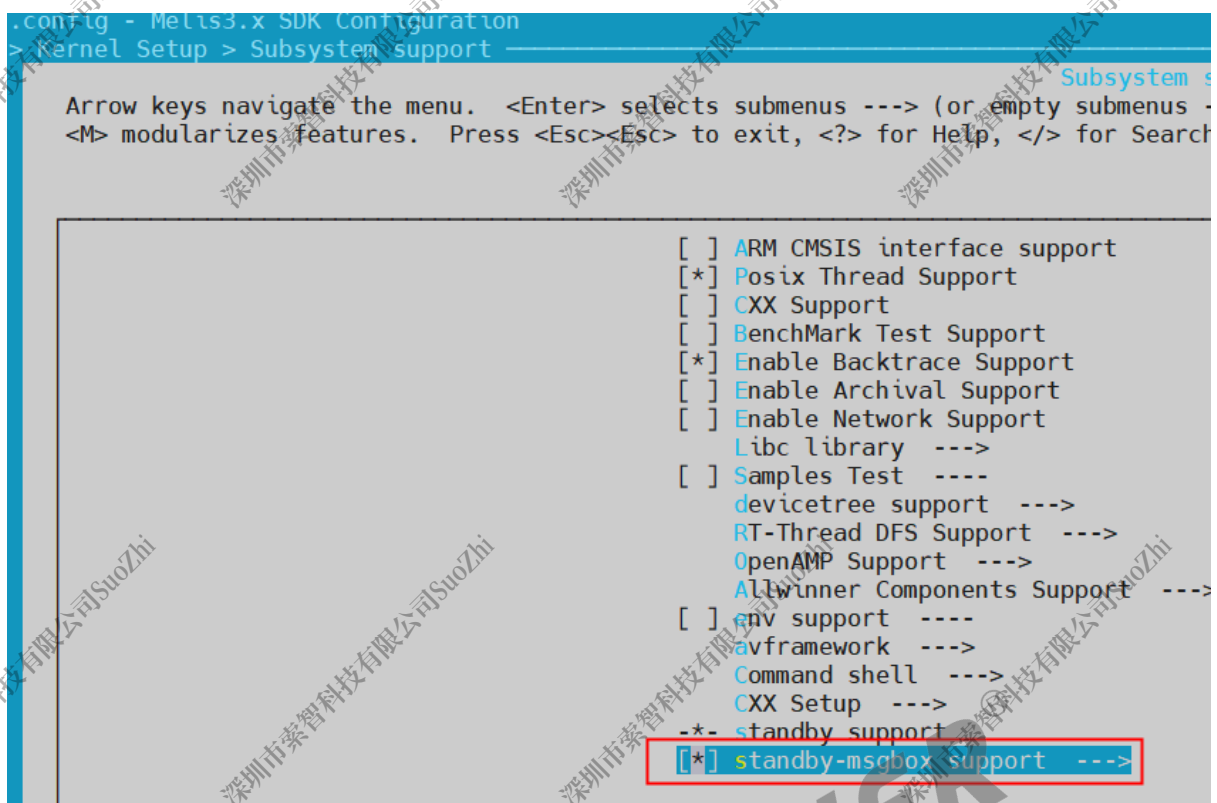


图 5-5: e907-standby 配置

5.5 编译打包

至此关于 E907 启动的配置完成，进行编译烧录即可

```
make -j16          # 编译tina
mmelis             # 编译melis
p                  # 烧录
```

6 AMP 环境搭建

AMP 环境用于 Linux 和 E907 间通信，Linux 依赖于 2 个驱动，melis 依赖于 openamp 驱动。

1. remoteproc 驱动：主要用来管理 E907 固件的加载器的
2. rpmsg：在 virtio 框架上实现的消息传送框架

6.1 Linux 配置

注意：需要前面的启动环境配置好后，再执行以下操作。

需要打开的配置有：

1. remoteproc 驱动
2. rpmsg 驱动

6.1.1 remoteproc 驱动

```
ckernel  
m kernel_menuconfig
```

选中

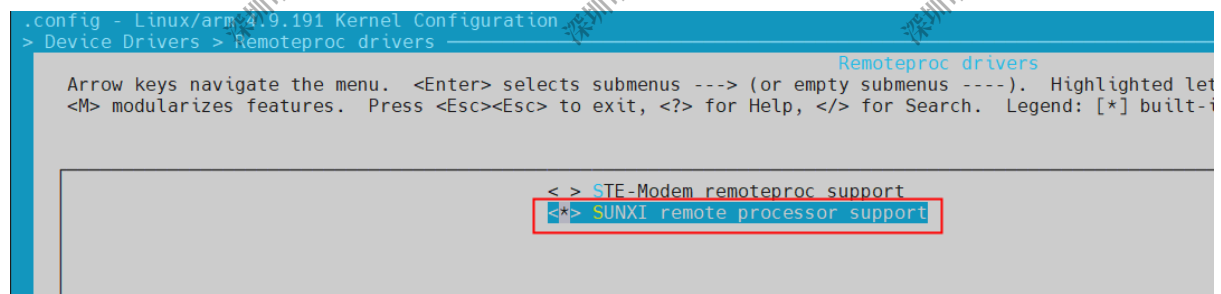


图 6-1: rproc config

6.1.2 rpmsg 驱动

```
ckernel
m kernel_menuconfig
# 红框必选,蓝色框为sdk提供的rpmsg demo,视情况而选择
# 建议选上 sunxi rpmsg ctrl driver 方便后面测试rpmsg通信功能
```

选中

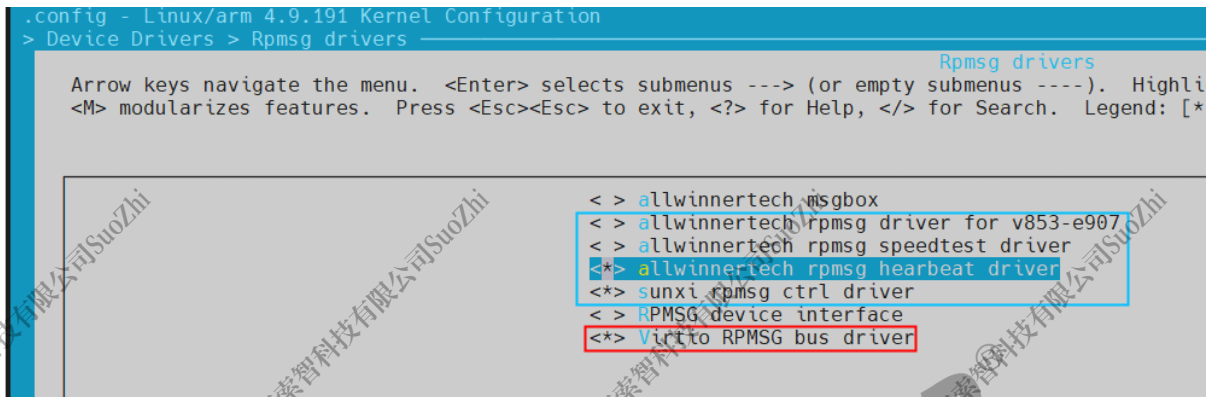


图 6-2: rpmsg config

6.2 melis 配置

主要进行 2 个配置：

1. msgbox 配置
2. openamp 配置

6.2.1 msgbox 配置

```
mmelis menuconfig #选择下面2项
```

选中

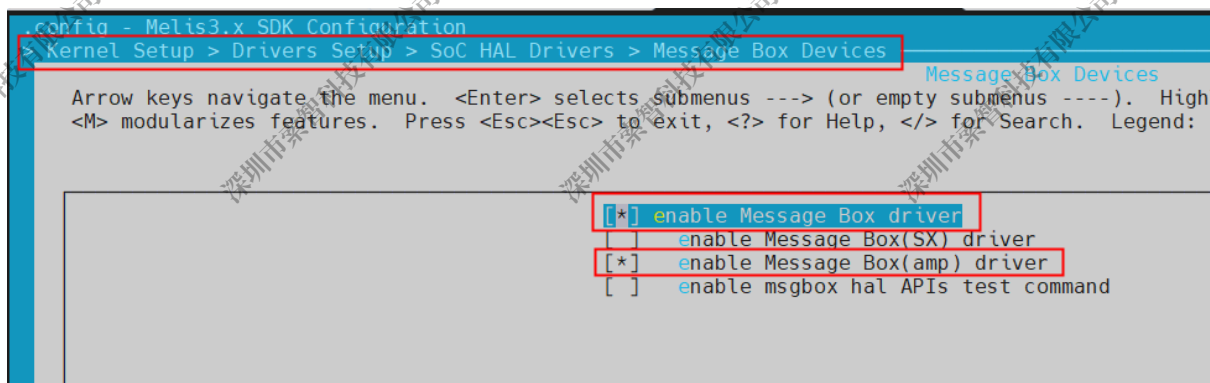


图 6-3: msgbox-melis config

6.2.2 openamp 配置

```

mmelis menuconfig
# 红框是必选, 蓝框是可选的 rpmsg demo

```

刚刚 Linux 端选择了 rpmsg heartbeat demo 和 ctrl driver, 我们这里也选上对应的驱动 heartbeat driver 和 client driver。

选中

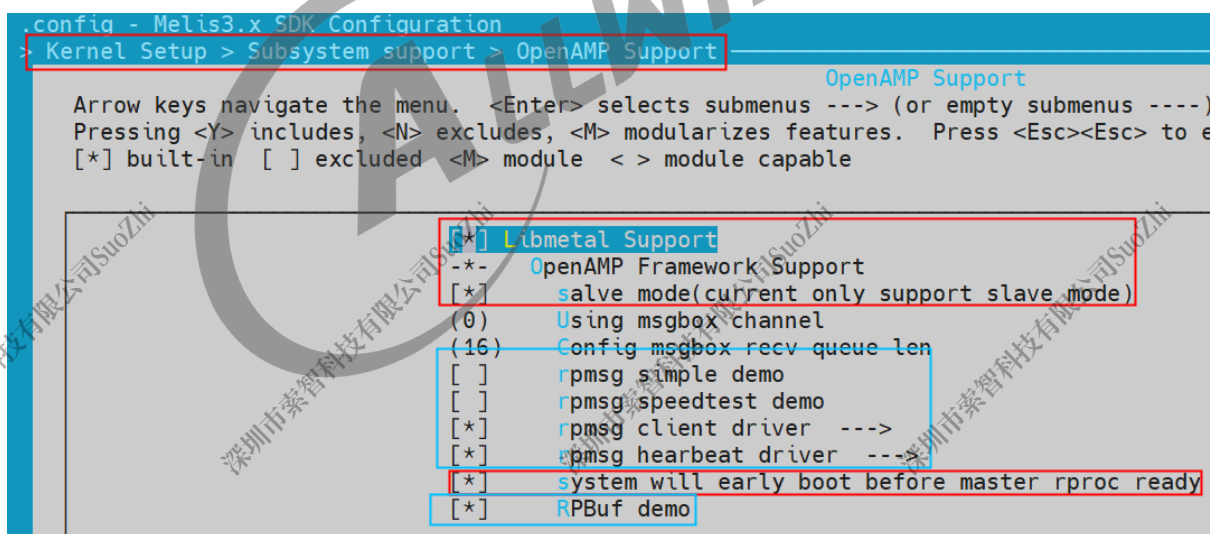


图 6-4: openamp config

为了方便在控制台测试 rpmsg 通信, rpmsg client driver 还需开启下面 2 个选项

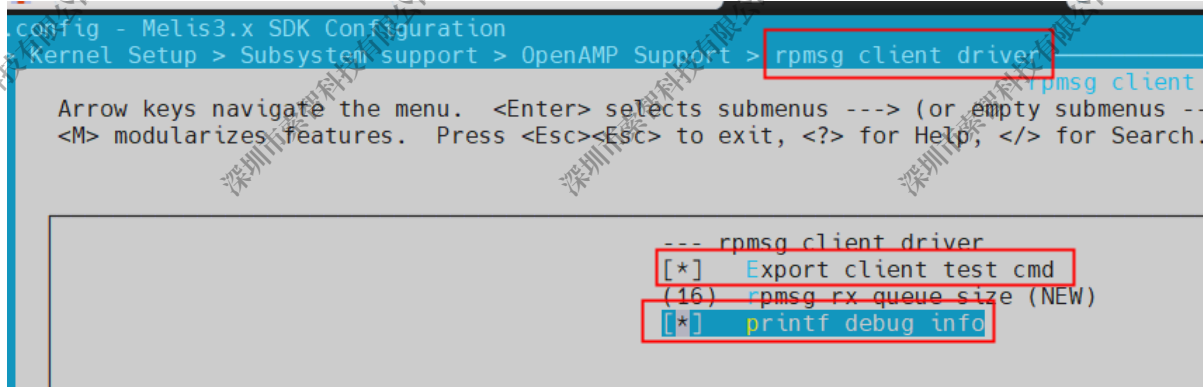


图 6-5: rpsmsg client config

6.3 打包

```
make -j16      # 编译tina
mmelis        # 编译melis
p
烧录
```

6.4 测试

本章节介绍一些 AMP 提供的控制台命令，用于测试 AMP 环境

6.4.1 E907 控制

1. 在 linux 控制台执行：echo stop > /sys/kernel/debug/remoteproc/remoteproc0/state
(停止 e907)
2. 在 linux 控制台执行：echo start > /sys/kernel/debug/remoteproc/remoteproc0/state
(启动 e907)

若控制台出现 remoteproc0: remote processor e907_rproc is now up, 表明启动 e907 成功。

如果使能了 rpsmsg_heartbeat 和 rpsmsg_ctrl 驱动，可以在 Linux 控制台 start 之后会看到如下输出：

```
root@TinaLinux:/#  
root@TinaLinux:/# echo start > /sys/kernel/debug/remoteproc/remoteproc0/state  
[ 139.195259] remoteproc0: powering up e907_rproc  
[ 139.205651] remoteproc0: Booting fw image melis-elf, size 210280  
[ 139.213208] virtio_rpmsg_bus virtio0: rpmsg host is online  
[ 139.219404] remoteproc0: registered virtio0 (type 7)  
[ 139.225216] remoteproc0: remote process already running  
[ 139.231190] remoteproc0: remote processor e907_rproc is now up  
[ 139.237945] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_ctrl addr 0x400  
[ 139.238568] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_heartbeat addr 0x401  
root@TinaLinux:/#  
root@TinaLinux:/#
```

图 6-6: rproc test

红框里面表示有 2 个设备成功创建，代表 rpmsg 正常。

6.4.2 rpmsg 通信测试

借助 rpmsg_ctrl 驱动帮助我们进行测试

6.4.2.1 名字监听

平台：melis 控制台

输入如下图命令：

eptdev_bind 命令：监听 name=test 的连接，最大连接数 5 个

```
msh >  
msh >eptdev_bind test 5  
msh >rpmsg_list_listen  
name          listen alive  
test          5 0  
msh >
```

图 6-7: rproc test

6.4.2.2 节点创建

平台：Linux 控制台

输入如下图的命令，进行节点创建

```
root@TinaLinux:/#  
root@TinaLinux:/# echo test > /sys/class/rpmsg/rpmsg_ctrl0/open  
[ 584.943213] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_client addr 0x402  
root@TinaLinux:/#  
root@TinaLinux:/# ls /dev/rpmsg*  
/dev/rpmsg0          /dev/rpmsg_ctrl0  
root@TinaLinux:/#  
root@TinaLinux:/# echo test > /sys/class/rpmsg/rpmsg_ctrl0/open  
[ 676.414406] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_client addr 0x403  
root@TinaLinux:/#  
root@TinaLinux:/# echo test > /sys/class/rpmsg/rpmsg_ctrl0/open  
[ 678.556031] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_client addr 0x404  
root@TinaLinux:/#  
root@TinaLinux:/# echo test > /sys/class/rpmsg/rpmsg_ctrl0/open  
[ 680.422041] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_client addr 0x405  
root@TinaLinux:/#  
root@TinaLinux:/# echo test > /sys/class/rpmsg/rpmsg_ctrl0/open  
[ 687.899242] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_client addr 0x406  
root@TinaLinux:/#  
root@TinaLinux:/# echo test > /sys/class/rpmsg/rpmsg_ctrl0/open  
[ 689.318495] rpmsg rpmsg_ctrl0: Remote is full  
sh: write error: Bad address  
root@TinaLinux:/# ls /dev/rpmsg*  
/dev/rpmsg0          /dev/rpmsg2          /dev/rpmsg4  
/dev/rpmsg1          /dev/rpmsg3          /dev/rpmsg_ctrl0  
root@TinaLinux:/#
```

图 6-8: rpmsg test

```
msh >  
msh >eptdev bind test 5  
msh >rpmsg_list_listen  
name          listen alive  
test          5 0  
msh >ctrldev: Rx 44 Bytes  
client: Rx 8 Bytes  
rpmsg0: binding  
send 0x13131411 to rpmsg0  
create rpmsg0 client success  
ctrldev: Rx 44 Bytes  
client: Rx 8 Bytes  
rpmsg1: binding  
send 0x13131411 to rpmsg1  
create rpmsg1 client success  
ctrldev: Rx 44 Bytes  
client: Rx 8 Bytes  
rpmsg2: binding  
send 0x13131411 to rpmsg2  
create rpmsg2 client success  
ctrldev: Rx 44 Bytes  
client: Rx 8 Bytes  
rpmsg3: binding  
send 0x13131411 to rpmsg3  
create rpmsg3 client success  
ctrldev: Rx 44 Bytes  
client: Rx 8 Bytes  
rpmsg4: binding  
send 0x13131411 to rpmsg4  
create rpmsg4 client success  
ctrldev: Rx 44 Bytes  
send 0x13131416 to rpmsg5  
rpmsg0: Rx 24 Bytes
```

图 6-9: rpmsg test

根据 log 可以看出，创建了一个 rpmsg0-4 5 个设备，因为 melis 只监听的 5 个，故最多只能创建 5 个。

6.4.2.3 节点通信

rpmsg 节点支持标准的文件操作，直接读写即可。

Linux 向 e907 发数据：

```
root@TinaLinux:/#  
root@TinaLinux:/# echo hello,rpmsg0.I am linux > /dev/rpmsg0  
root@TinaLinux:/# echo hello,rpmsg1.I am linux > /dev/rpmsg1  
root@TinaLinux:/# echo hello,rpmsg2.I am linux > /dev/rpmsg2  
root@TinaLinux:/# echo hello,rpmsg3.I am linux > /dev/rpmsg3  
root@TinaLinux:/# echo hello,rpmsg4.I am linux > /dev/rpmsg4  
root@TinaLinux:/#
```

图 6-10: rpmsg test

```
msh >  
msh >rpmsg0: Rx 24 Bytes  
Data:hello,rpmsg0.I am linux  
  
rpmsg1: Rx 24 Bytes  
Data:hello,rpmsg1.I am linux  
  
rpmsg2: Rx 24 Bytes  
Data:hello,rpmsg2.I am linux  
  
rpmsg3: Rx 24 Bytes  
Data:hello,rpmsg3.I am linux  
  
rpmsg4: Rx 24 Bytes  
Data:hello,rpmsg4.I am linux
```

图 6-11: rpmsg test

e907 向 Linux 发数据：

```
msh >rpmsg_list_epts
name          major          src - dst
rpmsg4        0              0x406 - 0x406
rpmsg3        0              0x405 - 0x405
rpmsg2        0              0x404 - 0x404
rpmsg1        0              0x403 - 0x403
rpmsg0        0              0x402 - 0x402
msh >
msh >eptdev_send 3 hello
will send hello to rpmsg3
msh >
```

图 6-12: rpmsg test

```
root@TinaLinux:/#
root@TinaLinux:/# cat /dev/rpmsg3
hello
```

图 6-13: test

6.4.2.4 节点关闭

Linux 主动释放：

```
^C
root@TinaLinux:/# ls /dev/rpmsg*
/dev/rpmsg0      /dev/rpmsg2      /dev/rpmsg4
/dev/rpmsg1      /dev/rpmsg3      /dev/rpmsg_ctrl0
root@TinaLinux:/# echo 2 > /sys/class/rpmsg/rpmsg_ctrl0/[ 1261.54
close
root@TinaLinux:/#
root@TinaLinux:/# ls /dev/rpmsg*
/dev/rpmsg0      /dev/rpmsg3      /dev/rpmsg_ctrl0
/dev/rpmsg1      /dev/rpmsg4
root@TinaLinux:/#
```

port MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

图 6-14: rpmsg test

```
rpmsg0          0
msh >
msh >eptdev_send 3 hello
will send hello to rpmsg3
msh >ctrldev: Rx 44 Bytes
send 0x13131411 to rpmsg2
rpmsg2: unbinding
msh >
```

图 6-15: rpmsg test

e907 主动释放：

```
msh >
msh >rpmsg_list_epts
name          major          src - dst
rpmsg4        0              0x406 - 0x406
rpmsg3        0              0x405 - 0x405
rpmsg1        0              0x403 - 0x403
rpmsg0        0              0x402 - 0x402
msh >eptdev_close 3
will close rpmsg3
rpmsg3: unbinding
msh >
```

图 6-16: rpmsg test

```
root@TinaLinux:/# ls /dev/rpmsg*
/dev/rpmsg0      /dev/rpmsg3      /dev/rpmsg_ctrl0
/dev/rpmsg1      /dev/rpmsg4
root@TinaLinux:/# [ 1432.766292] virtio_rpmsg_bus virtio0: destroying channel
root@TinaLinux:/# ls /dev/rpmsg*
/dev/rpmsg0      /dev/rpmsg1      /dev/rpmsg4      /dev/rpmsg_ctrl0
root@TinaLinux:/#
```

图 6-17: rpmsg test

e907 端接触监听，会释放所有的链接：

```
msh >
msh >eptdev_unbind test
unbind test
rpmsg4: unbinding
rpmsg1: unbinding
rpmsg0: unbinding
msh >
```

图 6-18: rpmsg test

```
root@TinaLinux:/#
root@TinaLinux:/# ls /dev/rpmsg*
/dev/rpmsg0      /dev/rpmsg1      /dev/rpmsg4      /dev/rpmsg_ctrl0
root@TinaLinux:/# [ 1539.044915] virtio_rpmsg_bus virtio0: destroying channel sunxi,rpmsg_client
[ 1539.054260] virtio_rpmsg_bus virtio0: destroying channel sunxi,rpmsg_client addr 0x403
[ 1539.063548] virtio_rpmsg_bus virtio0: destroying channel sunxi,rpmsg_client addr 0x402
root@TinaLinux:/# ls /dev/rpmsg*
/dev/rpmsg_ctrl0
root@TinaLinux:/#
```

图 6-19: rpmsg test

7 开发使用

7.1 rpmsg 内核开发

linux 端请参考 driver/rpmsg/rpmsg_client_e907.c。

melis 端请参考 ekernel/subsys/thirdparty/openamp/rpmsg_demo/ 目录下的文件。

7.2 rpmsg 用户层接口

控制台调试命令参考测试章节，这里列举代码使用示例。

Linux 端：

```
#include <linux/rpmsg.h>
# 创建端点
int fd;
struct rpmsg_ept_info info;
char ept_dev_name[32];

strcpy(info.name, "test");
info.id = 0xfffff;    # id由ioctl进行更新
fd = open(ctrl_dev, O_RDWR);
ret = ioctl(fd, RPMSG_CREATE_EPT_IOCTL, &info);
# 当ioctl返回值==0时，端点已经创建成功，设备节点会出现在/dev/rpmsg%d(=info.id)下
close(fd);

#读写设备节点
snprintf(ept_dev_name, 32, "/dev/rpmsg%d", info.id);
fd = open(ept_dev_name, O_RDWR);
write,read,poll...
close(fd);

# 关闭节点
fd = open(ctrl_dev, O_RDWR);
ret = ioctl(fd, RPMSG_DESTROY_EPT_IOCTL, &info);
close(fd);
```

melis 端：

方法 1：基于 rpmsg_ctrl 驱动，等待主机建立连接

```
// 头文件
#include <openamp/sunxi_helper/openamp.h>

static int ept_cb(struct rpmsg_endpoint *ept, void *data,
```

```
size_t len, uint32_t src, void *priv)
{
    // 收到数据
}

int bind_cb(struct rpmsg_ept_client *client)
{
    // client绑定,每个client代表一个连接
    // client->priv和client->ept->priv 可供用户使用
}

int unbind_cb(struct rpmsg_ept_client *client)
{
    // 连接关闭
}

int main()
{
    // cnt: 监听的数量,即最多对test创建cnt个连接
    // 最后一个参数 priv,其实里面设置的是client->priv
    rpmsg_client_bind("test", ept_cb, bind_cb, unbind_cb,
                     cnt, NULL);
    // do some things
    // 取消绑定会unbind所有与其相关的client
    rpmsg_client_unbind("test");
}
```

具体代码参考 `ekernel/subsys/thirdparty/openamp/rpmsg_demo/rpmsg_ctrl/test.c`;

方法 2: 基于 `rpmsg` 原生框架, `melis` 端主动创建连接, 触发主机端的 `rpmsg driver` 的 `probe`。

`melis` 端代码参考:

1. `ekernel/subsys/thirdparty/openamp/rpmsg_demo/demo.c`
2. `ekernel/subsys/thirdparty/openamp/rpmsg_demo/heartbeat.c`

Linux 端参考代码:

1. `drivers/rpmsg/rpmsg_client_e907.c`
2. `drivers/rpmsg/rpmsg_client_heart.c`

7.3 amp 控制台

SDK 在 Linux 端提供了进入 E907 控制台的功能, 配置步骤如下:

内核配置

```
ckernel
m kernel_menuconfig # 选择下图配置
```

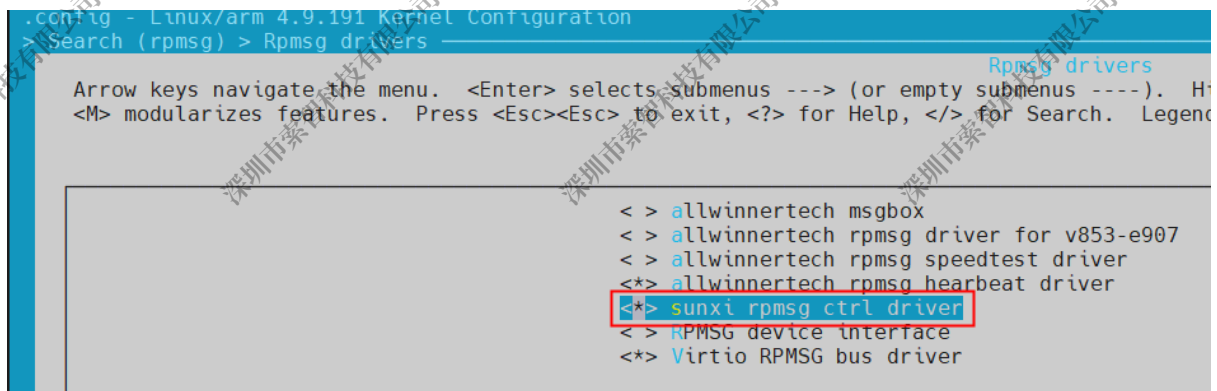



图 7-1: rpmmsg config

Tina 配置

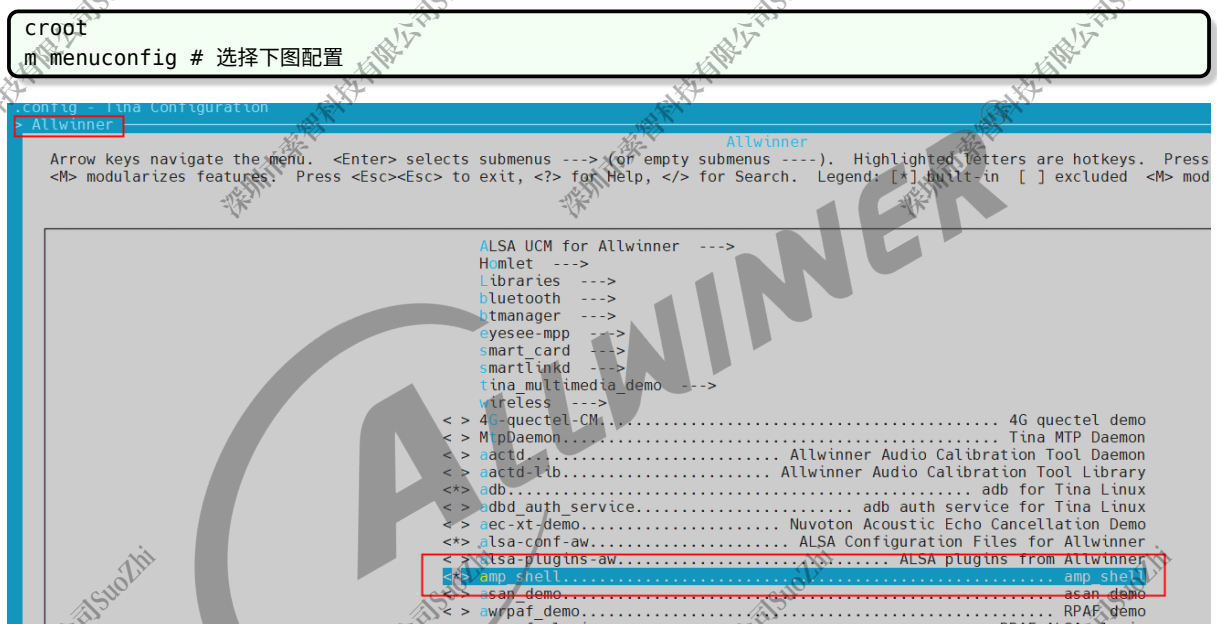


图 7-2: amp_shell config

melis 配置

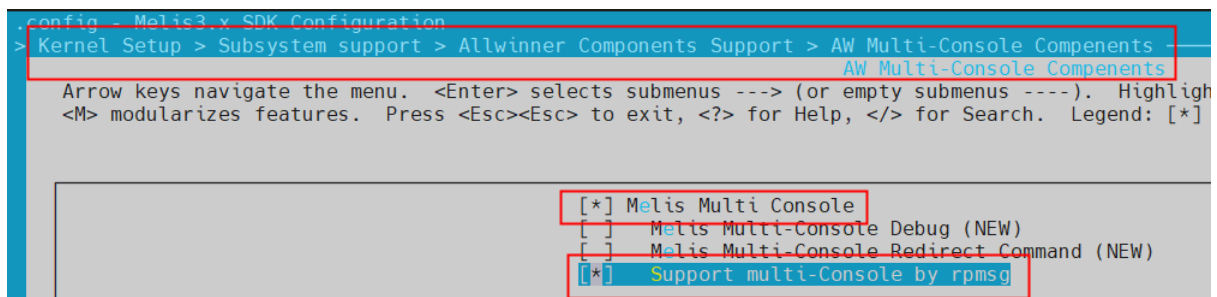


图 7-3: amp_shell config

编译 & 打包 & 下载

```
mmel1s -j32
make -j32
p
```

使用

在 `echo start > /sys/kernel/debug/remoteproc/remoteproc0/state` 后检查有无 `rpmsg_ctrl` 成功创建的 `log` 或者是否存在 `/dev/rpmsg_ctrl0` 节点。

如果正常，直接在 Linux 控制台啊输入 `amp_shell` 即可进入 `e907` 控制台，`amp_exit`退出控制台。

支持执行多次 `amp_shell`，开启多个控制台。

```
root@TinaLinux:/# echo start > /sys/kernel/debug/remoteproc/remoteproc0/state;
[ 56.656491] remoteproc0: powering up e907_rproc
[ 56.667000] remoteproc0: Booting fw image melis-elf, size 215912
[ 56.674564] virtio_rpmsg_bus virtio0: rpmsg host is online
[ 56.680750] remoteproc0: registered virtio0 (type 7)
[ 56.686553] remoteproc0: remote process already running
[ 56.692521] remoteproc0: remote processor e907_rproc is now up
[ 56.699267] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_ctrl addr 0x400
[ 56.699894] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_heartbeat addr 0x401
root@TinaLinux:/#
root@TinaLinux:/# ls /dev/rpmsg_ctrl0
/dev/rpmsg_ctrl0
root@TinaLinux:/#
```

图 7-4: amp_shell test

```
root@TinaLinux:/#
root@TinaLinux:/# ls /dev/rpmsg_ctrl0
/dev/rpmsg_ctrl0
root@TinaLinux:/# amp_shell
[ 168.531932] virtio_rpmsg_bus virtio0: creating channel sunxi,rpmsg_client addr 0x402

Success Create /dev/rpmsg0
=====
AMP Shell
=====
msh >msh >
msh >ps
msh >ps
thread                pri  status      sp      stack size max used left tick  error
-----
rpmsg0-console        25  ready      0x000001e8 0x00004000    19%  0x00000004 000
tshell                21  suspend    0x000001c8 0x00004000    14%  0x0000000a 000
ctrldev               6   suspend    0x00000178 0x00001000    61%  0x0000000a 000
vring-ipi            15  suspend    0x00000138 0x00002000    31%  0x0000000a 000
tidle                 31  ready      0x000001c8 0x00002000    05%  0x0000001e 000
timer                 8   suspend    0x000000e8 0x00004000    02%  0x00000001 000
msh >
msh >amp_exit
[ 213.181558] rpmsg rpmsg_ctrl0: close /dev/rpmsg0 endpoint
[ 213.190341] virtio_rpmsg_bus virtio0: destroying channel sunxi,rpmsg_client addr 0x402
root@TinaLinux:/#
```

图 7-5: amp_shell test

7.4 大数据传输

由于 rpmsg 特性，不适合传输大数据量；如需使用大数据传输，请参考本章节。

7.4.1 配置

内核打开 rpbuf 驱动：

```
m kernel_menuconfig
Device Drivers --->
  RPBuf drivers --->
    [*] RPBuf device interface
    <*> RPMsg-based RPBuf service driver
    <*> Allwinner RPBuf controller driver
    <*> Allwinner RPBuf sample driver
```

Note: Allwinner RPBuf sample driver 是一个简单的 rpbuf 内核层使用 demo，可以不使能。

e907 配置：

```
Kernel Setup
Subsystem support
  Allwinner Components Support
    RPBuf framework
      [*] RPMsg-based RPBuf service component
      [*] RPBuf controller component
      [*] RPMsg-based RPBuf service component demo
  OpenAMP Support
    [*] RPBuf demo
```

Tina 打开 rpbuf_demo 软件包：

```
m menuconfig
Allwinner --->
  RPBuf --->
    <*> rpbuf_demo
    <*> rpbuf_test
```

7.4.2 测试

rpmsg_test：会自动生成随机数据并附带 MD5 校验值，另一端收到会重新计算 MD5 并与收到的进行对比。

```
(e907) rpbuf_test -c -N "rpbuf_demo" -L 0x100000 # 创建size=0x100000的buffer
(Linux) rpbuf_test -d 1000 -s -L 0x100000 -N "rpbuf_demo" # 发送测试数据

(Linux) rpbuf_test -r -t 1000 -L 0x100000 -N "rpbuf_demo" # 接收数据
(e907) rpbuf_test -s -L 0x100000 -N "rpbuf_demo" #发送测试数据
(e907) rpbuf_test -N "rpbuf_demo" -d # 删除buffer
```

出现 success 表明校验成功。

过程 log 如下：

```

root@TinaLinux:/#
root@TinaLinux:/#
root@TinaLinux:/# rpbuff_test -d 1000 -s -L 0x100000 -N "rpbuff_demo"
data:59b0d02cdd8a1a0fd62654535d6f8c49... [md5:d8849ed8928ff0f8778ffe899ee36ea6]
data:70aaf9582e574a43915bd658cf0c3347... [md5:95ef3c649604e56cb6fa1e5bf7848cdb]
data:86a422057f237a774d90585e41aad944... [md5:851609c0b66b87a3db82e69b6a50b6bd]
data:9d9e4b31d0efa92b08c5da63b3478042... [md5:3c4e146e2bc2e644c84325fb5e13704d]
data:b498745d21bcd95fc4f95c6925e52640... [md5:69fe35e6a21f79b14fcdcbcaa5c4242fa]
data:ca929d09728809147f2edf6e9682cd3d... [md5:bfb1554d336d333a97e83adac1f16c29]
data:f786ef611421697cf697e3797abd1a39... [md5:28b06f74df164d4ca06ed71146b9dc8a]
data:0e81180e65ed9830b2cc657fec5ac136... [md5:f0a4140223e6d5ef7d100ae6fd461b45]
data:257b413ab6b9c8646d01e8045df86734... [md5:b686ca33c186f9b18e8ae63bdf726dd3]
data:3b756a660786f81829366a0acf950e32... [md5:2bb825458d047080010c47c049d9eca2]

root@TinaLinux:/#
root@TinaLinux:/#
root@TinaLinux:/# rpbuff_test -r -L 0x100000 -N "rpbuff_demo" # 接收数据
^C
root@TinaLinux:/# ^C
root@TinaLinux:/# rpbuff_test -r -L 0x100000 -N "rpbuff_demo"
data:3707882b993a6f68c4fa2c74b9bea737... check:fd4447d90b5b0be166695ed0a35b271b success
data:bfe37d347f048e2129373a15646f8f29... check:a5a25a5e6da7dbde21a89095c33e4990 success

```

发送数据

发送的数据及校验值

接收数据

表明校验通过

图 7-6: Linux 端 log

```

msh >
msh > rpbuff_test -c -N "rpbuff_demo" -L 0x100000
msh > buffer "rpbuff_demo" is available
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:59b0d02cdd8a1a0fd62654535d6f8c49... check:d8849ed8928ff0f8778ffe899ee36ea6 success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:70aaf9582e574a43915bd658cf0c3347... check:95ef3c649604e56cb6fa1e5bf7848cdb success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:86a422057f237a774d90585e41aad944... check:851609c0b66b87a3db82e69b6a50b6bd success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:9d9e4b31d0efa92b08c5da63b3478042... check:3c4e146e2bc2e644c84325fb5e13704d success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:b498745d21bcd95fc4f95c6925e52640... check:69fe35e6a21f79b14fcdcbcaa5c4242fa success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:ca929d09728809147f2edf6e9682cd3d... check:bfb1554d336d333a97e83adac1f16c29 success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:f786ef611421697cf697e3797abd1a39... check:28b06f74df164d4ca06ed71146b9dc8a success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:0e81180e65ed9830b2cc657fec5ac136... check:f0a4140223e6d5ef7d100ae6fd461b45 success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:257b413ab6b9c8646d01e8045df86734... check:b686ca33c186f9b18e8ae63bdf726dd3 success
buffer "rpbuff_demo" received data (addr: 0x48c00000, offset: 0, len: 1048576):
data:3b756a660786f81829366a0acf950e32... check:2bb825458d047080010c47c049d9eca2 success
buffer "rpbuff_demo": remote buffer destroyed

msh > buffer "rpbuff_demo" is available
buffer "rpbuff_demo": remote buffer destroyed
buffer "rpbuff_demo" is available

msh > rpbuff_test -s -L 0x100000 -N "rpbuff_demo"
data:3707882b993a6f68c4fa2c74b9bea737... [md5:fd4447d90b5b0be166695ed0a35b271b]

msh > rpbuff_test -s -L 0x100000 -N "rpbuff_demo"
data:bfe37d347f048e2129373a15646f8f29... [md5:a5a25a5e6da7dbde21a89095c33e4990]

msh > buffer "rpbuff_demo": remote buffer destroyed

```

创建buffer

收到数据并校验成功

发送数据

图 7-7: e907 端 log

rpbuf_demo：用于在控制台简单传输数据

```
(e907) rpbuf_demo -c -N "rpbuf_demo" -L 0x1000 # 创建size=4k的buffer
(Linux) rpbuf_demo -d 1000 -L 0x1000 -N "rpbuf_demo" -s "hello" # 发送数据,并在1000ms后释放
buffer
(Linux) rpbuf_demo -r -t 1000 -L 0x1000 -N "rpbuf_demo" # 接收数据
(e907) rpbuf_demo -s "hello" -N "rpbuf_demo" # 发送数据
(e907) rpbuf_demo -N "rpbuf_demo" -d # 删除buffer
```

```
root@TinaLinux:/#
root@TinaLinux:/# rpbuf_demo -d 1000 -L 0x1000 -N "rpbuf_demo" -s "hello"
root@TinaLinux:/#
root@TinaLinux:/# rpbuf_demo -r -L 0x1000 -N "rpbuf_demo"
Data received (offset: 0, len: 5):
 0x68 0x65 0x6c 0x6c 0x6f
^C
root@TinaLinux:/#
root@TinaLinux:/#
```

图 7-8: Linux 端 log

```
msh >rpbuf_demo -c -N "rpbuf_demo" -L 0x1000
msh >
msh >buffer "rpbuf_demo" is available
buffer "rpbuf_demo" received data (addr: 0x48b0c000, offset: 0, len: 5):
 0x68, 0x65, 0x6c, 0x6c, 0x6f,
buffer "rpbuf_demo": remote buffer destroyed

msh >buffer "rpbuf_demo" is available

msh >rpbuf_demo -s "hello" -N "rpbuf_demo"
msh >
msh >buffer "rpbuf_demo": remote buffer destroyed

msh >rpbuf_demo -N "rpbuf_demo" -d
```

图 7-9: e907 端 log

7.4.3 使用

内核层接口，参考 drivers/rpbuf/rpbuf_sample_sunxi.c：

Linux 端使用流程：

1. 在需要用到 rpbuf 接口的驱动的设备树节点种添加一条属性：rpbuf = <&rpbuf_controller0>;，可以创建多个 controller，当面默认只有一个rpbuf_controller0；
2. 获取 controller：调用 controller = rpbuf_get_controller_by_of_node(np, 0)；
3. 创建 buffer：调用 rpbuf_alloc_buffer(controller, name, len, ops, cbs, priv)；
4. 接收数据：收到数据时候会调用 cbd->rx_cb 回调
5. 判断状态：创建出的 buffer 不一样马上可用，需要用判断状态，调用 rpbuf_buffer_is_available (buffer)

6. 发送数据：

7. `buf_va = rpbuf_buffer_va(buffer);`
8. `buf_len = rpbuf_buffer_len(buffer);`
9. 直接对 `buf_va` 地址进行写入即可
10. `rpbuf_transmit_buffer(buffer, offset, data_len);`
11. 释放 `buffer`：`rpbuf_free_buffer(buffer);`

应用层端口，具体细节可以参考 `package/allwinner/rpbuf/`

1. 创建 `buffer`

2. `fd = open(0, O_RDWR);`
3. `ioctl(fd, RPBUF_CTRL_DEV_IOCTL_CREATE_BUF, &buffer->info);`
4. `buf_fd = open(buf_dev_path, O_RDWR);`
5. `addr = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, buf_fd, 0);`
6. 接收数据：
7. `rpbuf_receive_buffer_block(buffer, &offset, &data_len)` 阻塞接收
8. `rpbuf_receive_buffer_nonblock(buffer, &offset, &data_len)` 非阻塞接收
9. 发送数据：
10. `buf_va = rpbuf_buffer_va(buffer);`
11. 直接对 `buf_va` 地址进行写入即可
12. `rpbuf_transmit_buffer(buffer, offset, data_len);`
13. 释放 `buffer`：
14. `close(buf_fd);`
15. `ioctl(fd, RPBUF_CTRL_DEV_IOCTL_DESTROY_BUF, &buffer->info);`
16. `close(fd);`

e907 端接口，可以参考 `ekernel/subsys/aw/rpbuf/rpbuf_demo/rpbuf_demo.c`

e907 端使用流程：

1. 获取 `controller`：调用 `controller = rpbuf_get_controller_by_id(0);`；代码默认提供一个 `controller`，这里直接使用
2. 创建 `buffer`：调用 `rpbuf_alloc_buffer(controller, name, len, ops, cbs, priv);`
3. 接收数据：收到数据时候会调用 `cbd->rx_cb` 回调
4. 判断状态：创建出的 `buffer` 不一样马上可用，需要用判断状态，调用 `rpbuf_buffer_is_available(buffer)`
5. 发送数据：
6. `buf_va = rpbuf_buffer_va(buffer);`
7. `buf_len = rpbuf_buffer_len(buffer);`
8. 直接对 `buf_va` 地址进行写入即可
9. `rpbuf_transmit_buffer(buffer, offset, data_len);`
10. 释放 `buffer`：`rpbuf_free_buffer(buffer);`

7.4.4 Note

1. 关于 controller：代码默认已经提供了一个基于 rpmsg 实现的 controller0 了，正常情况下直接使用改 controller 即可
2. 关于 rdbuf_alloc_buffer 的 ops 参数：controller0 已经基于 ion 实现了内存分配函数。如无必要，使用 controller 的内存分配函数即可，即创建 buffer 时，ops 参数置 NULL。
3. 关于互斥：由于通信双方都能拿到的 buffer 的地址，难以在驱动实现互斥，所以需要在具体应用上**自行保证互斥**。

8 其他

8.1 rpmsg 需知

1. 端点是 rpmsg 通信的基础；每个端点都有自己的 src 和 dst 地址，范围（1 - 1023，除了 0x35）
2. **rpmsg 每次发送数据最大为 512-16 字节**；（数据块大小为 512，头部占用 16 字节）
3. rpmsg 使用 name server 机制，当 E907 创建的端点名，和 linux 注册的 rpmsg 驱动名一样的时候，rpmsg bus 总线会调用其 probe 接口。所以如果需要 Linux 端主动发起创建端点并通知 e907，则需要借助上面提到的 rpmsg_ctrl 驱动。
4. rpmsg 是串行调用回调的，故建议 rpmsg_driver 的回调中不要调用耗时长的函数，避免影响其他 rpmsg 驱动的运行

8.2 rpbuf 简介

rpbuf 全志基于 rpmsg 开发的一套通信机制，它主要解决 rpmsg 不适合传输大数据量的问题。其实现原理是使用 rpmsg 传输数据的地址，而不是数据的本身，避免了数据的多次拷贝以及每次传输不能大于 496 字节的限制。

rpbuf 中使用名字和长度来唯一标识一个 buffer，故不能创建相同名字的 buffer。

rpbuf 中的 buffer 有 3 个状态：

1. remote_dummy_buffers：该 buffer 远端已创建，本地未创建
2. local_dummy_buffers：该 buffer 本地已创建，远端未创建
3. buffers：远端、本地已创建，此时 buffer 才可用

8.3 修改 e907 地址

目前在 perf1 板子上，给 e907 预留的内存为：0x48000000 开始的 4M 空间

如果需要修改 E907 固件的运行地址和大小，可按如下步骤进行修改：

8.3.1 修改设备树 (Linux)

```
cconfigs
vim ../board.dts
# 找到e907_dram项, 修改成想要的地址, 例如这里向修改成0x49000000
e907_dram: riscv_memserve {
    reg = <0x0 0x49000000 0x0 0x00400000>;
    no-map;
};
# 重新编译内核
mkkernel
```

8.3.2 修改配置项 (melis)

```
mmelis menuconfig
# 如下图进行修改; e907没有mmu, 故第一项和第二项相等
# 将第一项和第二项改成 0x49000000
# 第三项为大小, 可按需修改
```

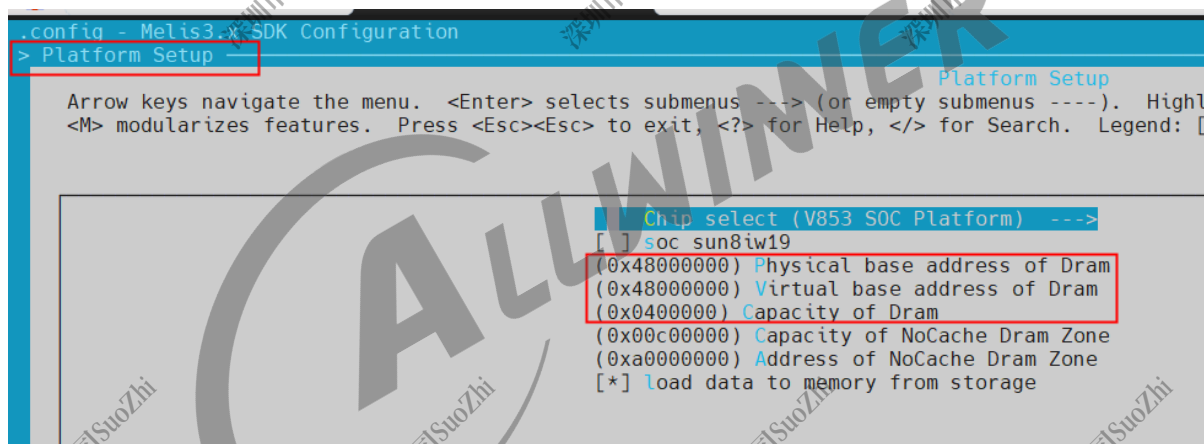


图 8-1: e907 dram config

8.3.3 修改链接脚本 (melis)

```
cmelis
vim source/projects/v853-e907-ver1-board/kernel.lds
# 如下图所示, 按照所需修改 DRAM_SEG_KRN 项目
mmelis -j16
```

```
1: p/v/kernel.lds
1  /*
2  * The OUTPUT_ARCH command specifies the machine architecture where the
3  * argument is one of the names used in the T-HEAD library.
4  */
5  OUTPUT_ARCH("riscv")
6  OUTPUT_FORMAT("elf32-littleriscv", "elf64-littleriscv", "elf32-littleriscv")
7
8  MEMORY
9  {
10     /*SRAM: 64K */
11     SRAM_SEG (rwx) : ORIGIN = 0x00020000, LENGTH = 0x00010000
12
13     /*DRAM_KERNEL: 1M */
14     DRAM_SEG_KRN (rwx) : ORIGIN = 0x48000000, LENGTH = 0x00400000
15 }
16
17 PHDRS
18 {
19     sbi PT_LOAD FLAGS(5); /* PF_R|PF_X */
20     boot PT_LOAD FLAGS(5); /* PF_R|PF_X */
```

图 8-2: e907 lds config

8.4 添加新板级注意事项

当用户需要添加新的板子时，需要注意修改 build/expand_melis.sh 来支持mmelis, cmelis命令。

例如，用户在添加了新的板级v853_user，在 melis 添加了新的板级e907_user，则需要对build/expand_melis.sh文件进行如下修改：

```
1 declare -A board_name_map=(
2 >---["v851_fastboot"]="v851-e907-perf2-board"
3 >---["v851_fastbootD"]="v851-e907-perf2-board"
4 >---["r853_scanp"]="v853-e907-ver1-board"
5 >---["v853_perf1"]="v853-e907-ver1-board"
6 >---["v853_user"]="e907_user"
7 )
8
```

图 8-3: 新板级配置

8.5 melis 系统

8.5.1 常用命令

1. help：列出当前系统支持的所有命令

2. p addr [len]: 打印内存数据
3. m addr value: 修改内存数据
4. top: 显示当前系统各个线程 CPU 占用率
5. ps: 显示当前系统各个线程状态
6. free: 查看当前系统内存信息

8.5.2 自定义命令

当用户想要在 e907 控制台上执行自定义的命令时候，可以用FINSH_FUNCTION_EXPORT_ALIAS导出自定义的函数。例如：

```
36 int user_cmd(int argc, const char **argv)
37 {
38 >---printf("hello world.\n");
39 >---return 0;
40 }
41 FINSH_FUNCTION_EXPORT_ALIAS(user_cmd, hello, user defined cmd);
```

图 8-4: 添加自定义命令

```
date - see now date
rdbuf demo - rdbuf demo
rdbuf_test - rdbuf test demo
eptdev_unbind - unbind rpmsg ept listen
eptdev_bind - bind rpmsg ept name
hello - user defined cmd

msh >
msh >hello
hello world.
msh >
msh >
```

图 8-5: 执行自定义命令

著作权声明

版权所有 © 2022 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的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所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。