MA35 系列非对称多核间处理

NuMicro® 64/32位系列微控制器范例代码介绍

文件信息

应用简述	本范例程序说明如何编写 MA35 家族 AMP 应用程式，透过建立多个端点来与他核建立多通道数据传输。
BSP 版本	Linux-5.10.x
开发平台	NuMaker-HMI-MA35D1-S1 或 NuMaker-IoT-MA35D1-A1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

AMP（非对称多处理）应用于一个多核心平台，一个或一组核心执行独立的操作系统，且多个操作系统间有数据交换需求的场合。MA35 家族平台架构如图 1-1 所示，由双核 Cortex-A35 与 Cortex-M4 组成。MA35 AMP 支持 Cortex-A35 双核间的核间通信 AMP-A 或是 Cortex-A35 与 Cortex-M4 间的核间通信 AMP-M。

AMP 多程序间的通信是通过端点的方式实现，本示例程序示范了 AMP-A 与 AMP-M 多端点间的数据交换，以下分别说明了两者的应用场合与功能。

- AMP-A

这是同质核 AMP，Cortex-A35 双核各自执行独立的操作系统，Core0 执行 Linux，Core1 执行 FreeRTOS。应用程序通过 AMP 建立多个端点，实现点对点多线程双向数据传输。

- AMP-M

这是异质核 AMP，双核 Cortex-A35 与 Cortex-M4 各自执行独立的操作系统，双核 Cortex-A35 执行 Linux，Cortex-M4 执行 FreeRTOS。应用程序通过 AMP 建立多个端点，实现点对点多线程双向数据传输。

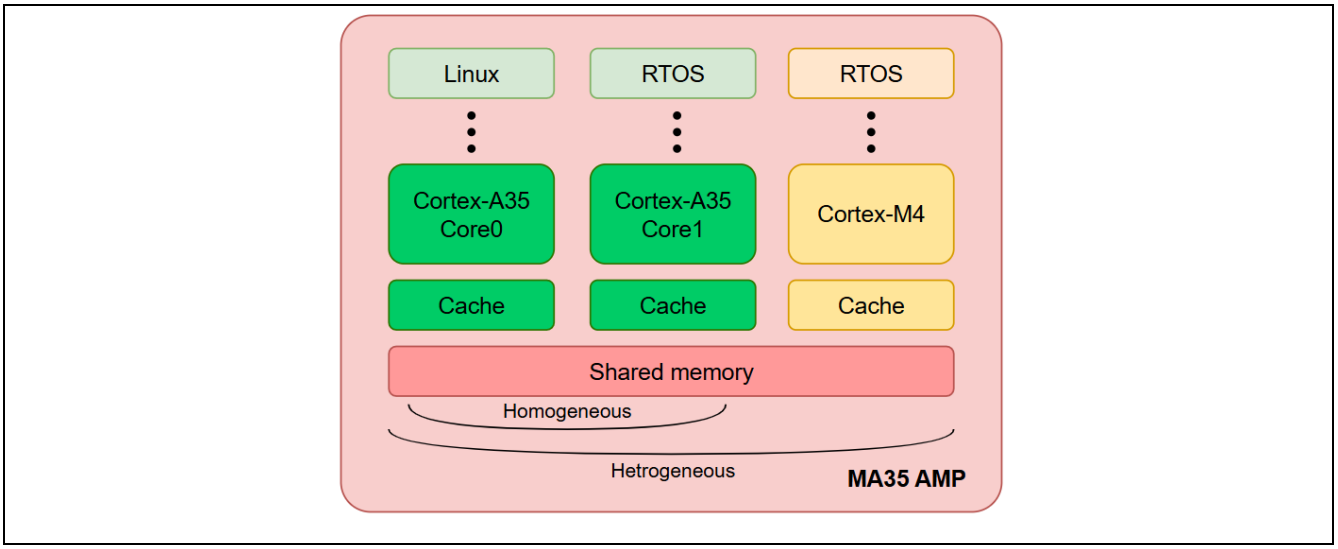

图 1-1 AMP 架构图

1.1 软件架构

AMP 框架可大致分为三个部分，分别有本地核心、远程核心与共享内存。实现核间通信的步骤包含 1. 端点的建立，2. 多核心间端点与端点建立连接，3. 建立连接后的通信。建立端点如图 1-2 所示，其中端点分为传输端点与接收端点，用户在建立传输端点时需提供端点名称与缓冲区大小，建立接收端点时需提供欲连接的端点名称。若端点成功建立，AMP 驱动会通过共享内存的资源表注册端点，并自动完成核与核间名称相匹配之端点的连接。

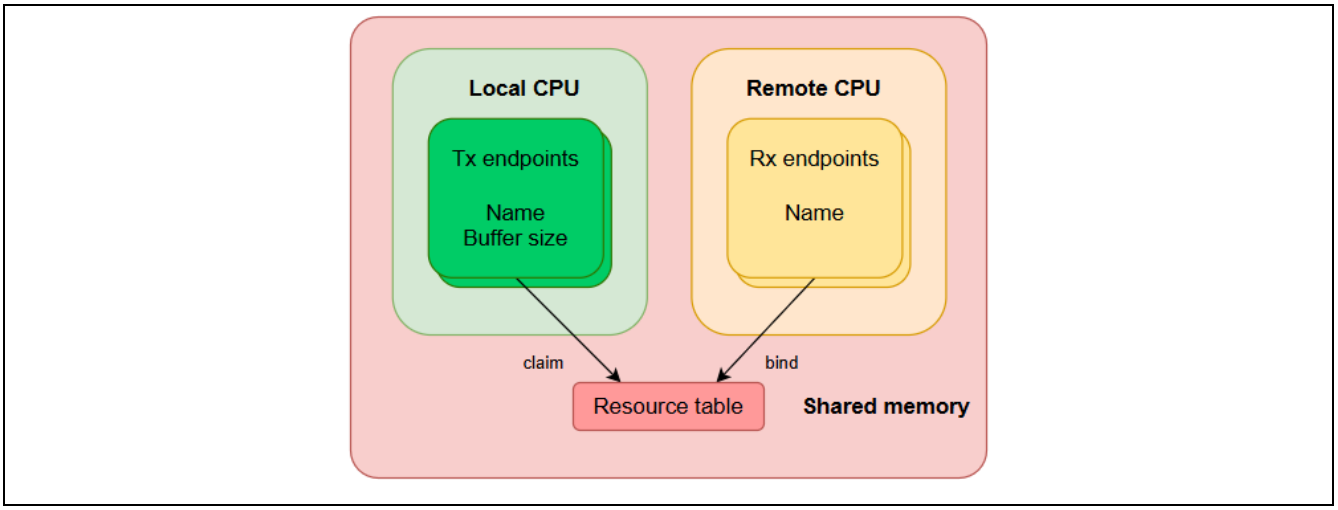

图 1-2 端点建立与连接示意图

重复建立端点的步骤，待驱动完成端点与端点的连接后，可通过共享内存的缓冲池开始多端点数据交换的应用如图 1-3 所示。由此可知，共享内存包含了资源表与缓冲池，其大小必须在设置阶段定义完成，章节 1.2 将详细说明。

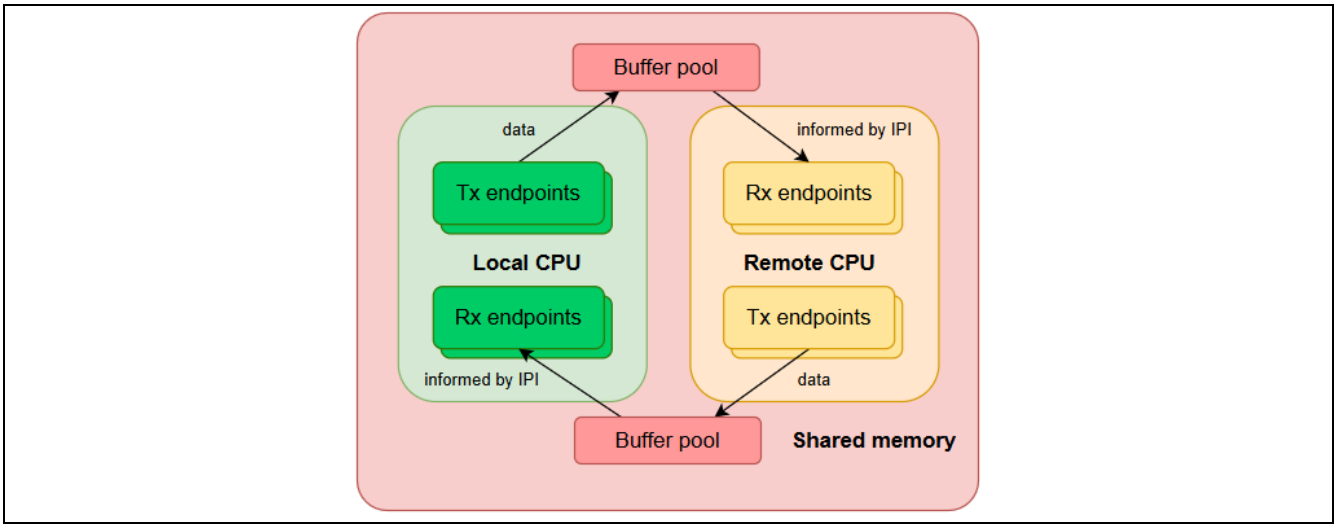

图 1-3 多端点资料交换示意图

1.2 驱动设置

1.2.1 环境设置

前文提到共享内存由资源表与缓冲池组成，用户在设置共享内存时必须考虑两者大小。此示例包含了由本地端到远程以及由远程到本地端各 16 KB 缓冲池，预留给驱动使用的资源表 2 KB，因此共享内存大小总和为 34 KB。一般而言，资源表预设 2 KB 已足够，驱动也会在预留不足的情况回报警告，可参考图 5-2 之执行结果。

新唐提供了 Buildroot 与 Yocto 环境，可通过 TrustedFirmware-A (TF-A)，将映像载入 Core1 或 CM4。请参考 *Buildroot User Manual* 或是 *Yocto User Manual* 开始建立内核映像。

1.2.1.1 Buildroot

AMP-A 的使用场合，请选择载入 A35 Core1 image 与「IPI support」，并将编译完成的 Core1 映像放入*/output/image*。

```
Bootloaders  --->
    [*]   Add SCP BL2 image into FIP Image
            Load Image into FIP Image (A35 Image)  --->
    [*]     IPI support
    (0x84000000) Base address of shared memory for AMP
    (0x8800) Size of shared memory for AMP
    (AMP_Core1RTOS.bin) SCP_BL2 binary file names
    (0x88000000) The execution address of CORE1
    (0x2000000) The execution size of CORE1
```

AMP-M 的使用场合，请选择载入 CM4 image 与「IPI support」，并将编译完成的 CM4 映像放入*/output/image/RTP-BSP*。

```
Bootloaders  --->
    [*]   Add SCP BL2 image into FIP Image
            Load Image into FIP Image (RTP M4 Image)  --->
    [*]     IPI support
    (0x84000000) Base address of shared memory for AMP
    (0x8800) Size of shared memory for AMP
    (RTP-BSP/AMP_CoreMRTOS.bin) SCP_BL2 binary file names
```

参照 *Buildroot User Manual*,完成编译后，若初次使用 TF-A 通过 System Control Processor (SCP) 载入映像，请先执行以下命令，确保环境参数正确更新。

```
$ make arm-trusted-firmware-dirclean
$ make arm-trusted-firmware-rebuild uboot-rebuild linux-rebuild; make
```

1.2.1.2 Yocto

AMP-A 的使用场合，请选择载入 A35 Core1 image 与 「TFA_SCP_IPI」，并将编译完成的 Core1 映像放入 *deploy/images/{machine}*。

/yocto/sources/meta-ma35d1/conf/machine/numaker-som-ma35d16a81.conf:

```
# Load RTP-SCP into FIP image and run M4 or CORE1
TFA_LOAD_SCP = "yes"

# Need to set binary file from deploy/images/{machine}
# yes = M4, no = CORE1
TFA_SCP_M4 = "no"
TFA_SCP_BIN = "AMP_Core1RTOS.bin"

#These are vaild when TFA_SCP_M4 set to "no"
TFA_SCP_IPI = "yes"
TFA_SCP_BASE = "0x88000000"
TFA_SCP_LEN = "0x200000"
```

AMP-M 的使用场合，请选择载入 CM4 image，并将编译完成的 CM4 映像放入 *deploy/images/{machine}/m4proj*。

```
# Load RTP-SCP into FIP image and run M4 or CORE1
TFA_LOAD_SCP = "yes"

# Need to set binary file from deploy/images/{machine}
# yes = M4, no = CORE1
TFA_SCP_M4 = "yes"
TFA_SCP_BIN = "m4proj/AMP_CoreMRTOS.bin"
```

1.2.2 内核设置

使用 AMP-A 或 AMP-M 时，内核需开启相对应的驱动。请注意 AMP-M 仅在有 Cortex-M4 的平台如 MA35D1 系列可使用。

```
Device Drivers --->
    Rpmsg drivers --->
    -*- RPMSG device interface
    < > MA35D1 Shared Memory Driver
    <*> MA35D1 AMP-M Driver
    <*> MA35 series AMP-A Driver
```

1.2.3 设备树设置

编译后请确认 amp 端点或 ampm 端点相对应的 status 属性为「okay」，reserved-memory 标签内的 rpmsg_buf 节点起始地址为 0x84000000，缓冲区与资源表大小为 0x8800 字节。

```
reserved-memory {
    rpmsg_buf: rpmsg_buf@0 {
```

```
        reg = <0x0 0x84000000 0x0 0x8800>
        no-map;
    };
};

ampm: ampm {
    compatible = "nuvoton,ma35d1-ampm";
    memory-region = <&rpmsg_buf>;
    rxipi = <&hwsem 6>;
    txipi = <&hwsem 7>;
    status = "disabled";
};

amp: amp {
    comapaible = "nuvoton,ma35-amp";
    memory-region = <&rpmsg_buf>;
    rxipi = <&timer8>;
    txipi = <&timer9>;
    status = "disabled";
};
```

1.2.4 外设设置

Cortex-A35 Core0 默认使用 UART0 打印系统日志，本示例程序另外将 UART16 配置给远程核心的 Core1 或 Cortex-M4。周边设备的默认权限已满足 AMP-M 的应用场景，对于 AMP-A 的应用场景，请将 UART16 的权限配置给 TZNS(TrustZone non-secure)如下所示。用户可参考 *TF-A User Manual* 完成 SSPCC(System Security Peripheral Configuration Controller)设置。

```
sspcc: sspcc@404F0000 {
    compatible = "nuvoton,ma35d1-sspcc";
    reg = <0x0 0x404F0000 0x0 0x1000>;
    config = <UART0_TZNS>,
        ...
        <UART16_TZNS>,
    };
```

1.2.5 FreeRTOS 设置

在 Core1 或 CM4 项目中，共享内存的设置需要与章节 1.2.1 匹配。本示例使用默认的起始地址 0x84000000，缓冲区大小同样使用 16+16 KB。用户可以在各自的 Core1 或 CM4 项目中找到相应的设置。

OpenAMPConfig.h:

```
/* Configurable parameters */
#define SHARED_RSC_TABLE        ( 0x84000000UL )
#define SHARED_MEM_PA           ( SHARED_RSC_TABLE + SHARED_BUF_OFFSET )
#define RING_TX_SIZE            ( 0x4000 )
#define RING_RX_SIZE            ( 0x4000 )
```

2. 用户程序介绍

此示例程序所示范的同质核 AMP-A 与异质核 AMP-M，在 CA35 Core0 端共用一份 Linux 应用程序，与其搭配的核心则有其各自的 FreeRTOS 项目。由于两项目在 API 的使用方法一致，本文使用 AMP-A 之 Core0 应用与 Core1 应用进行说明。

示例在 Core0 端与 Core1 端都开了 2 个 TX 端点与 2 个 RX 端点，实现 2+2 个 task 的双向沟通。其中任务 A 与任务 B 名称两两相同的端点每 10 毫秒进行约 0.5 KB 数据传输，任务 C 与任务 D 名称两两相同的端点每 100 毫秒进行约 5 KB 数据传输如图 2-1 所示。

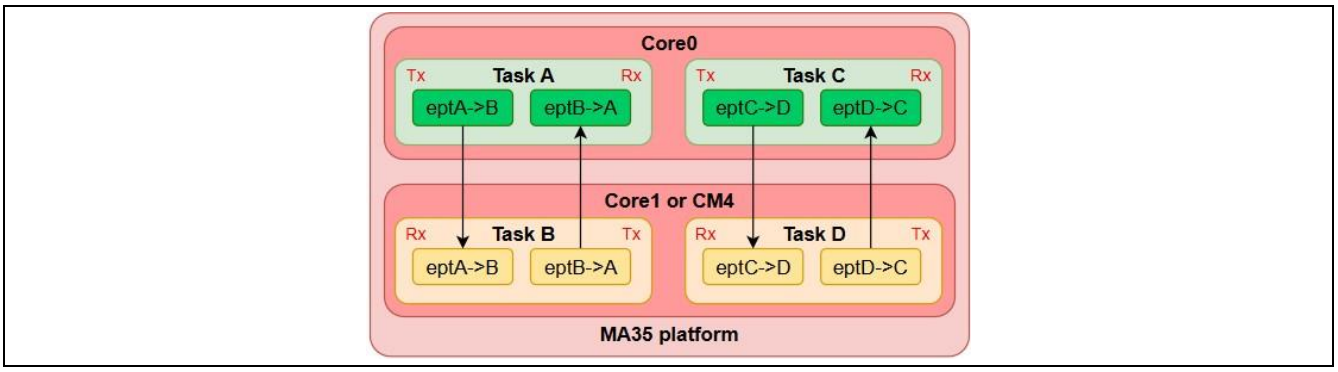

图 2-1 范例架构图

软件流程可分为以下步骤:

1. 初始化并开启端点之母设备。
2. 建立 TX 或 RX 端点及其线程。
3. 待驱动完成端点间的连接后开始数据传输。

用户可参考章节 2.1 与 2.2 应用程序各自的「eptinst」，修改 TX 或 RX 端点开始设计自己的应用。

```
struct amp_endpoint eptinst[] = {};
```

2.1 Linux 应用程序

```
#define RPMSG_CRTL_DEV_ID   0

#define RPMSG_CREATE_EPT_IOCTL _IOW(0xb5, 0x1, struct rpmsg_endpoint_info)
#define RPMSG_DESTROY_EPT_IOCTL _IO(0xb5, 0x2)

#define EPT_TYPE_TX         0x01
#define EPT_TYPE_RX         0x10

#define THREAD_SLEEP_MS     1000

struct rpmsg_endpoint_info {
    char name[32]; // Tx: name of local ept; Rx: name of remote ept to bind with
    __u32 type;    // Tx or Rx ept
    __u32 size;    // Tx: request data length in byte; Rx: reserved
};

struct amp_endpoint {
    struct rpmsg_endpoint_info eptinfo;
    task_fn_t task_fn; // User-designed task
    int fd;
    pthread_t threadHandle;
    void *ept_priv;
};

#define TASKA_TX_SIZE   0x400
#define TASKA_RX_SIZE   0x400
#define TASKC_TX_SIZE   0x2800
#define TASKC_RX_SIZE   0x2800

/* user defined endpoint */
struct amp_endpoint eptinst[] = {
    { {"eptA->B", EPT_TYPE_TX, TASKA_TX_SIZE}, taskA_send }, /* Task A */
    { {"eptB->A", EPT_TYPE_RX}, taskA_read },                /* Task A */
    { {"eptC->D", EPT_TYPE_TX, TASKC_TX_SIZE}, taskC_send }, /* Task C */
    { {"eptD->C", EPT_TYPE_RX}, taskC_read },                /* Task C */
};

int amp_create_ept(int *fd, struct amp_endpoint *amp_ept) {
    char devname[32];
    int id = ioctl(fd[0], RPMSG_CREATE_EPT_IOCTL, &amp_ept->eptinfo);
    if (id < 0) {
        printf("Failed to create endpoint \"%s\".\n", amp_ept->eptinfo.name);
        return id;
    }

    printf("rpmsg%d: %s endpoint \"%s\" is created.\n", id,
           (amp_ept->eptinfo.type == EPT_TYPE_TX) ? "Tx" : "Rx",
           amp_ept->eptinfo.name);

    snprintf(devname, sizeof(devname), "/dev/rpmsg%d", id);
    amp_ept->fd = open(devname, O_RDWR | O_NONBLOCK);
    if (amp_ept->fd < 0) {
        printf("Failed to open device %s.\n", devname);
```

```
        return -1;
    }

    printf("%s: device is opened.\n", devname);
    return id;
}

int amp_open(int *fd) {
    char dev_path[32];
    snprintf(dev_path, sizeof(dev_path), "/dev/rpmsg_ctrl%d", RPMSG_CRTL_DEV_ID);
    fd[0] = open(dev_path, O_RDWR | O_NONBLOCK);
    if (fd[0] < 0) {
        printf("Failed to open AMP.\n");
        return -ENOENT;
    }

    for (int i = 0; i < sizeof(eptinst) / sizeof(eptinst[0]); i++) {
        if (amp_create_ept(fd, &eptinst[i]) < 0) {
            return -EINVAL;
        }
    }
    return 0;
}

int amp_destroy_ept(struct amp_endpoint *amp_ept) {
    if (ioctl(amp_ept->fd, RPMSG_DESTROY_EPT_IOCTL, NULL)) {
        printf("Failed to destroy endpoint \"%s\".\n", amp_ept->eptinfo.name);
        return -1;
    }
    close(amp_ept->fd);
    printf("Endpoint \"%s\" is destroyed.\n", amp_ept->eptinfo.name);
    return 0;
}

int amp_close(int *fd) {
    for (int i = 0; i < sizeof(eptinst) / sizeof(eptinst[0]); i++) {
        amp_destroy_ept(&eptinst[i]);
    }
    close(fd[0]);
    return 0;
}

void *taskA_send(void *arg) {
    struct amp_endpoint *ept = (struct amp_endpoint *)arg;
    char tx_buf[TASKA_TX_SIZE];

    while (1) {
        int size = rand() % TASKA_TX_SIZE;
        size = size ? size : TASKA_TX_SIZE;

        for (int i = 0; i < size; i++) {
            tx_buf[i] = size + i;
        }

        if (write(ept->fd, tx_buf, size) < 0 && errno == EPERM) {
            sysprintf("%s: Remote closed.\n", ept->eptinfo.name);
```

```
        }

        usleep(THREAD_SLEEP_MS * 10);
    }
    pthread_exit(NULL);
}

void *taskA_read(void *arg) {
    struct amp_endpoint *ept = (struct amp_endpoint *)arg;
    char rx_buf[TASKA_RX_SIZE];
    struct pollfd fds = { .fd = ept->fd, .events = POLLIN | POLLHUP };

    while (1) {
        if (poll(&fds, 1, 0) > 0) {
            if (fds.revents & POLLIN) {
                ssize_t rxlen = 0;
                while ((rxlen += read(ept->fd, rx_buf + rxlen, sizeof(rx_buf))) > 0);
            }
            if (fds.revents & POLLHUP) {
                sysprintf("%s: Remote closed.\n", ept->eptinfo.name);
            }
        }
        usleep(THREAD_SLEEP_MS * 2);
    }
    pthread_exit(NULL);
}

// Similar adjustments for taskC_send and taskC_read

int main(int argc, char **argv) {
    printf("\nMA35D1 AMP core#0 demo\n");

    int fd;
    if (amp_open(&fd)) {
        printf("Failed to init AMP.\n");
        exit(1);
    }

    for (int i = 0; i < sizeof(eptinst) / sizeof(eptinst[0]); i++) {
        if (pthread_create(&eptinst[i].threadHandle, NULL, eptinst[i].task_fn,
&eptinst[i])) {
            printf("Create %s error: %s\n", eptinst[i].eptinfo.name, strerror(errno));
            exit(1);
        }
    }

    for (int i = 0; i < sizeof(eptinst) / sizeof(eptinst[0]); i++) {
        pthread_join(eptinst[i].threadHandle, NULL);
    }

    amp_close(&fd);
    return 0;
}
```

2.2 RTOS 应用程序

```
#define EPT_TYPE_TX            0x01
#define EPT_TYPE_RX            0x10

struct amp_endpoint {
    struct rpmsg_endpoint_info eptinfo;
    rpmsg_ept_cb cb;           // Rx endpoint callback
    TaskFunction_t task_fn;    // User designed task
    struct rpmsg_endpoint ept;
    TaskHandle_t taskHandle;
    void *ept_priv;            // Reserved for passing cookies
};

#define TASKB_TX_SIZE   0x400
#define TASKD_TX_SIZE   0x2800

/* User defined endpoints */
struct amp_endpoint eptinst[] = {
    { {"eptA->B", EPT_TYPE_RX}, ReadTaskB_cb, vPollTaskB },        /* Task B */
    { {"eptB->A", EPT_TYPE_TX, TASKB_TX_SIZE}, NULL, vSendTaskB }, /* Task B */
    { {"eptC->D", EPT_TYPE_RX}, ReadTaskD_cb, vPollTaskD },        /* Task D */
    { {"eptD->C", EPT_TYPE_TX, TASKD_TX_SIZE}, NULL, vSendTaskD }, /* Task D */
};
char tx_bufB[TASKB_TX_SIZE];
char tx_bufD[TASKD_TX_SIZE];

TaskHandle_t createTaskHandle;

/**
 * @brief Create endpoint and its task
 *
 * @param rpdev pointer to rpmsg_device
 * @param amp_ept pointer to an inst of amp_endpoint
 * @return int
 */
int amp_create_ept(struct rpmsg_device *rpdev, struct amp_endpoint *amp_ept)
{
    struct rpmsg_endpoint_info *info = &amp_ept->eptinfo;
    int ret;
    static int id = 0; // serial number

    if(info->type == EPT_TYPE_TX) {
        ret = ma35_rpmsg_create_txept(&amp_ept->ept, rpdev, info->name, info->size);
    } else if (info->type == EPT_TYPE_RX) {
        ret = ma35_rpmsg_create_rxept(&amp_ept->ept, rpdev, info->name, amp_ept->cb);
    } else {
        sysprintf("Invalid endpoint type.\n");
        ret = -1;
    }

    if(ret >= 0) {
        xTaskCreate( amp_ept->task_fn, amp_ept->eptinfo.name, configMINIMAL_STACK_SIZE,
&amp_ept->ept, tskIDLE_PRIORITY + 2, &amp_ept->taskHandle );
        sysprintf("rpmsg%d: %s endpoint \"%s\" is created.\n", id++,
```

```
            (info->type == EPT_TYPE_TX) ? "Tx" : "Rx",
            info->name);
    } else {
        sysprintf("Failed to create endpoint \"%s\", err: 0x%x.\n", info->name, ret);
    }

    return ret;
}

/**
 * @brief Start AMP
 *
 * @param rpdev pointer to rpmsg_device
 * @return int
 */
int amp_open(struct rpmsg_device *rpdev)
{
    int i, ret, inst;
    inst = sizeof(eptinst) / sizeof(eptinst[0]);

    for(i = 0; i < inst; i++) {
        ret = amp_create_ept(rpdev, &eptinst[i]);
        if (ret < 0) {
            return -EINVAL;
        }
    }

    return 0;
}

/**
 * @brief Recycle endpoint
 *
 * @param amp_ept pointer to an inst of amp_endpoint
 * @return int
 */
int amp_destroy_ept(struct amp_endpoint *amp_ept)
{
    int ret;

    if(amp_ept->taskHandle)
    {
        vTaskDelete(amp_ept->taskHandle);
        amp_ept->taskHandle = NULL;
    }

    ret = ma35_rpmsg_destroy_ept(&amp_ept->ept);
    if(ret)
        sysprintf("Failed to destroy endpoint \"%s\".\n", amp_ept->eptinfo.name);
    else
        sysprintf("Endpoint \"%s\" is destroyed.\n", amp_ept->eptinfo.name);

    return ret;
}

/**
```

```
 * @brief End AMP
 *
 * @return int
 */
int amp_close(void)
{
    int i, ret, inst;
    inst = sizeof(eptinst) / sizeof(eptinst[0]);

    for(i = 0; i < inst; i++) {
        ret = amp_destroy_ept(&eptinst[i]);
    }

    return 0;
}

/**
 * @brief User Rx callback, do not call this directly.
 *        The function is called when a POLLIN event is received.
 *
 * @param ept rpmsg endpoint
 * @param data pointer to rx buffer provided by driver
 * @param len length of rx buffer
 * @param src unused
 * @param priv unused
 * @return int always return RPMSG_SUCCESS
 */
int ReadTaskB_cb(struct rpmsg_endpoint *ept, void *data, size_t len, uint32_t src, void
*priv)
{
    char *rxbuf = data;
    int rxlen = len;
    (void)src;
    (void)priv;

    /* Start of user read function */
    throughput[0] += rxlen;
    /* End of user read function */

    return RPMSG_SUCCESS;
}

/**
 * @brief User poll task, call ma35_rpmsg_poll to get status
 *
 * @param pvParameters ept
 */
void vPollTaskB( void *pvParameters )
{
    int ret;

    for( ;; )
    {
        ret = ma35_rpmsg_poll(pvParameters);
        if(ret == RPMSG_ERR_PERM)
            sysprintf("%s: Remote closed.\n", pcTaskGetName(NULL));
```

```
        else if(ret == RPMSG_ERR_NO_BUFF)
            sysprintf("%s: Rx buffer is full.\n", pcTaskGetName(NULL));

        vTaskDelay(2);
    }
}

/**
 * @brief User Tx task, call ma35_rpmsg_send to send data
 *
 * @param pvParameters ept
 */
void vSendTaskB( void *pvParameters )
{
    int i, ret, size, j = 0;

    for( ;; )
    {
        /* Start of user write function */
        size = rand();
        size = size % TASKB_TX_SIZE;
        size = size ? size : TASKD_TX_SIZE;

        for(i = 0; i < size; i++)
            tx_bufB[i] = size + i;
        /* End of user write function */

        ret = ma35_rpmsg_send(pvParameters, tx_bufB, size);
        if(ret < 0) {
            if(ret == RPMSG_ERR_PERM)
                sysprintf("%s: Remote closed.\n", pcTaskGetName(NULL));
            else if(ret == RPMSG_ERR_NO_BUFF)
                sysprintf("%s: Tx blocking.\n", pcTaskGetName(NULL));
        }
        else {
            throughput[1] += ret;
        }

        vTaskDelay(10);
    }
}

// Similar adjustments for ReadTaskD_cb, vPollTaskD and vSendTaskD

void vEndpointCreateTask( void *pvParameters )
{
    struct rpmsg_device *rpdev = pvParameters;
    int delay = 1000;

    for( ;; )
    {
        if(ma35_rpmsg_remote_ready())
        {
            amp_open(rpdev);
            delay = ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
        }
```

```
        vTaskDelay(delay);
    }
}

int main(void)
{
    void *platform;
    struct rpmsg_device *rpdev;
    int ret;

    SYS_Init();

    sysprintf("\n\nMA35D1 AMP Core#1 demo\n");
    fflush(stdout);

    /* Initialize platform */
    ret = platform_init(0, NULL, &platform);
    if (ret) {
        sysprintf("Failed to initialize platform.\r\n");
        ret = -1;
    } else {
        rpdev = platform_create_rpmsg_vdev(platform, 0,
                        VIRTIO_DEV_DRIVER, // only driver, we support both tx/rx
                        NULL,
                        NULL);
        if (!rpdev) {
            sysprintf("Failed to create rpmsg virtio device.\r\n");
            ret = -1;
        } else {
            ret = 0;
        }
    }

    if(!ret)
    {
        xTaskCreate( vEndpointCreateTask, "CreateEpt", configMINIMAL_STACK_SIZE, rpdev,
tskIDLE_PRIORITY + 1, &createTaskHandle );
        vTaskStartScheduler();
        /* Should never be reached */
    }

    platform_release_rpmsg_vdev(rpdev, platform);
    platform_cleanup(platform);

    return 0;
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - Linux-5.10.x

3.2 硬件需求

- 电路组件
  - NuMaker-HMI-MA35D1-S1 或 NuMaker-IoT-MA35D1-A1

4. 目录信息

目录结构如下图所示。

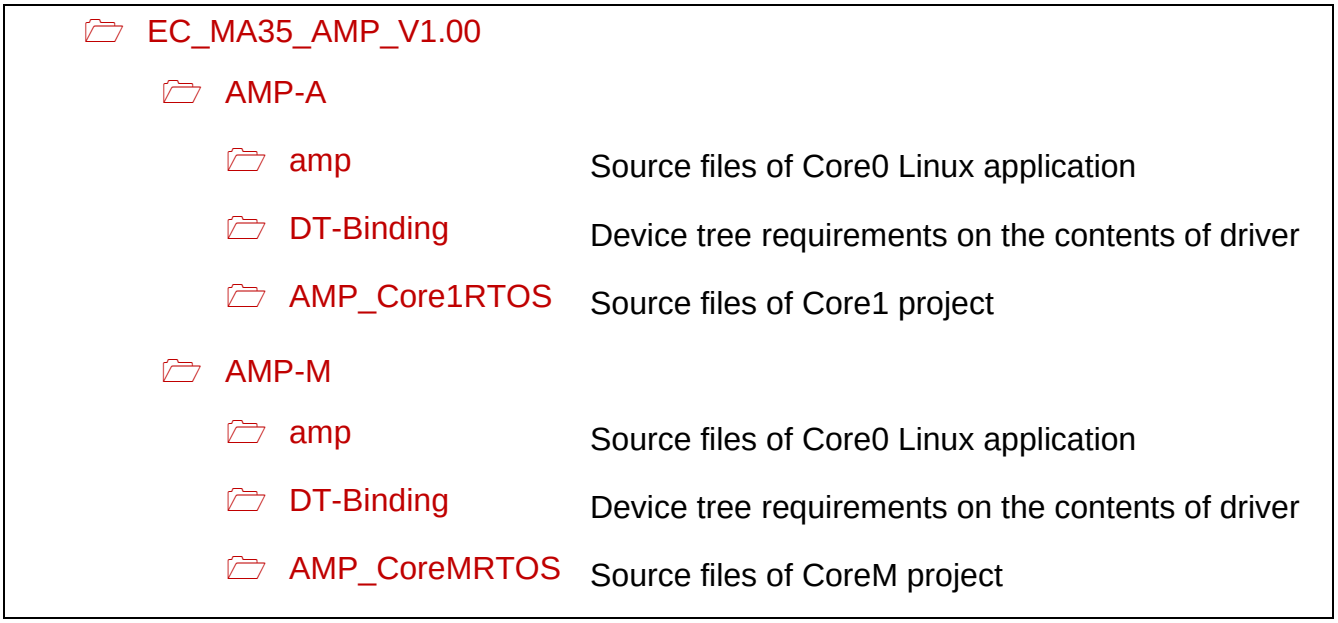

图 4-1 目录信息

5. 范例程序执行

1. 參考 1.2.1 完成环境设置。
2. 參考 1.2.2 完成内核设置。
3. 參考 1.2.3 完成设备树设置。
4. 參考用户手册建立包含 Core1 或 CM4 应用程序的内核映像。
5. 參考用户手册编译 Core0 之 Linux 应用程序。
6. 將内核 image 载入开发版后，执行 Core0 应用程序，开始 AMP 示例。

```
./amp.bin
```

7. 执行后 Core0 日志如图 5-1，Core1 日志如图 5-2，显示应用程序建立端点后，驱动回报名称两两相同的端点建立连接。
8. 开始数据传输。

```
# ./amp.bin

MA35D1 AMP core#0 demo

rpmsg0: Tx endpoint "eptA->B" is created.
/dev/rpmsg0: device is opened.
rpmsg1: Rx endpoint "eptB->A" is created.
/dev/rpmsg1: device is opened.
rpmsg2: Tx endpoint "eptC->D" is created.
/dev/rpmsg2: device is opened.
rpmsg3: Rx endpoint "eptD->C" is created.
/dev/rpmsg3: device is opened.
[   35.308103] rpmsg rpmsg1: Binded with remote ept "eptB->A" with size 1024.
[   35.316603] rpmsg rpmsg3: Binded with remote ept "eptD->C" with size 10240.
[   35.385208] rpmsg rpmsg0: Remote binded to "eptA->B".
[   35.392211] rpmsg rpmsg2: Remote binded to "eptC->D".
```

图 5-1 AMP-A Core0 系统日志

```
MA35D1 AMP Core#1 demo

Initialize remoteproc successfully.
Initializing rpmsg shared buffer pool... finished.
Resource table base: 0x84000000.
Minimum reserved memory size: 0x8600.
Initializing rpmsg IPI device... finished.
Start BindingTask...
rpmsg0: Rx endpoint "eptA->B" is created.
```

```
rpmsg1: Tx endpoint "eptB->A" is created.
rpmsg2: Rx endpoint "eptC->D" is created.
rpmsg3: Tx endpoint "eptD->C" is created.
Binded with remote ept "eptA->B" with size 1,024.
Remote binded to "eptB->A".
Binded with remote ept "eptC->D" with size 10,240.
Remote binded to "eptD->C".
```

图 5-2 AMP-A Core1 系统日志

6. 修订纪录

Date	Revision	Description
2025.01.23	1.00	初始版本。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.