MA35 系列非對稱多核間處理

NuMicro® 64/32位元系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例程式說明如何編寫 MA35 家族 AMP 應用程式，透過建立多個端點來與他核建立多通道資料傳輸。
BSP 版本	Linux-5.10.x
開發平台	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(Asymmetric Multiprocessing)應用於一個多核心平台，一個或一組核心執行獨立的作業系統，且多個作業系統間有資料交換需求的場合。MA35 家族平台架構如圖 1-1 所示，由雙核 Cortex-A35 與 Cortex-M4 組成。MA35 AMP 支援 Cortex-A35 雙核間的核間通訊 AMP-A 或是 Cortex-A35 與 Cortex-M4 間的核間通訊 AMP-M。

AMP 多程序間的通訊是透過端點的方式實現，本範例程式示範了 AMP-A 與 AMP-M 多端點間的資料交換，以下各別說明了兩者的應用場合與功能。

- AMP-A

此為同質核 AMP，Cortex-A35 雙核各自執行獨立的作業系統，Core0 執行 Linux，Core1 執行 FreeRTOS。應用程式透過 AMP 建立多個端點，實現點對點多執行緒雙向資料傳輸。

- AMP-M

此為異質核 AMP，雙核 Cortex-A35 與 Cortex-M4 各自執行獨立的作業系統，雙核 Cortex-A35 執行 Linux，Cortex-M4 執行 FreeRTOS。應用程式透過 AMP 建立多個端點，實現點對點多執行緒雙向資料傳輸。

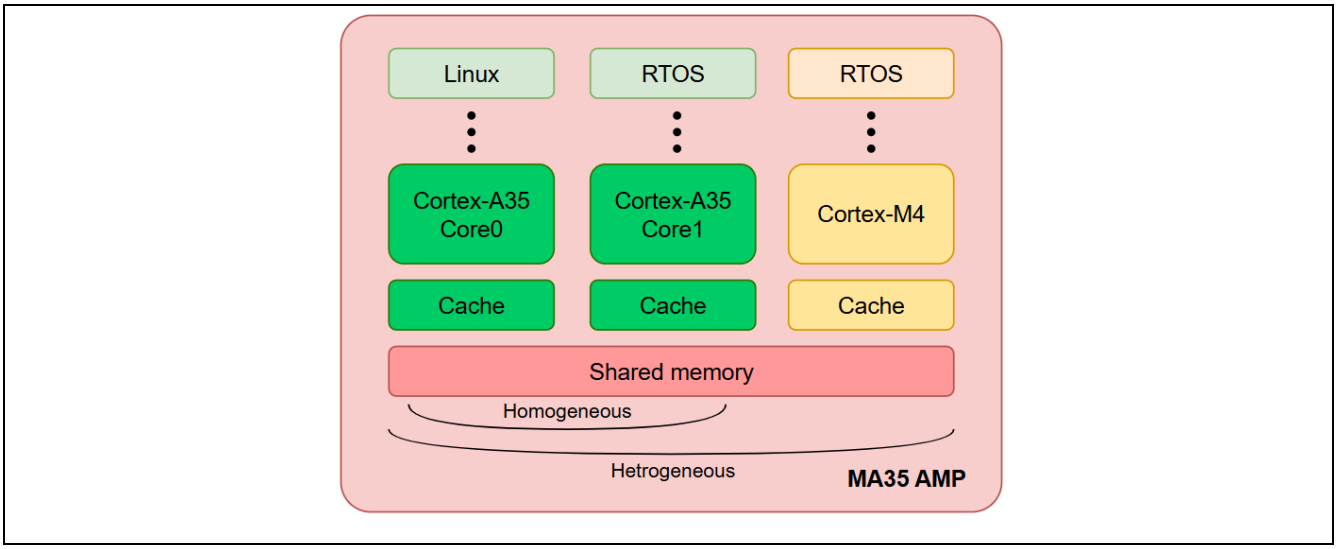

圖 1-1 AMP 架構圖

1.1 軟體架構

AMP 框架可大致分為三個部分，分別有本地核心、遠端核心與共享內存。實現核間通訊的步驟包含 1. 端點的建立，2. 多核心間端點與端點建立連結，3. 建立連結後的通訊。建立端點如圖 1-2所示，其中端點分為傳輸端點與接收端點，使用者在建立傳輸端點時需提供端點名稱與緩衝區大小，建立接收端點時需提供欲連接的端點名稱。若端點成功建立，AMP 驅動會透過共享內存的資源表註冊端點，並自動完成核與核間名稱相匹配之端點的連結。

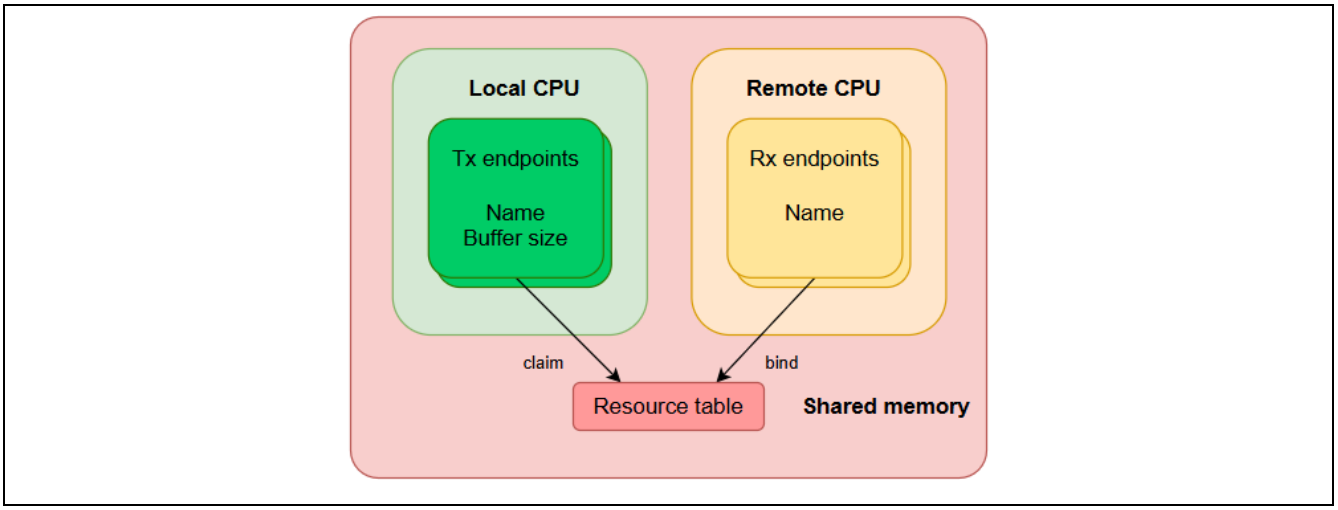

圖 1-2 端點建立與連結示意圖

重複建立端點的步驟，待驅動完成端點與端點的連結後，可透過共享內存的緩衝池開始多端點資料交換的應用如圖 1-3 所示。由此可知，共享內存包含了資源表與緩衝池，其大小必須在設定階段定義完成，章節 1.2 將詳細說明。

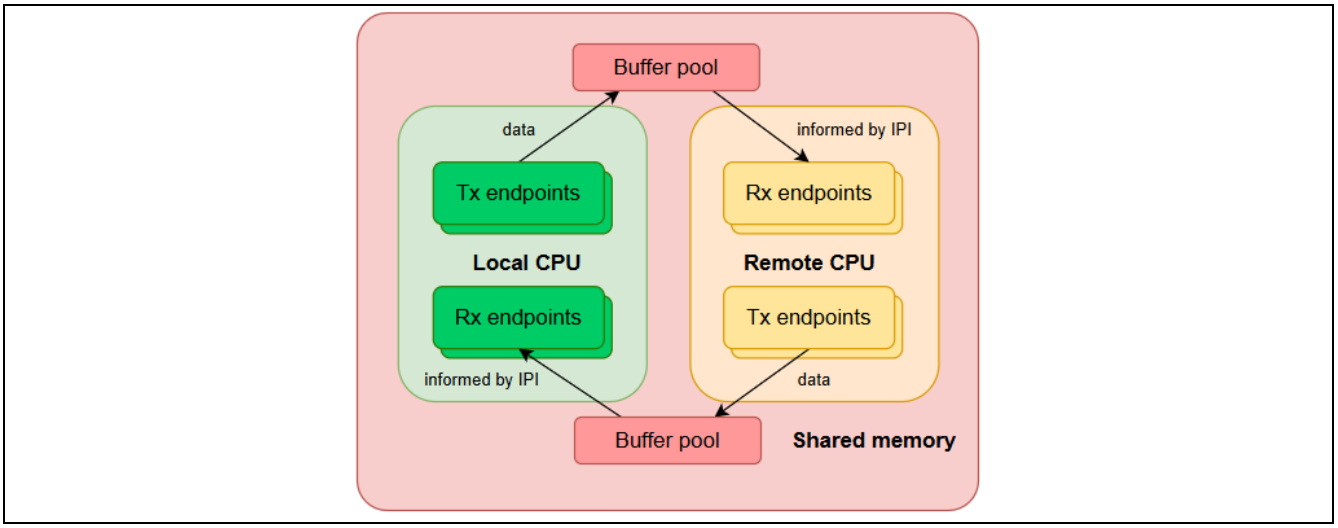

圖 1-3 多端點資料交換示意圖

1.2 驅動設定

1.2.1 環境設定

前文提到共享內存由資源表與緩衝池組成，使用者在設定共享內存時必須考慮兩者大小。此範例包含了由本地端到遠端以及由遠端到本地端各 16 KB 緩衝池，預留給驅動使用的資源表 2 KB，因此共享內存大小總和為 34 KB。一般而言，資源表預設 2 KB 已足夠，驅動也會在預留不足的情況回報警告，可參考圖 5-2 之執行結果。

新唐提供了 Buildroot 與 Yocto 環境，可透過 TrustedFirmware-A (TF-A)，將 image 載入 Core1 或 CM4。請參考 *Buildroot User Manual* 或是 *Yocto User Manual* 開始建立內核 image。

1.2.1.1 Buildroot

AMP-A 的使用場合，請選擇載入 A35 Core1 image 與「IPI support」，並將編譯完成的 Core1 image 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 的使用場合，請選擇載入 CM4 image 與「IPI support」，並將編譯完成的 CM4 image 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
```

參照 *Buildroot User Manual*,完成編譯後，若初次使用 TF-A 透過 System Control Processor (SCP) 載入 image，請先執行以下命令，確保環境參數正確更新。

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

1.2.1.2 Yocto

AMP-A 的使用場合，請選擇載入 A35 Core1 image 與「TFA_SCP_IPI」，並將編譯完成的 Core1 image 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 的使用場合，請選擇載入 CM4 image，並將編譯完成的 CM4 image 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 內核設定

使用 AMP-A 或 AMP-M 時，內核需開啟相對應的驅動。請注意 AMP-M 僅在有 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 設備樹設定

編譯後請確認 amp 端點或 ampm 端點相對應的 status 屬性為「okay」，reserved-memory 標籤內的 rpmsg_buf 節點起始位址為 0x84000000，緩衝區與資源表大小為 0x8800 位元組。

```
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 周邊設定

Cortex-A35 Core0 預設使用 UART0 打印系統訊息，本範例程式另外將 UART16 配置給遠端核心的 Core1 或是 Cortex-M4。周邊裝置的預設權限已滿足 AMP-M 的應用場合，AMP-A 的應用場合，請將 UART16 的權限配置給 TZNS(TrustZone non-secure)如下所示。使用者可參考 *TF-A User Manual* 完成 SSPCC(System security peripheral configuration controller)設定。

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

1.2.5 FreeRTOS 設定

Core1 或是 CM4 專案中，共享內存的設定需要與章節 1.2.1 匹配，此範例使用預設的起始位址 0x84000000，緩衝區大小同樣使用 16+16 KB，使用者可在 Core1 或 CM4 各自的專案找到相對應的設定。

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. 使用者程式介紹

此範例程式所示範的同質核 AMP-A 與異質核 AMP-M，在 CA35 Core0 端共用一份 Linux 應用程式，與其搭配的核心則有其各自的 FreeRTOS 專案。由於兩專案在 API 的使用方法一致，本文使用 AMP-A 之 Core0 應用與 Core1 應用進行說明。

範例在 Core0 端與 Core1 端都開了 2 個 TX 端點與 2 個 RX 端點，實現 2+2 個 task 的雙向溝通。其中任務 A 與任務 B 名稱兩兩相同的端點每 10 毫秒進行約 0.5 KB 資料傳輸，任務 C 與任務 D 名稱兩兩相同的端點每 100 毫秒進行約 5 KB 資料傳輸如圖 2-1 所示。

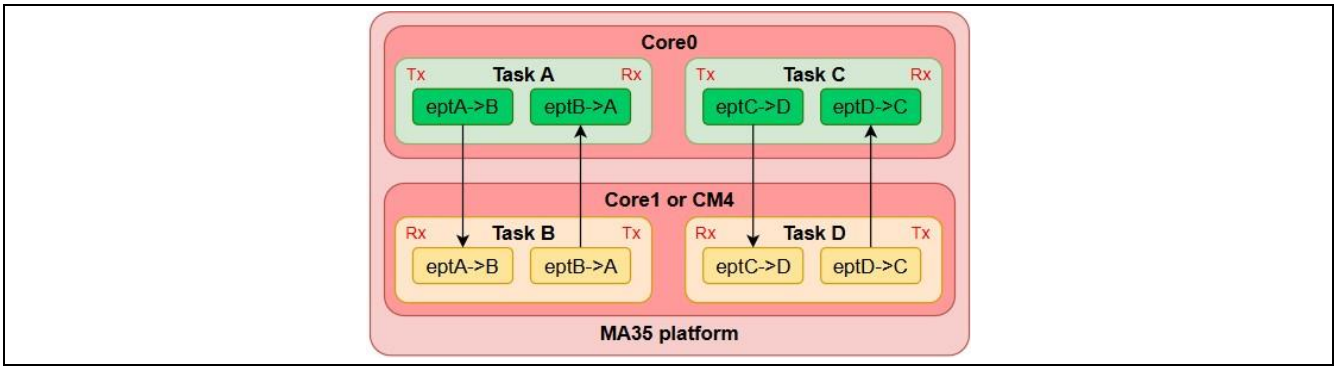

圖 2-1 範例架構圖

軟體流程可分為以下步驟:

1. 初始化並開啟端點之母裝置。
2. 建立 TX 或 RX 端點及其執行緒。
3. 待驅動完成端點間的連接後開始資料傳輸。

使用者可參考章節 2.1 與 2.2 應用程式各自的「eptinst」，修改 TX 或 RX 端點開始設計自己的應用。

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

2.1 Linux 應用程式

```
#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 應用程式

```
#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. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - Linux-5.10.x

3.2 硬體需求

- 電路元件
  - NuMaker-HMI-MA35D1-S1 或 NuMaker-IoT-MA35D1-A1

4. 目錄資訊

目錄結構如下圖所示。

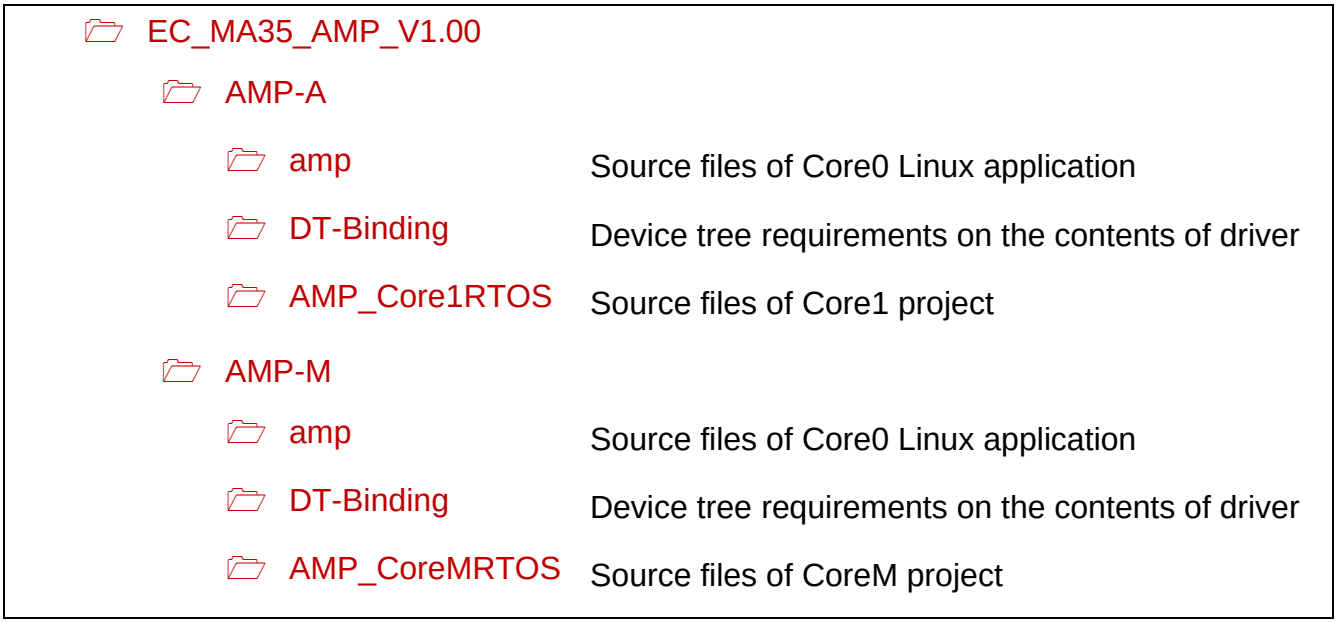

圖 4-1 目錄資訊

5. 範例程式執行

1. 參考 1.2.1 完成環境設定。
2. 參考 1.2.2 完成內核設定。
3. 參考 1.2.3 完成設備樹設定。
4. 參考使用者手冊建立包含 Core1 或 CM4 應用程式的內核 image。
5. 參考使用者手冊編譯 Core0 之 Linux 應用程式。
6. 將內核 image 載入開發版後，執行 Core0 應用程式，開始 AMP 範例。

```
./amp.bin
```

7. 執行後 Core0 紀錄如圖 5-1，Core1 紀錄如圖 5-2，顯示應用程式建立端點後，驅動回報名稱兩兩相同的端點建立連結。
8. 開始資料傳輸。

```
# ./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".
```

圖 5-1 AMP-A Core0 系統訊息

```
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".
```

圖 5-2 AMP-A Core1 系統訊息

6. 修訂紀錄

Date	Revision	Description
2025.01.23	1.00	初始發布。

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.