# Asymmetric Multiprocessing on MA35 Family MPUs

Example Code Introduction for 64/32-bit NuMicro® Family

## Document Information

| | |
|---|---|
| Application | This example program demonstrates how to develop applications for AMP (Asymmetric Multiprocessing) on the MA35 family microprocessors (MPUs) and facilitate multi-channel data transmission with other cores through the process of establishing multiple endpoints. |
| BSP Version | Linux-5.10.x |
| Hardware | NuMaker-HMI-MA35D1-S1 or 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. Overview

AMP (Asymmetric Multiprocessing) is applied to a multi-core platform where one or a group of cores execute independent operating systems, and there is a need for data exchange between multiple operating systems. The MA35 family platform architecture, as shown in Figure 1-1, consists of dual-core Cortex-A35 and Cortex-M4. The MA35 AMP supports inter-core communication between Cortex-A35 dual cores (AMP-A) or between Cortex-A35 and Cortex-M4 (AMP-M).

Communication between multiple processes in AMP is achieved through endpoints. This example program demonstrates data exchange between multiple endpoints in AMP-A and AMP-M, with the following description of their respective application scenarios and functions.

- AMP-A

This is a homogeneous core AMP where the Cortex-A35 dual cores each run independent operating systems, with Core0 running Linux and Core1 running FreeRTOS. The application establishes multiple endpoints through AMP to achieve point-to-point multi-threaded bidirectional data transmission.

- AMP-M

This is a heterogeneous core AMP where the dual-core Cortex-A35 and Cortex-M4 each run independent operating systems, with the dual-core Cortex-A35 running Linux and Cortex-M4 running FreeRTOS. The application establishes multiple endpoints through AMP to achieve point-to-point multi-threaded bidirectional data transmission.

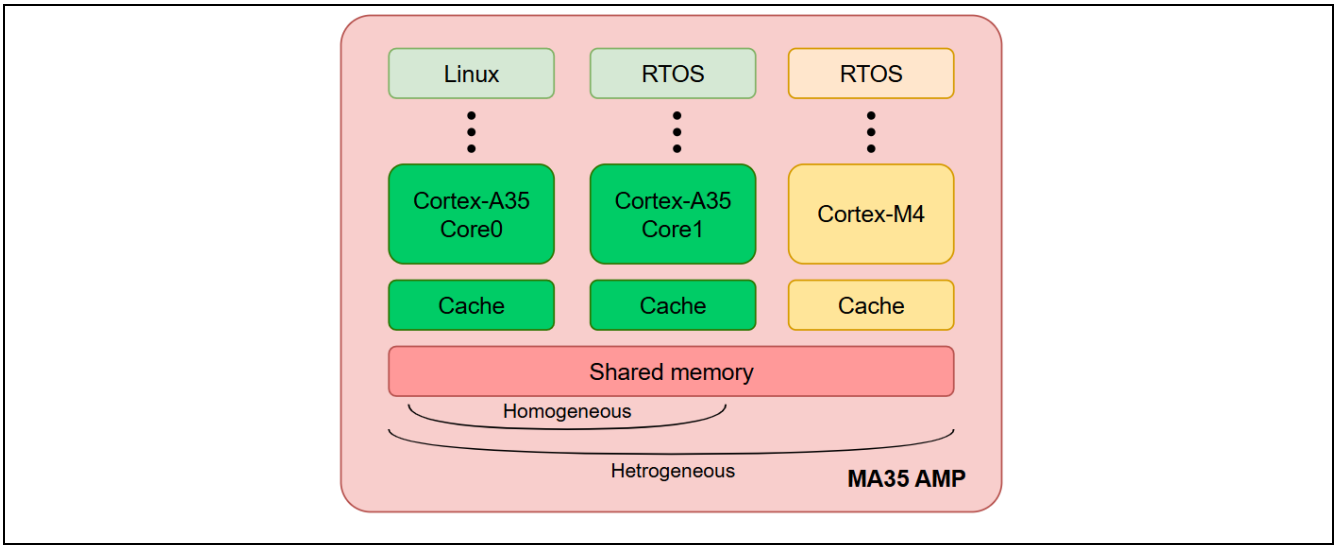


Figure 1-1 AMP Architecture

## 1.1 Software Architecture

The AMP framework can be roughly divided into three parts: local core, remote core, and shared memory. The steps to achieve inter-core communication include: 1. Establishing endpoints; 2. Linking endpoints between multiple cores; and 3. Communication after establishing the link. Establishing endpoints is shown in Figure 1-2, where endpoints are divided into transmission endpoints and reception endpoints. When establishing a transmission endpoint, the user needs to provide the endpoint name and buffer size, and when establishing a reception endpoint, the user needs to provide the name of the endpoint to connect to. If the endpoint is successfully established, the AMP driver will register the endpoint through the resource table of the shared memory and automatically complete the connection of endpoints with matching names between cores.

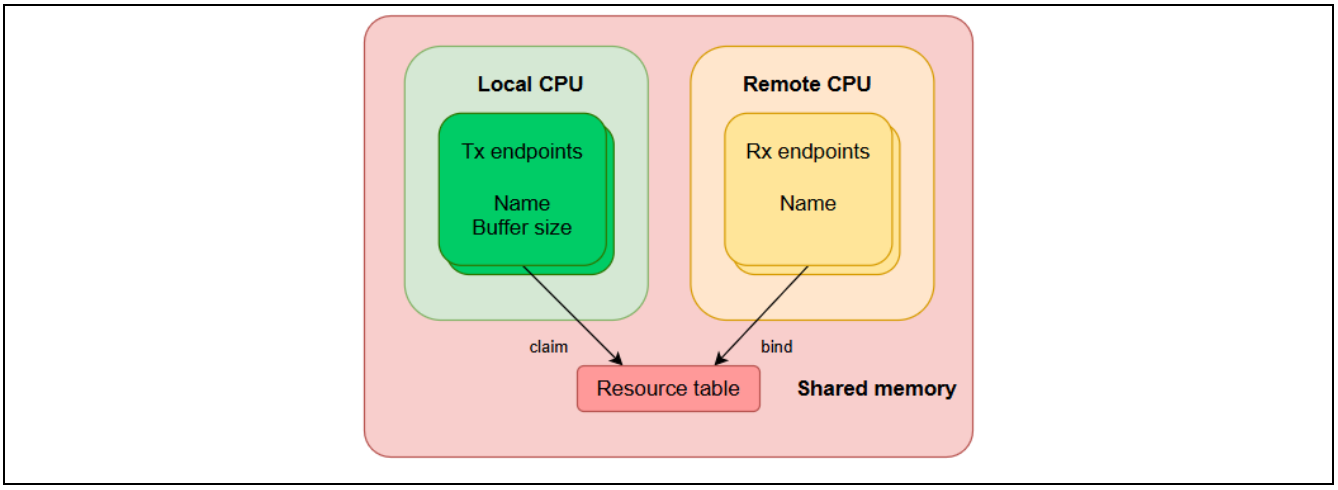


Figure 1-2 Endpoint Creation and Connection

By repeating the steps of establishing endpoints, once the driver completes the connection between endpoints, data exchange between multiple endpoints can begin through the shared memory buffer pool, as shown in Figure 1-3. It can be inferred that the shared memory includes both the resource table and the buffer pool. The size of these components must be defined during the setup phase, which will be detailed in section 1.2.

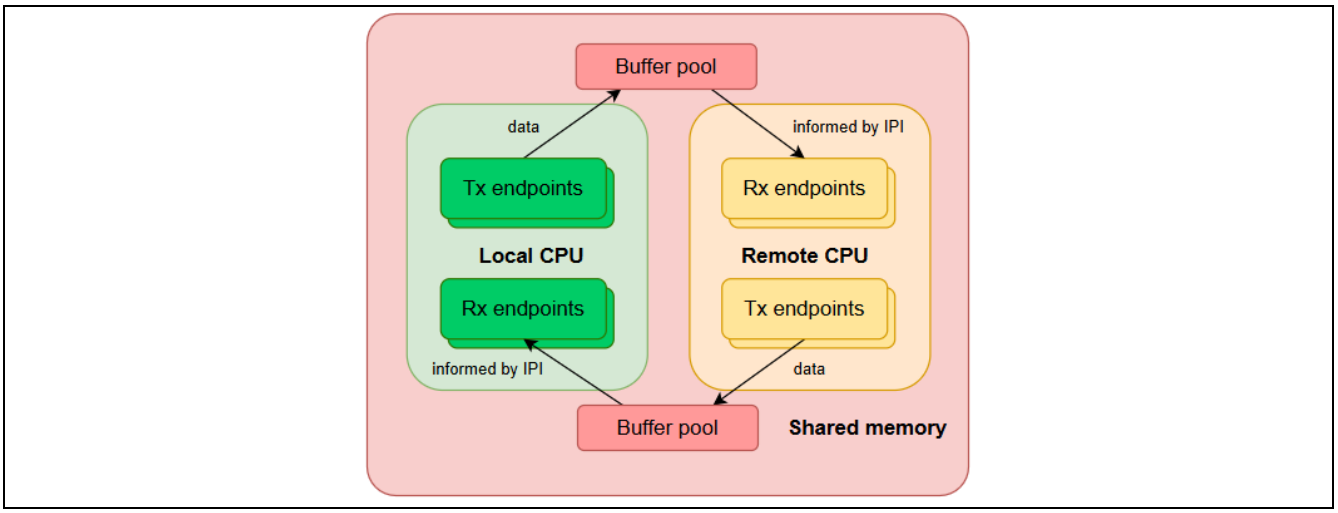


Figure 1-3 Multi-endpoint Data Exchange

## 1.2 Driver Settings

### 1.2.1 Environment Setup

As mentioned earlier, the shared memory consists of a resource table and a buffer pool. When setting up the shared memory, the user must consider the size of both. This example includes a 16 KB buffer pool from the local end to the remote end and a 16 KB buffer pool from the remote end to the local end, with a 2 KB resource table reserved for the driver, making the total shared memory size 34 KB. Generally, a 2 KB resource table is sufficient, and the driver will report a warning if the reserved space is insufficient, as shown in the execution result in Figure 5-2.

Nuvoton provides Buildroot and Yocto environments, which can load images into Core1 or CM4 through TrustedFirmware-A (TF-A). Please refer to the *Buildroot User Manual* or *Yocto User Manual* to start building the kernel image.

#### 1.2.1.1 Buildroot

For AMP-A applications, select the A35 Core1 image with "IPI support" and place the compiled Core1 image in */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
```

For AMP-M applications, select the CM4 image with "IPI support" and place the compiled CM4 image in */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
```

After completing the compilation according to the *Buildroot User Manual*, if using TF-A for the first time to load an image through the System Control Processor (SCP), please execute the following commands to ensure the environment parameters are correctly updated.

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

#### 1.2.1.2 Yocto

For AMP-A applications, select the A35 Core1 image with "TFA_SCP_IPI" and place the compiled Core1 image in *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"
```

For AMP-M applications, select the CM4 image and place the compiled CM4 image in *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 Kernel Setup

When using AMP-A or AMP-M, the kernel needs to enable the corresponding driver. Note that AMP-M is only available on platforms with Cortex-M4, such as the MA35D1 series.

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

### 1.2.3 Device Tree Setup

After compilation, ensure that the status attribute corresponding to the amp endpoint or ampm endpoint is "okay". The starting address of the rpmsg_buf node within the reserved-memory label should be 0x84000000, and the buffer and resource table size are 0x8800 bytes.

```
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 Peripheral Setup

The Cortex-A35 Core0 is configured by default to use UART0 for printing system log. In this example, UART16 is additionally assigned to the remote core, either Core1 or Cortex-M4. The default permissions of peripheral devices meet the requirements for AMP-M applications. For AMP-A applications, please assign the permissions of UART16 to TZNS (TrustZone non-secure) as shown below. Users can refer to the *TF-A User Manual* to complete the SSPCC (System Security Peripheral Configuration Controller) settings.

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

### 1.2.5 FreeRTOS setup

In the Core1 or CM4 project, the shared memory configuration must match section 1.2.1. This example uses the default starting address 0x84000000, with a buffer size of 16+16 KB. Users can find the corresponding settings in their respective Core1 or CM4 projects.

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. Code Description

This example program demonstrates homogeneous core AMP-A and heterogeneous core AMP-M, sharing a Linux application on the CA35 Core0 end, with their respective FreeRTOS projects on the corresponding cores. Since the API usage methods are consistent between the two projects, this document uses the AMP-A Core0 application and Core1 application for explanation.

The example opens 2 TX endpoints and 2 RX endpoints on both Core0 and Core1, achieving 2+2 task bidirectional communication. Tasks A and B, with endpoints of the same name, perform approximately 0.5 KB data transmission every 10 milliseconds, while tasks C and D, with endpoints of the same name, perform approximately 5 KB data transmission every 100 milliseconds, as shown in Figure 2-1.

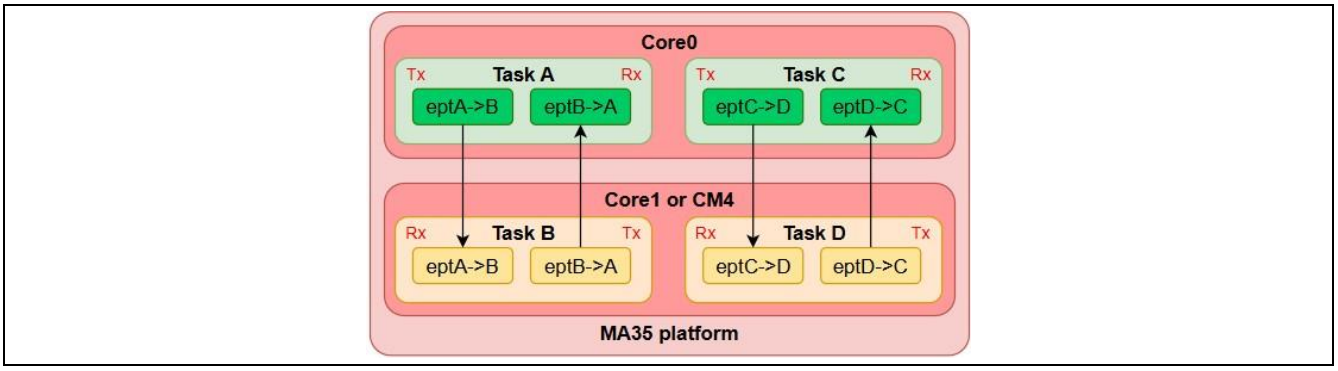


Figure 2-1 Example Architecture

The software process can be divided into the following steps:

1. Initialize and open the parent device of the endpoints.
2. Establish TX or RX endpoints and their threads.
3. Start data transmission after the driver completes the connection between endpoints.

Users can refer to sections 2.1 and 2.2 Linux and RTOS application for their respective "eptinst" to modify TX or RX endpoints and start designing their applications.

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

## 2.1 Linux Application

```
#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 Application

```
#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. Software and Hardware Requirements

## 3.1 Software Requirements

- BSP version
  - Linux-5.10.x

## 3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35D1-S1 or NuMaker-IoT-MA35D1-A1

# 4. Directory Information

The directory structure is shown below.

| | |
|---|---|
| EC_MA35_AMP_V1.00 | |
| AMP-A | |
| amp | Source files of Core0 Linux application |
| DT-Binding | Device tree requirements on the contents of driver |
| AMP_Core1RTOS | Source files of Core1 project |
| AMP-M | |
| amp | Source files of Core0 Linux application |
| DT-Binding | Device tree requirements on the contents of driver |
| AMP_CoreMRTOS | Source files of CoreM project |

Figure 4-1 Directory Information

# 5. Example Code Execution

1. Complete the environment setup (refer to section 1.2.1).
2. Complete the kernel setup (refer to section 1.2.2).
3. Complete the device tree setup (refer to section 1.2.3).
4. Build the kernel image containing the Core1 or CM4 image (refer to the relative user manual).
5. Compile the Linux application for Core0 (refer to the relative user manual).
6. Load the kernel image into the development board, execute the Core0 application, and start the AMP example.

```
./amp.bin
```

7. After execution, Core0 log as shown in Figure 5-1, and Core1 log as shown in Figure 5-2, indicate that the driver reports the connection of endpoints with matching names after the application establishes the endpoints.
8. Start data transmission.

```
# ./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".
```

Figure 5-1 AMP-A Core0 System Log

```
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".
```

Figure 5-2 AMP-A Core1 System Log

# 6. Revision History

| Date | Revision | Description |
|---|---|---|
| 2025.01.23 | 1.00 | Initial version. |

## 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.*