How to exchange large data buffers with the coprocessor - principle

How to exchange large data buffers with the coprocessor - principle

Stable: 12.02.2020 - 15:40 / Revision: 12.02.2020 - 15:36

Contents

1 Introduction

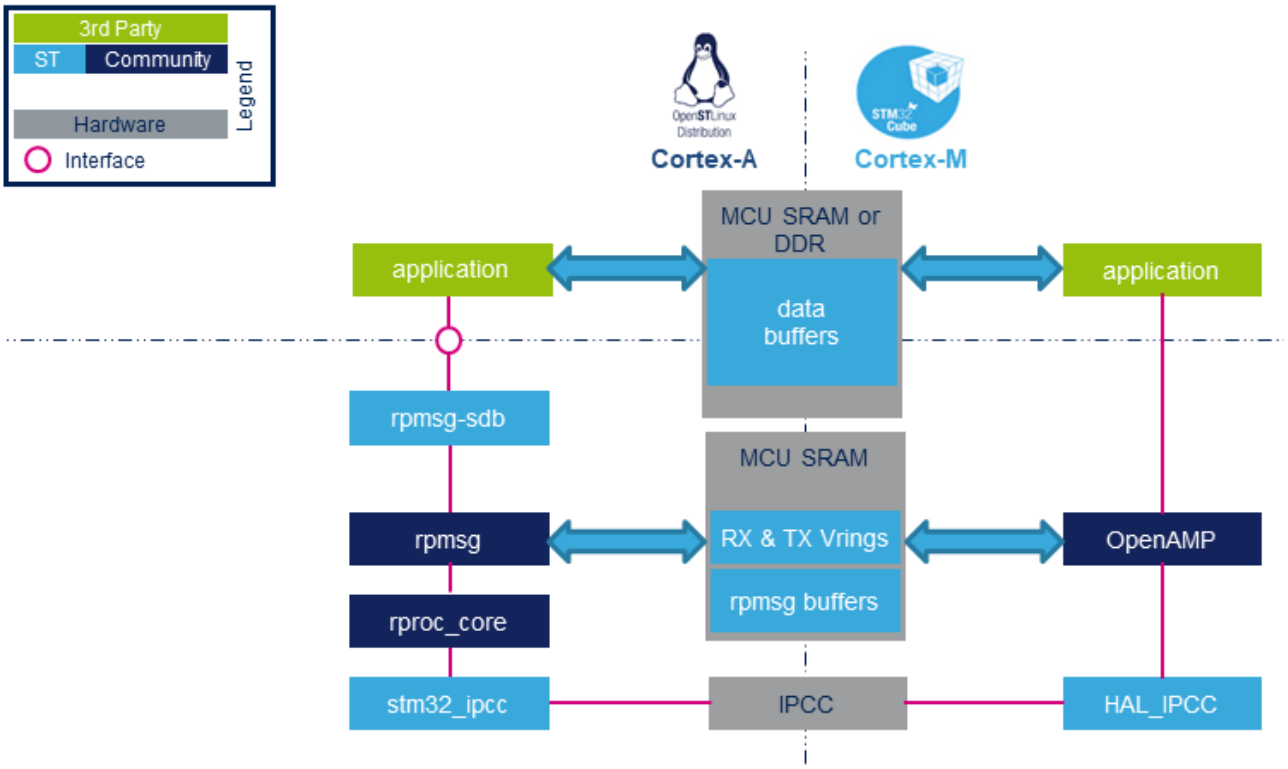
As explained in "Exchanging_buffers_with_the_coprocessor", the RPMsg protocol may be not efficient enough to directly exchange large buffers between the Cortexes. In this case implementing the indirect buffer exchange mode is recommended:

- allocate contiguous buffers to store the data to exchange
- use RPMsg to exchange references to these buffers with the remote processor.

This article gives an example mechanism that can be implemented to exchange indirect buffers between a main processor and a coprocessor.

2 Architecture overview

This architecture example relies on the rpmsg_sdb linux driver.



- On Cortex-A:
  - The rpmsg_sdb Linux driver implements the service to allocate and share buffers with the Cortex-M.
  - The Linux application requests and memory-maps (mmap) the buffers needed to access the associated memory.
- On Cortex-M:
  - The application has to implement the "rpmsg-sdb-channel" RPMsg service to manage the buffer information.
  - A DMA can be used to transfer data to/from DDR.

Refer to How to exchange large data buffers with the coprocessor - example for an example.

3 rpmsg_sdb driver

The RPMsg shared data buffer driver example is in charge of:

- Allocating large buffers in contiguous memory (DDR) and memory mapping them (mmap) for use by an application.
- implementing the RPMsg service to share buffer information (address, size) with the coprocessor.
- Sending events to a Linux application (relying on the eventfd interface) when buffers are available (on RPMsg message reception).

> This driver is provided as example. It implements only the transfer from Cortex-M to cortex-A

3.1 Configuration

3.1.1 Kernel configuration

No kernel configuration is needed. The rpmsg_sdb Linux driver is proposed as module and can be installed using the associated Yocto recipe.

3.1.2 Device tree

No device tree declaration is needed. The rpmsg_sdb driver is registered as an RPMsg driver. it is probed when the remote processor creates the "rpmsg-sdb-channel" service.

3.1.3 Source code

The source code is available in the rpsmg-sdb-mod Yocto recipe.

3.2 How to use

3.2.1 User space interface

The rpmsg_sdb driver exposes a **"/dev/rpmsg_sdb"** sysfs that offers an interface to allocate and manage the shared buffers.

- open/close: get/release file descriptor.

```
int fd;
fd= open('/dev/rpmsg_sdb');
close(fd);
```

- mmap: allocate and map memories

```
void *buff0_id, *buff1_id;

buff0_id = mmap(NULL, size,  PROT_READ | PROT_WRITE,         MAP_PRIVATE, fd, 0);
buff1_id = mmap(NULL, size,  PROT_READ | PROT_WRITE,         MAP_PRIVATE, fd, 0);
```

- RPMSG_SDB_IOCTL_SET_EFD ioctl: register event for a buffer

```
typedef struct
{
        int bufferId, eventfd;
} rpmsg_sdb_ioctl_set_efd;

int efd[NB_BUF];
rpmsg_sdb_ioctl_set_efd q_set_efd;

for (i=0;i<NB_BUF;i++){
      /* Create the evenfd, and sent it to kernel driver, for notification of buffer
full */
        efd[i] = eventfd(0, 0);
```

```
}
q_set_efd.bufferId = i;  /* i is the index of the buffer */
q_set_efd.eventfd = efd[i];

ioctl(fd, RPMSG_SDB_IOCTL_SET_EFD, buff0_id, &q_set_efd);
```

- RPMSG_SDB_IOCTL_GET_DATA_SIZE ioctl : get the size of a buffer

```
typedef struct
{
        int bufferId;
        uint32_t size;
} rpmsg_sdb_ioctl_get_data_size;

rpmsg_sdb_ioctl_get_data_size q_get_data_size;

ioctl(fd, RPMSG_SDB_IOCTL_GET_DATA_SIZE, buff0_id, eventfd);
```

p*manage event

```
while (1) {
      ret = poll(fds, NB_BUF, TIMEOUT * 1000);
      if (ret < 0) {
              perror("poll()");
              break;
      } else if (ret) {
              printf("Data is available now.\n");
      } else if (ret == 0) {
              printf("No data within five seconds.\n");
              break;
      }
      for (j=0;j<NB_BUF;j++){
              if (fds[j].revents & POLLIN) {
                      /* Event received for the buffer j: New data is available for
buffer j */
              }
      }
}
```

3.2.2 RPMsg messaging

The RPMsg protocol is used for communication with the Cortex-M:

- Information about the buffer allocated and mmaped is sent to the Cortex-M.

The message is structured in a string with following format: **"BxAyyyyyyyyLzzzzzzzz"**

- **x**: buffer index (32 bits, decimal format, no leading zero)
- **yyyyyyyy**: physical address of the buffer in DDR (32 bits, 8-digit hexadecimal format, leading zero)
- **zzzzzzzz**: length of the buffer (32 bits, 8-digit hexadecimal format, leading zero).

- Buffer filled event received from the Cortex-M:

When the Cortex-M4 has filled a buffer it can inform the Linux application by sending an RPMsg with following string format : **"BxLzzzzzzzz"'**.

- **x**: buffer index (32 bits, decimal format, no leading zero)
- **zzzzzzzz**: length of the buffer (32 bits, 8-digit hexadecimal format, leading zero).

On reception of this message the rpmsg_sdb driver sends an event to the application.

Remote Processor Messaging

Direct Memory Access

Doubledata rate (memory domain)

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)