Coprocessor resource table

A quality version of this page, approved on *11 February 2019*, was based off this revision.

Template:ArticleMainWriter Template:ArticleApprovedVersion

Contents

1 Role of the resource table

The resource table is a global variable declared as a structure in the coprocessor firmware. This table contains resources that the remote processor requires before being powered on, such as the allocation of a physically contiguous memory. In addition, the resource table may also contain resource entries that publish the existence of supported features or configurations by the remote processor, such as trace buffers and/or supported Virtio devices used for the IPC.

This table must be defined in a specific data section of the coprocessor firmware, parsed by the RemoteProc Linux® framework during the firmware load phase to:

- Allocate memories defined in the resource table carveout section (not used in the ST Arm® Cortex®-M4 firmware).
- Load the RPMsg and Virtio frameworks to support messaging services.
- Provide a user sysfs interface to access coprocessor traces for debug.

2 How to define the resource table

The resource table must be part of the firmware's ELF image, in order to be accessible by the Linux RemoteProc framework. This resource table can be a default table or customized depending on the enabled features.

In the STM32MCU cube firmware package, the resource table is defined in rsc_table.c

2.1 Default table

For the Cortex-M firmware that does not require interaction with the Linux OS, a default structure must be declared in the Cortex-M firmware

```
struct remote_resource_table __resource __attribute__((used)) rproc_resource = {
        .version = 1,
        .num = 0,
        .reserved = {0, 0},
        .offset = { 0 },
};
```

2.2 How to add trace for the log buffer

This feature allows to dump Cortex-M firmware traces on the linux side. For this a **system_log_buf** buffer defined in log.c file and declared in the resource table allows Linux to dump the associated memory area.

```
struct remote_resource_table __resource __attribute__((used)) rproc_resource = {
        .version = 1,
       .num = 1,  /*  rely to number of offsetof structures declared in .offset */
        .reserved = {0, 0},
        .offset = {
                offsetof(struct remote_resource_table, cm_trace),
        },
        .cm_trace = {
                RSC_TRACE,
                (uint32_t)system_log_buf, SYSTEM_TRACE_BUF_SZ, 0, "cm4_log",
        },
};
```

These logs can be retrieved after a firmware crash: refer to How to retrieve Cortex-M4 logs after crash for more information.

2.3 How to add RPMsg inter-processor communication

The messaging service is enabled by declaring:

- The rpmsg Virtio device for control,
- The rpmsg Virtio ring buffers for message management.

These structures are used by the Linux RemoteProc framework. The Remoteproc framework is in charge of allocating buffers associated to Vring (for both direction) in the shared memory and of providing information in the rpmsg_vring structures.

```
#define NUM_VRINGS                  0x02  /* number of Vring used , must be fixed to 2
(one for TX one for RX) */
#define VRING_ALIGN                 0x1000 /* must be fixed to 0x1000 (Linux constraint)
*/
#define VRING_TX                    -1  /* allocated by master processor */
#define VRING_RX                    -1  /* allocated by master processor */
#define VRING_SIZE                  8  /* number of 512 bytes buffers associated to a
Vring: can be customized */
 struct remote_resource_table __resource __attribute__((used)) rproc_resource = {
         .version = 1,
         .num = 1,  /* rely to number of offsetof structures declared in .offset */
         .reserved = {0, 0},
         .offset = {
                 offsetof(struct remote_resource_table, rpmsg_vdev),
         },
         /* Virtio device entry */
        .rpmsg_vdev= {
                RSC_VDEV, VIRTIO_ID_RPMSG_, 0, RPMSG_IPU_C0_FEATURES, 0, 0, 0,
                NUM_VRINGS, {0, 0},
        },

        /* Vring rsc entry - part of vdev rsc entry */
        .rpmsg_vring0 = {VRING_TX, VRING_ALIGN, VRING_SIZE, 1, 0},
        .rpmsg_vring1 = {VRING_RX, VRING_ALIGN, VRING_SIZE, 2, 0},
 };
```