perplexity

Detailed Explanation of the Resource Table Definition

The following code defines a **resource table** for RPMessage-based communication between a microcontroller (such as an M4F core) and Linux. This table is a special data structure that the Linux remoteproc framework reads when it loads the firmware for a remote processor. It describes what resources and features the remote firmware needs and supports, such as shared memory for messaging and trace buffers for debugging.

Let's break down the code line by line, assuming you are a beginner:

```
const RPMessage_ResourceTable gRPMessage_linuxResourceTable __attribute__ ((section (".re
{
    {
        1U,         /* we're the first version that implements this */
        2U,         /* number of entries, MUST be 2 */
        { 0U, 0U, } /* reserved, must be zero */
    },...
    {
        offsetof(RPMessage_ResourceTable, vdev),
        offsetof(RPMessage_ResourceTable, trace),
    },
    /* vdev entry */
    {
        RPMESSAGE_RSC_TYPE_VDEV, RPMESSAGE_RSC_VIRTIO_ID_RPMSG,
        0U, 1U, 0U, 0U, 0U, 2U, { 0U, 0U },
    },
    /* the two vrings */
    { RPMESSAGE_RSC_VRING_ADDR_ANY, 4096U, 256U, 1U, 0U },
    { RPMESSAGE_RSC_VRING_ADDR_ANY, 4096U, 256U, 2U, 0U },
    {
        (RPMESSAGE_RSC_TRACE_INTS_VER0 | RPMESSAGE_RSC_TYPE_TRACE),
        (uint32_t)gDebugMemLog, DebugP_MEM_LOG_SIZE,
        0, "trace:m4fss0_0",
    },
};
```

Line-by-Line Breakdown

1. Declaration and Attributes

```
const RPMessage_ResourceTable gRPMessage_linuxResourceTable __attribute__ ((section (".re
```

- `const RPMessage_ResourceTable gRPMessage_linuxResourceTable`: Declares a constant variable of type `RPMessage_ResourceTable` named `gRPMessage_linuxResourceTable`.

- `__attribute__ ((section (".resource_table"), aligned (4096)))`: Tells the compiler to place this variable in a special memory section called `.resource_table` and align it to a 4096-byte boundary. This is required so that Linux can easily find and read this table when loading the firmware[1] [2].

2. Resource Table Header

```
{
    1U,         /* we're the first version that implements this */
    2U,         /* number of entries, MUST be 2 */
    { 0U, 0U, } /* reserved, must be zero */
},
```

- This is the **header** of the resource table[1] [3]:
  - `1U`: Version number of the resource table format (should be 1).
  - `2U`: Number of resource entries that follow (in this case, 2).
  - `{ 0U, 0U }`: Reserved fields, must be zero.

3. Offsets to Resource Entries

```
{
    offsetof(RPMessage_ResourceTable, vdev),
    offsetof(RPMessage_ResourceTable, trace),
},
```

- This array specifies the **offsets** (positions) in the structure where each resource entry starts[3] [1].
  - `offsetof(RPMessage_ResourceTable, vdev)`: Offset of the vdev (virtio device) entry.
  - `offsetof(RPMessage_ResourceTable, trace)`: Offset of the trace buffer entry.
- These offsets allow Linux to find each resource entry in the table.

4. Virtio Device (vdev) Entry

```
{
    RPMESSAGE_RSC_TYPE_VDEV, RPMESSAGE_RSC_VIRTIO_ID_RPMSG,
    0U, 1U, 0U, 0U, 0U, 2U, { 0U, 0U },
},
```

- This entry describes a **virtio device** (used for RPMsg communication)[2] [3]:
  - `RPMESSAGE_RSC_TYPE_VDEV`: Indicates this is a virtio device entry.
  - `RPMESSAGE_RSC_VIRTIO_ID_RPMSG`: Indicates this device is for RPMsg (remote processor messaging).

- The following fields specify features, status, number of vrings (2), and other configuration details.
- `2U`: Number of vrings (ring buffers for message passing, one for TX, one for RX)[2].

5. Vring Entries

```
{ RPMESSAGE_RSC_VRING_ADDR_ANY, 4096U, 256U, 1U, 0U },
{ RPMESSAGE_RSC_VRING_ADDR_ANY, 4096U, 256U, 2U, 0U },
```

- These two entries define the **vrings** (circular buffers used for message passing)[2]:
  - `RPMESSAGE_RSC_VRING_ADDR_ANY`: The address for the vring is assigned by the host (Linux).
  - `4096U`: Alignment of the vring in memory (must be 4096 bytes for Linux compatibility).
  - `256U`: Number of buffers in the vring.
  - `1U` and `2U`: Vring indices (one for TX, one for RX).
  - `0U`: Reserved.

6. Trace Buffer Entry

```
{
    (RPMESSAGE_RSC_TRACE_INTS_VER0 | RPMESSAGE_RSC_TYPE_TRACE),
    (uint32_t)gDebugMemLog, DebugP_MEM_LOG_SIZE,
    0, "trace:m4fss0_0",
},
```

- This entry describes a **trace buffer** used for debugging:
  - `(RPMESSAGE_RSC_TRACE_INTS_VER0 | RPMESSAGE_RSC_TYPE_TRACE)`: Indicates this is a trace buffer entry with a specific version.
  - `(uint32_t)gDebugMemLog`: The address of the debug log buffer in memory.
  - `DebugP_MEM_LOG_SIZE`: The size of the debug log buffer.
  - `0`: Reserved.
  - `"trace:m4fss0_0"`: A name for the trace buffer (helpful for Linux to identify it).

Summary Table

Section	Purpose
Header	Version, number of entries, reserved
Offsets Array	Where each resource entry starts in the structure
vdev Entry	Declares the RPMsg virtio device for messaging
Vring Entries (2)	Define the two ring buffers for message passing

Section	Purpose
Trace Buffer Entry	Declares a memory buffer for debug trace logs

Why Is This Needed?

- **Linux remoteproc** reads this table to know what resources the remote firmware needs: shared memory for messaging, trace buffers, etc.[1] [2]
- **Virtio device and vrings** are required for RPMsg, which is the standard protocol for inter-processor messaging on embedded Linux systems[2] [3].
- **Trace buffer** allows Linux to collect debug information from the remote processor.

Key Points for Beginners

- The resource table is a special structure placed in a known memory section.
- It is required for Linux to set up communication and debugging with a remote microcontroller.
- Each part of the table (header, offsets, entries) has a specific purpose and format.
- The entries declare what features (like messaging and debug logging) the remote firmware supports and how they are configured.

If you need a further breakdown of any entry or want to see how this links to the actual messaging code, let me know!

⁂

1. https://e2e.ti.com/support/microcontrollers/arm-based-microcontrollers-group/arm-based-microcontrollers/f/arm-based-microcontrollers-forum/1286480/mcu-plus-sdk-am243x-does-ipcnotify-have-a-problem-when-messages-are-sent-from-both-sides-at-once
2. https://openamp.readthedocs.io/en/latest/protocol_details/resource_tbl.html
3. https://software-dl.ti.com/mcu-plus-sdk/esd/AM62X/09_00_00_19/exports/docs/api_guide_am62x/DRIVERS_IPC_RPMESSAGE_PAGE.html