THE LINUX FOUNDATION
OPEN SOURCE SUMMIT NORTH AMERICA

“Early Boot & Late Attach” of Remote Processors for Time Critical Applications

2025-06-24, Denver

Beleswar Prasad Padhi

Vaishnav Achath

About us: TI Processors and Open source

Decades of contribution and collaboration

Ingrained culture to give back to the community

Upstream FIRST!

Focus on long term, sustainable and quality products

Upstream and opensource ecosystem in device architecture

Upstream FIRST mentality!

Introduction to the Speakers

Beleswar Prasad Padhi, *Software Engineer at Texas Instruments.* Beleswar primarily works on Upstream Linux Kernel and U-Boot, as part of the Linux Core Product Development team for Jacinto Processors. Beleswar is a Free and Open Source Software Enthusiast and has contributed to several Open Source security projects like Metasploit and Mandiant as part of the Google Summer of Code program.

Vaishnav Achath, *Software Engineer at Texas Instruments India.* Vaishnav primarily works on Linux Kernel and U-Boot as part of the Texas Instruments Linux development team. Vaishnav is also a maintainer for TI platforms in Zephyr RTOS.

Texas Instruments

Overview

- Background
- Problem Statement
- Introduction to IPC
  - Resource Table
  - Data Flow
- Early Boot & Late Attach
  - Framework support
  - Challenges in this model
  - Migrating to this flow
- Debugging Lessons

Texas Instruments

Why so many CPUs?

- Maximizes efficiency of the SoC
- Some CPUs are designed to handle specific use cases:
  - ARM Cortex A: Application Processors (HLOS etc.)
  - ARM Cortex R: Realtime applications & safety critical
  - ARM Cortex M: Dedicated MCUs, multimedia
  - DSPs: Complex Image/Video Processing
    (FPAs, MMA accelerators)
- Offload to remote processor for specific use case
- Setting up IPC is important

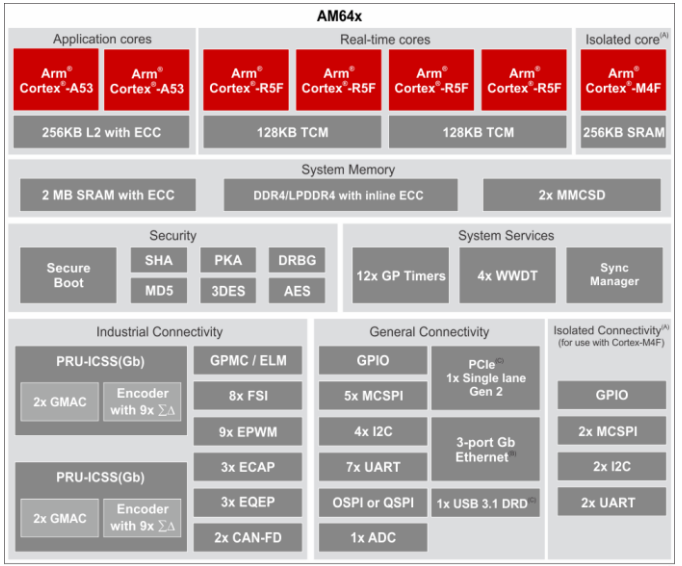


Fig. 1: TI AM64x Heterogeneous SoCs

Texas Instruments

Background

- Typically when Linux comes up:
  - Identifies available remoteprocs in the platform
  - Boots them with firmware if available

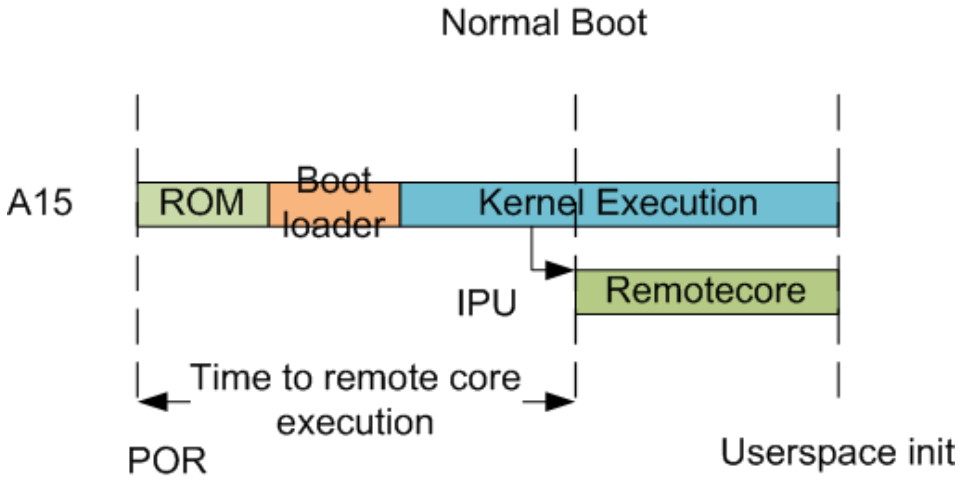


Fig. 2: Normal remoteproc boot flow

Texas Instruments

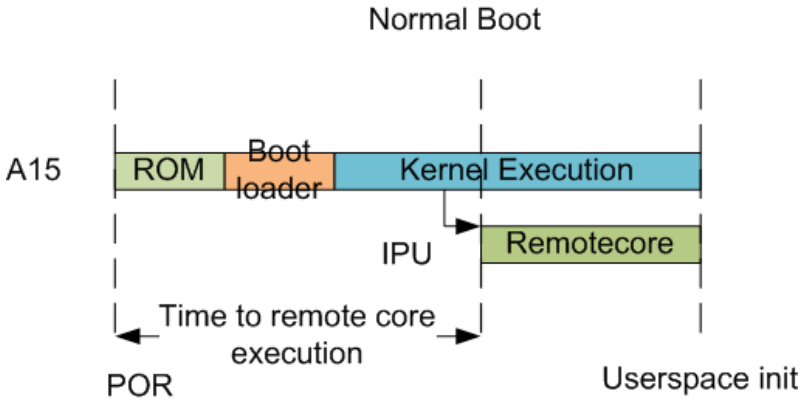
Problem Statement

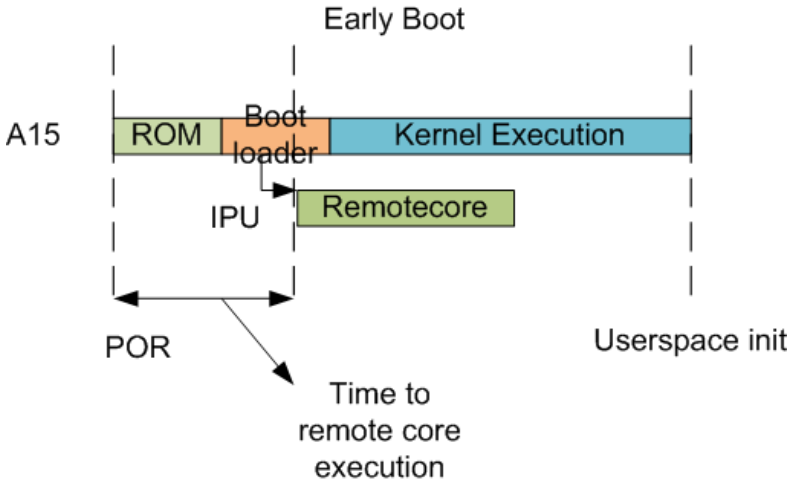
- Need of a fast boot of real-time MCUs in SoCs
- Automotive requirements are:
  - **Time-critical**: Require early chime, early display to come up in milliseconds
  - **Safety-oriented**: Need to be run on a safety MCUs like ARM Cortex R5F
  - Ex: Early Audio Chime, Early Display, Early Ethernet, Early Rear-View Camera etc.
- Typical Linux boot = 1 second(s)
- Requirement for early audio chime in 200ms
- Can’t wait for Linux to complete booting and then boot other rprocs

Texas Instruments

Solution

- “**Early Boot & Late Attach**”
  - Boot the remoteproc early in bootloader stage
  - Attach into remoteproc late in Kernel stage to setup IPC
  - Both stages are equally important





Texas Instruments

Introduction to IPC

- Typical flow to boot a rproc from host processor:
  - Load the rproc firmware
  - Allocate resources required by rproc
  - Start the remote processor
- So many OSes: Linux, RTOS, QNX etc.
- Need for a standard: **Resource Table**
  - Common standard to define the resources required by a rproc
  - struct resource_table in `include/linux/remoteproc.h`

Fig. 3: Resource table implementation

Texas Instruments

Introduction to IPC | Resources

- Resource table currently supports 4 types of resources:

```
struct fw_rsc_carveout {
        u32 da;
        u32 pa;
        u32 len;
        u32 flags;
        u32 reserved;
        u8 name[32];
} __packed;
```

Carveouts

```
struct fw_rsc_vdev {
        u32 id;
        u32 notifyid;
        u32 dfeatures;
        u32 gfeatures;
        u32 config_len;
        u8 status;
        u8 num_of_vrings;
        u8 reserved[2];
        struct fw_rsc_vdev_vring
} __packed;
```

Virtio device

```
struct fw_rsc_devmem {
        u32 da;
        u32 pa;
        u32 len;
        u32 flags;
        u32 reserved;
        u8 name[32];
} __packed;
```

Mappings

```
struct fw_rsc_trace {
        u32 da;
        u32 len;
        u32 reserved;
        u8 name[32];
} __packed;
```

Trace buffer

Texas Instruments

Introduction to IPC | Notification

- While allocating RSC_VDEV resource, Linux:
  - Writes status to resource table in shared memory as per progress
  - VIRTIO_CONFIG_S_ACKNOWLEDGE (0x01)
  - VIRTIO_CONFIG_S_DRIVER (0x02)
  - VIRTIO_CONFIG_S_DRIVER_OK (0x04)
  - Finally sends a mailbox notification to rproc saying resources are allocated
- Firmware can then announce the name service channel

```
struct fw_rsc_vdev {
        u32 id;
        u32 notifyid;
        u32 dfeatures;
        u32 gfeatures;
        u32 config_len;
        u8 status;
        u8 num_of_vrings;
        u8 reserved[2];
        struct fw_rsc_vdev_vring
} __packed;
```

Fig. 4: status field in RSC_VDEV

Texas Instruments

Introduction to IPC | Data Flow

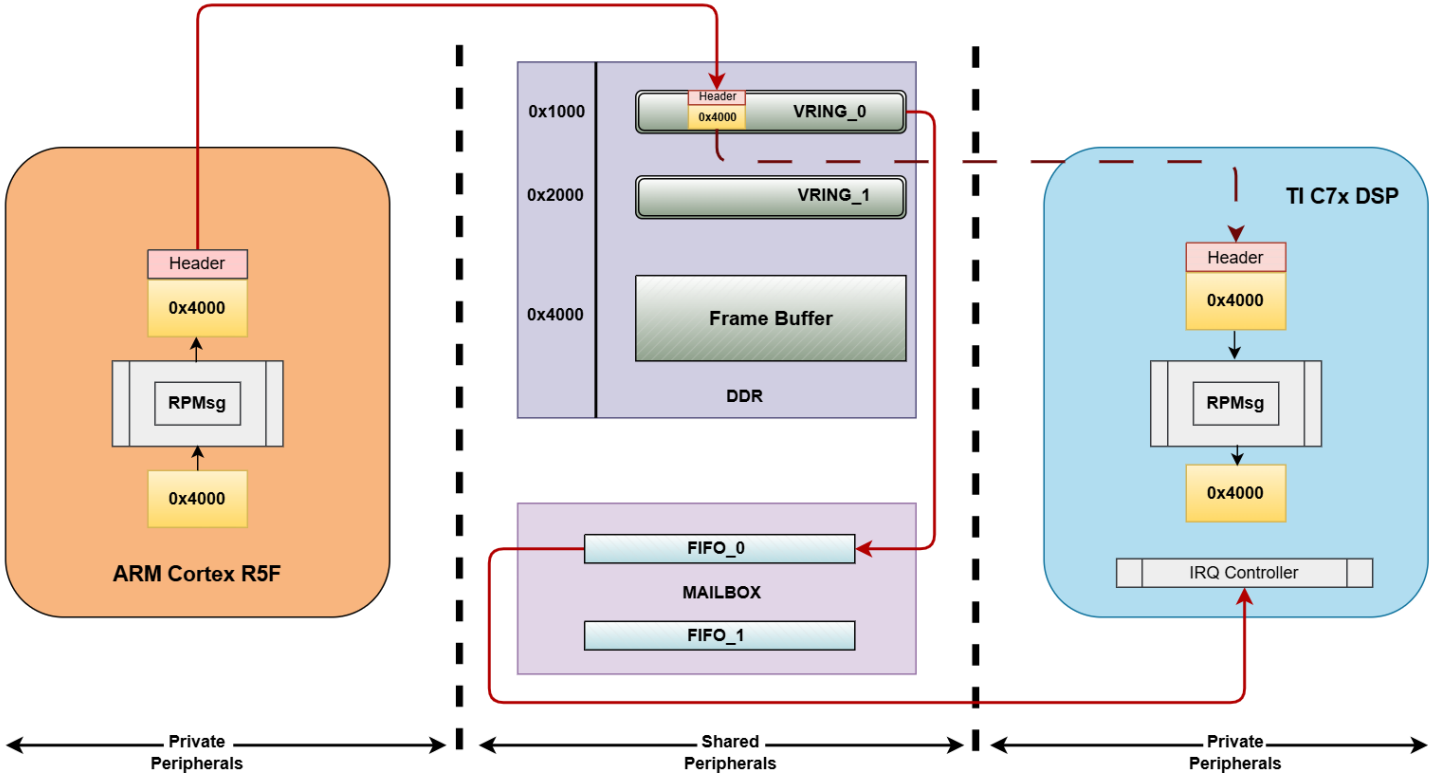


Fig. 5: Flowchart showing Data flow from R5F to DSP

Texas Instruments

Introduction to IPC | Data Flow

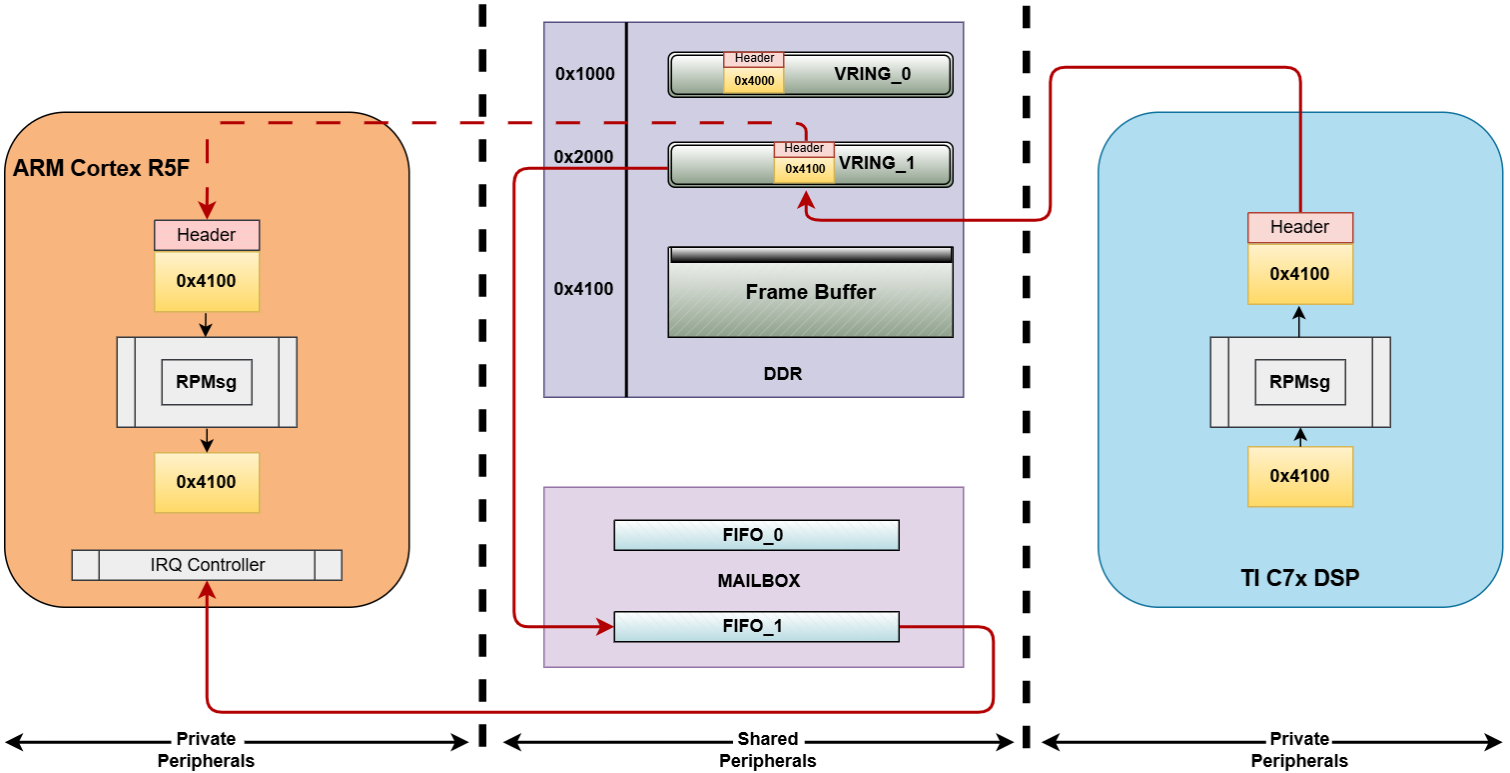


Fig. 6: Flowchart showing Data flow from DSP to R5F

Texas Instruments

Early Boot & Late Attach | Early Boot

- Boot the rproc (& dependent peripherals) at bootloader stage:
  - Have a driver for rproc in bootloader/ROM
  - Read the rproc firmware from boot media
  - Allocate resource table carveouts
  - Handle resource table mappings
    - Map PA to DA for rproc's IOMMU
    - Store page tables in reserved memory carveout
    - Configure IOMMU's TTB register to point to above carveout
  - Load the ELF segments
  - Start the timer
  - Start the remote processor
- Typical bootloaders like U-Boot have a remoteproc framework to handle lot of these functionalities.

Texas Instruments

Early Boot & Late Attach | Early Boot

- Bootloader takes care of everything needed to use the remote processor as a standalone entity.
- It does not do any initializations for IPC yet.
- The remoteproc can start doing its work now parallel to HLOS boot:
  - Displaying frame buffers for an early splash screen
  - Early audio chime
  - Capturing frames etc.

Texas Instruments

Early Boot & Late Attach | Late Attach

- Kernel needs to be smart
- Can’t blindly reset the peripheral like in normal flow
- Identify peripheral state; program accordingly
- Late attach support spanned across multiple frameworks:
  - remoteproc
  - iommu
  - timers
  - Any other peripheral that is used by the platform.

Texas Instruments

Early Boot & Late Attach | Late Attach

- **Remoteproc framework:**
- Generic framework to control a remote processor:
  - Allocating firmware resources
  - Loading/Parsing ELF firmware
  - Offers ops for platform specific code:
    - Prepare() = preparing the device before firmware loading
    - Start() = release the rproc from reset & run the rproc
    - Stop() = reset the rproc & halt the rproc
    - Unprepare() = unprepared the device for state management
    - Attach() = attach to the rproc (acquire mbox channels, pwrdomains)
    - Detach() = detach from the rproc

Texas Instruments

Early Boot & Late Attach | Late Attach

- **Remoteproc framework:**
  - Expects implementations to set the rproc state
  - If rproc is powered off, calls rproc_fw_boot()
  - If rproc is pre-running, calls rproc_attach()
  - **rproc_fw_boot():**
    - Reads firmware off disk
    - Parses resource table
    - Allocates resources
    - Loads firmware to rproc internal memories/carveouts
    - Starts the remoteproc
  - **rproc_attach():**
    - Expects pointer to pre-loaded resource table
    - Allocates resources
    - Registers the virtio device

```
if (rproc->state == RPROC_DETACHED) {
        dev_info(dev, "attaching to %s\n", rproc->name);
        ret = rproc_attach(rproc);
} else {
        dev_info(dev, "powering up %s\n", rproc->name);
        /* load firmware */
        ret = request_firmware(&firmware_p,
                               rproc->firmware, dev);

        ret = rproc_fw_boot(rproc, firmware_p);

        release_firmware(firmware_p);
}
```

Fig. 7: rproc_boot() in framework

Texas Instruments

Early Boot & Late Attach | Late Attach

- Challenge #1: **How to identify rproc state?**
- Different vendors have different ways:
  - **NXP iMX** -> Hardcodes a set of allowed boot methods in driver
    - » IMX_RPROC_NONE = processor reset is not handled at Linux, attach
    - » IMX_RPROC_RESET_CONTROLLER = handled, rproc_fw_boot
  - **STM32** -> Reads a DT property (st,syscfg-m4-state) which in turn reads a register for core state
  - **TI** -> Read the LPSC register of the core
  - **XLNX** -> Checks if a valid RSC table (version number) is available at base address of memory region

Early Boot & Late Attach | Late Attach

- Challenge #2: **How to get the pre-loaded rsc table?**
- Idea: Use some sort of stored info from bootloader,
  or find it in a hardcoded addr
- Different vendors have different methodologies:
  - **NXP iMX** -> Reserved memory base contains the rsc table
  - **STM32** -> Reads a DT property (st,syscfg-rsc-tbl)
    which inturn reads a register for core state
    (has to be written by their bootloader)
  - **TI** -> Reserved Memory base contains the rsc table
  - **XLNX** -> Reserved Memory base contains pointer to rsc table

Texas Instruments

Early Boot & Late Attach | Late Attach

- Challenge #3: **How to establish suspend/resume?**
- Framework supports shutting down both early booted & normal booted rproc
- Vendors have to decide if S/R is supported in their arch for Early Booted rproc
- If supported, suspend:
  - Save rproc context
  - get pm domains in .attach() hook
  - Notify rproc to relinquish its resources
- Resume:
  - Load rproc with cached firmware from DDR
  - Boot via rproc_fw_boot() [No more attached]
  - Restore context

```
/**
* rproc_shutdown() - power off the remote processor
* @rproc: the remote processor
*
* Power off a remote processor (previously booted with rproc_boot()).
*
* In case @rproc is still being used by an additional user(s), then
* this function will just decrement the power refcount and exit,
* without really powering off the device.
*
* Every call to rproc_boot() must (eventually) be accompanied by a call
* to rproc_shutdown(). Calling rproc_shutdown() redundantly is a bug.
*
* Notes:
* - we're not decrementing the rproc's refcount, only the power refcount.
*   which means that the @rproc handle stays valid even after rproc_shutdown()
*   returns, and users can still use it with a subsequent rproc_boot(), if
*   needed.
*
* Return: 0 on success, and an appropriate error value otherwise
*/
int rproc_shutdown(struct rproc *rproc)
```

Fig. 8: rproc_shutdown() in framework

Texas Instruments

Early Boot & Late Attach | Late Attach

- **IOMMU framework:**
- No changes at framework level
  - **iommu_paging_domain_alloc()**
  - **iommu_attach_device()**
- Implementations have to be intelligent to identify late-attach in .attach_dev() hook:
  - Read IOMMU state: PSC configs
  - If ON, fetch pre-programmed page tables from TTB register
  - If OFF, allocate a page table offset for mappings
  - Note: translation mapping sizes can be different when you fetch pre-programmed tables

```
static int omap_iommu_attach(struct omap_iommu
*obj,                        u32 *iopgd)
 {
    int err;
    u32 iopgd_pa;

    if (obj->late_attach) {
        iopgd_pa = iommu_read_reg(obj, MMU_TTB);
        iopgd = ioremap(iopgd_pa,
                        EARLY_PAGE_TABLES_SIZE);
        if (!iopgd)
            return -ENOMEM;
    } else {
        iopgd_pa = virt_to_phys(iopgd);
    }
```

Fig. 9: attach_dev hook in omap-iommu.c

Texas Instruments

Early Boot & Late Attach | Late Attach

- **Timer:**
- Can’t afford to reset timer in late attach
  - Used as systicks in RTOS
- Identify if the timer is pre-configured and attach gracefully
- No changes at framework level
- Implementations have to be intelligent:
  - Identify state by reading Counter register
  - If late attach, prevent configuring (set_source/set_load)

Texas Instruments

Early Boot & Late Attach | Migrate to Late Attach

- **Bootloader:**
  - Write drivers to start all peripherals and remoteproc
  - Save important configurations into peripherals registers
  - Pass on reserved memory into Kernel as part of DT fixups
- **Kernel:**
  - Implement attach/detach() ops for remoteproc framework
  - Identify peripherals existing state, and attach gracefully
  - Don’t force configure/reset any peripheral!

Texas Instruments

Debugging Lessons

- This section shares findings around everything that could go wrong while migrating to late attach flow:
  - **Selecting memory carveout addresses**
  - **Requesting carveouts via RSC_TABLE**
  - **Gracefully attaching to peripherals**
- Ref: Ported Late Attach patches from 4.19 to 6.1 for TI OMAP5 platform (AM57x evm)
- Platform overview:
  - ARM32 bit platform
  - Cortex A15 as general purpose processor
  - Cortex M4 rproc as Image Processing Unit (IPU)
  - TI C66x rproc as Digital Signal Processor (DSP)
  - Had 2 IOMMUs in each rproc subsystem

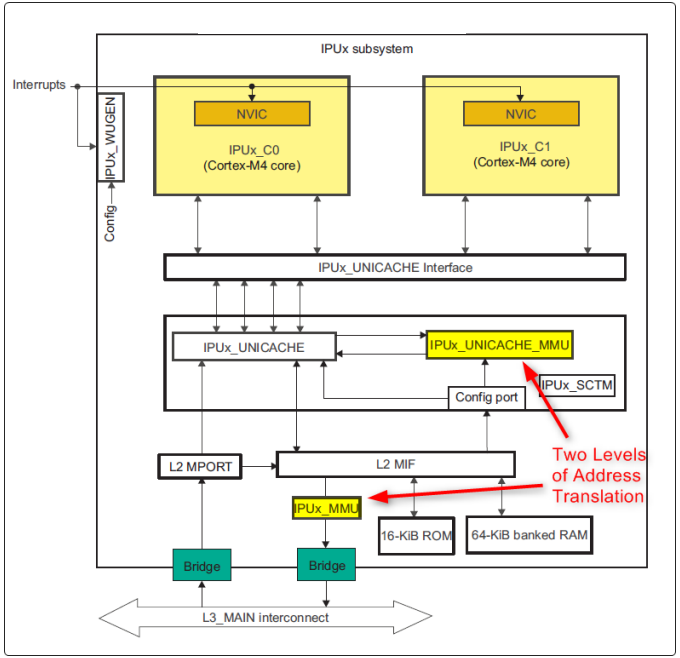


Fig. 10: IPU subsystem in TI OMAP AM57x

Texas Instruments

Debugging Lessons

- Problem #1: **Linux not booting with reserved memory**
- IOMMU Page tables carveout = 0x80800000
- Carveout overlapped with Linux decompressed image!
- Lesson: **Be careful with placing carveouts**… Updated carveout = 0x95700000

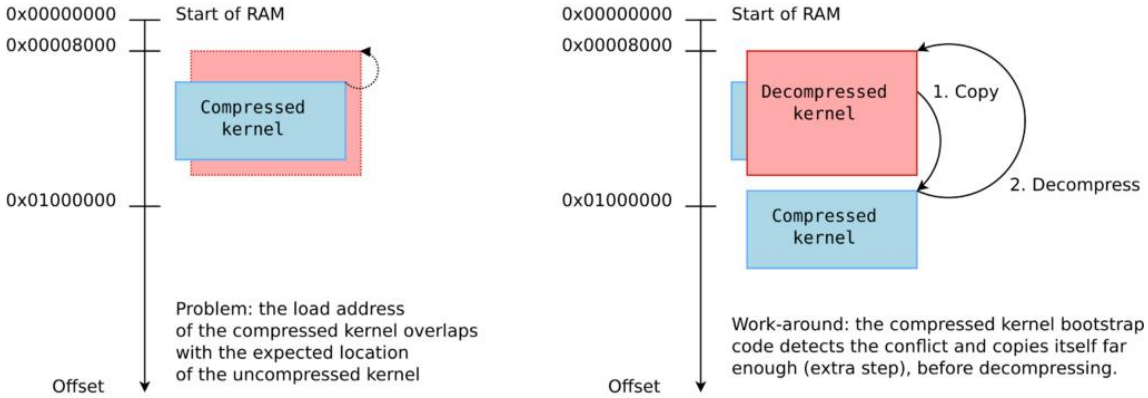


Fig. 11: Linux Boot flow on ARM32 platforms

Texas Instruments

Debugging Lessons

- Problem #2: **Not able to retrieve page tables**
- IOMMU TTB in:
  - U-Boot prompt => reads 0x95700000
  - Linux entry point => reads 0x95700000
  - Attach() in iommu linux driver => reads 0
- => IOMMU is getting reset somewhere between Kernel entry point and IOMMU probe.
- Put memory watchpoint on TTB.
- Culprit: Platform specific code programs all IPs into a known configuration state (OFF)
- Lesson: **Late-Attach needs to be supported at a system level & not just from remoteproc framework**

```
static int __init pwrdms_setup(struct powerdomain *pwrdm)
{
  pwrst->pwrdm = pwrdm;

  pwrst->next_state = pwrdm_get_valid_lp_state(pwrdm, false,
                                               PWRDM_POWER_RET);

  pwrst->next_logic_state = pwrdm_get_valid_lp_state(pwrdm, true,
                                                     PWRDM_POWER_OFF);

  return omap_set_pwrdm_state(pwrst->pwrdm, pwrst->next_state);
}
```

Fig. 12: pwrdm init for omap4 platform

```
[    0.604400] Call trace:
[    0.604492]  omap_set_pwrdm_state from pwrdm_for_each+0x34/0x5c
[    0.604522]  pwrdm_for_each from omap4_pm_init+0x5c/0x164
[    0.604553]  omap4_pm_init from __omap2_common_pm_late_init+0x3c/0x64
[    0.604553]  __omap2_common_pm_late_init from do_one_initcall+0x58
[    0.604583]  do_one_initcall from kernel_init_freeable+0x1d0/0x22c
[    0.604614]  kernel_init_freeable from kernel_init+0x1c/0x12c
[    0.604644]  kernel_init from ret_from_fork+0x14/0x38
```

Fig. 13: Call trace of omap4_pm_init

Texas Instruments

Debugging Lessons

- Problem #3: **Firmware code stuck in a busy loop**
- All resources are now allocated by Linux, so firmware should initiate RPMSG name service channel.
- Through debugger, turns out it is still busy looping on the status byte to be set to 0x07 in the resource table.
- Verified no cache related issues
- Checked IOMMU page tables carveout
- Carveout was zero-ed out leading to incorrect address translations

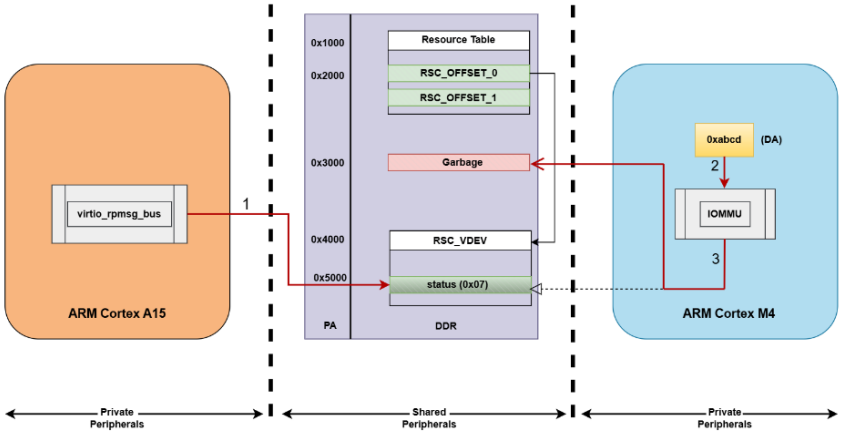


Fig. 14: rproc reading garbage value due to incorrect address translations

Texas Instruments

Debugging Lessons

- Problem #3: **Firmware code stuck in a busy loop**
- Again, put a memory watchpoint
- Culprit: dma_alloc_coherent() clears out allocated memories
  invoked from rproc framework while parsing RSC_CARVEOUT
- Introduced a non-zeroing flag which was passed to the dma_alloc_coherent() function to prevent this
- Lesson: **Ensure your carveouts are not cleared!**
- **Note**: Ongoing discussion about handling carveout resources in Late Attach mode.

```c
/**
 * rproc_alloc_carveout() - allocated specified carveout
 */
static int rproc_alloc_carveout(struct rproc *rproc,
                                struct rproc_mem_entry *mem)
{
  va = dma_alloc_coherent(dev->parent, mem->len, &dma, GFP_KERNEL);
  if (!va) {
    dev_err(dev->parent, "failed to allocate dma memory: len 0x%zx\n",
            mem->len);
    return -ENOMEM;
  }
}
```

Fig. 15: rproc core allocating carveout via dma_alloc_coherent()

```c
static void *__alloc_from_contiguous(struct device *dev, size_t size,
                                     pgprot_t prot, struct page **ret_page,
                                     const void *caller, bool want_vaddr,
                                     int coherent_flag, gfp_t gfp)
{
  unsigned long order = get_order(size);
  size_t count = size >> PAGE_SHIFT;

  page = dma_alloc_from_contiguous(dev, count, order, gfp & __GFP_NOWARN);
  if (!page)
    return NULL;

  __dma_clear_buffer(page, size, coherent_flag);
}
```

Fig. 16: dma_alloc_coherent() clearing allocated buffer

Texas Instruments

Debugging Lessons

- Problem #4: **Core unresponsive to IPC**
- Sent a ping from A15 core, no response from IPU
- Could not connect to IPU from debugger
- Verified IPC stack:
  - Mailbox
    - Verified mbox was generating irq upon message write
    - Verified rproc registered to correct mbox queue
  - Virtio
    - Checked virtio tx buffer was filled upon message write
    - But virtio rx buffer was empty

```
Void ti_sysbios_BIOS_startFunc__I(Void)
{

  (void)xdc_runtime_System_atexit(
    (xdc_runtime_System_AtexitHandler)ti_sysbios_BIOS_atExitFunc__I);
  IpcMgr_ipcStartup();
  ti_sysbios_BIOS_registerRTSLock();
  ti_sysbios_timers_dmtimer_Timer_startup__E();
  ti_sysbios_hal_Hwi_startup();
  ti_sysbios_knl_Swi_startup();
  xdc_runtime_System_printf("DEBUG: Before Starting Kernel Task\n");
  ti_sysbios_knl_Task_startup();
  xdc_runtime_System_printf("DEBUG: After Starting Kernel Task\n");
}
```

Fig. 17: RTOS init routine code flow

Texas Instruments

Debugging Lessons

- Problem #4: **Core unresponsive to IPC**
- Came back to 2 fundamental errors: **Power & Clock**
- Removed all reset & clock properties from DT node
- SYSFS:
  - No IPU pwrdomain nodes in sysfs
  - But one clkdomain node spotted
- Culprit: Platform clk driver hardcoded an Alias clock config node for IPU
- This was messing up clocks at runtime leading to core getting crashed.
- Lesson: **Rely on runtime system properties only!**
- Better programming practice: Put things at one place for one IP block

```
root@am57xx-evm:~# cat /sys/kernel/debug/pm_debug/
cam_pwrdm/       dsp1_pwrdm/      eve3_pwrdm/      rtc_pwrdm/
core_pwrdm/      dsp2_pwrdm/      eve4_pwrdm/      time
coreaon_pwrdm/   dss_pwrdm/       gpu_pwrdm/       vpe_pwrdm/
count            emu_pwrdm/       iva_pwrdm/       wkupaon_pwrdm/
cpu0_pwrdm/      enable_off_mode  l3init_pwrdm/
cpu1_pwrdm/      eve1_pwrdm/      l4per_pwrdm/
custefuse_pwrdm/ eve2_pwrdm/      mpu_pwrdm/

root@am57xx-evm:~# cat /sys/kernel/debug/clk/ipu
ipu-clkctrl:0000:0/    ipu-clkctrl:0010:0/    ipu-clkctrl:0028:0/
ipu-clkctrl:0000:22/   ipu-clkctrl:0010:24/   ipu-clkctrl:0030:0/
ipu-clkctrl:0000:24/   ipu-clkctrl:0018:0/    ipu-clkctrl:0030:24/
ipu-clkctrl:0000:28/   ipu-clkctrl:0018:24/   ipu1-clkctrl:0000:0/
ipu-clkctrl:0008:0/    ipu-clkctrl:0020:0/    ipu1-clkctrl:0000:24/
ipu-clkctrl:0008:24/   ipu-clkctrl:0020:24/   ipu2-clkctrl:0000:0/
```

Fig. 18: pwrdm and clkdm nodes in sysfs

```
43 static const struct omap_clkctrl_reg_data dra7_ipu1_clkctrl_regs[] __initconst = {
44         { DRA7_IPU1_MMU_IPU1_CLKCTRL, dra7_mmu_ipu1_bit_data,
             CLKF_HW_SUP | CLKF_NO_IDLEST, "ipu1-clkctrl:0000:24" },
45         { 0 },
46 };
```

Fig. 19: Alias DT clock node definitions

```
int __init dra7xx_dt_clk_init(void)
{
  int rc;
  struct clk *dpll_ck, *hdcp_ck;

  ti_dt_clocks_register(dra7xx_clks);

  omap2_clk_disable_autoidle_all();

  ti_clk_add_aliases();
}
```

Fig. 20: Alias clk init from drivers/clk/ti/clk-7xx.c

Texas Instruments

References

- Asymmetric Multiprocessing and Embedded Linux - Marek Novak & Dušan Červenka, NXP Semiconductor - ELCE 2017
- Boot time: choose your kernel loading address carefully
- TI OMAP AM57x Product Details - AM5729 data sheet, product information and support | TI.com

Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.
- Andrew Davis <afd@ti.com>
- Udit Kumar <u-kumar1@ti.com>
- Vignesh Raghavendra <vigneshr@ti.com>

Texas Instruments

Q&A

- Contact Information:
  - Beleswar Padhi <b-padhi@ti.com>
  - Vaishnav Achath <vaishnav.a@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.