ARM DMA-Mapping Framework Redesign and IOMMU integration

Marek Szyprowski, Kyungmin Park
Samsung Poland R&D Center, Samsung Electronics Corp.

m.szyprowski@samsung.com
kyungmin.park@samsung.com

Embedded Linux Conference Europe
Prague, 27 October 2011
Our goal

• To provide support for multimedia hardware available on latest Samsung SoCs (like ARM based S5PV210 and Exynos4)
• Multimedia devices: Video codec, camera interface, hdmi display interface, others
• Common requirement: large, physically contiguous memory buffers
Contiguous buffers

• Custom framework that reserves memory during system boot and then dynamically serves it to the device drivers
• Almost each hardware vendor provides its own solution: CMEM, PMEM, HWMEM, ... 
• We developed our own allocator – CMA.
IOMMU hardware

• Solves physical fragmentation issue
• Can map any physical memory pages into device virtual address space
• Increases security and reliability
• Requires additional driver and integration
Custom solutions – summary

• It was possible to get a working system in a short period of time

• No chance to get the drivers accepted in mainline kernel
  – A lot of maintenance works with each release
  – Hard to discuss any extensions to other kernel frameworks (like V4L2) if the drivers won’t be merged

• Advice: try to understand, reuse and extend the existing frameworks
DMA-mapping framework

• Common, kernel-wide, hardware independent framework for allocating and mapping buffers into DMA (device IO) address space

• Most popular functions:
  – dma_{alloc, mmap, free}_coherent()
  – dma_{map,unmap}_\{page, single, sg\}()

• How does it fit into our requirements?
ARM implementation – issues

• Allocation of physically contiguous memory is not reliable
  – relies on alloc_pages()
  – usually succeeds only on system boot
  – dynamic allocation is not really possible
  – limited size of a single buffer
  – memblock_reserve() + dma_declare_coherent() workaround means memory waste
Contiguous Memory Allocator (I)

• Provides a functionality of allocating big chunks of physically contiguous memory

• No limitation or restriction on the size of a single chunk

• Buffers can be allocated anytime when the system is running providing the enough memory is available in the system
Contiguous Memory Allocator (II)

• On system boot a specified memory region is being reserved

• CMA gives back reserved regions to the system memory pool but only for ‘movable’ memory pages
  – On the allocation request memory migration framework migrates these pages to other memory areas freeing physically contiguous chunk
  – ‘movable’ pages consist mainly of anonymous memory and page cache (file system buffers)
CMA and DMA-mapping (I)

• CMA provides dma_alloc_from_contiguous() call which can replace alloc_pages()
  – this was not enough to solve all ARM related issues...

• Other issues:
  – Aliasing between ‘coherent’ and cacheable low-memory mappings
  – GFP_ATOMIC allocations (migration requires sleeping)
ARM implementation – issues (II)

• Three different implementations merged together (arch/arm/mm/dma-mapping.c)
  – linear non-coherent (most systems)
  – linear coherent (noMMU and Intel ixp23xx)
  – ‘bounced’ for systems with restricted or limited/broken DMA engines

• Hard to have different implementations for different devices in the system

• Hard to add more implementations
DMA-Mapping patches (I)

• Our answer for the limitation of the current implementation of DMA-mapping

• Goal: to provide per-device implementation of DMA mapping methods and create generic, hardware independent IOMMU capable mapper
DMA-Mapping patches (II)

• Introduce ‘struct dma_map_ops’ style implementation like on other architectures
  – Easy to set methods on per-device basis
  – Simplify code (no more #ifdef and if() spaghetti)
• Separated ‘dma bounce’ implementation from the rest of the code
• Introduce dma_alloc_attrs() as a replacement for dma_alloc_coherent() and dma_alloc_writecombine()
DMA-Mapping patches (III)

• Introduce experimental IOMMU capable implementation
  – Use generic, hardware independent IOMMU API
  – Finally implemented all dma operations
  – Hide the fact that IOMMU is used from the client devices

• Tested on Samsung Exynos4 hardware with V4L2 multimedia devices and nVidia Tegra
Contiguous Memory Allocator patches

• Version 16 has been posted on 6 October 2011: http://lwn.net/Articles/461849/

• Experimental support for x86 DMA-mapping
• Complete support for ARM DMA-mapping integration

• Tested on Samsung SoCs, TI OMAP and other hardware
Summary

• We manage to get a working solution without the need of any custom frameworks, hiding as much as possible behind existing API
• The drivers call only a standard Linux kernel API
• The drivers use the same calls on systems with IOMMU (like Samsung Exynos4) and without (like Samsung S5PV210).
• Lessons learnt: double check the existing kernel API before introducing anything new