in mm/memblock.c

Use global variable memblock to track all available physical memory and all reserved physical memory.

```
struct memblock memblock __initdata_memblock = {
        .memory.regions         = memblock_memory_init_regions,
        .memory.cnt             = 1,  /* empty dummy entry */
        .memory.max             = INIT_MEMBLOCK_REGIONS,

        .reserved.regions       = memblock_reserved_init_regions,
        .reserved.cnt           = 1,  /* empty dummy entry */
        .reserved.max           = INIT_MEMBLOCK_REGIONS,

#ifdef CONFIG_HAVE_MEMBLOCK_PHYS_MAP
        .physmem.regions  = memblock_physmem_init_regions,
        .physmem.cnt            = 1,  /* empty dummy entry */
        .physmem.max            = INIT_PHYSMEM_REGIONS,
#endif

        .bottom_up              = false,
        .current_limit          = MEMBLOCK_ALLOC_ANYWHERE,
};
```

CONFIG_HAVE_MEMBLOCK_PHYS_MAP is not set in Granite2 and Gemstone2 LSP.

```
struct memblock {
        bool bottom_up;  /* is bottom up direction? */
        phys_addr_t current_limit;
```

```
    struct memblock_type memory;      ①
    struct memblock_type reserved;    ②
#ifdef CONFIG_HAVE_MEMBLOCK_PHYS_MAP
    struct memblock_type physmem;
#endif
};
```

①

record all available physical memory in it (array)

②

record all reserved physical memory in it (array)

```
struct memblock_type {
    unsigned long cnt;   /* number of regions */
    unsigned long max;/* size of the allocated array */
    phys_addr_t total_size;   /* size of all regions */
    struct memblock_region *regions;
};

struct memblock_region {
    phys_addr_t base;
    phys_addr_t size;
    unsigned long flags;
#ifdef CONFIG_HAVE_MEMBLOCK_NODE_MAP
    int nid;
#endif
};
```

static struct memblock_region **memblock_memory_init_regions**[INIT_MEMBLOCK_REGIONS] __initdata_memblock;

static struct memblock_region **memblock_reserved_init_regions**[INIT_MEMBLOCK_REGIONS] __initdata_memblock;

#define INIT_MEMBLOCK_REGIONS 128

即无论是"memory"的memblock_type还是"reserved"的memblock_type,最多可以有128个region。

global variable memblock.memory.regions = **memblock_memory_init_regions**;

global variable memblock.reserved.regions = **memblock_reserved_init_regions**;

==========

in include/linux/memblock.h

int memblock_add(phys_addr_t base, phys_addr_t size); ①

int memblock_reserve(phys_addr_t base, phys_addr_t size); ②

①

把[base, base + size)的region添加到"memory"的memblock_type(memblock.memory.regions)中去

"memory"的memblock_type中的region是Linux可以使用的total memory(包括了下面"reserved"的memblock_type中的memory)

②

把[base, base + size)的region添加到"reserved"的memblock_type(memblock.reserved.regions)中去

"reserved"的memblock_type中的region则是已经被占用，不能被分配出去的memory

==========

About memblock_add()

memblock_add()用于告知Linux可用的memory。目前有两种方式向Linux kernel报告可用memory.

1. atag方式，即u-boot通过ATAG_MEM这个tag来告知kernel当前系统有那些memory(通过ATAG_MEM)。这是比较老的方式，使用device tree以后就不适用了。

2. 在dts中指定memory。这是device tree的方式。

比如在mv6270-toc.dts中

```
memory {
    device_type = "memory";
    reg = <0 0x00000000 0 0x40000000>; /* 1 GB */
};
```

该device node就用于指定系统的memory。

in arch/arm/kernel/setup.c

```
void __init setup_arch(char **cmdline_p)
{
    const struct machine_desc *mdesc;

    setup_processor();
    mdesc = setup_machine_fdt(__atags_pointer);
    if (!mdesc)            ( A )
```

```
        mdesc = setup_machine_tags(__atags_pointer, __machine_arch_type);
    machine_desc = mdesc;
    machine_name = mdesc->name;

    .....
}
```

（A）

如果mdesc为NULL，表示kernel没找到device tree对应的dtb，则寻找ATAG(老的方式)。

in arch/arm/kernel/devtree.c

```
/**
 * setup_machine_fdt - Machine setup when an dtb was passed to the kernel
 * @dt_phys: physical address of dt blob
 *
 * If a dtb was passed to the kernel in r2, then use it to choose the
 * correct machine_desc and to setup the system.
 */
const struct machine_desc * __init setup_machine_fdt(unsigned int dt_phys)
{
    const struct machine_desc *mdesc, *mdesc_best = NULL;

#ifdef CONFIG_ARCH_MULTIPLATFORM
    DT_MACHINE_START(GENERIC_DT, "Generic DT based system")
    MACHINE_END

    mdesc_best = &__mach_desc_GENERIC_DT;
#endif
```

```
if (!dt_phys || !early_init_dt_verify(phys_to_virt(dt_phys)))
        return NULL;

mdesc = of_flat_dt_match_machine(mdesc_best, arch_get_next_mach);

if (!mdesc) {
        const char *prop;
        int size;
        unsigned long dt_root;

        early_print("\nError: unrecognized/unsupported "
                    "device tree compatible list:\n[ ");

        dt_root = of_get_flat_dt_root();
        prop = of_get_flat_dt_prop(dt_root, "compatible", &size);
        while (size > 0) {
                early_print("'%s' ", prop);
                size -= strlen(prop) + 1;
                prop += strlen(prop) + 1;
        }
        early_print("]\n\n");

        dump_machine_table(); /* does not return */
}

/* We really don't want to do this, but sometimes firmware provides buggy data */
if (mdesc->dt_fixup)
        mdesc->dt_fixup();
```

```
        early_init_dt_scan_nodes();

        /* Change machine number to match the mdesc we're using */
        __machine_arch_type = mdesc->nr;

        return mdesc;
}

void __init early_init_dt_scan_nodes(void)
{
        /* Retrieve various information from the /chosen node */
        of_scan_flat_dt(early_init_dt_scan_chosen, boot_command_line);

        /* Initialize {size,address}-cells info */
        of_scan_flat_dt(early_init_dt_scan_root, NULL);

        /* Setup memory, calling early_init_dt_add_memory_arch */
        of_scan_flat_dt(early_init_dt_scan_memory, NULL);
}

/**
 * early_init_dt_scan_memory - Look for an parse memory nodes
 */
int __init early_init_dt_scan_memory(unsigned long node, const char *uname,
                                     int depth, void *data)
{
        const char *type = of_get_flat_dt_prop(node, "device_type", NULL);          ③
        const __be32 *reg, *endp;
```

```
int l;

/* We are scanning "memory" nodes only */
if (type == NULL) {
        /*
         * The longtrail doesn't have a device_type on the
         * /memory node, so look for the node called /memory@0.
         */
        if (!IS_ENABLED(CONFIG_PPC32) || depth != 1 || strcmp(uname, "memory@0") != 0)
                return 0;
} else if (strcmp(type, "memory") != 0)                                    ④
        return 0;

reg = of_get_flat_dt_prop(node, "linux,usable-memory", &l);
if (reg == NULL)
        reg = of_get_flat_dt_prop(node, "reg", &l);                        ⑤
if (reg == NULL)
        return 0;

endp = reg + (l / sizeof(__be32));

pr_debug("memory scan node %s, reg size %d, data: %x %x %x %x,\n",
    uname, l, reg[0], reg[1], reg[2], reg[3]);

while ((endp - reg) >= (dt_root_addr_cells + dt_root_size_cells)) {
        u64 base, size;

        base = dt_mem_next_cell(dt_root_addr_cells, &reg);
        size = dt_mem_next_cell(dt_root_size_cells, &reg);
```

```
        if (size == 0)
            continue;
        pr_debug(" - %llx ,  %llx\n", (unsigned long long)base,
            (unsigned long long)size);

        early_init_dt_add_memory_arch(base, size);                    ⑥
    }

    return 0;
}
```

in drivers/of/fdt.c

```
void __init __weak early_init_dt_add_memory_arch(u64 base, u64 size)
{
    const u64 phys_offset = __pa(PAGE_OFFSET);

    if (!PAGE_ALIGNED(base)) {
        size -= PAGE_SIZE - (base & ~PAGE_MASK);
        base = PAGE_ALIGN(base);
    }
    size &= PAGE_MASK;

    if (base > MAX_PHYS_ADDR) {
        pr_warning("Ignoring memory block 0x%llx - 0x%llx\n",
                base, base + size);
        return;
    }
```

```
        if (base + size - 1 > MAX_PHYS_ADDR) {
                pr_warning("Ignoring memory range 0x%llx - 0x%llx\n",
                        ((u64)MAX_PHYS_ADDR) + 1, base + size);
                size = MAX_PHYS_ADDR - base + 1;
        }

        if (base + size < phys_offset) {
                pr_warning("Ignoring memory block 0x%llx - 0x%llx\n",
                        base, base + size);
                return;
        }
        if (base < phys_offset) {
                pr_warning("Ignoring memory range 0x%llx - 0x%llx\n",
                        base, phys_offset);
                size -= phys_offset - base;
                base = phys_offset;
        }
        memblock_add(base, size);                                                    ⑦
}
```

③

在dtb中寻找带有"device_type" property的device node

④

如果在device node中找到了"device_type" property，则检查其property value是否为"memory"

这里③+④，就是为了定位

```
memory {
        device_type = "memory";
        reg = <0 0x00000000 0 0x40000000>; /* 1 GB */
};
```

这个device node。

⑤

获得device node的"reg" property并在下面的code中提取出base和size。

⑥

⑦

把从memory device node中提取的memory的base和size确定的region通过memblock_add()添加到"memory"的memblock_type中的region array中去。

==========

About memblock_reserve()

memblock_reserve()用于把不能用作allocate的memory区分出来（reserved）。

比如在arch/arm/mm/init.c中

```
void __init arm_memblock_init(const struct machine_desc *mdesc)
{
        /* Register the kernel text, kernel data and initrd with memblock. */
#ifdef CONFIG_XIP_KERNEL
        memblock_reserve(__pa(_sdata), _end - _sdata);
#else
        memblock_reserve(__pa(_stext), _end - _stext);                ①
```

```
#endif
#ifdef CONFIG_BLK_DEV_INITRD                                    (A)
    /* FDT scan will populate initrd_start */
    if (initrd_start && !phys_initrd_size) {
        phys_initrd_start = __virt_to_phys(initrd_start);
        phys_initrd_size = initrd_end - initrd_start;
    }
    initrd_start = initrd_end = 0;
    if (phys_initrd_size &&
       !memblock_is_region_memory(phys_initrd_start, phys_initrd_size)) {
        pr_err("INITRD: 0x%08llx+0x%08lx is not a memory region - disabling initrd\n",
            (u64)phys_initrd_start, phys_initrd_size);
        phys_initrd_start = phys_initrd_size = 0;
    }
    if (phys_initrd_size &&
       memblock_is_region_reserved(phys_initrd_start, phys_initrd_size)) {
        pr_err("INITRD: 0x%08llx+0x%08lx overlaps in-use memory region - disabling initrd\n",
            (u64)phys_initrd_start, phys_initrd_size);
        phys_initrd_start = phys_initrd_size = 0;
    }
    if (phys_initrd_size) {
        memblock_reserve(phys_initrd_start, phys_initrd_size);

        /* Now convert initrd to virtual addresses */
        initrd_start = __phys_to_virt(phys_initrd_start);
        initrd_end = initrd_start + phys_initrd_size;
    }
#endif
```

```
    arm_mm_memblock_reserve();                                  ②

    /* reserve any platform specific memblock areas */
    if (mdesc->reserve)                                         ③
        mdesc->reserve();

    early_init_fdt_scan_reserved_mem();                         ④

    /*
     * reserve memory for DMA contigouos allocations,
     * must come from DMA area inside low memory
     */
    dma_contiguous_reserve(arm_dma_limit);

    arm_memblock_steal_permitted = false;
    memblock_dump_all();
}
```

①

把kernel本身的code和readonly data部分所占的memory给reserve。

_stext and _end are in vmlinux.lds。

```
walterzh$ nm vmlinux-3.18.7-yocto-standard | grep " _end"
c06a68f4 B _end
walterzh$ nm vmlinux-3.18.7-yocto-standard | egrep _stext
c0008300 T _stext
```

```
memblock_reserve(__pa(_stext), _end - _stext) =
memblock_reserve(0xc0008300 - 0xc000000, 0xc06a68f4 - 0xc0008300) =
```

memblock_reserve(0x8300, 0x69e5f4)

physical memory的[0x8300, 0x6a68f4)存放着kernel的code和readonly data.

(A)

如果由initramfs，自然需要把initramfs所占的空间给reserved.

由于Granite2 and Gemstone2 LSP启动时的boot cmd为

mmc dev 1;ext2load mmc 1:2 0x400000 /boot/uImage;ext2load mmc 1:2 0xf00000 /boot/mv6220-toc.dtb;setenv bootargs $bootargs root=/dev/mmcblk1p2 uio_pdrv_genirq.of_id=generic-uio rootwait;bootm 0x400000 - 0xf00000

这里bootm 0x400000 - 0xf00000中的"-"就代表initramfs。LSP没用到initramfs。

②

in arch/arm/mm/mmu.c

```
/*
 * Reserve the special regions of memory
 */
void __init arm_mm_memblock_reserve(void)
{
        /*
         * Reserve the page tables.  These are already in use,
         * and can only be in node 0.
         */
        memblock_reserve(__pa(swapper_pg_dir), SWAPPER_PG_DIR_SIZE);

#ifdef CONFIG_SA1111
```

```
        /*
         * Because of the SA1111 DMA bug, we want to preserve our
         * precious DMA-able memory...
         */
        memblock_reserve(PHYS_OFFSET, __pa(swapper_pg_dir) - PHYS_OFFSET);
#endif
}

#ifdef CONFIG_ARM_LPAE
/* the first page is reserved for pgd */
#define SWAPPER_PG_DIR_SIZE      (PAGE_SIZE + \
                                  PTRS_PER_PGD * PTRS_PER_PMD * sizeof(pmd_t))
#else
#define SWAPPER_PG_DIR_SIZE      (PTRS_PER_PGD * sizeof(pgd_t))
#endif
```

swapper_pg_dir是整个virtual memory的根，it is the virtual address of the initial page table.

相当于如下数组

```
pgd_t swapper_pg_dir[PTRS_PER_PGD];

#define PTRS_PER_PGD            2048

$ nm vmlinux-3.18.7-yocto-standard | grep swapper_pg_dir
c0003000 A swapper_pg_dir
```

__pa(swapper_pg_dir) = 0xc0003000 - 0xc0000000 = 0x00003000 (physical address)

SWAPPER_PG_DIR_SIZE = 2048 * 4 = 8K

即把swapper_pg_dir所占用的空间也reserve起来。

memblock_reserve(0x00003000， 8K);

physical memory [0x3000, 0x5000)被swapper_pg_dir占用。

③

in arch/arm/mach-pegmatite/pegmatite.c

```
DT_MACHINE_START(PEGMATITE_DT, "Marvell Pegmatite (Device Tree)")
#ifdef CONFIG_SMP
        .smp            = smp_ops(pegmatite_smp_ops),
#endif
        .init_machine  = pegmatite_dt_init,
        .map_io         = pegmatite_map_io,
        .init_early = pegmatite_init_early,
        .init_irq    = pegmatite_init_irq,
        .init_time  = pegmatite_timer_and_clk_init,
        .restart    = pegmatite_restart,
        .dt_compat     = pegmatite_dt_compat,
#ifdef CONFIG_ZONE_DMA
        .dma_zone_size    = SZ_256M,
#endif
MACHINE_END
```

Gr2 LSP没有定义mdesc->reserve()的callback function。

④

in drivers/of/fdt.c

```
/**
 * early_init_fdt_scan_reserved_mem() - create reserved memory regions
 *
 * This function grabs memory from early allocator for device exclusive use
 * defined in device tree structures. It should be called by arch specific code
 * once the early allocator (i.e. memblock) has been fully activated.
 */
void __init early_init_fdt_scan_reserved_mem(void)
{
	int n;
	u64 base, size;

	if (!initial_boot_params)
		return;

	/* Reserve the dtb region */
	early_init_dt_reserve_memory_arch(__pa(initial_boot_params),        ⑤
					  fdt_totalsize(initial_boot_params),
					  0);

	/* Process header /memreserve/ fields */
	for (n = 0; ; n++) {
		fdt_get_mem_rsv(initial_boot_params, n, &base, &size);                    ⑥
		if (!size)
			break;
		early_init_dt_reserve_memory_arch(base, size, 0);
	}
```

```
    of_scan_flat_dt(__fdt_scan_reserved_mem, NULL);                    ⑦
    fdt_init_reserved_mem();                                           ⑧
}
```

⑤

dtb本身占用的memory要reserve。位于physical memory的0xf00000。

ext2load mmc 1:2 0xf00000 /boot/mv6220-toc.dtb;...;bootm 0x400000 - 0xf00000

⑥

dts中可以使用如下语法来reserve memory

/memreserve/ 0x20000000-0x21ffffff;

G2 LSP的dts没有用到。

⑦

在dtb中寻找reserved-memory device node。比如在mv6270-toc.dts中

```
    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        linux,cma {
            compatible = "shared-dma-pool";
            reusable;
            size = <0 0x10000>;
            alignment = <0 0x2000>;
```

```
                linux,cma-default;
        };
    };

/**
 * fdt_scan_reserved_mem() - scan a single FDT node for reserved memory
 */
static int __init __fdt_scan_reserved_mem(unsigned long node, const char *uname,
                                          int depth, void *data)
{
    static int found;
    const char *status;
    int err;

    if (!found && depth == 1 && strcmp(uname, "reserved-memory") == 0) {          (A)
        if (__reserved_mem_check_root(node) != 0) {
            pr_err("Reserved memory: unsupported node format, ignoring\n");
            /* break scan */
            return 1;
        }
        found = 1;
        /* scan next node */
        return 0;
    } else if (!found) {
        /* scan next node */
        return 0;
    } else if (found && depth < 2) {
        /* scanning of /reserved-memory has been finished */
        return 1;
```

```
        }

        status = of_get_flat_dt_prop(node, "status", NULL);                    (B)
        if (status && strcmp(status, "okay") != 0 && strcmp(status, "ok") != 0)
                return 0;

        err = __reserved_mem_reserve_reg(node, uname);                         (C)
        if (err == -ENOENT && of_get_flat_dt_prop(node, "size", NULL))
                fdt_reserved_mem_save_node(node, uname, 0, 0);                 (D)

        /* scan next node */
        return 0;
}
```

CONFIG_OF_RESERVED_MEM=y

in drivers/base/dma-contiguous.c

obj-$(CONFIG_DMA_CMA) += dma-contiguous.o

CONFIG_DMA_CMA is not set.

in drivers/base/base/dma-coherent.c

obj-$(CONFIG_HAVE_GENERIC_DMA_COHERENT) += dma-coherent.o

CONFIG_HAVE_GENERIC_DMA_COHERENT=y

所以reserved-memory.linux,cma中的reserve memory由drivers/base/base/dma-coherent.c处理。

(A)

(B)

(C)

(D)

⑧

==========

kernel param "memblock=debug"可以打开memblock dump info.

mmc dev 1;ext2load mmc 1:2 0x400000 /boot/uImage;ext2load mmc 1:2 0xf00000 /boot/mv6220-toc.dtb;setenv bootargs $bootargs root=/dev/mmcblk1p2 uio_pdrv_genirq.of_id=generic-uio rootwait **memblock=debug** ;bootm 0x400000 - 0xf00000

"memblock=debug" enable __memblock_dump_all()

```
memblock_reserve: [0x00000000008300-0x000000006a8af3] flags 0x0 arm_memblock_init+0x4c/0x1c0
memblock_reserve: [0x00000000003000-0x00000000007fff] flags 0x0 arm_memblock_init+0x174/0x1c0
memblock_reserve: [0x00000000f00000-0x00000000f11fff] flags 0x0 early_init_fdt_scan_reserved_mem+0x48/0x98
memblock_reserve: [0x00000000f00000-0x00000000f11fff] flags 0x0 early_init_fdt_scan_reserved_mem+0x78/0x98
memblock_reserve: [0x0000002f7f0000-0x0000002f7fffff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
MEMBLOCK configuration:
 memory size = 0x3fc00000 reserved size = 0x6c77f4
 memory.cnt  = 0x2
 memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0
 memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0
 reserved.cnt  = 0x4
 reserved[0x0][0x00000000003000-0x00000000007fff], 0x5000 bytes flags: 0x0
 reserved[0x1][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0
 reserved[0x2][0x00000000f00000-0x00000000f11fff], 0x12000 bytes flags: 0x0
 reserved[0x3][0x0000002f7f0000-0x0000002f7fffff], 0x10000 bytes flags: 0x0
Forcing write-allocate cache policy for SMP
Memory policy: Data cache writealloc
memblock_reserve: [0x0000002f7ee000-0x0000002f7effff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ed000-0x0000002f7edfff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ecfa0-0x0000002f7ecfff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7eb000-0x0000002f7ebfff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ea000-0x0000002f7eafff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ecf70-0x0000002f7ecf9f] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7e9000-0x0000002f7e9fff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7e8000-0x0000002f7e8fff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7e7000-0x0000002f7e7fff] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_virt_alloc_try_nid_nopanic: 8388608 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 alloc_node_mem_map.constprop.82+0x8c/0xb8
memblock_reserve: [0x0000002efe7000-0x0000002f7e6fff] flags 0x0 memblock_virt_alloc_internal+0x
```

```
16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 4 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 free_area_in
it_node+0x378/0x3fc
memblock_reserve: [0x0000002f7ecf40-0x0000002f7ecf43] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 6144 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 zone_wait
_table_init+0x88/0xf4
memblock_reserve: [0x0000002efe5800-0x0000002efe6fff] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 4 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 free_area_in
it_node+0x378/0x3fc
memblock_reserve: [0x0000002f7ecf00-0x0000002f7ecf03] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 12288 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 zone_wai
t_table_init+0x88/0xf4
memblock_reserve: [0x0000002efe2800-0x0000002efe57ff] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 4 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 free_area_in
it_node+0x378/0x3fc
memblock_reserve: [0x0000002f7ecec0-0x0000002f7ecec3] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 12288 bytes align=0x0 nid=0 from=0x0 max_addr=0x0 zone_wai
t_table_init+0x88/0xf4
memblock_reserve: [0x0000002efdf800-0x0000002efe27ff] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid: 40 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 setup_arch+0x5b0/0
xb14
memblock_reserve: [0x0000002f7ece80-0x0000002f7ecea7] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_virt_alloc_try_nid: 40 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 setup_arch+0x5b0/0
xb14
memblock_reserve: [0x0000002f7ece40-0x0000002f7ece67] flags 0x0 memblock_virt_alloc_internal+0x
16c/0x1a8
memblock_reserve: [0x0000002efac5b0-0x0000002efdf7ff] flags 0x0 memblock_alloc_range_nid+0x68/0
x80
memblock_reserve: [0x0000002f7ecf50-0x0000002f7ecf6c] flags 0x0 memblock_alloc_range_nid+0x68/0
x80
memblock_reserve: [0x0000002f7ecf24-0x0000002f7ecf3e] flags 0x0 memblock_alloc_range_nid+0x68/0
x80
memblock_reserve: [0x0000002f7ecf08-0x0000002f7ecf22] flags 0x0 memblock_alloc_range_nid+0x68/0
x80
memblock_reserve: [0x0000002f7ecee4-0x0000002f7ecefe] flags 0x0 memblock_alloc_range_nid+0x68/0
x80
memblock_reserve: [0x0000002f7ecec8-0x0000002f7ecee2] flags 0x0 memblock_alloc_range_nid+0x68/0
x80
```

```
memblock_reserve: [0x0000002f7ecea8-0x0000002f7ecebf] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ece68-0x0000002f7ece7f] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ece28-0x0000002f7ece3f] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ece10-0x0000002f7ece27] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ecdf8-0x0000002f7ece0f] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ecde0-0x0000002f7ecdf7] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ecdc8-0x0000002f7ecddf] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_reserve: [0x0000002f7ecdb0-0x0000002f7ecdc7] flags 0x0 memblock_alloc_range_nid+0x68/0x80
memblock_virt_alloc_try_nid: 133 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 start_kernel+0xd0/0x408
memblock_reserve: [0x0000002f7ecd00-0x0000002f7ecd84] flags 0x0 memblock_virt_alloc_internal+0x16c/0x1a8
memblock_virt_alloc_try_nid: 133 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 start_kernel+0xfc/0x408
memblock_reserve: [0x0000002f7ecc40-0x0000002f7eccc4] flags 0x0 memblock_virt_alloc_internal+0x16c/0x1a8
memblock_virt_alloc_try_nid: 133 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 start_kernel+0x128/0x408
memblock_reserve: [0x0000002f7ecb80-0x0000002f7ecc04] flags 0x0 memblock_virt_alloc_internal+0x16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 4096 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_alloc_alloc_info+0x5c/0x98
memblock_reserve: [0x0000002efab580-0x0000002efac57f] flags 0x0 memblock_virt_alloc_internal+0x16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 4096 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_embed_first_chunk+0x4d8/0x768
memblock_reserve: [0x0000002efaa580-0x0000002efab57f] flags 0x0 memblock_virt_alloc_internal+0x16c/0x1a8
memblock_virt_alloc_try_nid_nopanic: 81920 bytes align=0x1000 nid=-1 from=0x10000000 max_addr=0x0 pcpu_dfl_fc_alloc+0x6c/0x74
memblock_reserve: [0x0000002ef96000-0x0000002efa9fff] flags 0x0 memblock_virt_alloc_internal+0x16c/0x1a8
__memblock_free_early: [0x0000002efa0000-0x0000002ef9ffff] pcpu_embed_first_chunk+0x618/0x768
__memblock_free_early: [0x0000002efaa000-0x0000002efa9fff] pcpu_embed_first_chunk+0x618/0x768
PERCPU: Embedded 10 pages/cpu @eef96000 s8384 r8192 d24384 u40960
memblock_virt_alloc_try_nid: 4 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_chunk+0x1c0/0x850
```

```
75.  memblock_reserve: [0x0000002f7ecb40-0x0000002f7ecb43] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
76.  memblock_virt_alloc_try_nid: 4 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_ch
     unk+0x1f0/0x850
77.  memblock_reserve: [0x0000002f7ecb00-0x0000002f7ecb03] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
78.  memblock_virt_alloc_try_nid: 8 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_ch
     unk+0x218/0x850
79.  memblock_reserve: [0x0000002f7ecac0-0x0000002f7ecac7] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
80.  memblock_virt_alloc_try_nid: 8 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_ch
     unk+0x240/0x850
81.  memblock_reserve: [0x0000002f7eca80-0x0000002f7eca87] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
82.  memblock_virt_alloc_try_nid: 120 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_
     chunk+0x574/0x850
83.  memblock_reserve: [0x0000002f7eca00-0x0000002f7eca77] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
84.  memblock_virt_alloc_try_nid: 68 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_c
     hunk+0x5d0/0x850
85.  memblock_reserve: [0x0000002f7ec980-0x0000002f7ec9c3] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
86.  memblock_virt_alloc_try_nid: 68 bytes align=0x0 nid=-1 from=0x0 max_addr=0x0 pcpu_setup_first_c
     hunk+0x730/0x850
87.  memblock_reserve: [0x0000002f7ec900-0x0000002f7ec943] flags 0x0 memblock_virt_alloc_internal+0x
     16c/0x1a8
88.  __memblock_free_early: [0x0000002efab580-0x0000002efac57f] pcpu_embed_first_chunk+0x730/0x768
89.  __memblock_free_early: [0x0000002efaa580-0x0000002efab57f] pcpu_embed_first_chunk+0x750/0x768
```

初始化期间那么多memblock_reserve()，但size又很小，attachmen中有log.

root@granite2:~# ls -l /sys/kernel/debug/memblock

-r--r--r-- 1 root root 0 Jan 1 1970 memory

-r--r--r-- 1 root root 0 Jan 1 1970 reserved

root@granite2:~# cat /sys/kernel/debug/memblock/memory

0: 0x0000000000000000..0x0000000005ffffff **from 0 to 96M**

1: 0x0000000006400000..0x000000003fffffff **from 100M to 1024M**

也就是Linux看到的physical ram (address space)是断续的，分为两段。原因是from 96M to 100M是给R4用的。

root@granite2:~# cat /sys/kernel/debug/memblock/reserved

0: 0x0000000000003000..0x0000000000007fff (swapper_pg_dir)

1: 0x0000000000008300..0x00000000006a8af3 (kernel code and readonly data)

2: 0x0000000000f00000..0x0000000000f11fff (dtb)

3: 0x000000002eed2000..0x000000002efa9fff

4: 0x000000002efac5b0..0x000000002f7ebfff

5: 0x000000002f7ec900..0x000000002f7ec943

6: 0x000000002f7ec980..0x000000002f7ec9c3

7: 0x000000002f7eca00..0x000000002f7eca77

8: 0x000000002f7eca80..0x000000002f7eca87

9: 0x000000002f7ecac0..0x000000002f7ecac7

10: 0x000000002f7ecb00..0x000000002f7ecb03

11: 0x000000002f7ecb40..0x000000002f7ecb43

12: 0x000000002f7ecb80..0x000000002f7ecc04

13: 0x000000002f7ecc40..0x000000002f7eccc4

14: 0x000000002f7ecd00..0x000000002f7ecd84

15: 0x000000002f7ecdb0..0x000000002f7ecec3

16: 0x000000002f7ecec8..0x000000002f7ecee2

17: 0x000000002f7ecee4..0x000000002f7ecefe

18: 0x000000002f7ecf00..0x000000002f7ecf03

19: 0x000000002f7ecf08..0x000000002f7ecf22

20: 0x000000002f7ecf24..0x000000002f7ecf3e

21: 0x000000002f7ecf40..0x000000002f7ecf43

22: 0x000000002f7ecf50..0x000000002f7ecf6c

23: 0x000000002f7ecf70..0x000000002f7fffff

start_kernel()

|

|

\|/

setup_arch()

|

|

\|/

arm_memblock_init()

in setup_arch() in arch/arm/kernel/setup.c

```
1.  void __init setup_arch(char **cmdline_p)
2.  {
3.          const struct machine_desc *mdesc;
4.
5.          setup_processor();
6.          mdesc = setup_machine_fdt(__atags_pointer);
    (1)
7.          if (!mdesc)
8.                  mdesc = setup_machine_tags(__atags_pointer, __machine_arch_type);    (2)
9.          machine_desc = mdesc;
10.         machine_name = mdesc->name;
11.
12.         if (mdesc->reboot_mode != REBOOT_HARD)
13.                 reboot_mode = mdesc->reboot_mode;
14.
15.         init_mm.start_code = (unsigned long) _text;
16.         init_mm.end_code   = (unsigned long) _etext;
17.         init_mm.end_data   = (unsigned long) _edata;
18.         init_mm.brk        = (unsigned long) _end;
19.
20.         /* populate cmd_line too for later use, preserving boot_command_line */
21.         strlcpy(cmd_line, boot_command_line, COMMAND_LINE_SIZE);
22.         *cmdline_p = cmd_line;
23.
24.         parse_early_param();
25.
26.         early_paging_init(mdesc, lookup_processor_type(read_cpuid_id()));        (3)
27.         setup_dma_zone(mdesc);
              (4)
28.         sanity_check_meminfo();
29. arm_memblock_init(mdesc);                                                        (5)
30.
31.         paging_init(mdesc);
32.
33.  ......
34.
```

```
35.  }
```

初始的关于memory information is from u-boot. (1)

在arm_memblock_init()中观察memblock的变化过程。（加入debug code）

```
1.  void __init arm_memblock_init(const struct machine_desc *mdesc)
2.  {
3.          /* Register the kernel text, kernel data and initrd with memblock. */
4.          printk("walter-0\n");
5.          __memblock_dump_all();
6.
7.  #ifdef CONFIG_XIP_KERNEL
8.          memblock_reserve(__pa(_sdata), _end - _sdata);
9.  #else
10.         memblock_reserve(__pa(_stext), _end - _stext);
11. #endif
12.
13.         printk("walter-1\n");
14.         __memblock_dump_all();
15.
16. #ifdef CONFIG_BLK_DEV_INITRD
17.         /* FDT scan will populate initrd_start */
18.         if (initrd_start && !phys_initrd_size) {
19.                 phys_initrd_start = __virt_to_phys(initrd_start);
20.                 phys_initrd_size = initrd_end - initrd_start;
21.         }
22.         initrd_start = initrd_end = 0;
23.         if (phys_initrd_size &&
24.             !memblock_is_region_memory(phys_initrd_start, phys_initrd_size)) {
25.                 pr_err("INITRD: 0x%08llx+0x%08lx is not a memory region - disabling initrd\n",
26.                        (u64)phys_initrd_start, phys_initrd_size);
27.                 phys_initrd_start = phys_initrd_size = 0;
28.         }
29.         if (phys_initrd_size &&
30.             memblock_is_region_reserved(phys_initrd_start, phys_initrd_size)) {
31.                 pr_err("INITRD: 0x%08llx+0x%08lx overlaps in-use memory region - disabling init
    rd\n",
32.                        (u64)phys_initrd_start, phys_initrd_size);
33.                 phys_initrd_start = phys_initrd_size = 0;
34.         }
35.         if (phys_initrd_size) {
```

```
36.                 memblock_reserve(phys_initrd_start, phys_initrd_size);
37.
38.                 /* Now convert initrd to virtual addresses */
39.                 initrd_start = __phys_to_virt(phys_initrd_start);
40.                 initrd_end = initrd_start + phys_initrd_size;
41.         }
42. #endif
43.         printk("walter-2\n");
44.         __memblock_dump_all();
45.
46.
47.         arm_mm_memblock_reserve();
48.
49.         printk("walter-3\n");
50.         __memblock_dump_all();
51.
52.
53.         /* reserve any platform specific memblock areas */
54.         if (mdesc->reserve)
55.                 mdesc->reserve();
56.
57.         printk("walter-4\n");
58.         __memblock_dump_all();
59.
60.
61.         early_init_fdt_scan_reserved_mem();
62.
63.         printk("walter-5\n");
64.         __memblock_dump_all();
65.
66.
67.         /*
68.          * reserve memory for DMA contigouos allocations,
69.          * must come from DMA area inside low memory
70.          */
71.         dma_contiguous_reserve(arm_dma_limit);
```

```
72.
73.          printk("walter-6\n");
74.          __memblock_dump_all();
75.
76.          arm_memblock_steal_permitted = false;
77.          memblock_dump_all();
78.  }
```

output如下(add comments)：

==========

walter-0

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x0

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x1

reserved[0x0][0x00000000000000-0xffffffffffffffff], 0x0 bytes flags: 0x0

Comments: 整个physical memory是1G - 4M (0x3fc00000),4M是被R4用的，这块内存Linux

根本认为不存在。

1G - 4M的内存被分为2块。

from 0M to 96M

from 100M to 1G

no reserved spave

walter-1

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x6a07f4

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x1

reserved[0x0][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0

walterzh$ nm vmlinux-3.18.7-yocto-standard | grep _stext

c0008300 T _stext

walterzh$ nm vmlinux-3.18.7-yocto-standard | grep " _end$"

c06a8af4 B _end

symbol defines in vmlinux.lds

_stext 到_end包括了kernel的code和readonly data。它们被载入到virtual address space的[c0008300, c06a8af4)，

也就是physical address space的[0x00000000008300-0x000000006a8af3]。这段空间被reserved。不能被再利用。

walter-2

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x6a07f4

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x1

reserved[0x0][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0

没什么变化，因为虽然CONFIG_BLK_DEV_INITRD = y，但Gemstone2 LSP并没有initrd

walter-3

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x6a57f4

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x2

reserved[0x0][0x00000000003000-0x00000000007fff], 0x5000 bytes flags: 0x0

reserved[0x1][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0

```
1.  /*
2.   * Reserve the special regions of memory
3.   */
4.  void __init arm_mm_memblock_reserve(void)
5.  {
6.          /*
7.           * Reserve the page tables.  These are already in use,
8.           * and can only be in node 0.
9.           */
10.         memblock_reserve(__pa(swapper_pg_dir), SWAPPER_PG_DIR_SIZE);
11.
12. #ifdef CONFIG_SA1111
13.         /*
14.          * Because of the SA1111 DMA bug, we want to preserve our
15.          * precious DMA-able memory...
16.          */
17.         memblock_reserve(PHYS_OFFSET, __pa(swapper_pg_dir) - PHYS_OFFSET);
18. #endif
19. }
```

walterzh$ nm vmlinux-3.18.7-yocto-standard | grep swapper_pg_dir

c0003000 A swapper_pg_dir

#define SWAPPER_PG_DIR_SIZE	(PTRS_PER_PGD * sizeof(pgd_t))

#define PTRS_PER_PGD		2048

typedef struct { pmdval_t pgd[2]; } pgd_t;

==> SWAPPER_PG_DIR_SIZE = 2048 * 8 = 16K = 0x4000

这是first leavel page directory entry table的space,同样被reserved。

Question：真正的swapper_pg_dir应该开始于0xc0004000,占用[0xc0004000,0x8000),但reserved space是[0x3000, 0x8000)。

offset不对，size也多了1K。Why？？？

walter-4

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x6a57f4

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x2

reserved[0x0][0x00000000003000-0x00000000007fff], 0x5000 bytes flags: 0x0

reserved[0x1][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0

由于在machine descriptor中并没有custimize .reserve() callback function,所以无变化。

walter-5

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x6c77f4

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x4

reserved[0x0][0x00000000003000-0x00000000007fff], 0x5000 bytes flags: 0x0

reserved[0x1][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0

reserved[0x2][0x00000000f00000-0x00000000f11fff], 0x12000 bytes flags: 0x0

reserved[0x3][0x0000002f7f0000-0x0000002f7fffff], 0x10000 bytes flags: 0x0

```c
/**
 * early_init_fdt_scan_reserved_mem() - create reserved memory regions
 *
 * This function grabs memory from early allocator for device exclusive use
 * defined in device tree structures. It should be called by arch specific code
 * once the early allocator (i.e. memblock) has been fully activated.
 */
void __init early_init_fdt_scan_reserved_mem(void)
{
        int n;
        u64 base, size;

        if (!initial_boot_params)
                return;

        /* Reserve the dtb region */
        early_init_dt_reserve_memory_arch(__pa(initial_boot_params),
                                          fdt_totalsize(initial_boot_params),
                                          0);

        /* Process header /memreserve/ fields */
        for (n = 0; ; n++) {
                fdt_get_mem_rsv(initial_boot_params, n, &base, &size);
                if (!size)
                        break;
                early_init_dt_reserve_memory_arch(base, size, 0);
        }

        of_scan_flat_dt(__fdt_scan_reserved_mem, NULL);
        fdt_init_reserved_mem();
}
```

/* Reserve the dtb region */

early_init_dt_reserve_memory_arch(__pa(initial_boot_params),

fdt_totalsize(initial_boot_params),

0);

reserve [0x00000000f00000-0x00000000f11fff] physical address space.

u-boot启动Linux kernel的命令如下：

bootcmd: mmc dev 1;ext2load mmc 1:2 0x400000 /boot/uImage;ext2load mmc 1:2 0xf00000 /boot/mv6220-toc.dtb;setenv bootargs $bootargs root=/dev/mmcblk1p2 uio_pdrv_genirq.of_id=generic-uio rootwait;bootm 0x400000 - **0xf00000**

这里的0xf00000就是/boot/mv6220-toc.dtb的载入address。这里也要reserve。

reserved[0x3][0x0000002f7f0000-0x0000002f7fffff], 0x10000 bytes flags: 0x0

？？？

walter-6

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0x6c77f4

memory.cnt = 0x2

memory[0x0][0x00000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x4

reserved[0x0][0x00000000003000-0x00000000007fff], 0x5000 bytes flags: 0x0

reserved[0x1][0x00000000008300-0x000000006a8af3], 0x6a07f4 bytes flags: 0x0

reserved[0x2][0x00000000f00000-0x00000000f11fff], 0x12000 bytes flags: 0x0

reserved[0x3][0x0000002f7f0000-0x0000002f7fffff], 0x10000 bytes flags: 0x0

dma_contiguous_reserve() function在Gemstone2 LSP上没有作用，memblock没有变化。

==========