The board of Granite2 or Gemstone2 own 1024M (1G) DRAM, but the memory region from 96M to 100M is for R4 core (running Threadx OS), the Linux kernel running on A53 cores could only access 1020M (1024 - 4) memory.

In 6220-toc.dts

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

The dts tell the kernel it own 1G (1024M) memory, but when kernel initializing, kernel could recognize it only own 1020M memory.

```
start_kernel()
    |
    |
   \|/
mm_init()
    |
    |
   \|/
mem_init()
```

in mem_init(), we could dump the all memory that kernel think it could manage.

in include/linux/memblock.h

```
struct memblock {
        bool bottom_up;  /* is bottom up direction? */
        phys_addr_t current_limit;
        struct memblock_type memory;
        struct memblock_type reserved;
#ifdef CONFIG_HAVE_MEMBLOCK_PHYS_MAP
        struct memblock_type physmem;
#endif
};
```

in mm/memblock.c

```
struct memblock memblock;
```

memblock.memory records all managed memory.

```
MEMBLOCK configuration:
 memory size = 0x3fc00000 reserved size = 0x0
 memory.cnt  = 0x2
 memory[0x0][0x0000000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0
 memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0
 reserved.cnt  = 0x1
 reserved[0x0][0x0000000000000000-0xffffffffffffffff], 0x0 bytes flags: 0x0
```

The kernel think it manages 2 segments memory blocks.

```
 0 - 0x05ffffff           (0 - 96M)
 0x06400000 - 0x3fffffff    (100M - 1G)
```

Question: How kernel know the memory from 96M to 100M doesn't belong to it ?

In fact, u-boot know it could only manage 1020M memory.

in u-boot startup log

```
=================================================
U-Boot 2014.01 (Oct 09 2015 - 15:16:10)

I2C:   ready
DRAM:  1020 MiB
MMC:   mv_sdh: 0, mv_sdh: 1, mv_sdh: 2
Using default environment
=================================================
```

step 1:

in u-boot/board/pegmatite/setup.c

```c
static void fixup_banks_for_reserved_ram(void)
{
    int curbank_mc = 0;
    int curbank_usable = 0;
    bank_info_t reserved = {
.start = 0x06000000,
.size  = 0x00400000,
    };

    memset(usable_banks, 0, sizeof(usable_banks));
    for (curbank_mc = 0; curbank_mc < ARRAY_SIZE(mc_banks); curbank_mc++)
    {
        if ((mc_banks[curbank_mc].start <= reserved.start) &&
           ((mc_banks[curbank_mc].start + mc_banks[curbank_mc].size) >= (reserved.start + reserved.size)))
        {
            bank_info_t before, after;

            before.start = mc_banks[curbank_mc].start;
            before.size  = reserved.start - before.start;

            after.start  = reserved.start + reserved.size;
            after.size   = mc_banks[curbank_mc].start + mc_banks[curbank_mc].size - after.start;

            if (before.size > 0)
            {
                usable_banks[curbank_usable].start = before.start;
                usable_banks[curbank_usable].size  = before.size;
                curbank_usable++;
            }
            if (after.size > 0)
            {
                usable_banks[curbank_usable].start = after.start;
                usable_banks[curbank_usable].size  = after.size;
                curbank_usable++;
            }
        }
        else
        {
            usable_banks[curbank_usable].start = mc_banks[curbank_mc].start;
            usable_banks[curbank_usable].size  = mc_banks[curbank_mc].size;
            curbank_usable++;
        }
    }
}

void dram_init_banksize(void)
{
    int i;

    for (i = 0; i < CONFIG_NR_DRAM_BANKS; i++) {
        gd->bd->bi_dram[i].start = usable_banks[i].start;
        gd->bd->bi_dram[i].size  = usable_banks[i].size;
    }
}

int dram_init(void)
{
    int i;

    read_mc_bank_info();
    fixup_banks_for_reserved_ram();

```

```
        gd->ram_size = 0;
        for (i = 0; i < ARRAY_SIZE(mc_banks); ++i) {
            if (mc_banks[i].start == 0) {
                gd->ram_size = min(mc_banks[i].size, (3UL * 1024 * 1024 * 1024));
                break;
            }
        }

        return 0;
    }
```

When u-boot initialization, it will get memory information by querying MC(Memory Controller) and make [0x06000000, 0x06400000), [96M, 100), is reserved.

read_mc_bank_info() function get memory bank information by querying MC.

fixup_banks_for_reserved_ram() function removes the reserved memory region from 1G memory space. There are 2 segments memory regions in usable_banks[] array.

usable_banks[0] = from 0 to 96M

usable_banks[1] = from 100M to 1024M

step 2:

```
do_bootm_linux()

   |

   |

  \|/

boot_prep_linux()    in u-boot/arch/arm/lib/bootm.c

   |

   |
   \|/

image_setup_linux()    in u-boot/common/image.c

    |

    |

   \|/

image_setup_libfdt()    in u-boot/common/image-fdt.c

    |

    |

   \|/

arch_fixup_memory_node()    in u-boot/arch/arm/lib/bootm-fdt.c

    |

    |

   \|/

fdt_fixup_memory_banks()
```

```c
int fdt_fixup_memory_banks(void *blob, u64 start[], u64 size[], int banks)
{
        int err, nodeoffset;
        int addr_cell_len, size_cell_len, len;
        u8 tmp[MEMORY_BANKS_MAX * 16]; /* Up to 64-bit address + 64-bit size */
        int bank;

        if (banks > MEMORY_BANKS_MAX) {
                printf("%s: num banks %d exceeds hardcoded limit %d."
                       " Recompile with higher MEMORY_BANKS_MAX?\n",
                       __FUNCTION__, banks, MEMORY_BANKS_MAX);
                return -1;
        }

        err = fdt_check_header(blob);
        if (err < 0) {
                printf("%s: %s\n", __FUNCTION__, fdt_strerror(err));
                return err;
        }

        /* update, or add and update /memory node */
        nodeoffset = fdt_path_offset(blob, "/memory");
        if (nodeoffset < 0) {
                nodeoffset = fdt_add_subnode(blob, 0, "memory");
                if (nodeoffset < 0) {
                        printf("WARNING: could not create /memory: %s.\n",
                                        fdt_strerror(nodeoffset));
                        return nodeoffset;
                }
        }
        err = fdt_setprop(blob, nodeoffset, "device_type", "memory",
                        sizeof("memory"));
        if (err < 0) {
                printf("WARNING: could not set %s %s.\n", "device_type",
                                fdt_strerror(err));
                return err;
        }

        addr_cell_len = get_cells_len(blob, "#address-cells");
        size_cell_len = get_cells_len(blob, "#size-cells");

        for (bank = 0, len = 0; bank < banks; bank++) {
                write_cell(tmp + len, start[bank], addr_cell_len);
                len += addr_cell_len;

                write_cell(tmp + len, size[bank], size_cell_len);
                len += size_cell_len;
        }

        err = fdt_setprop(blob, nodeoffset, "reg", tmp, len);
        if (err < 0) {
                printf("WARNING: could not set %s %s.\n",
                                "reg", fdt_strerror(err));
                return err;
        }
        return 0;
}
```

fdt_fixup_memory_banks() function do the following work.

replace original "memory" device node with new one according to start[] and size[].

blob points to dtb binary file.

original memory: 0 - 1024

New memory: 0 - 96M & 100M - 1024.

When Linux kernel get the modified dtb, it will know which memory regions it could manage.