bootm 0x400000 - 0xf00000

u-boot把uImage载入到4M(0x400000)边界，而dtb载入15M(0xf00000)边界。但compressed vmlinux被指定运行的地址

位于0x8000。

in u-boot/common/image.c

```
1.  ulong load_addr = CONFIG_SYS_LOAD_ADDR; /* Default Load Address */
```

这里的CONFIG_SYS_LOAD_ADDR就是可以由developer定制的compressed vmlinux的载入地址。

in u-boot/include/configs/pegmatite.h

```
1.  #define CONFIG_SYS_LOAD_ADDR    0x8000  /* default load address */
```

而具体把compressed vmlinux从uImage中提取出然后载入到0x8000，则在boot_get_kernel() function(in

u-boot/common/cmd_bootm.c)。

"bootm 0x400000 - 0xf00000"对应的handler是do_bootm() --- in common/cmd_bootm.c

这里argc = 3, argv[0] = "0x400000",argv[1] = "-", argv[2] = "0xf00000".

argv[0]是uImage被载入的physical affress，argv[2]是dtb被载入的physical address。

下面是与do_bootm()相关的function的分析

```
/**
 * boot_get_kernel - find kernel image
 * @os_data: pointer to a ulong variable, will hold os data start address
 * @os_len: pointer to a ulong variable, will hold os data length
 *
 * boot_get_kernel() tries to find a kernel image, verifies its integrity
 * and locates kernel data.
 *
 * returns:
 *     pointer to image header if valid image was found, plus kernel start
 *     address and length, otherwise NULL
 */
static const void *boot_get_kernel(cmd_tbl_t *cmdtp, int flag, int argc,
                char * const argv[], bootm_headers_t *images, ulong *os_data,
                ulong *os_len)
{
        image_header_t  *hdr;
        ulong           img_addr;
        const void *buf;
#if defined(CONFIG_FIT)
        const char      *fit_uname_config = NULL;
        const char      *fit_uname_kernel = NULL;
        int             os_noffset;
#endif

        /* find out kernel image address */
        if (argc < 1) {
                img_addr = load_addr;
                debug("*  kernel: default image load address = 0x%08lx\n",
                                load_addr);
#if defined(CONFIG_FIT)
```

```
32.         } else if (fit_parse_conf(argv[0], load_addr, &img_addr,
33.                                                   &fit_uname_config)) {
34.                 debug("*  kernel: config '%s' from image at 0x%08lx\n",
35.                                 fit_uname_config, img_addr);
36.         } else if (fit_parse_subimage(argv[0], load_addr, &img_addr,
37.                                                   &fit_uname_kernel)) {
38.                 debug("*  kernel: subimage '%s' from image at 0x%08lx\n",
39.                                 fit_uname_kernel, img_addr);
40. #endif
41.         } else {
42.                 img_addr = simple_strtoul(argv[0], NULL, 16);
                                                                  ①
43.                 debug("*  kernel: cmdline image address = 0x%08lx\n", img_addr);
44.         }
45.
46.         bootstage_mark(BOOTSTAGE_ID_CHECK_MAGIC);
47.
48.         /* copy from dataflash if needed */
49.         img_addr = genimg_get_image(img_addr);
50.
51.         /* check image type, for FIT images get FIT kernel node */
52.         *os_data = *os_len = 0;
53.         buf = map_sysmem(img_addr, 0);
54.         switch (genimg_get_format(buf)) {
55.         case IMAGE_FORMAT_LEGACY:
56.                 printf("## Booting kernel from Legacy Image at %08lx ...\n",
57.                                 img_addr);
58.                 hdr = image_get_kernel(img_addr, images->verify);
                                                      ②
59.                 if (!hdr)
60.                         return NULL;
61.                 bootstage_mark(BOOTSTAGE_ID_CHECK_IMAGETYPE);
```

```
62.
63.                 /* get os_data and os_len */
64.                 switch (image_get_type(hdr)) {
65.                 case IH_TYPE_KERNEL:
66.                 case IH_TYPE_KERNEL_NOLOAD:
67.                         *os_data = image_get_data(hdr);
                ③
68.                         *os_len = image_get_data_size(hdr);
              ④
69.                         break;
70.                 case IH_TYPE_MULTI:
71.                         image_multi_getimg(hdr, 0, os_data, os_len);
72.                         break;
73.                 case IH_TYPE_STANDALONE:
74.                         *os_data = image_get_data(hdr);
75.                         *os_len = image_get_data_size(hdr);
76.                         break;
77.                 default:
78.                         printf("Wrong Image Type for %s command\n",
79.                                 cmdtp->name);
80.                         bootstage_error(BOOTSTAGE_ID_CHECK_IMAGETYPE);
81.                         return NULL;
82.                 }
83.
84.                 /*
85.                  * copy image header to allow for image overwrites during
86.                  * kernel decompression.
87.                  */
88.                 memmove(&images->legacy_hdr_os_copy, hdr,
89.                         sizeof(image_header_t));
90.
```

```
91.                 /* save pointer to image header */
92.                 images->legacy_hdr_os = hdr;
93.
94.                 images->legacy_hdr_valid = 1;
95.                 bootstage_mark(BOOTSTAGE_ID_DECOMP_IMAGE);
96.                 break;
97.     #if defined(CONFIG_FIT)
98.         case IMAGE_FORMAT_FIT:
99.                 os_noffset = fit_image_load(images, FIT_KERNEL_PROP,
100.                                 img_addr,
101.                                 &fit_uname_kernel, &fit_uname_config,
102.                                 IH_ARCH_DEFAULT, IH_TYPE_KERNEL,
103.                                 BOOTSTAGE_ID_FIT_KERNEL_START,
104.                                 FIT_LOAD_IGNORED, os_data, os_len);
105.                 if (os_noffset < 0)
106.                         return NULL;
107.
108.                 images->fit_hdr_os = map_sysmem(img_addr, 0);
109.                 images->fit_uname_os = fit_uname_kernel;
110.                 images->fit_uname_cfg = fit_uname_config;
111.                 images->fit_noffset_os = os_noffset;
112.                 break;
113.     #endif
114.         default:
115.                 printf("Wrong Image Format for %s command\n", cmdtp->name);
116.                 bootstage_error(BOOTSTAGE_ID_FIT_KERNEL_INFO);
117.                 return NULL;
118.         }
119.
120.         debug("   kernel data at 0x%08lx, len = 0x%08lx (%ld)\n",
121.                         *os_data, *os_len, *os_len);
122.
```

```
123.            return buf;
124.    }
```

①

img_addr = 0x400000.

kernel: cmdline image address = 0x00400000 (in log)

img_addr指向载入的uImage，并且也指向image_header_t

```
1.  typedef struct image_header {
2.          __be32          ih_magic;       /* Image Header Magic Number    */
3.          __be32          ih_hcrc;        /* Image Header CRC Checksum    */
4.          __be32          ih_time;        /* Image Creation Timestamp     */
5.          __be32          ih_size;        /* Image Data Size              */
6.          __be32          ih_load;        /* Data  Load  Address          */
7.          __be32          ih_ep;          /* Entry Point Address          */
8.          __be32          ih_dcrc;        /* Image Data CRC Checksum      */
9.          uint8_t         ih_os;          /* Operating System             */
10.         uint8_t         ih_arch;        /* CPU architecture             */
11.         uint8_t         ih_type;        /* Image Type                   */
12.         uint8_t         ih_comp;        /* Compression Type             */
13.         uint8_t         ih_name[IH_NMLEN];      /* Image Name           */
14. } image_header_t;
```

比如 mkimage -l 可以显示image_header_t中的信息。

$ mkimage -l uImage

Image Name: Linux-3.18.7-yocto-standard

Created: Fri Dec 25 21:47:17 2015

Image Type: ARM Linux Kernel Image (uncompressed)

Data Size: 3100720 Bytes = 3028.05 kB = 2.96 MB

Load Address: 00008000

Entry Point: 00008000

②

检查img_addr->ih_magic == IH_MAGIC

```
#define IH_MAGIC        0x27051956      /* Image Magic Number           */
```

返回IMAGE_FORMAT_LEGACY format (G2 LSP的uImage的format)

③

```
static inline ulong image_get_data(const image_header_t *hdr)
{
        return ((ulong)hdr + image_get_header_size());
}
```

就是跳过uImage的前64 bytes的image_header_t structure，其后就是zImage。

*os_data pointer to zImage。

④

```
static inline uint32_t image_get_data_size(const image_header_t *hdr)
{
        return image_get_size(hdr);
}
```

就是返回image_header_t structure中的ih_size field,也就是zImage的size。

boot_get_kernel()主要就是为了返回zImage的address和size。

```
1.  static int bootm_find_os(cmd_tbl_t *cmdtp, int flag, int argc,
2.                          char * const argv[])
3.  {
4.          const void *os_hdr;
5.
6.          /* get kernel image header, start address and length */
7.          os_hdr = boot_get_kernel(cmdtp, flag, argc, argv,
8.                          &images, &images.os.image_start, &images.os.image_len);
9.          if (images.os.image_len == 0) {
10.                 puts("ERROR: can't get kernel image!\n");
11.                 return 1;
12.         }
13.
14.         /* get image parameters */
15.         switch (genimg_get_format(os_hdr)) {
16.         case IMAGE_FORMAT_LEGACY:
                                        ①
17.                 images.os.type = image_get_type(os_hdr);
        ②
18.                 images.os.comp = image_get_comp(os_hdr);
    ③
19.                 images.os.os = image_get_os(os_hdr);
              ④
20.
21.                 images.os.end = image_get_image_end(os_hdr);          ⑤
22.                 images.os.load = image_get_load(os_hdr);
        ⑥
23.                 break;
24. #if defined(CONFIG_FIT)
25.         case IMAGE_FORMAT_FIT:
26.                 if (fit_image_get_type(images.fit_hdr_os,
27.                                         images.fit_noffset_os, &images.os.type))
    {
```

```
28.                         puts("Can't get image type!\n");
29.                         bootstage_error(BOOTSTAGE_ID_FIT_TYPE);
30.                         return 1;
31.                 }
32.
33.                 if (fit_image_get_comp(images.fit_hdr_os,
34.                                         images.fit_noffset_os, &images.os.comp))
    {
35.                         puts("Can't get image compression!\n");
36.                         bootstage_error(BOOTSTAGE_ID_FIT_COMPRESSION);
37.                         return 1;
38.                 }
39.
40.                 if (fit_image_get_os(images.fit_hdr_os,
41.                                         images.fit_noffset_os, &images.os.os)) {
42.                         puts("Can't get image OS!\n");
43.                         bootstage_error(BOOTSTAGE_ID_FIT_OS);
44.                         return 1;
45.                 }
46.
47.                 images.os.end = fit_get_end(images.fit_hdr_os);
48.
49.                 if (fit_image_get_load(images.fit_hdr_os, images.fit_noffset_os,
50.                                         &images.os.load)) {
51.                         puts("Can't get image load address!\n");
52.                         bootstage_error(BOOTSTAGE_ID_FIT_LOADADDR);
53.                         return 1;
54.                 }
55.                 break;
56.   #endif
57.         default:
58.                 puts("ERROR: unknown image format type!\n");
```

```
59.                 return 1;
60.         }
61.
62.         /* find kernel entry point */
63.         if (images.legacy_hdr_valid) {
64.                 images.ep = image_get_ep(&images.legacy_hdr_os_copy);
    ⑦
65. #if defined(CONFIG_FIT)
66.         } else if (images.fit_uname_os) {
67.                 int ret;
68.
69.                 ret = fit_image_get_entry(images.fit_hdr_os,
70.                                           images.fit_noffset_os, &images.ep);
71.                 if (ret) {
72.                         puts("Can't get entry point property!\n");
73.                         return 1;
74.                 }
75. #endif
76.         } else {
77.                 puts("Could not find kernel entry point!\n");
78.                 return 1;
79.         }
80.
81.         if (images.os.type == IH_TYPE_KERNEL_NOLOAD) {
82.                 images.os.load = images.os.image_start;
83.                 images.ep += images.os.load;
84.         }
85.
86.         images.os.start = (ulong)os_hdr;
87.
88.         return 0;
89. }
```

①

G2 LSP中的uImage是IMAGE_FORMAT_LEGACY format

②

返回uImage header中的ih_type field,

Image Type: ARM Linux Kernel Image

③

返回uImage header中的ih_comp field, compressed type

(uncompressed)

④

返回uImage header中的ih_os field, (Operation System)

Linux

⑤

```
static inline ulong image_get_image_end(const image_header_t *hdr)
{
        return ((ulong)hdr + image_get_image_size(hdr));
}

static inline uint32_t image_get_image_size(const image_header_t *hdr)
{
        return (image_get_size(hdr) + image_get_header_size());
}
```

hdr指向uImage(0x400000), hdr + sizeof(hdr) + sizeof(zImage) = uImage的end

⑥

返回uImage header中的ih_load field, Data Load Address

Load Address: 00008000

也就是zImage将被载入到0x8000。

目前uImage在0x400000 (4M边界)，而zImage将被载入到0x8000(32K边界)。

0x400000 - 0x8000 = 0x3f8000 (4161536 bytes)

即uImage与zImage之间由416136 bytes space。

而zImage的大小为3100720。安全！否则就有问题。

Data Size: 3100720 Bytes = 3028.05 kB = 2.96 MB

⑦

获得zImage的入口地址.u-boot最终要跳转到改地址去运行Linux kernel.

Entry Point: 00008000

```
1.  static int bootm_find_other(cmd_tbl_t *cmdtp, int flag, int argc,
2.                             char * const argv[])
3.  {
4.          if (((images.os.type == IH_TYPE_KERNEL) ||
5.               (images.os.type == IH_TYPE_KERNEL_NOLOAD) ||
6.               (images.os.type == IH_TYPE_MULTI)) &&
7.              (images.os.os == IH_OS_LINUX ||
8.                   images.os.os == IH_OS_VXWORKS)) {
9.                  if (bootm_find_ramdisk(flag, argc, argv))
10.                         return 1;
11.
12. #if defined(CONFIG_OF_LIBFDT)
13.                 if (bootm_find_fdt(flag, argc, argv))
14.                         return 1;
15. #endif
16.         }
17.
18.         return 0;
19. }
```

```
1.  static int bootm_find_other(cmd_tbl_t *cmdtp, int flag, int argc,
2.                             char * const argv[])
3.  {
4.          if (((images.os.type == IH_TYPE_KERNEL) ||
5.               (images.os.type == IH_TYPE_KERNEL_NOLOAD) ||
6.               (images.os.type == IH_TYPE_MULTI)) &&
7.              (images.os.os == IH_OS_LINUX ||
8.                   images.os.os == IH_OS_VXWORKS)) {
9.                  if (bootm_find_ramdisk(flag, argc, argv))
                      ①
10.                         return 1;
11.
12. #if defined(CONFIG_OF_LIBFDT)
13.                 if (bootm_find_fdt(flag, argc, argv))
                            ②
14.                         return 1;
15. #endif
16.         }
17.
18.         return 0;
19. }
```

①

在G2 LSP中uImage的type为IH_TYPE_KERNEL and IH_OS_LINUX,所以会进入bootm_find_ramdisk() function。

但由于

"bootm 0x400000 - 0xf00000"

G2 LSP并没有ram disk，所以实际上没作用。

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

```
        if (argc >= 2)
                select = argv[1];
        /*
         * Look for a '-' which indicates to ignore the
         * ramdisk argument
         */
        if (select && strcmp(select, "-") ==  0) {
                debug("## Skipping init Ramdisk\n");
                rd_len = rd_data = 0;
        } else if (select || genimg_has_config(images)) {
```

②

```
static int bootm_find_fdt(int flag, int argc, char * const argv[])
{
        int ret;

        /* find flattened device tree */
        ret = boot_get_fdt(flag, argc, argv, IH_ARCH_DEFAULT, &images,
                           &images.ft_addr, &images.ft_len);
        if (ret) {
                puts("Could not find a valid device tree\n");
                return 1;
        }

        set_working_fdt_addr(images.ft_addr);

        return 0;
}
```

由于G2 LSP是单独指定dtb file的，而不是被嵌入在image file中的，所以boot_get_fdt()运行的路径比较简单，就是

images.ft_addr = 0xf00000, 而images.ft_len = dtb size(这个值记录在dtb本身的eader中)

set_working_fdt_addr(images.ft_addr);

==>

setenv_addr("fdtaddr", addr);

```
1.  static int bootm_load_os(bootm_headers_t *images, unsigned long *load_end,
2.                  int boot_progress)
3.  {
4.          image_info_t os = images->os;
5.          uint8_t comp = os.comp;
6.          ulong load = os.load;
7.          ulong blob_start = os.start;
8.          ulong blob_end = os.end;
9.          ulong image_start = os.image_start;
10.         ulong image_len = os.image_len;
11.         __maybe_unused uint unc_len = CONFIG_SYS_BOOTM_LEN;
12.         int no_overlap = 0;
13.         void *load_buf, *image_buf;
14. #if defined(CONFIG_LZMA) || defined(CONFIG_LZO)
15.         int ret;
16. #endif /* defined(CONFIG_LZMA) || defined(CONFIG_LZO) */
17.
18.         const char *type_name = genimg_get_type_name(os.type);
19.
20.         load_buf = map_sysmem(load, unc_len);
21.         image_buf = map_sysmem(image_start, image_len);
22.         switch (comp) {
23.         case IH_COMP_NONE:
                                        ①
24.                 if (load == blob_start || load == image_start) {
25.                         printf("   XIP %s ... ", type_name);
26.                         no_overlap = 1;
27.                 } else {
                                                                    ②
28.                         printf("   Loading %s ... ", type_name);
29.                         memmove_wd(load_buf, image_buf, image_len, CHUNKSZ);
                        ③
30.                 }
31.                 *load_end = load + image_len;
32.                 break;
33. #ifdef CONFIG_GZIP
34.         case IH_COMP_GZIP:
35.                 printf("   Uncompressing %s ... ", type_name);
36.                 if (gunzip(load_buf, unc_len, image_buf, &image_len) != 0) {
37.                         puts("GUNZIP: uncompress, out-of-mem or overwrite "
38.                                 "error - must RESET board to recover\n");
39.                         if (boot_progress)
40.                                 bootstage_error(BOOTSTAGE_ID_DECOMP_IMAGE);
41.                         return BOOTM_ERR_RESET;
42.                 }
43.
44.                 *load_end = load + image_len;
45.                 break;
46. #endif /* CONFIG_GZIP */
47. #ifdef CONFIG_BZIP2
48.         case IH_COMP_BZIP2:
```

```
49.                 printf("   Uncompressing %s ... ", type_name);
50.                 /*
51.                  * If we've got less than 4 MB of malloc() space,
52.                  * use slower decompression algorithm which requires
53.                  * at most 2300 KB of memory.
54.                  */
55.                 int i = BZ2_bzBuffToBuffDecompress(load_buf, &unc_len,
56.                         image_buf, image_len,
57.                         CONFIG_SYS_MALLOC_LEN < (4096 * 1024), 0);
58.                 if (i != BZ_OK) {
59.                         printf("BUNZIP2: uncompress or overwrite error %d "
60.                                 "- must RESET board to recover\n", i);
61.                         if (boot_progress)
62.                                 bootstage_error(BOOTSTAGE_ID_DECOMP_IMAGE);
63.                         return BOOTM_ERR_RESET;
64.                 }
65.
66.                 *load_end = load + unc_len;
67.                 break;
68. #endif /* CONFIG_BZIP2 */
69. #ifdef CONFIG_LZMA
70.         case IH_COMP_LZMA: {
71.                 SizeT lzma_len = unc_len;
72.                 printf("   Uncompressing %s ... ", type_name);
73.
74.                 ret = lzmaBuffToBuffDecompress(load_buf, &lzma_len,
75.                                                image_buf, image_len);
76.                 unc_len = lzma_len;
77.                 if (ret != SZ_OK) {
78.                         printf("LZMA: uncompress or overwrite error %d "
79.                                 "- must RESET board to recover\n", ret);
80.                         bootstage_error(BOOTSTAGE_ID_DECOMP_IMAGE);
81.                         return BOOTM_ERR_RESET;
82.                 }
83.                 *load_end = load + unc_len;
84.                 break;
85.         }
86. #endif /* CONFIG_LZMA */
87. #ifdef CONFIG_LZO
88.         case IH_COMP_LZO: {
89.                 size_t size;
90.
91.                 printf("   Uncompressing %s ... ", type_name);
92.
93.                 ret = lzop_decompress(image_buf, image_len, load_buf, &size);
94.                 if (ret != LZO_E_OK) {
95.                         printf("LZO: uncompress or overwrite error %d "
96.                               "- must RESET board to recover\n", ret);
97.                         if (boot_progress)
98.                                 bootstage_error(BOOTSTAGE_ID_DECOMP_IMAGE);
99.                         return BOOTM_ERR_RESET;
100.                }
101.
102.                *load_end = load + size;
```

```
                break;
        }
#endif /* CONFIG_LZO */
        default:
                printf("Unimplemented compression type %d\n", comp);
                return BOOTM_ERR_UNIMPLEMENTED;
        }

#ifdef CONFIG_SYS_CACHELINE_SIZE
        *load_end = (*load_end + (CONFIG_SYS_CACHELINE_SIZE - 1)) & ~(CONFIG_SYS_
CACHELINE_SIZE - 1);
#endif
        flush_cache(load, (*load_end - load) * sizeof(ulong));

        puts("OK\n");
        debug("   kernel loaded at 0x%08lx, end = 0x%08lx\n", load, *load_end);
        bootstage_mark(BOOTSTAGE_ID_KERNEL_LOADED);

        if (!no_overlap && (load < blob_end) && (*load_end > blob_start)) {
                debug("images.os.start = 0x%lX, images.os.end = 0x%lx\n",
                        blob_start, blob_end);
                debug("images.os.load = 0x%lx, load_end = 0x%lx\n", load,
                        *load_end);

                /* Check what type of image this is. */
                if (images->legacy_hdr_valid) {
                        if (image_get_type(&images->legacy_hdr_os_copy)
                                        == IH_TYPE_MULTI)
                                puts("WARNING: legacy format multi component imag
e overwritten\n");
                        return BOOTM_ERR_OVERLAP;
                } else {
                        puts("ERROR: new format image overwritten - must RESET th
e board to recover\n");
                        bootstage_error(BOOTSTAGE_ID_OVERWRITTEN);
                        return BOOTM_ERR_RESET;
                }
        }

        return 0;
}
```

由于G2 LSP的uImage是未压缩的zImage，所以其实bootm_load_os()出奇的简单。

其实就是memmove(0x8000, 0x400040, zImage_size)

①

未压缩的zImage

②

uImage与zImage运行地址不同(非XIP)

③

在G2 LSP中就是如下一行code

memmove(to, from, len);

把从0x400000 + 0x40的zImage搬移到0x8000。

---

in do_bootm_states() function

```
        ......

        /* Load the OS */
        if (!ret && (states & BOOTM_STATE_LOADOS)) {
                ulong load_end;

                iflag = bootm_disable_interrupts();
                ret = bootm_load_os(images, &load_end, 0);
                if (ret == 0)
                        lmb_reserve(&images->lmb, images->os.load,
  ①
                                    (load_end - images->os.load));
                else if (ret && ret != BOOTM_ERR_OVERLAP)
                        goto err;
                else if (ret == BOOTM_ERR_OVERLAP)
                        ret = 0;
#if defined(CONFIG_SILENT_CONSOLE) && !defined(CONFIG_SILENT_U_BOOT_ONLY)
                if (images->os.os == IH_OS_LINUX)
                        fixup_silent_linux();
#endif
        }

        ......
```

①

在bootm_load_os()成功的情况下(ret = 0)，把已经被搬移到0x8000的zImage的这块space标记为"reserved"。

---

in do_bootm_states() function

```
......

#if defined(CONFIG_OF_LIBFDT) && defined(CONFIG_LMB)
        if (!ret && (states & BOOTM_STATE_FDT)) {
                boot_fdt_add_mem_rsv_regions(&images->lmb, images->ft_addr);
        ①
                ret = boot_relocate_fdt(&images->lmb, &images->ft_addr,
                ②
                                        &images->ft_len);
        }
#endif

......
```

①

```
/**
 * boot_fdt_add_mem_rsv_regions - Mark the memreserve sections as unusable
 * @lmb: pointer to lmb handle, will be used for memory mgmt
 * @fdt_blob: pointer to fdt blob base address
 *
 * Adds the memreserve regions in the dtb to the lmb block.  Adding the
 * memreserve regions prevents u-boot from using them to store the initrd
 * or the fdt blob.
 */
void boot_fdt_add_mem_rsv_regions(struct lmb *lmb, void *fdt_blob)
{
        uint64_t addr, size;
        int i, total;

        if (fdt_check_header(fdt_blob) != 0)
                return;

        total = fdt_num_mem_rsv(fdt_blob);
        for (i = 0; i < total; i++) {
                if (fdt_get_mem_rsv(fdt_blob, i, &addr, &size) != 0)
                        continue;
                printf("   reserving fdt memory region: addr=%llx size=%llx\n",
                       (unsigned long long)addr, (unsigned long long)size);
                lmb_reserve(lmb, addr, size);
        }
}
```

在dts中可以指定reserved space，这里u-boot就是解析其中的reserved space,并在u-boot的logic memory block中也标记之。

由此可见，u-boot并不general,它也需要与kernel的信息。

②

如果定义了"fdt_high" environment variable，可以在这里搬移dtb。但在G2 LSP中并没有定义，所以dtb还是会呆在0xf00000原地。

fdt_high = getenv("fdt_high");

---

in do_bootm_states() function

```
1.          ......
2.
3.          /* Call various other states that are not generally used */
4.          if (!ret && (states & BOOTM_STATE_OS_CMDLINE))
5.                  ret = boot_fn(BOOTM_STATE_OS_CMDLINE, argc, argv, images);
6.          if (!ret && (states & BOOTM_STATE_OS_BD_T))
7.                  ret = boot_fn(BOOTM_STATE_OS_BD_T, argc, argv, images);
8.          if (!ret && (states & BOOTM_STATE_OS_PREP))
9.                  ret = boot_fn(BOOTM_STATE_OS_PREP, argc, argv, images);
          ①
10.
11.         ......
```

①

G2 LSP 会运行这一支

这里的boot_fn = do_bootm_linux()

在u-boot即将跳转到Linux kernel的最后是运行do_bootm_linux() function。

```
1.  int do_bootm_linux(int flag, int argc, char *argv[], bootm_headers_t *images)
2.  {
3.          /* No need for those on ARM */
4.          if (flag & BOOTM_STATE_OS_BD_T || flag & BOOTM_STATE_OS_CMDLINE)
5.                  return -1;
6.
7.          if (flag & BOOTM_STATE_OS_PREP) {
8.                  boot_prep_linux(images);
                                                                          ①
9.                  return 0;
10.         }
11.
12.         if (flag & (BOOTM_STATE_OS_GO | BOOTM_STATE_OS_FAKE_GO)) {
13.                 boot_jump_linux(images, flag);
                                                        ②
14.                 return 0;
15.         }
16.
17.         boot_prep_linux(images);
18.         boot_jump_linux(images, flag);
19.         return 0;
20. }
```

①

```
static void boot_prep_linux(bootm_headers_t *images)
{
        char *commandline = getenv("bootargs");

        if (IMAGE_ENABLE_OF_LIBFDT && images->ft_len) {
#ifdef CONFIG_OF_LIBFDT
                debug("using: FDT\n");
                if (image_setup_linux(images)) {
                        printf("FDT creation failed! hanging...");
                        hang();
                }
#endif
        } else if (BOOTM_ENABLE_TAGS) {
                debug("using: ATAGS\n");
                setup_start_tag(gd->bd);
                if (BOOTM_ENABLE_SERIAL_TAG)
                        setup_serial_tag(¶ms);
                if (BOOTM_ENABLE_CMDLINE_TAG)
                        setup_commandline_tag(gd->bd, commandline);
                if (BOOTM_ENABLE_REVISION_TAG)
                        setup_revision_tag(¶ms);
                if (BOOTM_ENABLE_MEMORY_TAGS)
                        setup_memory_tags(gd->bd);
                if (BOOTM_ENABLE_INITRD_TAG) {
                        if (images->rd_start && images->rd_end) {
                                setup_initrd_tag(gd->bd, images->rd_start,
                                                 images->rd_end);
                        }
                }
                setup_board_tags(¶ms);
                setup_end_tag(gd->bd);
        } else {
                printf("FDT and ATAGS support not compiled in - hanging\n");
                hang();
        }
        do_nonsec_virt_switch();
}
```

在enable FDT(Flattened Device Tree)的情况下，image_setup_linux()会对dtb做某些修改。这曾经让我很困惑。

我在dts中定义的boot parameter，但当到了Linux kernel中发觉竟然“文不对题”了。我当时完全不知道u-boot竟然会在暗中直接修改

dtb，所以以为是dtc生成的dtb有什么问题，或者dtb是对的，但在kernel处理dtb时做了什么手脚。以至于我花了很大精力去hack dtb的

二进制format，并在kernel刚接触dtb时，dump出整个dtb的device node，来确认问题到底出在哪儿。后来才发觉，kernel收到的dtb就

已经是“文不对题”了，而静态的dtb文件又是对的。那么只可能是u-boot在做手脚了！而做手脚的地方就是这里image_setup_linux() function。

```
1.  int image_setup_linux(bootm_headers_t *images)
2.  {
3.          ulong of_size = images->ft_len;
4.          char **of_flat_tree = &images->ft_addr;
5.          ulong *initrd_start = &images->initrd_start;
6.          ulong *initrd_end = &images->initrd_end;
7.          struct lmb *lmb = &images->lmb;
8.          ulong rd_len;
9.          int ret;
10.
11.         if (IMAGE_ENABLE_OF_LIBFDT)
12.                 boot_fdt_add_mem_rsv_regions(lmb, *of_flat_tree);
                                    (A)
13.
14.         if (IMAGE_BOOT_GET_CMDLINE) {
15.                 ret = boot_get_cmdline(lmb, &images->cmdline_start,
16.                                        &images->cmdline_end);

               (B)
17.                 if (ret) {
18.                         puts("ERROR with allocation of cmdline\n");
19.                         return ret;
20.                 }
21.         }
22.         if (IMAGE_ENABLE_RAMDISK_HIGH) {
23.                 rd_len = images->rd_end - images->rd_start;
24.                 ret = boot_ramdisk_high(lmb, images->rd_start, rd_len,
25.                                         initrd_start, initrd_end);
26.                 if (ret)
27.                         return ret;
28.         }
29.
30.         if (IMAGE_ENABLE_OF_LIBFDT) {
31.                 ret = boot_relocate_fdt(lmb, of_flat_tree, &of_size);
32.                 if (ret)
33.                         return ret;
34.         }
35.
36.         if (IMAGE_ENABLE_OF_LIBFDT && of_size) {
37.                 ret = image_setup_libfdt(images, *of_flat_tree, of_size, lmb);
        (C)
38.                 if (ret)
39.                         return ret;
40.         }
41.
42.         return 0;
43. }
```

(A)

在u-boot的lmb(Logic memory block)中标记dtb本身所占的space为"reserved“

(B)

如果定义了CONFIG_SYS_BOOT_GET_CMDLINE，则原来dts中定义的boot parameter完全无效，u-boot会从getenv("bootargs")中获得此参数。

(C)

image_setup_libfdt()就比较有趣了，它会根据u-boot中获得的具体板子(board)的信息去修改dtb了。

??? (另文分析)

②

从u-boot跳转到kernel的最后一步。