由于u-boot会relocate,所以用XDB debug u-boot也需要re-load symbol!

_start in arch/arm/cpu/armv7/start.S

|

|

\|/

ENTRY(_main) in arch/arm/lib/crt0.S

|

|

\|/

board_init_f() in arch/arm/lib/board.c

|

|

\|/

relocate_code() in arch/arm/lib/relocate.c

|

-----|----------分界线

\|/

board_init_r() in arch/arm/lib/board.c

|

|

\|/

.....

分界线以前的code运行在0x0800,0000 (128M)边界的space，而分界线以后的code运行在某个更高的address，比如在G2 LSP上是0x3FE8,7000(接近1G的边界，由于G2 LSP的physical memory就是1G,也就是u-boot会把自己搬移到可用physical memory的顶部去运行)

in arch/arm/lib/crt0.S

```
/*
 * Set up intermediate environment (new sp and gd) and call
 * relocate_code(addr_moni). Trick here is that we'll return
 * 'here' but relocated.
 */

        ldr     sp, [r9, #GD_START_ADDR_SP]     /* sp = gd->start_addr_sp */
        bic     sp, sp, #7      /* 8-byte alignment for ABI compliance */
        ldr     r9, [r9, #GD_BD]                /* r9 = gd->bd */
        sub     r9, r9, #GD_SIZE                /* new GD is below bd */

        adr     lr, here
        ldr     r0, [r9, #GD_RELOC_OFF]         /* r0 = gd->reloc_off */
        add     lr, lr, r0
        ldr     r0, [r9, #GD_RELOCADDR]         /* r0 = gd->relocaddr */
        b       relocate_code
                        ①
here:

/* Set up final (full) environment */

        bl      c_runtime_cpu_setup     /* we still call old routine here */

        ldr     r0, =__bss_start        /* this is auto-relocated! */
        ldr     r1, =__bss_end          /* this is auto-relocated! */

        mov     r2, #0x00000000         /* prepare zero to clear BSS */

clbss_l:cmp     r0, r1                  /* while not at end of BSS */
        strlo   r2, [r0]                /* clear 32-bit BSS word */
        addlo   r0, r0, #4              /* move to next */
        blo     clbss_l

        bl coloured_LED_init
        bl red_led_on

        /* call board_init_r(gd_t *id, ulong dest_addr) */
        mov     r0, r9                  /* gd_t */
        ldr     r1, [r9, #GD_RELOCADDR] /* dest_addr */
        /* call board_init_r */
        ldr     pc, =board_init_r       /* this is auto-relocated! */

        /* we should not return here. */
```

①

在此之前的code运行在0x0800,0000开始的space，而此之后的code运行在0x3FE8,7000开始的space。

这里r0 register包含的就是要把u-boot搬往的新地址0x3FE8,7000。

当从relocate_code()返回时就不是运行在原有的低地址空间了(0x0800,0000)，这样XDB的关于u-boot的symbol都已经不对了，原来设置的断点和将要基于原来symbol设置的断点都是错的。你会发觉这些断点永远不会hit。

in arch/arm/lib/relocate.S

```
/*
 * void relocate_code(addr_moni)
 *
 * This function relocates the monitor code.
 *
 * NOTE:
 * To prevent the code below from containing references with an R_ARM_ABS32
 * relocation record type, we never refer to linker-defined symbols directly.
 * Instead, we declare literals which contain their relative location with
 * respect to relocate_code, and at run time, add relocate_code back to them.
 */

ENTRY(relocate_code)

        ldr     r1, =__image_copy_start /* r1 <- SRC &__image_copy_start */
                ①
        subs    r4, r0, r1              /* r4 <- relocation offset */
        beq     relocate_done           /* skip relocation */
        ldr     r2, =__image_copy_end   /* r2 <- SRC &__image_copy_end */

copy_loop:
        ldmia   r1!, {r10-r11}          /* copy from source address [r1]    */
        stmia   r0!, {r10-r11}          /* copy to   target address [r0]    */
        cmp     r1, r2                  /* until source end address [r2]    */
        blo     copy_loop

        /*
         * fix .rel.dyn relocations
                                                ②
         */
        ldr     r2, =__rel_dyn_start    /* r2 <- SRC &__rel_dyn_start */
        ldr     r3, =__rel_dyn_end      /* r3 <- SRC &__rel_dyn_end */
fixloop:
        ldmia   r2!, {r0-r1}            /* (r0,r1) <- (SRC location,fixup) */
        and     r1, r1, #0xff
        cmp     r1, #23                 /* relative fixup? */
        bne     fixnext

        /* relative fix: increase location by offset */
        add     r0, r0, r4
        ldr     r1, [r0]
        add     r1, r1, r4
        str     r1, [r0]
fixnext:
        cmp     r2, r3
        blo     fixloop

relocate_done:

#ifdef __XSCALE__
        /*
         * On xscale, icache must be invalidated and write buffers drained,
         * even with cache disabled - 4.2.7 of xscale core developer's manual
```

```
51.              */
52.             mcr     p15, 0, r0, c7, c7, 0   /* invalidate icache */
53.             mcr     p15, 0, r0, c7, c10, 4  /* drain write buffer */
54.     #endif
55.
56.             /* ARMv4- don't know bx lr but the assembler fails to see that */
57.
58.     #ifdef __ARM_ARCH_4__
59.             mov        pc, lr
60.     #else
61.             bx        lr
62.     #endif
63.
64.     ENDPROC(relocate_code)
```

①

__image_copy_start和__image_copy_end定义在u-boot.lds中。

copy u-boot code.

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
 . = 0x00000000;
 . = ALIGN(4);
 .text :
 {
  *(.__image_copy_start)
  arch/arm/cpu/armv7/start.o (.text*)
  *(.text*)
 }
 . = ALIGN(4);
 .rodata : { *(SORT_BY_ALIGNMENT(SORT_BY_NAME(.rodata*))) }
 . = ALIGN(4);
 .data : {
  *(.data*)
 }
 . = ALIGN(4);
 . = .;
 . = ALIGN(4);
 .u_boot_list : {
  KEEP(*(SORT(.u_boot_list*)));
 }
 . = ALIGN(4);
 .image_copy_end :
 {
  *(.__image_copy_end)
 }
 .rel_dyn_start :
 {
  *(.__rel_dyn_start)
 }
 .rel.dyn : {
  *(.rel*)
 }
 .rel_dyn_end :
 {
  *(.__rel_dyn_end)
 }
 _end = .;
 . = ALIGN(4096);
 .mmutable : {
  *(.mmutable)
 }
 .bss_start __rel_dyn_start (OVERLAY) : {
  KEEP(*(.__bss_start));
  __bss_base = .;
 }
 .bss __bss_base (OVERLAY) : {
  *(.bss*)
   . = ALIGN(4);
```

```
    __bss_limit = .;
  }
  .bss_end __bss_limit (OVERLAY) : {
   KEEP(*(.__bss_end));
  }
  .dynsym _end : { *(.dynsym) }
  .dynbss : { *(.dynbss) }
  .dynstr : { *(.dynstr*) }
  .dynamic : { *(.dynamic*) }
  .plt : { *(.plt*) }
  .interp : { *(.interp*) }
  .gnu : { *(.gnu*) }
  .ARM.exidx : { *(.ARM.exidx*) }
  .gnu.linkonce.armexidx : { *(.gnu.linkonce.armexidx.*) }
 }
```

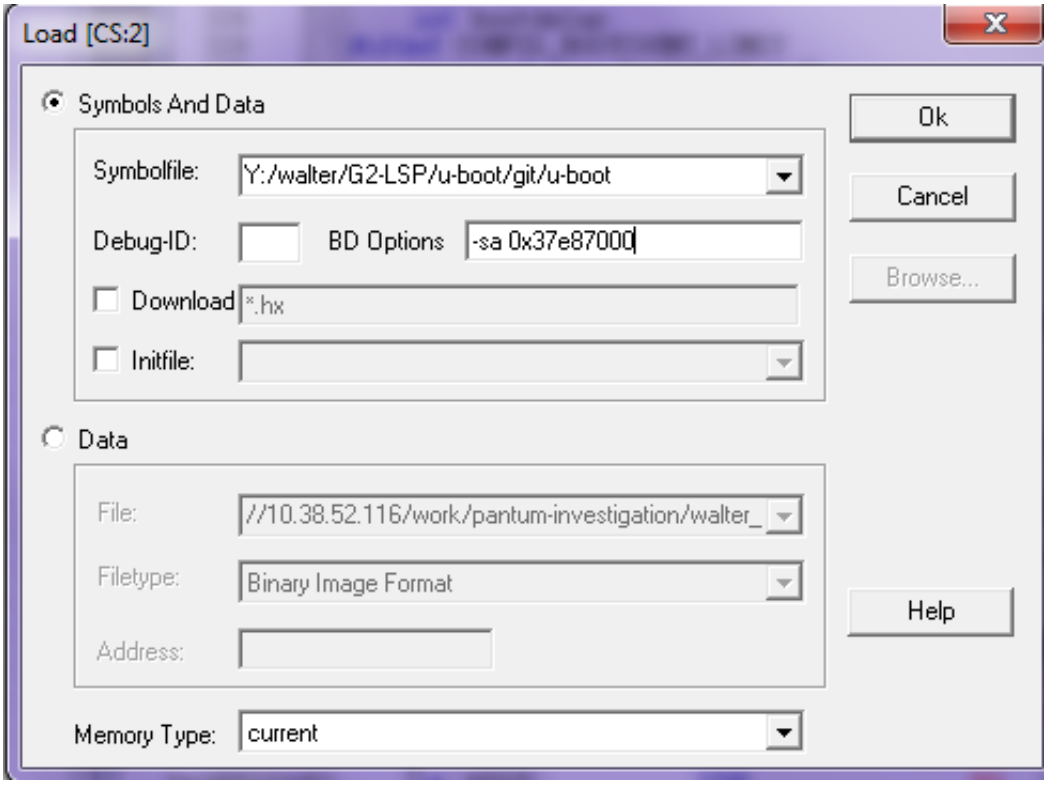
②

The moved u-boot data needs to fixup. The fixup is 0x37E8，7000 (0x3FE8,7000 - 0x0800,0000)

③

这条指令一执行就到新地址去执行了。

在执行③处的指令前，最好重新载入u-boot的symbol，这样XDB就可以认识在新地址运行的u-boot code了。



在BD Options中输入新地址与旧地址之间的offset, 0x3FE8,7000 - 0x0800,0000 = 0x37e87000