upc_uart.c用来verify UPC module的interrupt handling.但运行总是很诡异。

root cause:

```
/opt/armv7-marvell-linux-gnueabi-hard-4.6.4_x86_64_20140402/bin/arm-marvell-linux-gnueabi-objdump -d upc_uart.elf
```

```
upc_uart.elf:     file format elf32-littlearm



Disassembly of section .text:

00000000 <main>:
   0:   e92d4030        push    {r4, r5, lr}
   4:   e24dd014        sub     sp, sp, #20
   8:   eb000038        bl      f0 <test_init>
   c:   e3a03055        mov     r3, #85 ; 0x55
  10:   e5cd3004        strb    r3, [sp, #4]
  14:   e3a03050        mov     r3, #80 ; 0x50
  18:   e5cd3005        strb    r3, [sp, #5]
  1c:   e3a03043        mov     r3, #67 ; 0x43
  20:   e5cd3006        strb    r3, [sp, #6]
  24:   e3a03020        mov     r3, #32
  28:   e5cd3007        strb    r3, [sp, #7]
  2c:   e3a03054        mov     r3, #84 ; 0x54
  30:   e5cd3008        strb    r3, [sp, #8]

  34:   e3a03065        mov     r3, #101        ; 0x65

  38:   e5cd3009        strb    r3, [sp, #9]

  3c:   e3a03073        mov     r3, #115        ; 0x73

  40:   e5cd300a        strb    r3, [sp, #10]

  44:   e3a03074        mov     r3, #116        ; 0x74

  48:   e5cd300b        strb    r3, [sp, #11]

  4c:   e3a0300d        mov     r3, #13
```

......

0000064c <entry>:

```
 64c:   ea000006        b       66c <ResetEntry>


00000650 <Undefined_ISR>:

 650:   eafffffe        b       650 <Undefined_ISR>

 654:   ea000027        b       6f8 <SWI_ISR>


00000658 <Prefetch_ISR>:

 658:   eafffffe        b       658 <Prefetch_ISR>


0000065c <Abort_ISR>:

 65c:   eafffffe        b       65c <Abort_ISR>


00000660 <Reserved_ISR>:
 660:   eafffffe        b       660 <Reserved_ISR>
 664:   ea000027        b       708 <IRQ_ISR>
 668:   ea00002a        b       718 <FIQ_ISR>

0000066c <ResetEntry>:
 66c:   e59f10b4        ldr     r1, [pc, #180]  ; 728 <FIQ_ISR+0x10>
 670:   e1a0d001        mov     sp, r1
 674:   e10f0000        mrs     r0, CPSR
 678:   e3e0101f        mvn     r1, #31
 67c:   e0000001        and     r0, r0, r1
 680:   e3800012        orr     r0, r0, #18
 684:   e3c00080        bic     r0, r0, #128    ; 0x80
```

main()竟然被放在0地址，而本来该在0地址的entry却到了别的地方?!

in preboot_fm.s

```
.section .text

entry:
        b       ResetEntry
Undefined_ISR:
        b       Undefined_ISR
        b       SWI_ISR
```

```
Prefetch_ISR:
        b       Prefetch_ISR
Abort_ISR:
        b       Abort_ISR
Reserved_ISR:
        b       Reserved_ISR
        b       IRQ_ISR
        b       FIQ_ISR

/*********************************************************************
* Boot entry                                                        *
*********************************************************************/

ResetEntry:
        /* Init SVC Stack */
        ldr     r1, =SVCstack_FM
        mov     sp,r1

        /* Init IRQ Stack and Enable IRQ*/
        mrs     r0,cpsr
        ldr     r1, =0xffffffe0
        and     r0,r0,r1
        orr     r0,r0,#0x12
        bic     r0,r0,#0x80
        msr     cpsr,r0
```

```
ldr     r1, =IRQstack_FM

mov     sp,r1


/* Init FIQ Stack and Enable FIQ*/

mrs     r0,cpsr

ldr     r1, =0xffffffe0

and     r0,r0,r1

orr     r0,r0,#0x11

bic     r0,r0,#0x40

msr     cpsr_c,r0

ldr     r1, =FIQstack_FM

mov     sp,r1


/* Init User Stack and jump to main() */

mrs     r0,cpsr

ldr     r1, =0xffffffe0

and     r0,r0,r1

orr     r0,r0,#0x1f

msr     cpsr_c,r0

ldr     r1, =MainStack_FM

mov     sp, r1

bl      main


/* Store Results */

cmp     r0, #0
```

```
        ldr     r1, =0x900d0000
        orreq   r0, r1
        ldr     r1, =0x0bad0000
        orrne   r0, r1
        ldr     r1, =FM_Result
        str     r0, [r1]
        mcr     p15, 0, r0, c7, c0, 4

Should_not_get_here:
        b       Should_not_get_here
```

代码本意是当ARM core reset后，从0地址开始执行。entry应该位于0地址。

从entry跳转到ResetEntry,设置SVC mode stack, IRQ mode stack, FIQ mode stack，User mode stack，然后跳转到C语言的main()。

但由于link的参数没设置对，造成code的layout没有按照预想的来。由于main() function addree位于0地址，所以ARM core reset后，直接就运行main(),但运行环境没设置好，所以运行起来以后就什么诡异的现象都发生了。

upc_wait.c也面临相同的问题！

Question:

Why upctalk0.elf and upctalk1.elf could work？

Answer:

upctalk0.elf(upctalk1.elf)的memory layout同样错了。

```
$ $ arm-linux-gnueabi-objdump -d upctalk0.elf

upctalk0.elf：     文件格式 elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0:   e92d4070        push    {r4, r5, r6, lr}
   4:   e59f5048        ldr     r5, [pc, #72]   ; 54 <version_major+0x54>
   8:   e59f0048        ldr     r0, [pc, #72]   ; 58 <version_major+0x58>
   c:   eb000016        bl      6c <print>
  10:   e3a04000        mov     r4, #0
  14:   e3a06301        mov     r6, #67108864   ; 0x4000000
  18:   e5963000        ldr     r3, [r6]
  1c:   e1530004        cmp     r3, r4
  20:   1a000001        bne     2c <version_major+0x2c>
  24:   e59f0030        ldr     r0, [pc, #48]   ; 5c <version_major+0x5c>
  28:   eb00000f        bl      6c <print>
  2c:   e2844001        add     r4, r4, #1
  30:   e1540005        cmp     r4, r5
  34:   1afffff7        bne     18 <version_major+0x18>
  38:   e3a03301        mov     r3, #67108864   ; 0x4000000
  3c:   e3a02042        mov     r2, #66 ; 0x42
  40:   e5832000        str     r2, [r3]
  44:   e5932000        ldr     r2, [r3]
  48:   e3520056        cmp     r2, #86 ; 0x56
  4c:   1afffffc        bne     44 <version_major+0x44>
  50:   eaffffec        b       8 <main+0x8>
  54:   00989680        .word   0x00989680
  58:   000003d8        .word   0x000003d8
  5c:   000003ef        .word   0x000003ef
```

c语言的main() function被放在了0地址。这样当UPC ARM core被reset后，直接从main开始执行了。

```
int main(void)
{
   int i,k;
   volatile uint32_t *upc_done = (uint32_t *)0x04000000; // the main cpu sees this at 0xF9280000

   do {
      print("UPC0 is now running...");

      for(i = 0; i < 10000000; i++) {
         if((k = *upc_done) == i) {
            print("\r\n");
         }
```

```
        }

        *upc_done = 0x42; // tell UPC1 we are done

        while(*upc_done != 0x56); // now wait here until UPC1 is done
    } while(1);
}
```

该测试太简单了，就是无限循环的执行上面的code，并不需要其他运行环境，比如象upc_uart.c中的interrupt stack等。

所以upctalk0.elf和upctalk1.elf也只是看上去象正确运行了。

root cause:

upc_uart.elf由upc_uart.o, dbg_printf.o and preboot_fm.o链接而成。由于

in Makefile

```
# Link IO .o's into an ELF object file
define ELF_IO_PRGM_MACRO
$(1): $(basename $(1)).o $(TGT_EXTRA_OBJ)
        @echo $(MSG_LD) $$@
        $(V1) $(LD) $(CROSS_IO_LDFLAGS) -o $$@ $$^
endef
```

这里.o文件的次序不是按memory layout的需求安排的。

in temp/log.do_compile (yocto)

arm-poky-linux-gnueabi-ld -entry=0x0 -Ttext 0x0 -EL -Tdata 0x0401E000 -Tbss 0x0401E000 -o upc_uart.elf **upc_uart.o preboot_fm.o preboot_io.o dbg_printf.o**

这里有2个问题：

1. .o文件的次序不对。如果不在linker script中指定生成elf文件的memory layout，ld将按照.o出现的次序来安排lauout。这造成upc_uart.o的code放在meemory layout的最前面，也就是0地址。

正确的方式有两种

① 用linker script来定义code的layout

② 偷懒的方法是重新安排.o链接的次序

arm-poky-linux-gnueabi-ld -entry=0x0 -Ttext 0x0 -EL -Tdata 0x0401E000 -Tbss 0x0401E000 -o upc_uart.elf **preboot_fm.o upc_uart.o dbg_printf.o**

preboot_fm.o必须放在最前面，因为它包含的是启动code，必须位于0地址开始。

2. 如果upc_uart.elf在upc0上运行，则没有必要链接preboot_io.o

如果测试是这样的

# cat upc_uart.bin > /dev/upc0

# cat upc_wait.bin > /dev/upc1

即在upc0(Formater ARM core of upc)上运行upc_uart.bin，在upc1(IO ARM core of upc)上运行upc_wait.bin，则preboot_fm.o应该与upc_uart.o链接，而preboot_io.o与upc_wait.o链接。当然由于实际上Formater ARM core和IO ARM core几乎是等价的，所以

反之也行，但完全没必要也不应该把preboot_fm.o与preboot_io.o都链接进来。

walterzh$ /home/walterzh/gerrit/build-bundle/poky/build/tmp/sysroots/x86_64-linux/usr/bin/armv7a-vfp-neon-poky-linux-gnueabi/arm-poky-linux-gnueabi-ld -entry=0x0 -Ttext 0x0 -EL -Tdata 0x0401E000 -Tbss 0x0401E000 -o upc_uart.elf **preboot_fm.o upc_uart.o dbg_printf.o**

用objdump验证一下

```
walterzh$ /home/walterzh/gerrit/build-bundle/poky/build/tmp/sysroots/x86_64-linux/usr/bin/armv7a-vfp-neon-poky-linux-gnueabi/arm-poky-linux-gnueabi-objdump -d upc_uart.elf


upc_uart.elf:     file format elf32-littlearm



Disassembly of section .text:


00000000 <entry>:
   0:   ea000006        b       20 <ResetEntry>


00000004 <Undefined_ISR>:
   4:   eafffffe        b       4 <Undefined_ISR>
   8:   ea000027        b       ac <SWI_ISR>


0000000c <Prefetch_ISR>:
   c:   eafffffe        b       c <Prefetch_ISR>


00000010 <Abort_ISR>:
  10:   eafffffe        b       10 <Abort_ISR>


00000014 <Reserved_ISR>:
  14:   eafffffe        b       14 <Reserved_ISR>
  18:   ea000027        b       bc <IRQ_ISR>

  1c:   ea00002a        b       cc <FIQ_ISR>


00000020 <ResetEntry>:

  20:   e59f10b4        ldr     r1, [pc, #180]  ; dc <FIQ_ISR+0x10>

  24:   e1a0d001        mov     sp, r1

  28:   e10f0000        mrs     r0, CPSR

  2c:   e3e0101f        mvn     r1, #31

  30:   e0000001        and     r0, r0, r1

  34:   e3800012        orr     r0, r0, #18

  38:   e3c00080        bic     r0, r0, #128    ; 0x80

  3c:   e129f000        msr     CPSR_fc, r0

  40:   e59f1098        ldr     r1, [pc, #152]  ; e0 <FIQ_ISR+0x14>
```

entry确实位于0地址了。