socfpga 裸机linux amp双核通信

一、环境搭建

1.安装工具链和配置环境变量

1.1 bm

bm的编译环境是搭建在windows中的，**make和交叉编译工具链(包含库)**。

- **安装make** 将make.exe放在C:\windows\中，然后在cmd命令行中就可以不加路径名直接使用。
- **安装交叉编译工具链** 将arm-2012.03-56-arm-none-eabi.exe安装到C:/Program\ Files\ (x86)/CodeSourcery/Sourcery_CodeBench_Lite_for_ARM_EABI目录下，因为bm的makefile中指定了工具路径名就是它。

```
BUILD_TOOL_ROOT := C:/Program\ Files\
(x86)/CodeSourcery/Sourcery_CodeBench_Lite_for_ARM_EABI
```

1.2 uboot

ubuntu自带make，因此只需要**安装交叉编译工具链并配置好环境变量**即可完成uboot的编译环境搭建。

- **安装交叉编译工具链** 将gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux.tar.bz2解压到/opt目录。
- **配置环境变量** 在/etc/profile文件中添加如下命令。

```
export PATH=/opt/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/:$PATH
```

1.2 kernel

amp中的kernel和uboot使用同一个交叉编译工具链，因此搭建好了uboot的编译环境就等于搭建好了kernel的编译环境。

2. 制作和烧写包括所有部分的完整的sd卡镜像

2.1 制作合成镜像sd_image_bm.bin

在包含所有镜像的MKIMAGE目录下执行./make.sh，make.sh内容如下。

```
sudo ./make_sdimage_bm.sh -k uImage,socfpga.dtb -p preloader.bin -b u-boot.img -a bm.bin -f bm.rbf -r /home/hn/TEST/MKIMAGE/rootfs/ -o sd_image_bm.bin
```

2.2 将合成镜像sd_image_bm.bin烧写到sd卡

- **先用parted格式化sd卡**

  参考格式化SD卡制作启动项.txt。

- **烧写镜像到sd卡**

  打开windows虚拟机，先将sd_image_bm.bin传到windows虚拟机种，然后将格式化好的sd卡插上，然后通过windisker32将sd_image_bm.bin烧写到sd卡，然后得到了分区格式化并放好内容可直接启动的sd卡。

3.编译和替换镜像

3.1 bm

- **编译bm** 打开windows的cmd，然后cd到bm源码目录，make clean和make。
- **替换bm（bm.bin）** 将sd卡插到ubuntu虚拟机，然后输入如下命令。

```
/* sdb不是确定的，要先看sd卡是不是对应的设备节点 */
sudo dd if=bm.bin of=/dev/sdb3 bs=1M seek=1;sudo sync
```

- **替换fpga（bm.rbf）** 将sd卡插到ubuntu虚拟机，然后输入如下命令。

```
/* sdb不是确定的，要先看sd卡是不是对应的设备节点 */
sudo dd if=bm.rbf of=/dev/sdb3 bs=2M seek=1;sudo sync
```

3.2 uboot

- **配置编译uboot**

  在编译uboot之前先切换为root用户。

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean
/* 配置 */
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- socfpga_cyclone5_config
/* 编译uboot和spl */
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all
/* 通过u-boot-spl.bin制作preloader.bin */
cd spl
./mkimage.sh
```

- **替换spl (preloader.bin)**

  ```
  sudo dd if=preloader.bin of=/dev/sdb3 bs=64k seek=0;sudo sync
  ```

- **替换uboot (u-boot.img)**

  ```
  sudo dd if=u-boot.img of=/dev/sdb3 bs=64k seek=4;sudo sync
  ```

3.3 kernel

- **配置编译内核和设备树**

在编译kernel之前先切换为root用户。并且原来的内核配置有问题，之前通过分析定位到是少了CONFIG_UNINLINE_SPIN_UNLOCK=y的原因，因此可以通过修改socfpga_defconfig添加CONFIG_UNINLINE_SPIN_UNLOCK=y或者用有漏洞的socfpga_defconfig先make socfpga_defconfig得到一个.config，然后在这个.config中添加CONFIG_UNINLINE_SPIN_UNLOCK=y最后保存为一个名为socfpga_config的文件，以后编译前的配置不再使用通常的make socfpga_defconfig而是cp socfpga_config .config。

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean
/* 配置 */
#make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0x8000 socfpga_defconfig
cp  socfpga_config .config
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0x8000 menuconfig
/* 编译内核镜像和设备树镜像 */
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0x8000 uImage -j4
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0x8000 dtbs -j4
```

- **替换内核和设备树镜像**

  内核和设备树镜像在sd卡方式中无法单独替换(没有提供)，因此需要通过上面的2.1和2.2重新制作和烧写完整的sd卡。

二、uboot改造

uboot启动过程参考文档U-BOOT配置编译过程.pdf和U-BOOT启动过程.pdf。

1.修改文件列表

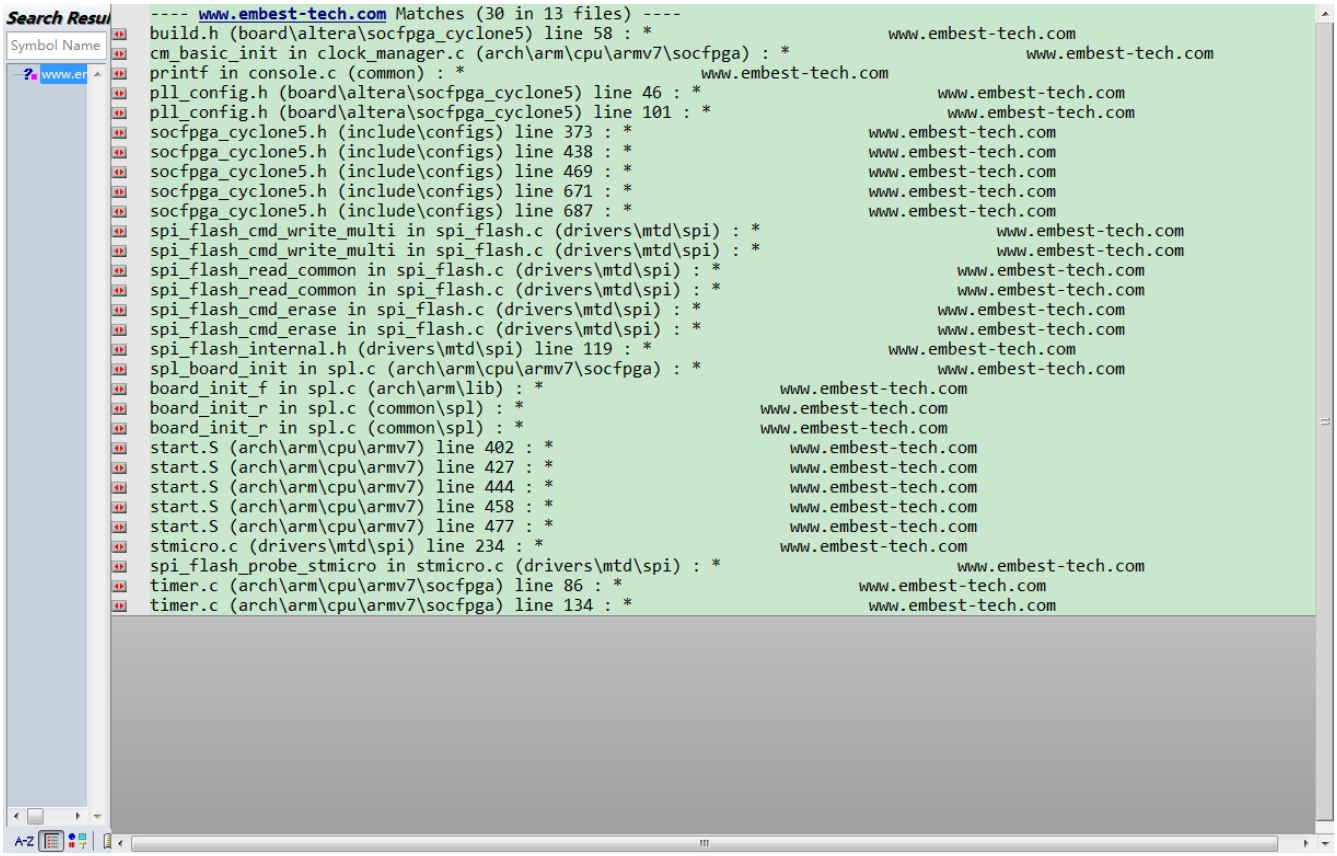
下面直接看amp中修改的地方。下面的表格就是amp的uboot（改了一些spl和uboot共有部分和其他的spl私有部分）修改的文件列表。

No.	Location
1	Board/altera/socfpga_cyclone5/build.h(spl for sd/qspi, and remove the serial support for spl)
2	Include/configs/socfpga_cyclone5.h(for qspi and sd)
3	Arch/arm/include/ams/arch-socfpga/amp_config.h(exportingspl function to bm etc.)
4	Arch/arm/cpu/armv7/start.S (for boot start time stamp)
5	Arch/arm/cpu/armv7/osc_timer1.S (the osc timer1 for calculating boot time for bm)
6	Common/spl/spl.c (load bm for sd and qspi)
7	Board/altera/socfpga_cyclone5/socfpga_cyclone5.c(boot bm)
8	Arch/arm/cpu/armv7/socfpga/clock_manager.c(delay using udelay function about 7us,not 7ms)
9	Arch/arm/cpu/armv7/socfpga/spl.c(don't reset the osc timer1)
10	Drivers/mtd/spi/spi_spl_load.c(load bm from the qspi)
11	Drivers/spi/cadence_qspi.c,cadence_qspi_apb.c,Drivers/mtd/spi/spi_flash.c(update for qspi driver)
12	Drivers/mmc/spl_mmc.c(load bm from the sd card)
13	Arch/arm/include/asm/arch-socfgpa(dwmmc.h),Drivers/mmc/altera_dw_mmc.c,dw_mmc.c(upate for sd/mmc driver)

其实上面的表格并没有列出所有的修改和更新过的文件，在源码中一般被amp修改过的地方会有如下注释：

```
********************************************************************************
*                        Embest Tech co., ltd
*                         www.embest-tech.com
********************************************************************************
```

在sourceinsight中ctrl+/搜索www.embest-tech.com可以得到如下：



2.主要修改部分

2.1 主干

从上文看出uboot(主要是spl)修改的文件很多，但我们只关注amp主要部分，以下是amp方案的关键核心修改路径，核心目的是**加载裸机固件并启动1核运行它**。

```
void board_init_r(gd_t *dummy1, ulong dummy2)
{
    u32 boot_device;
    ...
    boot_device = spl_boot_device();    //获取boot device类型(BOOT_DEVICE_MMC1)
    debug("boot device - %d\n", boot_device);

    /* 检测启动存储介质，然后从对应的存储介质中加载镜像 */
    switch (boot_device) {
    ...
#ifdef CONFIG_SPL_MMC_SUPPORT
    case BOOT_DEVICE_MMC1:
    case BOOT_DEVICE_MMC2:
    case BOOT_DEVICE_MMC2_2:
/*
********************************************************************************
*                       Embest Tech co., ltd
*                        www.embest-tech.com
********************************************************************************
*
* It will load bm.bin first when boot from the tf card.
```

```
*
*/      /* 0x1E100000,share parameters memory region for u-boot, bare metal, linux */
        amp_share_param_init();     //将amp共享区域清0
        /* get the start time stamp of loading bm.bin */
        load_bm_start();            //获取加载bm.bin的开始时间
        /* init sd/mmc,and look if there is a sd/mmc device,
          return a pointer to the device descritor */
        spl_mmc_probe();            //初始化sd/mmc控制器和设备
        /* load bm.bin to sdram from sd/mmc device */
        spl_mmc_load_bm_image_mbr();    ///将bm.bin从sd/mmc加载到sdram中
        /* get the end time stamp of loading bm.bin */
        load_bm_end();              //获取加载bm.bin的结束时间
        /* store spl_mmc_probe and some functions for reuse by bm */
        spl_mmc_save_func();  //保存sd/mmc相关的函数到共享内存给bm使用
        /* boot cpu1 to run bm.bin */
        boot_bm_on_cpu1();      //从cpu1启动bm
        /* wait for cpu1 completing config bm.rbf */
        cpu0_wait_cpu1_load_rbf();  //等待cpu1中运行的bm加载rbf完成
        /* load u-boot */
        spl_mmc_load_image();   //将u-boot镜像加载到ddr中

        break;
#endif
    ...
    default:
        debug("SPL: Un-supported Boot Device\n");
        hang();
    }

    /* 检查是什么镜像然后到相应的函数跳转 */
    switch (spl_image.os) {     //spl_image的域在spl_mmc_load_image()里填充
    case IH_OS_U_BOOT:
        debug("Jumping to U-Boot\n");
        break;
#ifdef CONFIG_SPL_OS_BOOT   //如果配置了spl直接启动内核
    case IH_OS_LINUX:
        debug("Jumping to Linux\n");
        spl_board_prepare_for_linux();
        jump_to_image_linux((void *)CONFIG_SYS_SPL_ARGS_ADDR);
#endif
    default:
        debug("Unsupported OS image.. Jumping nevertheless..\n");
    }

    /* 如果不是linux镜像就到这里跳转到启动镜像 */
    jump_to_image_no_args();
}
```

2.2 分支

未完待续

三、bm开发

见代码包bm_amp。

四、kernel改造

amp kernel修改的文件列表如下，另外还新增一个**amp.c**在目录drivers/char中，是测试双核通信的字符驱动。内核的修改主要有smp、中断和内存映射三部分，在下文描述。

No.	Location
1	Arch/arm/mach-socfpga/socfpga.c (memblock reserved for BM)
2	Arch/arm/command/gic.c (don't make any change to BM-relatedIRQs property)
3	Arch/arm/kernel/smp.c (exporting kernel function to bm etc.)
4	Arch/arm/mm/mmu.c (creating fixed mapping for amp)
5	Arch/arm/include/asm/pgtable.h (adjusting vmalloc region and allocating reserved region to amp)
6	Arch/arm/include/asm/amp_config.h (defining amp parameters)
7	Arch/arm/mm/cache-l2x0.c (locking l2ways by master, and making Linux don'ttouch to BM ways)

1.内存部分改造

参考文档amp linux 内存.pdf。

2.中断部分改造

参考文档amp linux 中断.pdf。

3.smp部分改造

3.1 0核smp初始化

- **smp第一部分初始化：打开所有cpu位图的第0位**

> start_kernel() --> boot_cpu_init()

```
static void __init boot_cpu_init(void)
{
    int cpu = smp_processor_id();
    /* Mark the boot cpu "present", "online" etc for SMP and UP case */
    set_cpu_online(cpu, true);
    set_cpu_active(cpu, true);
    set_cpu_present(cpu, true);
    set_cpu_possible(cpu, true);
}
```

```
/* 定义四个位图 */
static DECLARE_BITMAP(cpu_possible_bits, CONFIG_NR_CPUS) __read_mostly;
const struct cpumask *const cpu_possible_mask = to_cpumask(cpu_possible_bits);
EXPORT_SYMBOL(cpu_possible_mask);
```

```
static DECLARE_BITMAP(cpu_online_bits, CONFIG_NR_CPUS) __read_mostly;
const struct cpumask *const cpu_online_mask = to_cpumask(cpu_online_bits);
EXPORT_SYMBOL(cpu_online_mask);

static DECLARE_BITMAP(cpu_present_bits, CONFIG_NR_CPUS) __read_mostly;
const struct cpumask *const cpu_present_mask = to_cpumask(cpu_present_bits);
EXPORT_SYMBOL(cpu_present_mask);

static DECLARE_BITMAP(cpu_active_bits, CONFIG_NR_CPUS) __read_mostly;
const struct cpumask *const cpu_active_mask = to_cpumask(cpu_active_bits);
EXPORT_SYMBOL(cpu_active_mask);

static DECLARE_BITMAP(cpu_online_bits, CONFIG_NR_CPUS) __read_mostly;
const struct cpumask *const cpu_online_mask = to_cpumask(cpu_online_bits);
EXPORT_SYMBOL(cpu_online_mask);

/* 定义一个名字为name的包含bits个位的位图 */
#define DECLARE_BITMAP(name,bits) \
    unsigned long name[BITS_TO_LONGS(bits)]

/* 将位图bitmap从unsigned long数组转化为(struct cpumask *) */
#define to_cpumask(bitmap)                      \
    ((struct cpumask *)(1 ? (bitmap)                \
                : (void *)sizeof(__check_is_bitmap(bitmap))))

/* 该结构体包含固定大小的位图数组 */
typedef struct cpumask { DECLARE_BITMAP(bits, NR_CPUS); } cpumask_t;
```

- **设置smp_ops，并调用smp_ops.smp_init_cpus()**

> start_kernel() --> setup_arch() --> smp_set_ops()、smp_init_cpus()

```
void __init setup_arch(char **cmdline_p)
{
    ....
#ifdef CONFIG_SMP
    if (is_smp()) {
        smp_set_ops(mdesc->smp);    /* 设置smp_ops */
        smp_init_cpus();            /* 调用socfpga_smp_init_cpus */
    }
#endif
    ....
}
```

```
void __init smp_set_ops(struct smp_operations *ops)
{
    if (ops)
        smp_ops = *ops;
};
```

```
void __init smp_init_cpus(void)
{
    if (smp_ops.smp_init_cpus)  //socfpga_smp_init_cpus
        smp_ops.smp_init_cpus();
}

struct smp_operations socfpga_smp_ops __initdata = {
    .smp_init_cpus      = socfpga_smp_init_cpus,
    .smp_prepare_cpus   = socfpga_smp_prepare_cpus,
    /* 在amp方案中不会被调用，因为下面的socfpga_boot_secondary没有使用内核的启动复核的代码 */
    .smp_secondary_init = socfpga_secondary_init,
    .smp_boot_secondary = socfpga_boot_secondary,
#ifdef CONFIG_HOTPLUG_CPU
    .cpu_die        = socfpga_cpu_die,
#endif
};
```

- **smp第二部分初始化：获取硬件smp的cpu核数，然后和软件配置的cpu核数比较，取最小值来设置possible位图，并设置smp cross函数为gic_raise_softirq**

> start_kernel() --> setup_arch() --> smp_init_cpus()

```
/*
 * Initialise the CPU possible map early - this describes the CPUs
 * which may be present or become present in the system.
 */
static void __init socfpga_smp_init_cpus(void)
{
    unsigned int i, ncores;

    /* 获取开启了硬件smp的cpu核数 */
    ncores = scu_get_core_count(socfpga_scu_base_addr);

    /* 将所有开启了硬件smp的核的possible位置1 */
    for (i = 0; i < ncores; i++)
        set_cpu_possible(i, true);

    /* sanity check */
    /* 如果开启硬件smp的cpu核数大于软件配置的核数
     * 则将ncores设置为软件配置的核数
    */
    if (ncores > num_possible_cpus()) {
        pr_warn("socfpga: no. of cores (%d) greater than configured"
            "maximum of %d - clipping\n", ncores, num_possible_cpus());
        ncores = num_possible_cpus();
    }

    /* 重新根据软件配置的核数设置possible位图 */
    for (i = 0; i < ncores; i++)
```

```
        set_cpu_possible(i, true);

    /* 设置smp cross操作函数   */
    set_smp_cross_call(gic_raise_softirq);
}
```

- **smp第三部分初始化：为除了启动核之外每个possible的cpu创建idle线程（但这一步在amp中没有意义，个人认为应该替换为空函数），然后通过cpu_up做启动1核的工作（实际上1核已经由spl启动，这里主要是设置asp）。**

> start_kernel() --> rest_init() --> kernel_init() --> kernel_init_freeable() --> smp_init()

```
void __init smp_init(void)
{
    unsigned int cpu;

    idle_threads_init();    /* 为除了启动核之外每个possible的cpu创建idle线程 */

    /* FIXME: This should be done in userspace --RR */
    for_each_present_cpu(cpu) {
        if (num_online_cpus() >= setup_max_cpus)
            break;
        /*
         * 如果是cpu的online位图已经置1，则不会调用cpu_up(cpu)
         * 说明不会cpu_up(0)，但会cpu_up(1)且是在0核运行
         */
        if (!cpu_online(cpu))
            cpu_up(cpu);    /* smp路 */
    }

    /* Any cleanup work */
    printk(KERN_INFO "Brought up %ld CPUs\n", (long)num_online_cpus());
    smp_cpus_done(setup_max_cpus);
}
```

3.2 0核启动1核

需要事先声明，在未改造的内核中，1核是由smp内核部分启动，在amp方案中由spl启动，因此内核就不再启动1核，所以要将smp中启动1核的socfpga_boot_secondary替换为空函数。

> cpu_up(1) --> _cpu_up(1, 0) --> __cpu_up(1, idle)

```
/* 在_cpu_up中调用 */
int __cpuinit __cpu_up(unsigned int cpu, struct task_struct *idle)
{
    int ret;
    /*
     * We need to tell the secondary core where to find
     * its stack and the page tables.
```

```
     */
    secondary_data.stack = task_stack_page(idle) + THREAD_START_SP;
    secondary_data.pgdir = virt_to_phys(idmap_pgd);
    secondary_data.swapper_pg_dir = virt_to_phys(swapper_pg_dir);
    __cpuc_flush_dcache_area(&secondary_data, sizeof(secondary_data));
    outer_clean_range(__pa(&secondary_data), __pa(&secondary_data + 1));
    //printk("idmap_pgd = 0x%08x, swapper_gp_dir = 0x%08x\n", (u32)idmap_pgd,
(u32)swapper_pg_dir);
    /*
     * Now bring the CPU into our world.
     */
    /*
    ********************************************************************************
    *                           Embest Tech co., ltd
    *                           www.embest-tech.com
    ********************************************************************************
    */
    amp_init();     /* 初始化asp */
    ret = boot_secondary(cpu, idle);     /* 替换成了返回-1的空函数 */

    /* 因为返回-1所以以下部分不执行，这是smp的，目的是等待cpu1启动，但amp中已经由spl启动了 */
    if (ret == 0) {
        /*
         * CPU was successfully started, wait for it
         * to come online or time out.
         */
        wait_for_completion_timeout(&cpu_running,
                            msecs_to_jiffies(1000));

        if (!cpu_online(cpu)) {
            pr_crit("CPU%u: failed to come online\n", cpu);
            ret = -EIO;
        }
    } else {
        pr_err("CPU%u: failed to boot: %d\n", cpu, ret);
    }

    secondary_data.stack = NULL;
    secondary_data.pgdir = 0;

    return ret;
}
```

```
/*
********************************************************************************
*                           Embest Tech co., ltd
*                           www.embest-tech.com
********************************************************************************
*/
struct amp_share_param *asp;
EXPORT_SYMBOL(asp);
void amp_init(void)
{
```

```
    int i,tgcpu;
    asp = (struct amp_share_param *)AMP_SHARE_PARAM_START;
        //memset(asp, 0, sizeof(struct amp_share_param));
    /* 初始化共享函数 */
    asp->sta.printk_fn = (u32)printk;
    asp->sta.spinlock_trylock_fn = (u32)_raw_spin_trylock;
    asp->sta.spinlock_lock_fn  = (u32)_raw_spin_lock;
    asp->sta.spinlock_unlock_fn = (u32)_raw_spin_unlock;
    asp->sta._down_trylock_fn = (u32)down_trylock;
    asp->sta._up_fn          = (u32)up;
    /* 初始化共享自旋锁 */
    for(i = 0; i < RAW_SPIN_LOCK_OBJECT_NR; i++)
        asp->rslocks[i] = __RAW_SPIN_LOCK_UNLOCKED(rslocks);
    /* 初始化共享信号量 */
    for(i = 0; i < 16; i++)
        sema_init(&asp->semobj[i], 1);
    /* 发送sgi8给cpu1让cpu1锁定l2中的bm部分*/
     tgcpu = 1;
     gic_raise_softirq(cpumask_of(tgcpu), SGI_BRING_OTHER2CACHECOHERENCE);//yejc
}
```

五、疑难点

- sgi的虚拟中断号为什么是sgi硬件中断号+512

  答：因为在early_irq_init()中预先分配保留512个desc。然后再额外分配设置和gic直接连接和支持的所有中断的desc，其他的控制器的中断在驱动中需要时自行分配注册。

- bm中的内核映射表怎么知道从0x6000开始

  答：因为内核页表在物理地址0x4000(看amp linux内存报告的分析)，页表大小0x4000，bm用最后两g，因此从0x6000开始。

- boot_cpu_init()将online位图置1，但是在smp_init中条件检测!cpu_online(cpu)才会cpu_up

  答：在0核cpu_up(1)

- amp_init()中初始化的asp中的域在bm为什么可以使用，比如asp->sta.printk_fn = (u32)printk;放的是内核printk函数的虚拟地址，而bm直接使用该地址。

  答：因为内核的.text和.data都在内核虚拟地址空间的顶层2G，bm使用了内核的最后2G的页表。

- 内核高级追踪调试手段使用

  答：使用静态追踪(ftrace)或者动态追踪(编写kprobe驱动模块或者利用systemtap，systemtap本质也是使用kprobe)。