Linux kernel 3.18.7

start_kernel() in init/main.c

--> setup_arch() in arch/arm/kernel/setup.c

--> paging_init(mdesc) in arch/arm/mm/mmu.c

--> devicemaps_init(mdesc) in arch/arm/mm/mmu.c

--> early_trap_init(vectors)

```
	/*
	 * Allocate the vector page early.
	 */
	vectors = early_alloc(PAGE_SIZE * 2);        申请2 physical pages
	early_trap_init(vectors);

void __init early_trap_init(void *vectors_base)
{
#ifndef CONFIG_CPU_V7M
	unsigned long vectors = (unsigned long)vectors_base;
	extern char __stubs_start[], __stubs_end[];
	extern char __vectors_start[], __vectors_end[];
	unsigned i;

	vectors_page = vectors_base;

	/*
	 * Poison the vectors page with an undefined instruction.  This
	 * instruction is chosen to be undefined for both ARM and Thumb
	 * ISAs.  The Thumb version is an undefined instruction with a
	 * branch back to the undefined instruction.
	 */
	for (i = 0; i < PAGE_SIZE / sizeof(u32); i++)
		((u32 *)vectors_base)[i] = 0xe7fddef1;                //都填上invalid opcode

	/*
	 * Copy the vectors, stubs and kuser helpers (in entry-armv.S)
	 * into the vector page, mapped at 0xffff0000, and ensure these
	 * are visible to the instruction stream.
	 */
memcpy((void *)vectors, __vectors_start, __vectors_end - __vectors_start);
	 (1)
	memcpy((void *)vectors + 0x1000, __stubs_start, __stubs_end - __stubs_start);          (2)

	kuser_init(vectors_base);

	flush_icache_range(vectors, vectors + PAGE_SIZE * 2);
	modify_domain(DOMAIN_USER, DOMAIN_CLIENT);
#else /* ifndef CONFIG_CPU_V7M */
	/*
	 * on V7-M there is no need to copy the vector table to a dedicated
	 * memory area. The address is configurable and so a table in the kernel
	 * image can be used.
	 */
#endif
}
```

上面(1),(2)中

__vectors_start，__vectors_end

__stubs_start,__stubs_end

定义在vmlinux.lds的链接script中。

```
/*
 * The vectors and stubs are relocatable code, and the
 * only thing that matters is their relative offsets
 */
__vectors_start = .;
.vectors 0 : AT(__vectors_start) {
 *(.vectors)
}
. = __vectors_start + SIZEOF(.vectors);
__vectors_end = .;
__stubs_start = .;
.stubs 0x1000 : AT(__stubs_start) {
 *(.stubs)
}
. = __stubs_start + SIZEOF(.stubs);
__stubs_end = .;
```

上面的script中__vectors_start位于0地址，而__stubs_start位于0x1000地址。即

__vectors_start，__vectors_end可以占用地址空间的1st page，而

__stubs_start,__stubs_end可以占用地址空间的2nd page.

在early_trap_init（ ）中把位于地址空间1st page和2nd page的内容复制到vectors_base处（申请的2 physical pages）.

```
root@granite2:~# cat /proc/self/maps
00008000-00095000 r-xp 00000000 b3:22 358        /bin/busybox.nosuid
0009d000-0009e000 rw-p 0008d000 b3:22 358        /bin/busybox.nosuid
0009e000-000a0000 rw-p 00000000 00:00 0
000a5000-000a7000 rw-p 0008d000 b3:22 358        /bin/busybox.nosuid
468f0000-4690f000 r-xp 00000000 b3:22 5895       /lib/ld-2.18.so
46916000-46917000 r--p 0001e000 b3:22 5895       /lib/ld-2.18.so
46917000-46918000 rw-p 0001f000 b3:22 5895       /lib/ld-2.18.so
46920000-46a4b000 r-xp 00000000 b3:22 5558       /lib/libc-2.18.so
46a4b000-46a52000 ---p 0012b000 b3:22 5558       /lib/libc-2.18.so
46a52000-46a54000 r--p 0012a000 b3:22 5558       /lib/libc-2.18.so
46a54000-46a56000 rw-p 0012c000 b3:22 5558       /lib/libc-2.18.so
46a56000-46a58000 rw-p 00000000 00:00 0
b6f37000-b6f38000 rw-p 00000000 00:00 0
b6f3d000-b6f3e000 rw-p 00000000 00:00 0
bedbb000-beddc000 rw-p 00000000 00:00 0          [stack]
bef09000-bef0a000 r-xp 00000000 00:00 0          [sigpage]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

中断向量表位于0xffff0000,工作与high vectors.

由于vectors_base只是指向2 physical pages，还需要map the 2 physical pages to 0xffff0000.

下面的code就完成该工作。

in devicemaps_init()

```
        /*
         * Create a mapping for the machine vectors at the high-vectors
         * location (0xffff0000).  If we aren't using high-vectors, also
         * create a mapping at the low-vectors virtual address.
         */
        map.pfn = __phys_to_pfn(virt_to_phys(vectors));
        map.virtual = 0xffff0000;
        map.length = PAGE_SIZE;
#ifdef CONFIG_KUSER_HELPERS
        map.type = MT_HIGH_VECTORS;
#else
        map.type = MT_LOW_VECTORS;
#endif
        create_mapping(&map);

        if (!vectors_high()) {
                map.virtual = 0;
                map.length = PAGE_SIZE * 2;
                map.type = MT_LOW_VECTORS;
                create_mapping(&map);
        }

        /* Now create a kernel read-only mapping */
        map.pfn += 1;
        map.virtual = 0xffff0000 + PAGE_SIZE;
        map.length = PAGE_SIZE;
        map.type = MT_LOW_VECTORS;
        create_mapping(&map);
```

那么在0xffff0000开始的2 pages中到底是什么内容呢？

arch/arm/kernel/entry-armv.S

```
__vectors_start:
    W(b)    vector_rst
    W(b)    vector_und
```

```
	W(ldr)	pc, __vectors_start + 0x1000

	W(b)	vector_pabt

	W(b)	vector_dabt

	W(b)	vector_addrexcptn

	W(b)	vector_irq		( 0 )

	W(b)	vector_fiq


/*

 * Interrupt dispatcher

 */

	vector_stub	irq, IRQ_MODE, 4


	.long	__irq_usr		@  0  (USR_26 / USR_32)		( 1.1 ) 来自user mode的
interrupt

	.long	__irq_invalid		@  1  (FIQ_26 / FIQ_32)

	.long	__irq_invalid		@  2  (IRQ_26 / IRQ_32)

	.long	__irq_svc		@  3  (SVC_26 / SVC_32)		( 1.2 ) 来自svc mode的
interrupt

	.long	__irq_invalid		@  4

	.long	__irq_invalid		@  5

	.long	__irq_invalid		@  6

	.long	__irq_invalid		@  7

	.long	__irq_invalid		@  8

	.long	__irq_invalid		@  9

	.long	__irq_invalid		@  a

	.long	__irq_invalid		@  b
```

```
        .long __irq_invalid                @ c
        .long __irq_invalid                @ d
        .long __irq_invalid                @ e
        .long __irq_invalid                @ f


__irq_usr:
        usr_entry
        kuser_cmpxchg_check
        irq_handler                        (2.1)       macro
        get_thread_info tsk
        mov why, #0
        b      ret_to_user_from_irq
 UNWIND(.fnend           )
ENDPROC(__irq_usr)

__irq_svc:
        svc_entry
        irq_handler                    (2.2)

#ifdef CONFIG_PREEMPT
        get_thread_info tsk
        ldr    r8, [tsk, #TI_PREEMPT]         @ get preempt count
        ldr    r0, [tsk, #TI_FLAGS]           @ get flags
        teq    r8, #0                     @ if preempt count != 0
```

```
movne     r0, #0                          @ force flags to 0
        tst    r0, #_TIF_NEED_RESCHED
        blne svc_preempt
#endif

        svc_exit r5, irq = 1              @ return from exception
 UNWIND(.fnend            )
ENDPROC(__irq_svc)



/*
 * Interrupt handling.
 */
        .macro    irq_handler             (3)
#ifdef CONFIG_MULTI_IRQ_HANDLER
        ldr    r1, =handle_arch_irq       (4)
        mov r0, sp
        adr   lr, BSYM(9997f)
        ldr    pc, [r1]
#else
        arch_irq_handler_default
#endif
9997:
        .endm
```

Question: what is CONFIG_MULTI_IRQ_HANDLER?

CONFIG_MULTI_IRQ_HANDLER=y (in pegmatite's config)

```
#ifdef CONFIG_MULTI_IRQ_HANDLER
	.globl	handle_arch_irq
handle_arch_irq:
	.space	4
#endif
```

在vmlinux的image中handle_arch_irq只是一个没有填写有意义value的4字节空间。

但在kernel初始化完成后，硬件中断的流程还是很简单的，就是

（0）--> (1.1) --> (2.1) --> (3) --> (4) 来自user mode的interrupt

（0）--> (1.2) --> (2.2) --> (3) --> (4) 来自svc mode的interrupt

关于（4）是什么，接着看。

in drivers/irqchip/irq-gic.c

```
#ifdef CONFIG_OF
static int gic_cnt __initdata;

static int __initearly_trap_init
gic_of_init(struct device_node *node, struct device_node *parent)
{
        void __iomem *cpu_base;
        void __iomem *dist_base;
        u32 percpu_offset;
        int irq;

        if (WARN_ON(!node))
                return -ENODEV;

        dist_base = of_iomap(node, 0);
        WARN(!dist_base, "unable to map gic dist registers\n");

        cpu_base = of_iomap(node, 1);
        WARN(!cpu_base, "unable to map gic cpu registers\n");

        if (of_property_read_u32(node, "cpu-offset", &percpu_offset))
                percpu_offset = 0;

        gic_init_bases(gic_cnt, -1, dist_base, cpu_base, percpu_offset, node);
        if (!gic_cnt)
                gic_init_physaddr(node);

        if (parent) {
                irq = irq_of_parse_and_map(node, 0);
                gic_cascade_irq(gic_cnt, irq);
        }
        gic_cnt++;
        return 0;
}
IRQCHIP_DECLARE(gic_400, "arm,gic-400", gic_of_init);
IRQCHIP_DECLARE(cortex_a15_gic, "arm,cortex-a15-gic", gic_of_init);
IRQCHIP_DECLARE(cortex_a9_gic, "arm,cortex-a9-gic", gic_of_init);
IRQCHIP_DECLARE(cortex_a7_gic, "arm,cortex-a7-gic", gic_of_init);
IRQCHIP_DECLARE(msm_8660_qgic, "qcom,msm-8660-qgic", gic_of_init);
IRQCHIP_DECLARE(msm_qgic2, "qcom,msm-qgic2", gic_of_init);

#endif
```

in drivers/irqchip/irqchip.h,

```
/*
 * This macro must be used by the different irqchip drivers to declare
 * the association between their DT compatible string and their
 * initialization function.
 *
 * @name: name that must be unique accross all IRQCHIP_DECLARE of the
 * same file.
 * @compstr: compatible string of the irqchip driver
 * @fn: initialization function
 */
#define IRQCHIP_DECLARE(name, compat, fn) OF_DECLARE_2(irqchip, name, compat, fn)

#define _OF_DECLARE(table, name, compat, fn, fn_type)                   \
        static const struct of_device_id __of_table_##name              \
                __used __section(__##table##_of_table)                  \
                 = { .compatible = compat,                              \
                     .data = (fn == (fn_type)NULL) ? fn : fn  }

#define OF_DECLARE_2(table, name, compat, fn) \
                _OF_DECLARE(table, name, compat, fn, of_init_fn_2)
```

section: __irqchip_of_table

static const struct of_device_id __of_table_gic_400 = {

.compatible = "arm,gic-400",

.data = gic_of_init };

static const struct of_device_id __of_table_cortex_a15_gic = {

.compatible = "arm,cortex-a15-gic",

.data = gic_of_init };

static const struct of_device_id __of_table_cortex_a9_gic = {

.compatible = "arm,cortex-a9-gic",

.data = gic_of_init };

static const struct of_device_id __of_table_cortex_a7_gic = {

.compatible = "arm,cortex-a7-gic",

.data = gic_of_init };

而在vmlinux.lds的链接脚本中

```
.init.data : {

 *(.init.data) *(.meminit.data) *(.init.rodata) *(.meminit.rodata) . = ALIGN(8); __clk_of_table = .; *(__clk_of_table) *(__clk_of_table_end) . = ALIGN(8); __reservedmem_of_table = .; *(__reservedmem_of_table) *(__reservedmem_of_table_end) . = ALIGN(8); __clksrc_of_table = .; *(__clksrc_of_table) *(__clksrc_of_table_end) . = ALIGN(8); __cpu_method_of_table = .; *(__cpu_method_of_table) *(__cpu_method_of_table_end) . = ALIGN(32); __dtb_start = .; *(.dtb.init.rodata) __dtb_end = .; . = ALIGN(8); __irqchip_of_table = .; *(__irqchip_of_table) *(__irqchip_of_table_end)

 . = ALIGN(16); __setup_start = .; *(.init.setup) __setup_end = .;

 __initcall_start = .; *(.initcallearly.init) __initcall0_start = .; *(.initcall0.init) *(.initcall0s.init) __initcall1_start = .; *(.initcall1.init) *(.initcall1s.init) __initcall2_start = .; *(.initcall2.init) *(.initcall2s.init) __initcall3_start = .; *(.initcall3.init) *(.initcall3s.init) __initcall4_start = .; *(.initcall4.init) *(.initcall4s.init) __initcall5_start = .; *(.initcall5.init) *(.initcall5s.init) __initcallrootfs_start = .; *(.initcallrootfs.init) *(.initcallrootfss.init) __initcall6_start = .; *(.initcall6.init) *(.initcall6s.init) __initcall7_start = .; *(.initcall7.init) *(.initcall7s.init) __initcall_end = .;

 __con_initcall_start = .; *(.con_initcall.init) __con_initcall_end = .;

 __security_initcall_start = .; *(.security_initcall.init) __security_initcall_end = .;

 . = ALIGN(4); __initramfs_start = .; *(.init.ramfs) . = ALIGN(8); *(.init.ramfs.info)

}
```

即整个被包在__irqchip_of_table section中的struct of_device_id的array的首指针为variable __irqchip_of_table,尾部为__irqchip_of_table_end.

在kernel初始化时有如下初始化动作

```
asmlinkage __visible void __init start_kernel(void)

-->

  in  init_IRQ() function


      if (IS_ENABLED(CONFIG_OF) && !machine_desc->init_irq)

            irqchip_init();

      else

            machine_desc->init_irq();
```

in arch/arm/mach-pegmatite/pegmatite.c

```
DT_MACHINE_START(PEGMATITE_DT, "Marvell Pegmatite (Device Tree)")

#ifdef CONFIG_SMP

      .smp              = smp_ops(pegmatite_smp_ops),

#endif

      .init_machine  = pegmatite_dt_init,

      .map_io           = pegmatite_map_io,

      .init_early = pegmatite_init_early,

      .init_irq   = pegmatite_init_irq,

      .init_time  = pegmatite_timer_and_clk_init,

      .restart    = pegmatite_restart,

      .dt_compat        = pegmatite_dt_compat,
```

```
#ifdef CONFIG_ZONE_DMA
	.dma_zone_size	= SZ_256M,
#endif
MACHINE_END

static void __init pegmatite_init_irq(void)
{
	irqchip_init();
}
```

从上面的code看，实际上在machine_desc的定义中无需

.init_irq = pegmatite_init_irq

因为在未对.init_irq field赋值的情况下系统本身就会调用irqchip_init()。

in drivers/irqchip/irqchip.c

```
static const struct of_device_id
irqchip_of_match_end __used __section(__irqchip_of_table_end);

extern struct of_device_id __irqchip_of_table[];

void __init irqchip_init(void)
{
of_irq_init(__irqchip_of_table);
}
```

of_irq_init()会根据定义在dts中的interrupt controller的依赖关系进行初始化。一个原则是parent of interrupt controller先初始化，son of interrupt controller后初始化。

总结一下，在start_kernel（ ）--> init_IRQ() 中会调用gic interrupt controller的初始化函数

gic_of_init（ ）--> gic_init_bases()

```
void __init gic_init_bases(unsigned int gic_nr, int irq_start,
                    void __iomem *dist_base, void __iomem *cpu_base,
                    u32 percpu_offset, struct device_node *node)
{
        irq_hw_number_t hwirq_base;
        struct gic_chip_data *gic;
        int gic_irqs, irq_base, i;
        int nr_routable_irqs;

        BUG_ON(gic_nr >= MAX_GIC_NR);

        gic = &gic_data[gic_nr];


        ......

        if (gic_nr == 0) {
#ifdef CONFIG_SMP
                set_smp_cross_call(gic_raise_softirq);
                register_cpu_notifier(&gic_cpu_notifier);
#endif
                set_handle_irq(gic_handle_irq);
        }
```

```
        gic_chip.flags |= gic_arch_extn.flags;
        gic_dist_init(gic);
        gic_cpu_init(gic);
        gic_pm_init(gic);
}

#ifdef CONFIG_MULTI_IRQ_HANDLER
void __init set_handle_irq(void (*handle_irq)(struct pt_regs *))
{
        if (handle_arch_irq)
                return;

        handle_arch_irq = handle_irq;
}
#endif
```

gic interrupt controller在初始化时会用gic_handle_irq() function address 填写4字节的handle_arch_irq空间。

这样

（0）--> (1.1) --> (2.1) --> (3) --> gic_handle_irq() 来自user mode的interrupt

（0）--> (1.2) --> (2.2) --> (3) --> gic_handle_irq() 来自svc mode的interrupt

```
static void __exception_irq_entry gic_handle_irq(struct pt_regs *regs)
{
        u32 irqstat, irqnr;
        struct gic_chip_data *gic = &gic_data[0];
        void __iomem *cpu_base = gic_data_cpu_base(gic);

        do {
                irqstat = readl_relaxed(cpu_base + GIC_CPU_INTACK);
                irqnr = irqstat & GICC_IAR_INT_ID_MASK;

                if (likely(irqnr > 15 && irqnr < 1021)) {
                        handle_domain_irq(gic->domain, irqnr, regs);
                        continue;
                }
                if (irqnr < 16) {
                        writel_relaxed(irqstat, cpu_base + GIC_CPU_EOI);
#ifdef CONFIG_SMP
                        handle_IPI(irqnr, regs);
#endif
                        continue;
                }
                break;
        } while (1);
}
```

至于具体某个hardware的interrupt handler，则接着由gic_handle_irq()出发，继续前进。