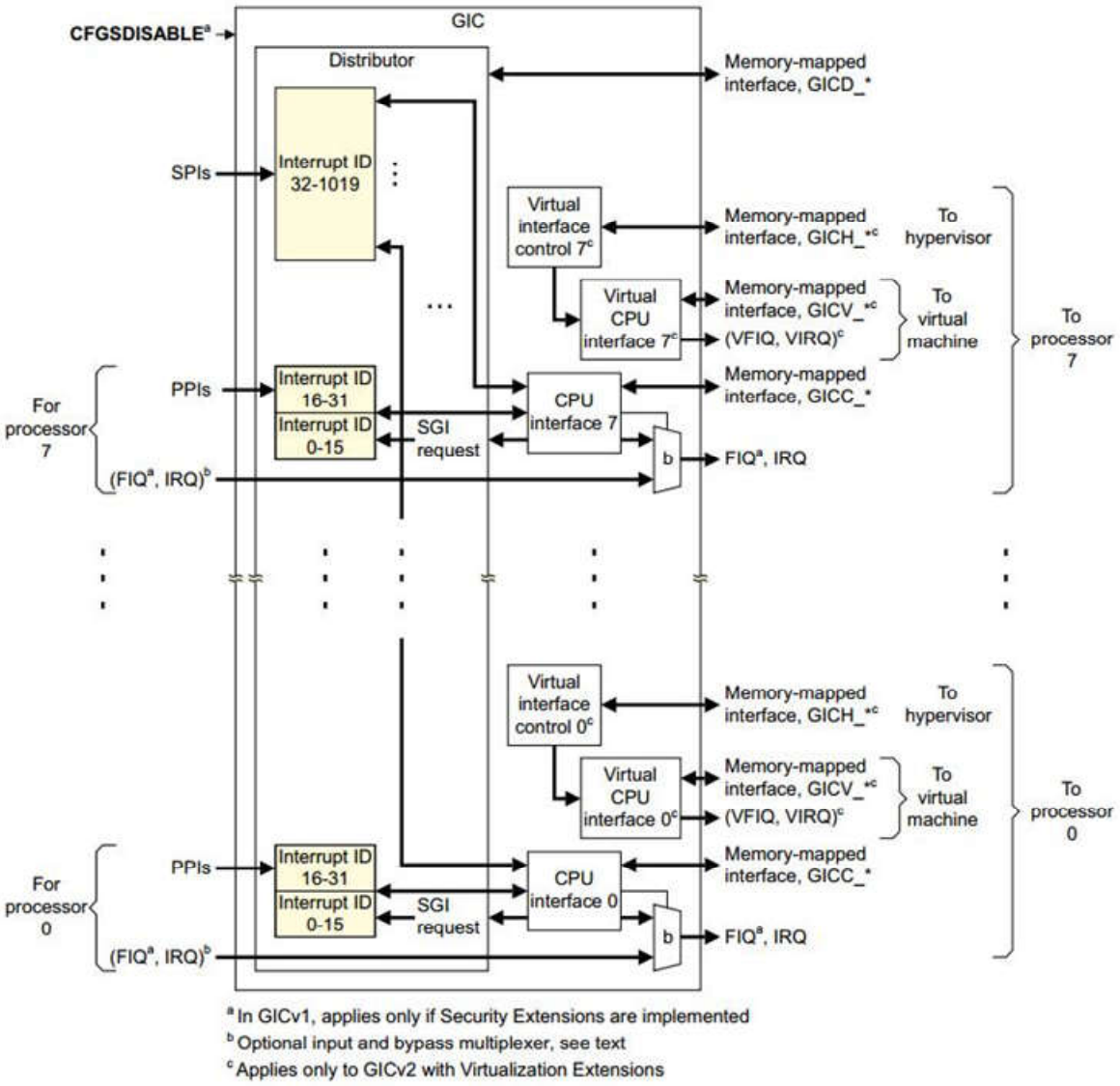


1.ARM SMP一般都与GIC连接，提供per processor interrupts(PPI)，shared processor interrupts(SPI)和software generated interrupts(SGI)

2.root GIC直接与cpu相连，一般拥有PPI和SGI；secondary GIC级联在root GIC上，不含有PPI和SGI；SPI则是全局中断，可以分发至任一可用cpu上进行处理

pkg

imx6q-kernel3.10.53

irq_desc->irq_data.chip_data 与 irq_domain->host_data 以及 irq_desc->irq_data.handler_data 都是保存gic_data[gic_nr]的地址 属于 struct gic_chip_data类型，初始化gic时候就对其成员进行了赋值，可以看出里面保存GIC的信息distributor和cpu的寄存器基地址 &&domain定义&&支持的中断数

struct gic_chip_data
+ dist_base : union gic_base
+ cpu_base : union gic_base
+ domain : struct irq_domain *
+ gic_irqs : unsigned int

struct irq_chip
+ name : const char *
+ flags : unsigned long
+ *irq_startup() : unsigned int
+ *irq_{shutdown/enable/disable}() : void
+ *irq_{ack/mask/mask_ack/unmask/eoi}() : void
+ *irq_{set_affinity/retrigger/set_type/set_wake}() : int
+ *cpu_online/cpu_offline() : void
+ *irq_bus_lock/irq_bus_sync_unlock() : void

struct irq_data
+ irq : unsigned int
+ hwirq : unsigned long
+ chip : struct irq_chip *
+ domain : struct irq_domain
+ handler_data : void *
+ chip_data : void *

struct legacy
+ size : unsigned int
+ first_irq : unsigned int
+ first_hwirq : irq_hw_number_t

union revmap_data
+ legacy : struct legacy
+ linear : struct linear
+ nomap : struct nomap
+ tree : struct radix_tree_root

struct nomap
+ max_irq : unsigned int

struct nomap
+ max_irq : unsigned int

struct irq_domain
+ link : struct list_head
+ revmap_type : unsigned int
+ revmap_data : union revmap_data
+ ops : const struct irq_domain_ops *
+ host_data : void *
+ inval_irq : irq_hw_number_t
+ of_node : struct device_node *

typedef unsigned long irq_hw_number_t;

struct irq_domain_ops
+ *match() : int
+ *map() : int
+ *unmap() : void
+ *xlate() : void

hwirq

通过映射关系 irq_find_mapping

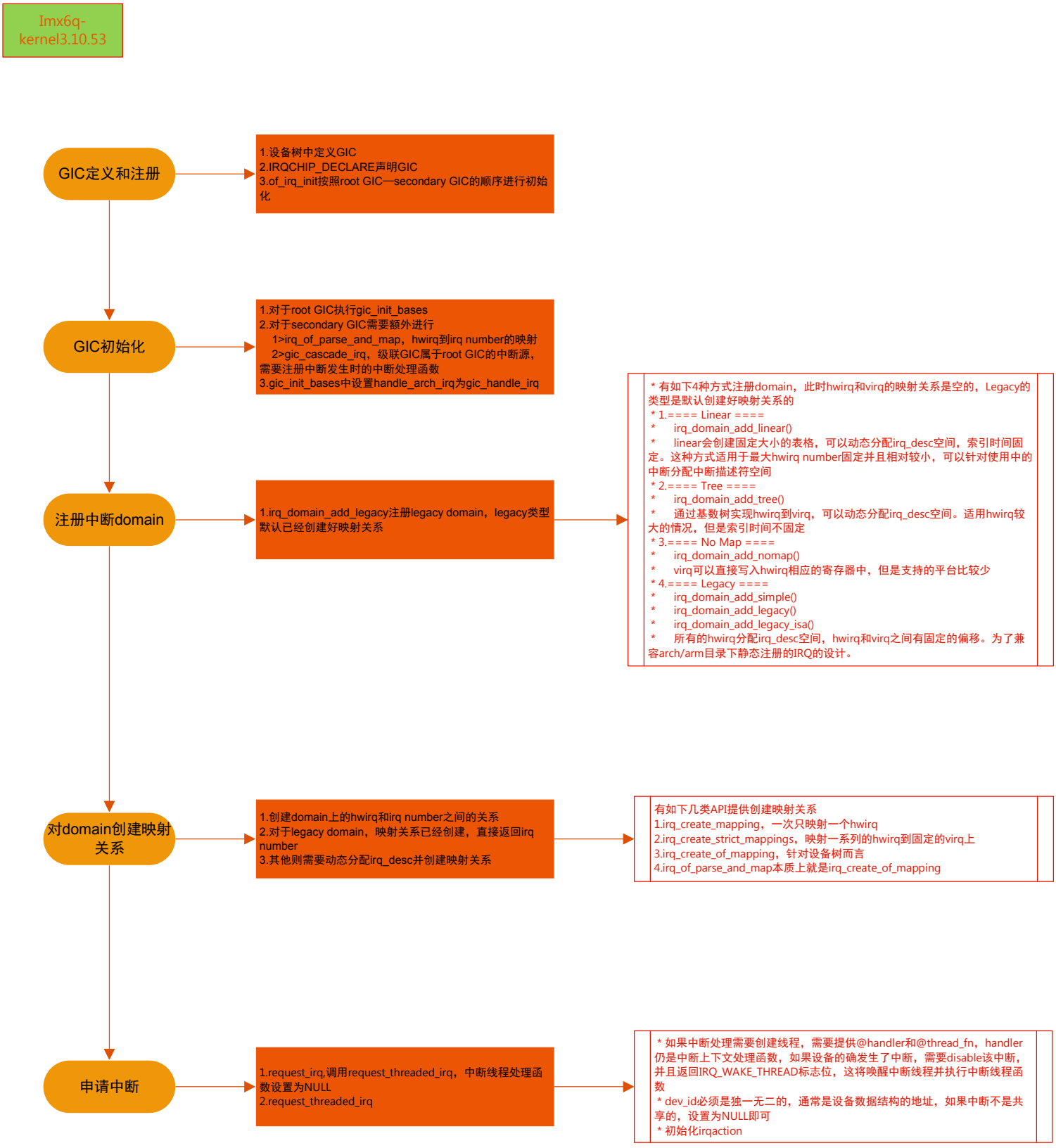
virq

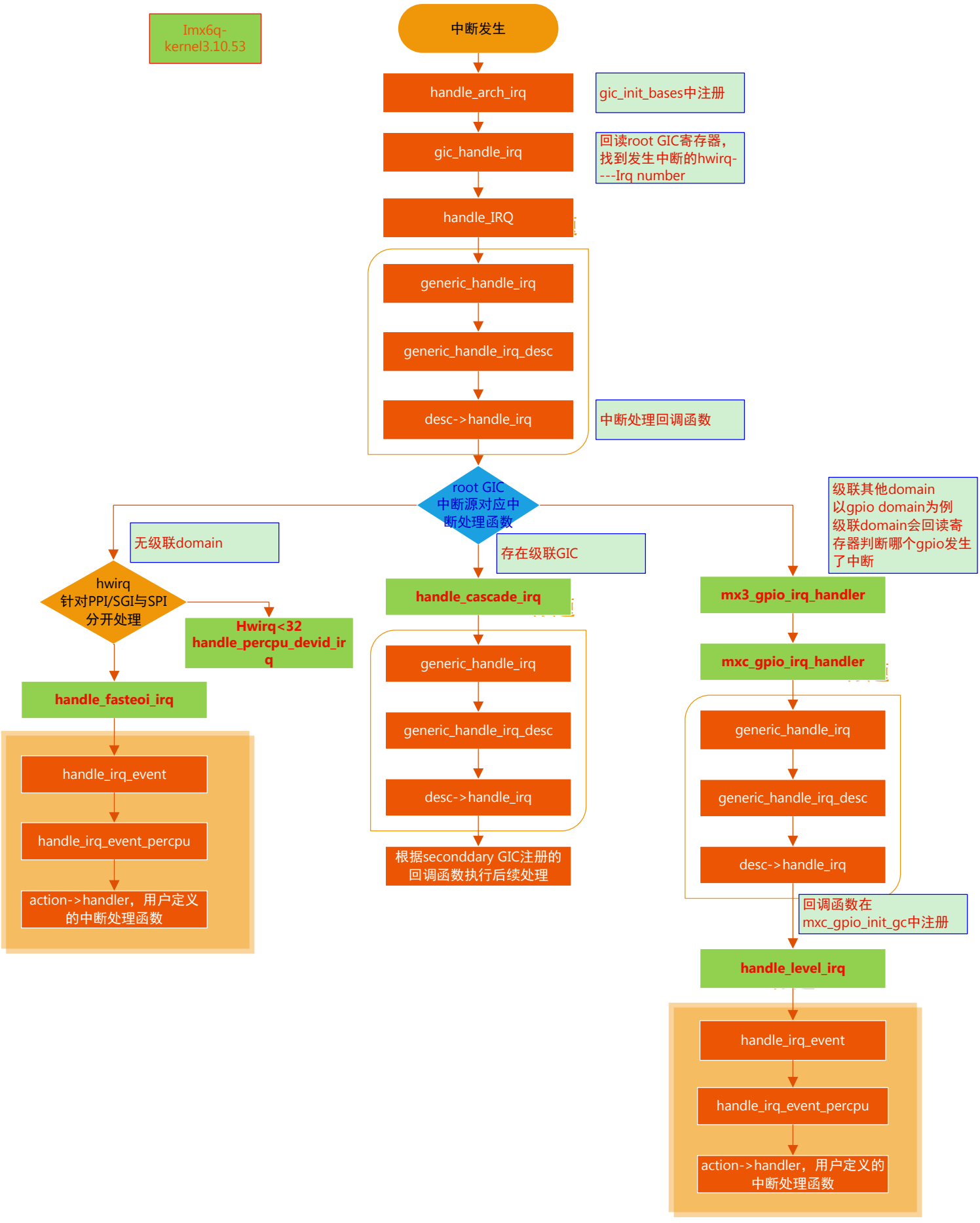
#define irq_to_desc(irq) (&irq_desc[irq])

struct irq_desc
+ irq_data : struct irq_data
+ handle_irq : irq_flow_handler_t
+ action : struct irqaction *
+ status_use_accessors : unsigned int

typedef void (*irq_flow_handler_t)(unsigned int irq, struct irq_desc *desc);

struct irqaction
+ handler : irq_handler_t
+ devid : void *
+ next : struct irqaction *
+ thread_fn : struct irqaction
+ thread : struct task_struct *
+ irq : unsigned int
+ flags : unsigned int
+ thread_flags : unsigned long
+ thread_mask : unsigned long
+ name : const char *
+ dir : struct proc_dir_entry *


Imx6q-kernel3.10.53
GIC定义和注册
1.设备树中定义GIC
2.IRQCHIP_DECLARE声明GIC
3.of_irq_init按照root GIC—secondary GIC的顺序进行初始化
GIC初始化
1.对于root GIC执行gic_init_bases
2.对于secondary GIC需要额外进行
1>irq_of_parse_and_map，hwirq到irq number的映射
2>gic_cascade_irq，级联GIC属于root GIC的中断源，需要注册中断发生时的中断处理函数
3.gic_init_bases中设置handle_arch_irq为gic_handle_irq
注册中断domain
1.irq_domain_add_legacy注册legacy domain，legacy类型默认已经创建好映射关系
* 有如下4种方式注册domain，此时hwirq和virq的映射关系是空的，Legacy的类型是默认创建好映射关系的
* 1.==== Linear ====
* irq_domain_add_linear()
* linear会创建固定大小的表格，可以动态分配irq_desc空间，索引时间固定。这种方式适用于最大hwirq number固定并且相对较小，可以针对使用中的中断分配中断描述符空间
* 2.==== Tree ====
* irq_domain_add_tree()
* 通过基数树实现hwirq到virq，可以动态分配irq_desc空间。适用hwirq较大的情况，但是索引时间不固定
* 3.==== No Map ====
* irq_domain_add_nomap()
* virq可以直接写入hwirq相应的寄存器中，但是支持的平台比较少
* 4.==== Legacy ====
* irq_domain_add_simple()
* irq_domain_add_legacy()
* irq_domain_add_legacy_isa()
* 所有的hwirq分配irq_desc空间，hwirq和virq之间有固定的偏移。为了兼容arch/arm目录下静态注册的IRQ的设计。
对domain创建映射关系
1.创建domain上的hwirq和irq number之间的关系
2.对于legacy domain，映射关系已经创建，直接返回irq number
3.其他则需要动态分配irq_desc并创建映射关系
有如下几类API提供创建映射关系
1.irq_create_mapping，一次只映射一个hwirq
2.irq_create_strict_mappings，映射一系列的hwirq到固定的virq上
3.irq_create_of_mapping，针对设备树而言
4.irq_of_parse_and_map本质上就是irq_create_of_mapping
申请中断
1.request_irq,调用request_threaded_irq，中断线程处理函数设置为NULL
2.request_threaded_irq
* 如果中断处理需要创建线程，需要提供@handler和@thread_fn，handler仍是中断上下文处理函数，如果设备的确发生了中断，需要disable该中断，并且返回IRQ_WAKE_THREAD标志位，这将唤醒中断线程并执行中断线程函数
* dev_id必须是独一无二的，通常是设备数据结构的地址，如果中断不是共享的，设置为NULL即可
* 初始化irqaction


Imx6q-kernel3.10.53
中断发生
handle_arch_irq
gic_init_bases中注册
gic_handle_irq
回读root GIC寄存器，找到发生中断的hwirq----Irq number
handle_IRQ
generic_handle_irq
generic_handle_irq_desc
desc->handle_irq
中断处理回调函数
root GIC中断源对应中断处理函数
无级联domain
存在级联GIC
级联其他domain以gpio domain为例级联domain会回读寄存器判断哪个gpio发生了中断
hwirq针对PPI/SGI与SPI分开处理
Hwirq<32 handle_percpu_devid_irq
handle_fasteoi_irq
handle_irq_event
handle_irq_event_percpu
action->handler，用户定义的中断处理函数
handle_cascade_irq
generic_handle_irq
generic_handle_irq_desc
desc->handle_irq
根据seconddary GIC注册的回调函数执行后续处理
mx3_gpio_irq_handler
mxc_gpio_irq_handler
generic_handle_irq
generic_handle_irq_desc
desc->handle_irq
回调函数在mxc_gpio_init_gc中注册
handle_level_irq
handle_irq_event
handle_irq_event_percpu
action->handler，用户定义的中断处理函数