vector_fiq

一般情况下，Linux kernel并没有利用ARM的FIQ feature (CONFIG_FIQ is disable)

```
/*=============================================================================
 * FIQ "NMI" handler
 *-----------------------------------------------------------------------------
 * Handle a FIQ using the SVC stack allowing FIQ act like NMI on x86
 * systems.
 */
        vector_stub     fiq, FIQ_MODE, 4

        .long   __fiq_usr                       @  0  (USR_26 / USR_32)
        .long   __fiq_svc                       @  1  (FIQ_26 / FIQ_32)
        .long   __fiq_svc                       @  2  (IRQ_26 / IRQ_32)
        .long   __fiq_svc                       @  3  (SVC_26 / SVC_32)
        .long   __fiq_svc                       @  4
        .long   __fiq_svc                       @  5
        .long   __fiq_svc                       @  6
        .long   __fiq_abt                       @  7
        .long   __fiq_svc                       @  8
        .long   __fiq_svc                       @  9
        .long   __fiq_svc                       @  a
        .long   __fiq_svc                       @  b
        .long   __fiq_svc                       @  c
        .long   __fiq_svc                       @  d
        .long   __fiq_svc                       @  e
        .long   __fiq_svc                       @  f

        .globl  vector_fiq_offset
        .equ    vector_fiq_offset, vector_fiq

        .section .vectors, "ax", %progbits
```

虽说没有利用FIQ feature，但万一什么东西触发了FIQ呢，所以kernel也有fiq的handler。

常规的__fiq_svc & __fiq_usr都比较简单，实质性的handler是handle_fiq_as_nmi()。

in arch/arm/kernel/traps.c

```
/*
 * Handle FIQ similarly to NMI on x86 systems.
 *
 * The runtime environment for NMIs is extremely restrictive
 * (NMIs can pre-empt critical sections meaning almost all locking is
 * forbidden) meaning this default FIQ handling must only be used in
 * circumstances where non-maskability improves robustness, such as
 * watchdog or debug logic.
 *
 * This handler is not appropriate for general purpose use in drivers
 * platform code and can be overrideen using set_fiq_handler.
 */
asmlinkage void __exception_irq_entry handle_fiq_as_nmi(struct pt_regs *regs)
{
        struct pt_regs *old_regs = set_irq_regs(regs);

        nmi_enter();

        /* nop. FIQ handlers for special arch/arm features can be added here. */

        nmi_exit();

        set_irq_regs(old_regs);
}
```

几乎没任何实质性的处理。

__fiq_abt()则比较复杂，因为这时当在abort mode下产生FIQ时的处理。本来abort mode就是CPU处于出错状态了，在此状态下再来FIQ，更是危急的event发生了。

```
/*
 * Abort mode handlers
 */

@
@ Taking a FIQ in abort mode is similar to taking a FIQ in SVC mode
@ and reuses the same macros. However in abort mode we must also
@ save/restore lr_abt and spsr_abt to make nested aborts safe.
@
        .align 5
__fiq_abt:
        svc_entry trace=0

 ARM(   msr     cpsr_c, #ABT_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( mov     r0, #ABT_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( msr     cpsr_c, r0 )
        mov     r1, lr          @ Save lr_abt
        mrs     r2, spsr        @ Save spsr_abt, abort is now safe
 ARM(   msr     cpsr_c, #SVC_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( mov     r0, #SVC_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( msr     cpsr_c, r0 )
        stmfd   sp!, {r1 - r2}

        add     r0, sp, #8                      @ struct pt_regs *regs
        bl      handle_fiq_as_nmi

        ldmfd   sp!, {r1 - r2}
 ARM(   msr     cpsr_c, #ABT_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( mov     r0, #ABT_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( msr     cpsr_c, r0 )
        mov     lr, r1          @ Restore lr_abt, abort is unsafe
        msr     spsr_cxsf, r2   @ Restore spsr_abt
 ARM(   msr     cpsr_c, #SVC_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( mov     r0, #SVC_MODE | PSR_I_BIT | PSR_F_BIT )
 THUMB( msr     cpsr_c, r0 )

        svc_exit_via_fiq
 UNWIND(.fnend          )
ENDPROC(__fiq_abt)
```

反汇编后的code如下

c0012200 <__fiq_abt>:

c0012200: e24dd044 sub sp, sp, #68 ; 0x44

c0012204: e31d0004 tst sp, #4

```
c0012208:	024dd004	subeq	sp, sp, #4
c001220c:	e88d1ffe	stm	sp, {r1, r2, r3, r4, r5, r6, r7, r8, r9, sl, fp, ip}
c0012210:	e8900038	ldm	r0, {r3, r4, r5}
c0012214:	e28d7030	add	r7, sp, #48	; 0x30
c0012218:	e3e06000	mvn	r6, #0
c001221c:	e28d2044	add	r2, sp, #68	; 0x44
c0012220:	02822004	addeq	r2, r2, #4
c0012224:	e52d3004	push	{r3}		; (str r3, [sp, #-4]!)
c0012228:	e1a0300e	mov	r3, lr
c001222c:	e887007c	stm	r7, {r2, r3, r4, r5, r6}
c0012230:	e321f0d7	msr	CPSR_c, #215	; 0xd7
c0012234:	e1a0100e	mov	r1, lr
c0012238:	e14f2000	mrs	r2, SPSR
c001223c:	e321f0d3	msr	CPSR_c, #211	; 0xd3
c0012240:	e92d0006	push	{r1, r2}
c0012244:	e28d0008	add	r0, sp, #8
c0012248:	ebffd8ad	bl	c0008504 <handle_fiq_as_nmi>
c001224c:	e8bd0006	pop	{r1, r2}
c0012250:	e321f0d7	msr	CPSR_c, #215	; 0xd7
c0012254:	e1a0e001	mov	lr, r1
c0012258:	e16ff002	msr	SPSR_fsxc, r2
c001225c:	e321f0d3	msr	CPSR_c, #211	; 0xd3
c0012260:	e1a0000d	mov	r0, sp
c0012264:	e9907ffe	ldmib	r0, {r1, r2, r3, r4, r5, r6, r7, r8, r9, sl, fp, ip, sp, lr}
c0012268:	e321f0d1	msr	CPSR_c, #209	; 0xd1
```

```
c001226c:	e280803c	add	r8, r0, #60	; 0x3c
c0012270:	e5909040	ldr	r9, [r0, #64]	; 0x40
c0012274:	e16ff009	msr	SPSR_fsxc, r9
c0012278:	e5900000	ldr	r0, [r0]
c001227c:	e8d88000	ldm	r8, {pc}^
```

???