11. 基于ARM Cortex-A9中断详解

原创 土豆居士 一口Linux 2020-12-28 11:27

收录于合集
#所有原创 206 #从0学arm 27

一、中断概念

操作系统中，中断是很重要的组成部分。出现某些意外情况需主机干预时，机器能自动停止正在运行的程序并转入处理新情况的程序，处理完毕后又返回原被暂停的程序继续运行。

有了中断系统才可以不用一直轮询（polling）是否有事件发生，系统效率才得以提高。

一般在系统中，中断控制分为三个部分：**「模块、中断控制器和处理器」**。

其中模块通常由寄存器控制是否使能中断和中断触发条件等；中断控制器可以管理中断的优先级等，而处理器则由寄存器设置用来响应中断。

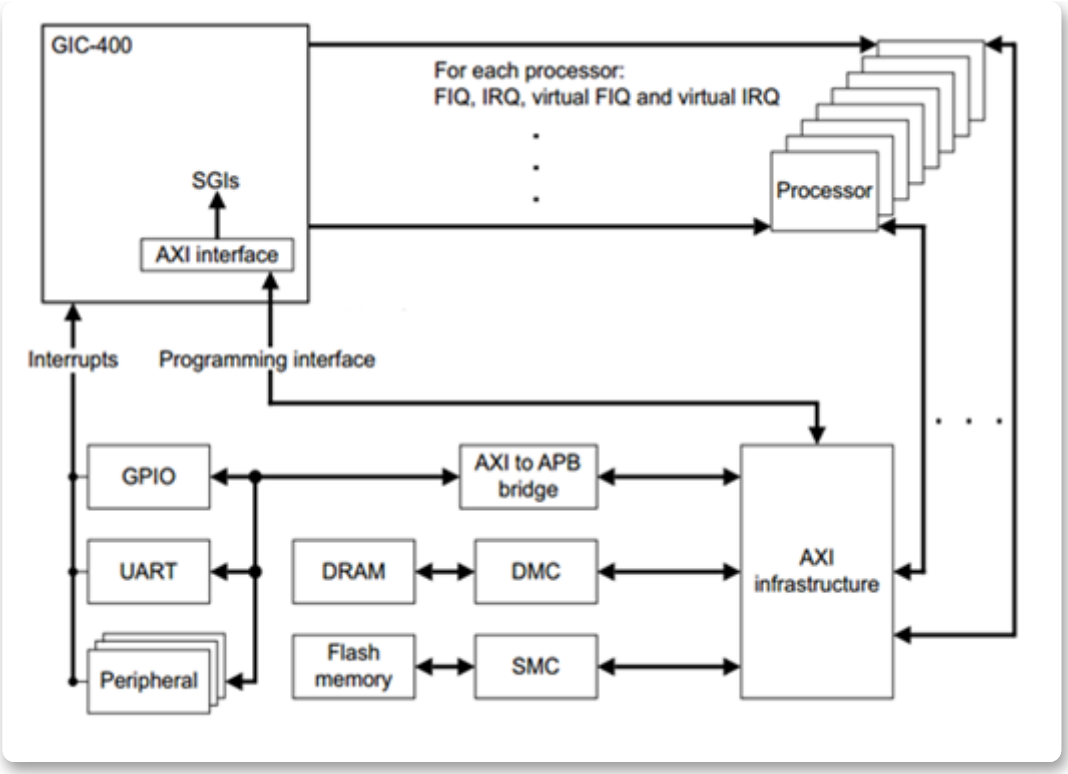
二、GIC

作为 ARM 系统中通用中断控制器的是 GIC（Generic Interrupt Controller），目前有四个版本，V1～V4(V2最多支持8个ARM core，V3/V4支持更多的ARM core，主要用于ARM64系统结构）。

【注意】对于一些老的ARM处理器，比如ARM11，Cortex-A8，中断控制器一般是VIC(向量中断控制器)。

1. GIC-400

下面以GIC-400为例，它更适合嵌入式系统，符合v2版本的GIC architecture specification。GIC-400通过AMBA（Advanced Microcontroller Bus Architecture）片上总线连接到一个或者多个ARM处理器上。

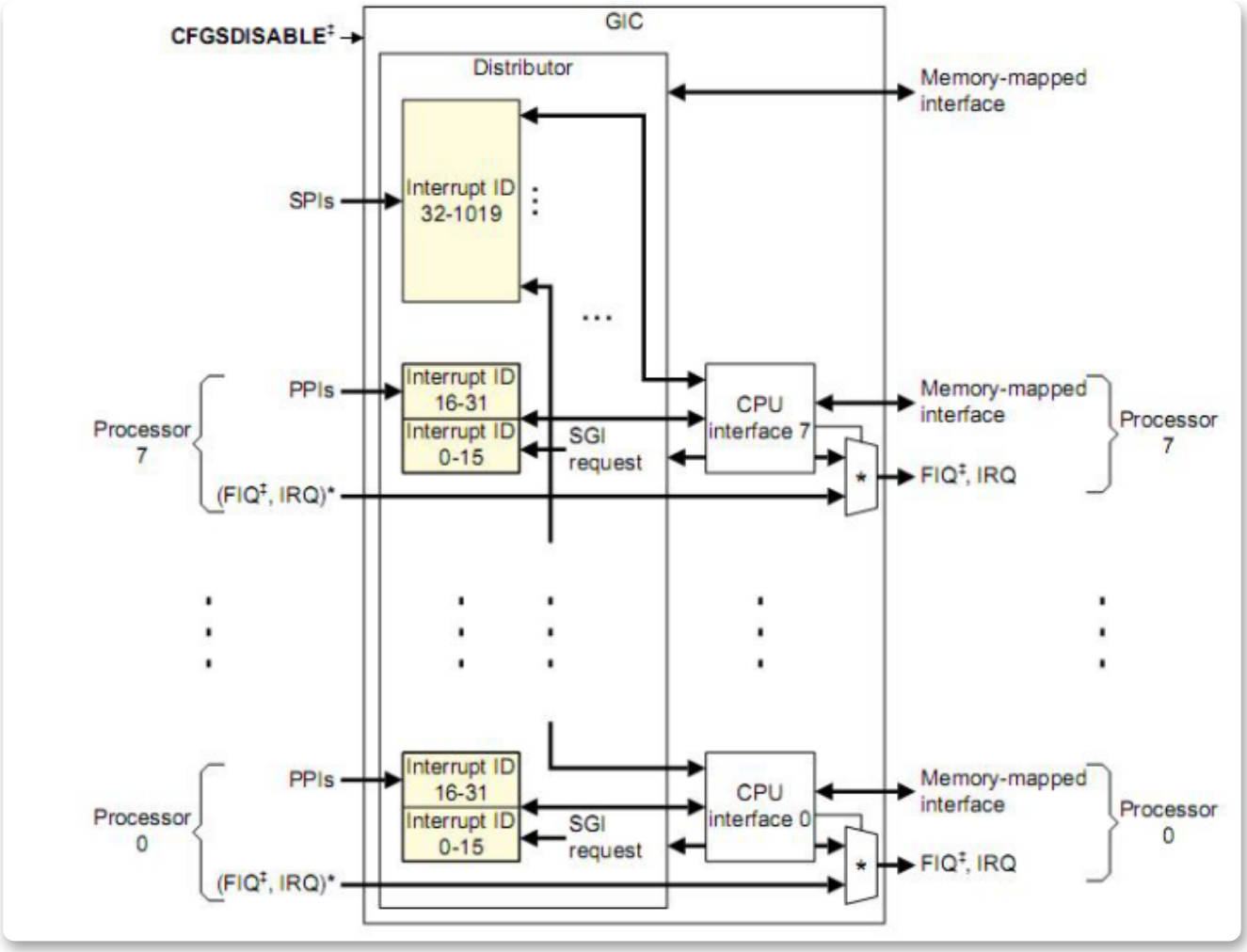


GIC中断控制器全局图

从上图可以看出， GIC 是联系外设中断和 CPU 的桥梁，也是各 CPU 之间中断互联的通道（也带有管理功能），它负责检测、管理、分发中断，可以做到：

1. 使能或禁止中断；
2. 把中断分组到Group0还是Group1（Group0作为安全模式使用连接FIQ ，Group1 作为非安全模式使用，连接IRQ ）；
3. 多核系统中将中断分配到不同处理器上；
4. 设置电平触发还是边沿触发方式（不等于外设的触发方式）；
5. 虚拟化扩展。

ARM CPU 对外的连接只有2 个中断：「IRQ和FIQ」 ，相对应的处理模式分别是一般中断（IRQ ）处理模式和快速中断（FIQ ）处理模式。所以GIC 最后要把中断汇集成2 条线，与CPU 对接。



GIC中断控制器结构

分发器：负责各个子中断使能，设置触发方式，优先级排序，分发到哪个 CPU 上；接口：负责总的中断的使能，状态的维护。

2. 分发器功能

分发器的主要的作用是检测各个中断源的状态，控制各个中断源的行为，分发各个中断源产生的中断事件到指定的一个或者多个CPU接口上。虽然分发器可以管理多个中断源，但是它总是把优先级最高的那个中断请求送往CPU接口。分发器对中断的控制包括：

- （a）中断使能或禁能控制。分发器对中断的控制分成两个级别，一个是全局中断的控制（GIC_DIST_CTRL），一旦禁止了全局的中断，那么任何的中断源产生的中断事件都不会被传递到CPU接口；另外一个级别是对针对各个中断源进行控制（GIC_DIST_ENABLE_CLEAR），禁止某一个中断源会导致该中断事件不会分发到CPU接口，但不影响其他中断源产生中断事件的分发。
- （b）控制将当前优先级最高的中断事件分发到一个或者一组CPU接口。
- （c）优先级控制。
- （d）中断属性设定，例如是电平触发还是边沿触发。
- （e）中断的设定。

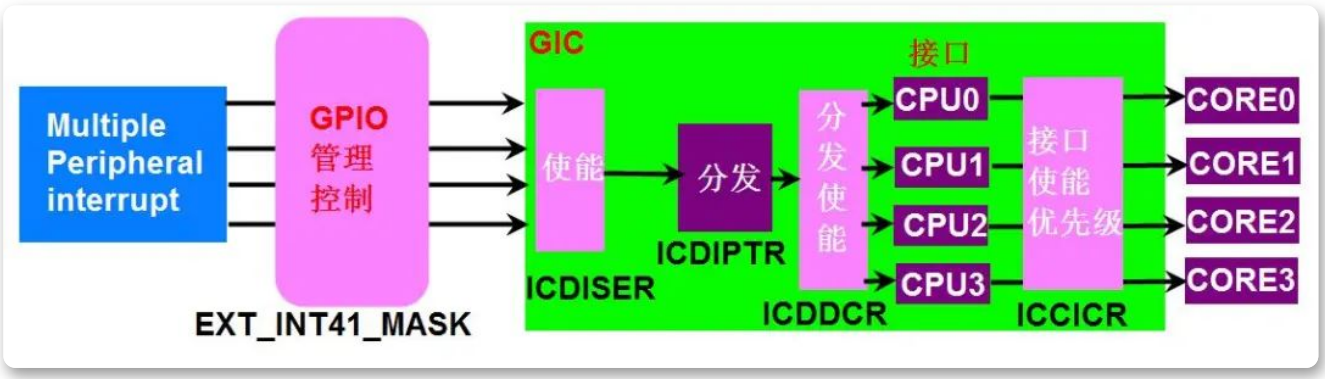
分发器可以管理若干个中断源，这些中断源用ID来标识，我们称之interrupt ID。

3. CPU接口功能

CPU接口主要用于和CPU进行接口。

主要功能包括：

- （a）使能或者禁止CPU接口向连接的CPU提交中断事件。对于ARM，CPU接口和CPU之间的中断信号线是nIRQCPU和nFIQCPU。如果禁止了中断，那么即便是分发器分发了一个中断事件到CPU接口，但是也不会提交指定的nIRQ或者nFIQ通知CPU。
- （b）ackowledging中断。CPU会向CPU接口应答中断，中断一旦被应答，分发器就会把该中断的状态从pending状态修改成active，如果没有后续pending的中断，那么CPU 接口就会deassert nIRQCPU和nFIQCPU信号线。如果在这个过程中又产生了新的中断，那么分发器就会把该中断的状态从pending状态修改成pending and active。此时，CPU接口仍然会保持nIRQ或者nFIQ信号的asserted状态，也就是向CPU通知下一个中断。
- （c）中断处理完毕的通知。当中断处理器处理完了一个中断的时候，会向写CPU 接口的寄存器从而通知GIC已经处理完该中断。做这个动作一方面是通知分发器将中断状态修改为deactive，另外一方面，可以允许其他的pending的中断向CPU接口提交。
- （d）设定优先级掩码。通过优先级掩码可以mask掉一些优先级比较低的中断，这些中断不会通知到CPU。
- （e）设定中断抢占的策略。
- （f）在多个中断事件同时到来的时候，选择一个优先级最高的通知CPU。



key中断管理模块图

以上图为例，该图是按键产生的中断信号要到达cpu所要经过的路径。

1. 外设中断源有很多，通常芯片厂商会设计若干个第一级中断控制器，进行第一次处理，key连接的是GPX1中断控制器，寄存器EXT_INT41_MASK用于使能该中断；
2. GIC主要包括分排气和cpu interface；

3. ICDISER用于使能分派器，ICDIPTR用于将中断信号分发给对应的cpu interface；
4. ICCICR用于使能CPU interface；
5. CPU上有两个引脚irq、fiq，gic最终会连接到CPU的irq，所有寄存器配置完毕后，按键一旦按下，那么就会给CPU的irq发送一个中断信号，cpu紧接着就会执行“4大步3小步”，进入中断异常处理流程。

三、中断分类

1. 中断源

硬件中断（Hardware Interrupt）

1. 可屏蔽中断（maskable interrupt）。硬件中断的一类，可通过在中断屏蔽寄存器中设定位掩码来关闭。
2. 非可屏蔽中断（non-maskable interrupt，NMI）。硬件中断的一类，无法通过在中断屏蔽寄存器中设定位掩码来关闭。典型例子是时钟中断（一个硬件时钟以恒定频率—如50Hz—发出的中断）。
3. 处理器间中断（interprocessor interrupt）。一种特殊的硬件中断。由处理器发出，被其它处理器接收。仅见于多处理器系统，以便于处理器间通信或同步。
4. 伪中断（spurious interrupt）。一类不希望被产生的硬件中断。发生的原因有很多种，如中断线路上电气信号异常，或是中断请求设备本身有问题。

软件中断（Software Interrupt）

软件中断SWI,是一条CPU指令，用以自陷一个中断。由于软中断指令通常要运行一个切换CPU至内核态的子例程，它常被用作实现系统调用（System call）。

外部中断

1. I/O设备：如显示器、键盘、打印机、A / D转换器等。
2. 数据通道：软盘、硬盘、光盘等。数据通道中断也称直接存储器存取（DMA）操作中断，如磁盘、磁带机或CRT等直接与存储器交换数据所要求的中断。
3. 实时时钟：如外部的定时电路等。在控制中遇到定时检测和控制，为此常采用一个外部时钟电路（可编程）控制其时间间隔。需要定时时，CPU发出命令使时钟电路开始工作，一旦到达规定时间，时钟电路发出中断请求，由CPU转去完成检测和控制工作。
4. 用户故障源：如掉电、奇偶校验错误、外部设备故障等。

产生于CPU内部的中断源

1. 由CPU得运行结果产生：如除数为0、结果溢出、断点中断、单步中断、存储器读出出错等。
2. 执行中断指令swi
3. 非法操作或指令引起异常处理。

2. 中断类型

GIC 中断类型有3种：SGI(Software-generated interrupt)、PPI(Private peripheral interrupt)、SPI(Shared peripheral interrupt)。

1. SGI: SGI为软件可以触发的中断，统一编号为0~15(ID0-ID7是不安全中断，ID8-ID15是安全中断)，用于各个core之间的通信。该类中断通过相关联的中断号和产生该中断的处理器的CPUID来标识。通常为边沿触发。

2. PPI: PPI为每个 core 的私有外设中断，统一编号为 16-31 。例如每个 CPU 的 local timer 即 Arch Timer 产生的中断就是通过 PPI 发送给 CPU 的（安全为29，非安全为30）。

通常为边沿触发和电平触发。

3. SPI: SPI 是系统的外设产生的中断，为各个 core 公用的中断，统一编号为 32~1019，如 global timer 、uart 、gpio 产生的中断。通常为边沿触发和电平触发。

Note：电平触发是在高或低电平保持的时间内触发，而边沿触发是由高到低或由低到高这一瞬间触发；在GIC中PPI和SGI类型的中断可以有相同的中断ID。

3. 中断分派模式

1. 1-N mode（SPIs using the GIC 1-N model）表示中断可以发给所有的CPU，但只能由一个CPU来处理中断；换句话说，这种类型的中断有N个目标CPU，但只能由其中一个来处理；当某一个处理器应答了该中断，便会清除在所有目标处理器上该中断的挂起状态。

2. N-N mode（PPIs and SGIs using the GIC N-N model）表示中断可以发给所有CPU，每个CPU可以同时处理该中断。当该中断被某一个处理器应答了，这不会影响该中断在其他CPU接口上的状态。

举两个例子说明:

1）UART 接收到一包数据，产生了一个中断给GIC，GIC可以将该中断分配给CPU0-7中任何一个处理；假设该中断分配给CPU0处理了，那么在中断处理函数里面会把接收到的数据从

UART FIFO读出。可以想象一下，如果CPU0在读数据时，另外一个CPU也在处理该中断，恰巧也在读数据，那么CPU0读到的数据是不全的。这就是1-N model中断，或者说SPI中断。

2）比如CPU0给CPU1-7发送中断，想告知对方自己正在处理某个进程A。这种场景下，CPU1-7都接收到中断，都进入中断处理函数，CPU1-7获取到CPU0的信息后，在进程调度时，就可以绕开进程A，而自己调度其他进程。

注：这个例子只是说明N-N model，实际上进程调度不都全是这样的。

4. 通用中断处理

当GIC接收到一个中断请求，将其状态设置为Pending。重新产生一个挂起状态的中断不影响该中断状态。

中断处理顺序：

① GIC决定该中断是否使能，若没有被使能对GIC没有影响；

② 对于每个Pending中断，GIC决定目标处理器；

③ 对于每个处理器 ，Distributor根据它拥有的每个中断优先级信息决定最高优先级的挂起中断，将该中断传递给目标CPU Interface；

④ GIC Distributor将一个中断传递给CPU Interface后，该CPU Interface决定该中断是否有足够的优先级将中断请求发给CPU；

⑤ 当CPU开始处理该异常中断，它读取GICC_IAR应答中断。读取的GICC_IAR获取到中断ID，对于SGI，还有源处理器ID。中断ID被用来查找正确的中断处理程序。

GIC识别读过程后，将改变该中断的状态：

a）当中断状态变为active时，如果该中断挂起状态持续存在或者中断再次产生，中断状态将从Pending转化为pending & active

b）否则，中断状态将从pending状态变为active

⑥ 当中断完成中断处理后，它需要通知GIC处理已经完成。这个过程称为 priority drop and interrupt deactivation：

a）总是需要向EOIR寄存器写入一个有效的值(end of interrupt register) b）也需要接着向GICC_DIR写入值(deactivate interrupt register)

5. 中断优先级

软件可以通过给每一个中断源分配优先级值来配置中断优先级。优先级的值是个8位的无符号二进制数，GIC支持最小16和最大256的优先级级别。

如果GIC实现的优先级少于256，那么优先级字段的低阶位为RAZ/WI。这就意味着实现的优先级字段个数范围是4～8，如下图所示：

Implemented priority bits	Possible priority field values	Number of priority levels
[7:0]	0x00-0xFF (0-255), all values	256
[7:1]	0x00-0xFE, (0-254), even values only	128
[7:2]	0x00-0xFC (0-252), in steps of 4	64
[7:3]	0x00-0xF8 (0-248), in steps of 8	32
[7:4]	0x00-0xF0 (0-240), in steps of 16	16

Effect of not implementing some priority field bits

Note:

1)、如何确定优先级字段所支持的优先级位？通过软件往可写GICD_IPRIORITYn优先级字段写入0XFF，然后回读出该字段的值便可以确定优先级字段所支持的优先级位(因为有些位没实现是RAZ/WI)

2)、ARM 推荐在检查中断优先级范围之前先：• 对于外设中断，软件先禁用该中断 • 对于SGI，软件先检查该中断确定为inactive

6. 中断抢占

在一个active中断处理完之前，CPU interface支持发送更高优先级的挂起中断到目标处理器。这种情况必要条件如下：

- 该中断的优先级高于当前CPU interface 被屏蔽的优先级
- 该中断的组优先级高于正在当前CPU interface处理的中断优先级

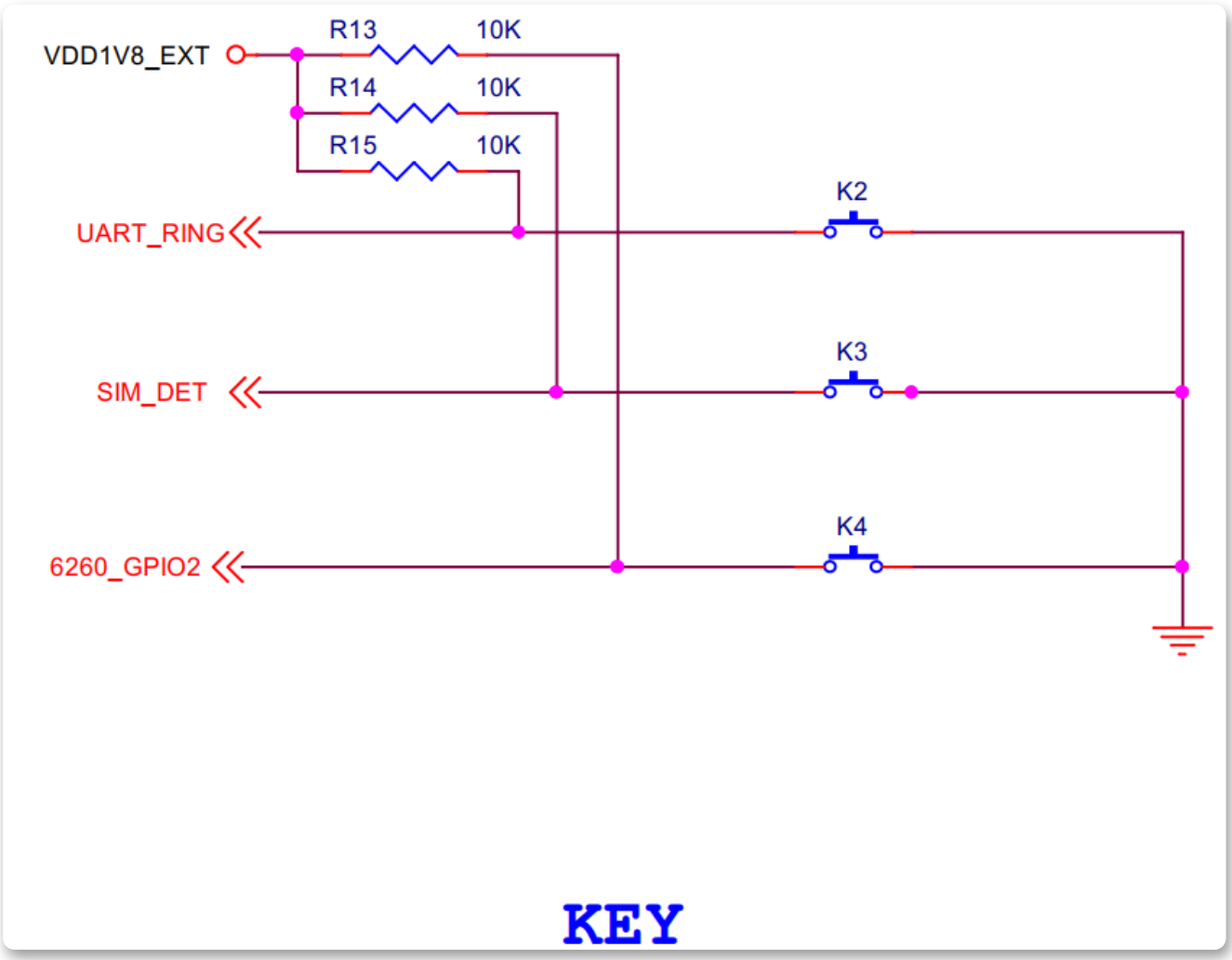
7. 中断屏蔽

CPU interface的GICC_PMR寄存器定义了目标处理器的优先级阀值，GIC仅上报优先级高于阀值的pending中断给目标处理器。寄存器初始值为0，屏蔽所有的中断。

四、FS4412中断外设-key

下面我们来分析FS4412开发板的第一个中断设备按键。

1．电路图



key

由该电路图可得：

1．按键k2 连接在GPX1_1引脚

2．控制逻辑 k2 按下 ---- K2闭合 ---- GPX1_1 低电压 k2 常态 ---- K2打开 ---- GPX1_1 高电压

以下是key2与soc的连接，

key与soc的连接

可以看到key2复用了GPIX1_1这个引脚，同时该引脚还可以作为中断【XEINT9】使用。

顺便看下GPXCON寄存器的配置

6.2.3.198 GPX1CON

- Base Address: 0x1100_0000
- Address = Base Address + 0x0C20, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
GPX1CON[1]	[7:4]	RW	0x0 = Input 0x1 = Output 0x2 = Reserved 0x3 = KP_COL[1] 0x4 = Reserved 0x5 = ALV_DBG[5] 0x6 to 0xE = Reserved 0xF = WAKEUP_INT1[1]	0x00

GPX1CON

由上图所示，

1．GPX1CON地址为0x1100C20;

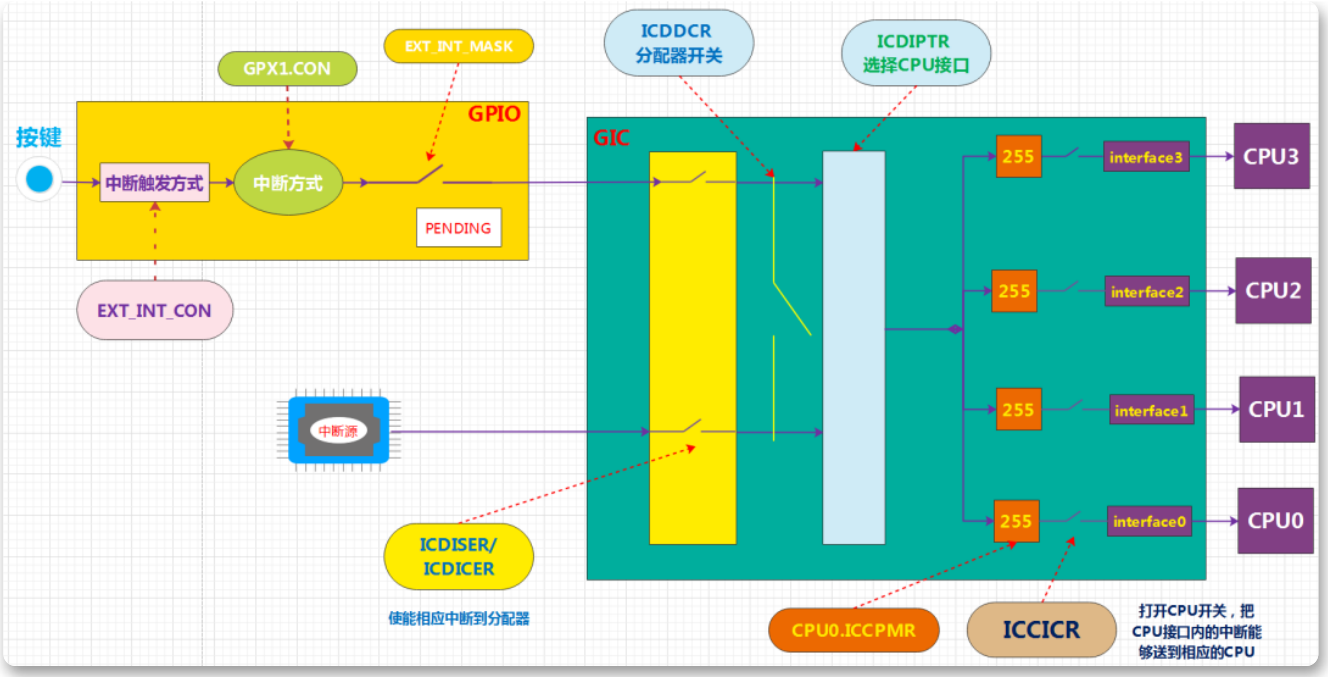
2．key2如果要做为输入设备，只需要将GPX1CON[7:4]设置为0x0；

3．key2如果要做为中断信号，只需要将GPX1CON[7:4]设置为0xf。

2．key中断处理

中断配置

key与soc的关系图如下图所示：



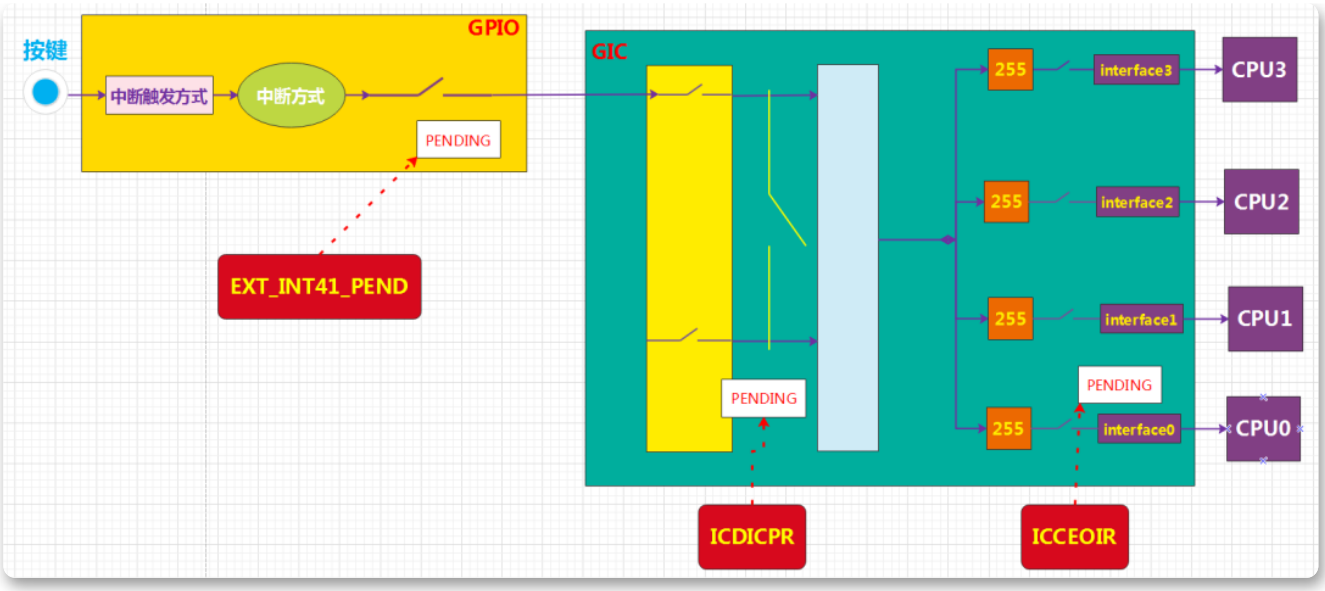
按键中断寄存器配置流程

由上图所示：

1. 按键是直接连到GPIO控制器的
2. EXT_INT_CON用来设置按键中断的触发方式，下降沿触发
3. GPX1CON寄存器用于设置该GPIO位中断信号输入
4. EXT_INT_MASK用于使能该中断
5. ICDISER用于使能相应中断到分配器
6. ICDDCR分配器开关
7. ICDIPTR选择CPU接口
8. ICCPMR设置中断屏蔽优先级
9. ICCICR打开CPU开关，把CPU接口内的中断能够送到相应的CPU

清中断

CPU处理完中断，需要清除中断，对于按键来说，有3个寄存器需要操作：



清中断

由上图所示：

1．EXT_INT41_PEND清相应的中断源

2．ICDICPR中断结束后，清相应中断标志位，此标志位由硬件置位

3．ICCEOIR中断执行结束，清cpu内相应的中断号，由硬件填充

3．寄存器汇总

前面分析了按键连接的是GPX1_1，现在我们来看下对应的寄存器应该如何配置

【1】、GPIO控制器

1．GPX1PUD

Name	Bit	Type	Description	Reset Value
GPX1PUD[n]	[2n + 1:2n] N = 0 to 7	RW	0x0 = Disables Pull-up/Pull-down 0x1 = Enables Pull-down 0x2 = Reserved 0x3 = Enables Pull-up	0x5555

将GPX1_1引脚的上拉和下拉禁止

```
GPX1PUD[3:2]= 0b00;
```

2．GPX1CON

GPX1CON[1]	[7:4]	RW	0x0 = Input 0x1 = Output 0x2 = Reserved 0x3 = KP_COL[1] 0x4 = Reserved 0x5 = ALV_DBG[5] 0x6 to 0xE = Reserved 0xF = WAKEUP_INT1[1]	0x00

GPX1CON

将GPX1_1引脚功能设置为中断功能

```
GPX1CON[7:4] = 0xf
```

3. EXT_INT41CON

EXT_INT41_CON[1]	[6:4]	W	Sets signaling method of EXT_INT41[1] 0x0 = Low level 0x1 = High level 0x2 = Triggers Falling edge 0x3 = Triggers Rising edge 0x4 = Triggers Both edge 0x5 to 0x7 = Reserved	0x0

EXT_INT41CON

配置成成下降沿触发：

```
EXT_INT41CON[6:4] = 0x2
```

4. EXT_INT41_MASK

EXT_INT41_MASK[1]	[1]	RW	0x0 = Enables Interrupt 0x1 = Masked	0x1

EXT_INT41_MASK

中断使能寄存器

```
EXT_INT41_MASK[1] = 0b0
```

5. EXT_INT41_PEND 中断状态寄存器

EXT_INT41_PEND[1]	[1]	RWX	0x0 = Not occur 0x1 = Interrupt Occurs	0x0

EXT_INT41_PEND

当GPX1_1引脚接收到中断信号，中断发生，中断状态寄存器EXT_INT41_PEND 相应位会自动置1 注意：中断处理完成的时候，需要清除相应状态位。置1清0.

```
EXT_INT41_PEND[1] =0b1
```

【2】GIC

根据外设中断名称EINT9来查看该中断对应的GIC中维护的HW id。【所有的中断源在芯片厂商设计的时候都分配了唯一的一个ID，GIC通过该ID来驱动中断源】

查看芯片手册（datasheet -- 9.2表）

Table 9-2 GIC Interrupt Table (SPI[127:0])

SPI Port No	ID	Int_I_Combiner	Interrupt Source	Source Block
25	57	–	EINT[9]	External Interrupt

GIC中断源表

通过【9.2中断源表】找到和外设中断标示对应的中断控制器中断标识（GPIO有32个可被唤醒寄存器）其对应「EINT[9]，中断ID为57」，这是非常重要的，在后面的寄存器设置中起很大作用；

1） ICDISER使能相应中断到分配器

- Base Address: 0x1049_0000
- Address = Base Address + 0x0100, Reset Value = 0x0000_FFFF (ICDISER0_CPU0)
- Address = Base Address + 0x0104, Reset Value = 0x0000_0000 (ICDISER1_CPU0)
- Address = Base Address + 0x0108, Reset Value = 0x0000_0000 (ICDISER2_CPU0)
- Address = Base Address + 0x010C, Reset Value = 0x0000_0000 (ICDISER3_CPU0)
- Address = Base Address + 0x0110, Reset Value = 0x0000_0000 (ICDISER4_CPU0)
- Address = Base Address + 0x4100, Reset Value = 0x0000_FFFF (ICDISER0_CPU1)
- Address = Base Address + 0x8100, Reset Value = 0x0000_FFFF (ICDISER0_CPU2)
- Address = Base Address + 0xC100, Reset Value = 0x0000_FFFF (ICDISER0_CPU3)

Name	Bit	Type	Description	Reset Value
Set-enable bits	[31:0]	RW	For SPIs and PPIs, for each bit: Reads) 0 = Disables the corresponding interrupt. 1 = Enables the corresponding interrupt. Writes) 0 = No effect. 1 = Enables the corresponding interrupt. A subsequent Read of this bit returns the value 1.	0x0

ICDISER

ICDISER用于使能相应中断到分配器，一个bit控制一个中断源，一个ICDISER可以控制32个中断源，这里INT[9]对应的中断ID为57，所以在ICDSER1中进行设置，57/32 =1余25，所以这里在ICDISER1第25位置一。

```
ICDISER.ICDISER1 |= (0x1 << 25);     //57/32 =1...25 取整数（那个寄存器） 和余
```

2. ICDIPTR选择CPU接口

- Address = Base Address + 0x0820, Reset Value = 0x0000_0000 (ICDIPTR8_CPU0)
- Address = Base Address + 0x0824, Reset Value = 0x0000_0000 (ICDIPTR9_CPU0)
- Address = Base Address + 0x0828, Reset Value = 0x0000_0000 (ICDIPTR10_CPU0)
- Address = Base Address + 0x082C, Reset Value = 0x0000_0000 (ICDIPTR11_CPU0)
- Address = Base Address + 0x0830, Reset Value = 0x0000_0000 (ICDIPTR12_CPU0)
- Address = Base Address + 0x0834, Reset Value = 0x0000_0000 (ICDIPTR13_CPU0)
- Address = Base Address + 0x0838, Reset Value = 0x0000_0000 (ICDIPTR14_CPU0)
- Address = Base Address + 0x083C, Reset Value = 0x0000_0000 (ICDIPTR15_CPU0)
- Address = Base Address + 0x0840, Reset Value = 0x0000_0000 (ICDIPTR16_CPU0)
- Address = Base Address + 0x0844, Reset Value = 0x0000_0000 (ICDIPTR17_CPU0)
- Address = Base Address + 0x0848, Reset Value = 0x0000_0000 (ICDIPTR18_CPU0)

ICDIPTR

Name	Bit	Type	Description	Reset Value
CPU targets, byte offset 3	[31:24]	RW	Processors in the system number from 0, and each bit in a CPU targets field refers to the corresponding processor. Refer to *Table 9-10* for more information.	0x0
CPU targets, byte offset 2	[23:16]	RW		0x0
CPU targets, byte offset 1	[15:8]	RW		0x0
CPU targets, byte offset 0	[7:0]	RW	For example, a value of 0x3 means that the Pending interrupt is sent to processors 0 and 1. For ICDIPTR0 to ICDIPTR7, a Read of any CPU targets field returns the number of the processor that performs the read.	0x0

ICDIPTR

Table 9-10 Meaning of CPU Targets Field Bit Values

CPU Targets Field Value	Interrupt Targets
0bxxxxxxx1	CPU interface 0
0bxxxxxx1x	CPU interface 1
0bxxxxx1xx	CPU interface 2
0bxxxx1xxx	CPU interface 3
0bxxx1xxxx	CPU interface 4
0bxx1xxxxx	CPU interface 5
0bx1xxxxxx	CPU interface 6
0b1xxxxxxx	CPU interface 7

选择cpu

ICDIPTR寄存器每8个bit 控制一个中断源，其中CPU0可以处理160个中断源，所以需要40个寄存器。要选择cpu0第一个bit必须是1。

设置SPI[25]/ID[57]由cpu0处理，57/4=16余1 所以选择寄存器ICDIPTR14的第2个字节[15:8]。

```
 //SPI 25 interrupts are sent to processor 0
 //57/4 = 14..1 14号寄存器的[15:8]
ICDIPTR.ICDIPTR14 |= 0x01<<8;
```

3. ICDDCR使能分配器

9.5.1.12 ICDDCR

- Base Address: 0x1049_0000
- Address = Base Address + 0x0000, Reset Value = 0x0000_0000

Name	Bit	Type	Description	Reset Value
RSVD	[31:1]	–	Reserved	0x0
Enable	[0]	RW	Global enabled for monitoring peripheral interrupt signals and forwarding pending interrupts to the CPU interfaces. 0 = GIC ignores all peripheral interrupt signals and does not forward pending interrupts to the CPU interfaces. 1 = GIC monitors the peripheral interrupt signals and forwards pending interrupts to the CPU interfaces.	0x0

还寄存器用于使能分配器。

```
ICDDCR =1;
```

4. ICCPMR 优先级屏蔽寄存器，设置cpu0能处理所有的中断。比如中断屏蔽优先级为255，该值表示优先级最低，所有的中断都能响应。

9.5.1.2 ICCPMR_CPUn

- Base Address: 0x1048_0000
- Address = Base Address + 0x0004, Reset Value = 0x0000_0000 (ICCPMR_CPU0)
- Address = Base Address + 0x4004, Reset Value = 0x0000_0000 (ICCPMR_CPU1)
- Address = Base Address + 0x8004, Reset Value = 0x0000_0000 (ICCPMR_CPU2)
- Address = Base Address + 0xC004, Reset Value = 0x0000_0000 (ICCPMR_CPU3)

Name	Bit	Type	Description	Reset Value
RSVD	[31:8]	–	Reserved	0x0
Priority	[7:0]	RW	The priority mask level for the CPU0 interface When the priority of an interrupt is higher than the value that this field indicates, the interface signals the interrupt to the processor. 256 priority levels support 0x00 – 0xFF (0 to 255), all values	0x0

ICCPMR

```
CPU0.ICCPMR = 0xFF;//设置cpu0 中断屏蔽优先级为255  最低，所有中断都能响应）
```

5. ICCICR 全局使能cpu0中断处理

9.5.1.1 ICCICR_CPUn

- Base Address: 0x1048_0000
- Address = Base Address + 0x0000, Reset Value = 0x0000_0000 (ICCICR_CPU0)
- Address = Base Address + 0x4000, Reset Value = 0x0000_0000 (ICCICR_CPU1)
- Address = Base Address + 0x8000, Reset Value = 0x0000_0000 (ICCICR_CPU2)
- Address = Base Address + 0xC000, Reset Value = 0x0000_0000 (ICCICR_CPU3)

Name	Bit	Type	Description	Reset Value
RSVD	[31:1]	–	Reserved	0x0
Enable	[0]	RW	Global enable for signaling of interrupts by the CPU Interface to the connected processors. 0 = Disables signaling of interrupts 1 = Enables signaling of interrupts	0x0

It enables the signaling of interrupts to the target processors. In a GIC that implements the Security Extensions, this register provides additional global controls for handling Secure interrupts. This register is banked to provide Secure and Non-secure copies.

ICCICR

EXYNOS 4412一共有4个cpu,用4个寄存器分别来控制4个cpu，每个寄存器的bit[0]用于全局控制对应的cpu。我们选择cpu0处理中断，将bit[0]置1即可。

```
CPU0.ICCICR |= 0x1;
使能中断到CPU。
```

6. ICCIAR

9.5.1.4 ICCIAR_CPUn

- Base Address: 0x1048_0000
- Address = Base Address + 0x000C, Reset Value = 0x0000_03FF (ICCIAR_CPU0)
- Address = Base Address + 0x400C, Reset Value = 0x0000_03FF (ICCIAR_CPU1)
- Address = Base Address + 0x800C, Reset Value = 0x0000_03FF (ICCIAR_CPU2)
- Address = Base Address + 0xC00C, Reset Value = 0x0000_03FF (ICCIAR_CPU3)

Name	Bit	Type	Description	Reset Value
RSVD	[31:13]	–	Reserved	0x0
CPUID	[12:10]	R	For SGIs, in a multiprocessor implementation, this field identifies the processor that requests the interrupt. It returns the number of the CPU interface that made the request. For all other interrupts, this field returns as zero.	0x0
ACKINTID	[9:0]	R	The interrupt ID	0x3FF

ICCIAR

当中断发生之后，中断的HW id值会由硬件写入到寄存器ICCIAR[9:0]中；对于SGIs来说，多处理器环境下，CPU的interface值写入到[12:10]中。

读取HW id：

```
int irq_num;
irq_num = CPU0.ICCIAR&0x3ff;  //获取中断号
```

五、代码实现

要处理中断异常，必须安装异常向量表，异常的处理流程可以参考前面的文章《6. 从0开始学ARM-异常、异常向量表、swi》

1. 异常向量表基址

异常向量表地址是可以修改的，比如uboot在启动的时候，会从flash中搬运代码到RAM中，而flash的异常向量表地址和ram的地址肯定不一样，所以搬运完代码后，就必须要修改对应的异常向量表地址。

修改异常向量表的地址的需要借助协处理器指令mcr：

```
ldr r0,=0x40008000
mcr p15,0,r0,c12,c0,0  @ Vector Base Address Register
```

上述命令是将地址0x40008000设置为异常向量表的地址，关于mcr指令，我们没有必要深究，知道即可。

RAM中异常向量表地址我们选用的是0x40008000，以下是exynos4412 地址空间分布。

exynos4412 地址分布

2. 异常向量表安装

```
.text
.global _start
_start:
  b  reset
  ldr  pc,_undefined_instruction
  ldr  pc,_software_interrupt
  ldr  pc,_prefetch_abort
```

```
  ldr  pc,_data_abort
  ldr  pc,_not_used
  ldr  pc,=irq_handler
  ldr  pc,_fiq
reset:

 ldr r0,=0x40008000

 mcr p15,0,r0,c12,c0,0  @ Vector Base Address Register
init_stack:
//初始化栈
……
b main //跳转至c的main函数

irq_handler: //中断入口函数

 sub  lr,lr,#4
 stmfd sp!,{r0-r12,lr}
 .weak do_irq
 bl do_irq
 ldmfd sp!,{r0-r12,pc}^

stacktop:    .word   stack+4*512//栈顶
.data
stack:  .space  4*512 //栈空间
```

中断入口函数do_irq()

```
void do_irq(void)
{
 static int a = 1;
 int irq_num;
 irq_num = CPU0.ICCIAR&0x3ff;  //获取中断号
 switch(irq_num)
 {
  case 57:
   printf("in the irq_handler\n");
   //清GPIO中断标志位
   EXT_INT41_PEND = EXT_INT41_PEND |((0x1 << 1));
   //清GIC中断标志位
   ICDICPR.ICDICPR1 = ICDICPR.ICDICPR1 | (0x1 << 25);
  break;
 }
  //清cpu中断标志
 CPU0.ICCEOIR = CPU0.ICCEOIR&(~(0x3ff))|irq_num;位
}
```

实现按键中断的初始化函数key_init()：

```
void key_init(void)
{
    GPX1.CON =GPX1.CON & (~(0xf << 4)) |(0xf << 4); //配置引脚功能为外部中断
    GPX1.PUD = GPX1.PUD & (~(0x3 << 2));  //关闭上下拉电阻
    EXT_INT41_CON = EXT_INT41_CON &(~(0xf << 4))|(0x2 << 4); //外部中断触发方
    EXT_INT41_MASK = EXT_INT41_MASK & (~(0x1 << 1));  //使能中断
    ICDDCR = 1;  //使能分配器
    ICDISER.ICDISER1 = ICDISER.ICDISER1 | (0x1 << 25); //使能相应中断到分配器
    ICDIPTR.ICDIPTR14 = ICDIPTR.ICDIPTR14 & (~(0xff << 8))|(0x1 << 8); //选
    CPU0.ICCPMR = 255; //中断屏蔽优先级
    CPU0.ICCICR = 1;   //使能中断到CPU
   return ;
}
```

六、轮询方式

除了中断方式之外我们还可以通过轮询方式读取按键的信息，原理如下：

循环检测GPX1_1引脚输入的电平，为低电压时，按键按下，为高电平时，按键抬起。

1．配置GPX1_1引脚功能为输入，设置内部上拉下拉禁止。

```
GPX1.CON = GPX1.CON &(~(0xf<<4)) ;
GPX1.PUD = GPX1.PUD & ~(0x3 << 2);
```

2．按键消抖：按键按下后由于机械特性，会在极短的时间内出现电平忽0忽1，所以我们检测到按键按下后，需要给一个延时，然后再判断按键是不是仍然按下。

3．代码实现

```
int main (void)
{
 led_init();
 pwm_init();
 GPX1.CON = GPX1.CON &(~(0xf<<4))|0x0<<4;
 while(1)
 {
     if(!(GPX1.DAT & (0x1<<1)))  // 返回为真，按键按下
     {
      delay_ms(10);
         if(!(GPX1.DAT & (0x1<<1))) //二次检测，去抖
         {
             GPX2.DAT |= 0x1 << 7;  //Turn on LED2
             delay_ms(500);
             beep_on();
             GPX2.DAT &= ~(0x1<<7);  //Turn off LED2

             delay_ms(500);

             while(!(GPX1.DAT & (0x1<<1)));
             beep_off();
         }
     }
 }
   return 0;
}
```

推荐阅读

【1】【从0学ARM】你不了解的ARM处理异常之道

【2】为什么使用结构体效率比较高？ **必读**

【3】9. 基于Cortex-A9 LED汇编、C语言驱动编写**必读**

【4】一文包你学会网络数据抓包**必读**

【5】10. 基于Cortex-A9的pwm详解 **必读**

进群，请加一口君个人微信，带你嵌入式入门进阶。

收录于合集 #从0学arm 27

上一篇
10. 基于Cortex-A9的pwm详解

下一篇
12. 如何基于Cortex-A9的UART从头实现printf函数

阅读原文

喜欢此内容的人还喜欢

Konva实现图片自适应裁剪
A逐梦博客

面试连环问--操作系统
阿Q正砖

pinia

睡不着所以学编程