|  |
| --- |
|  |
| Codezero源码分析 |
| ——drivers目录 |

|  |
| --- |
|  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 当前版本： | 1.0 | | 日期： | 2014-07-01 | | 作者： | 康乔 | |

版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本状态 | 作者 | 参与者 | 起止日期 | 备注 |
| 1.0 | 康乔 |  | 2014-06-30至2014年7月20日 |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 概述

本目录为drivers目录，包含了一些对底层硬件支持的驱动代码。例如中断控制器、定时器（计数器）、通用异步首发传输器（UART）等。

# 总体功能描述

下面按照目录结构介绍drivers目录下的所有文件。

1. irq：中断控制器的驱动程序
   1. gic 包含gic.c 通用的中断控制器支持程序
   2. omap3 包含 omap3\_init.c omap3处理器的中断控制器支持程序
   3. pl190 包含 pl190\_vic.c pl190 vectored中断控制器的支持程序
2. timer
   1. omap 包含timer.c omap GP 定时器驱动
   2. sp804包含timer.c SP804 primecell驱动
3. uart
   1. omap 包含uart.c omap设备使用的UART驱动
   2. pl011 包含 uart.c pl011的UART驱动

# Sconscript文件

在drivers目录的所有子目录下，每有一个.c文件，都对应着一个Sconscript文件，所有的Sconstruct文件结构类似。下面以drivers/timer/omap/Sconscript文件为例，文件代码如下：

Import('env')

from scripts.config.projpaths import \*

from scripts.config.config\_invoke import \*

config = configuration\_retrieve()

platform = config.platform

# Platforms using omap\_timer

#这里定义一个列表，包含了所有使用该设备的体系结构列表

plat\_list = 'beagle'

# The set of source files associated with this SConscript file.

#这里定义一个列表，包含与该SC文件有关的所有源代码文件

src\_local = []

#如果当前正在编译的体系结构需要用到该设备驱动，则把该文件包含进去。

if plat\_list == platform:

src\_local += ['timer.c']

#编译源代码并返回

obj = env.Object(src\_local)

Return('obj')

# 文件1：irq/gic/gic.c

## 功能描述（对于复杂操作建议画出流程图）

本文件中1-8函数为主要的功能函数。剩余的函数为可能用到的辅助函数。

1. **static** inline **struct** gic\_data **\***get\_gic\_data**(**l4id\_t irq**)  
   该函数作用为：以一个中断请求号irq为参数，通过中断向量表结构，找到该中断对应的控制芯片，从而找到芯片的数据信息。**
2. l4id\_t gic\_read\_irq**(void** **\***data**)  
   该函数为1的逆操作，效果为以芯片的data为参数，找到对应的irq并返回**
3. **gic\_mask\_irq和gic\_unmask\_irq函数为对irq对应的gic\_data中的某个字段进行一些位操作，具体作用不详**
4. **void** gic\_eoi\_irq**(**l4id\_t irq**)  
   该函数的作用为，把irq对应的gic\_data字段的中cpu->eoi字段的值设置为irq**
5. **void** gic\_ack\_and\_mask**(**l4id\_t irq**)**  
   该函数的作用为调用gic\_mask\_irq和gic\_eoi\_irq两个函数。
6. **gic\_set\_pending和gic\_clear\_pending同3，也是对gic\_data中的字段进行位操作。**
7. **void** gic\_cpu\_init**(int** idx**,** **unsigned** **long** base**)  
   详见代码注释。**
8. **void** gic\_dist\_init**(int** idx**,** **unsigned** **long** base**)  
   该函数与7类似，是初始化gic\_data[idx]的dist字段。**
9. **void** gic\_set\_target**(**l4id\_t irq**,** u32 cpu**)**作用为设置gic\_data的dist->target字段。
10. u32 gic\_get\_target**(**u32 irq**)  
    获得gic\_data的dist->target字段**
11. **void** gic\_set\_priority**(**u32 irq**,** u32 prio**)  
    与9类似**
12. u32 gic\_get\_priority**(**u32 irq**)  
    与10类似**

## 数据结构

本文件包含的数据结构主要是gic\_data：

**struct** gic\_data **{**  
    **struct** gic\_cpu **\***cpu**;**  
    **struct** gic\_dist **\***dist**;**  
**};**

gic\_data中的gic\_cpu和gic\_dist定义分别如下：

**struct** gic\_cpu  
**{**  
    u32   control**;**      */\* Control Register \*/*  
    u32   prio\_mask**;**        */\* Priority Mask \*/*  
    u32   bin\_point**;**        */\* Binary Point Register \*/*  
    u32   ack**;**          */\* Interrupt \*/*  
    u32   eoi**;**          */\* End of Interrupt \*/*  
    u32   running**;**      */\* Running Priority register \*/*  
    u32   high\_pending**;**     */\* Highest Pending Register \*/*  
**};**

**struct**  gic\_dist**{**  
    u32   control**;**              */\* Control Register \*/*  
    u32   **const**   type**;**             */\* Type Register \*/*  
    u32   dummy1**[**62**];**               */\* -r- \*/*  
    u32   set\_en**[**NREGS\_1\_BIT\_PER\_INT**];**      */\* Enable Set \*/*  
    u32   clr\_en**[**NREGS\_1\_BIT\_PER\_INT**];**      */\* Enable Clear \*/*  
    u32   set\_pending**[**NREGS\_1\_BIT\_PER\_INT**];** */\* Set Pending \*/*  
    u32   clr\_pending**[**NREGS\_1\_BIT\_PER\_INT**];** */\* Clear Pending \*/*  
    u32   active**[**NREGS\_1\_BIT\_PER\_INT**];**      */\* Active Bit registers \*/*  
    u32   dummy2**[**32**];**               */\* -r- \*/*  
    u32   priority**[**NREGS\_4\_BIT\_PER\_INT**];**    */\* Interrupt Priority \*/*  
    u32   target**[**NREGS\_4\_BIT\_PER\_INT**];**      */\* CPU Target Registers \*/*  
    u32   config**[**NREGS\_2\_BIT\_PER\_INT**];**      */\* Interrupt Config \*/*  
    u32   level**[**NREGS\_2\_BIT\_PER\_INT**];**       */\* Interrupt Line Level \**

    u32   dummy3**[**64**];**               */\* -r- \*/*  
    u32   soft\_int**;**             */\* Software Interrupts \*/*  
    u32   dummy4**[**55**];**               */\* -r- \*/*  
    u32   id**[**NID**];**              */\* Primecell ID registers \*/*  
**};**

## 代码注释

**void** gic\_cpu\_init**(int** idx**,** **unsigned** **long** base**)**  
**{  
// 定义一个指向gic\_cpu结构的指针cpu**  
    **volatile** **struct** gic\_cpu **\***cpu**;**  
//为gic\_data结构中的cpu字段指定地址  
    gic\_data**[**idx**].**cpu **=** **(struct** gic\_cpu **\*)**base**;**  
//为cpu赋值，方便下面代码的书写

//进行一些初始化工作。  
    cpu **=** gic\_data**[**idx**].**cpu**;**  
    */\* Disable \*/*  
    cpu**->**control **=** 0**;**  
    */\* Set \*/*  
    cpu**->**prio\_mask **=** 0xf0**;**  
    cpu**->**bin\_point **=** 3**;**  
    */\* Enable \*/*  
    cpu**->**control **=** 1**;**  
**}**

## 疑问或改进建议

暂无

## 体系结构和平台相关的代码

无

# 文件2：omap3\_init.c

## 功能描述（对于复杂操作建议画出流程图）

本文件中主要是使用到的函数有两个，另有若干个辅助函数。

1. **void** omap3\_intc\_init**(void)  
   该函数为omap3的中断控制器驱动程序的初始化程序**
2. **void** omap3\_intc\_reset**(unsigned** **long** base**)  
   该函数进行一些参数重置，被omap3\_intc\_init调用**

## 数据结构

无

## 代码注释

略

## 疑问或改进建议

无

## 体系结构和平台相关的代码

包含的头文件**/include/**l4/drivers/irq/omap3/omap3\_intc.h中包含平台相关的代码

# 文件3：pl190\_vic.c

## 功能描述（对于复杂操作建议画出流程图）

本文件中主要用到的函数为**void** pl190\_vic\_init**(void)，是pl190中断控制器驱动程序的初始化函数。**

## 数据结构

无

## 代码注释

略

## 疑问或改进建议

无

## 体系结构和平台相关的代码

无

# 文件4：timer/omap/timer.c与timer/ sp804/timer.c

## 功能描述（对于复杂操作建议画出流程图）

这两个文件分别针对omap的timer和sp804的timer的编写了驱动程序，并明确遵循了通用的计时器库函数API。

我们以omap的timer为例，来看一下它提供的API接口。

初始化的关键的函数是timer\_init\_periodic，调用它的函数是timer\_init。

Timer\_init\_periodic函数代码如下：

**void** timer\_init\_periodic**(unsigned** **long** timer\_base**)**  
**{**  
    **volatile** u32 reg**;**  
  
    */\* Reset the timer \*/*

*//重置寄存器*  
    timer\_reset**(**timer\_base**);**

//设置寄存器的模式  
    */\* Set timer to autoreload mode \*/*  
    reg **=** read**(**timer\_base **+** OMAP\_TIMER\_TCLR**);**  
    reg **|=** **(**1 **<<** OMAP\_TIMER\_MODE\_AUTORELAOD**);**  
    write**(**reg**,** timer\_base **+** OMAP\_TIMER\_TCLR**);**

//注释部分解释定时器粒度的计算  
    */\*  
    \* Beagle Board RevC manual:  
    \* overflow period = (0xffffffff - TLDR + 1)\*PS\*(1/TIMER\_FCLK)  
    \* where,  
    \* PS: Prescaler divisor (we are not using this)  
    \*  
    \* Beagle board manual says, 26MHz oscillator present on board.  
    \* U-Boot divides the sys\_clock by 2 if sys\_clk is >19MHz,  
    \* so,we have sys\_clk frequency = 13MHz  
    \*  
    \* TIMER\_FCLK = 13MHz  
    \* So, for 1ms period, TLDR = 0xffffcd38  
    \*  
    \*/*  
    timer\_load**(**timer\_base**,** 0xffffcd38**);**  
  
    */\* Clear pending Interrupts, if any \*/*  
    write**(**7**,** timer\_base **+** OMAP\_TIMER\_TISR**);**  
  
    */\* Enable inteerupts \*/*  
    write**((**1 **<<** OMAP\_TIMER\_INTR\_OVERFLOW**),** timer\_base **+** OMAP\_TIMER\_TIER**);**  
**}**

## 数据结构

无

# 文件5：uart/omap/uart.c

## 功能描述（对于复杂操作建议画出流程图）

该文件针对omap处理器的uart编写了驱动程序

## 数据结构

## 代码注释

## 疑问或改进建议

## 体系结构和平台相关的代码

# 文件6：uart/pl011/uart.c

## 功能描述（对于复杂操作建议画出流程图）

该文件针对pl011的uart编写了驱动程序

## 数据结构

## 代码注释

## 疑问或改进建议

## 体系结构和平台相关的代码

# 总结