DJYOS 定时器组件硬件接口说明

编写：朱海兵 2015 年 3 月 23 日
Review：_年_月_日
审阅:罗侍田

贡献者列表

DJYOS 开发团队。

1.概述

DJYOS 的定时器管理，分软件定时器(SoftTimer)和硬件定时器(HardTimer)两部分。

软件定时器组件，允许用户申请任意多个定时器（受内存容量限制），依用户所需要的精度，它的时钟源可以选择系统 tick，定时精度也不超过 tick 间隔；也可以选择硬件定时器，其精度只受硬件定时器本身和中断响应延迟的限制。

如何选择 tick 还是硬件定时器做时基呢？

软件定时器初始化时，如果参数选择 CN_TIMER_SOURCE_TICK。即：

ModuleInit_SoftTimer(CN_TIMER_SOURCE_TICK);

则表明使用 tick 作为时钟源。

如果选择 CN_TIMER_SOURCE_HARD，则使用硬件定时器作为时钟源。

如果用硬件定时器，则需要确认在“工程目录 \\src\user\critical\critical.c”文件的 critical 函数中调用 ModuleInit_HardTimer 函数的代码行不被注释。此时，ModuleInit_SoftTimer 函数会调用 HardTimer 模块的功能，申请硬件定时器，如果申请不到，则返回错误。

HardTimer 组件并不是 SoftTimer 组件专用的，它可以管理除 Tick 定时器以外的全部硬件定时器，提供通用的高精度定时服务。HardTimer 模块标准化了硬件接口，如果你需要使用硬件定时器服务，就需要编写标准接口函数，否则，本文剩余部分，就不需要阅读了。

当然，软件所需要的定时功能千差万别，硬件定时器提供的支持也百花齐放，HardTimer 模块不可能穷举所有可能，它只提供通用的定时服务。如果你所需要的定时功能咱没有提供，你大可绕开 timer 模块，天马行空，怎么写驱动都行。

关于 DJYOS 定时器组件的使用方法可详见《都江堰操作系统用户手册》的“定时器组件”章节。

2.移植的源码目录

如果使用 CPU 内置定时器，在 eclipse 工程中的目录中将有如下文件：

src->OS_code->bsp->cpudrv->src->cpu_peri_timer.c。

相应的头文件目录为：

src->OS_code->bsp->cpudrv-> src->cpu_peri_timer.h。

在文件系统（硬盘）中的目录结构是：

djysdk\djysrc\bsp\cpudrv\cpu_name\src\cpu_peri_timer.c。

下面阐述怎么编写这两个文件。

3.实现标准硬件接口函数

DJYOS 定时器底层驱动需要提供四个函数，分别是 ModuleInstall_HardTimer(HardTimer 初始化)及硬件定时器模块需要调用的三个钩子函数。

3.1 硬件定时器初始化

void ModuleInstall_HardTimer(void);

头文件：

cpu_peri_timer.h

参数：无。

返回值：无。

说明：

该函数功能为定时器初始化，需要完成以下四个工作：

1. 使能定时器时钟；
2. 配置时钟频率；
3. 对定时器芯片结构体 tagTimerChip 成员赋值；
4. 调用系统函数 TimerHard_RegisterChip 将该定时器注册到系统时钟芯片中。

tagTimerChip 在 timer_hard.h 中定义，具体如下：

```
struct tagTimerChip
{
    char                    *chipname;          //chip名字，必须为静态型
    fnTimerHardAlloc        timerhardalloc;     //分配定时器
    fnTimerHardFree         timerhardfree;      //释放定时器
    fnTimerHardCtrl         timerhardctrl ;     //操作控制定时器
};
```

tagTimerChip 中 fnTimerHardAlloc、fnTimerHardFree、fnTimerHardCtrl 为上述提及的三个钩子函数指针。

3.2 分配定时器

typedef ptu32_t (*fnTimerHardAlloc)(u32 cycle,fnTimerIsr timerisr);

头文件：

timer_hard.h

参数：

cycle：指定分配定时器的定时周期，该属性可使用 API 函数进行更改设定（单位：微秒）。

timerisr：用户实现的定时器中断服务函数，定时期限到后，由定时器中断调用。fnTimerIsr 的原型为：typedef u32 (*fnTimerIsr)(ufast_t irq_no)；其中 irq_no 为定时器对应的中断号。在 cpu_peri_int_line.h 中定义了 CPU 所有的中断源对应的中断号，在 critical.c 中定义了常量数组 tg_IntUsed，该数组用于配置实际工程使用到的中断源，详情见 cpu_peri_int_line.h 及

critical.c。根据实际使用的定时器编号可通过查看 tg_IntUsed 获得该定时器对应的中断号。另外需要注意的是由于硬件定时器中断属性设置为异步信号，DJYOS 内部已经实现了清中断，因此用户实现的中断服务函数 timerisr 不必清中断。

返回值：

NULL 分配不成功，否则返回定时器句柄，定时器句柄数据类型为 ptu32_t，其具体内容由 BSP 工程师根据各平台实际情况定义。

说明：

分配定时器函数需要完成以下工作：

1. 选出一个可使用的定时器并对其句柄进行复制；
2. 设置定时器属性，默认状态下：停止计数、设置定时器中断为异步中断并将其挂接到 DJYOS 的中断系统中，设置定时器周期；
3. 返回定时器句柄。

3.3 释放定时器

typedef bool_t (*fnTimerHardFree)(ptu32_t timerhandle);

头文件：

timer_hard.h

参数：

timerhandle，待释放定时器句柄。

返回值：

true 成功；false 失败。

说明：

该过程基本上是分配的逆向过程，将指定的定时器恢复到分配前的状态，一般要停止计数器、切断中断线等。

3.4 控制定时器

```
typedef bool_t (*fnTimerHardCtrl)(ptu32_t timerhandle,
                                  enum _ENUM_TIMER_CTRL_TYPE_ ctrlcmd,
                                  ptu32_t inoutpara);
```

头文件：

timer_hard.h

参数：

timerhandle：待操作的定时器句柄；

ctrlcmd： 操作命令；

inoutpara：输入输出参数，根据不同的情况而定。

返回值：

true 操作成功 ；false 操作失败。

说明：

该函数用于实现对某个指定定时器进行特定操作，根据操作码实现各自特定操作，在 timer_hard.h 中定义了定时器以下操作码如下：

```
enum _ENUM_TIMER_CTRL_TYPE_
```

```
{
    EN_TIMER_STARTCOUNT = 0, //使能计数
    EN_TIMER_PAUSECOUNT,     //停止计数
    EN_TIMER_SETCYCLE,       //设置周期，单位 us
    EN_TIMER_SETRELOAD,      //设置定时器为单次触发，还是周期性触发
    EN_TIMER_ENINT,          //中断使能
    EN_TIMER_DISINT,         //中断禁止
    EN_TIMER_SETINTPRO,      //中断属性设置，指将定时器中断设置为实时中
                             //断 or 异步信号。
    EN_TIMER_GETTIME,        //获取计时时间，指从设定的周期算起，即
                             // cycle-剩余时间,表示已经走掉的时间
    EN_TIMER_GETCYCLE,       //获取定时周期
    EN_TIMER_GETID,          //获取定时器 ID，采用一个 32 位数表示定时器 ID
                             //号，其中高 16 位为 intID(中断号) ，低 16 位为
                             // timerID(定时器号)
    EN_TIMER_GETSTATE,       //获取定时器状态，在 timer_hard.h 中定义了定时
                             //器的 5 种状态，详情见 timer_hard.h
};
```

在该函数中分别实现上述各个操作码对应的操作功能即可。

中断属性设置(EN_TIMER_SETINTPRO)方法为，当中断属性设置为异步信号，则只需调用系统函数 Int_EvttDisConnect(ufast_t ufl_line)将该中断设置为异步信号(ufl_line 为中断号)；如果需要设置为实时信号，需要先调用系统函数 Int_EvttDisConnect(ufast_t ufl_line)将原来挂接的异步信号中断线断开(由于在分配定时器时默认将定时器中断设置为异步信号)，然后调用系统函数 Int_SettoReal(ufast_t ufl_line)将该定时器中断类型设置为实时信号。

注意：如果定时器中断属性设为实时中断，则用户实现的中断服务函数 ISR 中必须清中断，且不能调用任何系统服务；如果设定为异步信号，则无须清中断，且允许调用全部系统调用。

3.自定义硬件接口

DJYOS 定时器组件提供了定时器的通用定时服务，但是软件实际需求的功能千变万化，有可能 DJYOS 的定时器组件不能完全满足用户的实际需求。由于一般硬件定时器个数通常有多个，用户完全可以根据实际情况将一部分定时器注册到 DJYOS 的定时器组件中，另一部分根据实际需要编写相应的驱动函数提供特殊的服务，例如用于 PWM。