s3c2410提供了5个16位的Timer（Timer0~Timer4），其中Timer0~Timer3支持Pulse Width Modulation—— PWM（脉宽调制 ）。Timer4是一个内部定时器（internal timer），  
PCLK 是Timer的信号源，我们通过设置每个Timer相应的Prescaler和Clock Divider把PCLK转换成输入时钟信号传送给各个Timer的逻辑控制单元(Control Logic),事实上每个Timer都有一个称为输入时钟频率（Timer input clock Frequency）的参数，这个频率就是通过PCLK，Prescaler和Clock Divider确定下来的，每个Timer 的逻辑控制单元就是以这个频率在工作。下面给出输入时钟频率的公式：

Timer input clock Frequency = PCLK / {prescaler value+1} / {clock divider }  
{prescaler value} = 0~255  
{ clock divider } = 2, 4, 8, 16

然而并不是每一个Timer都有对应的Prescaler和Clock Divider，从上面的原理图我们可以看到Timer0,Timer1共用一对Prescaler和Clock Divider,Timer2,Timer3,Timer4共用另一对Prescaler和Clock Divider，s3c2410的整个时钟系统模块只存在两对Prescaler和Clock Divider。

我曾经在讨论watchdog的文章中提到，watchdog也是一种定时器，他的工作就是在一个单位时间内对一个给定的数值进行递减和比较的操 作，而我们这篇文章讨论的定时器他的工作内容和watchdog在本质上是一样的。定时器在一个工作周期（Timer input clock cycle）内的具体工作内容主要有3个。分别是：

1．       对一个数值进行递减操作  
2．        把递减后的数值和另一个数值进行比较操作  
3．        产生中断或执行DMA操作

在启用Timer之前我们会对Timer进行一系列初始化操作，这些操作包括上面提到的设置Prescaler和Clock Divider，其中还有一个非常重要的就是要给Timer两个数值，我们分别称之为Counter(变量,用于递减)和Comparer（定值,用于比 较）,Counter会被Timer 加载到COUNT BUFFER REGISTER（TCNTB）,而Comparer会被Timer 加载到和COMPARE BUFFER REGISTER（TCMPB），每个Timer都有这样两个寄存器。当我们设置完毕启动Timer之后，Timer在一个工作周期内所做的就是先把 TCNTB中的数值（Counter）减1，之后把TCNTB中的数值和TCMPB中的数值(Comparer)进行对比，若Counter已经被递减到 等于Comparer,发生计数超出，则Timer产生中断信号（或是执行DMA操作）并自动把Counter重新装入TCNTB（刷新TCNTB以重新 进行递减）。以上就是Timer的工作原理。

下面我们结合代码具体说明如何对Timer0进行初始化并开启它。  
首先我假设我的PCLK是50700000Hz

// define Timer register  
#define rTCFG0 (\*(volatile unsigned int \*)0x51000000)  
#define rTCFG1 (\*(volatile unsigned int \*)0x51000004)  
#define rTCON (\*(volatile unsigned int \*)0x51000008)  
#define rTCNTB0 (\*(volatile unsigned int \*)0x5100000C)  
#define rTCMPB0 (\*(volatile unsigned int \*)0x51000010)  
#define rTCNTO0 (\*(volatile unsigned int \*)0x51000014)  
#define rTCNTB1 (\*(volatile unsigned int \*)0x51000018)  
#define rTCMPB1 (\*(volatile unsigned int \*)0x5100001C)  
#define rTCNTO1 (\*(volatile unsigned int \*)0x51000020)  
#define rTCNTB2 (\*(volatile unsigned int \*)0x51000024)  
#define rTCMPB2 (\*(volatile unsigned int \*)0x51000028)  
#define rTCNTO2 (\*(volatile unsigned int \*)0x5100002C)  
#define rTCNTB3 (\*(volatile unsigned int \*)0x51000030)  
#define rTCMPB3 (\*(volatile unsigned int \*)0x51000034)  
#define rTCNTO3 (\*(volatile unsigned int \*)0x51000038)  
#define rTCNTB4 (\*(volatile unsigned int \*)0x5100003C)  
#define rTCNTO4 (\*(volatile unsigned int \*)0x51000040)

void timer0\_config()  
{  
/\*  
                Timer0的prescaler由rTCFG0 的 0~7 bit决定  
                Prescaler=119  
\*/  
                rTCFG0=119          
/\*  
                Timer0的divider value由TCFG1的 0~3 bit决定,设置为3表示divider value = 1/16  
                rTCFG1的第20~23bit用于决定Timer计数超出后所采取的响应，我们使用了中断模式（20~23bit全部为0），  
                即计数超出后产生中断  
\*/  
                rTCFG1=3;  
          
                rTCNTB0=26406;  
                rTCMPB0=0;  
}  
由于我们的PCLK是50700000Hz, 根据Timer input clock Frequency的计算公式我们如下计算Timer0的时钟输入频率：

prescaler value = 119  
divider value = 1/16  
PCLK= 50700000  
Timer input clock Frequency =50700000/ (119+1)/(1/16)=26406

也就是说通过设置prescaler和divider value之后，Timer0的工作频率为26406，也就是说一秒内Timer0会进行26406次递减和比较操作，假设我们现在是要让Timer0每 1秒产生一次中断的话，我们应该设置Counter=26406和Camparer=0,既：  
rTCNTB0=26406;  
rTCMPB0=0;

如果我们要让Timer0每0.5秒产生一次中断，则我们应该设置Counter=26406/2和Camparer=0,既：  
rTCNTB0=13203;  
rTCMPB0=0;

如果我们要让Timer0每0.25秒产生一次中断，则我们应该设置Counter=26406/4和Camparer=0,既：  
rTCNTB0=6601;  
rTCMPB0=0;

初始化完Timer后我们要开启它。

void timer0\_start()  
{  
/\*  
               Update TCNTB0 & TCMPB0  
               rTCON寄存器的第1位是刷新Timer0的COUNT BUFFER REGISTER（TCNTB）和  
                COMPARE BUFFER REGISTER（TCMPB），由于是第一次加载Counter和Comparer，  
                所以我们需要手动刷新它们  
\*/  
               rTCON|=1<<1;  
/\*  
               置rTCON第0位为1，开启Timer0  
               把rTCON第1位置为0，停止刷新TCNTB0 和 TCMPB0  
               置rTCON第3位为1，设置Counter的加载模式为自动加载（auto reload），这样每当  
               Timer计数超出之后（此时TCNTB的值等于TCMPB的值），Timer会自动把原来我们给  
               定的Counter重新加载到TCNTB中  
\*/  
        rTCON=0x09;          
}

附件：







