

实用操作系统课程实验报告

|  |  |
| --- | --- |
| 实验名称： | 实验六  鸿蒙LiteOS-a内核移植——系统时钟移植 |
| 实验日期： | 2023-11-24 |
| 实验地点： | 文宣楼B313 |
|  | |
| 学号： | 33920212204567 |
| 姓名： | 任宇 |
| 专业年级： | 软工2021级 |
| 学年学期： | 2023-2024学年第一学期 |

1. 实验目的

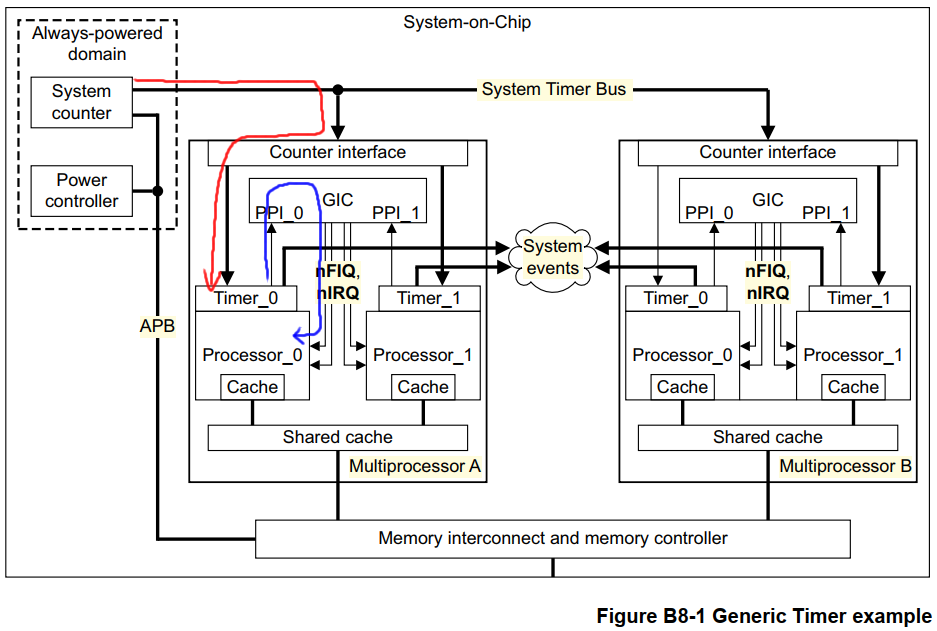
* 进行鸿蒙LiteOS-a内核的系统时钟移植

1. 实验内容和步骤

操作系统需要一个系统时钟，但是各类芯片都有自己的定时器并且对应的编程方法各不相同，这给系统移植带来困难。而Generic Timer从硬件层面上解决了这个问题。Generic Timer 是 ARM 架构中的一种硬件定时器。Generic Timer 提供了一种统一的方法来实现定时和延时功能，其主要特点包括：

* 硬件实现：它是处理器的一部分，直接在硬件层面提供计时功能，确保了计时的准确性和效率。
* 可编程性：Generic Timer 提供了一组可编程的寄存器，使得操作系统能够根据需要配置计时器的行为，例如设置定时周期、调整定时器的模式等。

Generic Timer分为两部分：共享的System Counter和各个Processor专有的Timer。System Counter给所有的Processor提供统一的时间，而Timer则是可以生成中断，这是实现任务调度和时间管理的关键。 下图是Generic Timer的硬件框图，红线表示时钟，而蓝线表示中断。



在使用时，一共有三个步骤：

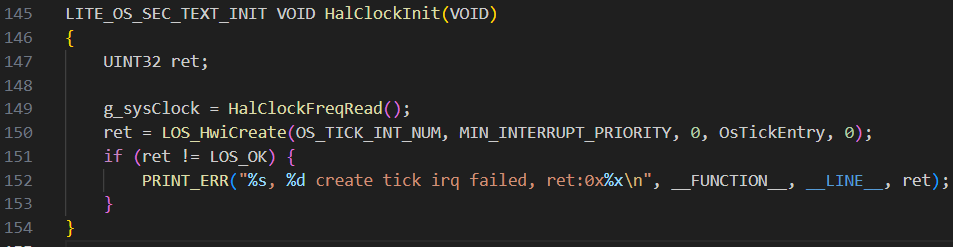
1. 设置时钟源；
2. 设置/启动System Counter；
3. 设置每一个Processor的Timer，设置比较值、使能中断、使能Timer等。

对于步骤一和二，在uboot中已经设置；对于步骤三，LiteOS\_a中也有完善的代码。因此，本次实验以阅读、分析源码为主，并不涉及修改源码，以下是对源码的阅读及分析：

文件路径为： openharmony\kernel\liteos\_a\platform\hw\arm\timer\arm\_generic

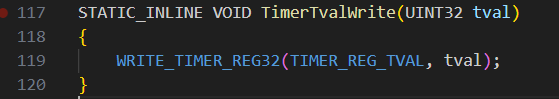
1. 初始化：

在HalClockInit函数中，首先通过HalClockFreqRead函数获得System Counter的频率并用以设置中断周期，接着调用LOS\_HwiCreate函数注册中断，具体代码如图所示：

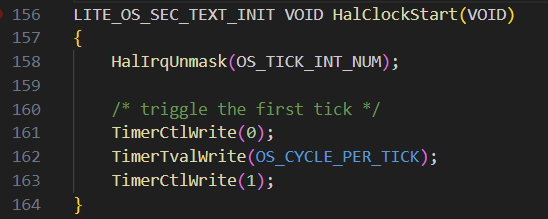


1. 启动Timer：

对于HalClockStart函数，它完成了两步操作——使能中断以及设置Timer Value寄存器。对于使能中断，是将中断号传入HalIrqUnmask函数当中；对于设置TimeValue寄存器，则是调用TimerTvalWrite函数进行设置，其实上就是将OS\_CYCLE\_PER\_TICK的值赋给对应的寄存器。

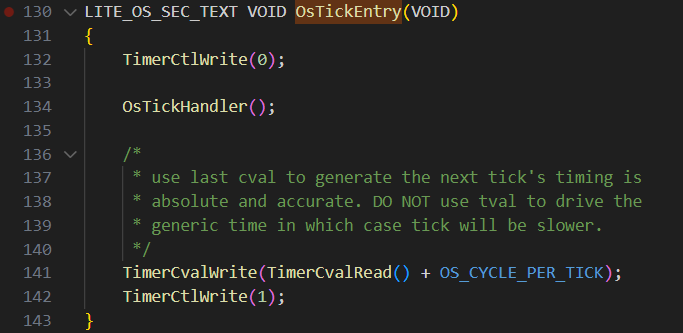


在宏定义中，OS\_CYCLE\_PER\_TICK = g\_sysClock / 100，也就是10MS之后产生中断。 HalClockStart函数完整代码如下：



1. 中断处理：

在使能中断后，就可以执行中断处理，这就要调用之前注册的中断处理函数就会被调用，即OsTickEntry函数。在OsTickEntry函数中，首先会停止Timer，接着调用OsTickHandler函数来处理，处理后就会调用TimerCvalWrite函数来更新Cval（比较寄存器），设置它的值为当前比较寄存器值 + OS\_CYCLE\_PER\_TICK。更新后启动Timer。



1. 实验总结

本次实验主要关注于鸿蒙LiteOS-a内核的系统时钟在ARM架构中的移植工作，特别是对Generic Timer的应用。Generic Timer提供了统一的方法来实现定时和延时功能。这次实验的重点在于理解Generic Timer的工作原理及其在操作系统中的应用。

在实验过程中，我通过分析arm\_generic\_timer.c文件来理解时钟的初始化（HalClockInit函数）、启动（HalClockStart函数）以及中断处理（OsTickEntry函数）的过程。通过这个实验，我更深入地理解了在不同硬件平台上实现操作系统时钟功能的复杂性和必要性，同时也体会到Generic Timer在简化这一过程中发挥的重要作用。

1. 遇到的困难及解决方法

无