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

2.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。

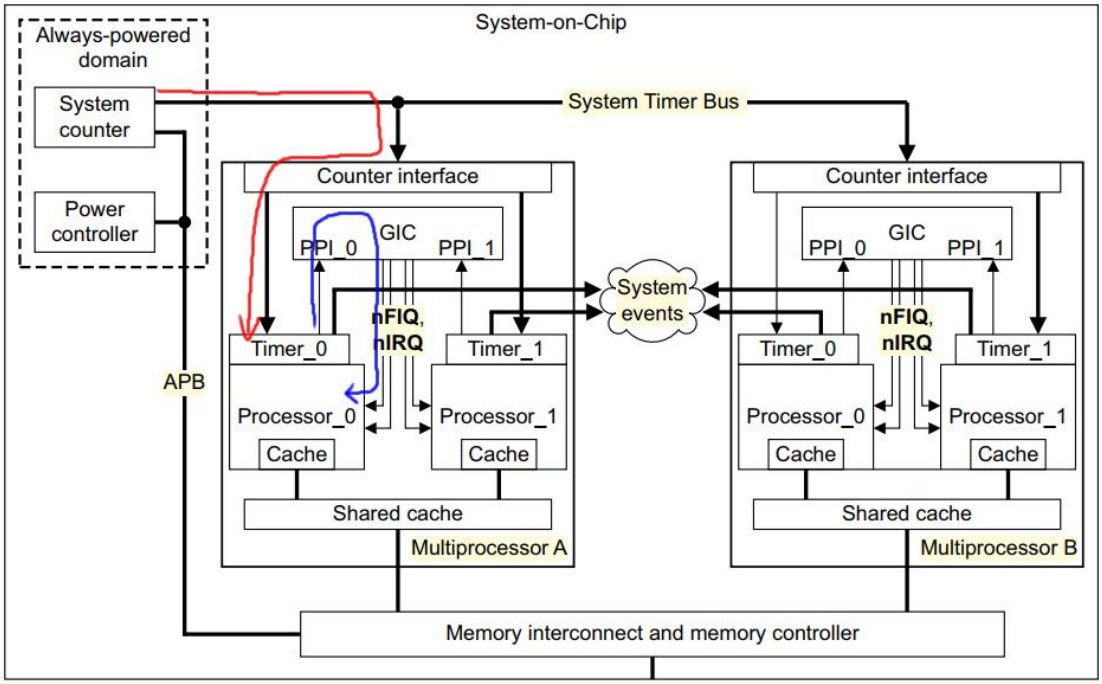

Figure B8-1 Generic Timer example

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

(一) 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：

```
145  LITE_OS_SEC_TEXT_INIT VOID HalClockInit(VOID)
146  {
147      UINT32 ret;
148
149      g_sysClock = HalClockFreqRead();
150      ret = LOS_HwiCreate(OS_TICK_INT_NUM, MIN_INTERRUPT_PRIORITY, 0, OsTickEntry, 0);
151      if (ret != LOS_OK) {
152          PRINT_ERR("%s, %d create tick irq failed, ret:0x%x\n", __FUNCTION__, __LINE__, ret);
153      }
154  }
```

(二)　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。

```
117  STATIC_INLINE VOID TimerTvalWrite(UINT32 tval)
118  {
119      WRITE_TIMER_REG32(TIMER_REG_TVAL, tval);
120  }
```

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：

```
156  LITE_OS_SEC_TEXT_INIT VOID HalClockStart(VOID)
157  {
158      HalIrqUnmask(OS_TICK_INT_NUM);
159
160      /* triggle the first tick */
161      TimerCtlWrite(0);
162      TimerTvalWrite(OS_CYCLE_PER_TICK);
163      TimerCtlWrite(1);
164  }
```

(三)　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。

```
130 LITE_OS_SEC_TEXT VOID OsTickEntry(VOID)
131 {
132     TimerCtlWrite(0);
133
134     OsTickHandler();
135
136     /*
137      * use last cval to generate the next tick's timing is
138      * absolute and accurate. DO NOT use tval to drive the
139      * generic time in which case tick will be slower.
140      */
141     TimerCvalWrite(TimerCvalRead() + OS_CYCLE_PER_TICK);
142     TimerCtlWrite(1);
143 }
```

3.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。

4.遇到的困难及解决方法

无