实验6 鸿蒙 LiteOS-a 内核移植——系统时钟移植

《实用操作系统》实验报告　22920212204392 黄勖

1　实验环境

Windows10 21H2、Vmware Workstation Pro 16、Ubuntu18.04

配置了相关的软件。

2　实验目的

鸿蒙 liteos 分析和实现系统时钟的移植

本次实验主要以理论阐述以及代码阅读，无需要自行编写的代码。

3　实验步骤与内容

3.1　通用定时器的理论分析

在操作系统中，需要一个系统时钟，各类芯片都有自己的定时器，它们的编程方法互不相同，这给系统移植带来麻烦。
Generic Timer是ARM推荐的一种硬件实现实现，可以实现统一的编程方法。
Generic Timer分为两部分：共享的System Counter、各个Processor专有的Timer。

- **System Counter**：给所有Processor提供统一的时间。该部分与外界的时钟源相连，时钟源在硬件上无法统一，当时钟源每次产生一个脉冲 System counter 就自增 1，之后传给后面的 Timer
- **Timer**：可以设置周期性的事件，给Processor提供中断信号，每个处理器都有各自的不同Timer；每个 Timer 可以 各自设置比较值，当 System Counter 的值达到这个比较值就产生一次时钟中断给 GIC（中断控制器），这也就决定了每个定时器产生中断的时间可以不一样

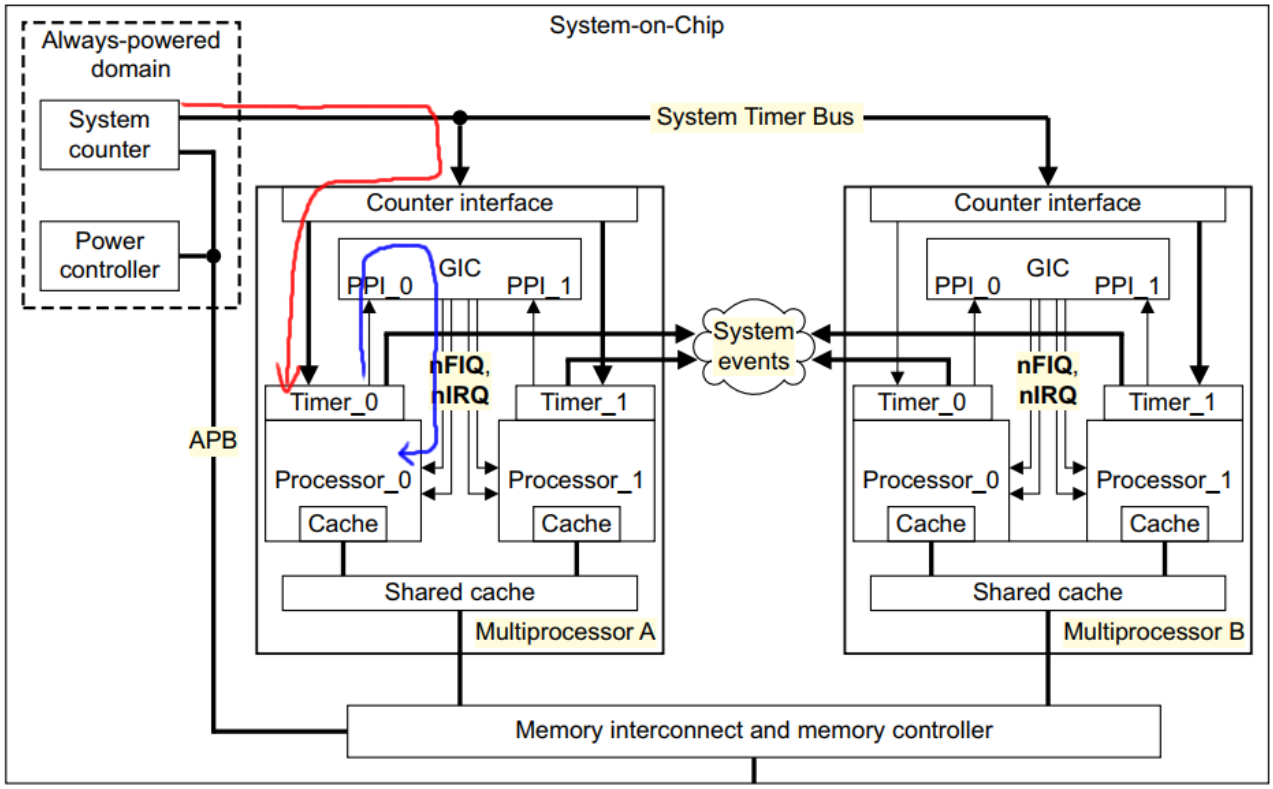


Figure B8-1 Generic Timer example

3.1.1 SystemCounter两种访问方式

SystemCounter是给所有Processor使用的，它有两种访问方式：

- CP15协处理器命令：某个Processor去访问它时可以使用CP15协处理器命令。
- MemoryMapped寄存器：
  - 既然它是给所有Processor使用的，那么应该提供更高级的访问方法(System Level)
  - 而且有些Processor并没有实现CP15，所有也应该提供MemoryMapped的方法

3.1.2 CP15寄存器

下面这个表格列出了所有的寄存器，包括SystemCounter和Timer，不仅仅是SystemCounter。

Table B8-2 Generic Timer registers

Name, VMSA[a]	Name, PMSA[a]	CRn	opc1	CRm	opc2	Width	Type	Description
CNTFRQ	CNTFRQ	c14	0	c0	0	32-bit	RW	Counter Frequency register
CNTPCT	CNTPCT	-	0	c14	-	64-bit	RO	Physical Count register
CNTKCTL	CNTKCTL	c14	0	c1	0	32-bit	RW	Timer PL1 Control register
CNTP_TVAL	CNTP_TVAL			c2	0	32-bit	RW	PL1 Physical TimerValue register
CNTP_CTL	CNTP_CTL				1	32-bit	RW	PL1 Physical Timer Control register
CNTV_TVAL	CNTV_TVAL			c3	0	32-bit	RW	Virtual TimerValue register
CNTV_CTL	CNTV_CTL				1	32-bit	RW	Virtual Timer Control register
CNTVCT	CNTVCT	-	1	c14	-	64-bit	RO	Virtual Count register
CNTP_CVAL	CNTP_CVAL		2			64-bit	RW	PL1 Physical Timer CompareValue register
CNTV_CVAL	CNTV_CVAL		3			64-bit	RW	Virtual Timer CompareValue register
CNTVOFF[b]	-[b]		4			64-bit	RW[b]	Virtual Offset register
CNTHCTL[c]	-[c]	c14	4	c1	0	32-bit	RW	Timer PL2 Control register
CNTHP_TVAL[c]	-[c]			c2	0	32-bit	RW	PL2 Physical TimerValue register
CNTHP_CTL[c]	-[c]				1	32-bit	RW	PL2 Physical Timer Control register
CNTHP_CVAL[c]	-[c]	-	6	c14	-	64-bit	RW	PL2 Physical Timer CompareValue register

3.1.3 MemoryMapped寄存器

Table D5-1 CNTControlBase memory map

Offset	Name	Type	Description
0x000	CNTCR	RW	Counter Control Register.
0x004	CNTSR	RO	Counter Status Register.
0x008	CNTCV[31:0]	RW	Counter Count Value register.
0x00C	CNTCV[63:32]	RW	
0x010-0x01C	-	UNK/SBZP	Reserved.
0x020	CNTFID0	RO or RW	Frequency modes table. CNTFID0 is the base frequency, and each CNTFID*n* is an alternative frequency. For more information see *The frequency modes table* on page D5-2400.
0x020+4*n*[a]	CNTFID*n*	RO or RW	
0x024+4*n*[a]	-	RO or RW	RAZ. Marker for end of Frequency modes table.
(0x028+4*n*)-0x0BC[a]	-	UNK/SBZP	Reserved.
0x0C0-0x0FC	-	IMPLEMENTATION DEFINED	Reserved for IMPLEMENTATION DEFINED registers.
0x100-0xFCC	-	UNK/SBZP	Reserved.
0xFD0-0xFFC	CounterIDn	RO	Counter ID registers 0-11.

a. *n* is in the range 0-23.

在u-boot代码中可以看到这样的结构体：

```
/* System Counter */
struct sctr_regs {
    u32 cntcr;   // control register, 启动/停止
    u32 cntsr;   // status register, 是否启动/停止, 使用哪个频率
    u32 cntcv1;  // count value lower register
    u32 cntcv2;  // count value upper register, cntcv1和cntcv2组成64位的计数值
    u32 resv1[4];
    u32 cntfid0; // base frequency register, 必须等于SystemCounter的输入频率
    u32 cntfid1; // cntfid1和cntfid2: 其他频率
    u32 cntfid2;
    u32 resv2[1001];
    u32 counterid[1];
};
```

3.1.4 Timer特性

每个Processor都有一个Timer，它有3个寄存器，只能使用协处理器命令方位(CP15)：

- 64位的比较寄存器(CVAL)：当SystemCounter的值等于它时，产生事件(中断)，被称为upcounter
  - SystemCounter总是增长的，所以Timer的64位比较寄存器也只能设置为大于SystemCounter的值
- 32位的TimerValue寄存器(TVAL)
  - 它是downconter
  - 比如设置为1000，表示再经过1000个时钟之后，就会产生事件(中断)
  - 实质是：设置64位的比较寄存器，让它等于SystemCounter+1000
- 32位的控制寄存器(CTL)
  - 使能/禁止Timer
  - 使能输出：是否能产生事件(中断)
  - 状态：是否能产生了事件(中断)

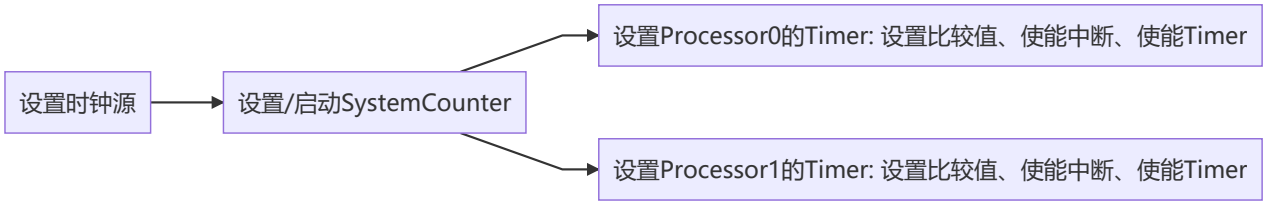
Table B8-1 Timer registers summary for the Generic Timer

	PL1 physical timer[a]	PL2 physical timer[b]	Virtual timer
CompareValue register	CNTP_CVAL	CNTHP_CVAL	CNTV_CVAL
TimerValue register	CNTP_TVAL	CNTHP_TVAL	CNTV_TVAL
Control register	CNTP_CTL	CNTHP_CTL	CNTV_CTL

a. Registers are Banked in a processor implementation that includes the Security Extensions.
b. Implemented only in a processor implementation that includes the Virtualization Extensions.

3.2 针对通用定时器的代码阅读

使用通用定时器的三个步骤：



- 设置时钟源：芯片相关，一般u-boot里做好了
- 设置/启动SystemCounter：一般u-boot里做好了
- 设置Processor的Timer：
  - 设置比较值、使能中断、使能Timer
  - 注册中断处理函数

3.2.1 源码分析

源码位于 `drivers\timer\arm_generic_timer.c` 文件中

3.2.2 HalClockInit 初始化

它做了2件事：

- 读出SystemCounter的频率：以后设置中断周期时要用
- 注册中断处理函数

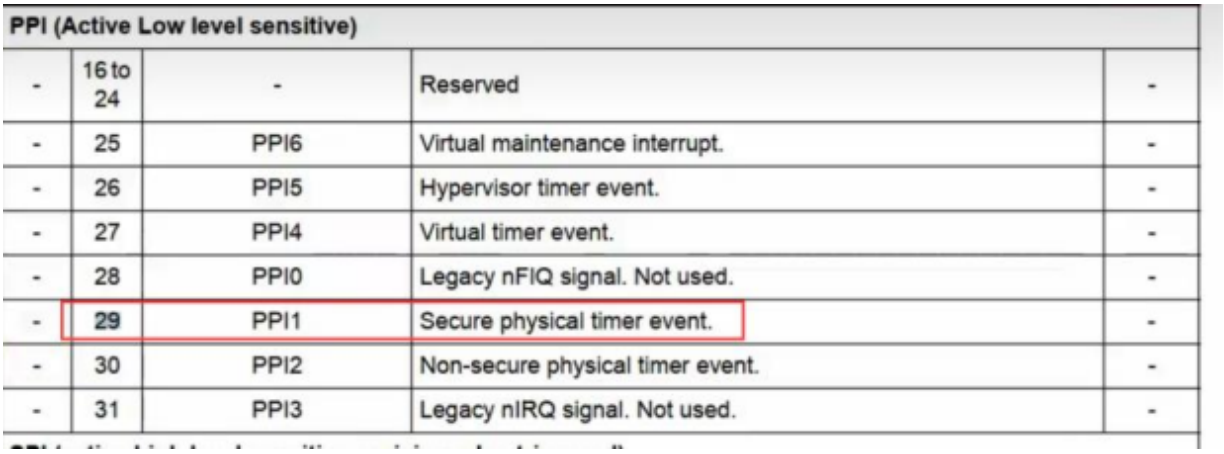
```
LITE_OS_SEC_TEXT_INIT VOID HalClockInit(VOID)
{
    UINT32 ret;

    g_sysClock = HalClockFreqRead();
    ret = LOS_HwiCreate(OS_TICK_INT_NUM, MIN_INTERRUPT_PRIORITY, 0,
OsTickEntry, 0);
    if (ret != LOS_OK) {
        PRINT_ERR("%s, %d create tick irq failed, ret:0x%x\n", __FUNCTION__,
__LINE__, ret);
    }
}
```

3.2.3 HalClockStart 启动Timer

它做了2件事：

- 使能中断：中断号是29



PPI (Active Low level sensitive)				
-	16 to 24	-	Reserved	-
-	25	PPI6	Virtual maintenance interrupt.	-
-	26	PPI5	Hypervisor timer event.	-
-	27	PPI4	Virtual timer event.	-
-	28	PPI0	Legacy nFIQ signal. Not used.	-
-	29	PPI1	Secure physical timer event.	-
-	30	PPI2	Non-secure physical timer event.	-
-	31	PPI3	Legacy nIRQ signal. Not used.	-

- 设置TimerValue寄存器：
  - OS_CYCLE_PER_TICK = g_sysClock / 100，也就是10MS之后产生中断
  - 设置TimerValue寄存器的实质，就是设置**比较寄存器(CVAL) = 当前SystemCounter值 + OS_CYCLE_PER_TICK**

```
LITE_OS_SEC_TEXT_INIT VOID HalClockStart(VOID)
{
    HalIrqUnmask(OS_TICK_INT_NUM);

    /* triggle the first tick */
    TimerCtlWrite(0);
    TimerTvalWrite(OS_CYCLE_PER_TICK);
    TimerCtlWrite(1);
}
```

3.2.4 HalTickEntry 中断处理

它做了2件事：

- 调用OsTickHandler
- 设置下一次中断时间：
  - 重新设置 Cavl 寄存器为当前值加上 1000，也就是在下一个 10ms 让 Timer 产生一次中断

```
LITE_OS_SEC_TEXT VOID OsTickEntry(VOID)
{
    TimerCtlWrite(0);

    OsTickHandler();
```

```
    /*
     * use last cval to generate the next tick's timing is
     * absolute and accurate. DO NOT use tval to drive the
     * generic time in which case tick will be slower.
     */
    TimerCvalWrite(TimerCvalRead() + OS_CYCLE_PER_TICK);
    TimerCtlWrite(1);
}
```

4 问题和解决方法

本次实验过程为阅读代码，未出现问题。

5 实验体会

通过这次实验，我了解了时钟系统的概念，实现方式，以及了解了通用计时系统 Generic Timer 的组织形式和原理机制，掌握了鸿蒙系统如何采用时钟系统。