Timer 类相关设计思想

## 一：利用MRT 的 one – shot mode 来实现定时器：

首先每个类中都有一个 timer\_ 变量，该变量用来访问底层寄存器，和硬件打交道。构造函数私有化来实现单例模式，并把MRT通道设置为one-shot mode，关闭中断，该构造函数根据传入的参数来构造使用不同通道的计时器，同时利用工厂instance返回该类的静态对象，根据传入参数的值来决定返回使用哪个通道的计数器的指针。

利用 Start\_Millisecond() 函数来实现开始计数，该函数传入以 ms 为单位的数值，在函数中清除中断标志，然后设置 IVAL 值，计数器开始计数。

利用 IsExpired（）函数来判断计数是否停止。

**1：One-shot mode 下：**

**MRT 定时器的开启和停止**：

**1）向计数器写非零值：**

如果 MRT空闲，向INTVAL 写入一个非零值，则立即开启计数，计数到零则置位中断位，如果使能中断，则还向CPU申请中断。

如果MRT 正在运行，写入一个非零值，如果 LOAD = 1，则立即开始以新值计

数，并且更新新值时不产生中断。如果Load = 0，不产生影响（不重复计数）。

**2）向计数器写零值**

**Load = 1，则立即停止**, **无中断**

如果 **Load = 0**，不对计数过程产生影响，**计数到零有中断**。

**2：Repeat Mode 下：**

**1）向计数器写非零值：**

计数器空闲时：开启计数器就是向计数器写一个非零值，计数器开启计时，计数到零产生中断，开始下一轮计数。

计数器不空闲：

向计数器写一个非零值，如果 **Load = 1**，立即以新计数值计数，**不产生中断**

如果 **Load = 0**，则下一轮开始以新计数值计数，**到零产生中断**。

**2）向计数器写零值：**

如果 **Load = 1**,则立即停止计数，并且**不产生中断**

如果 **Load = 0**,则下一轮停止计数，然后本轮计数到零后，**产生中断**。

## 二：利用 RIT 来实现长定时器

利用RIT的重复计数模式来实现one-shot mode模式：就是在RIT的中断中停止计数，并把中断标志清除,并把及时结束标识符置位。该中断函数的实现为：

**extern** "C" **void** **RIT\_IRQHandler**(**void**)

{

/\* Clear the RITEN bit, and set the RITENBR bit, set the RITENCLR bit for clearing

\* the timer to 0 when counter reach the condition, set 1 for RITINT to clear the

\* interrupt flag

\*/

Chip\_RIT\_Disable(LPC\_RITIMER);

Chip\_RIT\_ClearIntStatus(LPC\_RITIMER);

**if**(rit\_callback != 0)

{

(\*rit\_callback)();

}

}

在中断中把计数器停止，清除中断，然后判断回调函数是否为空，不为空则调用。

在 LongTimer 的工厂中，每次调用工厂时都会传入一个回调函数。这个回调函数将在中断中被调用。

## 二：利用 MRT 来实现周期定时器

利用该定时器的重复模式，但是有一点需要注意，该定时器只能够做基本的定时任务，并且重复定时，目前定时以 10ms 为单位，并且回调函数被设为了对LED的点亮计数，计数结束，就会关闭LED。目前只有一个任务，就是对LED进行点亮计数，后面打算实现任务轮询，把回调设为一个数组，每一个周期都要处理数据。