第2章 基于时间触发的合作式调度器

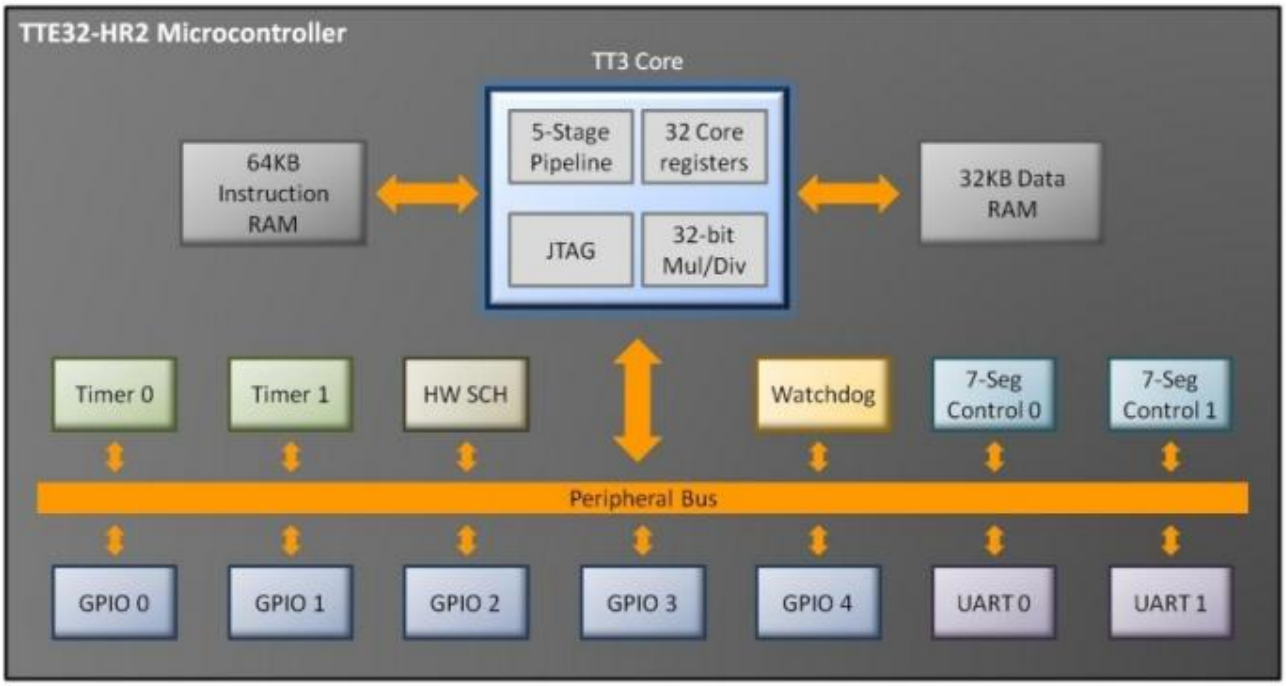
2.1 调度器介绍

简单的说，调度器就是使用相关的调度算法来决定当前需要执行的任务。所有的调度器有一个共同的特性：调度器可以区分就绪态任务和挂起任务（由于延迟，信号量等待，邮箱等待，事件组等待等原因而使得任务被挂起）。调度器可以选择就绪态中的一个任务，然后激活它（通过执行这个任务）。当前正在执行的任务是运行态的任务。不同调度器之间最大的区别就是如何分配就绪态任务间的完成时间。

嵌入式实时操作系统的核心就是调度器和任务切换，调度器的核心就是调度算法。任务切换的实现在各个 RTOS 中区别不大，基本相同的架构，任务切换也是相似的。调度算法就有些区别了，比如咱要讲的 μCOS-III 和 FreeRTOS 在抢占式调度器算法上就是两种不同的调度方法，下面我们主要了解一下合作式调度器，抢占式调度器和时间片调度器。

2.1.1 合作式调度器

合作式调度器实现起来比较简单，易学易用，但是要真正的运用好这种调度器，还是要用心研究下的，要不就不存在那么多嵌入式领域的工程师做这方面的研究，比如咱们上期教程里面说的 Michael J. Pont，他是这方面的资深专家，有自己的团结和公司，他们设计了一款处理器内嵌基于时间触发的合作式调度器，重要的是已经商用。这么说来应该算是这个领域成功的典范，下面的截图是他们设计这款处理器的框图。



上面的 HW SCH 就是硬件调度器的意思。

如何理解合作式调度器呢，如果用过 FreeRTOS 的话，FreeRTOS 就支持合作式调度（和本期教程要讲的调度方式稍有区别）。没有用过 FreeRTOS 的话，μCOS-II 里面的 Timer 组件也有类似的功能（μCOS-III 里面还保留着这个功能），其实这些东西都是相同的，而且在 Micrium 公司出的μCOS-II 或μCOS-III 配套官方书籍中说的不可剥夺性内核（Non-Preemptive Kernel）和咱们这里要说的合作式调度器也是类似的。简单的来说，合作式调度器就是根据用户的设置时刻（周期或者单次）来执行相应的任务，每个时刻只有一个任务可以执行，这些任务间不支持被强占，直到该任务自愿放弃 CPU 的控制权。下面要说的合作式调度器特性就是摘自时间触发嵌入式模式那本书。

合作式调度器

- 合作式调度器提供了一种单任务的的系统结构

操作：

- 在特定的时刻被调度运行（以周期性或者单次方式）
- 当任务需要运行的时候，被添加到等待队列。
- 当 CPU 空闲的时候，运行等待队列中的下一个（如果有的话）。
- 任务运行直到完成，然后由调度器来控制。

实现：

- 这种调度器很简单，用少量代码即可实现。
- 该调度器必须一次只为一个任务分配存储器。
- 该调度器通常完全由高级语言（比如“C”）实现。
- 该调度器不是一种独立的系统，它是开发人员代码的一部分。

新能：

- 设计阶段需要小心以快速响应外部事件。

可靠性和安全性：

- 合作式调度简单，可预测，可靠并且安全。

2.1.2 抢占式调度器

在实际的应用中，不同的任务需要不同的响应时间。例如，我们在一个应用中需要使用电机，键盘和 LCD 显示。电机比键盘和 LCD 需要更快速的响应，如果我们使用前面说的合作式调度器或者后面要说的时间片调度，那么电机将无法得到及时的响应，这时抢占式调度是必须的。

如果使用了抢占式调度，最高优先级的任务一旦就绪，总能得到 CPU 的控制权。当一个运行着的任务使一个比它优先级高的任务进入了就绪态，当前任务的 CPU 使用权就被剥夺了，或者说被挂起了，那个高优先级的任务立刻得到了 CPU 的控制权。如果是中断服务子程序使一个高优先级的任务进入就绪态，中断完成时，中断了的任务被挂起，优先级高的那个任务开始运行。

使用抢占式调度器，使得最高优先级的任务什么时候可以执行，可以得到 CPU 的控制权是可知的，同时使得任务级响应时间得以最优化。

总的来说，学习抢占式调度掌握最关键的一点是：抢占式调度器会为每个任务都分配一个优先级，调度器会激活就绪任务中优先级最高的任务。

上期教程提到的 embOS，FreeRTOS，μCOS-II，μCOS-III，RTX 都支持抢占式调度，可以说这种调度算法在小型嵌入式 RTOS 中极为流行。抢占式调度器给任务带来快速响应的同时也使得任务间的同步和通信机制显的很麻烦，而且源码中的很多地方都需要设置临界段（通过开关中断来实现）。

2.1.3 时间片调度器

在小型的嵌入式 RTOS 中，最常用的的时间片调度算法就是 Round-robin 调度算法。这种调度算法可以用于抢占式或者合作式的多任务中，时间片调度适合用于不要求任务实时响应的情况下。

实现 Round-robin 调度算法需要给同优先级的任务分配一个专门的列表，用于记录当前就绪的任务，并为每个任务分配一个时间片（也就是需要运行的时间长度，时间片用完了就进行任务切换）。

目前 embOS，FreeRTOS，μCOS-III 和 RTX 都支持 Round-robin 调度算法。