**电子科技大学**



**嵌入式系统实验**

**实验三**

**学号：202222080625**

**姓名：乔翱**

**学院：计算机科学与工程学院**

1. **实验名称**

时钟管理和资源管理的学习

1. **实验时间**

2023年5月23日

**三、实验内容和目的**

1、学习时钟管理，如何设置时钟任务，包括通过时钟完成定期的任务，通过时钟设置事件或激活某个任务。

2、学习资源管理，多个任务共享一个资源时，如何实现才能避免死锁发生。

3、完成实验三的要求。

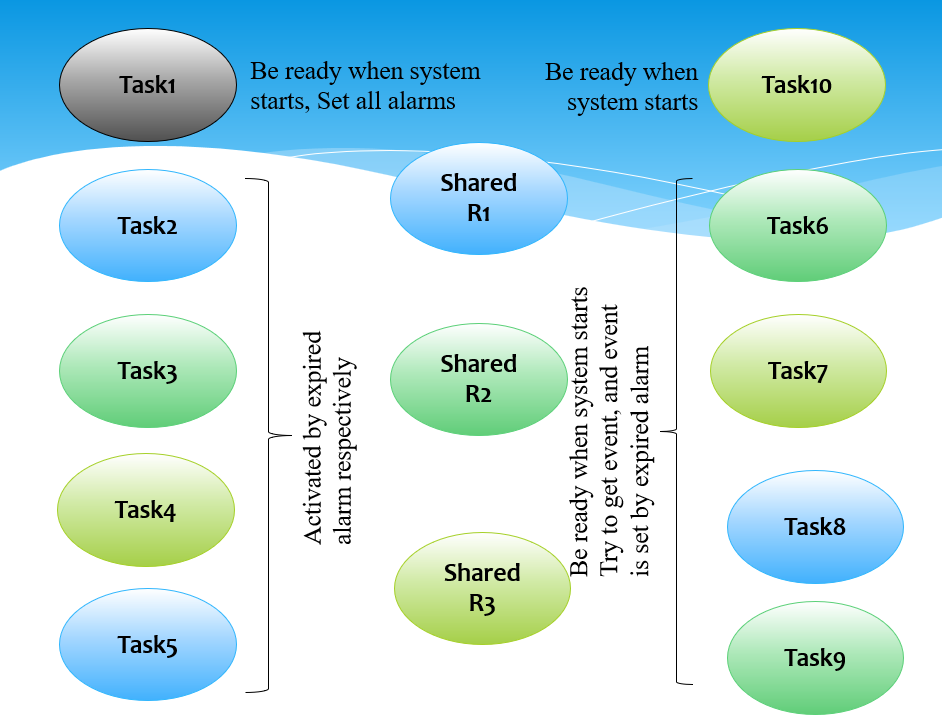
**四、实验设计**

实验总共设计了10个任务，任务一作为初始化任务，并且在任务1的实现函数中，设置并初始化所有的时钟。

实验总共设计了八个时钟。前四个时钟的作用是激活任务二到任务五，后四个时钟用来为任务六到九设置事件。

任务2,5,8共享资源1，任务3,6,9共享资源2，任务4,7,10共享资源3.这些任务想要执行后面的功能，必须首先获取相应的资源，并且每个任务在执行完自己的功能后需要释放资源。

各个任务的关系图如下所示。



**五、任务配置**

按照实验设计对总共10个任务进行配置，项目中任务的相关配置如下。对于任务的初始状态，任务2到5是由时钟激活的，所以这四个任务的初始状态不是ready，而任务六到九的初始状态是ready，这四个任务都在等待时钟设置的事件，之后才能继续进行。

|  |
| --- |
| /\* task configuration data \*/  const T\_OSEK\_TASK\_ConfigTable\_Struct osekConfig\_TaskTable[OCC\_NTSKS]=  {  /\*Task1 ID:0 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-0-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask1, //task function  Task1, //task ID  OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED, //task initial attributes  &taskStack[Task1][TASK\_STACK\_SIZE-1], //top of the task stack  &taskStack[Task1][0] //bottom of the task stack    },  /\*Task2 ID:1 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-1-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask2,  Task2,  OSEK\_TASK\_EXTENDED,  &taskStack[Task2][TASK\_STACK\_SIZE-1],  &taskStack[Task2][0]  },  /\*Task3 ID:2 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-2-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask3,  Task3,  OSEK\_TASK\_EXTENDED,  &taskStack[Task3][TASK\_STACK\_SIZE-1],  &taskStack[Task3][0]  },  /\*Task4 ID:3 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-3-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask4,  Task4,  OSEK\_TASK\_EXTENDED,  &taskStack[Task4][TASK\_STACK\_SIZE-1],  &taskStack[Task4][0]  },    /\*Task5 ID:4 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-4-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask5,  Task5,  OSEK\_TASK\_EXTENDED,  &taskStack[Task5][TASK\_STACK\_SIZE-1],  &taskStack[Task5][0]  },  /\*Task6 ID:5 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-5-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask6,  Task6,  OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED,  &taskStack[Task6][TASK\_STACK\_SIZE-1],  &taskStack[Task6][0]  },  /\*Task7 ID:6 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-6-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask7,  Task7,  OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED,  &taskStack[Task7][TASK\_STACK\_SIZE-1],  &taskStack[Task7][0]  },  /\*Task8 ID:7 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-7-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask8,  Task8,  OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED,  &taskStack[Task8][TASK\_STACK\_SIZE-1],  &taskStack[Task8][0]  },  /\*Task9 ID:8 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-8-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask9,  Task9,  OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED,  &taskStack[Task9][TASK\_STACK\_SIZE-1],  &taskStack[Task9][0]  },  /\*Task10 ID:9 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-9-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask10,  Task10,  OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED,  &taskStack[Task10][TASK\_STACK\_SIZE-1],  &taskStack[Task10][0]  },  /\*Task Idle\*/  {  (T\_OSEK\_TASK\_Entry)FuncIdle,  OSEK\_TASK\_IDLE\_ID,  OSEK\_TASK\_ACTIVE,  },  }; |

**六、实验步骤**

1、修改osprop.h中关于任务数、时钟数量和资源数量的配置。

2、在cfg.h和cfh.c配置相关任务，包括任务的定义声明以及任务相关属性的配置，重点配置任务的初始状态。

配置时钟以及资源，时钟的相关配置如下。

|  |
| --- |
| //ALARM configuration  T\_OSEK\_ALARM\_ConfigTable osekConfig\_AlarmTable[OCC\_NALMS] =  {  {  Task2, //the task notified by the Alarm    0x00, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task3, //the task notified by the Alarm    0x00, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task4, //the task notified by the Alarm    0x00, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task5, //the task notified by the Alarm    0x00, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task6, //the task notified by the Alarm    0x60, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task7, //the task notified by the Alarm    0x70, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task8, //the task notified by the Alarm    0x80, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  },    {  Task9, //the task notified by the Alarm    0x90, //the event to be set for the task  SysCounter, //name of the counter that the alarm uses  }  }; |

3、在init.c中编写每个任务函数，实现每个任务的相关功能。

在任务1中设置各个时钟，时钟设置函数总共有三个参数，第一个参数是要设置的时钟，第二个时钟是相对开始时间，第三个时钟是每隔多少tick时钟执行。

在每个任务获取资源的时候，需要注意资源获取的顺序，保证任务不会形成死锁。

|  |
| --- |
| SetRelAlarm(TaskAlarm1, 3000, 10000);  status = GetResource(Resource1);  printf("task2 get resource1...\n");  if(status != E\_OK)  {  printf("task2 get resource1 error: %d\n", status);  } |

**七、实验结果及分析**

在IDE中编译，并运行，实验结果如下：





分析实验结果，可以看到程序按照预先设计进行，并且时钟任务会按照设置的时间，每隔一段时间进行。时钟成功激活任务或为相应的任务设置事件。符合之前的设计。

对于共享同一资源的任务，按照之前的设计，并没有形成死锁，这些任务在执行后面的函数功能前会先获取资源，在执行后会释放资源。

**八、实验结论及心得体会**

通过本次实验，学习了如何使用时钟，来设置时钟任务，使用时钟来定期执行某些功能，包括事件的设置以及任务的激活。对于多个不同的任务需要同一资源的情况，学习了如何获取资源，并且通过合适的资源获取顺序，不会造成死锁的发生。