CCSv5.5 中 DSP/BIOS 的搭建

说明：鉴于目前网络上关于如何在 CCS5.5 中怎么运用 DSP/BIOS 的详细指导十分不全面，本人在搜索过程中也是一头雾水，故写此文档，第一便于大家互相学习交流，第二方便本单位后续技术积累。文中截图系本人亲自操作截取。

1：创建常规 CCS5.5 工程

1） 打开 CCS，选择 **File > New > CCS Project**。

2） 在 **Project name** 栏输入要创建的工程名字，以 test 为例。

3） 在 **Family** 栏选定你所使用的 DSP 的家族系列，以 C6000 为例。

4） 在 **Variant** 栏选定你所使用的 DSP 系列，以 C674x Floating-point DSP 为例，在后面的选框中选中具体的 DSP 型号。以 TMS320C6748 为例。

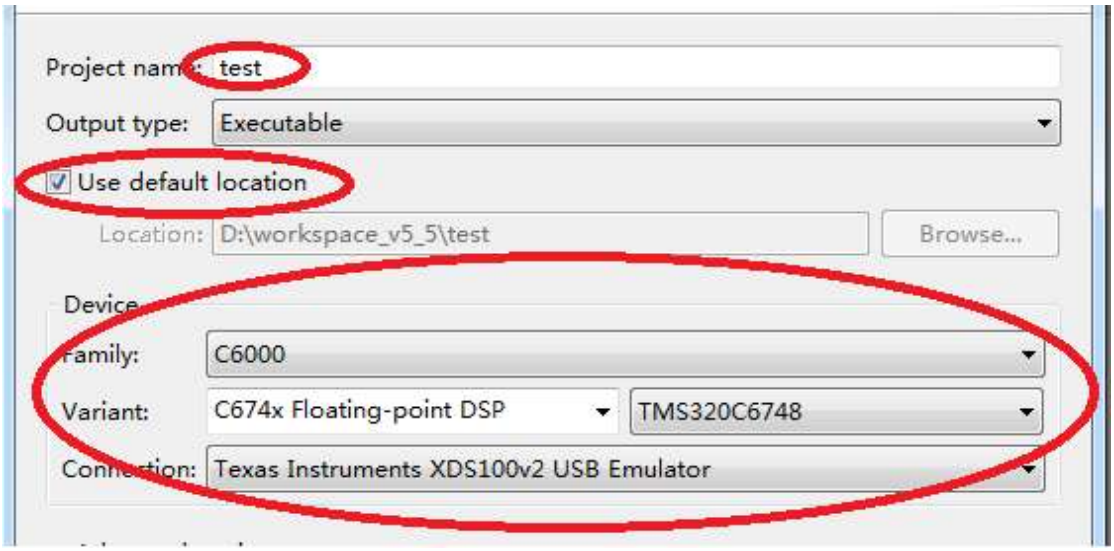
5） 在 **Connection** 栏选择你所使用的仿真器型号。

6） **Advanced settings** 高级选项，主要时选择芯片的大小端，编译器版本，一般情况下这里不需要设置。

7）在 **Project templates and examples** 栏选中带 main.c 的空白工程。

8）点击 **Finish** 按钮，完成。

至此一个普通的 CCS5.5 工程就创建完毕，剩下的就 BIOS 如何引用过来的问题了。



2：引入 DSP/BIOS 系统

注 1：因为刚才在创建工程的时候已经产生了一个名为 C6748.cmd 的链接命令文件，在这里需要删除这个链接命令文件，因为 DSP/BIOS 在创建的过程中会产生一份新的链接命令文件。并且新的链接命令文件会把一些用到的应用库包含进来，例如 bios.a62,rtdx.lib,rts64plus.lib 等程序库。大多数 DSP/BIOS 生成的链接命令文件会满足所有的存储段分配，也可以后续再通过 MEM 管理器进行控制。

注 2：假如你的工程之前有包含 vectors.asm 源文件，同样需要移除这个文件，因为 DSP/BIOS 会自动定义硬件中断向量表。就是说假如你使用了 DSP/BIOS 系统，中断向量的管理权也就交给了 DSP/BIOS。

好的，做好以上准备工作后，下面我们就开始一步一步的创建 DSP/BIOS 的应用，我们这里以一个最简单的应用例程进行说明，在这里会带领大家创建一个包含有两个任务的应用程序，第一个任务执行把 LED 点亮的工作，第二个任务执行把 LED 点灭的工作。

添加 DSP/BIOS 配置到当前工程

1） 选择 **File > New > DSP/BIOS v5.x Configuration File**。

2） 检查 **Filename** 栏的 tcf 文件名是否和你的工程名一致。这里名为 test.tcf。点击 **Next** 按钮。

3） 选择所属的器件型号平台，我的是 **ti.platforms.evm6748**，点击 **Next** 按钮。

4） 将默认选中三个 DSP/BIOS 特性选中，点击 **Finish** 按钮。

Real-Time Analysis 若禁止，则 LOG、STS 不可用。

RTDX 若禁止，则实时分析数据不可实现。

TSK Manager 允许你使用信号量和任务让出功能。

注：在这里会有一个叫做“指定 xdc 工具安装的”对话框弹出，我目前也不清楚在这里不指定会有什么影响，点击 ok 跳过貌似也没有什么影响，不知道是不是我的 CCS 安装引起的这个问题，有待研究，总之你先点击 ok 就行了，接着点击 yes 按钮。创建 tcf 文件完毕。

在这里你可以先编译一下你所创建的工程，如果你是按照我所描述的步骤进行创建的话，编译应该是没有错误可以通过编译的。

注：编译通过后你可以在左侧工程导航栏的 Debug 文件夹下看到一系列 DSP/BIOS 所创建的文件，如

testcfg_c.c 文件：　定义 DSP/BIOS 结构体和内容。

testcfg.cmd 文件　链接命令文件

testcfg.h 文件　包含 DSP/BIOS 模块头文件、声明对象的外部变量。

testcfg.s62 文件　DSP/BIOS 配置的汇编文件

testcfg.h62　汇编语言头文件

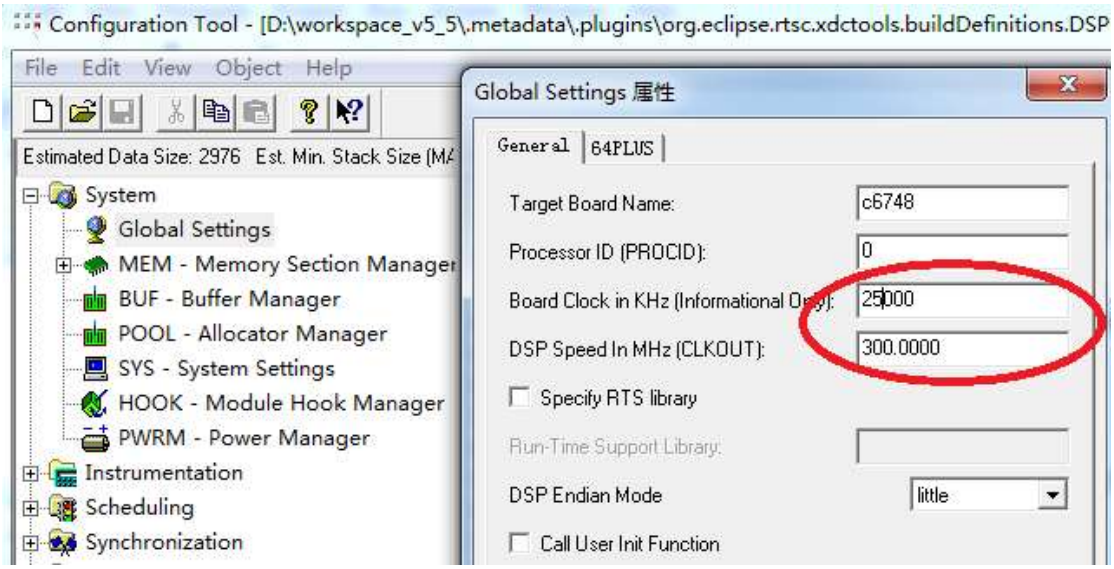
不好意思这一步忘记截图了，过程比较简单，照着做就行了。

3：添加任务和信号量

这一步就是创建 DSP/BIOS 各个管理模块的对象，首先对全局属性进行一下必要的设置，在左侧的工程导航栏双击 test.tcf 打开管理器。

3.1：全局属性设置

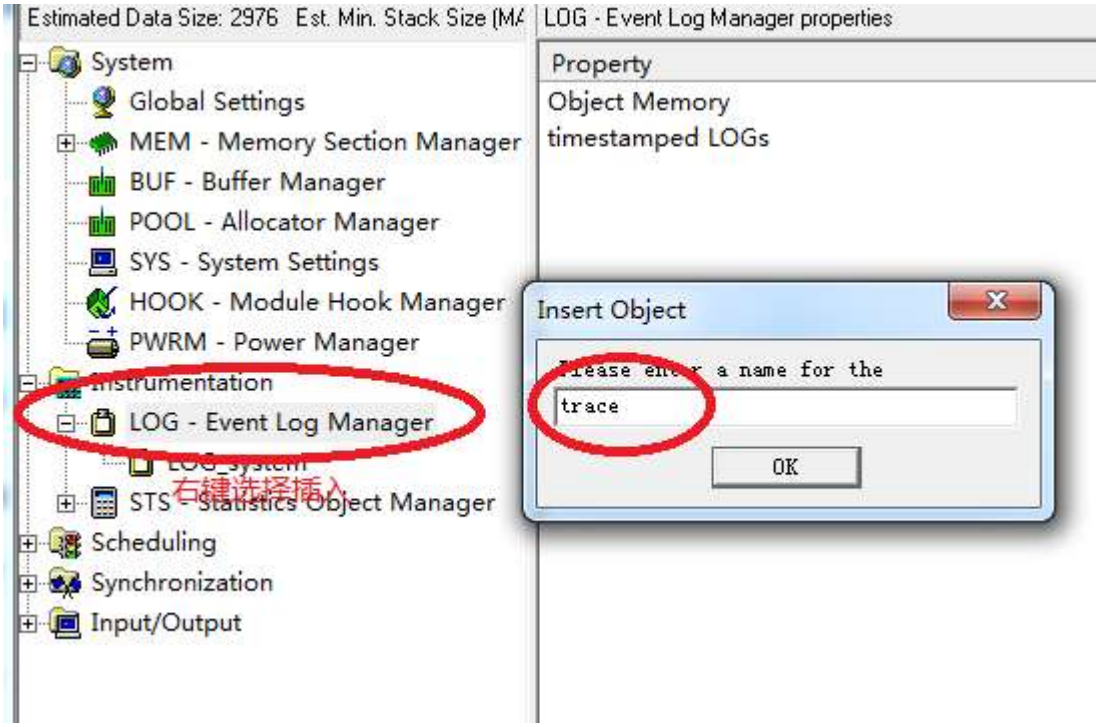
选中 **System** 栏下的 **Global Settings**，右键，选择属性按钮，进行如下设置，这里主要是设置 CPU 运行的时钟，因为我将来要把我的 DSP 运行在 300MHz 的频率，外部接的晶振是 25MHz，所以设置如下。



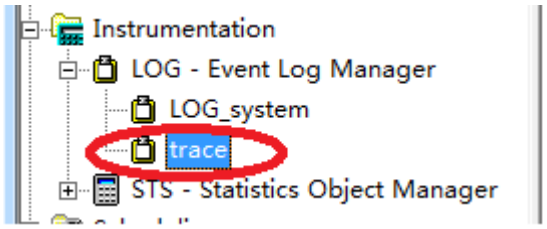
3.2：LOG 模块的设置

LOG 模块可以帮助我们调试将来的代码，可以利用模块本身的 LOG_printf 函数在 CCS 环境里面打印信息，对我们调试代码十分有用。而且占用的 CPU 资源很小，几乎不影响 CPU 的性能。下面说说具体的配置方法。

选择 **Instrumentation** 子目录下的 **LOG-Event Log Manager**，右键选择 **Insert LOG**，在打开的对话框中为你要创建的模块起个名字，一般以 trace 命名。如图：



生成 LOG 对象之后如图所示



这样 LOG 模块就设置好了，下面说一下使用举例，很简单

LOG_printf(&trace,"Task1 LED on");

就可以在 CCS 的相应窗口中看到打印信息。

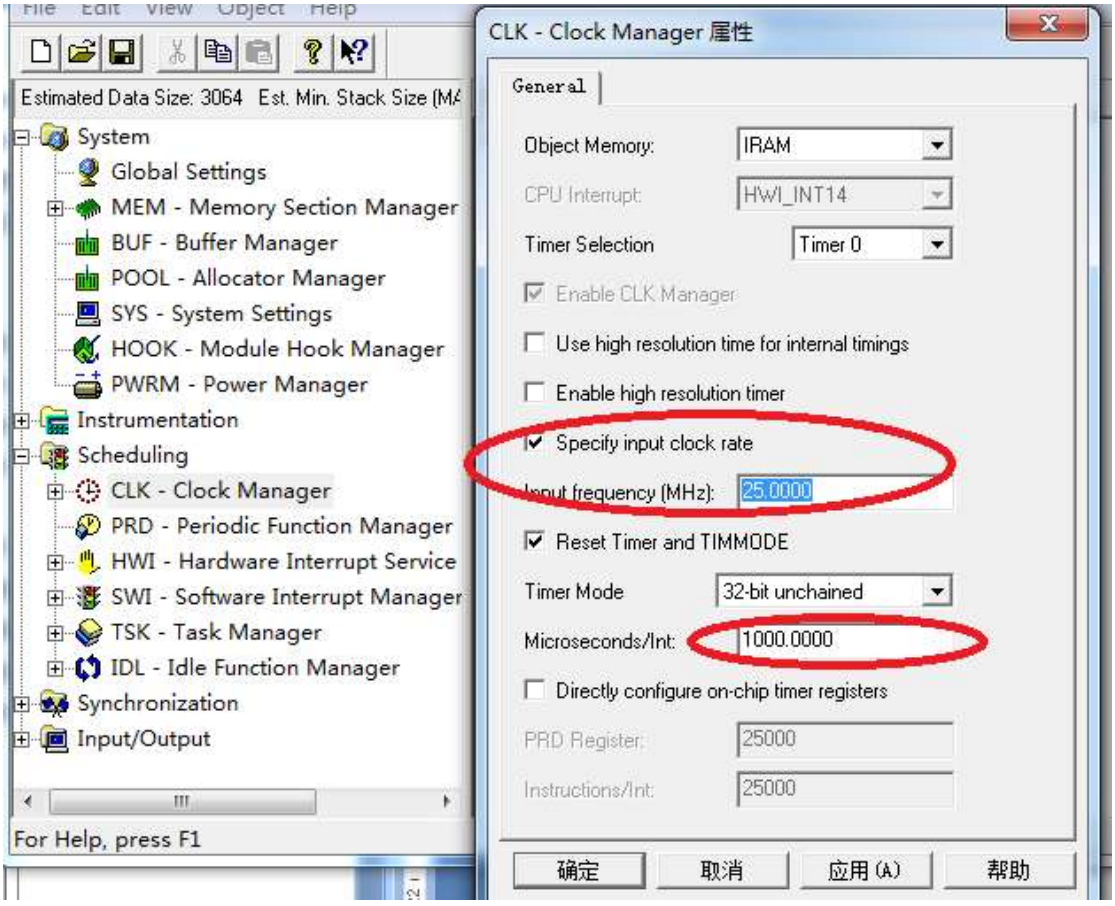
3.3：PRD 对象的创建

PRD 又叫周期函数管理器，它使用系统时钟 CLK 进行驱动，为任务的睡眠提供依据。很多 DSP/BIOS 函数都有一个超时参数 timeout，例如你在任务中使用 TSK_sleep()的函数，这个函数是一个具有超时参数的函数，被调用之后，当系统时钟的变化次数达到超时参数的值时，任务将会推出被阻塞状态。
假如你的系统时钟配置分辨率设置为 1ms，并且希望当前任务阻塞 1s 的时间，那么应该这样调用 TSK_sleep();

TSK_sleep(1000);　任务将被阻塞 1s 钟。

这里使用 C6748 的定时器 0 来做为 CLK 模块的驱动源，使用低分辨率时钟指定输入时钟源为 25MHz。设置时间精度为 1ms 一次，具体配置如图所示。

选择 **Scheduling** 子目录下的 **CLK-Manager**，右键选择属性按钮。



3.4：任务创建

注：每个任务都拥有自己独立的堆栈，任务共有四种状态

运行态(Running)

就绪态(Ready)　　已被调度，等待执行

挂起态(Block)　　也叫阻塞态，等待某个事件发生或某些资源可用

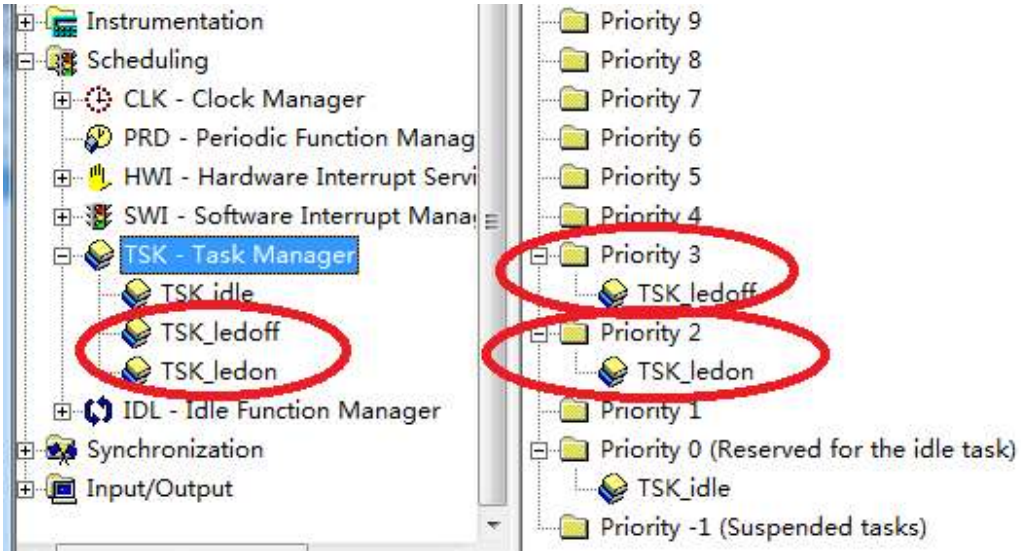
终止态(Terminated)　被终止，不会再执行

其中相同优先级的任务，任务调度器会根据它们在配置工具里列出的顺序进行调度。空闲任务属于 DSP/BIOS 系统的后台线程，有限地最低，其它任何任务线程都可以抢占它，空闲任务用来监测系统状态或执行其它后台操作。

创建步骤

选择 **TSK – Task Manager** ，右键选择插入，并为每个任务起个名字，这里我命名了两个名称分别为 TSK_ledon、TSK_ledoff 的任务。优先级分别为 2 和 3。两个任务一个用来点亮 led，一个用来灭掉 led。这样程序运行起来就会看到 led 在闪烁。

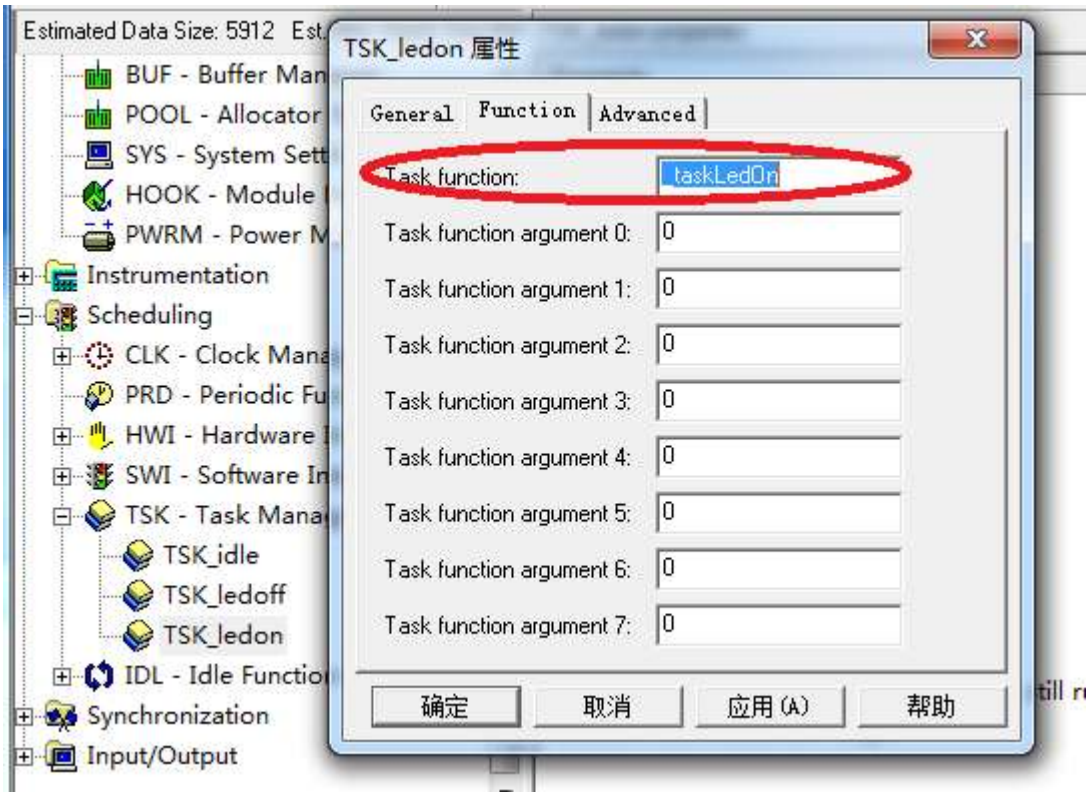
创建的任务后如图：



上述只是创建了任务，接下来还要在为每个任务指定一个函数入口名称。

选中 TSK_ledon，右键选择属性，进行设置，在弹出的对话框中选择 Function 选项，在 Task　function：栏输入要调用的函数入口名称，这里我命名为 taskledon。

注：taskledon 前面要加上一个下划线，这一点一定不能忘记。



同样为 TSK_ledoff 任务指定入口函数名称为 taskledoff。

再此先将代码部分贴出来：

首先在主函数开始前需要包含几个必要的头文件过来

```
#include <std.h>
#include <log.h>
#include <tsk.h>
#include "testcfg.h"


#include "hw_types.h"
#include "psc.h"
#include "soc_C6748.h"
```

```
#include "gpio.h"
void Delay(unsigned int delay)
{
    while(delay--);
}
```

//主函数，我手里的板子在 GP2_1 上接的是一只 Led。对应管脚号为 34，主函数再此小工程里主要完成 GPIO 的初始化工作。

```
int main(void) {
    PSCModuleControl(SOC_PSC_1_REGS, HW_PSC_GPIO, PSC_POWERDOMAIN_ALWAYS_ON,
                     PSC_MDCTL_NEXT_ENABLE);
    HWREG(0x01C14138) = 0x08000000u;
    GPIODirModeSet(SOC_GPIO_0_REGS, 34, GPIO_DIR_OUTPUT);
    GPIOPinWrite(SOC_GPIO_0_REGS, 34,GPIO_PIN_LOW);
    Delay(5000000);
    GPIOPinWrite(SOC_GPIO_0_REGS, 34,GPIO_PIN_HIGH);
    SEM_post(&SEM0);
    return 0;
}

void taskledon()
{
    while(1)
    {
    SEM_pend(&SEM0, SYS_FOREVER);
    GPIOPinWrite(SOC_GPIO_0_REGS, 34,GPIO_PIN_LOW);
    TSK_sleep(500); //Delay(5000000);
    SEM_post(&SEM1);
    LOG_printf(&trace, "Task ledon DONE");
    }
}

void taskledoff()
{
    while(1)
    {
    SEM_pend(&SEM1, SYS_FOREVER);
    GPIOPinWrite(SOC_GPIO_0_REGS, 34,GPIO_PIN_HIGH);
    TSK_sleep(500); //Delay(5000000);
    SEM_post(&SEM0);
    LOG_printf(&trace, "Task ledoff DONE");
    }
}
```

3.5：创建信号灯

上述代码两个任务之间依赖信号灯来触发任务进入就绪状态，实现任务之间的同步和通信。信号灯有一个内部计数器，计有效的资源数，若信号灯大于 0，任务请求该信号灯不会被阻塞。

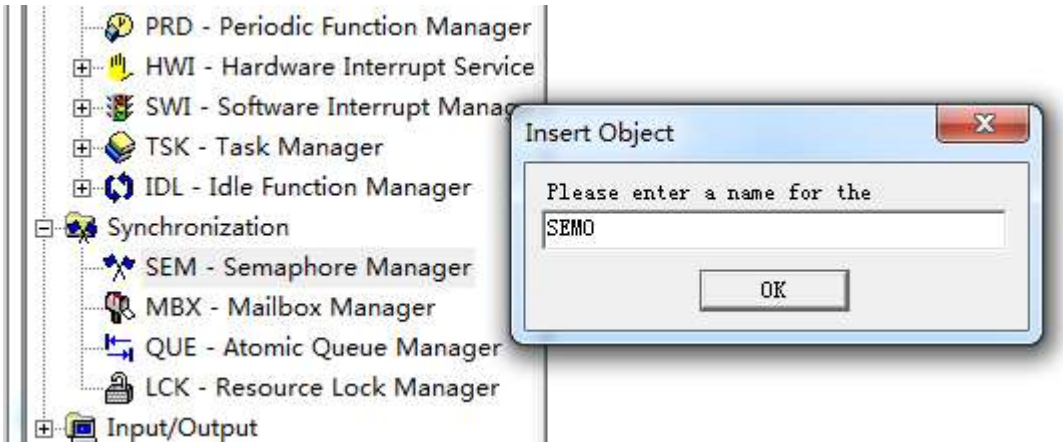
SEM_pend(sem,timeout)：等待一个信号灯，如果信号灯值大于 0，则对计数值做简单的减 1 并返回，否则等待 SEM_post 发布信号。超时参数允许任务等待直到超时，或无限等待(SYS_FOREVER)，或者不等待(取值 0)，返回值代表请求信号灯是否成功。

SEM_post(sem)：发布信号灯。若有任务等待该信号灯，SEM_post 会从等待队列中将该任务删除并将其放入就绪队列等待调度。如果没有任务等待这个信号，SEM_post 则简单将计数值加 1 并返回。

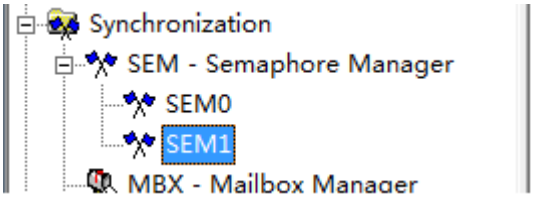
细心的你可能已经发现上述贴出来的示例代码用到了两个信号灯，分别为 SEM0，SEM1。其中 SEM0 用来调度点亮 LED 的任务 taskledon，SEM1 用来调度灭掉 LED 的任务 taskledoff。

创建方法

选择 Synchronization 子目录的 SEM-Semaphore Manager，右键选择插入选项。



如上图所示，创建两个信号灯，创建好后如下图所示：



这样就可以在程序里面使用信号灯了。

4：运行

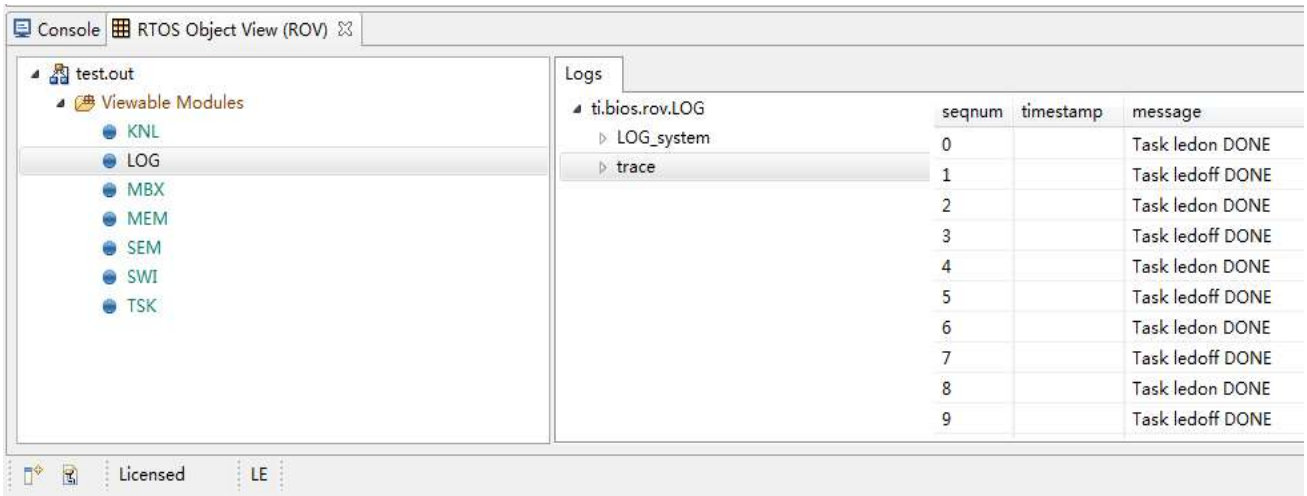
4.1：编译

编译没有错误。

4.2：运行

运行过程中可以看到 LED 在不停的闪烁。

暂停后可以看到的 CCS 界面的 LOG 模块的打印信息如下图：



到此，恭喜你已经可以搭建简单的 DSP/BIOS 应用工程了。

china_fpga@163.com