上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

DSP/BIOS实时操作系统

-- DSP培训课件之六

上海交大-TI联合DSP实验室

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

学习内容

- DSP/BIOS安装及开发步骤
- DSP/BIOS组件的介绍
- DSP/BIOS应用程序的执行顺序
- DSP/BIOS线程
- DSP/BIOS实时分析
- 总结

TI DSP培训以及技术服务简介

上海交大BME-美国德州仪器联合DSP实验室成立于2007年，是国内最权威的TI技术服务于培训机构。实验室有TI（C6000，C2000，C5000，达芬奇，多核DSP）全系列开发平台，提供DSP ，MSP430等技术培训与技术服务，项目合作等。培训内容有

1）CCS开发环境精解与实例；

2）DSP/SYS BIOS 实例；

3）C6000/C5000/C2000全系列DSP架构以及汇编，C语言，混合编程等；

4）HPI，EMIF，EDMA，Timer等外设；

5）C6416、DM642， C6678多核EVM开发平台实例；

6）Bootloader 原理以及实例等。

常年开班，三人以上集体报名8折优惠，学生5折。

联系电话：13651621236（牛老师），

邮件报名：jhniu@sjtu.edu.cn ，niujinhai@yahoo.com.cn

颁发TI授权的培训证书

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

DSP/BIOS安装及开发步骤

DSP/BIOS的安装

- 安装文件名：

  dsp_bios_setupwin32_5_31_02_08.exe（路径：CCS3.3\CCS安装\setup\DSP_BIOS）

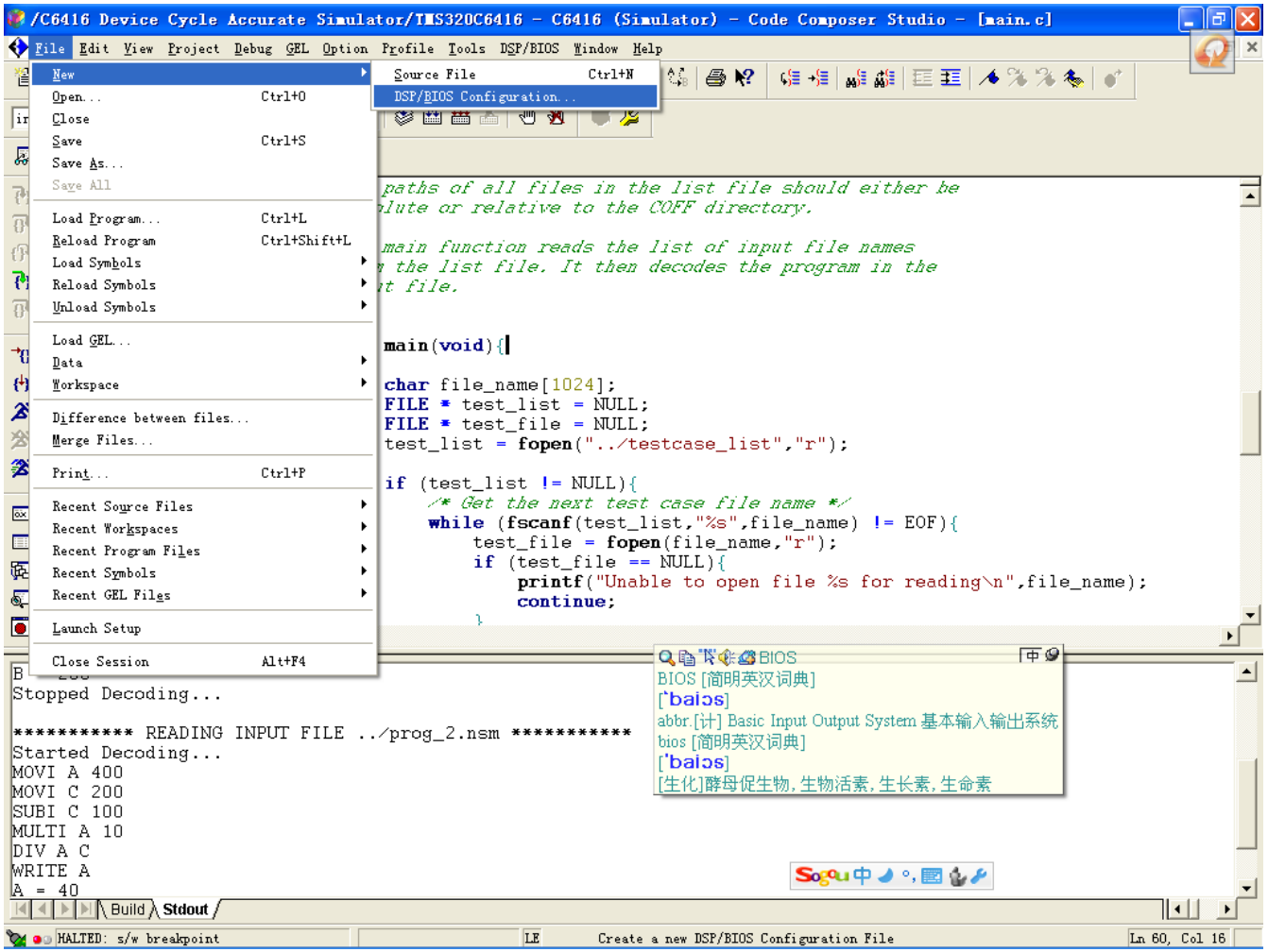
- 双击安装图标，根据安装向导安装即可。

DSP/BIOS程序的开发步骤

- 1、使用C语言或汇编语言编写程序框架；
- 2、使用配置工具创建程序使用的对象；
- 3、保存配置文件，保存时产生的文件将在编译和链接程序中使用；
- 4、使用工程文件编译和链接程序；
- 5、使用simulator或初始硬件（initial hardware）和DSP/BIOS插件测试程序；
- 6、重复2-5步直到程序运行正确；
- 7、当硬件产品设计好后，修改配置文件以支持硬件，并在硬件上测试程序。

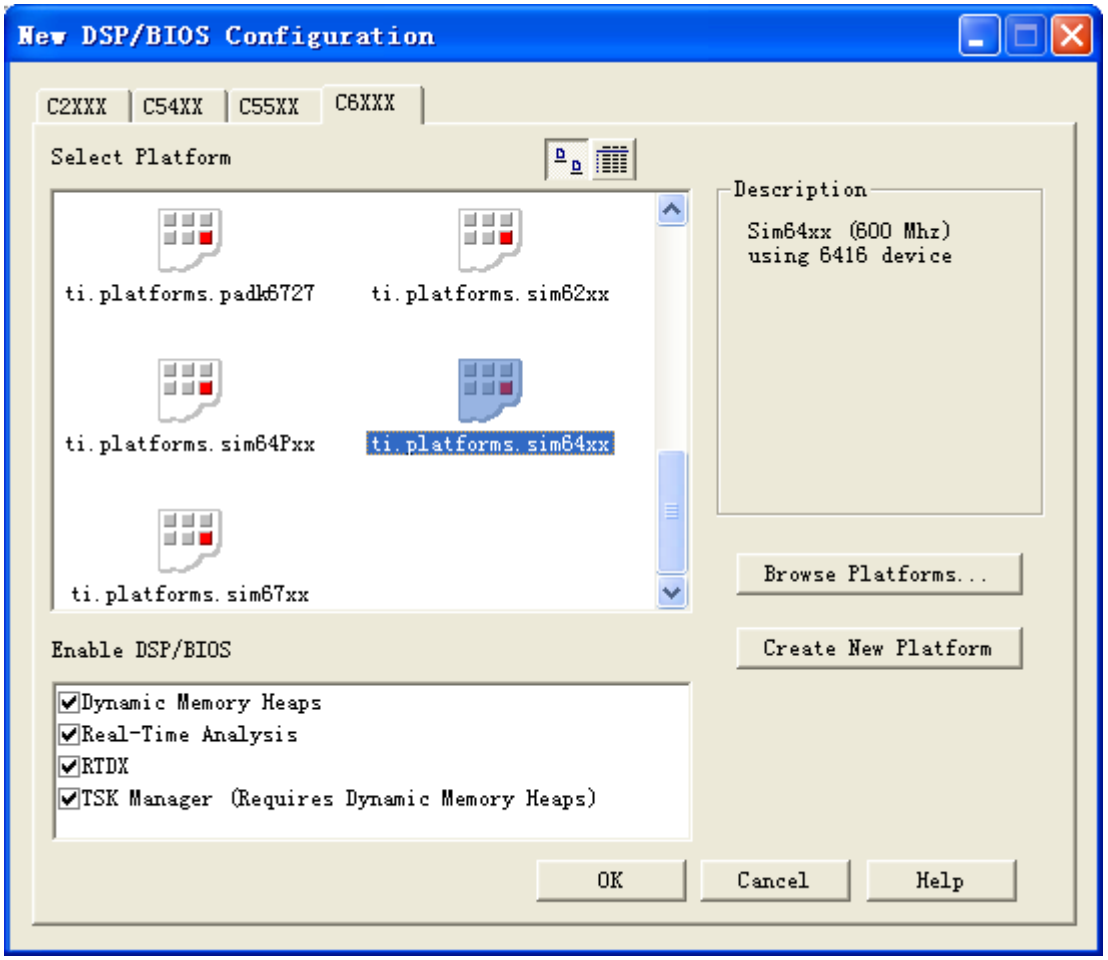
新建一个DSP/BIOS对象

选择File → New → DSP/BIOS Configuration



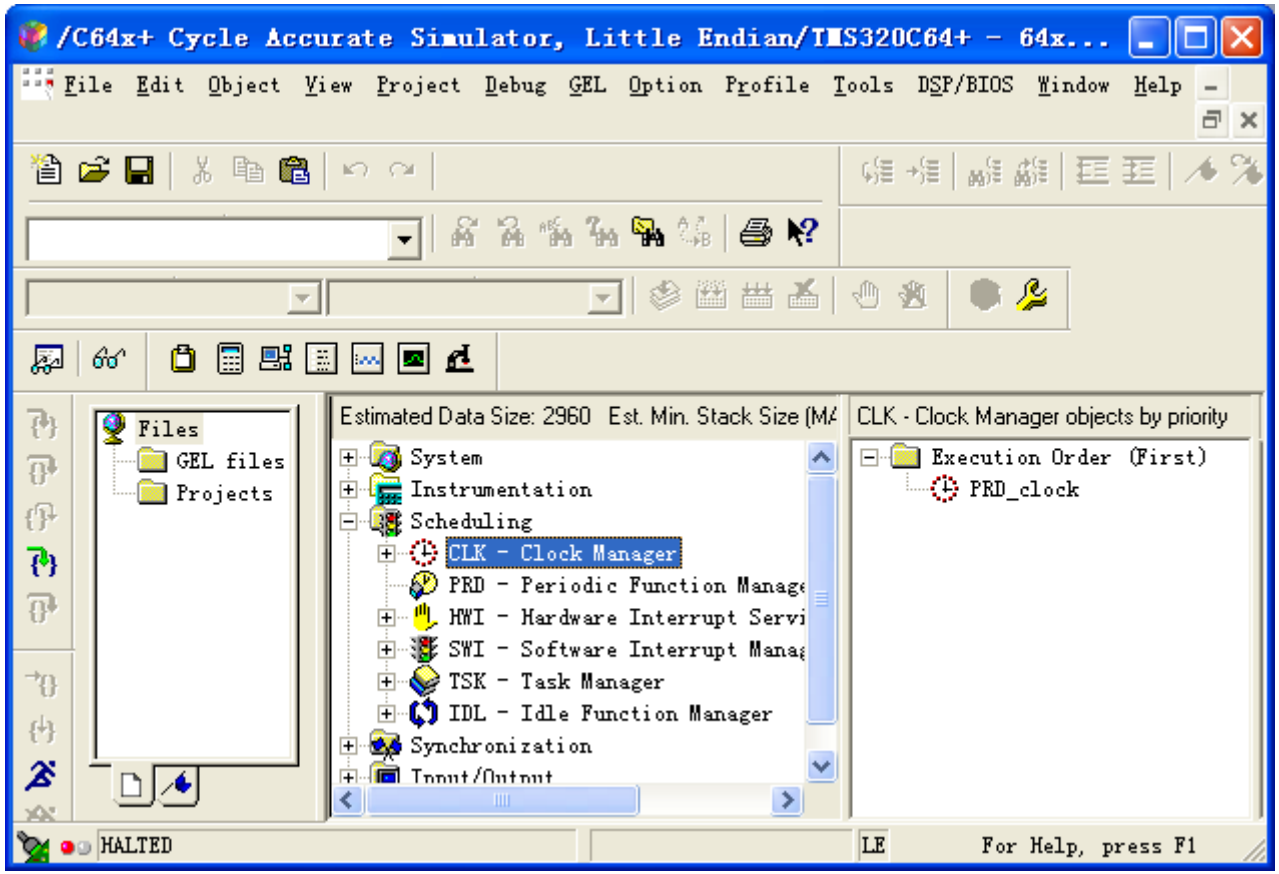
新建一个DSP/BIOS对象

- 在弹出的对话框中选择需要的开发平台(如在此选择sim64xx），单击OK。



新建一个DSP/BIOS对象

- 在配置窗口中新建、修改DSP/BIOS对象并保存。

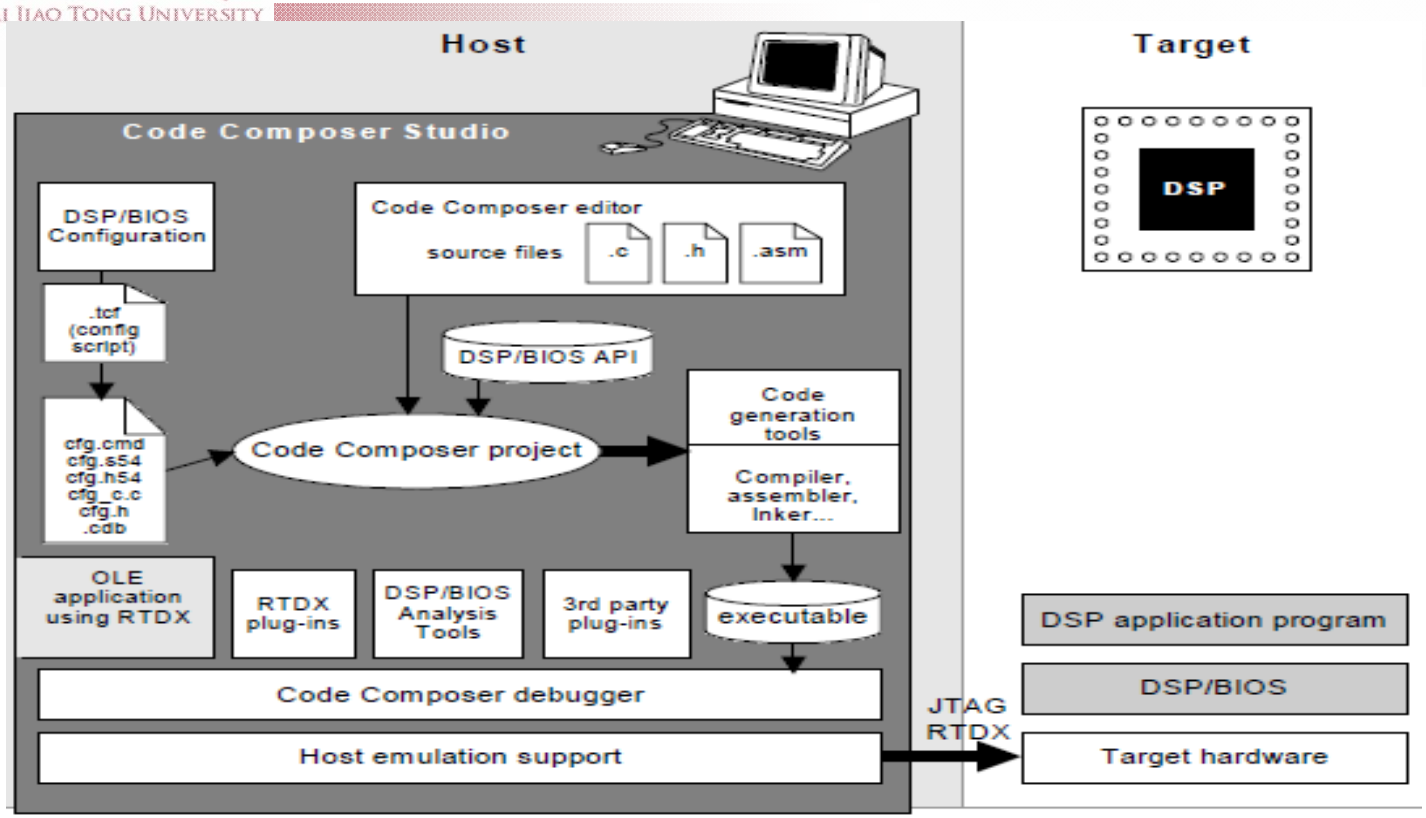


新建一个DSP/BIOS对象

- 将配置文件保存为XXX.tcf,此时还会产生其他的一些程序必需用到的链接命令文件（*cfg.cmd）、头文件（*cfg.h)、汇编源文件（*cfg.s64)等。
- 将产生的文件添加到工程中，注意有些文件可以自动添加有些则要手动添加。

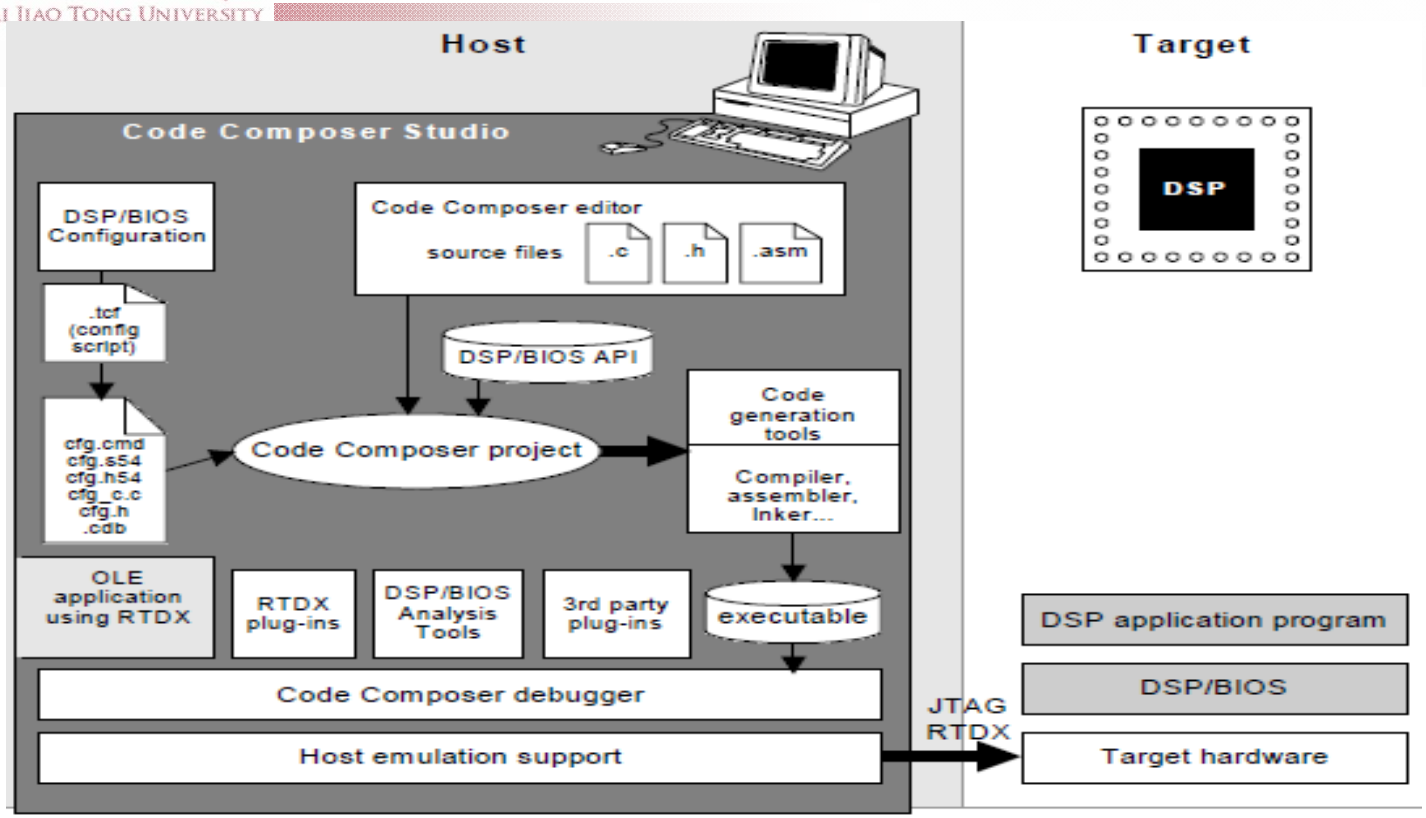
DSP/BIOS组件的介绍

DSP/BIOS组件



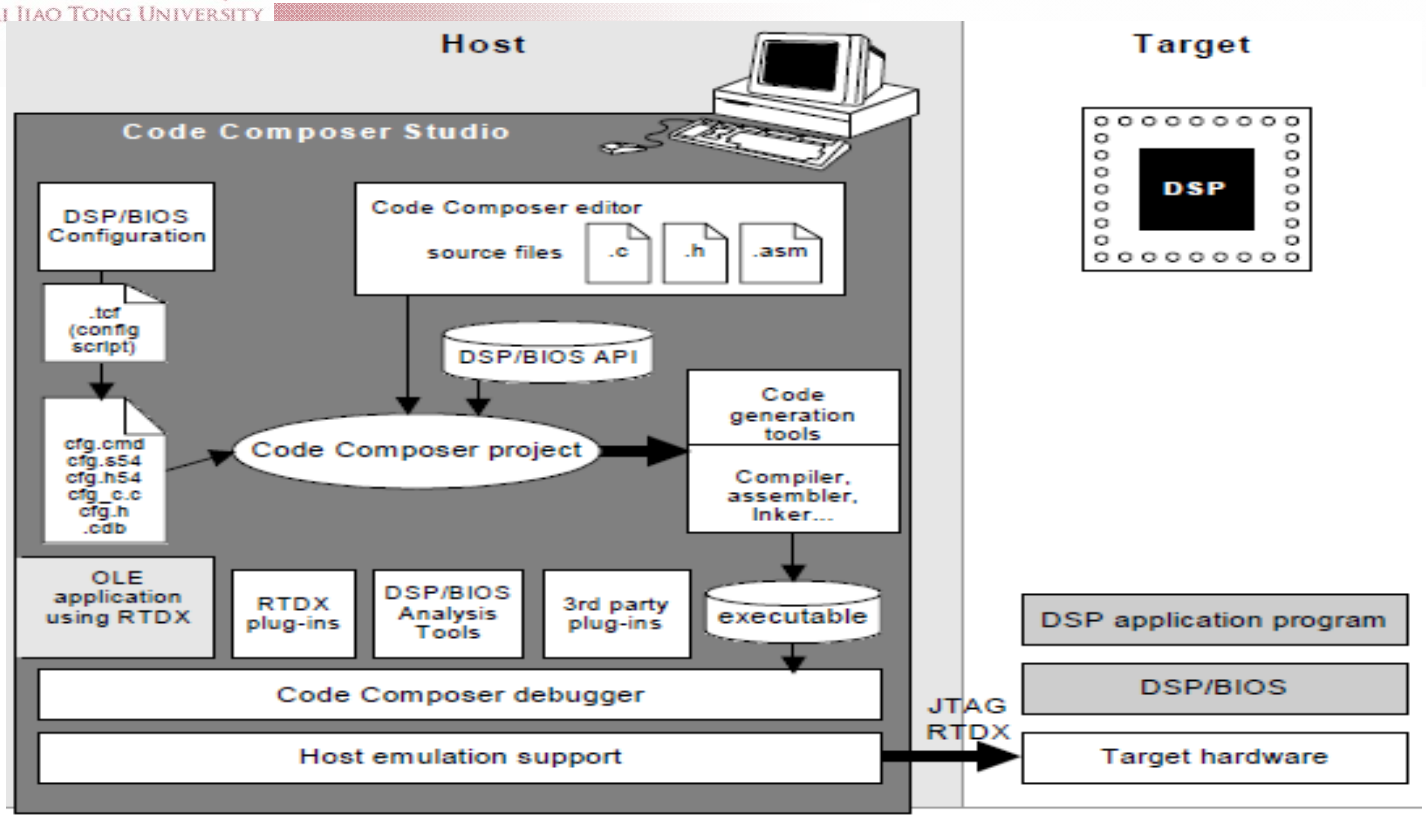
- DSP/BIOS API(Application Programming Interface):当我们在主机上编写源程序（C、C++、汇编）时，可以在源程序中调用API函数（如实验中调用了SWI模块的SWI_pose函数）

DSP/BIOS组件



- DSP/BIOS配置工具（DSP/BIOS Configuration）：在配置工具中可以静态的创建和删除DSP／BIOS对象。配置文件保存在.tcf文件中，同时还会产生其他用于编译和链接的文件。

DSP/BIOS组件



- DSP/BIOS插件(DSP/BIOS Analysis Tools）：利用该插件，开发人员可以测试DSP目标系统板上的应用程序，监视CPU的负荷(load)、时序(timing)、日志(logs)以及线程执行等。

DSP/BIOS - API 模块

- DSP/BIOS API被分成很多不同的模块，只有需要的模块才会链接到应用程序中，被使用的模块要在源代码中包含相应的头文件。应用程序通过调用DSP/BIOS API函数来完成各种诊断调试功能。
- DSP/BIOS API专门为实时DSP程序优化，可以为嵌入式程序提供基本的运行服务。与标准C库函数（如puts函数）不同，它可以再不中断目标板硬件的情况下对DSP系统进行实时分析。同时，DSP/BIOS API代码占用更少的空间，运行速度比标准C输入/输出更快。一个DSP程序可以根据需要使用一个或多个DSP/BIOS模块。

DSP/BIOS - API 模块

- 对于C程序，API函数需要在头文件中声明。对于DSP汇编程序，API函数需要在宏中声明。为了减少开发周期和代码量，DSP/BIOS实时库采用汇编语言编写，因此，使用汇编语言编写DSP/BIOS应用程序（不是DSP应用程序）比使用C语言编写更为方便。

DSP/BIOS – API 模块

设备/实时分析

LOG	Message Log manger
STS	Statistics accumulator manager
TRC	Trace manager
RTDX	Real-Time Data Exchange manager

线程类型

HWI	Hardware interrupt manager
SWI	Software interrupt manager
TSK	Multitasking manager
IDL	Idle function & processing loop manager

时钟和周期函数

CLK	System clock manager
PRD	Periodic function manger

线程间的同步与通信

SEM	Semaphores manager
MBX	Mailboxes manager
LCK	Resource lock manager

输入/输出

PIP	Data pipe manager
HST	Host input/output manager
SIO	Stream I/O manager
DEV	Device driver interface

内存和低级原语

MEM	Memory manager
SYS	System services manager
QUE	Queue manager
ATM	Atomic functions
GBL	Global setting manager

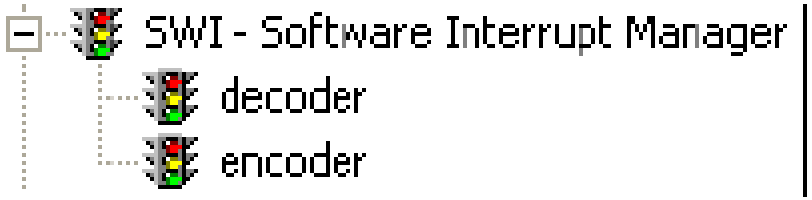
DSP/BIOS Configuration

- 有两种方法创建DSP/BIOS对象：

1、通过函数动态创建或删除对象：

```
prog.module("SWI").create("encoder");
prog.module("SWI").create("decoder");
```

2、使用配置工具静态的创建和删除对象：



- 我们一般采用配置工具创建或删除对象，因为：

1、能与DSP/BIOS插件更好的结合起来；

2、可以减少应用程序的代码量；

3、改善运行性能，缩短程序执行时进行系统设置的时间；

- DSP/BIOS配置工具是一个与Windows资源管理器具有相似界面的可视化编辑器，创建目标程序DSP/BIOS API所调用的运行对象和设置其属性。其配置工具模块树窗口如图：

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

BIOS配置组件

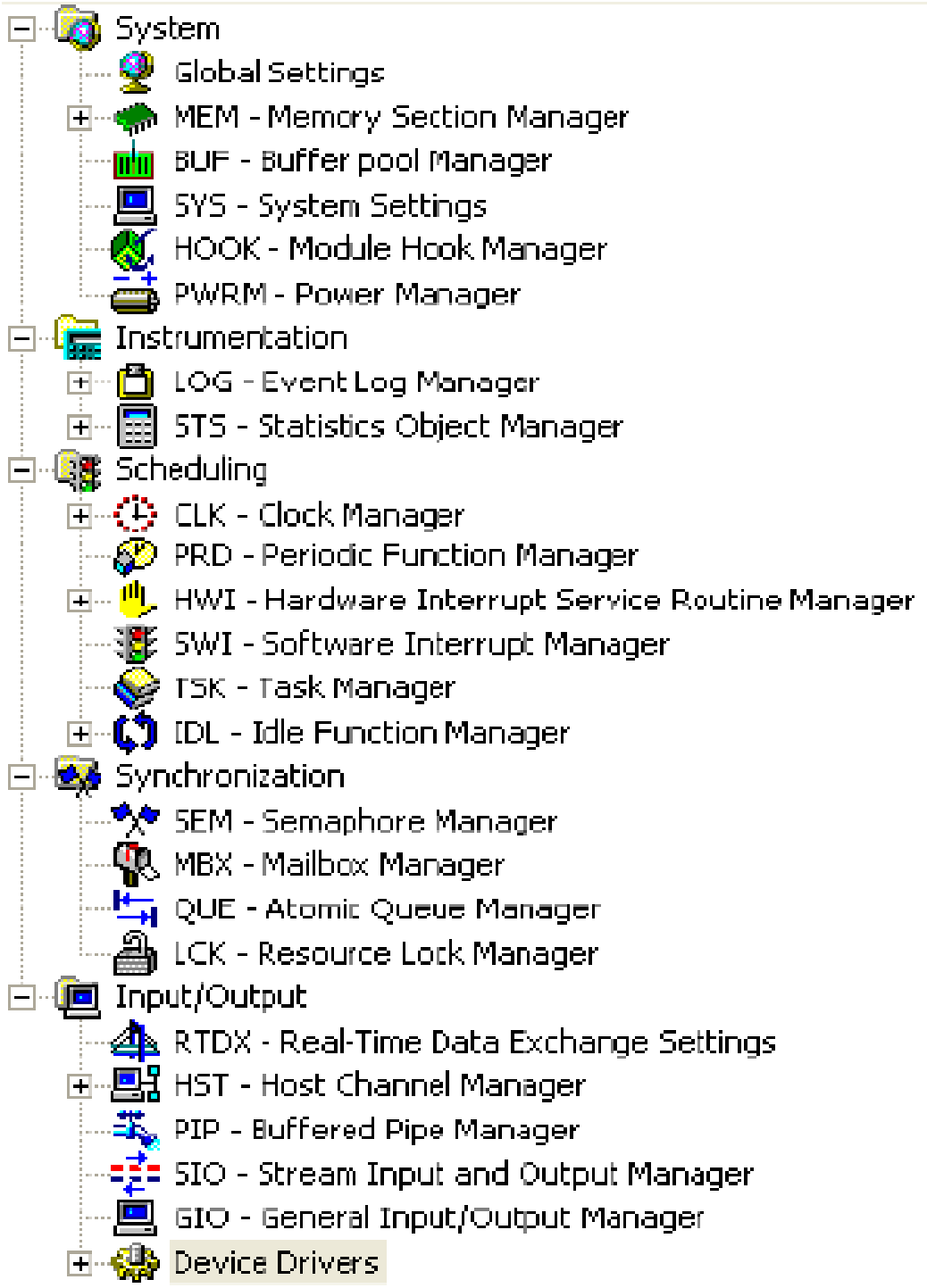
1）系统

2）设备

3）调度

4）同步

5）输入输出



DSP/BIOS插件

DSP/BIOS插件的实时分析特性

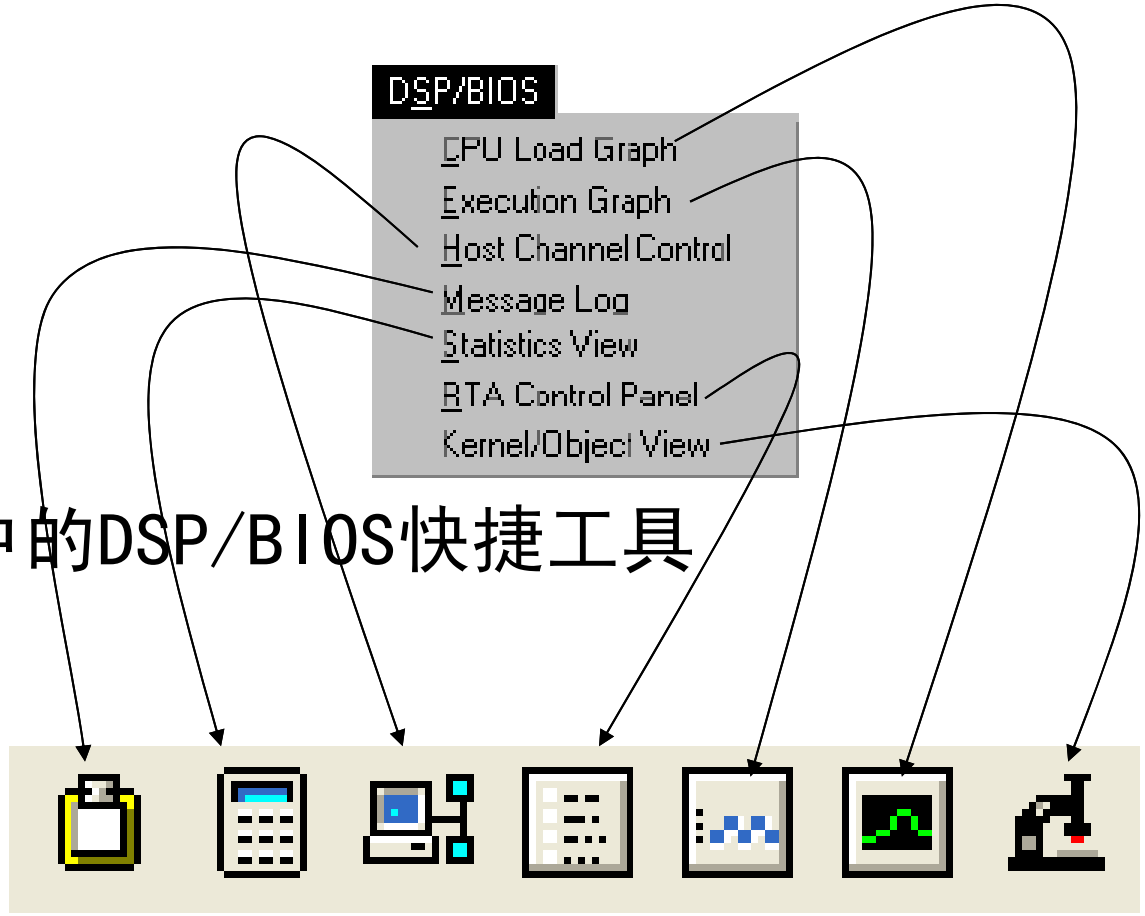
CCS能利用DSP/BIOS插件对DSP应用程序进行实时分析，并可以实时监测DSP应用程序的运行，同时对DSP应用程序的实时性的性能影响很小。其提供的实时分析特性如下：

1、程序跟踪：能显示写入目标日志的事件并在程序执行过程中反应动态控制流程；

2、性能监控：能动态跟踪和统计DSP目标系统板上的资源使用情况，如DSP处理器的负载和线程的时序；

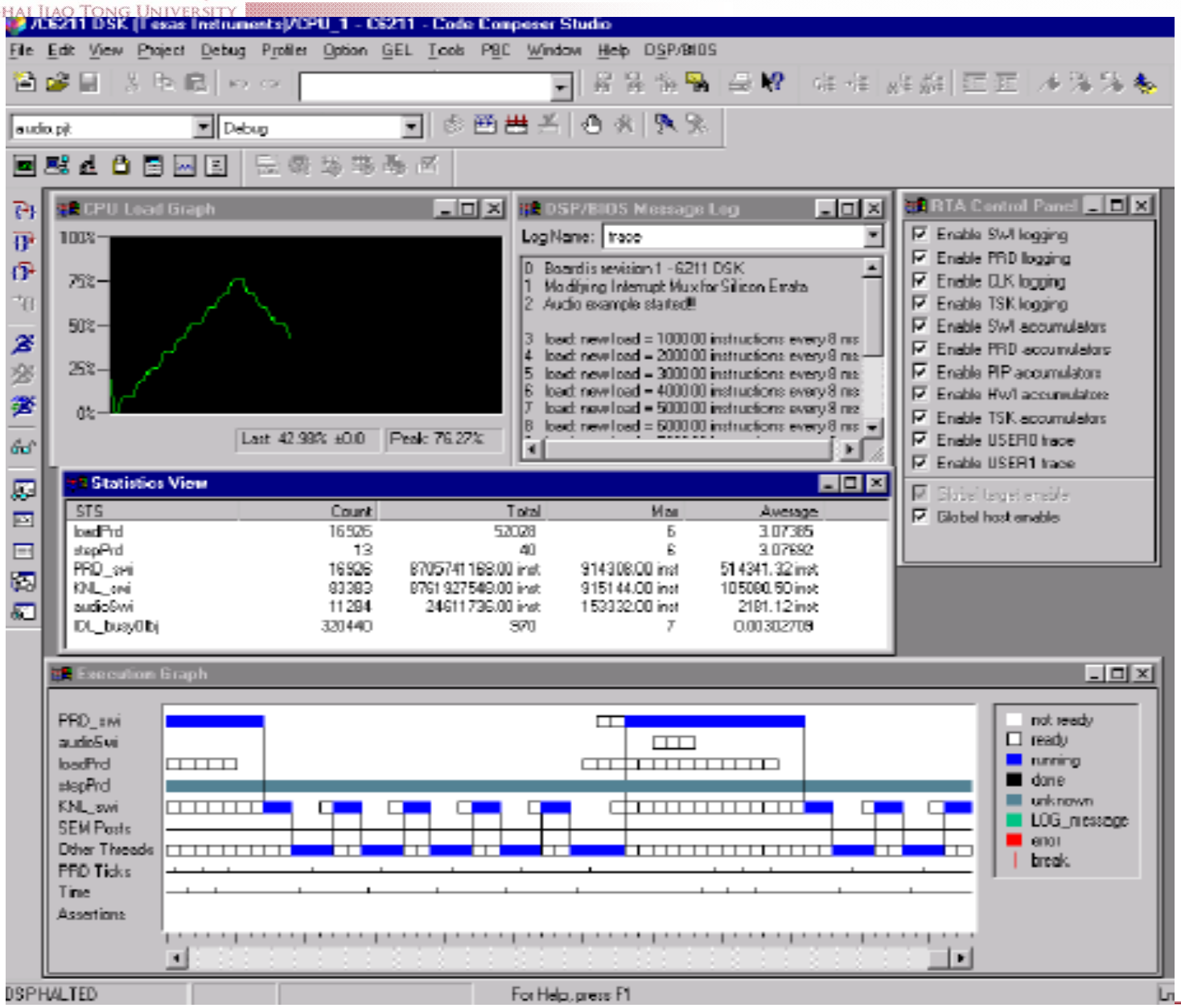
3、文件流：能将DSP目标系统板上的I/O对象与PC主机上的文件关联起来。

- CCS中的DSP/BIOS菜单



- CCS中的DSP/BIOS快捷工具

实时分析窗口


/C6211 DSK | Texas Instruments]/CPU_1 - C6211 - Code Composer Studio
File Edit View Project Debug Profiler Option GEL Tools PBC Window Help DSP/BIOS
audio.pjt
Debug
CPU Load Graph
100%
75%
50%
25%
0%
Last: 42.98% ±0.0
Peak: 76.27%
DSP/BIOS Message Log
Log Name: trace
0 Board is revision 1 - 6211 DSK
1 Modifying Interrupt Mux for Silicon Errata
2 Audio example started!
3 load: new load = 100000 instructions every 8 ms
4 load: new load = 200000 instructions every 8 ms
5 load: new load = 300000 instructions every 8 ms
6 load: new load = 400000 instructions every 8 ms
7 load: new load = 500000 instructions every 8 ms
8 load: new load = 600000 instructions every 8 ms
RTA Control Panel
Enable SWI logging
Enable PRD logging
Enable CLK logging
Enable TSK logging
Enable SWI accumulators
Enable PRD accumulators
Enable PIP accumulators
Enable HWI accumulators
Enable TSK accumulators
Enable USER0 trace
Enable USER1 trace
Global host enable
Statistics View
STS Count Total Max Average
loadPrd 16926 52028 6 3.07385
stepPrd 13 40 6 3.07692
PRD_swi 16926 8705741.00 inst 914308.00 inst 514341.32 inst
KNL_swi 83383 8761927.00 inst 915144.00 inst 105080.90 inst
audioSwi 11284 24611736.00 inst 153032.00 inst 2181.12 inst
IDL_busyObj 320440 970 7 0.003027
Execution Graph
PRD_swi
audioSwi
loadPrd
stepPrd
KNL_swi
SEM Posts
Other Threads
PRD Ticks
Time
Assertions
not ready
ready
running
done
unknown
LOG_message
error
break
DSP HALTED
For Help, press F1

DSP/BIOS应用程序的执行顺序

DSP/BIOS应用程序的执行顺序

当一个DSP/BIOS应用程序执行时，执行顺序由启动文件boot.s54文件(C54x平台)或autoinit.c文件和boot.snn文件（C6000和C55x平台）中的调用和指令所决定。在bios.ann文件和bios.ann库文件中提供了一个已经编译好的启动文件，源代码位于产品中已分配的固定磁盘内。启动文件中源代码所指定的DSP/BIOS启动时序如下。用户不用改变启动时序。

DSP/BIOS应用程序的执行顺序

1） 初始化DSP：DSP / BIOS应用程序从C 或C ++环境语言入口点c_int00处开始执行。当复位后，复位中断矢量把程序入口点设置到c_int00处。同时：

- 对于C54x平台，在C_int00处开始时，系统堆栈指针（SP）被设置为指向堆栈底部。状态寄存器（如st0和st1）也被初始化。
- 对于c55x平台，在C_int00处开始时，数据（用户）堆栈指针（XSP）和系统堆栈指针（XSSP）都各自的指向用户和系统堆栈的底部。另外XSP对齐于一个平滑的地址边界。
- 对于c6000平台，在C_int00处开始时，系统堆栈指针(B15)和全局页面指针（the global page pointer）(B14)分别指向堆栈段底部和.bss的起始处。控制寄存器（如AMR，IER和CSR）都被初始化。

DSP/BIOS应用程序的执行顺序

2）初始化.cinit记录中的.bss：一旦堆栈建立，可以通过调用初始化例程来初始化.cinit记录中的各变量。

3）调用BIOS_init初始化应用程序中使用到的DSP/BIOS模块： BIOS_init由配置对象产生并位于programcfg.snn文件中。 BIOS_init负责基本模块的初始化。BIOS_init包含应用程序中使用到的每一个DSP/BIOS模块的MOD_init宏。具体过程如下：

DSP/BIOS应用程序的执行顺序

- **HWI_init 设置ISTP和中断选择寄存器：**在c6000平台中，它设置IER中的NMIE位，同时它还清除所有平台中的IFR寄存器。想要了解更多关于你使用的平台的HWI模块信息，请参考TMS320系列的DSP/BIOS API参考指南。

注意：

在配置一个中断时，DSP/BIOS插入到中断服务例程中的中断服务表的合适位置。但是DSP/BIOS并不使能IER中的中断位。这一工作必须由用户自己在启动或应用程序执行时的任何合适的时间进行。

DSP/BIOS应用程序的执行顺序

- **HST_init 初始化主机I/O通道接口：**这一例程的具体情况依赖于主机与DSP目标系统链接的特定实现。例如，在c6000平台中，如果RTDX被使用，HST_init就会使能IER中为RTDX预留的硬件中断位。
- **IDL_init 计算空闲循环指令数：**如果在配置idle对象属性时，将自动计算空闲循环指令数设置为真，那么IDL_init将在这里开始计算空闲循环指令执行次数。空闲循环指令框用来校准显示所计算的CPU负载图上的CPU负荷（详见3.4.2节，CPU负载，3-20页）。

DSP/BIOS应用程序的执行顺序

4）**处理.pinit表：**.pinit表由一系列指向用于初始化的函数的指针组成。对于C++程序，全局对象的类构造函数在.pinit进程中执行。

5）**调用程序主程序：**在所有的DSP/BIOS模块初始化完成后，用户的主程序就被调用。这一主程序可以用汇编、C、C++或这些语言的组合来编写。因为C编译器在主函数名前自动加了一个下划线前缀，所以主函数的形式可以是用C、C++编写的main()或用汇编语言编写的_main()。既然无论是硬件中断还是软件中段都还没有被打开，你可以在主函数中详细的为你的应用程序进行初始化（例如，调用你自己的硬件初始化程序）。

DSP/BIOS应用程序的执行顺序

6）调用BIOS_start启动DSP/BIOS ：和BIOS_init一样，BIOS_start也是通过配置产生的，位于programcfg.snn文件中。BIOS_start在主函数返回后被调用，它主要负责使能DSP/BIOS模块并且为每一个DSP/BIOS模块调用MOD_startup宏。如果TSK管理器在配置时被使能，那么对BIOS_start的调用不会返回。具体过程如下：

- CLK_startup宏 设置PRD寄存器，使能在CLK管理器中选择的计数器在IER（c6000平台）或IMR（C5400平台）中的位。（此宏只有在配置工具中使能了CLK管理器时才起作用。）

DSP/BIOS应用程序的执行顺序

- PIP_startup　为每一个已经创建的PIP（管道）对象调用notifyWriter函数。
- SWI_startup　使能软件中断。
- HWI_startup　在c6000平台中，通过置位CSR中的GIE位或者在C5400平台中通过清除ST1寄存器中的INTM位使能硬件中断。
- TSK_startup　使能任务调度并启动准备队列中优先级最高的任务。若应用程序没有任务在准备队列，TSK_idle执行并调用IDL_loop。一旦TSK_startup被调用，应用程序开始，而且不从TSK_startup或BIOS_start中返回。只有任务管理器在配置时被使能时TSK_startup才运行。

DSP/BIOS应用程序的执行顺序

7）**执行闲置循环：**用户可以通过一到两种方法进入闲置循环。第一种方法，使能任务管理器。任务调度程序运行TSK_idle，TSK_idle调用IDL_loop。第二种方法，任务管理器未使能，因此BIOS_start调用返回，接着调用IDL_loop，引导程序进入DSP/BIOS无限闲置循环。在此时，软件或硬件中断可以发生并抢占Idle的执行。因为idle loop管理目标系统与主机的通信，所以在idle loop期间可以进行主机-目标系统间的数据传输。

DSP/BIOS应用程序的执行顺序

高级启动：仅限C5500平台

- 在c5500平台中，通过设置寄存器IVPD和IVPH，该平台允许对矢量表（总长度是256各字节）的初始化进行重新编程。默认情况下，硬件复位这两个寄存器为0xFFFF，复位向量提取至0xFF-FF00。为了将矢量表放至存储器其他位置，必需在硬件复位后向IVPD和IVPH写入相应的地址，同时做一个软件复位，此时IVPD和IVPH中的新值就能生效了。

DSP/BIOS应用程序的执行顺序

- HWI_init将配置过的矢量表地址加载到IVPD和IVPH，但是其后要跟一个软件复位以确保新的IVPD和IVPH生效。

- C5500平台支持三种可能的堆栈模式（详见表2-3）。为了将处理器配置为任何一种非默认模式，用户需要通过CCS调试工具适当的设置位28和29到复位矢量位置。想了解更多的内容请参考TMS320C55x DSP CPU参考指南。

DSP/BIOS应用程序的执行顺序

表2-3：C5500平台下的堆栈模式

堆栈模式	描述	复位矢量设置
2×16 快速返回	独立于SP/SSP，使用RETA/CFCT用于快速返回功能	XX00：XXXX：<24位矢量地址>
2×16 慢返回	独立于SP/SSP，不使用RETA/CFCT	XX01：XXXX：<24位矢量地址>
1×32 慢返回（重置默认）	与SP/SSP同步，不使用RETA/CFCT	XX02：XXXX：<24位矢量地址>

另外，DSP/BIOS配置必需在HWI管理器中设置堆栈模式属性，使之与应用程序的模式相符合。详细内容请参考TMS320C5000 DSP/BIOS API参考指南。

DSP/BIOS线程

关键字

- 非抢占（No preemption）：资源不能被抢占，这意味着只有持有资源的进程才能释放它。.
- 对象（Object）：DSP/BIOS提供的数据和代码结构集，如一个事件，任务，信号量。
- 挂起（Pend）：等待一个事件资源，暂时被淘汰出内存。
- 触发（Post）：给一个事件信号，如触发一个软中断，即将软中断置于准备队列中。
- 抢占（Preemption）：一个更高优先级的函数（或线程）打断优先级更低的函数（或线程）。
- 优先级调度（Priority scheduling）：优先级调度可以是抢占的也可以是非抢占的。一个抢占式优先级调度算法在有更高优先级的进程到来时会抢占（释放）CPU，
- 进程（Process）：一个任务或执行线程。
- 调度器（Scheduler）：管理线程间执行的系统软件。
- 调度（Scheduling）：用于共享资源的方法。
- 信号量（Semaphore）：同步系统对象，使任务间能同步活动

DSP/BIOS线程

IDL：空闲函数模块，管理空闲状态函数，当目标程序没有更高优先级的程序执行时，DSP处于空闲状态，此间可进行主机与目标系统的数据传输；

HWI：硬件中断模块，为硬件中断程序提供支持，在配置文件中可指定发生时要运行的函数；频率可达200KHz（5us），处理时限在2us～100us .

SWI：软件中断模块，管理软件中断，当一个目标程序使用API调用发送一个SWI对象时SWI模块调度相关函数的执行，软件中断共有15个优先级但都比硬件优先级低。时限100us以上，SWI允许HWI将一些非关键处理在低优先级上延迟执行，这样可减少在中断服务程序中的驻留时间。

TSK：任务模块，管理比软件中断优先级低的任务线程；任务与软件中断不同的地方在于在运行过程中可以被挂起。DSP/BIOS提供了一些任务间同步和通讯的机制，包括队列、信号灯和邮箱。

DSP/BIOS线程类型

HWI
硬中断

- **HWI** 的优先级由硬件设置
  一个中断对应一个**ISR .**

SWI
软中断

- **SWI**有**15**个优先级
  多个**SWI**可以位于同一个优先级.

TSK
Tasks

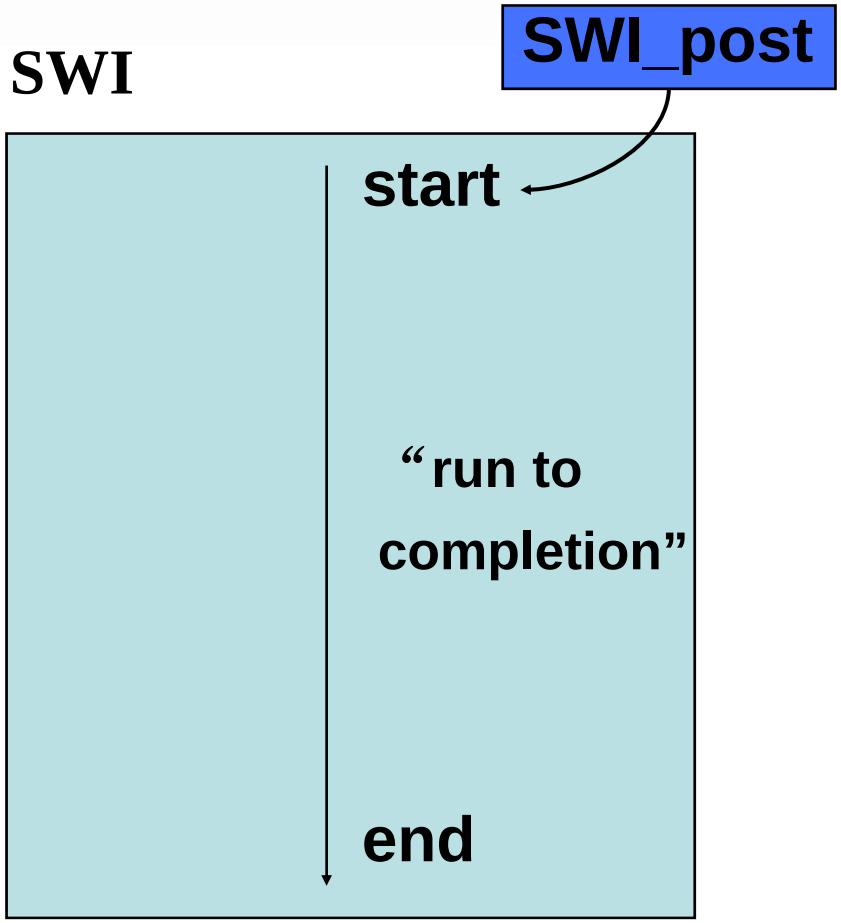
- **TSK**有**15**个优先级
  多个**TSK**可以位于同一个优先级.

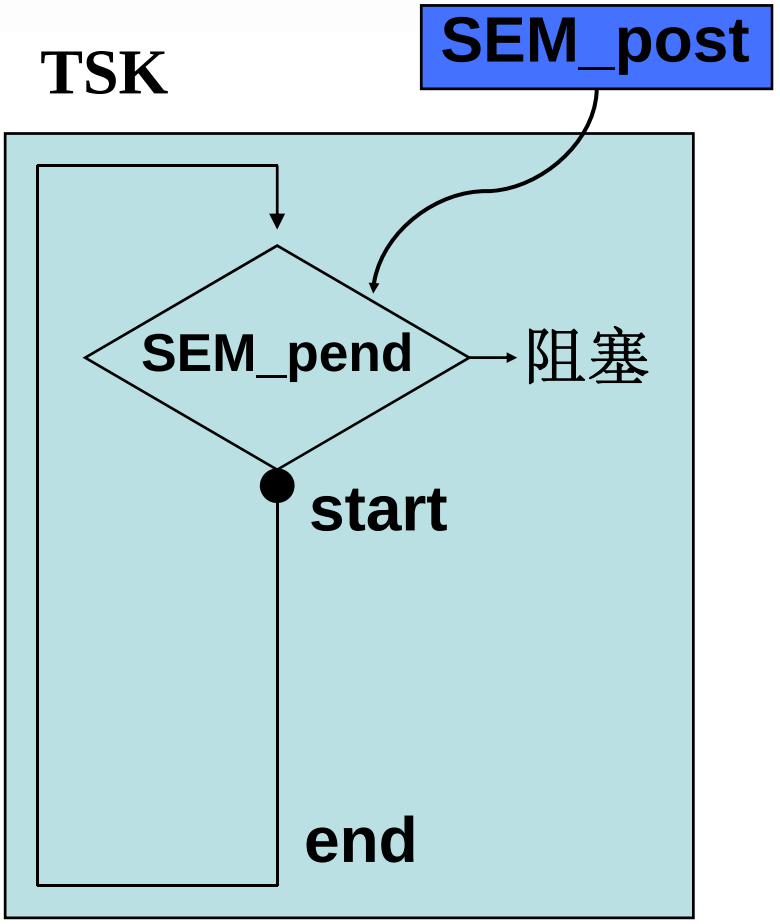
IDL
后台

- 多个**IDL**函数可以无限循环.

HWI 由硬件中断触发.
IDL 运行于后台线程.
那么是什么触发SWI与TSK呢?

触发SWI与TSK





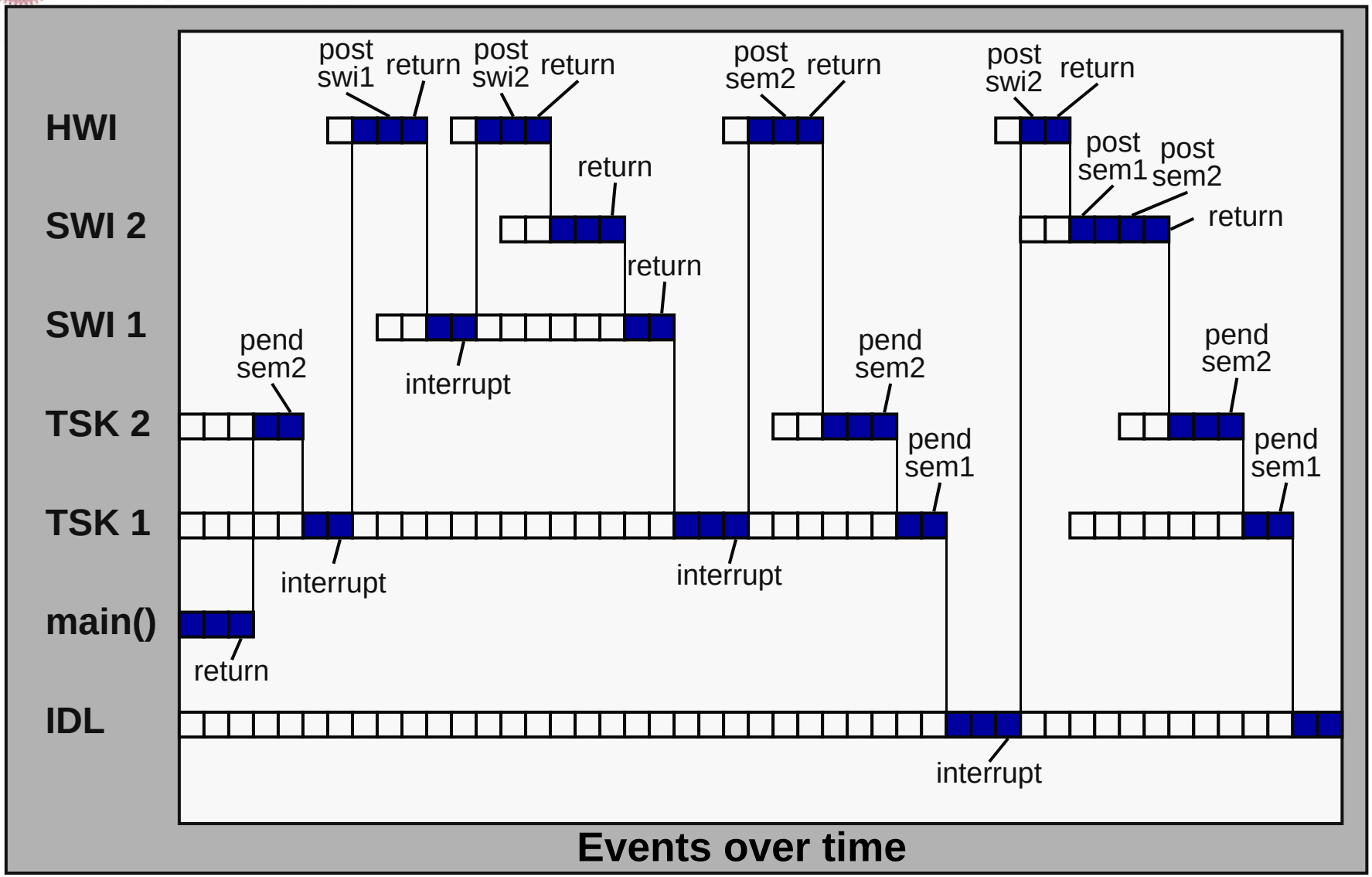
SWI 不能挂起，不能被阻塞。
SWI 常从函数中返回.

TSK只有在不需要时返回,否则一直无限循环。任务在等待某个资源时可以被阻塞。

如何选择线程类型

- 线程延时和数据传输速率。
- 多层次中断响应：
  - HWI 较快 (for sample-by-sample response time).
  - SWI 更慢 (triggered to process frame).
- 线程的优先级。
- 堆栈需求：
  - SWI使用系统堆栈。
  - TSK使用自定义的堆栈。.
- 同步及通信方式：
  - SWI和TSK的方式不同。.
- 用户的偏好及易于使用。

线程抢占的例子


HWI
SWI 2
SWI 1
TSK 2
TSK 1
main()
IDL
post swi1
return
post swi2
return
post sem2
return
post swi2
return
return
post sem1
post sem2
return
return
pend sem2
interrupt
pend sem2
pend sem2
pend sem1
pend sem1
interrupt
interrupt
return
interrupt
Events over time

实验室例子

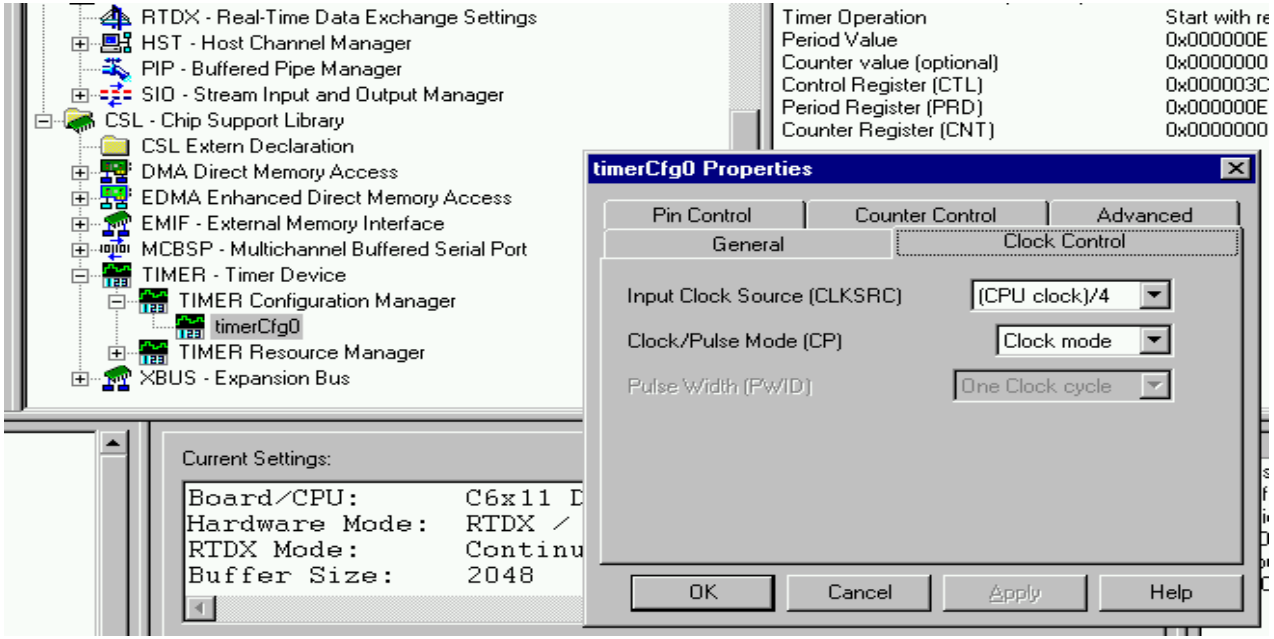
- 目标：
  - (1) 设置内部定时器1产生频率8kHz的信号
  - (2) 创建一个由内部定时器1触发的硬件中断
  - (3) 创建一个软件中断，在硬件中断中触发
  - (4) 创建一个任务，在硬件中断中触发
  - (5) 创建一个信号量，可以被任务函数使用

（1）设置内部定时器1

(1)创建一个新的工程，命名为“bios_lab.pjt”.

(2)创建一个新的配置文件，命名“bios_lab2.cdb”.

(3)在“Timer Configuration Manager”中创建一个timer对象并命名为“timerCfg0”，设置其属性:

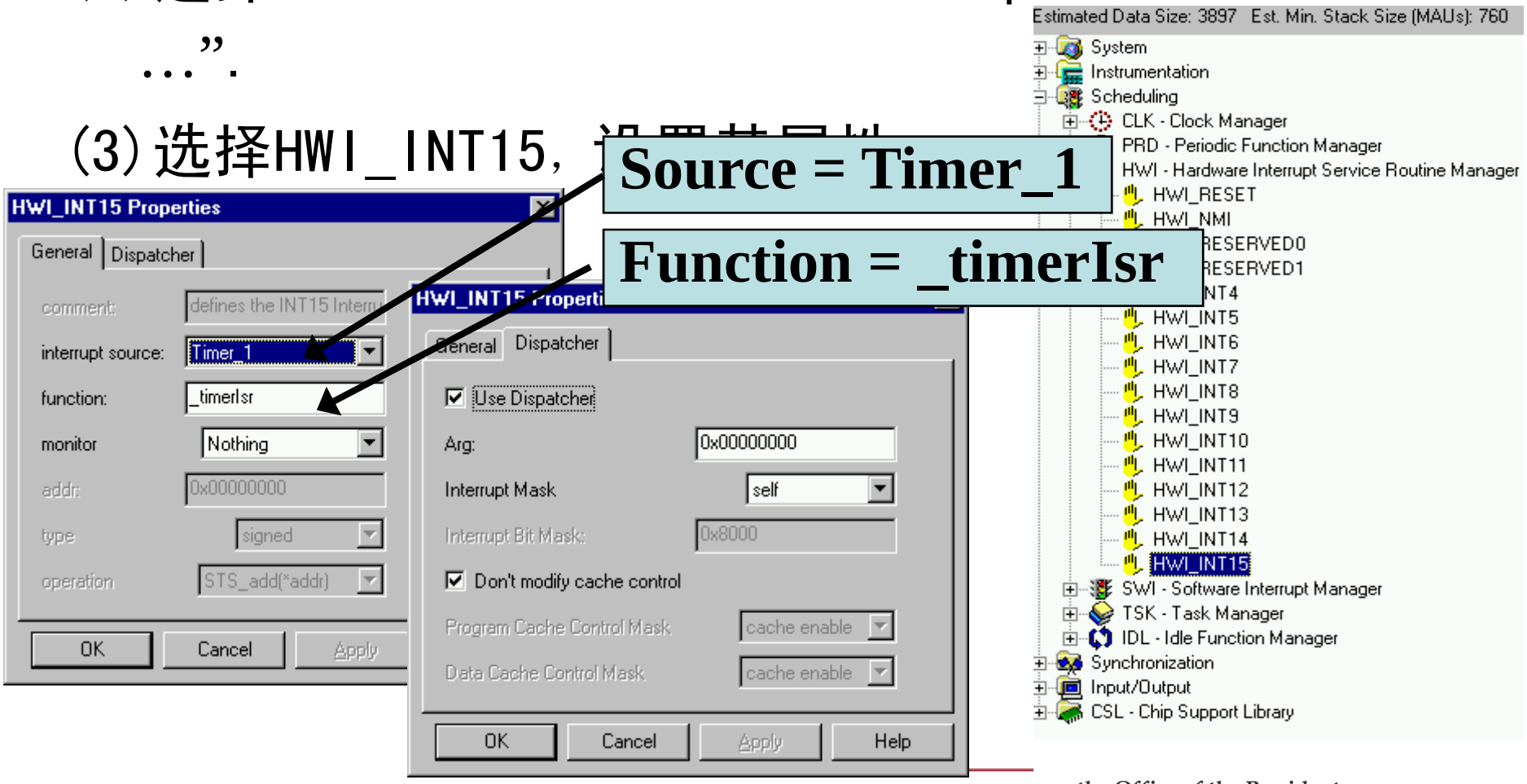


（2）设置硬件中断

(1)打开tcf文件

(2)选择“HWI - Hardware Interrupt Service ...”.

(3)选择HWI_INT15，设置其属性



（2）设置硬件中断

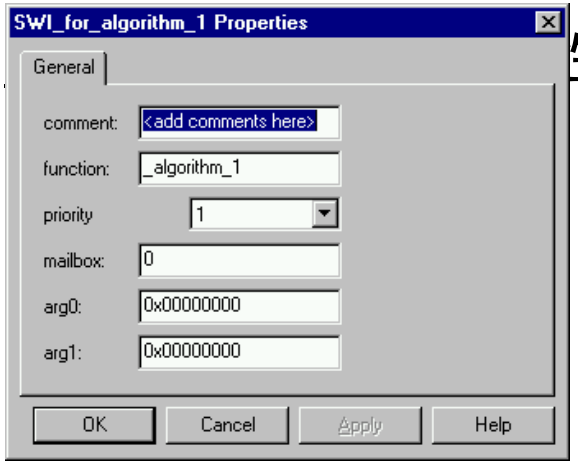
(4)在源程序中用C语言编写中断服务程序

```
void timerIsr (void)
{
/* Put your code here */
}
```

（3）创建一个软件中断

(1)打开tcf文件，选择“SWI – Software Interrupt Manager”创建一个名为“SWI_for_algorithm_1”的SWI对象

(2)更改“SWI_for_algorithm_1”的属性为：



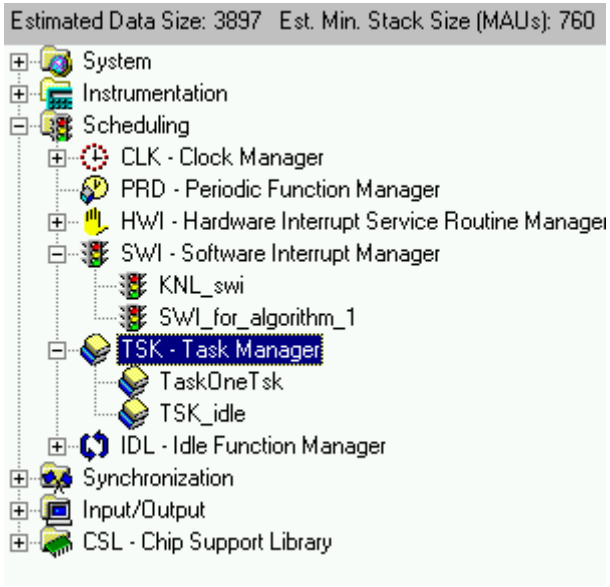
（3）创建一个软件中断

(3)在源程序中用C语言编写中断服务程序

```
void algorithm_1 (void)
{
/* Put your code here */
}
```

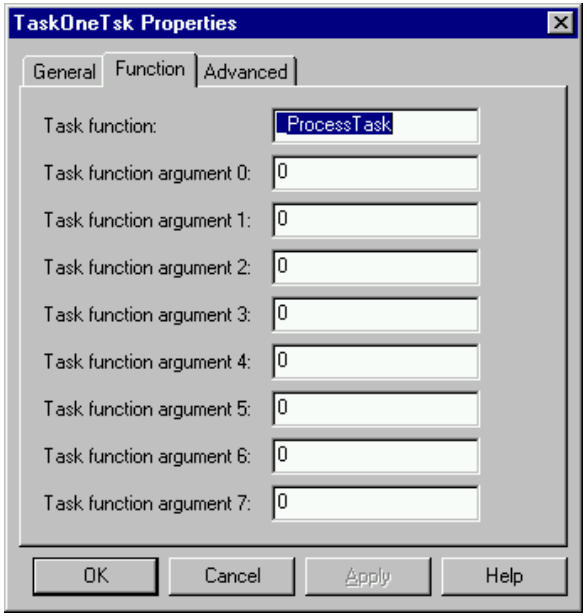
（4）创建一个任务

(1)打开tcf文件，选择“TSK - Task Manager”，创建一个名为“TaskOneTsk”的新的TSK对象。



（4）创建一个任务

(2)更改属性为：

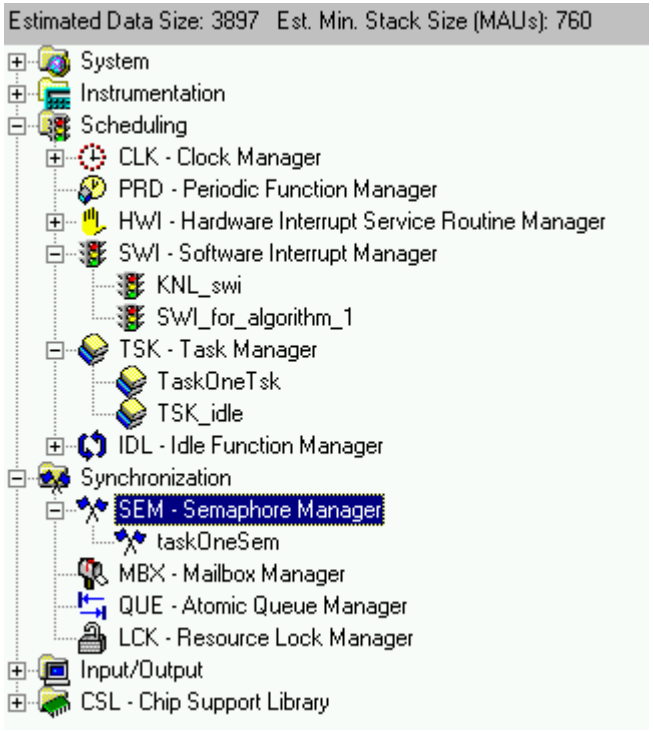


(3)在源程序中用C语言编写中断服务程序

```
void ProcessTask (void)
{
/* Put your algorithm here */
}
```

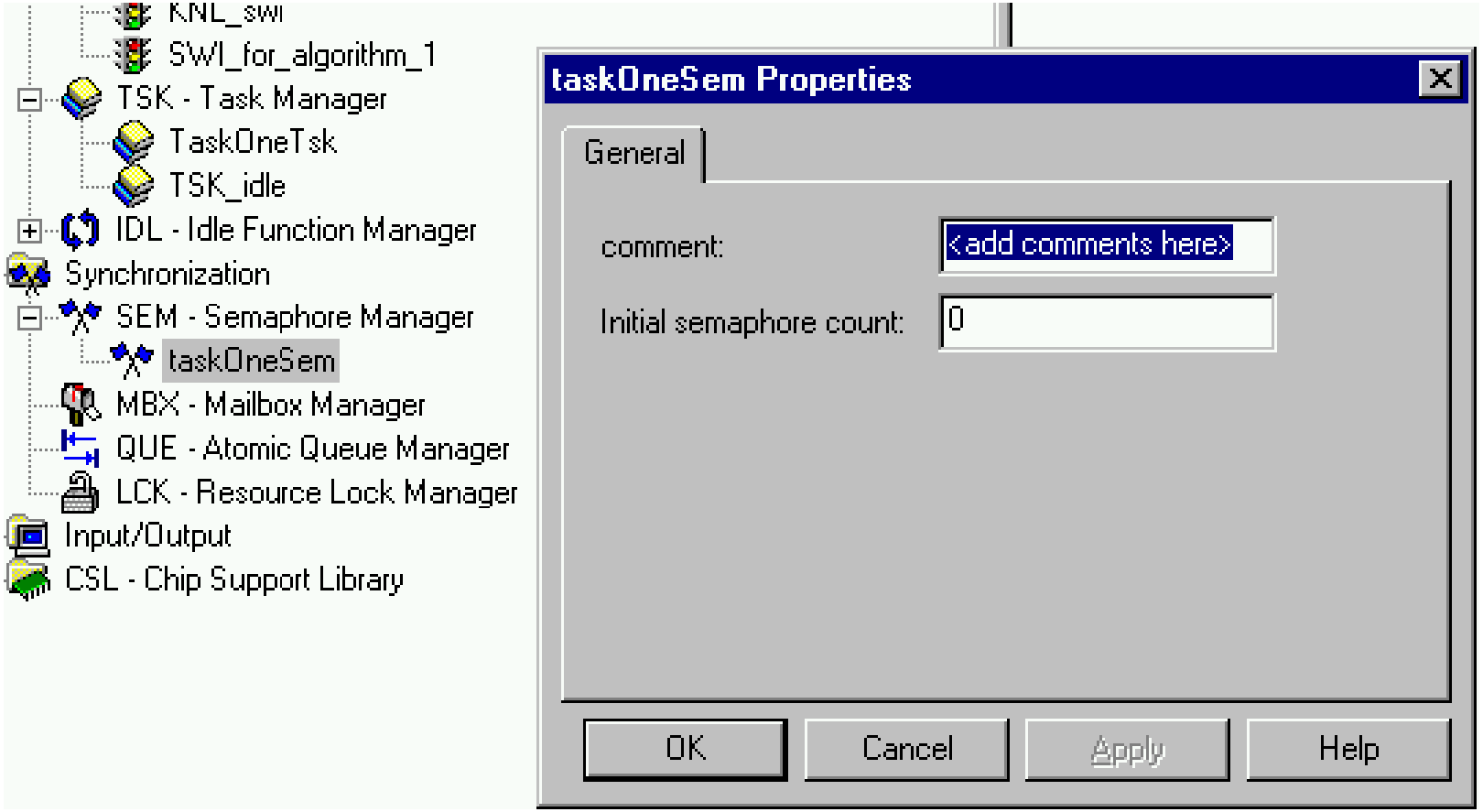
（5）创建一个信号量

(1)打开tcf文件，选择 “SEM - Semaphore Manager”，创建一个名为 “taskOneSem”的信号量对象.



（5）创建一个信号量

(2) 更改属性为：



触发软件中断和任务

(1) 软件中断：在源程序中写入以下代码以触发软件中段

```
SWI_post (&SWI_for_algorithm_1);
```

(2) 任务：在源程序中写入以下代码将任务放入准备队列

```
SEM_post (&taskOneSem);
```

关于任务

- 一个任务可以被挂起，直到SWI执行完。
- 任务通常是无限循环的，执行期间会检测信号量。
- 任务可以抢占自己。如：

```
void ProcessTask (void)
{
    while (1)
    {
        SEM_pend (&taskOneSem, SYS_FOREVER);
        /* Insert your code here */
    }
}
```

源代码

```
void main (void)
{
      /* Put all your setup code here */
      return; /*DSP BIOS starts after the return */
}

/* Hardware Interrupt */
void timerIsr (void)
{
      /* Put your code here */
      SWI_post (&SWI_for_algorithm_1);
      SEM_post (&taskOneSem);
}

/*Software Interrupt */
void algorithm_1 (void)
{
      /* Put your code here */
}

/* Task */
void ProcessTask (void)
{
      while (1)
      {
            SEM_pend (&taskOneSem, SYS_FOREVER);
            /* Insert your code here */
      }
}
```

DSP/BIOS实时分析

介绍

- 传统的分析调试方法是让程序运行停止，然后检查变量和存储器；
- 这种调试方法在实时系统中是无效的，因为实时系统是以程序的连续性、不确定运行和严格的时序要求为特征的；
- DSP/BIOS仪表类API函数和DSP/BIOS插件是对传统调试工具的补充，可以用来监视程序的实时运行；
- 那么在不停下程序的执行的情况下是怎样实现数据的监测的呢？

介绍

1） 目标板与主机之间的通信是在后台线程（IDL）中执行的（如：CPU执行NOP操作或等待中断）。后台线程具有最低优先级，因此数据的通信不会影响应用系统的实时性能。

2） 数据操作是由主机方处理的，因此可以释放CPU去执行更有用的任务。

3） 每一个STS对象在数据存储器中只占用4个字的空间，也就是主机只需要传送4个字就可以从一个统计对象中上载数据。

介绍

4） LOG,STS和TRC模块执行速度很快，执行时间是个常量。 如下:

- LOG_printf and LOG_event: 大约 32 cycles
- STS_add: 大约 18 cycles
- STS_delta: 大约 21 cycles
- TRC_enable and TRC_disable: 大约 6 cycles

实时分析功能模块-LOG模块

- LOG（Event Log Manager）日志对象实时捕获事件信息。系统事件通过系统日志捕获，在配置工具下也可以创建其他日志，在程序中还可以向任何日志添加信息。
- LOG模块中的函数:

(1)LOG_disable( ):禁用系统日志;

(2)LOG_enable( ):启用系统日志;

(3)LOG_error( ): 写一个用户错误事件到系统日志中;

(4)LOG_event( ): 追加未格式化的消息到消息日志;

(5)LOG_message( ):写一个用户信息到系统日志中;

(6)LOG_printf( ):追加一个格式化的信息到信息日志中;

(7)LOG_reset( ): 重置系统日志;

例子- LOG_printf

- Printf（）函数大约需要多少时钟周期？

  >3400（个）

- 而LOG_printf（）呢？

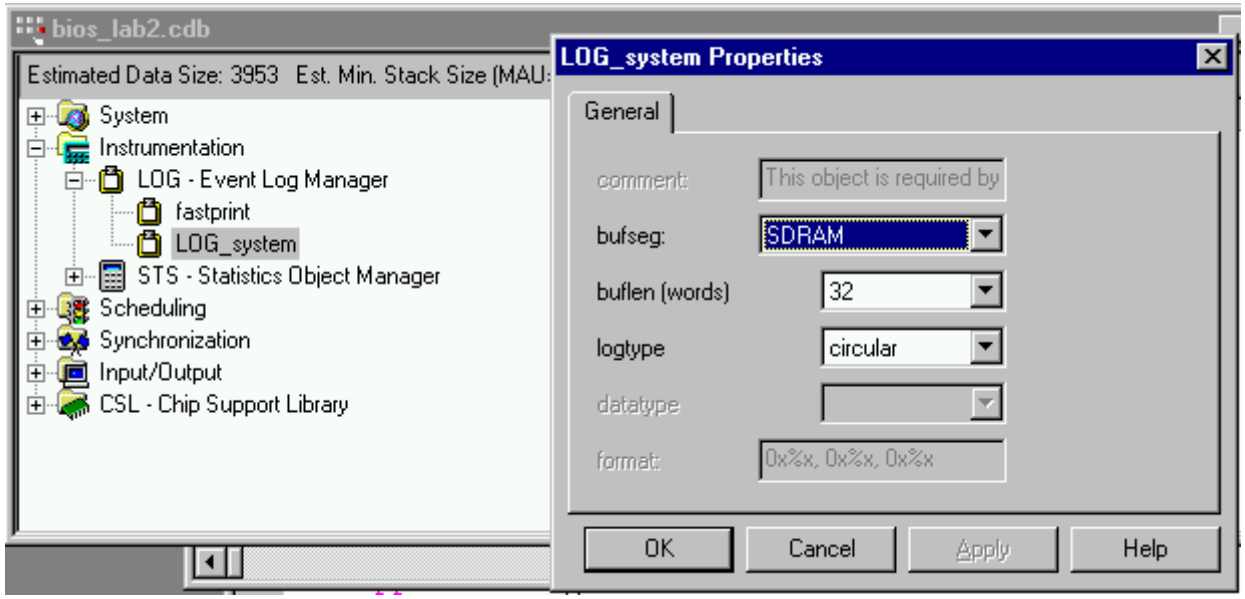
  大约60（个）

例子- LOG_printf

1）利用配置工具创建一个LOG对象：

(a)打开tcf文件，选择instrumentation，打开“LOG - Event Log Manager”.

(b)创建一个新对象，命名为“fastprint”，修改其属性：



例子- LOG_printf

2）使用LOG_printf函数时在源程序中添加以下代码：

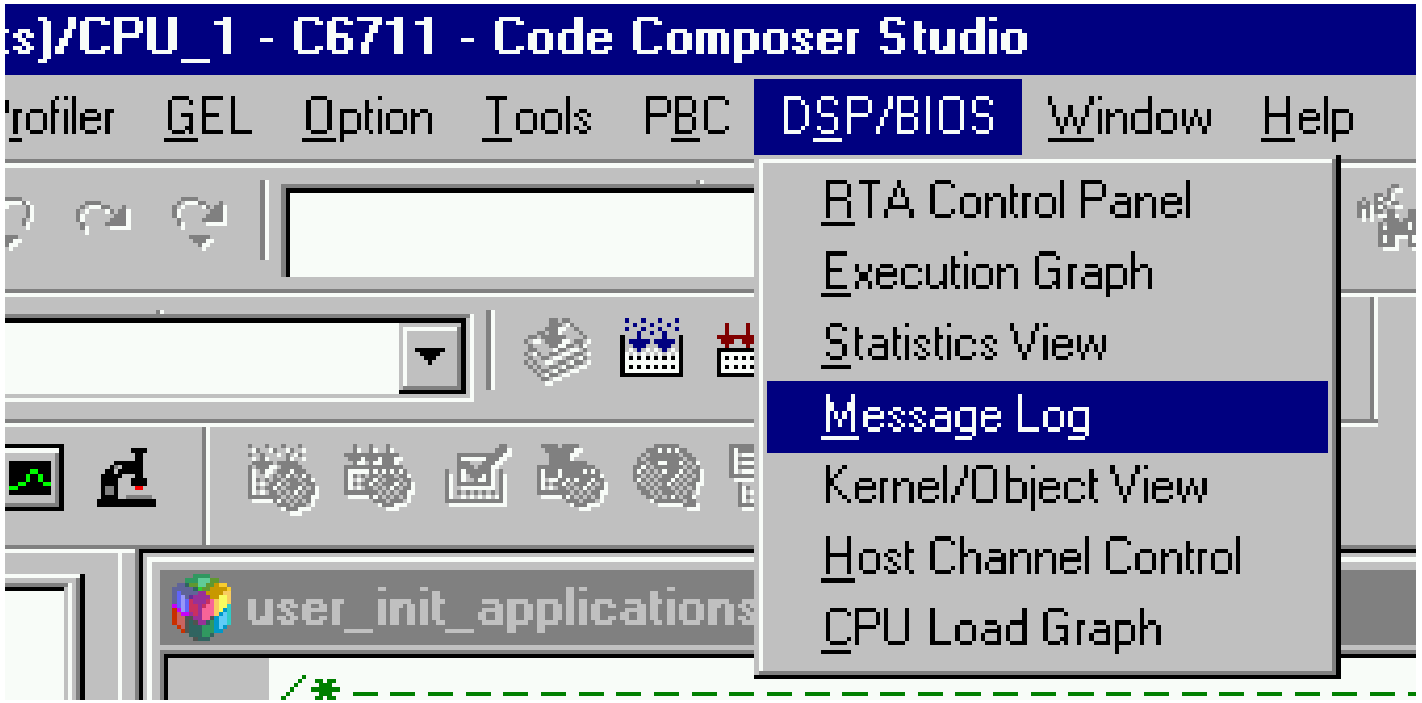
```
/* #include <stdio.h>   NOT required */
#include <std.h> /* this is required by all DSP/BIOS modules */
#include <log.h> /* this is required by the LOG module */

extern far LOG_Obj fastprint;

void algorithm_1 (void)
{
     LOG_printf (&fastprint, “Algorithm 1 is running\n”);
}
```

例子- LOG_printf

3）打开Message Log 窗口，观察输出：



实时分析功能模块-STS模块

- STS（Statistics Object Manager）统计对象实时捕获任何变量的计数值（Count）、最大值（Maximum）及总和值（Total）。有关SWI（软件中断）、PRD（周期函数）、HWI（硬件中断）及PIP（管道）对象的统计数据将被自动捕获。同时，在程序中还可以创建统计对象来捕获其他统计数据。
- STS 中的函数:

  (1)STS_add( ): 使用提供好值来上载数据;

  (2)STS_delta( ): 使用设定值与提供值间的差来上载数据;

  (3)STS_reset( ):重置STS对象中的数值;

  (4)STS_set( ):保存设定值;

总结

- DSP/BIOS是一个可升级的实时内核。它主要是为需要实时调度和同步以及主机-目标系统通讯和实时监测（Instrumentation）的应用而设计的。
- DSP/BIOS 操作系统，包括:
  1 代码尺寸小的实时库.
  2 调用API函数，使用实时库的服务
  3 易用的配置工具
  4 含有实时分析的程序
- DSP/BIOS 的调度包含:
  1 固定优先级的抢先式调度器
  2 多种线程类型

TI DSP培训以及技术服务简介

上海交大BME-美国德州仪器联合DSP实验室成立于2007年，是国内最权威的TI技术服务于培训机构。实验室有TI（C6000，C2000，C5000，达芬奇，多核DSP）全系列开发平台，提供DSP ，MSP430等技术培训与技术服务，项目合作等。培训内容有

1）CCS开发环境精解与实例；

2）DSP/SYS BIOS 实例；

3）C6000/C5000/C2000全系列DSP架构以及汇编，C语言，混合编程等；

4）HPI，EMIF，EDMA，Timer等外设；

5）C6416、DM642， C6678多核EVM开发平台实例；

6）Bootloader 原理以及实例等。

常年开班，三人以上集体报名8折优惠，学生5折。

联系电话：13651621236（牛老师），颁发TI授权证书

邮件报名：jhniu@sjtu.edu.cn ，niujinhai@yahoo.com.cn

颁发TI授权的培训证书

DSP实验室介绍

- 美国德州仪器（TI）—上海交通大学（SJTU）联合DSP实验室成立于2007年10月，位于上海交大闵行校区，致力于TI DSP技术的推广，以及相关数字信号处理算法的研究与开发，为客户提供优质的产品与服务，涉及的技术领域有，无线通信，音频/视频信号处理，医学信号/图像处理，数字马达控制等。实验室研发与培训教师主要由上海交通大学青年教师承担，同时聘请了多位有企业工作背景的DSP技术专家为实验室的顾问。

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

DSP实验室介绍

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

End

Thanks