

LCDK6748 DSP LAB指导材料

杜伟韬 duweitao@cuc.edu.cn
广播电视数字化工程中心 ECDAV

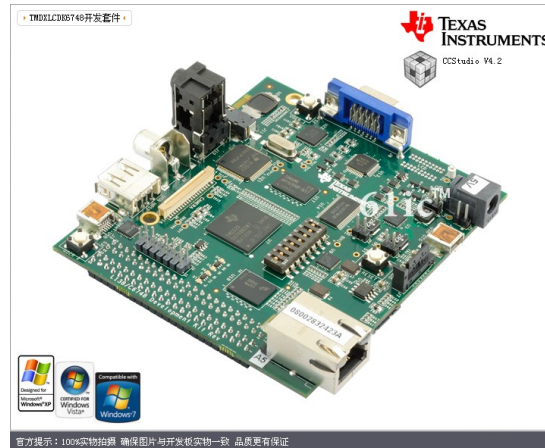
杨刚 gangy@cuc.edu.cn
信息工程学院 电子工程系

中国传媒大学 Communication University of China

LCDK6748 DSP LAB

Part III

指导材料



杜伟韬 duweitao@cuc.edu.cn
广播电视数字化工程中心 ECDAV

杨刚 gangy@cuc.edu.cn
信息工程学院 电子工程系

中国传媒大学 Communication University of China

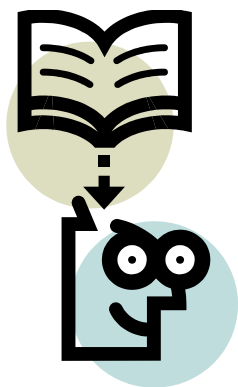
本课程实验内容

- Hello World on DSP
- LED 系列实验
 - 点灯实验-直接配置寄存器
 - 点灯实验-使用GEL函数
 - 点灯实验-使用StarterWare库函数
 - 通过按键中断控制走马灯实验-使用StarterWare库函数
- 音频实时采集回放实验：使用中断、I2C控制器、McASP控制器、AIC31 Codec、DMA控制器和StarterWare库函数
- 存储器布局和访存性能测试实验：使用片内RAM，片外DDR、malloc函数、定时器，观察MAP文件，本实验基于音频采集回放实验
- 使用FIR滤波器的实时音频均衡器实验，本实验基于音频采集回放实验
- DSP/BIOS: Hello World 和系统时间打印实验
- DSP/BIOS: 线程任务（TSK）和信号量（SEM）实验
- DSP/BIOS: 按键中断和点灯实验：使用BIOS中断调度器
- DSP/BIOS: 音频实时采集、回放实验：使用BIOS中断调度器

LAB1

dsp/bios

LOG_printf & CLK



背景知识

关于处理器和操作系统

- 处理器的寻址模式
 - 实模式 vs 保护模式, 物理地址 vs 虚拟地址
- DSP处理器—主要用于计算任务
 - 工作在实模式下
 - 没有 内存管理单元
 - 所有任务均可以直接访问物理地址
- 操作系统
 - 支持内存管理的通用系统 Windows , Linux等
 - 无内存管理的嵌入式系统 ucos , dsp/bios等

为什么需要操作系统

- 任务管理—隔离与切换调度
- 应用层的软件重用
 - 操作系统和驱动程序屏蔽硬件细节
- 便于调试
 - 16进制的RAM View不如ASCII友好
- 操作系统的局限性
 - 标准操作系统：过多层次的抽象导致驱动软件开发工作量较大，中断的响应速度较慢。
 - 实模式操作系统：任务隔离和保护性不好，单个进程出错容易造成系统崩溃。

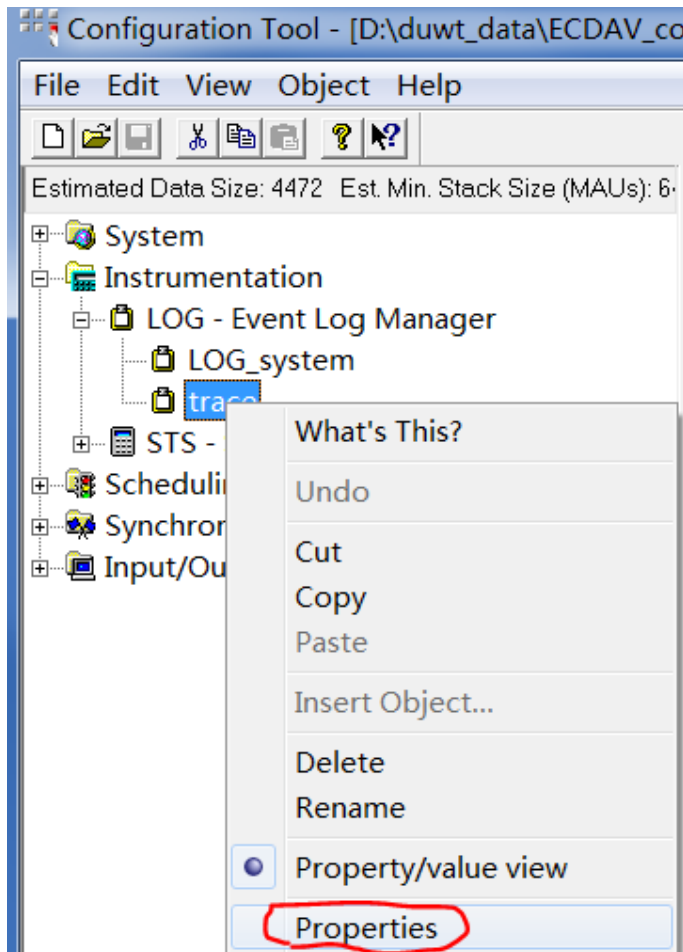
LOG_printf 函数

- DSP/BIOS提供的实时打印函数
- 系统资源开销较小
- 可以向内存打印实时数据
- 将处理器暂停后，CCS将DSP内存中的数据导入到开发机观察。
- 向指定的信息内存区域打印
- 信息内存填充模式有2种
 - Fixed模式，填满之后不再继续填充内存区域
 - Circular模式，回绕到内存首地址继续填充内存区域

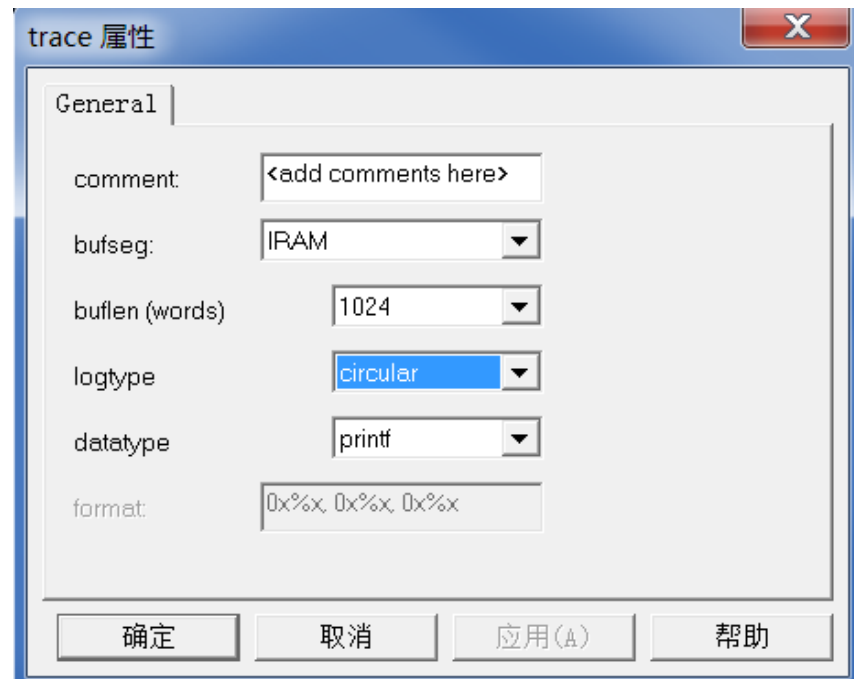


上机操作

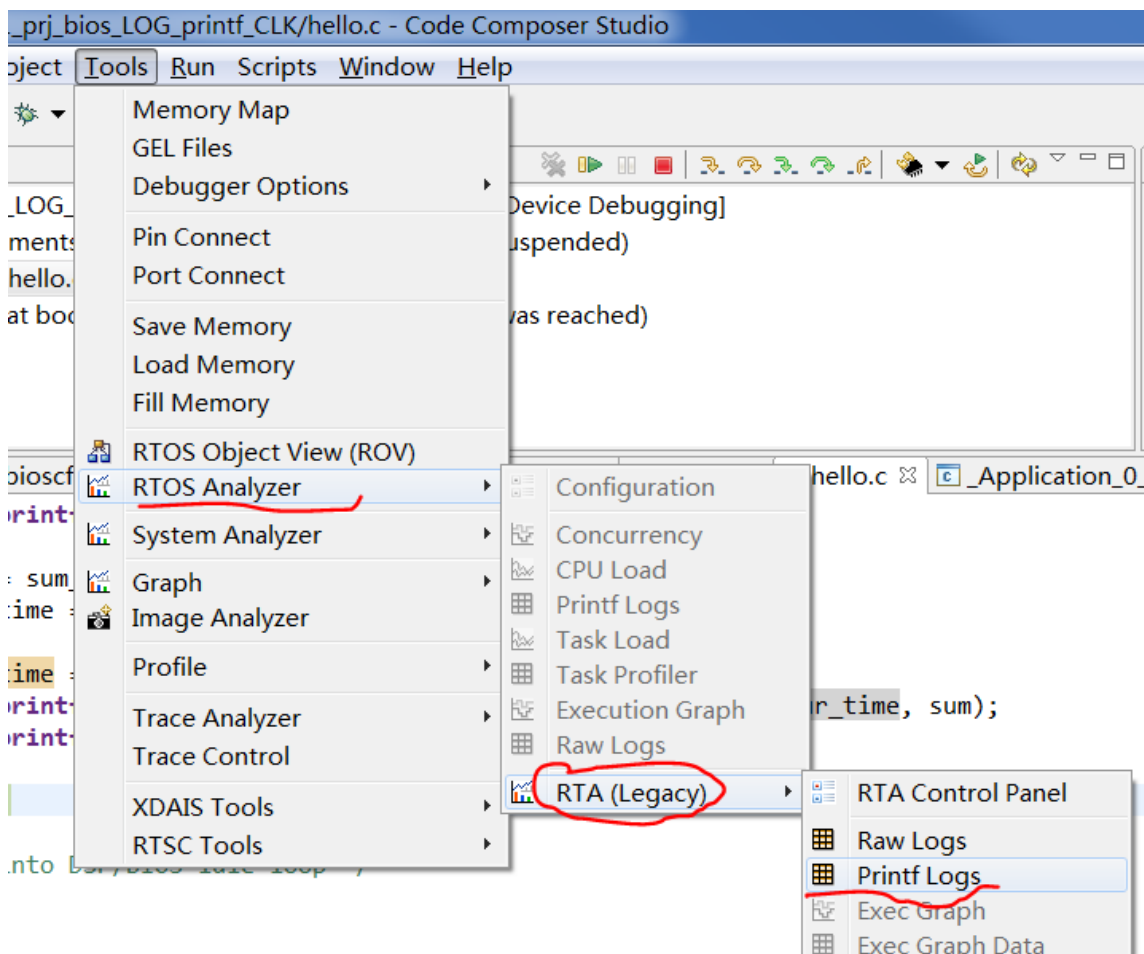
选中并切换到LAB1项目



- 在edit模式下找到hello.tcf。双击打开
- 找到 trace 对象，右键-设定属性
- 注意观察缓冲区模式 logtype
 - 即前文所述的循环填充和固定填充模式



编译项目，连接仿真器到LCDK并下载代码



- 运行代码，约1秒钟后暂停处理器
- 然后打开printf的日志窗口
- 本项目代码功能简介
- 本代码用于计算递增数列的循环累加和
- 在每次开始计算之前和之后，读取处理器定时器的计数值。
- 然后得到用于计算的周期数
- 请自行阅读代码理解以上过程

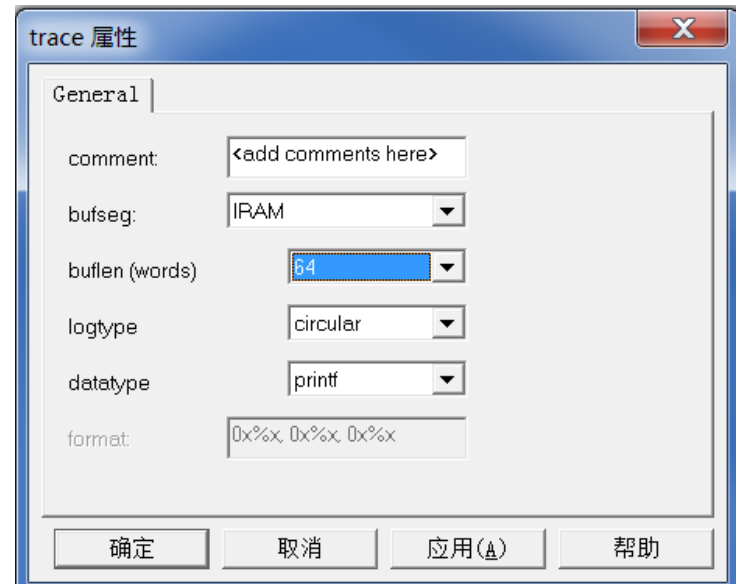
观察LOG_printf日志窗口的输出结果

ti...	seqID	formattedMsg	logger
0	0	Start Sum Loop	trace
1	1	Loop 0	trace
2	2	Sum start at time 0x9f38e11	trace
3	3	Sum Done at Time 0x9f38e97, sum val is 0	trace
4	4	Loop time is 134	trace
5	5	Loop 1	trace
6	6	Sum start at time 0x9f38ef5	trace
7	7	Sum Done at Time 0x9f38f90, sum val is 1	trace
8	8	Loop time is 155	trace
9	9	Loop 2	trace
10	10	Sum start at time 0x9f38fee	trace
11	11	Sum Done at Time 0x9f3909e, sum val is 3	trace

- 观察每次打印的信息
- 请统计Loop Time的变化规律，然后请思考
 - 全部循环次数的打印信息都收集全了么？
 - 有什么证据支持你做出以上结论？
 - Loop Time的变化有什么规律么？
 - 为什么有这种规律呢？

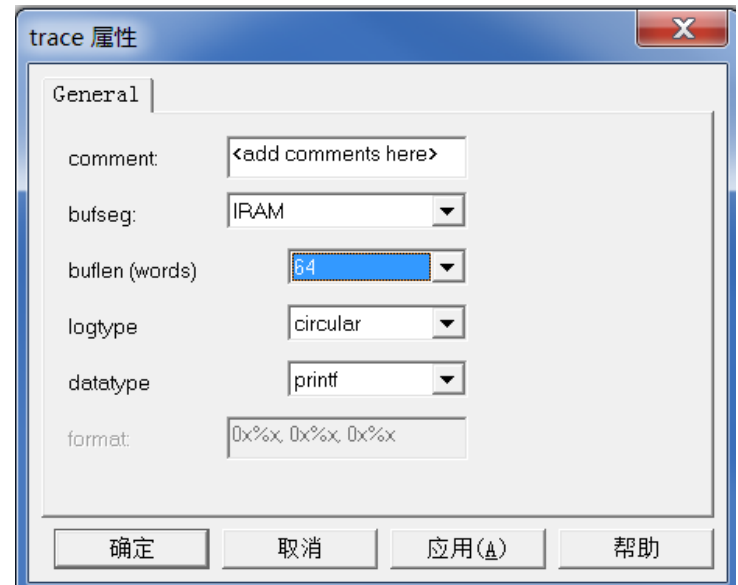
LAB1 作业

- 修改hello.tcf中，trace对象的属性
- 把buflen这一项改为64
- 缓冲区模式为循环填充
- 然后保存hello.tcf
- 然后重新编译项目
- 下载，运行，暂停
- 观察LOG_printf 的输出信息
- 请问，此时你能观察到全部的程序打印信息么？为什么？
- 那么你观察到的打印信息是哪一部分么？为什么



LAB1 作业 (续1)

- 修改hello.tcf中，trace对象的属性
- 把buflen这一项改为64
- 缓冲区模式为 **固定填充**
- 然后保存hello.tcf
- 然后重新编译项目
- 下载，运行，暂停
- 观察LOG_printf 的输出信息
- 请问，此时你能观察到全部的程序打印信息么？为什么？
- 那么你观察到的打印信息是哪一部分么？为什么



LAB1 作业 (续2)

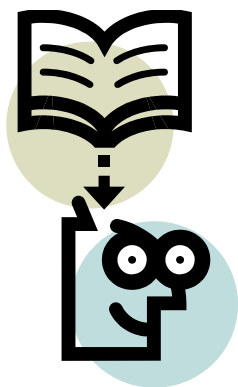
- 请思考和总结一下
- trace对象的把buflen参数，以及缓冲区模式参数的设定，对我们调试程序又怎样的影响，对系统的资源开销有什么影响。

LAB2

线程任务 (TSK)

AND

信号量 (SEM)



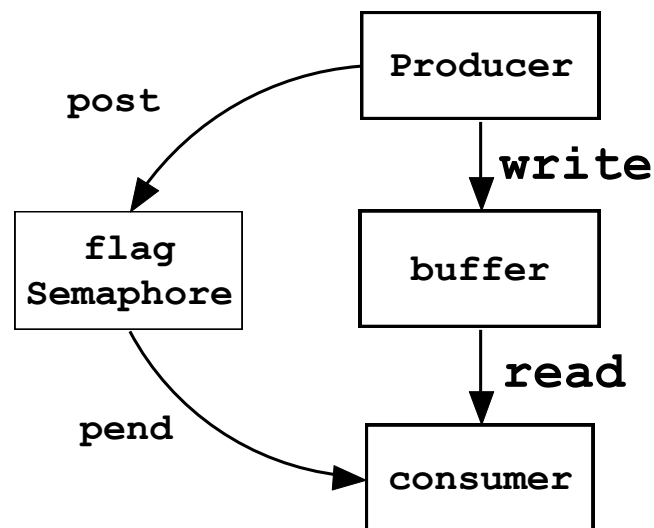
背景知识

操作系统的进程和线程

- 进程：一种重量级的多任务模型
 - 最早见于多用户终端的分时系统
 - 每个独立运行的任务隔离程度较高
 - 进程独立享有一个完整的虚拟地址空间
 - 需要通过系统调用互相传递信息
- 线程：一种轻量级的多任务模型
 - 一个进程可以启动多个线程
 - 线程之间共享内存
 - 线程可以通过系统调用进行同步或是互斥
 - 也可以自行设计同步策略（需要小心）

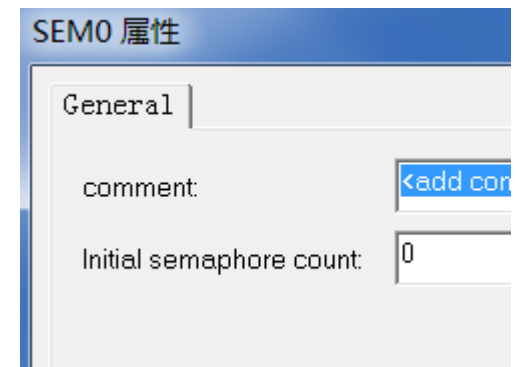
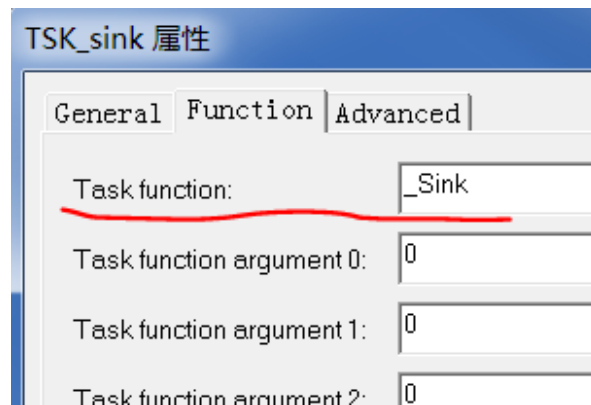
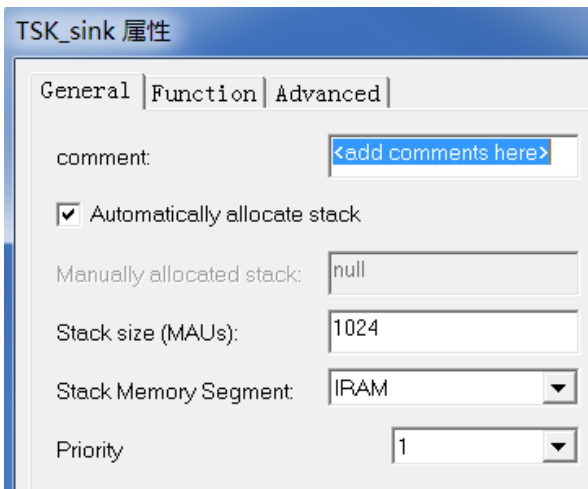
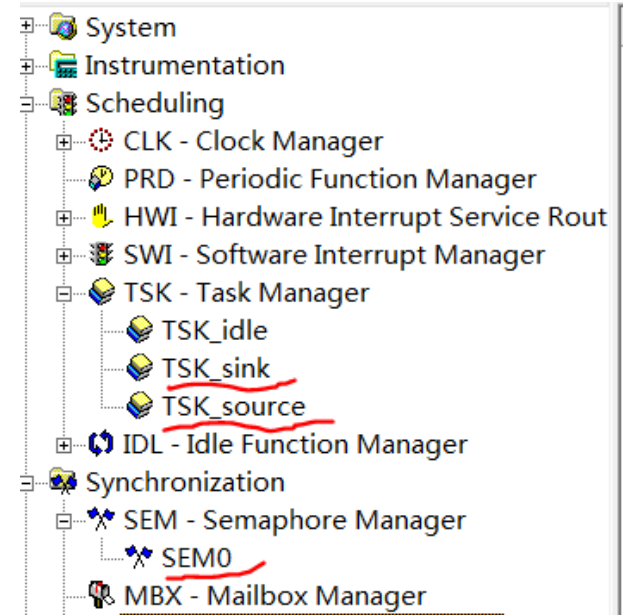
信号量

- 一种用于标识当前资源状态的方法
- 共享内存的互斥模型：
 - 生产者：写入数据
 - 消费者：读取数据
 - 信号量：标识缓冲区的使用状态
 - Post操作：提升信号量，状态值加1
 - Pend操作：获取信号量，若成功则信号量减1，若信号量为0，则线程阻塞
- 使用信号量进行线程同步
 - 生产者：先写缓冲区，再post信号量，然后自行阻塞（yield）
 - 消费者：先pend信号量，成功后读取数据进行处理。



DSP/BIOS的多任务支持

- 在main函数return之后，bios的调度器启动
- TCF管理器中可以创建TSK和SEM
- TSK（任务）的设定参数
 - 堆栈的尺寸和物理内存位置
 - 任务对应的C函数名
- SEM（信号量）的设定参数
 - 初始值



本实验的多任务模型

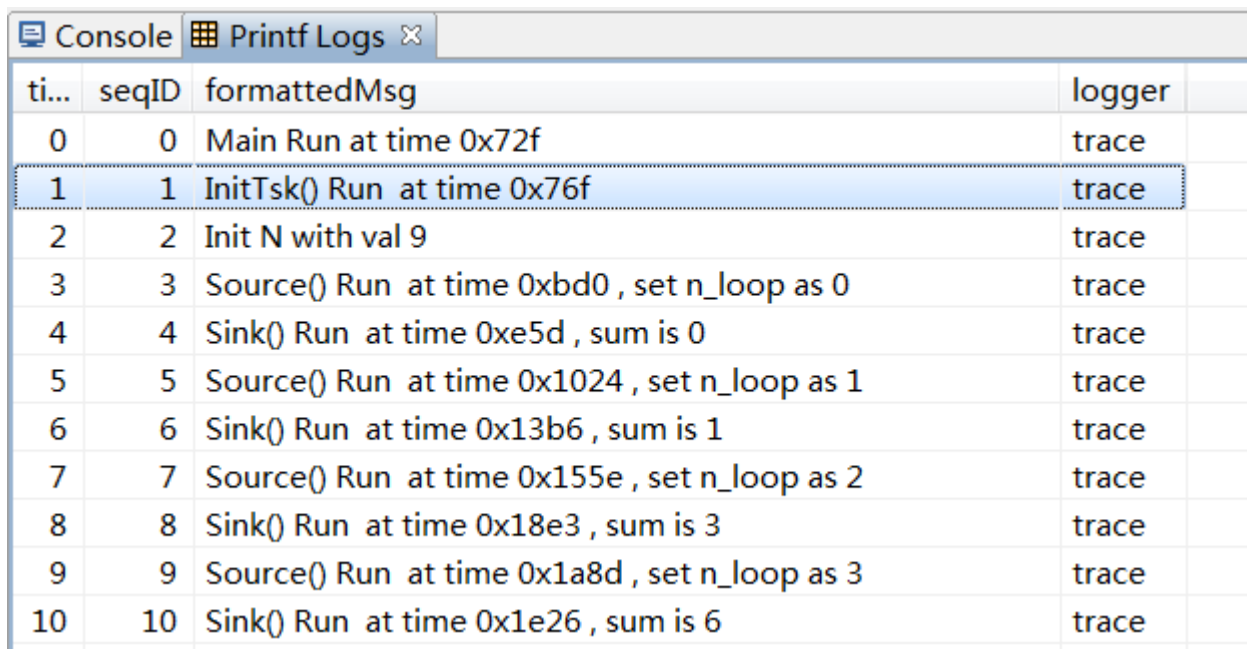
- 本实验的2个线程共享的数据为一个全局变量
 - 该变量是一个计算参数
- 生产者线程 Source
 - 修改计算参数
 - Post信号量
 - 自行阻塞
- 消费者线程 Sink
 - Pend信号量 (可能被阻塞)
 - 读取参数
 - 完成计算任务



上机操作

选中并切换到LAB2项目

- 下载并运行代码
- 暂停处理器，观察LOG_printf 的输出
- 思考并理解进程调度的过程



The screenshot shows a debugger window titled "Printf Logs" with a table of log entries. The table has columns for time (ti...), sequence ID (seqID), formatted message (formattedMsg), and logger. The entries show a sequence of operations from a main thread (seqID 0) and a task (seqID 1). The task performs a loop of source and sink operations, with the sum increasing from 0 to 6.

ti...	seqID	formattedMsg	logger
0	0	Main Run at time 0x72f	trace
1	1	InitTsk() Run at time 0x76f	trace
2	2	Init N with val 9	trace
3	3	Source() Run at time 0xbd0 , set n_loop as 0	trace
4	4	Sink() Run at time 0xe5d , sum is 0	trace
5	5	Source() Run at time 0x1024 , set n_loop as 1	trace
6	6	Sink() Run at time 0x13b6 , sum is 1	trace
7	7	Source() Run at time 0x155e , set n_loop as 2	trace
8	8	Sink() Run at time 0x18e3 , sum is 3	trace
9	9	Source() Run at time 0x1a8d , set n_loop as 3	trace
10	10	Sink() Run at time 0x1e26 , sum is 6	trace

LAB2 作业

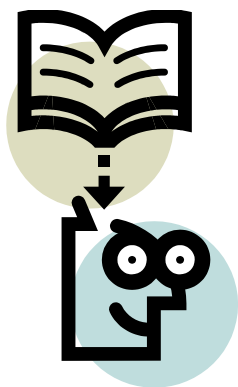
- 按照以下需求修改代码
- 创建2个信号量，semA和semB
- 创建2个计算任务线程，sinkA和sinkB
- Source任务在生成计算参数时
 - 利用信号量操作
 - 奇数号参数，启动sinkA任务完成计算
 - 偶数号参数，启动sinkB任务完成计算
- 通过LOG_printf打印观察任务的切换过程和计算结果

LAB3

DSP/BIOS HWI中断

和

GPIO / LED



背景知识

利用DSP/BIOS响应中断

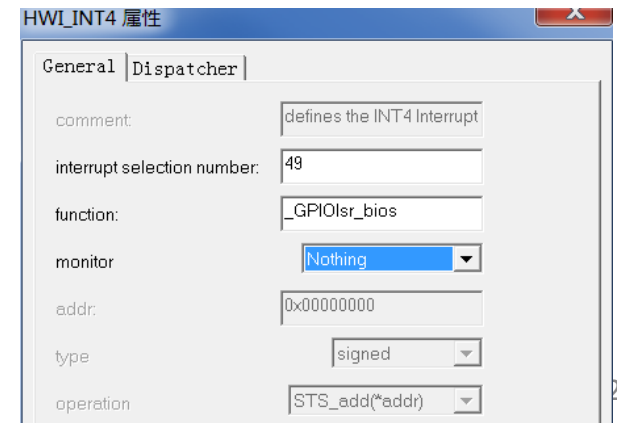
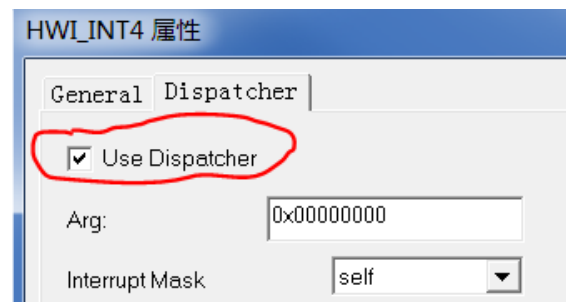
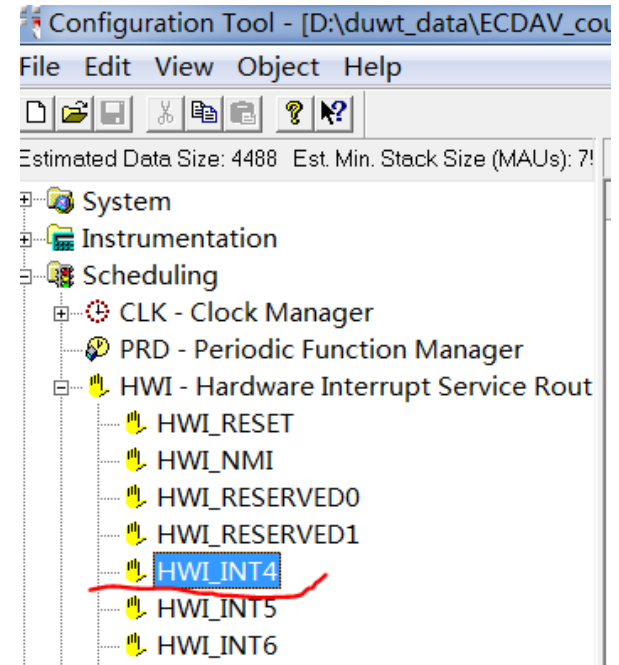
- 先行说明
 - 本实验内容基本与之前“StarterWare的GPIO按键中断”实验内容类似
 - 区别在于，中断函数的分配使用bios进行
- 使用TCF管理器
 - 选择指定的中断号，设定其属性
 - 填写中断所选择的事件号
 - 填写中断处理函数，注意下划线前缀
 - 另外，需要选中“使用中断分配器”

另外还需要在初始化函数中包含以下头文件

```
#include <c62.h>
```

在系统初始化时，执行以下代码：

```
C62_enableIER (C62_EINT4);
```

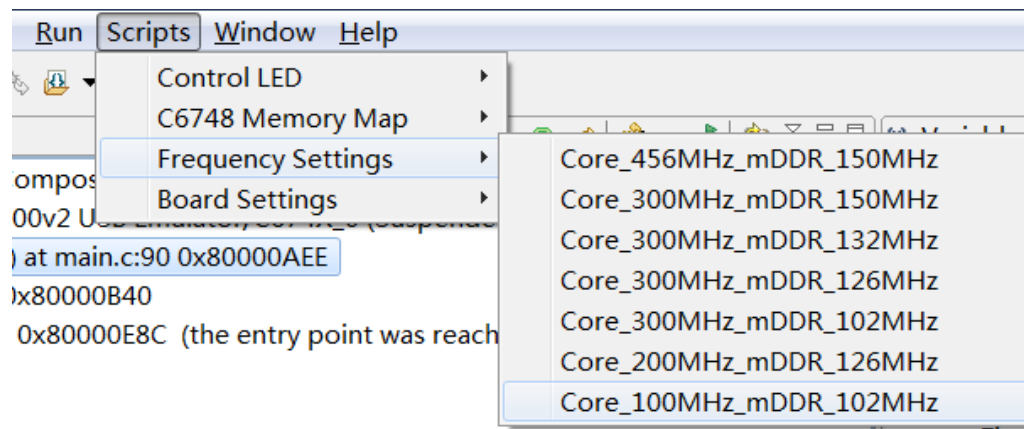




上机操作

切换active project 至LAB3

- 编译并Debug该项目
- 运行代码，观察LED闪烁情况。
- 按下按键 USER1（图中红圈）
- 观察LED闪烁情况的变化
- 更改处理器主频再次观察按键前后的LED闪烁

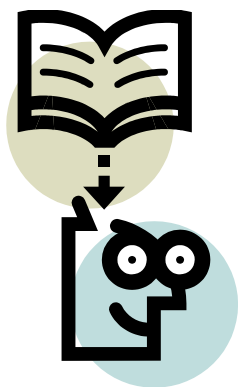


LAB3作业

- 本实验作业和之前的Part I - LAB5作业一致。
- 请在DSP/BIOS的环境中重新完成一遍。

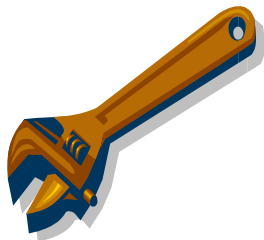
LAB4

DSP/BIOS 和 音频滤波器



背景知识

本实验和PART II -LAB2 知识结构类似，区别在于使用DSP/BIOS调度和分配中断



上机操作

&

LAB作业

和Part II-LAB2一致

请重新做一次

注意观察 LOG_printf 的打印

*That's ALL
Happy DSP Life*