第一部分

DSP/BIOS 用户手册

绪　　言

关于本手册

本手册编译自 TI DSP/BIOS 文档“TMS320 DSP/BIOS User's Guide,SPRU423F,Nov. 2004”。

DSP/BIOS 使得用户能够在 TI 的 TMS320 DSP 器件上开发出实时的嵌入式软件。DSP/BIOS 提供了一个很小的固件(firmware)实时库及一组易于使用的实时追踪和分析工具。

用户应该阅读并熟悉相应 DSP 平台的 TMS320 DSP/BIOS API 函数参考手册,其中包含了 DSP/BIOS 中所有 API 函数的详细说明,可以更好地理解本手册。

在阅读本手册之前,建议用户先从 CCS 的在线指南中有关 DSP/BIOS 的部分学习 DSP/BIOS 的大致使用方法。在 CCS 中选择菜单“Help Tutorial”即可打开 CCS 在线指南。本手册较深入地讨论了 DSP/BIOS 的多个方面,需要用户至少对 DSP/BIOS 的使用方法有一个基本的理解,所以使用帮助系统是非常必要的。

符号约定

本手册使用下列约定:

- 程序列表、程序示例和交互显示都使用一种特殊的字体显示。

这里给出一个程序列表的例子:

```
Void copy(HST_Obj * input, HST_Obj * output)
{
    PIP_Obj * in, * out;
    Uns * src, * dst;
    Uns size;
}
```

- 方括号“[”和“]”用来标识一个可选的参数,用户在使用一个可选参数时,指定的是方括号中的信息。
- 数字 54 用来代表用户使用的 DSP 平台,如果用户的 DSP/BIOS 平台是基于 C62x 的,在合适的地方将 54 替换为 62 即可。例如 C6000 DSP/BIOS 汇编语言 API 头文件的后缀名为 .h62,对 C2800 平台,则应为 .h28。对 C64x、C55x 或 C28x DSP 平台,这时使用 64、55 或 28 代替 54 即可。
- 一些特定器件所特有的信息使用下列图标来标识:

第 1 章　DSP/BIOS 概述

DSP/BIOS 是一个尺寸可伸缩的实时内核，它是为那些需要实时线程调度与同步、主机与目标 DSP 间通信或实时监测的应用而设计的。DSP/BIOS 提供了抢占式多线程、硬件抽象、实时分析和配置工具。

1.1　DSP/BIOS 的特色与优点

DSP/BIOS 在设计上采用了以下一些技术来最小化目标 DSP 上的存储器需求和 CPU 开销：

- 所有的 DSP/BIOS 对象都可以在配置工具中静态建立并被绑定到可执行程序中，不仅减小了代码量，还优化了内部数据结构。
- 监测数据（如日志和统计数据）在主机端（而非目标 DSP 端）被格式化处理。
- API 函数被模块化，因此只有那些应用程序用到的 API 函数才会被绑定到可执行程序中。
- 大部分 API 库函数使用汇编语言实现优化，使得其执行所需的指令周期数达到了可能的最小值。
- 目标 DSP 和主机 DSP/BIOS 分析工具之间的通信在后台空闲循环中完成，以确保分析工具不会影响应用程序所要完成的任务。如果 CPU 太忙以致不能执行后台任务时，DSP/BIOS 分析工具会暂时停止从目标 DSP 接收信息直到 CPU 空闲。
- 错误检测所需的存储器和 CPU 开销被限制到最小。但作为交换，也引入了一些 API 函数调用时的约束，用户应参考 API 手册中的详细说明，在开发时满足这些约束。

此外，DSP/BIOS API 还为程序的开发提供多种灵活的选择：

- 一些用于特殊情况下的 DSP/BIOS 对象可以在应用程序中动态建立或删除。应用程序既可以使用动态建立的对象，也可以使用静态建立的对象。
- 提供了多种类型的线程以满足各种情况的需要：硬件中断、软件中断、任务、空闲函数、周期函数等。用户还可以控制线程的优先级和阻塞（blocking）特性。
- 提供了实现线程间通信与同步的数据结构，包括信号灯、邮箱和资源锁。
- 提供了两种 I/O 模型："管道"和"流"，以提供最大的灵活性和输入/输出能力。其中"管道"用于目标 DSP/主机之间的通信以及线程间的简单数据传输。"流"则用于更为复杂的 I/O 操作并且支持设备驱动。
- 提供的低级系统原语可以简化错误处理，简化一般数据结构的建立以及简化存储器使用的管理。

DSP/BIOS API 标准化了对 TI 大部分 DSP 器件的编程，并且提供了强大而且易用的

程序开发工具，这些工具通过以下途径缩短了建立 DSP 应用程序的时间：

- 配置工具可以自动生成代码来静态声明程序中用到的 DSP/BIOS 对象。
- 配置工具可以在应用程序编译链接之前通过校验属性提前检测出错误。
- 使用 Tconf 脚本语言可以进一步增强 DSP/BIOS 配置。用户可使用文本编辑器修改配置脚本，如向其中加入条件分支、循环、命令行参数检测等。
- 在程序运行时自动对 DSP/BIOS 对象信息进行记录和统计，无须额外编程，其他的监测则是根据需要编程实现。
- DSP/BIOS 分析工具提供了对程序行为的实时监测功能。
- DSP/BIOS 提供了一个标准 API，使得 DSP 算法开发者开发出的代码更易于整合到其他应用程序中。
- DSP/BIOS 和 Code Composer Studio IDE 集成在一起，免费且无须任何运行许可证，并由 TI 公司提供全方位的支持。DSP/BIOS 是 TI 的 eXpress DSP™实时软件技术的关键组件。

1.2 DSP/BIOS 组件

图 1-1 给出了在 CCS 程序生成和调试环境中的 DSP/BIOS 组件。

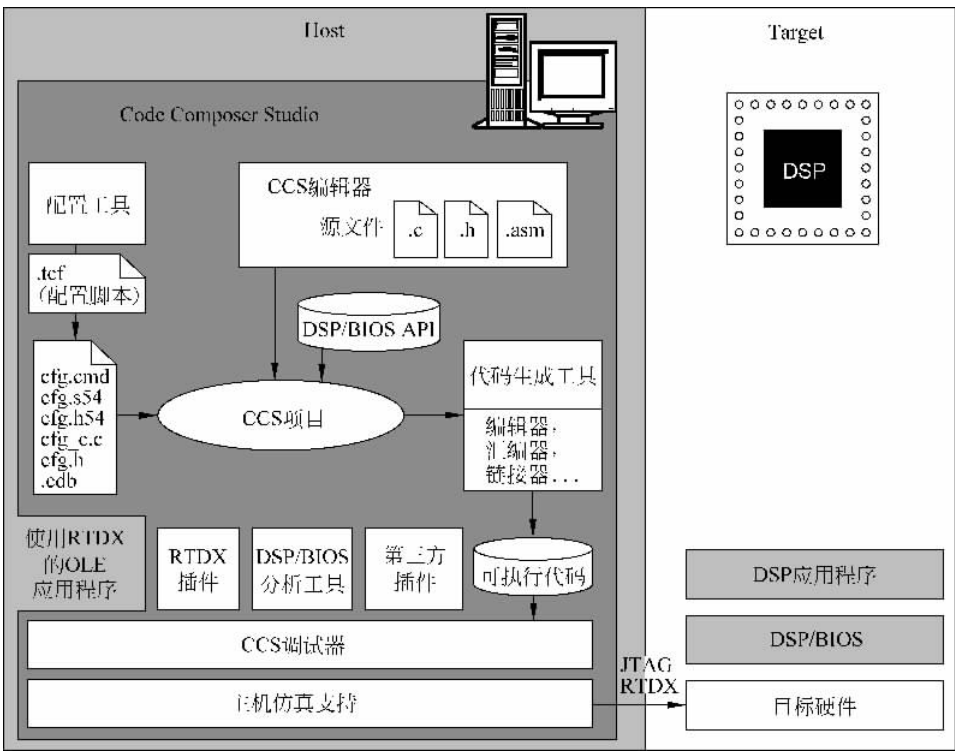


图 1-1 DSP/BIOS 组件

- **DSP/BIOS API**：用户在 PC 端使用 C、C++ 或汇编语言编写调用了 DSP/BIOS API 函数的应用程序。
- **DSP/BIOS 配置**：用户创建一个 DSP/BIOS 配置，定义了程序中要使用的静态对象。并且该配置会生成相应的代码文件，和应用程序一起进行编译链接。
- **DSP/BIOS 分析工具**：CCS 中的分析工具使用户可以测试和分析目标 DSP 上应用程序的运行，包括对 CPU 负荷、日志、线程执行情况的监测等。

下面对 DSP/BIOS 组件做简要介绍。

1.2.1　DSP/BIOS 实时内核和 API

DSP/BIOS API 是以模块划分的，根据应用程序中配置和使用的模块的不同，DSP/BIOS 的代码长度为 500 字到 6500 字不等。一个 DSP/BIOS 模块所有的 API 函数都以其模块名为开头，表 1-1 列出了所有的 DSP/BIOS 模块名。

表 1-1　DSP/BIOS 模块

模 块 名 称	说　　明
ATM	使用汇编语言编写的原子(atomic)函数
BUF	定长缓冲池管理器
C28，C54，C55，C62，C64	目标 DSP 特有函数
CLK	时钟管理器
DEV	设备驱动接口
GBL	全局设置管理器
GIO	通用 I/O 管理器
HOOK	钩子函数管理器
HST	主机通道管理器
HWI	硬件中断管理器
IDL	空闲函数管理器
LCK	资源锁管理器
LOG	事件日志管理器
MBX	邮箱管理器
MEM	存储器管理器
PIP	缓冲管道管理器
PRD	周期函数管理器
PWRM	功率管理器(仅 C55x 具有)
QUE	原子队列管理器
RTDX	实时数据交换设置
SEM	信号灯管理器
SIO	流 I/O 管理器
STS	统计对象管理器
SWI	软件中断管理器
SYS	系统服务管理器
TRC	追踪管理器
TSK	多任务管理器

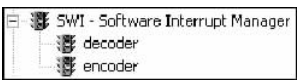
应用程序通过调用 DSP/BIOS API 函数来使用 DSP/BIOS，所有的 DSP/BIOS 模块都提供 C 调用接口，此外一些模块还包含优化过的汇编语言宏。在遵循调用规则的情况下，大多数的 C 调用接口也可以在汇编程序中调用。有一些 C 语言接口实际上是 C 语言宏，所以汇编程序不能使用。关于所有 DSP/BIOS 模块可用的 C 语言和汇编语言接口的描述，请查阅相应平台的 TMS320 DSP/BIOS API 函数参考手册。

1.2.2 DSP/BIOS 配置

DSP/BIOS 配置允许用户静态地创建对象并设置对象属性，从而优化用户应用程序，既改善了程序运行的性能，也缩短了程序开发的进度。

DSP/BIOS 配置对应的源文件是一个 Tconf 脚本，其文件扩展名为.tcf。用户可以通过两种方式得到一个 DSP/BIOS 配置。

- **图形方式**。用户可以使用 DSP/BIOS 配置工具打开并浏览配置脚本。其界面类似于 Windows 的资源管理器，如图 1-2 所示。当使用图形方式来编辑 DSP/BIOS 配置时，用户可以在树型视窗中创建对象，然后在对话框里设置其属性。



- **文本方式**。用户可以使用 CCS 或其他的文本编辑器对配置脚本的内容进行编辑。在这种方式下，用户将使用 JavaScript 语法来对配置编程。

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

一般情况下，这两种方式可以结合起来使用。图形化的编辑器是创建一个初始配置的好方法，而用文本方式编辑脚本则使得用户可以进一步增强配置。

不论使用哪种方式，用户都可以设置 DSP/BIOS 实时库在运行时所使用的一系列参数。用户创建的对象可以被应用程序的 DSP/BIOS API 调用所使用。这些对象具体可包括软件中断、任务、I/O 流以及事件日志。

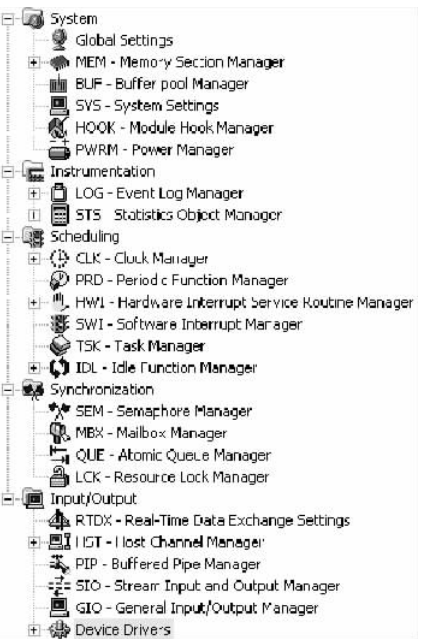


图 1-2 配置工具模块树

当用户对一个配置进行保存时，配置工具自动生成相应的文件并将其包含在当前项目(project)中。当静态配置 DSP/BIOS 时，DSP/BIOS 对象都是在程序运行前被预先设置好并被绑定到可执行程序中的。另外，DSP/BIOS 应用程序也可以在运行时动态地建立和删

除对象。

静态地创建对象，除了能通过消除运行时代码和优化内部数据结构来使目标存储器开销减至最小，还可以在程序编译之前通过对象属性有效性检测以及时地发现错误。

更多的细节请参见 DSP/BIOS 在线帮助和 2.2 节“静态配置 DSP/BIOS 应用程序”。

1.2.3　DSP/BIOS 分析工具

DSP/BIOS 分析工具可以帮助开发者在 CCS 环境中实现对 DSP/BIOS 应用程序的实时分析，使用户能够以可视化的方式监测一个运行中的 DSP/BIOS 应用程序，而几乎不影响应用程序的实时性能。DSP/BIOS 分析工具可以通过选择 CCS 中的“DSP/BIOS”菜单打开，如图 1-3 所示。

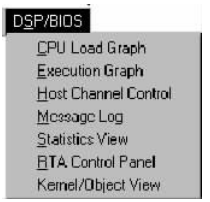


图 1-3　CCS 中的 DSP/BIOS 菜单

关于每个分析工具的具体细节请参见 DSP/BIOS 在线帮助及第 3 章的内容。

传统的调试方法只能在执行中的程序外部进行，要对程序进行实时调试分析，就需要在目标程序中包含实时监测服务，即在目标 DSP 上运行检测代码。通过使用 DSP/BIOS 的 API 函数和对象，应用程序可以自动地监测目标 DSP，实时采集信息并通过 CCS 分析工具上传到主机。

DSP/BIOS 提供的实时程序分析功能主要包括：

- **程序追踪**：显示被记录到目标日志的事件，反映程序执行过程中的动态控制流。
- **性能监测**：监测能够反映目标资源使用情况的统计信息，如处理器负荷及计时信息等。
- **文件流**：将目标 DSP 端的 I/O 对象绑定到主机文件，形成文件流。

当与 CCS 其他的通用调试功能轮流使用时，DSP/BIOS 实时分析工具能够在执行过程中实时地对目标程序的内部行为进行观察，然而传统的调试方法需要停止目标程序的运行来观察当前的内部行为，因此对程序内部行为的观察能力不足。即使在调试器停止程序之后，用户仍然可以使用主机 DSP/BIOS 分析工具捕获到的历史信息，对程序以往的内部行为进行观察和分析。

在软件开发的后期，当正常的调试手段在那些由时间相关的交互操作所引发的问题面前变得无效时，DSP/BIOS 分析工具则会像一个软件版的硬件逻辑分析器那样，完成一些特殊任务。

图 1-4 给出了若干 DSP/BIOS 分析工具的界面图。图 1-5 给出了 DSP/BIOS 分析工具的工具条，可以通过选择菜单项“View”→“Plug-in Toolbars”→“DSP/BIOS”显示和隐藏该工具条。

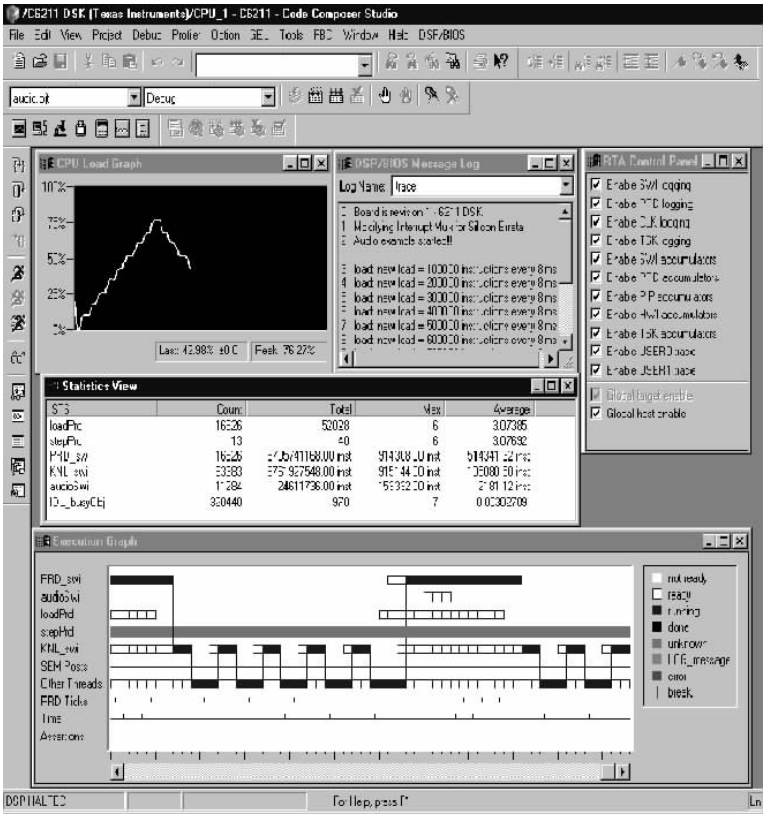


图 1-4 CCS 分析工具面板

图 1-5 DSP/BIOS 分析工具的工具条

1.3 命 名 规 则

DSP/BIOS 的每个模块都有一个唯一的名称,用来作为该模块的对象名、操作(函数)名或头文件的前缀。这个名称通常由 3 个或更多个大写字母和数字组成。

文中,DSP/BIOS 生成的源文件名称中的 54 这两个数字代表用户使用的具体 DSP 平台。如果用户 DSP 平台是 C6200,用 62 代替 54 即可。如一个 C6200 平台的 DSP/BIOS 汇编语言头文件的后缀名为.h62。如果用户 DSP 平台为 C55,用 55 代替 54 即可。

所有以 3 个大写字母和一个下划线开始的标识符(XXX_*)都被当作保留字。

1.3.1　模块头文件名

每个 DSP/BIOS 模块都有两个头文件，包含该模块接口中所有可用的常量、数据类型和函数的声明。

- **XXX.h**：为 C 程序提供的 DSP/BIOS API 头文件。用户的 C 语言源文件应该包含 std.h 头文件以及使用到的所有模块的头文件。
- **XXX.h54**：为汇编程序提供的 DSP/BIOS API 头文件。用户的汇编源文件应该包含所有使用到的模块的头文件。该头文件中包含了相应器件特有的一些宏定义。

用户应用程序必须在特定的程序源文件中包含每个用到的模块的头文件，另外 C 语言源文件中还必须在模块头文件之前包含 std.h，该头文件中含有标准类型和常量的定义。在 std.h 头文件之后，其他头文件的次序可以任意。例如：

```
#include <std.h>
#include <tsk.h>
#include <sem.h>
#include <prd.h>
#include <swi.h>
```

DSP/BIOS 有一些模块是在内部使用的，这些模块没有正式的文件说明，并在不断更新之中。但这些内部模块的头文件作为 DSP/BIOS 的一部分，也会在编译和链接 DSP/BIOS 程序时被自动包含进来。

1.3.2　对象名称

在 DSP/BIOS 配置中默认包含的系统对象，其名称都以 3 个或 4 个字母的模块名称为开头。如默认配置中包含一个名为 LOG_system 的 LOG 对象。

注意：*由用户静态创建的对象可由用户按照某种共同的规则来命名。用户可以将模块名作为对象名的后缀，例如，可将一个对数据进行编码的 TSK 对象命名为 encoderTsk。*

1.3.3　操作名

一个 DSP/BIOS API 函数的命名格式为 MOD_action，其中 MOD 为拥有该操作的模块的字母代码，action 代表该操作具体执行的动作。例如，SWI_post 函数由 SWI 模块定义，它会触发一个软件中断。

DSP/BIOS API 函数中有许多是系统内建(built-in)的，它们由各种系统内建的对象使用，以下给出一些例子：

- **CLK_F_isr**：由一个 HWI 对象使用，提供低分辨率 CLK 时钟。
- **PRD_F_tick**：由 CLK 对象 PRD_clock 使用，用来管理 PRD_swi 和系统时钟。

- **PRD_F_swi**：被 PRD_tick 触发，用来运行 PRD 函数。
- **_KNL_run**：由最低优先级的 SWI 对象 KNL_swi 调用，来运行任务调度。这其实是一个名为 KNL_run 的 C 函数。如果在汇编代码里进行调用，就需要在函数名前面加上一个下划线。
- **_IDL_loop**：由最低优先级的 TSK 对象 TSK_idle 调用，来运行 IDL 函数。
- **IDL_F_busy**：由 IDL 对象 IDL_cpuLoad 调用，来计算当前的 CPU 负荷。
- **RTA_F_dispatch**：由 IDL 对象 RTA_dispatcher 调用，来采集实时分析数据。
- **LNK_F_dataPump**：由 IDL 对象 LNK_dataPump 调用，来管理实时分析数据和主机通道数据向主机的传输。
- **HWI_unused**：并非函数名，该字符串在配置中用来标记未使用的 HWI 对象。

注意：*用户应用程序代码中不能调用任何以 MOD_F_开头的内建函数。以 MOD_和 MOD_F_形式开头的符号名称被保留由系统内部使用。*

1.3.4 数据类型名

DSP/BIOS API 不直接使用 C 语言的基本数据类型，如 int 或 char。相反的，为了确保对其他支持 DSP/BIOS API 的处理器的可移植性，DSP/BIOS 定义了自己的标准数据类型。这些数据类型如表 1-2 所示，它们都定义在 std.h 头文件里。

表 1-2 DSP/BIOS 标准数据类型

类　型	说　明
Arg	既能携带指针又能携带 Int 类型参数的数据类型
Bool	布尔型数据
Char	字符数据
Fxn	指向函数的指针
Int	带符号整型数据
LgInt	带符号长整型数据
LgUns	无符号长整型数据
Ptr	通用指针
String	以 \0 结束的字符串
Uns	无符号整型数据
Void	空类型

在 std.h 中还定义了其他一些数据类型，但并不被 DSP/BIOS API 使用。

另外，在 DSP/BIOS 中，标准常量 NULL(0)用来表示一个空的指针值，常量 TRUE(1)和 FALSE(0)用来表示布尔类型的值。

DSP/BIOS API 模块使用的对象结构体都使用 MOD_Obj 形式的命名约定，其中 MOD