Q/LS

龙芯中科技术有限公司企业标准

Q/LS 0022-2015

龙芯 CPU 开发系统 vxWorks BSP 开发规范

2015-06-20 发布　　2015-07-01 实施

龙芯中科技术有限公司　批准

文档更新记录	文档编号	Q/LS 0022-2015
	文档名	龙芯 CPU 开发系统 vxWorks BSP 开发规范
	版本号	V1.03
	创建人	研发中心
	创建日期	2015-06-20

更新历史

序号	更新日期	版本号	更新内容
1	2015.06.20	V1.03	发布文档初始版本 V1.03 版。

目 次

前　言

本规范是龙芯中科技术有限公司制定的企业规范，暂无国家相关行业通用规范可参考。

本规范涉及到龙芯 CPU 开发系统 vxWorks BSP 开发的相关要求，内容包括 vxWorks 系统架构、vxWorks 启动过程、windRiver 目录结构、BSP 命名规范、BSP 配置规范、编译规范、与 CPU 相关的宏定义及使用规范、设备驱动模型、中断管理模型、windML 驱动开发规范、调试规范、版本信息、代码风格和测试规范等。

本规范的起草单位：龙芯中科技术有限公司。

本规范的起草人：王洪虎、周学智、刘大同、李轶、毛卫龙、薛雨、吴苗苗。

本规范审核人：齐子初、段玮。

本规范批准人：胡伟武。

龙芯 CPU 开发系统 vxWorks BSP 开发规范

1 范围

本规范规定龙芯 1 号、2 号、3 号系列 CPU 开发系统 32 位 vxWorks BSP 的详细开发要求,内容包括 vxWorks 系统架构、vxWorks 启动过程、windRiver 目录结构、BSP 命名规范、BSP 配置规范、编译规范、与 CPU 相关的宏定义及使用规范、设备驱动模型、中断管理模型、windML 驱动开发规范、调试规范、版本信息、代码风格和测试规范等。建议其它系统厂商遵循此规范开发相关产品。

2 术语

a) vxWorks: vxWorks 操作系统是美国 WindRiver 公司于 1983 年设计开发的一种嵌入式实时操作系统（RTOS）,它以其良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中，如卫星通讯、军事演习、弹道制导、飞机导航等。

b) windRiver：风河（Wind River）公司是 Intel(NASDAQ: INTC)的全资子公司，也是全球领先的嵌入式和移动软件提供商。

c) windRiver workbench：windRiver 提供的集成开发环境。

d) BSP：板级支持包（*BSP*）是介于主板硬件和操作系统中驱动层程序之间的一层，一般认为它属于操作系统一部分，主要是实现对操作系统的支持，为上层的驱动程序提供访问硬件设备寄存器的函数包，使之能够更好的运行于硬件主板。

e) WindML: WindML 即 Wind Media Library（媒体库），它支持多媒体程序运行于嵌入式操作系统，风河公司设计它主要是用来提供基本的图形、视频和声频技术以及提供一个设计标准设备驱动程序框架。

f) LS:是 Loongson 的缩写，通常作为芯片型号的前缀出现，如 LS3A 表示龙芯 3A 芯片。

3 概述

本章涉及 vxWorks 系统架构、vxWorks 启动过程等方面的内容。VxWorks 美国 Wind River 公司推出的一种嵌入式实时操作系统，它以其良好的可靠性和卓越的实时性被广泛地应用于通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中。vxWorks 操作系统具有良好的实时性和稳定性、高效的任务管理、灵活的任务间通信方式、高度可裁剪、方便移植等特点。

3.1 vxWorks 系统架构

vxWorks 操作系统主要有 5 个组成部分，包括板级支持包 BSP、Wind 微内核、网络设备系统、文件系统以及 I/O 系统。vxWorks 的系统架构如图 1 所示:

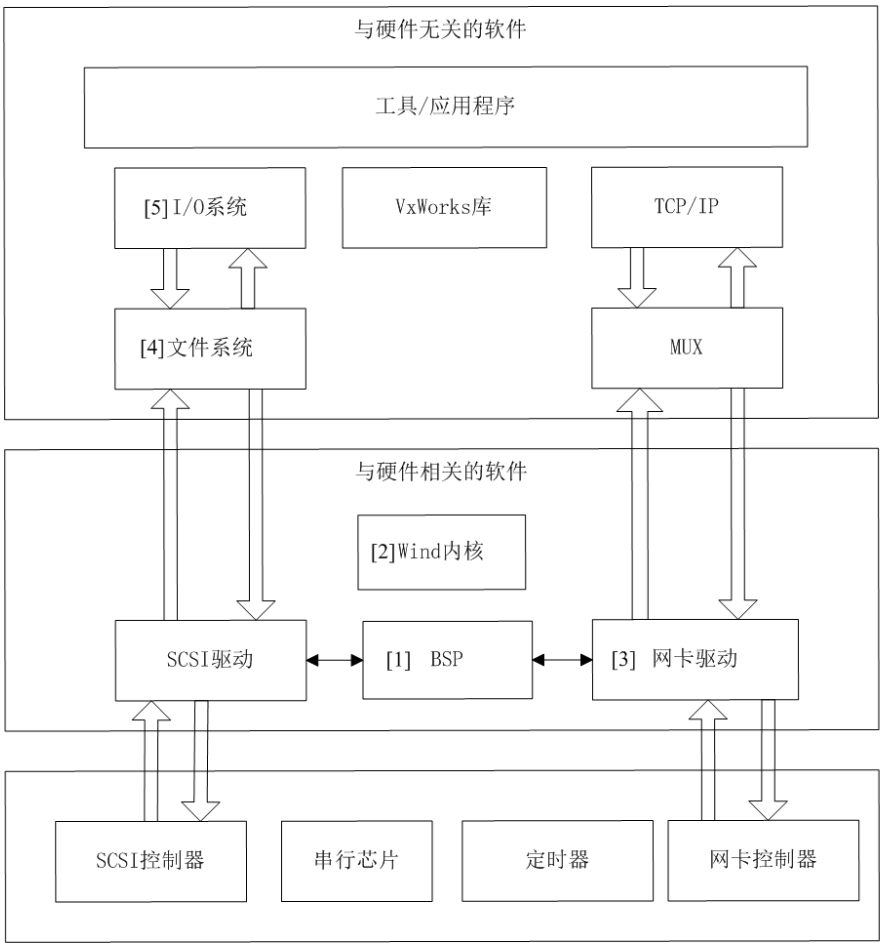

图 1 vxWorks 的系统架构

针对vxWorks系统架构注释如下：

[1] 板级支持包(BSP，Board Support Package)

在vxWorks操作系统中，板级支持包为介于底层硬件环境和vxWorks之间的软件接口，其主要功能是对各种硬件功能提供统一的软件接口，包括硬件的初始化、中断的产生和处理、硬件时钟和计时器管理、内存地址映射、内存空间大小、实时内核载入等。

[2] 高性能的实时操作系统内核

vxWorks的核心是高性能的微内核Wind。它支持实时任务切换、中断、基于优先级抢占式任务调度和时间片轮转调度等。Wind微内核体积小，响应快，减少了系统开销，从而保证了对外部异步事件的实时、稳定可靠的响应。

[3] 网络设备

网络设备是一类特殊的设备，它通过多层协议与应用程序交换信息。应用程序采用Socket接口作为操作函数集合。

[4] 文件系统

文件系统是操作系统实现的重要组成部分。vxWorks操作系统支持dosFs、rawFs、HRFS和ROMFS等多种文件系统。

[5] I/O系统

vxWorks 提供的 I/O 系统与 ANSI C 兼容，包括符合 POSIX 标准的异步 I/O 系统以及 Unix 标准的基本 I/O 系统。I/O 系统提供了与设备无关的用户接口。

4 BSP 命名及工程配置规范

4.1 BSP 命名规范

BSP 的名称以对应板卡的处理器+桥片+板卡类别+板卡形式+功能用途+特殊标识+版本号命名，相同处理器和桥片的 BSP 源码相同，以代码中以宏来区分具体板卡，发布时只保留与该板卡相关的代码。

例如：3A780E 开发板，命名为 LS3A_780E_Dev_vxWorksBSP_V1.0.2_Release; 3A780E 6U CPCI 开发板，命名为 LS3A_780E_6U_CPCI_vxWorksBSP_V1.0.2_Beta。

4.2 工程配置规范

如果想要更改工程的配置，不要只在内核配置组件里面添加组件，这只是临时的添加，需要将对应的宏在 BSP 下的 config.h 文件里面包含。

例如：添加网络驱动 syn 的组件 INCLUDE_SYNGMAC_VXB_END 。

需要将这个宏在 config.h 里面定义如下：

```
#define INCLUDE_SYNGMAC_VXB_END
```

5 PMON 功能调用区域保留规范

龙芯 PMON 规范中，为使 PMON 支持加载大文件及待机唤醒等功能，而把 PMON 的链接地址上调到时了 256M-16M 的地方(0x8f000000～0x8fffffff)，并且此空间在操作系统内核中应保留做为 BIOS 功能调用区(关机、重启、传参等)，故 vxWorks 地址空间配置时应保留出这 16MB 内存空间。

6 与 CPU 相关的宏定义及使用规范

每个 BSP 中与 CPU 相关的宏定义，如地址、寄存器等都统一放到与 BSP 同名头文件中，如 LS3A_780E.h。

如：

```
1   #ifdef CONFIG_LS2H_SB
2   #define LS2H_IO_REG_BASE                0x1b000000
3   #else
4   #define LS2H_IO_REG_BASE                0x1f000000
5   #endif
6   /* CHIP CONFIG regs */
7   #define LS2H_CHIP_CFG_REG_BASE          (LS2H_IO_REG_BASE + 0x00d00000)
8   #define LS2H_INT_REG_BASE               (LS2H_CHIP_CFG_REG_BASE + 0x0040)
9   #define LS2H_INT_ISR0_REG               (LS2H_CHIP_CFG_REG_BASE + 0x0040)
```

```
10  #define LS2H_INT_IEN0_REG               (LS2H_CHIP_CFG_REG_BASE + 0x0044)

11  #define LS2H_INT_SET0_REG               (LS2H_CHIP_CFG_REG_BASE + 0x0048)

12  #define LS2H_INT_CLR0_REG               (LS2H_CHIP_CFG_REG_BASE + 0x004c)
```

7 大内存配置规范

vxWorks 支持内存扩展，支持动态 TLB 页面切换，但页面切换是一个特殊的异常处理过程，异常处理、TLB 缺页处理、TLB 维护等都需要系统开销，这对 vxWorks 的实时性会有不利影响。龙芯处理器有 64 个 TLB 项，每项支持最大 16MB 页面，这样可以在限制最大内存 1GB 的条件下设置为静态 TLB 映射，避免 TLB 异常带来的系统开销。

7.1 内存容量限制

为了兼顾 vxWorks 的实时性和大内存需求，龙芯处理器上所有的 vxWorks 原则上使用的最大内存不超过 1GB，如有特殊需求需要超过 1GB 的不在本规范限制之内。

7.2 TLB 页面配置

所有的 TLB 页面配置最大为 16MB，最小为 16KB，不得使用 4KB 页面。

8 设备驱动模型

8.1 基于 vxbus 架构的设备驱动

8.1.1 vxbus 架构驱动框架

以 ahci 为例：

```
1   LOCAL DRIVER_INITIALIZATION vxbIntelAhciFuncs =
2   {
3       vxbIntelAhciInstInit,         /* devInstanceInit * /
4       vxbIntelAhciInstInit2,        /* devInstanceInit2 * /
5       vxbIntelAhciInstConnect       /* devConnect * /
6   };
7
8   LOCAL PCI_DRIVER_REGISTRATION vxbIntelAhciStorageRegistration =
9   {
10      {
11          NULL,               /* pNext == NULL * /
12          VXB_DEVID_DEVICE,  /* This is a device driver * /
13          VXB_BUSID_PLB,     /* hardware resides on a PCI bus * /
14          VXB_VER_4_0_0,   /* targeting vxbus version 2 api * /
15          "intelAhciSata",     /* driver named IntelAhci * /
16          &vxbIntelAhciFuncs, /* set of 3 pass functions * /
17          NULL,               /* no methods * /
```

```
18          NULL,               /* no probe function, use device id and vendor id * /
19      },
20      NELEMENTS(vxbIntelAhciIdList),
21      &vxbIntelAhciIdList[ 0 ],
22  };
```

注：以上驱动相关函数需要展开，下面会有具体示例。

8.1.2 hwconf.c 文件

a) 添加驱动的相关资源信息：

根据实际驱动中 devResourceGet()函数需要的信息做相应赋值。

例如 USB 的 OHCI：

```
1   struct hcfResource vxbPciUsbOhciDev0Resources[] =
2       {
3       {"regBase",     HCF_RES_INT,   {(void *)0xb2e08000} },
4       {"cpuToBus", HCF_RES_ADDR,   {(void*)usbMemToPci} },
5       {"busToCpu", HCF_RES_ADDR,   {(void *)usbPciToMem} },
6        {"dataSwap", HCF_RES_ADDR,   {(void *)NULL} } ,
7       };
8   #define vxbPciUsbOhciDev0Num NELEMENTS(vxbPciUsbOhciDev0Resources)
```

b) 注册中断以 USB 为例：

在中断控制器中挂载设备中断：

```
1   /* pin, driver, unit, index */
2   {33, "vxbPlbUsbOhci", 0, 0},
```

将设备资源挂载到 list 里面：

```
1   const struct hcfDevice hcfDeviceList[] = {
2       {     "vxbPlbUsbOhci",      0,      VXB_BUSID_PLB,      0,      vxbPciUsbOhciDev0Num,
    vxbPciUsbOhciDev0Resources },
3
4   }
```

8.1.3 修改 BSP

在 BSP 的文件 hwconf.c 里面添加驱动的相关注册信息:

```
1   const struct hcfResource wrsampleResources[] = {
2       {"regBase", HCF_RES_INT, {(void * )0}},       /* dummy reg base address * /
3   };
4   #define wrsampleNum NELEMENTS(wrsampleResources)
```

资源 list hcfDeviceList[]驱动的名字需要和你驱动中的 drvName 对应。

```
    { "wrsample", 0, VXB_BUSID_PLB, 0, wrsampleNum, wrsampleResources },
```

这是简单的注册实例 hwconf.c :

```
1   /* hwconf.c - HW configuration support module * /
2
3   #include <vxWorks.h>
4   #include <vxBusLib.h>
5   #include <hwif/vxbus/vxBus.h>
6   #include <hwif/vxbus/vxbPlbLib.h>
7   #include <hwif/vxbus/hwConf.h>
8   #include <config.h>
9
10  const struct hcfResource wrsampleResources[] = {
11      { "regBase",          HCF_RES_INT,    {(void * )0} },
12  };
13  #define wrsampleNum NELEMENTS(wrsampleResources)
14
15  const struct hcfDevice hcfDeviceList[] = {
16      { "wrsample", 0, VXB_BUSID_PLB, 0, wrsampleNum, wrsampleResources },
17  };
18  const int hcfDeviceNum = NELEMENTS(hcfDeviceList);
```

注:上面写的是把驱动程序添加到 BSP 里面,把驱动程序文件放到 BSP 目录中,把 cdf 文件放到 comps/vxWorks 下，还需要将 dr、dc 文件添加到 comps/src/hwif 里面，各个文件的含义和上述一样，只是放置位置不同。

8.2 vxbus 驱动组件添加

以 vxWorks 自带的驱动示例程序为例，包含驱动文件 wrsample.c wrsample.h 和其它驱动配置文件。

- 40wrsample.cdf：驱动组件, 定义宏 DRV_DEMO_WRSAMPLE 和宏 DRV_DEMO_WRSAMPLE_DEBUG
- wrsample.dc：驱动注册函数原型，这个驱动注册函数为 wrsampleRegister()，驱动注册函数被重定向到${WIND_BASE}/target/config/all/vxbUsrCmdLine.c
- wrsample.dr ： 驱动注册函数被重定向到 ${WIND_BASE}/target/config/all /vxbUsrCmdLine.c ，并在函数 hardWareInterFaceBusInit()调用。如果 DRV_DEMO_WRSAMPLE 在 BSP 的 config.h 定义了，注册函数必须以下面的格式来定义：
  - #ifdef <所加驱动的宏>
  - 注册函数 register();
  - #endif /* 所加驱动的宏 */
- wrsample.c：VxBus 驱动文件
- wrsample.h： VxBus 驱动头文件

说明：需要将文件 40wrsample.cdf 拷贝到路径${WIND_BASE}/target/config/comps/vxWorks 才会在组件配置里面出现项对应的宏，此时驱动需要在组件里面添加之后才能被编译到 vxWorks 镜像中去。

另外，还需要将驱动文件放到 BSP 文件夹里面，并将 wrsample.o 添加到 BSP 里面的 Makefile 里面去。然后编译工程的时候就会将驱动文件一起编译。（需要关掉内核工程再打开，否则 cfg 配置文件不会加载到工程里面；更改 Makefile 需要重新建工程，否则对于现有工程无效）

8.3 非 vxbus 架构的设备驱动

非 vxbus 架构的驱动有两种，一种是指 vxWorks 库里自带的驱动，另一种是从非 vxWorks 系统移植而来的驱动。

如果是 vxWorks 库自带且现有 vxWorks 版本继续支持 legacy 驱动，则继续延用；
如果是从非 vxWorks 系统移植而来的驱动，则按 vxbus 架构重新改写。

9 中断管理模型

9.1 中断控制器

9.1.1 中断控制器的代码存放位置

a）在 BSP 中有 intCtlr 的文件夹，存放中断相关的代码；

b）每个中断控制器有单独的文件夹，存放在 intCtlr；

c）需要硬件工程师提供的整个板卡中断连接说明文档；

9.1.2 中断控制器驱动的编译

对于要编译的文件应当写进 Makefile 中，其它方式则为非标准，例如：

在 sysLib.c 中有一句：

#include "vxbMipsLsnIntCtlr.c"

这种编译方式是非标准的。

9.1.3 中断控制器驱动规范

中断寄存使用规范：与板卡相关的寄存器地址应当使用宏，而不是直接写地址；
中断号计算规范：应当使用中断控制器的状态寄存器和中断使能寄存器相与的结果；
中断处理规范：systemIsr 的处理应当使用系统提供的标准接口，见 8.2 节中说明；
例如：

下边一段代码是某 BSP 中 vxbMipsLsnIntCtlr.c 文件的中断分发函数，其中

ints（中断号计算）和 systemIsr（中断处理）即为非规范方式，需要按规范重新设计。

```
1   LOCAL VOID vxbMipsLsnIntCtrlIsr
2       (
3       VXB_DEVICE_ID       pInst
4       )
5       {
6       UINT32 ints;
7       int i;
8       VXB_LSN_INT_DRV_CTRL * pDrvCtrl;
9
10      pDrvCtrl = (VXB_LSN_INT_DRV_CTRL *)(pInst->pDrvCtrl);
11
```

```
12      {
13      /* get active interrupts */
14      vxbMipsLsnIntLevelGet (pInst, &ints);
15      /* scan for active interrupts */
16      ints &= (1<<2)|(1<<5)|(1<<6)|(1<<7)|(1<<4)|(1<<16)|(1<<18)|(1<<20)|(1<<21); /*modified
    by jian & yuan,20130308*/
17      if (ints != (UINT32)0)
18          {
19          for (i = 0;i < LSN_CONTROLLER_INT_INPUTS && ints != 0;i++,
20               ints >>= 1)
21              {
22              #if 0
23              if((ints & 0x01) !=0 && i == 2)
24                   systemIsr(p1553b->BuConf);
25           else
26          #endif
27               if (((ints & 0x01) != 0) && (i!=0))
28                   {
29                   VXB_INTCTLR_ISR_CALL(&pDrvCtrl->isrHandle, i);
30                   }
31              }
32          }
33      }
34      return;
35      }
```

9.2 设备中断

9.2.1 设备中断函数

函数原型如下：

a） 基于 vxbus 的设备驱动

```
1   STATUS vxbIntConnect
2       (
3       struct vxbDev * pDev,
4       int         index,
5       VOIDFUNCPTR     pIsr,
6       void *      pArg
7       );
8   STATUS vxbIntDisconnect
9       (
10      struct vxbDev * pDev,
```

```
11      int         index,
12      VOIDFUNCPTR     pIsr,
13      void *      pArg
14      );
        STATUS vxbIntEnable
15      (
16      struct vxbDev * pDev,
17      int         index,
18      VOIDFUNCPTR     pIsr,
19      void *      pArg
20      );
21  STATUS vxbIntDisable
22      (
23      struct vxbDev * pDev,
24      int         index,
25      VOIDFUNCPTR     pIsr,
26      void *      pArg
27      );
```

b）传统的设备驱动（非 vxbus 模型）

```
1   STATUS intConnect (VOIDFUNCPTR * vector, VOIDFUNCPTR routine,
2   int parameter);
3   STATUS   intDisconnect (VOIDFUNCPTR * vector, VOIDFUNCPTR routine,
4                                   int parameter);
5   int intEnable (int);
6   int intDisable (int);
```

9.2.2 设备中断处理函数规范化

对于共享中断，设备的中断处理函数应当先判断此中断是否是本设备引发的，如果是则继续执行，否则立即返回。

10 调试规范

10.1 EDR 调试

EDR(Error Detection and Reproting)的错误探测机制：遇到了严重的错误会进入 EDR 的处理流程；

10.1.1 需要添加的组件

- INCLUDE_EDR_PM
- INCLUDE_EDR_ERRLOG
- INCLUDE_EDR_SHOW
- INCLUDE_EDR_SYSDBG_FLAG
- INCLUDE_MEM_EDR_SHOW

■ INCLUDE_MEM_EDR_RTP_SHOW

10.1.2 使用方法

edrShow( ) Show all records.

edrFatalShow( ) Show only FATAL severity level records.

edrInfoShow( ) Show only INFO severity level records.

edrKernelShow( ) Show only KERNEL event type records.

edrRtpShow( ) Show only RTP (process) event type records.

edrUserShow( ) Show only USER event type records.

edrIntShow( ) Show only INTERRUPT event type records.

edrInitShow( ) Show only INIT event type records.

edrBootShow( ) Show only BOOT event type records.

edrRebootShow( ) Show only REBOOT event type records.

10.1.3 EDR 调试出错处理

按照 vxWorks 默认配置应该是没问题的，如果出错如：

ED&R was unable to initialize

Check that PM_RESERVED_MEM and EDR_ERRLOG_SIZE areproperly defined in the BSP

edrLibInit failure eas :pmRegionAddr failed

value = -1 = 0xffffffff

可能是因为 PM_RESERVED_MEM 太小，适量增大 PM_RESERVED_MEM 的数值。

10.2 EJTAG 调试

使用 EJTAG 进行 GDB 调试，需要借助 debug 调试信息，将加有调试信息的 vxWorks 映像拷贝到 ejtag 运行所在目录，并在 PMON 下 load 该 vxWorks 内核映像。打开 ejtag 调试软件（ejtag_debug_usb.exe），使用命令 gdb vxWorks 即可进行调试。详细配置方法可参照附录 A 《Ejtag 调试》。

10.3 WDB 调试

WDB 调试需要选中 INCLUDE_WDB_COMM_END、INCLUDE_WDB_RTP 组件。WDB 调试功能需要 WDB 的网络配置与板卡启动时默认网络的 IP 保持一致。做好相应配置后，即可启动 vxWorks 并连接 WDB，下载需要调试的应用程序，开始调试。

详细配置方法可参照附录 B 《vxWorks6.7 下 WDB 调试》

10.4 调试信息输出

遵循几个原则：

- 在中断中不要有调试信息，如果必须打印，可以使用 logMsg；
- 在调试过程中可以使用串口的轮询输出；
- 在 vxWorks 启动完毕、系统初始化的时候，不要使用串口的轮询输出；
- 在提交代码时需将调试信息移除，如果必须添加调试信息可以使用宏控制是否输出；

11 版本管理信息

11.1代码管理

vxWorks 代码使用 git 进行版本管理

11.2 版本信息记录

在 vxWorks 的 config.h 文件中增加板卡信息和代码版本信息，如 LS3A_780E、commit cef928ae7958f7008fa46172cfe983c5af12dbb7

Author: wanghonghu <wanghonghu@loongson.cn>

Date: Mon Sep 16 10:02:13 2013 +0800

BSP 版本信息例如 vxWorks6.7/6.8 系统会默认打印。

12 代码风格

vxWorks BSP 主要采用 C 语言写成，部分文件采用汇编，其 C 语言部分编码风格参见附录 C 的建议。

13 测试规范

BSP 开发完成后，应进行质量测试，主要包括功能测试、性能测试和稳定性测试等，参见附录 D。

14 BSP 说明文档

BSP 开发完成后，增加一个 PDF 格式的说明文档，存放在产品配置库中，至少包含以下内容：

1) 地址空间说明
2) 中断路由说明
3) 镜像工程建立和编译说明
4) 内核库建立和编译说明
5) 开发板基本构成说明

15 开发和发布流程

1) 相同处理器和桥片的各类板卡，代码仓库中各 BSP 包使用同一份源代码，使用宏来区分板卡相关的代码。
2) 原则上，系统研发部负责各类开发板 vxWorks BSP 包的适配；事业部负责基于开发板的客户定制版 vxWorks BSP 包的适配。
3) 系统研发部和事业部共享代码开发库，以 git 形式进行版本管理，各个 BSP 开发人员均可提交 patch，经过审核后合并到代码库中。
4) 事业部和系统研发部根据项目需求，开发相关 vxWorks BSP 包并在通过内部测试后，按产品库配置管理要求形成 BSP 产品，并提交科研管理部。
5) 科研管理部根据产品库配置清单取得完整的 BSP 产品后，提交质量办进行产品质量验收。
6) 质量办根据产品库配置中的测试大纲，完成 BSP 包的测试验证。通过质量办测试验证后，正式进入科研管理部产品库存档，否则退回到研发部继续开发。
7) 事业部在对外发布 BSP 源代码时，应进行 Release 工作，只保留与本板卡相关的代码，去除与本板卡无关的代码。

附录 A

（规范性附录）
Ejtag 调试

使用 EJTAG 进行 GDB 调试，需要借助 debug 调试信息，BSP 工程默认是带有调试信息的，因此还需要在库工程中增加调试信息。

创建一个库的工程，打开所生成的库工程的 Makefile，增加以下内容：

temp_include_current = ADDED_CFLAGS+=-I.\ -g

然后保存文件，编译库工程。将编译的带有调试信息的 MIPS 库拷贝到。如 A.1 所示：

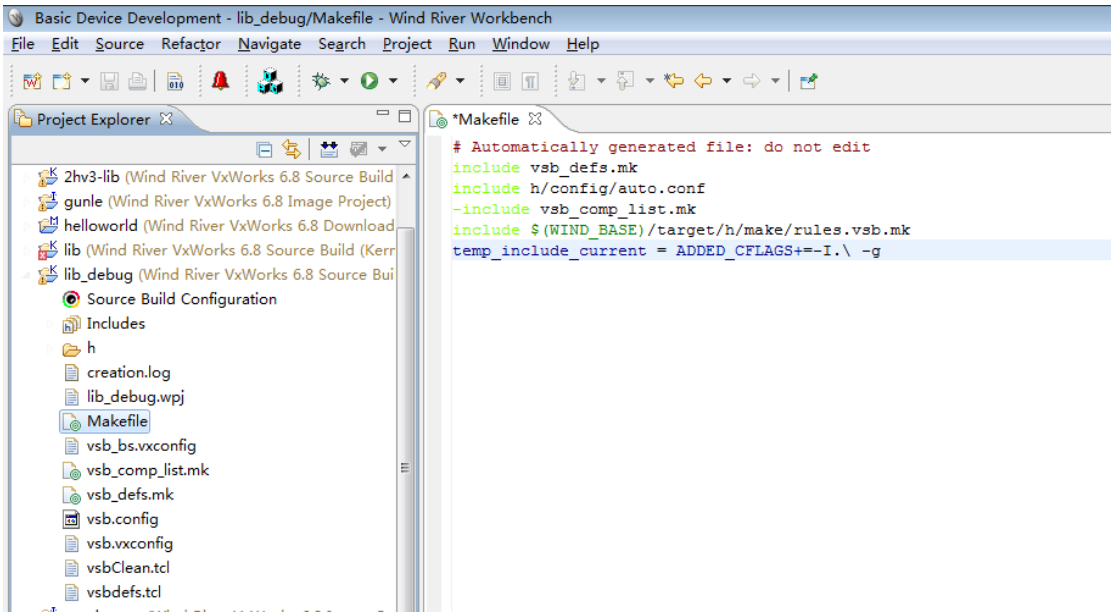

图 A.1 库工程增加调试信息

先将系统启动所使用的 vxWorks 映像拷贝到 ejtag 运行所在目录，并在 PMON 下 load 该 vxWorks 内核映像。如图 A.2 所示

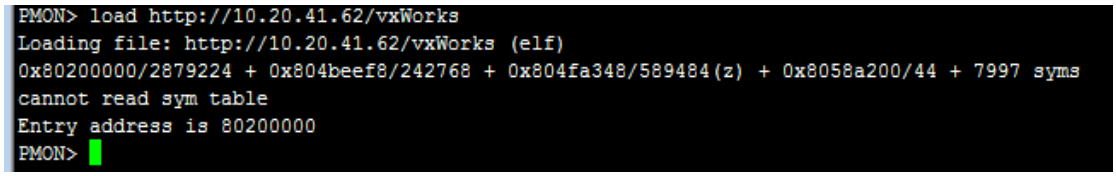

图 A.2 加载 vxWorks 内核

之后，打开 Windosws 下的 ejtag 调试软件（ejtag_debug_usb.exe），在 ejtag 软件中加载板卡相关配置 source configs/config.ls2h，然后输入命令 gdb vxWorks。在 gdb 中可以增加断点，例如使用 b sysHwInit 命令，在 sysHwInit()函数位置加断点。增加断点后，就可以在 PMON 下使用 g 命令，运行 vxWorks 系统了。当系统运行到 sysHwInit()函数位置时，就会在 GDB 中暂停，此时就可以使用 GDB 命令进行调试了。运行效果如下图 A.3 所示：

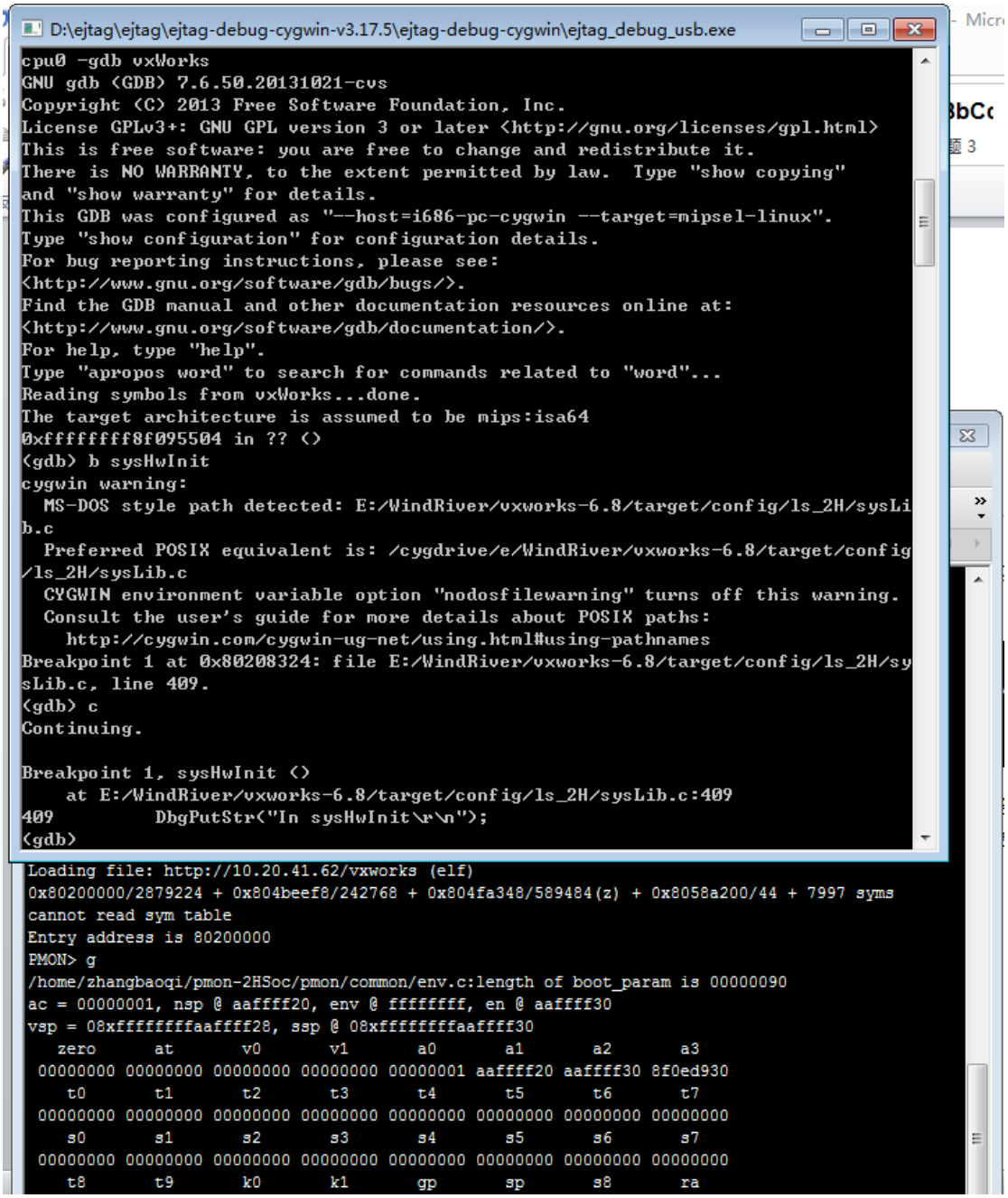

图 A.3 使用 ejtag 进行 gdb 调试运行效果

附录 B

（规范性附录）
vxWorks6.7 下 WDB 调试

B.1 工程配置

(1)创建好 BSP 工程(我的工程名字叫 2G3U_6.7)需要配置如下相关组件：

配置 default bootline 宏的主机 IP(10.0.0.25)，目标机 IP(10.0.0.82)。(注意：网络设备名 syn 与自己 BSP 中 Hwconfig.c 中注册的网络设备名称一致，即 vxWorks 启动后 ifconfig 打印出的网络设备名)，如图 B.1 所示。

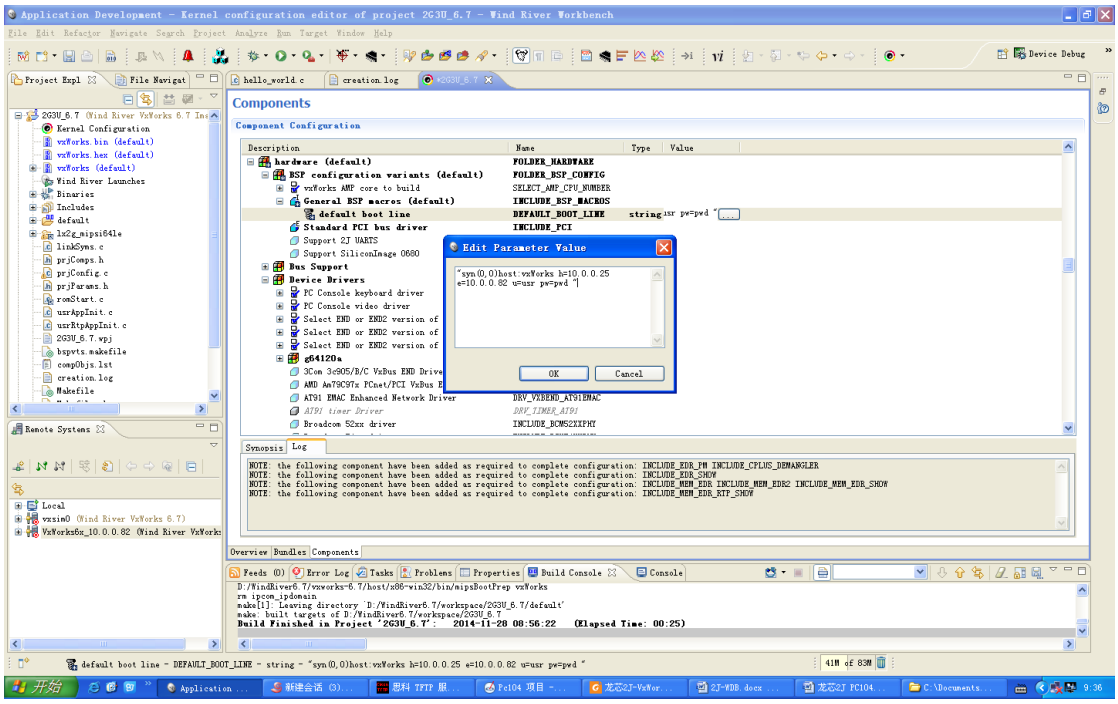

图 B.1 BSP 工程配置图

(2)配置组件 INCLUDE_WDB_COMM_END 相应的 IP 为“10.0.0.82”(一定要加双引号)，如图 B.2 所示。

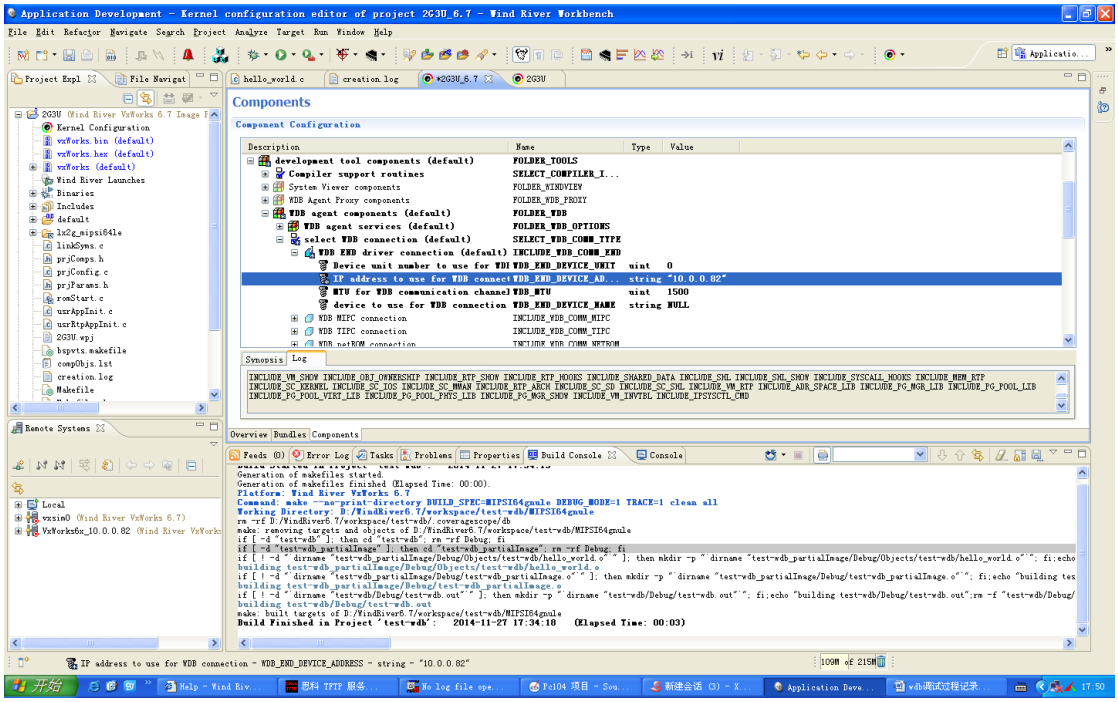

图 B.2 WDB_COMM_END 组件配置图

(3)包含组件 INCLUDE_WDB_RTP(对比相关组件与图片的组件一致)。

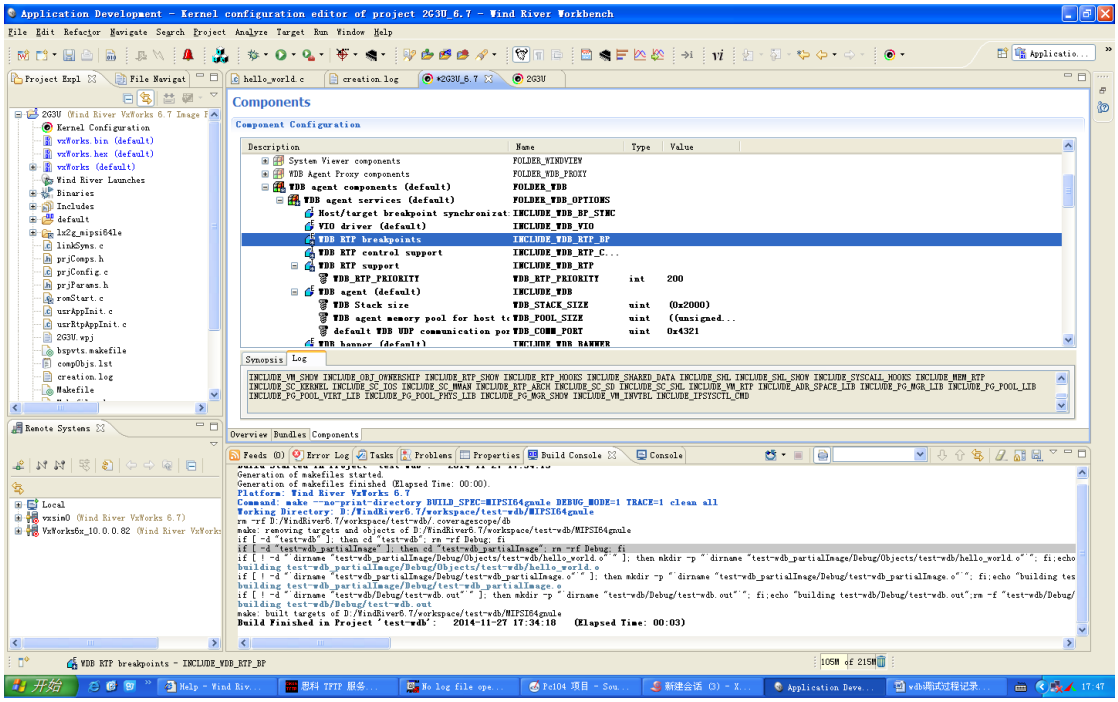

图 B.3 WDB_WDB_RTP 组件配置图

(4)重新编译工程。

如图 B.4 所示。

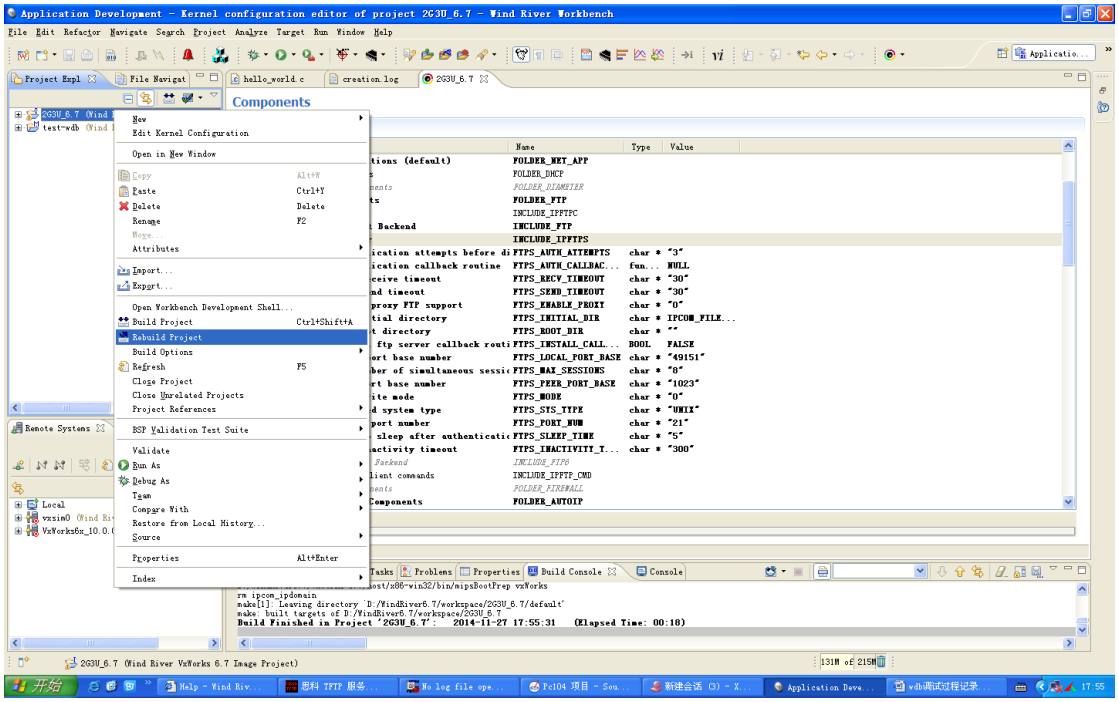

图 B.4 WDB 编译

说明：这个工程编译出来的内核 vxWorks 内部集成 WDB 相关组件功能，有 WDB Server 可以支持主机与目标机在线调试（如:对应用程序的单步调试、断点调试功能）。

B.2 创建应用工程

（1）新建一个工程选择 vxWorks Download kernel module project。

（2）工程名字任意（test_wdb）。

（3）选上 RemoteSystemsTempFiles。

（4）选择 MIPSI64gnule 编译工具链。

（5）添加一个应用调试文件（名字加 hello_world.c）。

（6）在 hello_world.c 里面添加应用程序。

（7）重新编一下工程，此时在安装环境 worksbench 的 workspace 里能看见 test-wdb.out(D:\WindRiver6.7\workspace\test-wdb\MIPSI64gnule\test-wdb\Debug)。

说明：编译应用工程在 D:\WindRiver6.7\workspace\test-wdb\MIPSI64gnule\test-wdb\Debug 目录下有一个后缀为.out 的文件，该文件支持 vxWorks6.7 的 WDB 应用程序在线调试功能，支持源码可见的单步执行、断点调试等一系列 WDB 相关调试手段。

B.3 建立 WDB 连接

B.3.1 开机后界面

vxWorks 在配置好 WDB 后会打印：

WDB Comm Type: WDB_COMM_END

WDB: Ready.

说明：系统启动起来，vxWorks 的 LOGO 下面有如上 WDB 正确信息，WDB 初始化以及网络组件初始化能够正确工作。

B.3.2 新连接

(1)右键选择“vxsim0”，再选择“new”，“Connection”。

(2)选择“Wind River VxWorks 6.x Target Server Connection”

(3)选择“file”，“Browse”选择 vxWorks 镜像路径，最后点击“Finish”完成连接建立。

说明：workbench 会自动连接目标机启动的内核，加载符号表，及相关文件。

B.3.3 download 应用程序

(1)下载配置如图 B.5 所示。

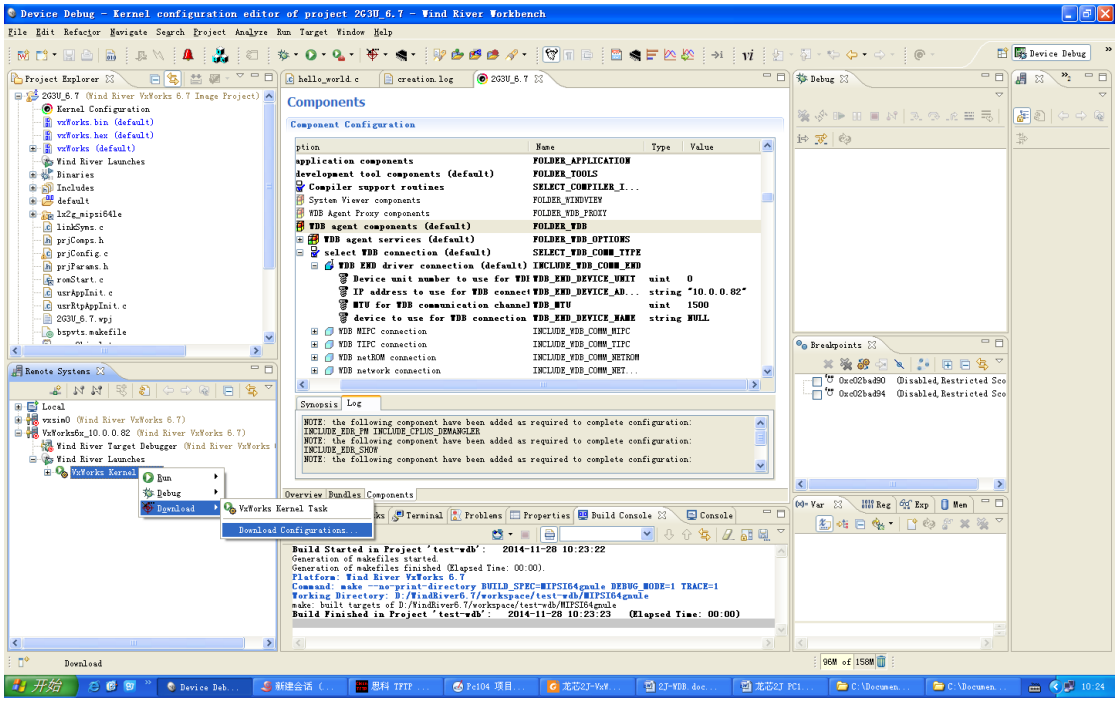

图 B.5 WDB 下载配置图

(2)选择“Download”，“Add”添加需要调试的文件(即编出来的 test-wdb.out)，如图 B.6 所示。

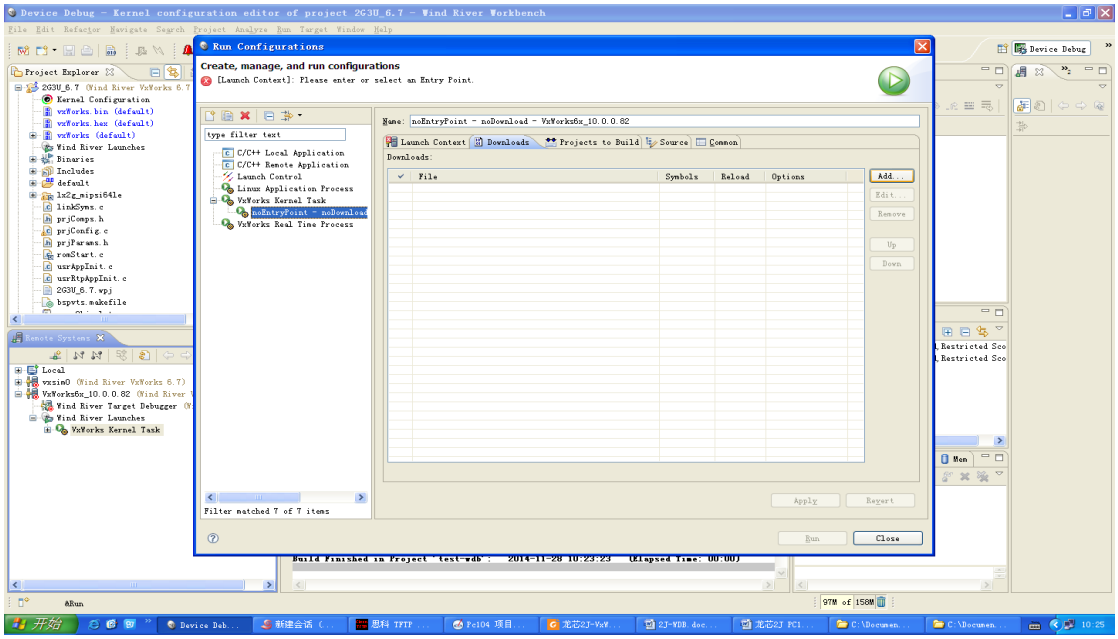

图 B.6 WDB 下载配置图（2）

（3）添加文件完成后，apply，然后 ok。

将应用程序 download 到目标机里，且加载符号表到 WDB 后，此时 WDB 可以支持应用程序的原码调试了。

B.4 选择入口函数

选择 main，如图 B.7 所示。

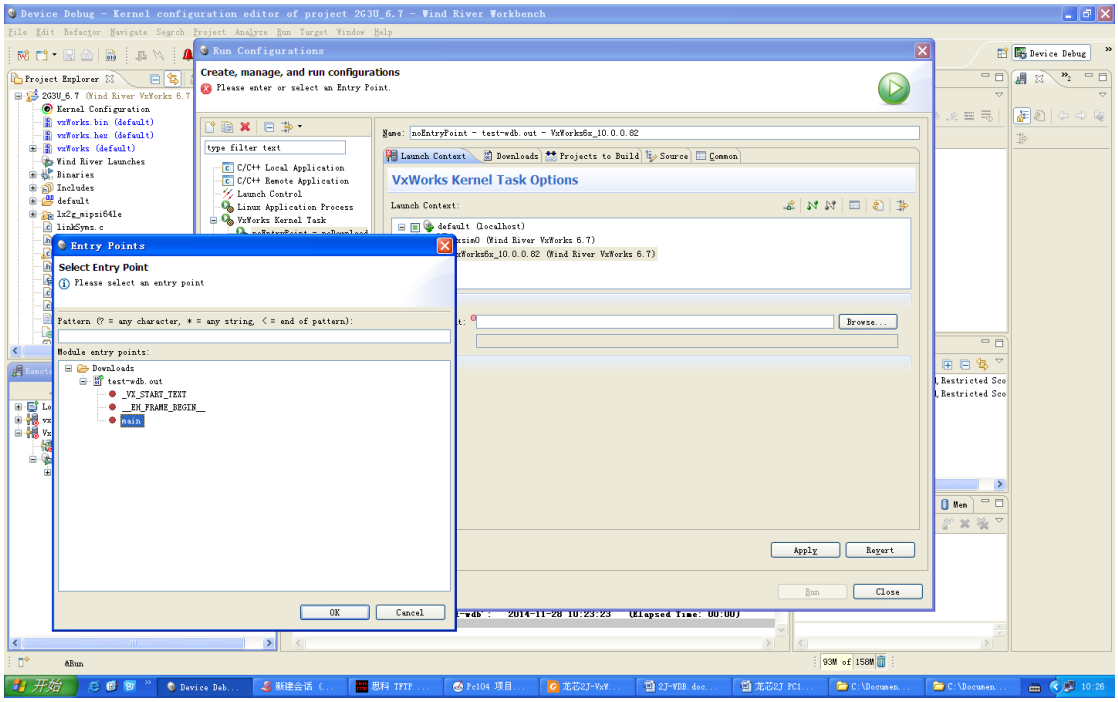

图 B.7 WDB 调试入口函数选择

B.5 WDB 调试

使用单步调试看左下角变量 i 的值的增加，如图 B.8 所示。

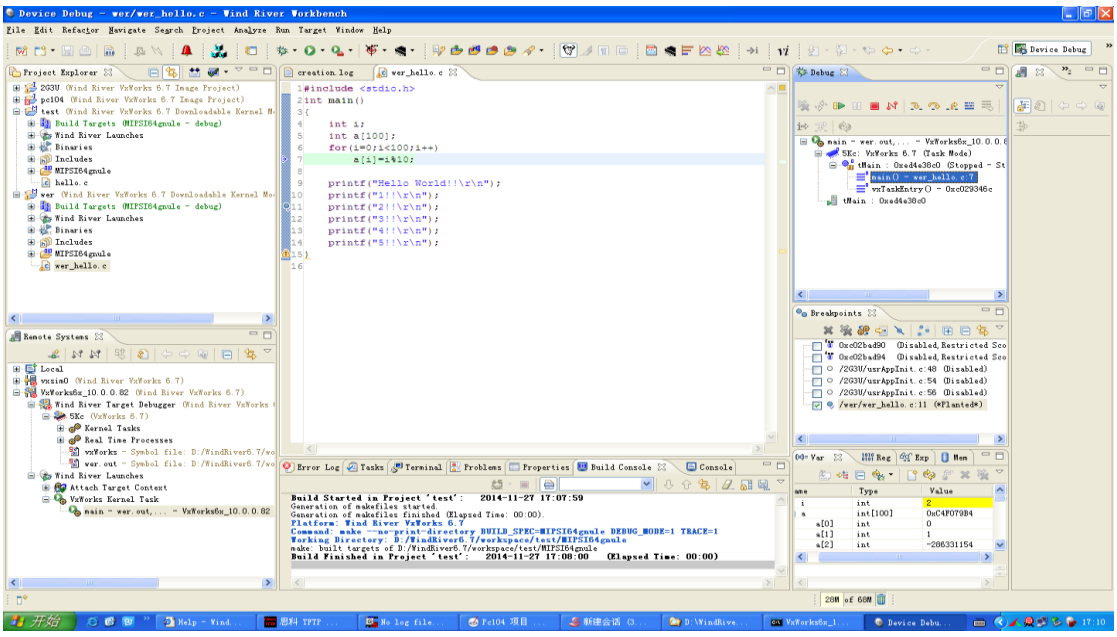

图 B.8 单步调试及 watch 功能图

附录 C

（规范性附录）

vxWorks 的 C 代码编程风格

C.1 说明

vxWorks 历史代码不强制要求按此约定修改，新写模块自本规范发布之日按此约定执行。本附录用到如下关键字的含义如下：

原则：必须坚持的指导思想

规则：强制必须遵守的约定

建议：必须加以考虑的约定

C.2 文件

文件概指 .c 和 .h 文件，.h 文件提供接口，.c 文件提供具体实现。合理的头文件布局可以减少编译时间及降低代码维护难度，正确使用头文件可令代码在可读性、文件大小和性能上大为改观，现引入以下方法来帮助合理规划头文件。

1） 规则——头文件 #define 保护

说明：所有头文件都应该使用 #define 防止头文件被多重包含。

示例：loongson3_dev.h

```
#ifndef LOONGSON3_DEF_H_
#define LOONGSON3_DEF_H_
…
#endif
```

2） 规则——头文件中适合放置接口声明，不适合放置实现

说明：头文件是模块(Module)或单元(Unit)的对外接口。头文件中应放置对外部的声明，如对外提供的函数声明、宏定义、类型定义等。

内部使用的函数(相当于类的私有方法)声明不应放在头文件中。

内部使用的宏、枚举、结构定义不应放入头文件中。

变量定义不应放在头文件中，应放在 .c 文件中。

变量的声明尽量不要放在头文件中，亦即尽量不要使用全局变量作为接口。变量是模块或单元的内部实现细节，不应通过在头文件中声明的方式直接暴露给外部，应通过函数接口的方式进行对外暴露。即使必须使用全局变量，也只应当在 .c 中定义全局变量，在 .h 中仅声明变量为全局。

3） 规则——禁止 #include “*.c”

说明：该方法会会使得代码可读性变差以及导致一系列编译问题，如 a.c 中使用了 #include “b.c”，则当 b.c 有更新时，由于 Makefile 中没有显示指定这种依赖，会导致 b.c 的更新没有被编译，增加了调试困难。

4） 建议——每一个 .c 文件应有一个同名 .h 文件，用于声明需要对外公开的接口

说明：如果一个 .c 文件不需要对外公布任何接口，则其就不应当存在，除非它是程序的入口，如 main 函数所在的文件。

其它 .c 文件只能通过包含头文件的方式使用该 .c 提供的接口，禁止在其它 .c 中通过 extern 的方式使用该 .c 提供的外部函数接口、变量，当外部函数接口发生改变时，extern 方式容易导致声明与定义不一致等弊端。

5） 建议——C 文件描述

说明：建议 C 文件按如下示例进行布局，将相同内容相邻放置。

示例：

```
 /*************************************************************
对源程序的简要说明，如主要完成功能，独立或依赖关系等
亦可附加作者、完成日期、版本等信息
*************************************************************/

#include <stdio.h>  //包含所需头文件

unsigned int g_device_counts = 0;     //定义全局变量

#define MAX_COUNT   128     //定义所需宏

static int index = 0;  //定义所需本地变量

struct struct_name {    //定义本地数据结构
…
};

void func (unsigned char flag, int mask) //函数声明

/*
*函数注释
*/
void func (unsigned char a,  /* 注释参数 a 的含义 */
          int b, /* 注释参数 b 的含义 */
          unsigned long c)      /* 注释参数 c 的含义 */
{
…
}
```

C.3 命名

1） 规则——通用命名规则

说明：统一采用 UNIX Like 风格：单词用小写字母，每个单词直接用下划线“_”分割，例如 text_mutex，kernel_text_address。

函数命名， 变量命名， 文件命名应具备描述性，不要过度缩写。类型和变量应该是名词， 函数名可以用 “命令性” 动词(如 open， find 等)。

2） 原则——除了常见的通用缩写以外，不使用单词缩写，不使用汉语拼音

说明：较短的单词可通过去掉“元音”形成缩写，较长的单词可取单词的头几个字母形成缩写，一些单词有大家公认的缩写，常用单词的缩写必须统一。

示例：一些常见可以缩写的例子:

argument	可缩写为	arg
buffer	可缩写为	buff
clock	可缩写为	clk
command	可缩写为	cmd
compare	可缩写为	cmp
configuration	可缩写为	cfg
device	可缩写为	dev
error	可缩写为	err
hexadecimal	可缩写为	hex
increment	可缩写为	inc
initialize	可缩写为	init
maximum	可缩写为	max
message	可缩写为	msg
minimum	可缩写为	min
parameter	可缩写为	para
previous	可缩写为	prev
register	可缩写为	reg
semaphore	可缩写为	sem
statistic	可缩写为	stat
synchronize	可缩写为	sync
temp	可缩写为	tmp

3） 建议——用正确的反义词组命名具有互斥意义的变量或相反动作的函数等

示例:

```
add/remove              begin/end         create/destroy
insert/delete           first/last    get/release
increment/decrement    put/get            add/delete
```

```
lock/unlock         open/close        min/max
old/new                  start/stop        next/previous
source/target            show/hide         send/receive
source/destination   copy/paste        up/down
```

4) 建议——宏命名

说明：采用全大写字母，单词之间加下划线 ”_” 的方式命名(枚举同样建议使用此方式定义)。

C.4 注释

1) 原则——通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的

说明:清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

2) 规则——注释应放在其代码上方相邻位置或右方，不可放在下面。如放于上方则需与其上面的代码用空行隔开，且与下方代码缩进相同

示例:

```
/* active statistic task number */
#define MAX_ACT_TASK_NUMBER 1000
#define MAX_ACT_TASK_NUMBER 1000 /* active statistic task number */
```

3) 规则——修改代码时，要维护代码周边的所有注释，以保证注释与代码的一致性，不再有用的注释要删除

4) 建议——残留代码

残留代码请及时清除，不要简单使用 #if 0 将其注释掉。若是一些有用的功能代码块，在后续调试、开发过程还需要用到的，请将其用函数方式封装好，并详细注释。

C.5 排版

1) 规则——程序块要采用缩进风格编写，缩进为 4 个空格，不使用制表位(Tab)

说明：把源程序中的 Tab 字符转换成 4 个空格，一个缩进等级是 4 个空格，变量定义和可执行语句要缩进一个等级，函数的参数过长时，也要缩进。

2) 规则——相对独立的程序块之间、变量说明之后必须加空行

示例:

```
//如下例子不符合规范。
if (!valid_ni(ni))
{
// program code
```

```
...
}
repssn_ind = ssn_data[index].repssn_index;
repssn_ni = ssn_data[index].ni;

/*应如下书写*/
if (!valid_ni(ni))
{
/* program code*/
...
}

repssn_ind = ssn_data[index].repssn_index;
repssn_ni = ssn_data[index].ni;
```

3) **规则——多个短语句不允许写在同一行内，即一行只写一条语句**

示例:

```
/*不好的排版 */
int a = 5; int b= 10;

/* 较好的排版 */
int a = 5;
int b= 10;
```

4) **规则——在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符(如->)，后不应加空格。**

说明：采用这种松散方式编写代码的目的是使代码更加清晰。

在已经非常清晰的语句中没有必要再留空格，如括号内侧(即左括号后面和右括号前面)不需要加空格，多重括号间不必加空格，因为在 C 语言中括号已经是最清晰的标志了。在长语句中，如果需要加的空格非常多，那么应该保持整体清晰，而在局部不加空格。给操作符留空格时不要连续留两个以上空格。

示例:

(1) 逗号、分号只在后面加空格，示例：

```
int a, b, c;
```

(2) 比较操作符，赋值操作符“=”、 “+=”，算术操作符“+”、“%”，逻辑操作符“&&”、“&”，位域操作符“<<”、“^”等双目操作符的前后加空格，示例：

```
if (current_time >= MAX_TIME_VALUE)
a = b + c;
a *= 2;
a = b ^ 2;
```

(3)“!”、“~”、“++”、“—”、“&”(地址操作符)等单目操作符前后不加空格，示例：

```
p = 'a';          /* 内容操作"*"与内容之间 */
flag = !is_empty; /* 非操作"!"与内容之间 */
p = &mem;         /* 地址操作"&" 与内容之间 */
i++;          /* "++","--"与内容之间 */
```

(4)“->”、“.”前后不加空格，示例：

```
p->id = pid;     /* "->"指针前后不加空格 */
```

(5) if、for、while、switch 等与后面的括号间应加空格，使 if 等关键字更为突出、明显，示例：

```
 if (a >= b && c > d)
```

5) **建议——注释符(包括 ‘/*’ ‘*/’)与注释内容之间要用一个空格进行分隔**

说明：这样可以使注释的内容部分更清晰。

现在很多工具都可以批量生成、删除‘//’注释，这样有空格也比较方便统一处理。

6) **建议——源程序中关系较为紧密的代码应尽可能相邻**

7) **建议——水平留白**

说明：水平留白的使用因地制宜，但永远不要在行尾添加没意义的留白。添加冗余的留白会给其他人编辑时造成额外负担，因此，行尾不要留空格。如果确定一行代码已经修改完毕，请将多余的空格去掉。

附录 D

（规范性附录）
vxWorks BSP 测试规范

为验证龙芯开发板 BSP 是否完成了客户所规定的各项指标，并保证 vxWorks 系统运行时的稳定性，编写了本测试规范。本测试规范包括：功能测试、稳定性测试、性能测试、重点问题回归测试、VTS 测试等。

D.1 功能测试

测试目的

规范开发板I/O接口测试方案，保证开发板I/O接口的可用性

测试项目

1.VGA接口

2.AUDIO接口

3.PS/2 键盘接口

4.PS/2 鼠标接口

5.网络测试

6.U盘测试

7.USB键盘测试

8.USB鼠标测试

9.IDE硬盘电子盘接口

10.串口

D.1.1 VGA 接口

（a)、测试步骤

1.用VGA连线连接开发板与显示器VGA接口

2.启动vxWorks的图形系统windml的测试程序ugldemo

3.显示

（b)、检验标准

显示器输出画面无异常

D.1.2 AUDIO 接口

（a)、测试步骤

1.耳机和耳麦接入开发板AUDIO对应的输入输出接口

2.插入U盘，U盘里有要播放的音乐文件，还可以存储录入的音乐文件

2.启动vxWorks

3.播放音乐

4.输入音频录制函数开始录制

5.对着耳麦输入音频，然后播放验证

（b)、检验标准

1.播放音乐，声音正常且无杂音

2.可以成功录制音频

D.1.3 PS/2 键盘

（a）、测试步骤

1.PS2 接口连接PS2 键盘。键盘的生产厂家及型号无特殊要求。推荐使用HP、双飞燕这两款品牌

2.PS2 键盘在vxWorks字符模式和图形模式下分别进行验证

（b）、**检验标准**

1.vxWorks启动后，字符模式下，分别测试单个键、多个组合键敲键盘是否输出正确，是否会使vxWorks死机

2.启动图形模式的测试用例uglDemo，测试敲键盘的任意键时，是否有中断响应导致uglDemo图形变化。

D.1.4 PS/2 鼠标

（a）、测试步骤

1.PS2 接口连接PS2 鼠标。鼠标的生产厂家及型号无特殊要求。

2.PS2 鼠标需要在图形系统windml下验证

（b）、**检验标准**

1.vxWorks启动后，启动图形系统的测试用例winDemo，测试鼠标单击、双击、右击、上下左右移动、滑动滚轮等功能都可以正常使用。

D.1.5 网络测试

（a）、测试步骤

1.用百兆交换机连接主机和主板

2.vxWorks启动后，ping指定主机ip

3.ftp功能测试

（b）、检验标准

1.查看ping的打印语句判断通信是否正常。

2. 使用ftp命令上传文件到目标机的电子盘，再下载在主机上，并比对获取到的文件与原文件是否一致

D.1.6 U 盘测试

（a）、测试步骤

1.在USB接口插上U盘

（b）、检验标准

1.vxWorks下可以列出U盘设备符号，查看U盘上的文件，并执行创建文件、读文件、写文件操作

D.1.7 USB 键盘测试

（a）、测试步骤

1.USB接口连接USB键盘

2.启动vxWorks，在字符模式下使用USB键盘，分别测试单个键和多个组合键是否可用，是否会导致系统死机

3.使用ugldemo测试USB键盘的中断是否能导致图形变化

（b）、检验标准

1.单个键功能可用，多个键组合使用不会导致系统死机

2.USB键盘的中断能导致图形变化

D.1.8 USB 鼠标测试

（a）、测试步骤

1.USB接口连接USB鼠标

2.启动vxWorks，在图形模式下执行winDemo测试鼠标

(b)、检验标准

1. 启动图形系统的测试用例winDemo，测试鼠标单击、双击、右击、上下左右移动、滑动滚轮等功能都可以正常使用。

D.1.9 IDE 硬盘电子盘接口

(a)、测试步骤

1.IDE接口连接硬盘，电子盘

(b)、检验标准

1.vxWorks shell下可以列出IDE设备符号，可以查看盘上的文件，并创建文件、读文件、写文件。

D.1.10 串口

(a)、测试步骤

1.串口线连接主板与主机的串口

2.编译vxWorks镜像时配置输出shell在串口

3.打开主机的串口超级终端

(b)、检验标准

1.vxWorks启动后从串口shell可以输入命令

2.使用测试程序test_serial测试输入输出是否正常

D.2 稳定性测试

测试目的

确定系统稳定运行需要哪些条件，测试某些设备长期运行的稳定性，检查设备驱动长期运行及在复杂环境下运行是否出错。可以测单个设备的稳定性，也可以测多个设备同时运行的稳定性。

测试项目

1.网卡驱动的内存边界测试

2.ATA驱动的内存边界测试

3.USB协议栈的兼容性测试

4.地址不对齐异常处理测试

5. 浮点异常处理测试

6. 网络协议压力测试

7.网络传输文件后压缩比较综合测试

8.系统整体稳定性测试

9.大文件（硬盘、U盘、网络压力测试）传输压力测试

10.内存压力测试

11.系统循环重启测试（3*3）

D.2.1 网卡驱动的内存边界测试

(a)、测试步骤

1.编译前，在网卡驱动函数synInstInit2 前端，分别加入多种malloc（0x10000000）语句，其中分配的内存大小从小到大可变，依次为 0x08000000, 0x10000000, 0x18000000, 0x20000000, 0x28000000, 0x30000000。编译vxWorks三方库和镜像（编译配置不允许包含INCLUDE_RTP，该选项需要一定的内存开销，可能会导致内存不够用），然后启动vxWorks

2. 使用网线连接主板的网口和主机

(b)、检验标准

1.进入vxWorks的shell，使用ping语句查看网络是否能通。

D.2.2 ATA 驱动的内存边界测试

(a)、测试步骤

1.编译前，在ATA驱动函数ataDrv前端，分别加入多种malloc（0x10000000）语句，其中分配的内存大小从小到大可变，依次为0x08000000, 0x10000000, 0x18000000, 0x20000000, 0x28000000, 0x30000000。编译vxWorks镜像（编译配置不允许包含INCLUDE_RTP，该选项需要一定的内存开销，可能会导致内存不够用），启动vxWorks。

2. 将电子盘插到主板上对应IDE电子盘接口

(b)、检验标准

1.进入vxWorks的shell，查看电子盘盘符，其中文件可读。

D.2.3 USB 协议栈的兼容性测试

(a)、测试步骤

1. USB接口连接U盘，使用金士顿、索尼、东芝、JetFlash这几款品牌

2. USB键盘连接USB键盘，联想品牌

3. 启动vxWorks

4. 将联想品牌USB键盘换成双飞燕品牌

(b)、检验标准

1.vxWorks的shell，查看U盘盘符，这四个U盘的分区都可以被识别，文件可读。

2.在vxWorks的shell， 联想和双飞燕USB键盘都可用。

D.2.4 地址不对齐异常处理测试

(a)、测试步骤

1.启动vxWorks

(b)、检验标准

1.在vxWorks的shell，执行testLw，查看打印结果是否测试通过。

D.2.5 浮点异常处理测试

(a)、测试步骤

1.启动vxWorks

(b)、检验标准

1.在vxWorks的shell，执行testFloat，查看打印结果是否测试通过。

D.2.6 网络协议压力测试

(a)、测试步骤

1.用百兆交换机连接两块主板，使用tcp测试程序tcpServer，tcpClient，长时间测试。

2.用百兆交换机连接两块主板，使用udp测试程序udpServer，udpClient，长时间测试。

3.用百兆交换机连接两块主板，使用tcp测试程序blasterTCPvx，blasteeTCPvx，长时间测试，测试完后计算丢包率。

4.用百兆交换机连接两块主板，使用udp测试程序blasterUDPvx，blasteeUDPvx，长时间测试，测试完后计算丢包率。

(b)、检验标准

测试过程中通信无中断

D.2.7 网络传输文件后压缩比较综合测试

(a)、测试步骤

1.启动主板的vxWorks

2.用千兆网线连接主板和主机

3.在vxWorks的shell下执行拷贝操作，将主机上的文件拷贝到vxWorks的U盘/电子盘等块设备中，拷贝两次

4. 将拷贝过来的两个文件进行压缩，然后比较两个文件内容是否一致

5. 跳转至 2，循环操作

(b)、检验标准

1.文件不一致会报错，否则一直循环进行该测试

D.2.8 系统整体稳定性测试

(a)、测试步骤

1.用百兆网线连接主板和主机

3.把电子盘插到对应的接口

4.耳机插到指定的音频接口

5.U盘插到USB口

6.启动vxWorks

7.在vxWorks的shell下执行拷贝操作，将主机上的文件拷贝到vxWorks的U盘/电子盘等块设备中，拷贝两次

8. 将拷贝过来的两个文件进行压缩，然后比较两个文件内容是否一致

9. 跳转至 7，循环操作

10.循环播放U盘里的音频文件

11.循环播放ugldemo

(b)、检验标准

测试过程中，所有功能并行执行并无异常出现

D.3 性能测试

测试目的

确定 IO 设备的性能，如读写速度、访问带宽等。

测试项目

1.使用iperf测试最大TCP和UDP带宽、延时

2.使用Linux命令测网络带宽

3.测IDE硬盘/电子盘的读写速度

4.内存性能测试stream

D.3.1 使用 iperf 测试最大 TCP 带宽和 UDP 带宽、抖动

1.启动两个主板的vxWorks

2.进入vxWorks的shell，指定带宽测试UDP抖动，服务器端输入命令 iperf “-u -s -i 1” ，客户端输入命令 iperf“-c serverip -u -i 1 -t 10 -b 1M”

3. 重新启动两个主板的vxWorks

4.进入vxWorks的shell，测试TCP带宽，服务器端输入命令 iperf “-s -i 1 -w 1M”，客户端输入命令iperf “-c serverip -i 1 -t 10 -w 1M”

D.3.2 使用 Linux 命令测网络带宽

1.启动主板的vxWorks

2.用千兆网线连接主板和Linux主机

2.在Linux主机的命令行下输入命令 Ping -s 65500 -i 0.0001 主板ip

3. 在Linux主机的另外一个命令行下输入命令ifstat查看主板与Linux主机间的网络带宽

D.3.3 测 IDE 硬盘/电子盘的读写速度

1.将电子盘插到主板的IDE电子盘接口，将IDE硬盘插到底板的IDE硬盘接口

2.启动vxWorks

3.输入命令 testAtaWrite “/ata0a/1”, 30 测试写速度，输入命令estAtaRead “/ata0a/1”, 10 测试读速度

4. 输入命令SpeedTest同样测试电子盘写速度

5. 输入命令testAtaWrite “/ata1a/1”, 30 测试硬盘写速度，输入命令testAtaRead “/ata1a/1”, 10 测试硬盘读速度

D.3.4 内存速度测试

1.编译版本为 5.9 的stream程序到vxWorks内核代码中

2.启动主板的vxWorks

3.在vxWorks的shell下执行函数调用stream

D.4 重点回归测试

测试目的

针对以前vxWorks系统曾经出现过的问题，重新执行当时复现问题所做的测试程序，看改正代码后问题是否还能复现，确保以前的问题已经得到解决

测试项目

1.百兆环境或千兆环境百兆网线下gmac不好用的问题，问题表现为系统启动后到PMON下网络不通，重插拔网线或重启交换机后才通。对该问题进行强化测试，主板通过四根线直连主机，PMON下自动加载镜像并启动，启动后到vxWorks下ping主机，若ping不通则说明问题复现，否则重启，若哪次PMON下无法加载镜像，也说明问题复现。

2.用域名攻击程序dnsflood和定时发包程序udpTest综合测试，用这样的组合曾经复现过GMAC网卡与CPU之间的存储一致性问题，重复该测试看现在问题是否已经得到修整。

D.4.1 千兆网线测 gmac

(a)、测试步骤

1. 用千兆网线连接主机和主板
2. 启vxWorks
3. ping命令测主机和主板之间网络是否通畅

(b)、检验标准

1.可以ping通

D.4.2 百兆环境测 gmac

(a)、测试步骤

1. 用百兆交换机和网线连接主机和主板
2. 启vxWorks
3. ping命令测主机和主板之间网络是否通畅

(b)、检验标准

1.可以ping通

D.4.3 百兆四根线测 gmac

(a)、测试步骤

1. 用百兆四根线网线连接主机和主板
2. 启vxWorks
3. ping命令测主机和主板之间网络是否通畅

(b)、检验标准

1.可以ping通

D.4.4 十兆网络环境测 gmac

（a)、测试步骤

1. 用十兆交换机连接主机和主板
2. 启vxWorks
3. ping命令测主机和主板之间网络是否通畅

（b)、检验标准

1.可以ping通

D.4.5 reboot 后的网络测试

（a）、测试步骤

1. 在PMON下设置环境变量，set ifconfig syn1:ip地址，set al tftp://主机ip/vxWorks

2. 手动重启PMON 100 次，每次起来后测试网络是否可用。

3. 在vxWorks的代码usrAppInit中要求每次起来之后都ping主机测试网卡syn0 是否可用，不可用则打印报错，可用则直接reboot重启，如此反复

（b）、检验标准

1.若网络在PMON和vxWorks都一直通畅表示该问题已经完全解决

D.4.6 域名攻击程序和定时发包程序综合测试

（a）、测试步骤

1. 编译域名攻击程序dnsflood与定时收发包程序udpTest到vxWorks镜像中

2. 连接主板和windows主机

2. 启动vxWorks

3. 在vxWorks的shell输入dnsflood的函数调用及udpTest，同时打开主机上的udpTest发包工具，开始长时间压力测试

（b）、检验标准

1.若出错，则主机上的发包工具无法成功发出包

D.5 VTS 测试

D.5.1 测试目的

BSP VTS 主要测试 vxWorks 内核与 BSP 包的接口是否完善，上层协议所需要的 API 是否可用。

D.5.2 测试项目

BSP VTS 测试分为 3 个方面： 基本功能测试、VxBus 测试和 OS 接口测试。具体测试模块如下：

基本功能测试部分:

- tmBspApi 检测BSP 的基本API
- tmAuxClock 测试辅助时钟的功能性
- tmModel 测试sysModel函数
- tmNvRam 非易失性RAM测试
- tmSysClock 测试系统的时钟功能
- tmTimeStamp 检测BSP时间戳定时器的功能

VxBus测试部分:

- vxbTest vxBus接口测试
- vxbTestParamSys vxBus参数子系统测试
- vxbTestSio vxBus模式的串口设备测试
- vxbTestDmaBuffer vxBus的dma buffer系统
- vxbTestDmaLib vxBus的dma库测试
- vxbTestTimerLib vxBus时钟库测试

- vxbTestTimerDriver vxBus兼容的时钟驱动
- vxbTestAccessVirtDriver 访问虚拟驱动
- vxbTestAccessPciDriver 访问PCI
- OS接口测试部分
- tmSockLib tcp连接测试
- tmSpinLockLib自旋锁测试
- tmHookLib vxWorks的钩子函数测试
- tmPipeDrv2 管道的测试
- tmVxAtomicLib 原子变量API的测试
- tmIsrLib 测试中断连接和断开
- tmTaskShow 任务信息测试
- tmPthreadLib 有关线程API的测试
- tmSyscallLib 系统调用相关测试
- tmTaskLib 测试任务的API（激活，挂起，恢复，优先级设置，保护被调用任务，任务控制块等）
- tmTaskInfo 任务的选项，名字，任务的状态（是否准备运行）的测试
- tmErrnoLib 错误码的测试
- tmAnsiTime 测试系统时间和实时时钟同步的API的功能
- tmKernelLib 内核库测试
- tmSemMLib 互斥信号量的测试
- tmHashLib 散列表API的测试
- tmTaskCreateLib 任务创建（对于不同的option）和删除的API测试
- tmMacroTest 宏的测试（ANSI，浮点运算，errno宏，ASSERT宏，offsetof宏以及CPUSET宏）
- tmAnsiCtype 对于数字，大小写字母的判断，大小写字母的转换，标点符号或特殊符号，是否为可打印字符的判断
- tmPmLib 测试重启过程中对特殊内存区的保护
- tmVmTest 测试vmBaseLib/vmLib的功能
- tmWindStressTest 内核压力测试
- tmPgMgrLib 页面管理测试（pgMgrCreate()，pgMgrDelete()等）
- tmShmLib 使用正确的参数测试shm_open
- tmPxTraceEventId 测试一个进程使用到的所有事件
- tmIntArchLib 架构无关的中断测试
- tmTaskVarLib 任务变量有关的API的测试
- tmSemRWLib 读写信号量的测试
- tmWorkQLib 工作队列的测试
- tmCacheDmaMallocLib 测试cacheDmaMalloc
- tmSigLib 信号集API的测试和使用
- tmSemCLib 计数器信号量的测试
- tmLoginLib 测试login
- tmTaskHookLib 任务钩子函数的测试

- tmPingLib ping测试
- tmHeapStressTest内存堆压力测试
- tmTaskRestartLib 任务重启的测试
- tmSemBLib 二进制信号量的测试
- tmPriInherit 互斥信号量有关的测试
- tmEnvLib 环境变量的设置
- tmIoBasicLib 文件的打开关闭，读写等操作
- tmWdCreateLib 看门狗定时器API的测试
- tmAdrSpaceLib 检测系统地址空间，测试创建，分配和删除是否正常
- tmMemPartLib 内存管理API测试
- tmMemDrvLib 不同的参数设置partition的内存
- tmAnsiSetjmp 测试setjmp()和longjmp()函数
- tmMemLib MemPartLib的增强库测试
- tmCacheLib cache操作API测试
- tmPipeDrv 任务间通信，管道的测试
- tmPoolLib 内存池（以不同的参数创建的内存池及其管理）
- tmSemMLibInline 互斥信号量的测试
- tmVxCpuLib vxCpu 库测试

D.5.3 测试方法

BSP VTS 的测试步骤和方法基本过程是:在开发板上加载 vxWorks 镜像，在 vxWorks 的 shell 环境下运行测试程序，保存测试输出，利用主机上 Workbench 中的工具对测试输出进行分析统计，获取测试结果。

具体步骤如下：

- 启动 PMON，并配置环境变量使其可以自动加载 vxWorks
  - set ifconfig syn0:172.17.107.180 设置板子 ip 地址
  - set a1 tftp://172.17.107.26/vxWorks 设置远程的 vxWorks 镜像地址
- 使用串口线连接开发板和主机，打开串口终端，设置将串口输出信息保存到文件 vxtest.log
- 在主机中启动 tftp 服务，设置好根目录，将 vxWorks 镜像放到 tftp 根目录设置下，在开发板上远程加载 vxWorks
- 在 vxWorks 的 shell 下运行 vxTestV2 “-em -v 4”，开始 VTS 测试,测试需要半小时到一个小时之间
- 将 文 件 vxtest.log 复 制 到 C:\WindRiver\workspace\2J3u\vxtest\logs\default\lx2g_mipsi64le.gnule.lx2g_mipsi64le.profile_default.lib.up.bspvts.run.log
- 运行 Workbench，在 Project Explorer 视图中相应的工程名上右击鼠标，选择 open workbench development shell 菜单，进入 host 端的 shell 环境
- 在 shell 下输入命令可得到 html 格式的测试报告

测试报告列出四种类型的统计结果

ABORT:表示测试结果可以忽略；

FAIL：表示测试没有通过；

SKIP：表示因为测试条件问题而没有进行测试，比如 testcase 描述段中有 disable 标签；

EXCLUDED：表示测试条件不满足而没有进行测试，比如 testcase 描述段中的依赖模块没有而跳过；

其他未列出的测试项均表示测试通过。

附录 E

（资料性附录）

windML 及图形驱动介绍

E.1 windML 介绍

E.1.1 媒体库概述

风河 Windml 媒体库提供了在嵌入式 vxWorks 系统中运行多媒体程序的框架，媒体库设计提供了包括了图形，视频，音频等媒体程序接口，及驱动开发框架。

媒体库的组成主要包括了两个主要的部分：

- 软件开发套件(SDK, Software Development Kit)
- 驱动开发套件(DDK，Driver Development Kit)

Windml 媒体库的组成如图 E.1 所示：

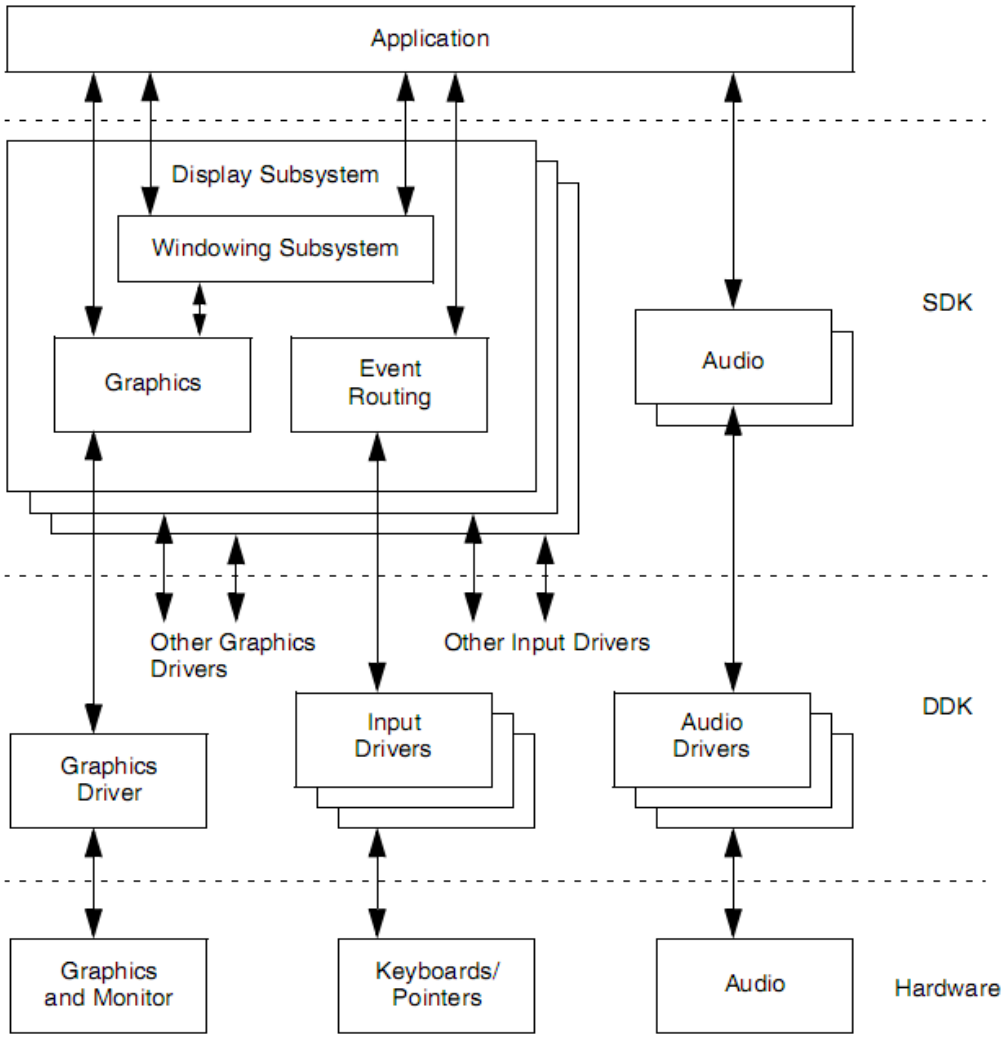

图 E.1 windML 架构图

软件开发套件：软件开发包用来开发应用程序，提供了图形 API，输入处理，多媒体，字体，内存管理等。

驱动开发套件：驱动开发套件提供了驱动的开发框架，便于快速实现驱动

E.1.2 图形设备驱动

图形硬件：通常称为显卡或 GPU，是一种专门负责图形显示的专用硬件，通常开发图形硬件驱动

帧缓冲(frame buffer)：提供了显示图像的内存，组织一系列的像素，每个像素都代表了颜色和在屏幕上的位置

内存控制器(memory controller)：定义了帧缓冲访问内存的时序

图形处理器(graphic process unit)：专门用来做渲染的图形处理器

显示接口（display interface)：显示时序：各种分辨率的行，场频率的信号，颜色表： 存储了各种颜色

E.1.3 图形驱动概述

图形驱动是用来驱动图形硬件的驱动程序，在 2D 驱动程序的实现上，包括了下面的一些组件：

a) 通用驱动

图形 API 的软件实现，提供了经过优化的基本图形绘制能力

b) 硬件抽象层

隐藏了不同硬件的细节，增加了可移植性

c) 内存管理

提供了从内存池了分配内存的机制，如果 2D API 层需要和驱动层共享内存则提供了 UGL_CALLOC, UGL_MALLOC, UGL_REALLOC 这些宏来进行内存，而不是直接调用 malloc 来进行内存分配如果需要让处理器(CPU)和图形设备（GPU）共享内存，则需要使用 uglDeviceMemAlloc() uglDeviceMemFree（）来进行内粗分配

2D 驱动层次如图 E.2 所示：

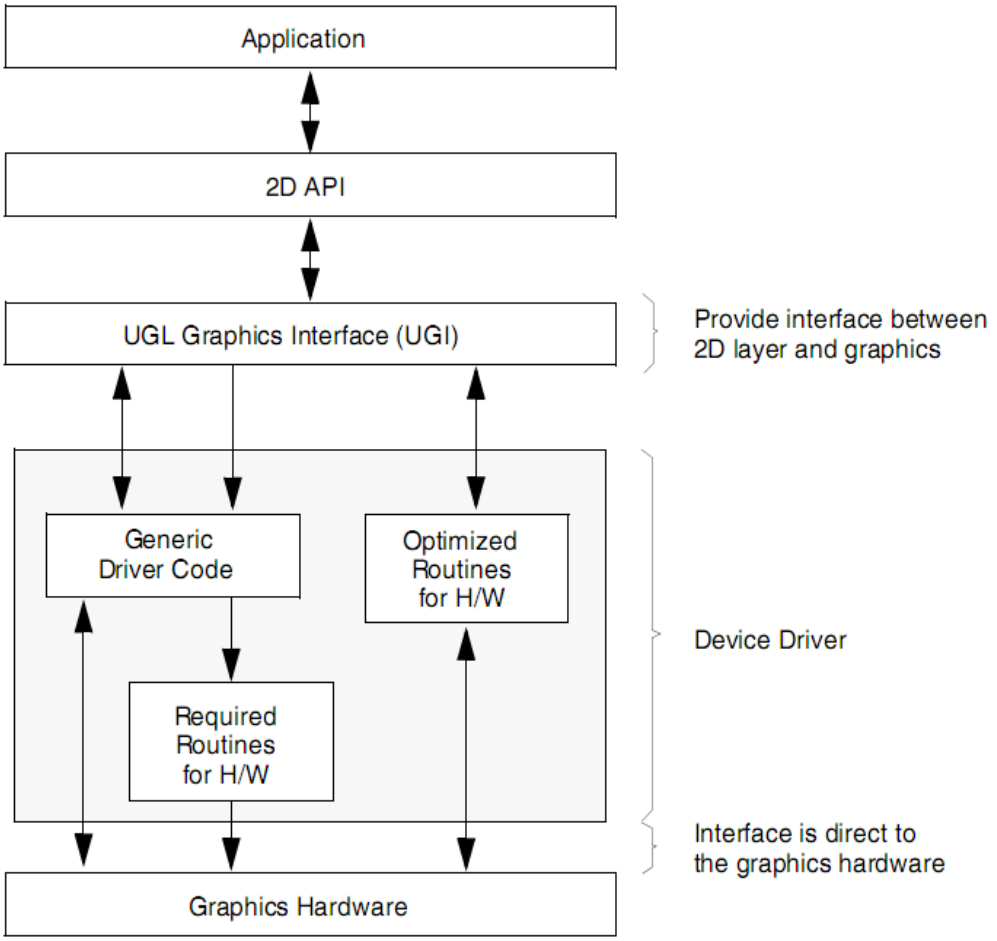

图 E.2 2D 驱动层次

E.2 开发流程

Windml 下驱动开发流程如图 5 所示：

1) 创建一个数据库入口

2) 参考一个类似的已有的驱动，将驱动放到 windml 中，添加 makefile

3) 调试修改

4）测试

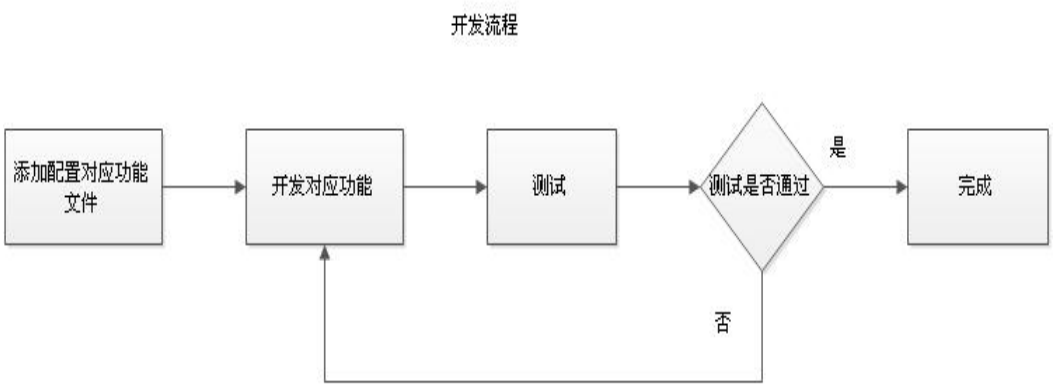

图 E.3 windML 驱动开发流程图

附录 F

（资料性附录）

vxWorks 设备驱动参考范例

F.1 40wrsample.cdf 文件范例

```
Component DRV_DEMO_WRSAMPLE {
        NAME            sample VxBus driver
        SYNOPSIS        sample 3rd-party VxBus driver provided by Wind River
        REQUIRES        INCLUDE_VXBUS \
                        INCLUDE_PLB_BUS
        MODULES         wrsample.o
        PROTOTYPE       void wrsampleRegister(void);
        INIT_RTN        wrsampleRegister();
        INIT_AFTER      INCLUDE_PLB_BUS
        _INIT_ORDER     hardWareInterFaceBusInit
        _CHILDREN       FOLDER_DRIVERS
}
```

F.2 wrsample.dc 文件范例

```
#ifdef DRV_DEMO_WRSAMPLE
IMPORT void wrsampleRegister(void);
#endif /* DRV_DEMO_WRSAMPLE */
```

F.3 wrsample.dr 文件范例

```
#ifdef DRV_DEMO_WRSAMPLE
    wrsampleRegister();
#endif /* DRV_DEMO_WRSAMPLE */
```

F.4 wrsample.h 文件范例

```
#ifndef __INCwrsampleh
#define __INCwrsampleh

/* includes */

#include <vxWorks.h>
#include <vxBusLib.h>
#include <hwif/vxbus/vxBus.h>

```

```
/* defines */

#define WRSAMPLE_LOG_TO_CONSOLE         1
#define WRSAMPLE_GET_SAMPLE_FUNCS       2
#define WRSAMPLE_SET_SAMPLE_FUNCS       3

/* typedefs */

struct wrsampleDriverControl
    {
    int           (*output) (char * fmt,
                  int a1, int a2, int a3, int a4, int a5, int a6);
    STATUS        (*logToConsole) (VXB_DEVICE_ID instID);
    VXB_DEVICE_ID instID;
    };
#endif /* __INCwrsampleh */
```

F.5 wrsample.c 文件范例

```
/* includes */
#include <vxWorks.h>
#include <stdio.h>           /* for printf() */
#include <logLib.h>          /* for logMsg() */
#include <string.h>          /* for strcmp() */

#include <vxBusLib.h>            /* for VXB_DEVICE_ID */
#include <hwif/vxbus/vxBus.h>
#include <hwif/util/hwMemLib.h>       /* for hwMemAlloc() */
#include <hwif/vxbus/hwConf.h>        /* for devResourceGet() and
                       * hcfDeviceGet()
                       */
#include <driverControl.h>       /* for struct vxbDriverControl */

#include "wrsample.h"

/* defines */

/* debug */

#ifdef WRSAMPLE_DBG_ON

```

```
39  /*
40   * NOTE: printf() and logMsg() debugging cannot be used before the
41   * call to wrsampleInstConnect().  Any use before that may cause the
42   * system to crash before finishing the boot process.  To debug
43   * the probe routine, init routine, and init2 routine, either
44   * use a hardware debugger, move the specified functionality to
45   * wrsampleInstConnect(), or manually call the driver registration
46   * routine after boot.
47   */
48
49  int wrsampleDebugLevel = 0;
50
51  #ifndef WRSAMPLE_DBG_MSG
52  #define  WRSAMPLE_DBG_MSG(level,fmt,a,b,c,d,e,f)  if  (  wrsampleDebugLevel  >=  level  )
    logMsg(fmt,a,b,c,d,e,f)
53  #endif /* WRSAMPLE_DBG_MSG */
54  #undef LOCAL
55  #define LOCAL
56
57  #else /* WRSAMPLE_DBG_ON */
58
59  #define WRSAMPLE_DBG_MSG(level,fmt,a,b,c,d,e,f)
60
61  #endif /* WRSAMPLE_DBG_ON */
62
63  /* typedefs */
64
65  struct wrsamplepDrvCtrl
66      {
67      int customMethodCount;
68      int (*output)(char * fmt, int a1, int a2, int a3, int a4, int a5, int a6);
69      };
70
71  /* locals */
72
73  LOCAL void wrsampleInstInit(VXB_DEVICE_ID pInst);
74  LOCAL void wrsampleInstInit2(VXB_DEVICE_ID pInst);
75  LOCAL void wrsampleInstConnect(VXB_DEVICE_ID pInst);
76
```

```
77  /* vxBus 架构的驱动都会有这三个函数资源 */
78  LOCAL struct drvBusFuncs wrsampleFuncs =
79      {
80      wrsampleInstInit,         /* devInstanceInit */
81      wrsampleInstInit2,        /* devInstanceInit2 */
82      wrsampleInstConnect       /* devConnect */
83      };
84
85  LOCAL struct vxbDevRegInfo wrsampleDevRegistration =
86      {
87      NULL,                 /* pNext */
88      VXB_DEVID_DEVICE,     /* devID */
89      VXB_BUSID_PLB,        /* busID = PLB */
90      VXB_VER_4_0_0,        /* vxbVersion */
91      "wrsample",           /* drvName */ /*设备名称*/
92      &wrsampleFuncs,       /* pDrvBusFuncs */
93      NULL,                 /* pMethods */
94      NULL,                 /* devProbe */
95      NULL                  /* pParamDefaults */
96      };
97
98  LOCAL STATUS wrsampleDriverCtrl
99      (
100     VXB_DEVICE_ID               instId,
101     struct vxbDriverControl *   pControl
102     );
103
104 /*
105  * This sample represents a custom driver class.  Normal VxBus
106  * driver methods are not available, but we can use the generic
107  * {driverControl}() method to accomplish what we need to.
108  * Therefore, we publish only that method.
109  *
110  * Note that the structures used by the {driverControl}() method
111  * must be defined in a per-driver or per-application header file,
112  * in addition to the driverControl.h header file.  In this case,
113  * the header file is target/3rdparty/windriver/sample/wrsample.h
114  * and applications are responsible for ensuring that they include
115  * that file when they need to interact with the instance.
```

```
116  */
117
118 LOCAL device_method_t wrsample_methods[] =
119     {
120     DEVMETHOD(driverControl, wrsampleDriverCtrl),
121     { 0, 0}
122     };
123
124 /* ------------------------------------------------------------ */
125
126 /* local defines       */
127
128 /* driver functions */
129
130 /*******************************************************************************
131 *
132 * wrsampleRegister - register wrsample driver
133 *
134 * This routine registers the wrsample driver and device recognition
135 * data with the vxBus subsystem.
136 *
137 * RETURNS: N/A
138 *
139 * ERRNO
140 */
141 /*设备注册函数*/
142 void wrsampleRegister(void)
143     {
144     vxbDevRegister((struct vxbDevRegInfo *)&wrsampleDevRegistration);
145     }
146
147 /*******************************************************************************
148 *
149 * wrsampleInstInit - initialize wrsample device
150 *
151 * This is the wrsample initialization routine.
152 *
153 * RETURNS: N/A
154 *
```

```
155 * ERRNO
156 */
157 /*初始化函数*/
158 LOCAL void wrsampleInstInit
159     (
160     VXB_DEVICE_ID pInst
161     )
162     {
163     struct wrsamplepDrvCtrl * pDrvCtrl;
164
165     /* to store the HCF device */
166
167     HCF_DEVICE *        pHcf = NULL;
168
169     /* check for vaild parameter */
170
171     if (pInst == NULL)
172         return;
173
174     /* allocate the memory for the structure */
175
176     pDrvCtrl = (struct wrsamplepDrvCtrl *)
177         hwMemAlloc (sizeof (struct wrsamplepDrvCtrl));
178
179     /* check if memory allocation is successful */
180
181     if (pDrvCtrl == NULL)
182         return;
183
184     /* get the HCF device from vxBus device structure */
185
186     pHcf = hcfDeviceGet (pInst);
187
188     /* if pHcf is NULL, no device is present */
189
190     if (pHcf == NULL)
191         return;
192
193     /* retrieve the integer value for count */
```

```
194
195     /*
196      * resourceDesc {
197      * The count resource specifies the initial start count.
198      * If not specified, 0 will be set by default. }
199      */
200
201     if (devResourceGet(pHcf,"count", HCF_RES_INT,
202     (void *)&pDrvCtrl->customMethodCount) != OK)
203         pDrvCtrl->customMethodCount = 0;
204
205     /* retrieve the integer value for output function */
206
207     /*
208      * resourceDesc {
209      * The output resource specifies the output function.
210      * If not specified, logMsg() will be set by default. }
211      */
212
213     if (devResourceGet(pHcf,"output", HCF_RES_ADDR,
214     (void *)&pDrvCtrl->output) != OK)
215         pDrvCtrl->output = logMsg;
216
217     /* publish methods */
218
219     pInst->pMethods = &wrsample_methods[0];
220
221     /* per-device init */
222
223     pInst->pDrvCtrl = pDrvCtrl;
224
225     }
226
227 /*******************************************************************************
228 *
229 * wrsampleInstInit2 - initialize wrsample device
230 *
231 * This is seconde phase initialize routine for VxBus driver.
232 *
```

```
233 * RETURNS: N/A
234 *
235 * ERRNO
236 */
237 /*这个初始化函数可以根据具体驱动添加相应的代码*/
238 LOCAL void wrsampleInstInit2
239     (
240     VXB_DEVICE_ID pInst
241     )
242     {
243     }
244
245
246 /*******************************************************************************
247 *
248 * wrsampleInstConnect - VxBus connect phase routine for wrsample driver
249 *
250 * This is connect phase routine.
251 *
252 * RETURNS: N/A
253 *
254 * ERRNO: not set
255 */
256
257 LOCAL void wrsampleInstConnect
258     (
259     VXB_DEVICE_ID pInst
260     )
261     {
262     WRSAMPLE_DBG_MSG(100,"wrsampleInstConnect() called\n", 1,2,3,4,5,6);
263     }
264
265 /*******************************************************************************
266 *
267 * wrSampleLogToConsole - Display message on console for instance
268 *
269 * This routine displays a message on the console for the specified
270 * instance.
271 *
```

```
272 * RETURNS: N/A
273 *
274 * ERRNO: not set
275 */
276
277 STATUS wrSampleLogToConsole
278     (
279     VXB_DEVICE_ID instID
280     )
281     {
282     struct wrsamplepDrvCtrl * pDrvCtrl;
283
284     if ( instID == NULL )
285         return(ERROR);
286
287     pDrvCtrl = instID->pDrvCtrl;
288
289     if ( pDrvCtrl == NULL )
290         return(ERROR);
291
292     (*pDrvCtrl->output)("wrsample instance at 0x%08x, call count = %d\n",
293             (int)instID, pDrvCtrl->customMethodCount, 3,4,5,6);
294
295     pDrvCtrl->customMethodCount++;
296
297     return(OK);
298     }
299
300 #ifdef WRSAMPLE_DBG_ON
301
302 /*******************************************************************************
303 *
304 * wrsamplepDrvCtrlShow - show pDrvCtrl for sample driver
305 *
306 * This routine displays a message about pDrvCtrl of the specified instance.
307 *
308 * RETURNS: N/A
309 *
310 * ERRNO: not set
```

```
311 */
312
313 int wrsamplepDrvCtrlShow
314     (
315     VXB_DEVICE_ID pInst
316     )
317     {
318     printf("pDrvCtrl @ 0x%08x\n", pInst->pDrvCtrl);
319
320     return(0);
321     }
322
323 #endif /* WRSAMPLE_DBG_ON */
324
325 /*******************************************************************************
326 *
327 * wrsampleDriverCtrl - handle {driverControl}() method calls
328 *
329 * This routine handles driverControl method call from application layer.
330 *
331 * RETURNS: ERROR or OK
332 *
333 * ERRNO: not set
334 */
335
336 LOCAL STATUS wrsampleDriverCtrl
337     (
338     VXB_DEVICE_ID               instId,
339     struct vxbDriverControl *   pControl
340     )
341     {
342     struct wrsampleDriverControl * pSampleCtrl;
343     struct wrsamplepDrvCtrl * pDrvCtrl;
344
345     /* check whether this is for us */
346
347     if ( instId == NULL || pControl == NULL )
348         return(ERROR);
349     if ( strcmp(pControl->driverName, "wrsample" ) != 0 )
```

```
350         return(ERROR);
351
352     /* get pDrvCtrl from VxBus instance */
353
354     pDrvCtrl = instId->pDrvCtrl;
355     pSampleCtrl = pControl->drvCtrlInfo;
356
357     /* switch on command to run */
358
359     switch (pControl->drvCtrlCmd)
360         {
361         case WRSAMPLE_LOG_TO_CONSOLE:
362             /* run console log method */
363             wrSampleLogToConsole(instId);
364             break;
365
366         case WRSAMPLE_GET_SAMPLE_FUNCS:
367             /* retrieve functions */
368             pSampleCtrl->logToConsole = wrSampleLogToConsole;
369             pSampleCtrl->instID = instId;
370             pSampleCtrl->output = pDrvCtrl->output;
371             break;
372
373         case WRSAMPLE_SET_SAMPLE_FUNCS:
374             /* set display output function */
375             if ( pSampleCtrl->output == NULL )
376                 pDrvCtrl->output = logMsg;
377             else
378                 pDrvCtrl->output = pSampleCtrl->output;
379             break;
380
381         default:
382             return(ERROR);
383         }
384
385     return(OK);
386     }
```