摘要

随着时代的进步，我国航天事业迅速发展，对星载计算机的需求也随之增长，性能要求同样越来越高。因此，设计高效可靠的星载计算机硬件平台，并使之支持实时嵌入式操作系统是本文的研究目的。

本文首先介绍了课题组研制的以PowerPC处理器为核心，具备SDRAM、Flash以及FPGA的星载嵌入式计算机最小系统硬件平台，在分析硬件平台配置的基础上，设计并实现了VxWorks嵌入式操作系统的BSP（Board Support Package，板级支持包），具体工作包括硬件平台的上电初始化配置，相关硬件驱动程序的编写及修改，对闪存文件系统(TrueFFS)的支持与配置，以及系统引导方式以及系统调试通道的实现等。本文还给出了系统可靠性设计方案，实现SDRAM的Parity/ECC功能，使用RS编码冗余的方式提高Flash的可靠性。

本文使用CodeWarrior调试软件对系统软硬件进行调试与测试，结果表明系统性能指标达到了设计要求，工作良好。

关键字： 星载计算机　PowerPC　VxWorks　BSP

Abstract

With the great progress of our aerospace industry, the demand for on-board computer with high performance is growing rapidly. The purpose of this thesis is to design a highly efficient and reliable on-board computer hardware which can support real-time operating system.

The hardware platform of the on-board embedded computer minimum system, which has SDRAM, Flash, FPGA and PowerPC processor as its core, is introduced in this thesis. The focus of this article is the design and implementation of the BSP on the embedded operating system VxWorks. The main work includes the initial configuration of hardware, modification of related hardware driver, support and configuration of TrueFFS, booting methods of operating system, configuration of system debugging channel and so on. A reliable design of the system which includes the implementation of Parity/ECC on SDRAM and RS coding on Flash is given in this article.

The hardware and software of the system are tested by CodeWarrior. The testing result shows that the system designed in this thesis reaches the indicators and works normally.

Keyword: On-board Computer PowerPC VxWorks BSP

目录

第一章 绪论

1.1 研究背景及意义

1957 年 10 月 4 日第一颗人造卫星“斯普特尼克 1 号”的升空，标志着人类自此进入了太空时代。随着航天技术的不断发展，卫星在通讯、遥感、侦查、气象观测等领域占据越来越重要的地位。

我国航天事业也随着 21 世纪的到来迅猛发展，载人航天圆满成功，探月工程稳步进行。对星载计算机的需求也随之增长，性能要求越来越高，尤其是自主控制、自主导航、信息处理等对星载计算机的处理能力是个考验，而卫星有效载荷日益向可成像化设备发展，增大了需要获取的信息量与信息种类，并且需要在轨进行数据处理、压缩和融合，这对星载计算机性能要求颇高，特别是高速、高吞吐率的有效载荷，对星载计算机的性能和实时处理能力是个更为苛刻的考验。

与此同时，星载计算机的性能也受到严重制约。在空间环境中，星载计算机需要在空间环境恶劣的辐射中工作，辐射影响主要分为总剂量积累效应(Total Ionizing Dose，TID)和单粒子效应(Single-Event Effects，SEE)两类[1]。总剂量积累效应是电子设备长期处于高强度辐射环境下而逐渐老化的一种效应，它会对电子系统及其器件造成许多不良影响（比如延长信号传播时延，降低时钟频率，降低门输出驱动能力，降低噪声容限，增加漏流等），从而导致系统性能低下且不可逆的退化，最终使得设备彻底无法工作。单粒子效应是指由单个高能粒子的影响，导致的电子器件状态改变，可粗分为单粒子翻转和单粒子锁定两种典型辐射效应，另外还包括单粒子多位翻转、单粒子栅破裂、单粒子烧毁和单粒子瞬态等[2]。

由于卫星的研制花销巨大，开发周期长，升空后不易维修，况且空间环境苛刻，所以要求星载计算机系统具有极高的稳定性，即使出现故障也能够检测和恢复。为此设计高速稳定的星载计算机硬件平台，运行实时稳定的嵌入式操作系统是星载计算机发展的趋势。

课题组研制了以 PowerPC 处理器为核心，具有 SDRAM、Flash 存储设备以及 FPGA 的嵌入式最小系统硬件平台，并在此基础上设计编写了 VxWorks 操作系统的 BSP，使 VxWorks 操作系统正常运行。目的在于研究高性能的 PowerPC 处理器，使以其为核心的硬件平台，搭配 VxWorks 操作系统组成的最小系统能够稳定工作。

1.2 国内外研究和发展概况

目前对于星载计算机的设计，各国都有自己的方案。各大主流处理器在航天领域都有应用[3,4,5]。

X86 系列处理器在微小卫星中应用非常广泛，由于其体系架构深入人心，虽然速率比不上最新技术的处理器，但是对于星务管理需求是可以胜任的。X86 系列处理器在 NASA 的小型探测器卫星、英国 Surrey 的小卫星平台、中科院的“创新一号”和哈工大的“探索一号”卫星中均有应用。PowerPC 系列处理器具有很高的性能，美国所使用的最高端的星载计算机就是基于 PowerPC 体系结构的。SPARC 体系结构的处理器也越来越多的应用于航天系统，欧洲航天局(ESA，European Space Agency)已成功将基于 SPARC V7 的抗辐射加固处理器应用于航天工程，而 SPARC V8 也开始在航天工程应用。ARM 体系在卫星上也有一些应用，如 NASA 的 GPS 卫星、Surrey 的“SNAP-1”和清华“纳星一号”均使用了 StrongARM 系列处理器。德国 EADS 采用了 SuperH 处理器。另外，8/16 位 MCU 以及 DSP 在卫星上均有非常广泛的应用，如清华一号上曾使用了 C31 系列 DSP，而瑞典 2000 年发射的 Munin 纳型卫星就应用了 DSP 设计其星载计算机。

目前在国内外航空航天领域中使用较多的嵌入式实时操作系统主要有 VxWorks、RTEMS、Neutrino、eCOS、μC/OS-II 等。

VxWorks 以其实时性与稳定性著称，已经广泛应用于各种航天航空等对实时性要求严格的场合。RTEMS 是前美国军方研制的嵌入式操作系统（采用 ADA 语言编写），支持多处理器和实时操作，实时性能很好，曾经被美国国防部用来控制导弹等精密设备。Neutrino 操作系统是 QNX 公司推出的实时操作系统，也被应用于医药器材、互联网路由器到电讯系统、911 报警装置、处理控制系统、甚至航天系统[6]。eCOS(embedded Configuration Operating System)是由 RedHat 公司推出的小型实时操作系统，最低编译核心可小至 10K 的级别。清华大学“纳星一号”上使用了 μC/OS-II[7]。

1.3 论文的主要工作及内容安排

以基于PowerPC架构MPC8245处理器的嵌入式最小系统以及FPGA作为硬件平台，采用VxWorks操作系统作为软件平台，在此基础上开发及调试了支持VxWorks操作系统的BSP，并进一步提出了系统可靠性设计。全文共分六章，主要内容和结构安排如下：

第一章绪论。概述本课题的研究背景及意义，介绍目前国内外星载嵌入式计

算机发展情况，给出本文的内容安排。

第二章星载MPC8245最小系统设计。概述本课题的需求，介绍系统组成与设计方案以及硬件部分选型与设计。

第三章VxWorks及其BSP分析。介绍嵌入式实时操作系统VxWorks的概念与特点，板级支持包(BSP)的概念与组成及其集成开发环境Tornado。

第四章VxWorks BSP设计与实现。详细描述了VxWorks BSP的设计方案，以及处理器、SDRAM、Flash、串口等硬件的配置方法，最终实现了BSP设计，生成bootrom以及VxWorks映像。本章最后提出了系统可靠性设计，增强了系统的可靠性。

第五章VxWorks BSP调试与测试。介绍了利用CodeWarrior及其配套在线仿真调试器USB TAP调试VxWorks BSP的方法，通过对软硬件的调试解决了出现的问题，最终实现了本课题需求。

第六章结束语。简要总结本论文的工作成果，并对将来的进一步工作做了展望。

第二章 星载MPC8245最小系统设计

2.1 系统需求分析

由于星载嵌入式计算机运行环境特殊，所以需要弄清空间环境对计算机系统有何影响，并结合本课题的需求给出系统设计方案。

2.1.1 空间环境制约因素

空间环境是指距地球100km以外的外层空间。空间环境中存在空间辐射、极端温度、高真空等，其中空间辐射对电子器件影响较大。太空中分布着各式各样的带电粒子，与卫星相互作用会产生各种效应，主要有总剂量积累效应和单粒子效应。这两种效应会对星载设备的性能有一定程度的损害，可能导致卫星设备在轨工作的异常或故障。

总剂量积累效应是一种积累性损伤，它是由高能带电粒子多次反复轰击半导体电路引起的，多次轰击引起电离的电荷在半导体电路中累积。当达到一定程度后，就会使元器件性能下降甚至毁坏，比如导致MOS晶体管的阈值电压漂移、双极型晶体管的增益下降等等，从而影响星载设备的正常工作。单粒子效应是单个高能的质子或重离子导致的微电子器件状态的改变，可能造成改变器件逻辑状态的单粒子翻转(SEU)、单粒子锁定(SEL)以及单粒子击穿(SEB)。

单粒子翻转是电子元器件受到高能带电粒子的轰击，导致的PN结瞬时充放电现象，从而使器件逻辑状态发生翻转。它是瞬时故障，通过刷新可以恢复到原来的状态。单粒子锁定是电子元器件受到高能带电粒子的轰击导致半导体电路内部电极之间的短路，这会引起电流突然增大，此时外部信号无法改变电路的状态。它是可恢复故障，锁定时器件不会自动退出此状态，只能通过断电后重新上电才能恢复，但如果电路被锁定时间太长或电流过大，则可能成为永久性故障。单粒子击穿是电子元器件受到足够高能量的高能带电粒子的轰击而导致的半导体电路被击穿的现象。它是永久性故障，器件一旦被击穿就无法复原。其中，单粒子翻转是空间辐射效应的主要形式，是造成星载计算机故障的重要因素[8,9,10,11]。

2.1.2 系统设计要求

星载PowerPC嵌入式最小系统由PowerPC处理器加上FPGA构成的硬件平台

和嵌入式操作系统及其 BSP 组成。BSP 软件在存储器上固化运行，为系统硬件平台提供操作系统支持。根据课题需求，对硬件平台的功能及核心器件有如下要求：

(1) 硬件平台中所有元器件均为工业级；

(2) 处理器选用 PowerPC 系列处理器；

(3) 与处理器配套的 SDRAM 容量不小于 128MB，Flash 容量不小于 8MB；

(4) 需要有串口作为系统配套调试接口；

(5) 选用 Xilinx 公司工业级 FPGA 产品，门数应不少于 100 万门；

(6) 外部接口信号包括 LVDS、LVTTL，其中 LVTTL 双向 36 路，LVDS 双向 12 路，LVDS 收发器件考虑选用 TI 公司的 SN65LVDS31D，SN65LVDS32D；

(7) 系统所需的电源由外部提供，电源部分对应单独的接口和接插件，外部只提供 5V 电源，板上所需的 3.3V、1.5V 等通过相应的转换器件获取；

(8) FPGA 工作时钟考虑通过板上的晶振或者外部频综提供，工作时根据需要进行选择，外部频综接口应有相应的整形电路；

(9) 操作系统采用 VxWorks。

2.2 系统组成及工作原理

2.2.1 系统组成

系统总体设计框图如图 2.1 所示。处理任务由 PowerPC 最小系统在操作系统的支持下完成，对符合相应电平要求的外部信号由外部信号接口直接送至 FPGA，而其它电平不匹配的外部信号经过相应的电平转换后送至 FPGA，经过 FPGA 预处理后，送至 PowerPC 处理器处理，经 PowerPC 处理后的数据交由 FPGA 转发到外部接口。

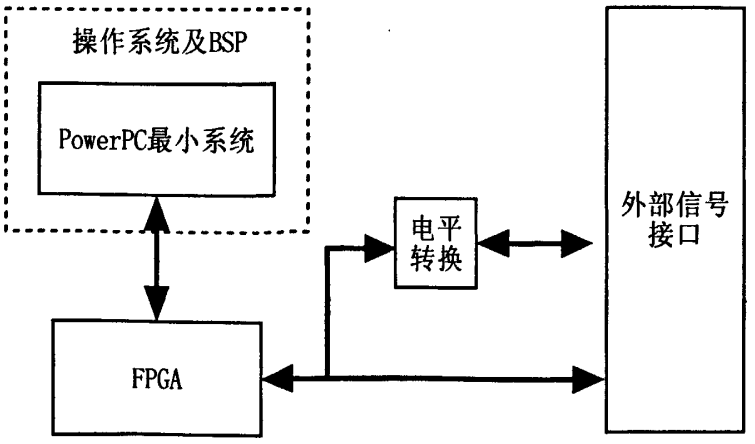

图 2.1 系统总体设计框图

整个系统硬件平台组成框图如图2.2所示。硬件平台包括PowerPC处理器、128MB SDRAM、16MB以及64MB Flash、串口、JTAG调试电路、FPGA及其配置电路和外围接口电路。

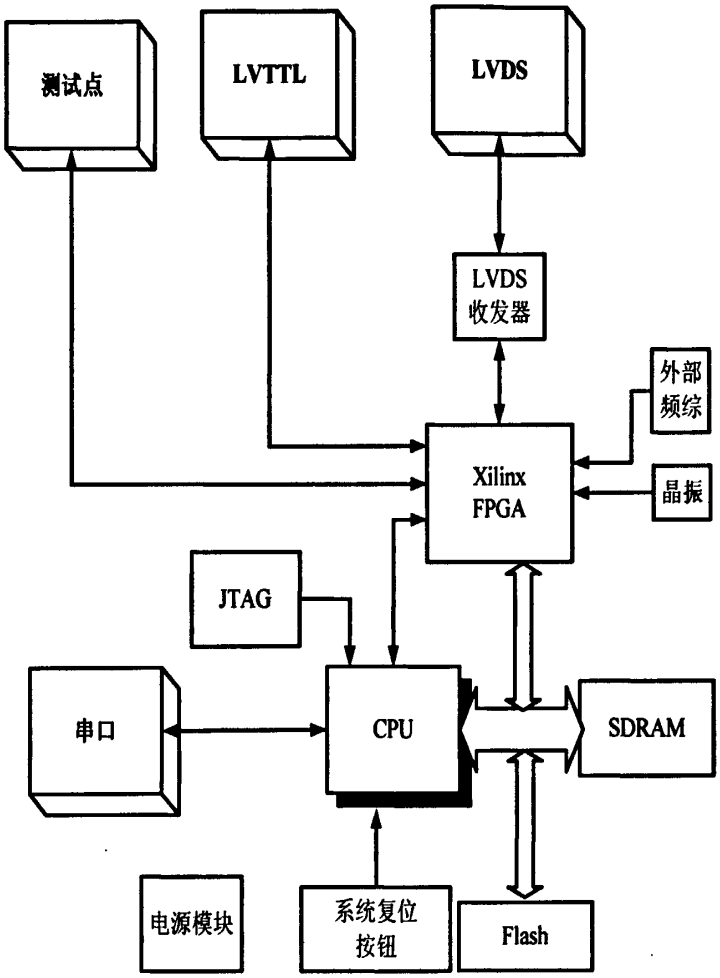

图2.2 系统硬件组成

2.2.2 工作原理

BSP 软件负责系统初始化以及操作系统引导，为系统硬件平台提供驱动程序，管理系统资源，为上层用户应用程序屏蔽底层硬件。SDRAM 作为系统内存，8MB 的 Flash 专门作为存放 BSP 引导程序 bootrom 的存储体，64MB Flash 作为存放 VxWorks 操作系统与用户数据的存储体。系统上电后首先从启动地址 0xFFF00100 开始执行程序，此地址为 BSP 引导程序 bootrom 入口地址，位于 8MB Flash 地址空间内，引导程序执行系统初始化任务，加载 VxWorks 操作系统到 SDRAM，并执行操作系统的初始化，操作系统初始化完成后开始执行用户任务。系统通过串口和 JTAG 接口与主机相连，串口可作为 VxWorks 操作系统映像下载接口和应用

程序调试接口，JTAG 接口可由主机上调试软件通过 USB TAP 调试器连接，通过 JTAG 接口可以直接控制处理器，用于系统各项调试。FPGA 作为处理器与外部信号通信的一个中转，在与外部接口信号通信过程中，外部接口信号经转换后全部连接至 FPGA，接口数据包经过 FPGA 提取后交由 PowerPC 处理器处理，处理结果经由 FPGA 转发到外部信号接口。

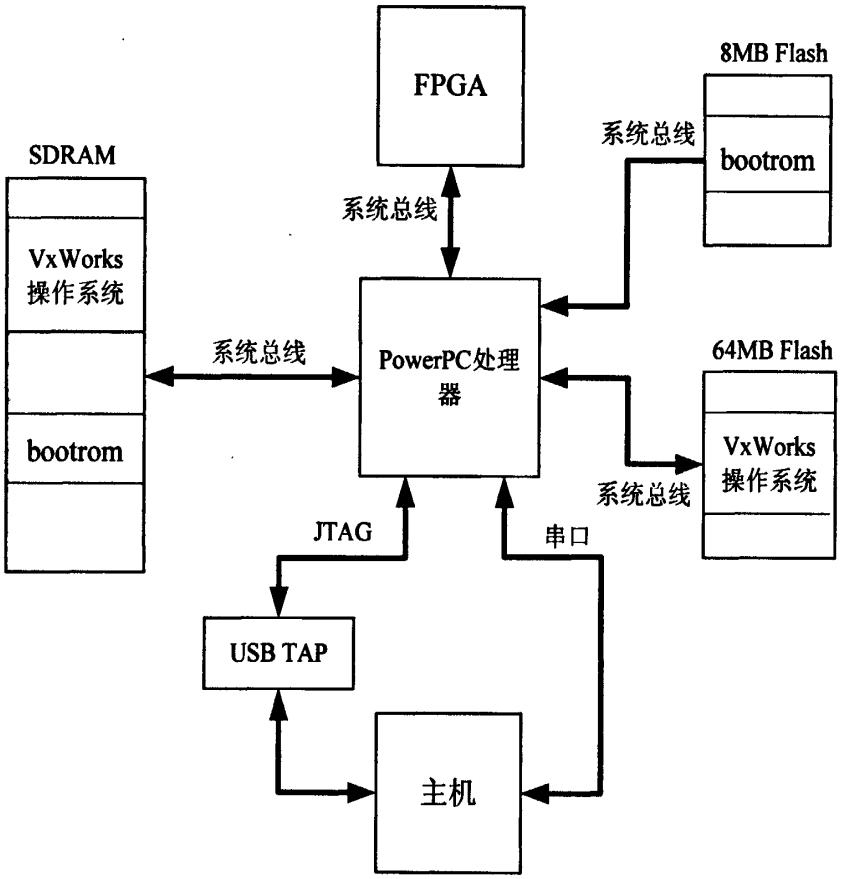

图 2.3 系统工作原理

2.3 系统硬件设计

2.3.1 芯片选型

(1) 电源部分

整个系统需要 5V、3.3V、2V、1.8V 和 1.5V 五种电源，其中 5V 由外接电源供给，3.3V、1.8V、1.5V 分别由外接 5V 电源经 LT1764A-3.3、LT1764A-1.8、LT1764A-1.5 低压差稳压源转换得到，2V 的电源是由 3.3V 通过 MSK5150 转换得到，每种电源在接入系统前都设计有滤波电路以提升电源信号质量。

(2) 复位电路

由于系统中同时存在多种电压，使用 TP3307-18M 来监测整个系统的电压状况。在系统上电期间以及电源系统供电稳定后，如果供给电压不满足系统设定条件，TP3307-18M 会产生复位信号 UP_RST 复位处理器，以保护整个系统的安全。它也提供手动复位的功能。

(3) 处理器

作为整个系统的核心部件，处理器选用 Freescale 的 MPC8245，它具有以下特性[12]：

① 高性能的 603e 处理核，具备硬件浮点单元，默认主频 333MHz；

② 32 位地址总线宽度，最大直接寻址空间可达 4GB；

③ 64 位数据总线宽度，支持 Parity、RMW 以及 ECC 功能；

④ 存储控制器能够支持频率 133MHz，最大支持容量 2GB 的 SDRAM；

⑤ 集成两路 UART、I^2C、两路 DMA 控制器、PCI2.2 规范兼容 PCI 接口、PCI 仲裁器、通用 I/O 和 Flash(ROM)接口、SDRAM 接口等丰富的外围设备。

(4) SDRAM

SDRAM 选用 WENPN16M72V 芯片，它是一款容量为 128MB 的高速 CMOS 动态随机存储器，工作电压为 3.3V，总线频率可以配置成 100、125、133MHz，数据总线宽度是 72 位，有效宽度 64 位，8 位用于支持奇偶校验或者 ECC 校验功能，采用 PBGA 封装有效地减少了板上空间。

(5) Flash 及其外围电路

两块 Flash 分别选用 White Electronic Designs 公司生产的 W72M64V 和 W78M64V。芯片工作电压为 3.3V，数据总线位宽为 64 位，容量分别为 16MB 和 64MB。其中 W72M64V 用来存放 bootrom，W78M64V 用于存放 VxWorks 操作系统以及应用程序。

(6) FPGA 及其外围电路

FPGA 选用 Virtex-II 系列 XQ2V3000，Virtex-II 系列产品采用先进的 0.15μm/0.12μm CMOS 8 层金属混合工艺设计，内核电压为 1.5V，根据输入输出参考电压的不同设计可支持多种接口标准，内部时钟频率可达 420MHz，被认为是高速低耗的最理想设计。XQ2V3000 具有 300 万门电路，最大 I/O 引脚个数达到 720 个。XQ2V3000 对应的配置芯片采用 XCF16PVO48，用以完成 FPGA 程序的上电加载。

2.3.2 核心硬件设计

硬件电路设计过程中根据处理器寻址范围以及所连接器件，对系统硬件资源地址作出分配，地址范围以及作用如表 2-1 所示。

表 2-1 系统地址资源分配

芯片	地址范围	作用
WEDPN16M72V	0x00000000~0x07FFFFFF(128MB)	SDRAM 系统内存
W72M64V	0xFF800000~0xFFFFFFFF(8MB)	存放 bootrom
W78M64V	0x7C000000~0x7FFFFFFF(64MB)	存放 VxWorks 映像和用户数据
XC2V3000	0x78000000~0x7BFFFFFF(64MB)	外部数据预处理

在表 2-1 中，W72M64V、W78M64V、XC2V3000 片选信号分别为 RCS0、RCS2 和 RCS3。W72M64V 是一片 16MB 的 Flash，这里由于 RCS0 选中器件地址空间包含处理器启动地址，它最大仅支持 8MB，所以将 W72M64V 地址线最高位接地，仅使用其起始的 8MB 空间。在设计中还需要注意的是 MPC8245 处理器与 SDRAM 连接采取经过匹配电阻后直接连接，而与 Flash 和 FPGA 连接还需要加驱动芯片 SNV74LVTF162245 和 SNV74LVTF162244，以增强驱动能力。

在系统设计中需要对 MPC8245 处理器做初始化信号配置，MPC8245 上电时某些信号线的高低电平用于设置 MPC8245 某些寄存器初始值。以下是需要配置的信号线以及配置的参数。

(1) PLL_CONFIG[0:4]信号线

根据 PLL_CONFIG[0:4]这 5 根信号线状态配置 PLL。决定外部逻辑/存储器总线频率和处理器频率由 PCI 输入时钟(PCI_SYNC_IN)倍频倍数。配置 PLL_CONFIG[0:4]=0b10110，此时输入时钟为 33MHz，倍频倍数为两倍，得到外部逻辑/存储器总线频率为 66MHz，处理器频率为 264MHz。

(2) AS 信号线

下拉 AS 信号线为低电平，设置 CKO 信号为处理器 CKO。

(3) MDL[0]、FOE 信号线

这两根信号线决定 RCS0 所选 ROM 读写位宽，分别拉高和拉低 MDL[0]与 FOE，设置数据读写位宽为 64 位。

(4) PMAA0、PMAA1 信号线

设定存储器信号匹配电阻，分别拉低和拉高 PMAA0 和 PMAA1 匹配 40Ω负载。

(5) PMAA2 信号线

设定 PCI 和 PIC 控制器输出信号匹配电阻，拉低 PMAA2 信号匹配 20Ω负载。

(6) RCS0 信号线

拉高 RCS0 信号线使 ROM 置于处理器/存储器数据总线。

(7) SDMA1 信号线

选择是否开启扩展地址模式，拉低 SDMA1 信号线开启扩展地址模式，使SDMA12、SDMA13、SDMA14、RCS2 和 RCS3 信号线使能以支持扩展 ROM 空间。

(8) SDMA0 信号线

选择是否开启 DUART 信号，拉低 SDMA0 以开启 DUART 串口信号 SOUT1、SIN1、SOUT2、SIN2。

2.4 本章小结

本章首先分析了空间环境对星载设备的影响，根据课题的设计要求，介绍了课题组研制的星载 MPC8245 最小系统的系统组成与工作原理，最后介绍了系统硬件设计中的关键点，这是下文 BSP 设计实现时需要参考的关键部分。

第三章 VxWorks 及其 BSP 分析

VxWorks 是美国 WindRiver 公司设计开发的一种嵌入式实时操作系统(RTOS)。它采用微内核结构，具备高可靠性和实时性。VxWorks 的一个优点是应用程序的开发可以与目标平台硬件和结构无关，这是由于它把所有特定的硬件功能都集成在一个被称作 BSP（Board Support Package，板级支持包）的软件中，并且提供模拟平台供程序开发人员不用关心实际硬件来开发软件。

板级支持包 BSP 是介于硬件和操作系统之间的一层软件层，应该说是属于操作系统的一部分，主要目的是为了支持操作系统，使之能够更好地运行于目标硬件平台。

在使用嵌入式操作系统 VxWorks 时，可以根据硬件平台移植相关 BSP 或者对某一设备的驱动进行修改来兼容当前平台。在做这些工作之前，除了要对处理器及相关的硬件有所了解，还要深入理解 VxWorks 的 BSP。

3.1 VxWorks 操作系统分析

VxWorks 操作系统是风河公司设计开发的一种嵌入式实时操作系统（其基本结构如图 3.1 所示），它良好的持续发展能力、高性能的微内核以及友好的集成开发环境，使 VxWorks 在嵌入式实时操作系统领域占据一席之地。由于具备良好的可靠性和卓越的实时性，VxWorks 操作系统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。

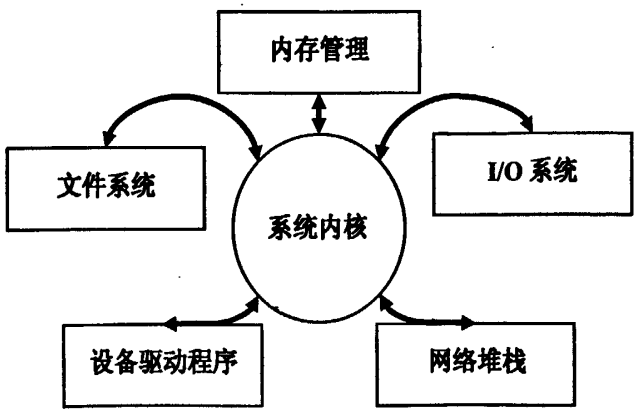

图 3.1 实时系统结构示意图

VxWorks 是一个具有可伸缩、可裁剪和高可靠性的嵌入式操作系统，它支持几乎所有的主流处理器，比如：PowerPC、X86、68K、SPARC、ARM、SuperH、ColdFire 等等。所谓可伸缩性是指 VxWorks 提供了诸多应用编程接口(API)供用户

选择使用；可裁剪性是指用户可以根据需求自行对 VxWorks 组件进行配置，产生具有各种不同功能集的操作系统映像；可靠性是指能够胜任一些关键性任务，比如飞行控制。

3.1.1 VxWorks 操作系统的特点

VxWorks 的主要特点[13,14]如下：

(1) 高性能的微内核

VxWorks 的微内核具有全部实时特性，包括快速的多任务调度、丰富的中断支持以及同时对抢占式调度和时间片轮转调度的支持。它高效的任务管理功能体现在：支持多任务，没有任务数限制；支持抢占调度和时间片轮转调度；快速可靠的上下文切换；支持 256 个任务优先级。VxWorks 提供了快速、灵活的任务间通信机制，包括共享内存、消息队列、Socket、POSIX 管道及信号量、远程过程调用(RPC)和信号量(Semaphore)等。提供的信号量有 3 种：二进制信号量、计数信号量和互斥信号量。

(2) 可裁剪的系统组件

VxWorks 在设计之初就具有可裁剪特性，使得开发者可以对操作系统的功能进行增减，从而更符合用户需求。VxWorks 的内核最小可以裁减至 8KB，利用 Tornado 的工程项目管理工具，可以十分轻松地对 VxWorks 的各种功能选项进行增减。

(3) 丰富的网络支持

VxWorks 是第一个集成标准 TCP/IP 网络功能的实时操作系统。它可以支持众多网络协议：BSD 4.4 TCP/IP；IP、IGMP、CIDR、TCP、UDP、ARP；标准 Berkeley Sockets；SLIP、CSLIP、PPP 等等。

(4) POSIX1003.1b 兼容

VxWorks 不仅支持 POSIX 1003.1 规范的基本系统调用，包括进程原语、文件目录、I/O 原语、语言服务和目录管理等。另外，VxWorks 还支持 POSIX 1003.1b 实时扩展标准，包括异步 I/O、计数信号量、消息队列、信号、内存管理（页面锁定）和调度控制等。

(5) 可靠性

用户都希望自己能在一个工作稳定、可以信赖的操作系统环境中工作，所以操作系统的可靠性是用户需要考虑的首要问题。而高度稳定和可靠一直是 VxWorks 的一个突出优点。自从 1983 年 VxWorks 操作系统设计成功以来，它已被成功的应用于航天、航空、舰船、通信、医疗等关键领域，得到了广泛的验证。

(6) 实时性

系统实时性反映在限定时间内，系统能够执行完规定的功能并且能够响应外部异步事件的能力。实时性的强弱可由完成规定功能和做出相应响应时间的长短来衡量的，由此嵌入式操作系统可以分为非实时的、软实时的和硬实时的。

VxWorks 的实时性非常好，是属于硬实时性操作系统。这是由于系统微内核精炼而集中，系统资源占用极少，任务调度、任务间通信和中断处理等系统进程精简而高效，所以它们造成的延迟很小。VxWorks 提供的多任务机制对任务的调度采用了优先级抢占调度(Preemptive Priority Scheduling)和时间片轮转调度(Round-Robin Scheduling)两种方式。当优先级高的任务准备就绪时会从优先级低的任务中抢占处理器执行时间，优先级相同时平均分配处理器执行时间，如图 3.2 所示。这样可使系统任务灵活调度，能够获得更好的实时性。

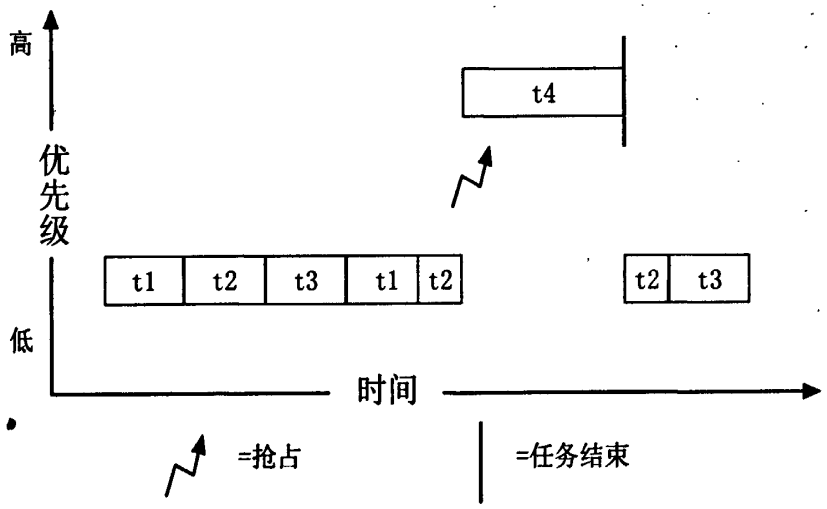

图 3.2 多任务调度模式

简单的可以概括为：可裁剪的微内核体系结构；有效的多任务管理，256 个多任务优先级，支持抢占和轮转调度，快速的上下文切换；灵活高效的任务间通信机制；中断处理响应时间短；支持 POSIX 1003.1b 实时扩展标准；支持多种成熟的标准网络协议；灵活的启动方式：ROM、软盘、硬盘、串口、网络等；多处理器支持；灵活、快速的 I/O 系统；支持 dosFs 等多种文件系统；提供兼容标准 C 的辅助函数以及对 C++的支持。

3.1.2 VxWorks 操作系统组成

VxWorks 嵌入式实时操作系统主要由微内核 Wind、I/O 系统、文件系统、网络系统和虚拟内存管理等几部分组成[15]，VxWorks 系统组成如图 3.3 所示。

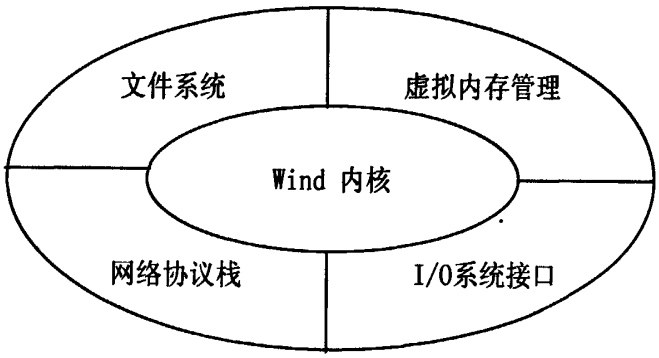

图 3.3 VxWorks 系统组成示意图

(1) 微内核 Wind

Wind 内核是整个操作系统的核心，Wind 内核实现的功能包括：多任务调度、任务间的同步和进程间的通信机制、内存管理、看门狗、定时器和中断处理等。提供符合实时系统标准的 POSIX 1003.1b 接口，以提高应用程序代码的移植性。内核使用中断驱动和基于优先级的调度方式，它可以缩短上下文切换所需时间开销和中断延迟。在 VxWorks 操作系统中，每个程序都可以启动成为一个独立的任务，拥有属于自己的上下文和堆栈。内核只与处理器相关，而与外接设备无关，不关心总线类型、内存大小和 I/O 设备等，底层的 BSP 会驱动具体的硬件设备。

(2) I/O 系统

VxWorks 操作系统提供了一个快速灵活的与 ANSI C 兼容的 I/O 系统，包括 UNIX 标准的缓冲 I/O 和 POSIX 标准的异步 I/O。VxWorks 还提供网络驱动、管道驱动、SCSI 驱动、磁盘驱动等等 I/O 设备驱动程序。

(3) 文件系统

VxWorks 可以支持多个基于块设备的本地文件系统。由于这些块设备都使用相同的标准接口，因此文件系统可以自由地与这些设备驱动程序结合在一起。VxWorks 的 I/O 体系结构使得在 VxWorks 系统中，可以在同一时刻里挂载多个不同文件系统。VxWorks 可支持的本地文件系统有：dosFs、rt11Fs、rawFs、tapeFs 等。除了这些文件系统，VxWorks 还支持闪存文件系统(TrueFFS)、CdromFS 等。另外 TSFS 是主机的文件系统在目标机上的映射，串口调试时常用。VxWorks 操作系统中的文件系统是独立的存在的而不是和设备或者设备驱动捆绑在一起的，任何一个设备都可以使用任何文件系统，用户还可以使用在文件系统、驱动程序和 VxWorks 的 I/O 系统之间的标准接口自己创建文件系统。

(4) 网络系统

强大的网络功能是 VxWorks 的另一大特色。VxWorks 使用了统一的驱动接口 MUX，保证上层网络协议的可移植性。同时也包含了多种网络协议，包括 ICMP、IGMP、IP、UDP、OSPF、RIP 等，以及各种网络服务，包括有 FTP、Telnet、Http

和 DNS 等。在这样广泛协议的支持下，用户任务既支持对文件进行远程存取，又可以实现远程过程调用。

(5) 虚拟内存管理

VxWorks 提供了两种虚拟内存管理接口，一个是绑定在 VxWorks 中的虚拟内存接口，这是最基本的内存管理单元(MMU)，在配置文件中加入宏定义 INCLUDE_MMU_BASIC 即可包含该组件，另一个是可选组件 VxVMI(VxWorks 虚拟内存接口)，实现了更多的 MMU 功能，VxVMI 能够通过设置某一块内存区域的写保护属性，为 VxWorks 中的代码段和中断处理向量表提供写保护，从而防止被改写，提高设备运行的可靠性。这也可以使开发人员集中精力编写自己的应用程序，而不必过于关心无意中修改关键代码段或引发耗时的系统错误。

3.2 VxWorks 系统 BSP 的分析

3.2.1 BSP 概述

BSP 是介于底层硬件和上层软件之间的底层软件开发包，是属于操作系统的一部分。其主要功能包括系统初始化，提供驱动程序和整合软硬件，具体功能[16]包括：

(1) 系统硬件初始化，主要是处理器的初始化，为整个软件系统提供底层硬件支持；

(2) 提供设备驱动程序和系统中断服务程序；

(3) 配置操作系统的功能，为软件系统提供一个实时多任务的运行环境；

(4) 初始化操作系统，为操作系统的正常运行做好准备。

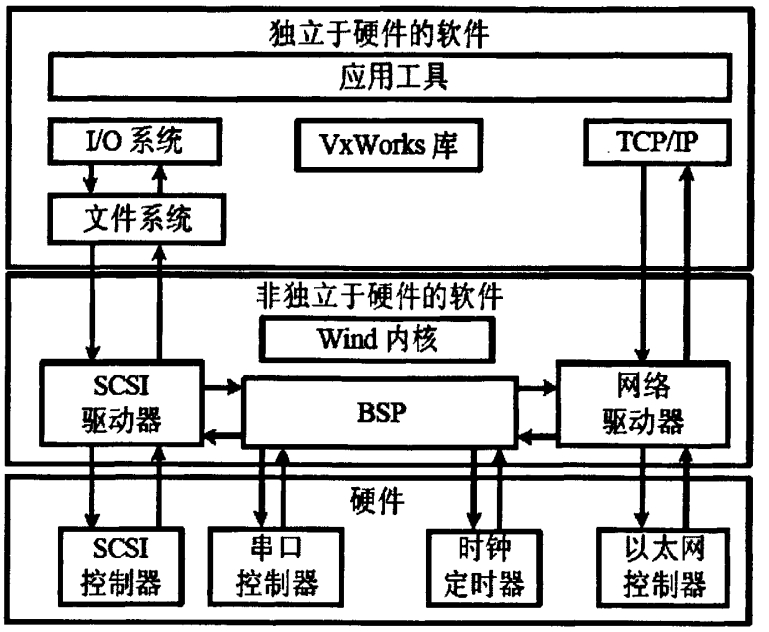

图 3.4 BSP 在 VxWorks 中的层次图

BSP 在 VxWorks 中的层次如图 3.4 所示。VxWorks 针对不同的处理器，会有不同的 BSP。即使同一种处理器，由于外接设备的差异，BSP 的相应部分也不一定一样。所以根据具体硬件设计编写和修改 BSP 保证系统正常运行是非常重要的。

BSP 的概念是针对嵌入式操作系统而言的，而靠 BIOS 引导的桌面操作系统像 DOS、Windows 和 UNIX 等是没有 BSP 这一说法的。BSP 在功能上类似于 PC 机主板上的 BIOS，BIOS 主要功能是负责在 PC 机开机时检测并初始化系统设备（设置栈指针、中断分配和内存初始化等）载入操作系统，还要调度操作系统向硬件发出的指令，它并不包含基本的硬件驱动。而 BSP 和操作系统是不可分割的，它们绑在一起运行在主板上。此外 BSP 还必须提供和目标系统有关的驱动（串口、网口等）。

3.2.2 BSP 的组成

BSP 的组成文件都存放在\target 目录下，里面会包含许多文件夹。BSP 主要组成文件在\target\config\all 和\target\config\bspname 这两个文件夹里。all 文件夹包含所有 BSP 所共享的文件，任何 bspname 文件夹内的 BSP 都需要包含 all 文件夹内的文件。BSP 主要文件目录[15,17]包括：

(1) 目录 taget\config\all

这个目录下的文件主要包含了 configAll.h、bootInit.c、bootConfig.c 和 usrConfig.c 4 个文件。这 4 个文件只在 all 文件夹内出现，默认所有 BSP 都会包含这 4 个文件。主要作用是定义了所有 VxWorks 的设置以及 bootrom 的初始化代码。

一般情况不要更改里面的任何文件，需要修改时可以将该文件复制备份到指定BSP文件夹中，通过包含修改的备份文件来达到目的。特别要注意的是这里面的文件只在用户建立工程时才被包含调用，在编译工程时不再调用，所以用户在建立工程以后对此目录下的文件所做的改动对工程来说是无效的，只能在工程中做相应的改动。如果使用控制台来编译 BSP 文件则会一直调用，任何修改都会对 BSP 编译造成影响。文件夹内包含：

① configAll.h：该文件包括了所有 VxWorks 的默认配置宏定义。

② bootInit.c：bootrom 映像的第二阶段的初始化代码。romStart()函数包含在该文件内，它是由 romInit.s 中的 romInit()函数执行完后调用的，实现的功能包括：对压缩的 VxWorks 映像进行解压缩，代码/数据段从 ROM 拷到 RAM。

③ bootConfig.c：bootrom 映像的主要初始化和控制文件。

④ usrConfig.c：VxWorks 映像的主要初始化代码，包括根任务、系统初始化、网络初始化、时钟中断程序等。

(2) 目录 target\config\comps\src

这个目录下全部是涉及系统核心的模块组成文件，一般可以在 target\config\all\usrConfig.c 中找到相应的函数调用。

(3) 目录 target\config\bspname

bspname 是指具体目标板的名字，如 wrPpmc824x 表示一款使用 PowerPC 处理器的目标板。文件夹内可能包含文件有：

① README：该文件包含了 BSP 的版本记录。它记录了当前 BSP 版本的功能、以前每次修改的内容和其它相关信息。

② Makefile：Makefile 文件是通过命令行方式对创建映象文件的过程进行控制的规则文件。该文件内还包含了特定的规则与依赖，以及生成引导映像的文件格式。

③ config.h：config.h 文件是用户在配置自己的 BSP 时最主要的配置文件，定义了所有与目标系统相关的包含文件和配置信息、宏定义等，有诸如 BSP 的版本号、存储器缓存和 MMU 配置、网络共享储存定义、RAM 和 ROM 的定位以及容量的配置、NVRAM（非易失性存储器）参数、默认的系统引导参数行定义、外部总线地址映射、网卡中断号及中断向量的设置等等。命令行方式编译 BSP 时，对 VxWorks 系统组件的配置和裁剪需要在该文件内体现，可以通过#define 和#undef 宏来定义和增减系统组件。

④ romInit.s：该文件由汇编语言编写，是 bootrom 引导程序入口所在文件。系统上电后首先要执行该文件中的初始化代码。它的主要功能是进行处理器初始化，具体包括屏蔽中断、初始化内存系统、初始化堆栈指针和系统寄存器等。文件内主要包含了 romInit()函数，它必须设计成执行过程与内存地址无关的代码

(PIC)。

⑤ sysALib.s：该文件也是由汇编语言编写，是操作系统引导后首先执行的代码。包含 sysInit()函数，它是 VxWorks 5.5 操作系统运行的入口函数，也是在 RAM 中执行的第一个函数。它包含系统基本的端口读写函数、处理器检测函数，会禁止中断和 Cache，建立堆栈，转而执行 usrConfig.c 文件中的 usrInit()函数。

⑥ sysLib.c：该文件包含了目标系统相关的库文件，提供给目标系统设备的驱动程序，核心程序是 sysHwInit()，它调用设备驱动程序中的初始化程序，完成系统硬件初始化。如串口、时钟、中断控制器、网卡等等都在本文件中完成初始化工作。该文件还要完成内存地址映射，使操作系统可以访问被映射的内存地址，完成这个功能需要在 sysPhysMemDesc 数组中增加元素。

⑦ sysSerial.c：该文件是可选文件，包含目标板上串口设备驱动程序的配置和初始化。

⑧ sysNet.c，sysScsi.c：这两个文件也是可选文件，初始化目标板上网络接口和 SCSI 设备。

⑨ bspname.h：该文件为目标板参数配置头文件，使用 bspname.h 来设置所有非通用的与特定目标系统相关的信息，包括定义中断向量或中断号、I/O 设备地址、设备寄存器位的含义、系统时钟和辅助时钟的最大和最小时钟的速率等用户自定义配置。

⑩ 其它：包含根据目标板需要添加的驱动程序源文件。

对于不同的目标系统，用户可以通过修改这些文件来完成 BSP 与 VxWorks 在不同平台上的移植。

除了目录 target\config 下的这些核心程序外，在目录 target\src 和 target\h 下 VxWorks 提供了所有有关 BSP 下的组件驱动源程序以及头文件，这里的组件包括硬盘设备、软盘设备、PCI 设备、网络设备、SCSI 设备、串口设备、定时器及 TrueFFS 设备等。bspname 文件夹内文件在 Makefile 和 depend.bspname 等文件的控制下经过编译、链接，最后将生成所需映像。

3.3 VxWorks 启动过程分析

首先分析一下 VxWorks 的映像类型，在不同的应用情况下可分为图 3.5 所示的两种类型[18]：可引导映像(Bootable Image)和可下载映像(Loadable Image)。

(1) 可下载映像(Loadable Image)

该类型的映像一般用于调试，实际包括两部分：VxWorks 和 bootrom，两部分是独立创建的。其中 bootrom 包括被压缩的 bootrom 映像(bootrom)、非压缩的 bootrom 映像(bootrom_uncmp)和驻留 ROM 的 bootrom 映像(bootrom_res)三种类型。

bootrom 的作用是初始化目标板，特别需要初始化调试下载通道以及调试信息输出通道，为 VxWorks 的引导启动做准备。VxWorks 映像需要通过主机与目标机之间的某种连接方式（如串口和网口）被 bootrom 引导下载到目标机的内存中运行。

(2) 可引导映像(Bootable Image)

可引导映像是将 bootrom 引导代码映像和 VxWorks 操作系统映像合二为一的映像，有基本的 Wind 内核及 VxWorks 经裁剪后的组件。它常常作为提供给用户的最终产品而直接烧写至系统 ROM 中，作用包括引导目标系统，并根据 VxWorks 操作系统映像的类型将其加载到适当的内存位置开始引导启动操作系统。它包括非驻留 ROM 的映像和驻留 ROM 的映像两种类型。非驻留 ROM 映像在 BSP 初始化时把 VxWorks 操作系统映像完全搬到 RAM 中执行，如图 3.6 所示。驻留 ROM 映像在 BSP 初始化时，把 VxWorks 操作系统映像中的 data 段复制到 RAM 中，而其代码仍在 ROM 中运行，如图 3.7 所示。

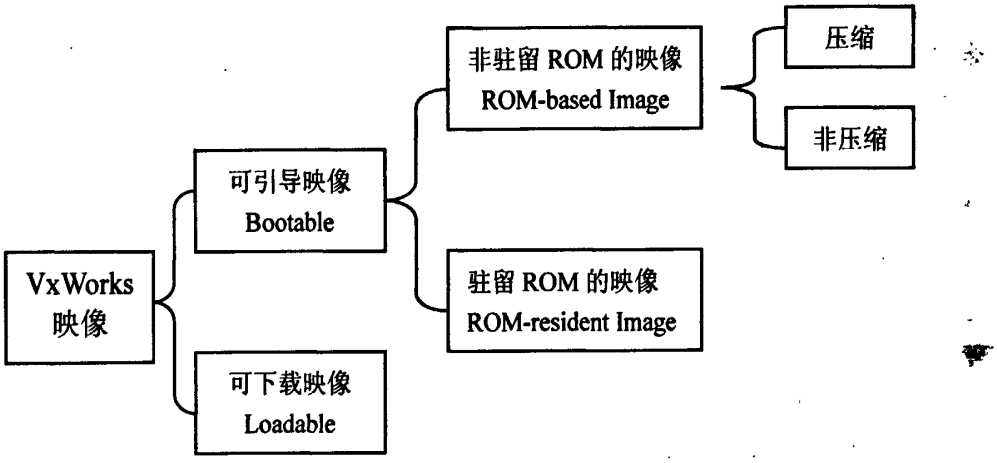

图 3.5 VxWorks 映像分类

VxWorks 操作系统的启动涉及到两种映像文件：bootrom 映像和 VxWorks 映像，它们每一类又可以再细分为不同类型的映像。一般基于 RAM 的 VxWorks 映像是用于目标板的调试阶段，而基于 ROM 的 VxWorks 映像用于成品阶段。其实两个映像分别体现了两个启动过程：bootrom 的引导过程和 VxWorks 的加载过程。

完成了 bootrom 的引导过程，下一步就是加载 VxWorks 映像文件，可以有多种加载方式：通过本地磁盘、通过网络、通过串口等。

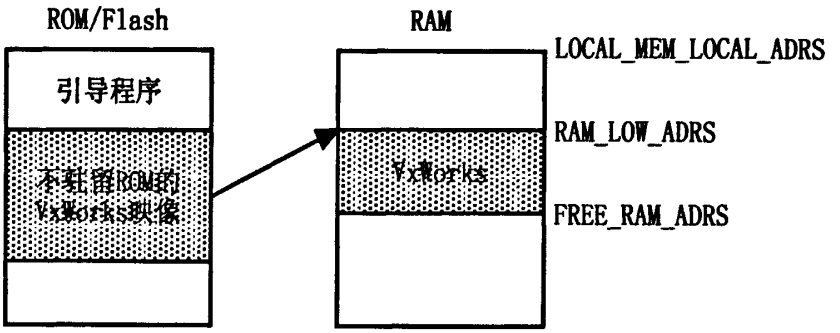

图 3.6 非驻留 ROM 映像加载

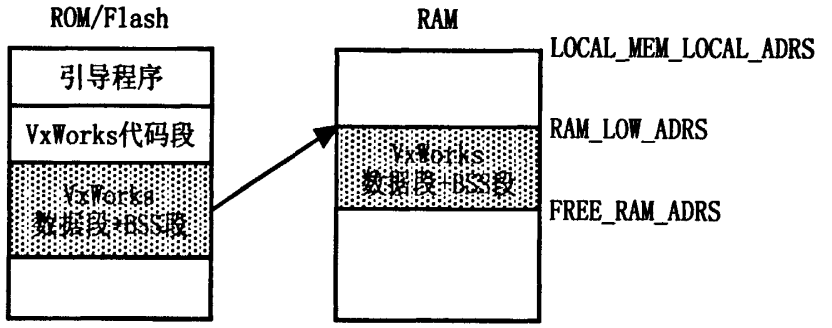

图 3.7 驻留 ROM 映像加载

bootrom 映像启动过程中，依次执行 BSP 中下列几个文件中的代码：

(1) romInit.s 中的 romInit()，完成如下功能：

① 保存系统启动类型，关闭中断，初始化处理器。

② 初始化内存系统，通常需要初始化内存控制器寄存器，屏蔽缓存。对于 SPARC 系列处理器，必须开启 MMU。

③ 将堆栈指针定位于被拷贝 bootrom 映像在内存中的地址，设置相关寄存器准备进入 romStart()程序执行 C 代码。

(2) bootInit.c 中的 romStart()，完成如下功能：

完成的功能包括跟据不同的映像类型，将不同的 VxWorks 映像进行解压缩后加载到 RAM 中运行，非压缩的 VxWorks 映像直接加载到 RAM 中运行。

(3) bootConfig.c 的 usrInit()，完成如下功能：

① 对 VxWorks 映像的 bss 段清零。

② 调用 intVecBaseSet()设置矢量基地址表。

③ 调用 sysHwInit()初始化异常中断矢量，初始化系统硬件。

④ 调用 usrKernalInit()初始化 wind 内核。

⑤ 调用 KernalInit()启动 wind 内核，启动 usrRoot()任务。

(4) bootConfig.c 中的 usrRoot()，完成如下功能：

① 设置操作系统相关时钟。

② 创建设备并安装设备驱动程序。

③ 下载 VxWorks 映像[19]。

VxWorks 映像下载完成后，操作系统的引导启动顺序与 bootrom 映像类似，它从 BSP 中的 sysALib.s 文件中的 sysInit()开始运行，功能类似于 romInit()，主要完成重新初始化目标板。然后是 usrConfig.c 中的 usrInit()，类似于 bootConfig.c 中的同名函数，完成相似功能，其中 excVecInit()完成异常中断矢量初始化，最后是 usrRoot()设置操作系统时钟、创建设备、安装设备驱动程序、初始化 I/O 系统、初始化并挂载文件系统、调用用户应用程序等。

整个启动流程如图 3.8 所示。

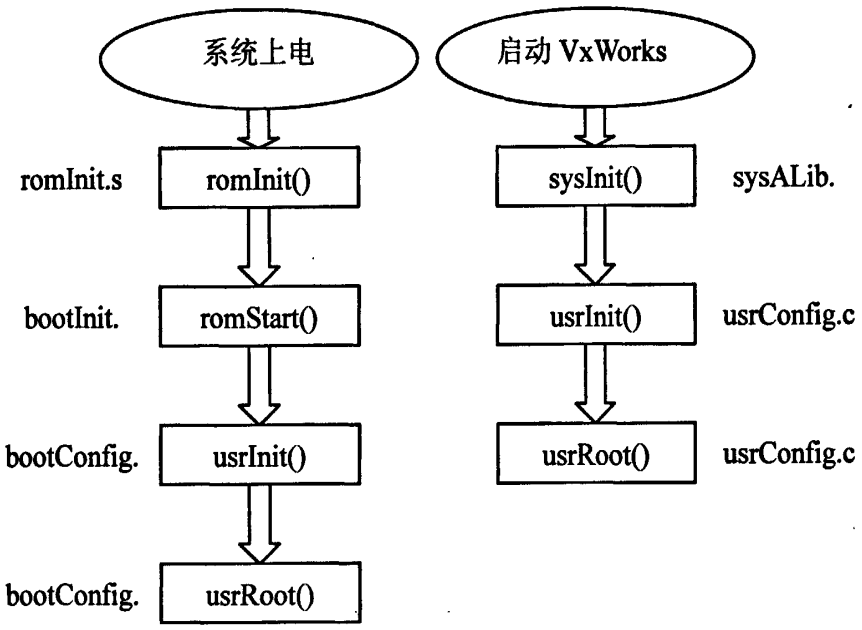

图 3.8 VxWorks 启动流程图

3.4 Tornado 开发环境简介

Tomado 集成开发软件是与嵌入式实时操作系统配套使用的一款开发调试环境，是编写 VxWorks 嵌入式应用程序的软件开发平台，是交叉开发环境在主机上运行的部分，也是开发和调试 VxWorks 系统一个有力的组成部分。Tomado 是集成了编辑器、编译器和调试器于一体的高度集成的窗口环境，给嵌入式系统开发人员提供了一个不受目标机资源限制的超级开发和调试环境[15]。图 3.9 表示了 Tornado 环境的基本组成。

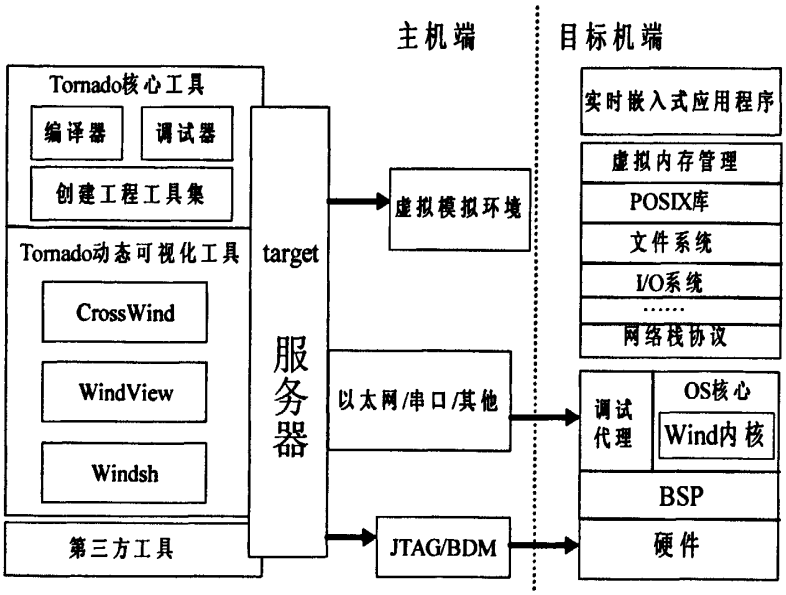

图 3.9 Tornado 环境结构

Tornado 是一种交互式的软件开发集成环境，集成了设计开发和分析等特性，使之成为一个有机整体，它提供一种有效的方式开发嵌入式实时应用程序，同时对目标机系统的影响做到最小。Tornado 的主要组成部分[20]有：

(1) 集成的源代码编辑器。具备标准的文档处理能力，调试程序时追踪代码执行，编译连接出现错误或警告时可以迅速定位到出错位置。

(2) 工程管理工具(Project)。这是一个功能强大的图形化工具，可以通过图形接口来自动配置 VxWorks 操作系统以及其它组件。它可以执行自动的依赖性分析、程序代码容量计算和自动裁剪向导，大大缩短了开发周期。

(3) 集成的 C 和 C++编译器和 Make 工具。提供 GNU 以及 Diab 的交叉编译器，支持 C 和 C++语言。并对编译器进行了一些优化，允许开发者能够迅速产生紧凑而高效代码。

(4) 浏览器 Browser。Browser 是一款图形化工具，用户通过 Browser 可以直观的看到系统中信号量、消息队列、内存分配、看门狗计时器、堆栈使用情况、目标机处理器使用率、对象模块和符号表以及每个任务的详细资源信息。Browser 汇总了应用程序、内存消耗和目标内存映射三个功能。

(5) 图像化的交叉调试器(Debugger) CrossWind。源码级调试器 CrossWind 提供给开发者图形化和命令行接口来完成全部的调试功能。调试器包含许多观察窗口，可以查看或修改存储器值、变量、寄存器值等，也可通过命令行的命令形式完成程序反汇编、设置断点、单步执行等调试功能。在宿主机上使用 CrossWind，可以创建、运行和调试目标机运行系统中的任务。

(6) 命令执行工具 WindSh。WindSh 作为用户与目标机交互的接口，用户命令

通过 Tornado 统一的命令解释器接口 WindSh 翻译后由目标机实现。Tornado 命令解释器提供了一个简单、强大的功能，它不仅能解释和执行几乎所有的 C 语言表达式，包括函数的调用和系统符号表中变量的引用，而且可以通过特定调试指令实现所有的调试功能。

(7) 目标机软件逻辑分析仪 WindView。WindView 是一个图形化的可视诊断和分析用户目标机系统的工具。它可以直观的提供给用户 VxWorks 应用程序执行期间的详细动态行为：以不同的图形和符号表示任务、中断和系统对象交互以及上下文切换间的复杂关系，使用户非常容易地观察出任务、中断程序之间的相互关系，是在嵌入式系统应用开发期间一款非常强大的分析工具。

(8) 集成仿真器 VxSim。VxSim 是一个运行在主机系统上的仿真器，提供与真实目标机一致的调试和仿真运行环境，并支持 CrossWind、WindView 和 Browser 等工具。VxSim 作为 Tornado 核心工具包含在软件包中，可以使软件开发者不必关心相关硬件状况开发应用软件。因而开发者可以在没有 BSP、操作系统配置和目标机硬件的情况下，基于 VxSim 开始软件开发工作，缩短项目开发时间，提高开发效率。

3.5 本章小结

本章首先介绍了MPC8245最小系统中运行的嵌入式操作系统VxWorks的概况与特点，接着介绍了 BSP（板级支持包）的概念与组成。本章分析了 VxWorks 嵌入式操作系统的启动流程，最后简介了其集成开发环境 Tornado 软件。

第四章 VxWorks BSP 设计与实现

4.1 BSP 设计目标

本课题 BSP 设计目标包含以下几个方面：

(1) 处理器初始化。此代码由汇编语言编写，是系统上电后立即执行的代码，它初始化处理器相关寄存器，包括 PCI 管理、电源管理、时钟驱动等。由于它是第一个执行的代码，它的正确性十分重要。

(2) SDRAM 及 Flash 配置。SDRAM 与 Flash 配置是在处理器相关寄存器配置中完成的，所以它也由汇编程序编写，包括配置地址范围、时序等等。

(3) 串口调试环境搭建。利用 MPC8245 处理器内部的 DUART1 和 DUART2，编写串口输出调试汇编代码以及驱动程序；配置 CONSOLE 通道和 WDB 通道，让主机通过 Target Server 与目标板连接，通过 TSFS 方式进行 VxWorks 操作系统下载并且调试系统。

(4) 配置 TFFS 文件系统。用于管理 W78M64V 这片 Flash，可以存放 VxWorks 操作系统映像以及用户数据文件。

(5) 其它配置。配置包括 MMU 内存管理单元、时钟频率、设备中断和 VxWorks 操作系统组件等等。

4.2 BSP 设计与功能实现

VxWorks 的 BSP 设计一般遵循图 4.1 所示流程。首先需要选择一个与目标系统处理器相近的模板 BSP，在此基础上对 BSP 文件进行修改来实现操作系统对目标板的支持；选择好模板 BSP 后，需要对包含文件进行增减来匹配目标板硬件，去除无用硬件驱动与配置加入所需硬件驱动与配置，可以参考 target\src 和 target\h 下的 drv 文件夹内相应硬件驱动程序。下面进行最重要的一步 BSP 修改，通过修改 BSP 内文件具体函数与配置实现操作系统对目标板的支持。修改完成并编译生成 bootrom 映像与 VxWorks 映像，在目标板上调试后若系统稳定运行，所需功能均实现则 BSP 设计完成；若系统需求变更或者调试出现问题则需要返回到文件增减或 BSP 修改步骤继续完善 BSP。

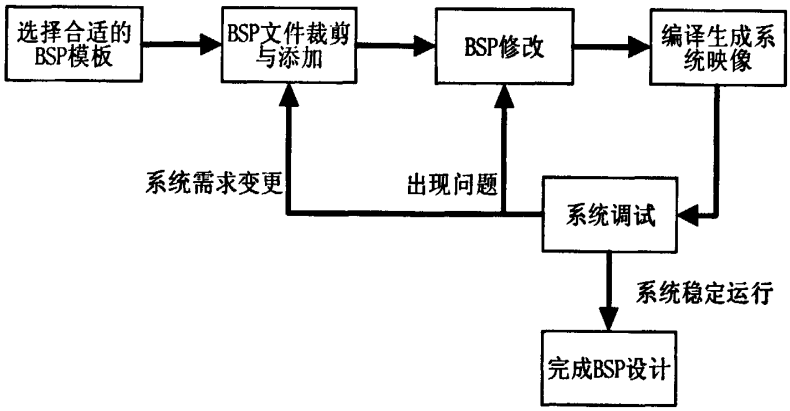

图 4.1 BSP 设计流程

VxWorks 操作系统的 BSP 文件存放在 \target\config 文件夹内，BSP 开发时尽量选择相近的 BSP 模板进行修改，减少代码量编写。首先做好开发前的准备工作：

(1) 复制模板 BSP 文件夹为 myMPC8245_3 文件夹（文件夹名字可以自己定义），复制\target\config\all 文件夹到 myMPC8245_3 内，并改名为 myMPCAll。这样做的目的是建立一个独立的开发环境，避免对本 BSP 的修改影响到其它 BSP。

(2) BSP 文件裁剪。删除 myMPC8245_3 文件夹内的无用文件，例如网络配置以及网卡驱动、NvRam 配置等等，减少文件个数与代码阅读量。

4.2.1 处理器、SDRAM、Flash 配置

处理器、SDRAM、Flash 的初始化配置在 romInit.s 文件中完成。处理器核对配置寄存器的访问需要通过基于内存映射的配置端口，这类似于访问 PCI 设备寄存器。对配置寄存器的读写首先需要向 CONFIG_ADDR 端口地址 0xFEC00000 写入配置寄存器地址，再对 CONFIG_DAT 端口地址 0xFEE00000 进行读写操作，即相当于对该配置寄存器进行读写操作。下面对 romInit.s 文件进行修改：

(1) 存储器配置，需要配置下面几个寄存器。

① 存储器边界寄存器(Memory Boundary Registers)

需要配置 8 个 32 位寄存器来界定存储器每个 Bank 的起止范围。本课题所选 SDRAM 为 128MB，所以分配 Bank0 为 128MB 作为 SDRAM 的地址空间。寄存器介绍与设置如下：

存储器起始地址寄存器 MSAR(Memory Start Address Register)分为 MSAR1 与 MSAR2；扩展存储器起始地址寄存器 EMSAR(Extended Memory Starting Address Register)分为 EMSAR1 与 EMSAR2。通过这四个寄存器来设置存储器各个 Bank 的起始地址。

存储器结束地址寄存器 MEAR(Memory End Address Register)分为 MEAR1 与

MEAR2；扩展存储器结束地址寄存器 EMEAR(Extended Memory End Address Register)分为 EMEAR1 与 EMEAR2。通过这四个寄存器来设置存储器各个 Bank 的结束地址。存储器边界寄存器内部结构见图 4.2 和图 4.3。

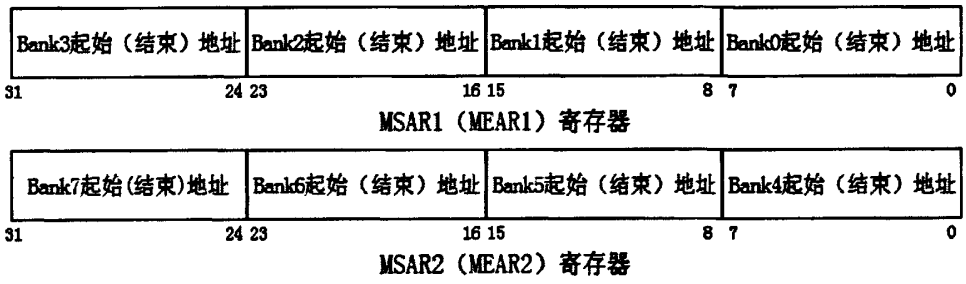

图 4.2 存储器起始（结束）地址寄存器结构

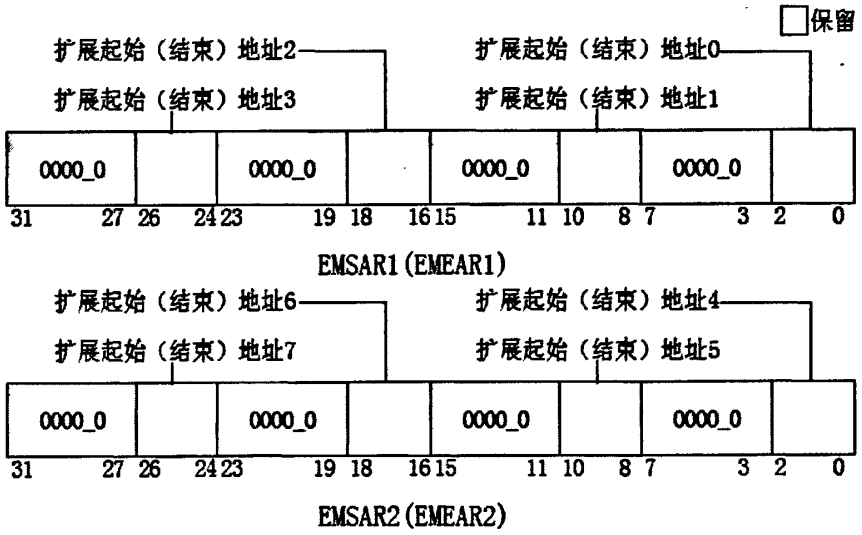

图 4.3 扩展存储器起始（结束）地址寄存器结构

分别设置这 8 个寄存器为：

MSAR1：0x80008000；MSAR2：0x80008000；EMSAR1：0x01010000；

EMSAR2：0x03030202；MEAR1：0xFF7FFF7F；MEAR2：0x01010000；

EMEAR1：0xFF7FFF7F；EMEAR2：0x03030202。

设置完成后，即分配给每个 Bank 为 128M，起止地址如下：

Bank0：0x00000000~0x07FFFFFF；Bank1：0x08000000~0x0FFFFFFF；

Bank2：0x10000000~0x17FFFFFF；Bank3：0x18000000~0x1FFFFFFF；

Bank4：0x20000000~0x27FFFFFF；Bank5：0x28000000~0x2FFFFFFF；

Bank6：0x30000000~0x37FFFFFF；Bank7：0x38000000~0x3FFFFFFF。

② 存储器使能寄存器(Memory Bank Enable Register)

这是一个 8 位的寄存器，每一位对应一个 Bank 的使能状态。设置为 0x01 使得 Bank0 有效，其它 Bank 无效。

③ 存储器页面模式寄存器(Memory Page Mode Register)

这仍是一个 8 位寄存器，存放的是 PGMAX 参数，它控制 MPC8245 访问当前可访问页面的时间。当 PGMAX 为 0x00 时，页面模式被禁用。它可由式(4-1)[12]估计：

$$PGMAX<[t_{RAS(MAX)} - (\text{worst-case memory access time}) - 2]/64 \qquad (4\text{-}1)$$

PGMAX 描述了 SDRAM 从激活到预加电的时间间隔 $t_{RAS(MAX)}$，PGMAX 的值乘以 64 以后产生实际的间隔周期数。上面公式中有两个变量 $t_{RAS(MAX)}$ 和 worst-case memory access time（最差情况下存储器访问时间）。

对于 SDRAM，PGMAX 参数定义可以由图 4.4 描述：

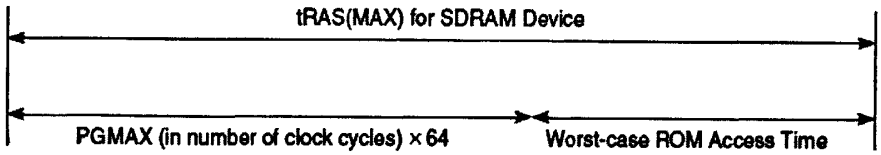

图 4.4 PGMAX 参数定义

$t_{RAS(MAX)}$ 可以从 SDRAM 芯片手册上面查到，WEDPN16M72VR SDRAM 的 $t_{RAS(MAX)}$ 最大值为 120000ns，SDRAM 总线频率 66MHz，所以 $t_{RAS(MAX)}$=120×66=7920 个时钟周期。

worst-case ROM Access Time 在本课题中为最长 Flash 访问时间。Flash 采用 non-burst 64 位读取，访问时间需要 3+ROMFAL 个时钟周期。ROMFAL 是对 ROM 进行多次读写之间需要等待的时间，在后面会提到的 MCCR1 寄存器中设置。设置它为最大值 0b11111，十进制为 31。所以 worst-case memory access time=3+31=34 个时钟周期。

可以估得 PGMAX=[7920-34-2]/64=123.1875=0x7B。

④ 存储器控制寄存器(Memory Control Configuration Register)

包括 4 个 32 位的寄存器 MCCR1、MCCR2、MCCR3 和 MCCR4，可以配置所有的 RAM 和 ROM 参数。MCCR1 中的 MEMGO 位开启存储器接口，所以先不设置 MCCR1 中的 MEMGO 位，等 MCCR4 设置完成后再将 MCCR1 寄存器 MEMGO 位置位。为了稳定性考虑，所有有关时钟周期的选择尽量选择最大周期，下面仅介绍寄存器重要位的配置方法：

MCCR1：

清除 SDRAM_EN 位，使能 SDRAM；

设置 Bank0 为 13 行×n 列×4 个逻辑 Banks。

MCCR2：

设置 REFINT 刷新间隔。可由下面的式(4-2)[12]估算：

$$REFINT<\frac{RP}{(n+1)16}-\frac{16(ROH)}{16}-\frac{TWACC}{16} \quad (4\text{-}2)$$

式(4-2)中，RP 是 SDRAM 的刷新周期，它等于每个 Bank 刷新时间 t_{REF} × Bank 总数×SDRAM 频率；n 是每个 Bank 的行数×Bank 总数/16；ROH 是刷新开销时间；TWACC 是存储器总线上最慢设备的访问时间。由于 ROH 与 $\frac{TWACC}{16}$ 参数较小，将其忽略，通过查询 SDRAM 芯片手册求得：REFINT=128=0b10000000。

MCCR3：

设置 REFREC 时钟周期：REFREC 为 SDRAM 从刷新到激活状态所需时钟周期，WEDPN16M72VR 的 t_{RFC}=70ns，REFREC=70ns×66MHz=5 个时钟周期。

MCCR4：

设置 PRETOACT 时钟周期：PRETOACT 为 SDRAM 从预加电到激活的时钟周期，由 t_{RP} 决定，算得 PRETOACT=20ns×66MHz=0x02；

设置 ACTOPRE 时钟周期：ACTOPRE 是 SDRAM 从激活到预加电的时钟周期，由 t_{RAS} 决定，算得 ACTOPRE=50ns×66MHz=0x04；

设置 ACTORW 时钟周期：ACTORW 是 SDRAM 从激活到读或者写的时钟周期，由 t_{RCD} 决定，算得 ACTORW=20ns×66MHz=0x02。由于开启 in-line ECC/parity 需要最小 3 个时钟周期，此处设置 ACTORW=0x03；

MCCR1：

设置 MEMGO 位开启存储器接口。

(2) 扩展 Flash 配置，需要配置扩展 ROM 控制寄存器 ERCR(Extended ROM Configuration Registers)，包括 ERCR1、ERCR2、ERCR3 和 ERCR4。ERCR1 与 ERCR2 分别控制片选 RCS2 与 RCS3 所接的 ROM，本课题中 RCS2 接 W78M64V，RCS3 与 FPGA 相连。RCS2 与 RCS3 默认均开启，使用默认设置即可。ERCR3 与 ERCR4 设置 RCS2 与 RCS3 的地址范围，这里设置 RCS2 选中从 0x7C000000 开始的 64MB 空间，RCS3 选中从 0x78000000 开始的 64MB 空间。

(3) 其它寄存器配置。

① 处理器接口配置寄存器 PICR(Processor Interface Configuration Registers)，分为 PICR1 和 PICR2。置位 PICR1 的 RCS0 位，使 RCS0 所选 ROM 处在 processor/memory 数据总线；置位 FLASH_WR_EN 使 Flash 可写；置位 MCK_EN 开启 Machine Check 功能；PICR2 使用默认设置。

② 外围逻辑电源管理配置寄存器 PMCR(Peripheral Logic Power Management Configuration Register)，分为 PMCR1 和 PMCR2。置位 PMCR1 的 NO_NAP_MSG

位，使 MPC8245 不会进入 NAP 模式之前在 PCI 总线上广播 HALT 指令；置位 NO_SLEEP_MSG 位，使 MPC8245 不会再进入 SLEEP 模式之前在 PCI 总线上广播 SLEEP 信息；置位 LP_REF_EN 位，使 MPC8245 在进入 SLEEP 模式后也能刷新 SDRAM；配置 CKO_MODE 位为 0b01，选择系统内部 sys_logic_clk 信号作为测试输出时钟源；配置 CKO_SEL 位为 0b0，选择 CKO 信号是 processor core 时钟驱动；配置 PMCR2 的 PCI_HOLD_DEL 位为 0b10 选择 33MHz PCI 总线。

③ 输出驱动控制寄存器 ODCR(Output Driver Control Register)，配置 DRV_PCI 位为 0b1，选择 PCI/PCI 信号匹配 40Ω 负载；配置 DRV_MEM_CTRL[1-2]位为 0b01，选择所有标准的存储器信号匹配 40Ω 负载；配置 DRV_PCI_CLK 位为 0b10，为 PCI_CLK[0:4] 以及 PCI_CLK_SYNC_OUT 输出选择匹配 20Ω 负载；配置 DRV_MEM_CLK 位为 0b11，为 SDRAM_CLK[0:3]以及 SDRAM_SYNC_OUT 输出选择匹配 20Ω 负载。

④ 时钟驱动控制寄存器 CDCR(CLK Driver Control Register)，配置为 0xFC08 屏蔽除 SDRAM_CLK0、SDRAM_CLK1、SDRAM_CLK2 外其它时钟输出。

⑤ Embedded Utilities Memory Block Base Address Register，本寄存器存放 MPC8245 内部嵌入设备的内存基地址，范围可从 0x8000_0000~0xFDFF_FFFF，这里设为 0xFC000000。

⑥ PCI 命令寄存器 PCI Command Register，置位 Bus Master 位使 MPC8245 作为 PCI 主控制器；置位 Memory Space 位，使 MPC8245 能够响应 PCI 内存空间访问。

⑦ PCI 状态寄存器 PCI Status Register，写入 0xFFFF 清除所有异常状态。

初始化配置完成还要修改 config.h，sysLib.c 以及 Makefile 文件加入操作系统和编译时需要的信息。

修改 config.h 文件，修改下面描述的宏定义使系统识别 8MB Flash 与 SDRAM 信息。

```
#define ROM_BASE_ADRS       0xFF800000          /* Flash 起始地址*/
#define ROM_TEXT_ADRS       0xFFF00100          /*Flash 程序引导地址*/
#define ROM_SIZE            0x00800000          /*Flash 容量*/
#define LOCAL_MEM_LOCAL_ADRS 0x00000000         /*RAM 起始地址*/
#define LOCAL_MEM_SIZE      0x08000000          /*128Mbytes*/
#define RAM_LOW_ADRS        0x00100000          /* VxWorks 下载地址*/
#define RAM_HIGH_ADRS       0x01F00000          /*bootrom 装载地址 */
```

修改 Makefile 对应项与 config.h 中设置一致，需要注意的是 Makefile 文件中 16 进制数不必加 0x 头。

```
ROM_TEXT_ADRS    = FFF00100
```

ROM_SIZE = 00800000

RAM_LOW_ADRS = 00100000

RAM_HIGH_ADRS = 01F00000

修改 wrPpmc824x.h，加入自定义宏，设置 8MB 与 64MB Flash 的基地址与容量：

```
#define FLASH8_BASE  0xFF800000
#define FLASH8_LEN   0x00800000
#define FLASH64_BASE 0x7C000000
#define FLASH64_LEN  0x04000000
```

修改 sysLib.c 中的 sysPhysMemDesc[]结构，添加 SDRAM 段、Flash 段以及 FPGA 段到 MMU 映射中使操作系统可以访问。比如 8MB 的 Flash 可以加入下面这段代码，其它设备添加方法类似。

```
{
(void *) FLASH8_BASE,    (void *) FLASH8_BASE,FLASH8_LEN,
VM_STATE_MASK_VALID    |    VM_STATE_MASK_WRITABLE    |
VM_STATE_MASK_CACHEABLE |    VM_STATE_MASK_GUARDED,
VM_STATE_VALID    |    VM_STATE_WRITABLE    |
VM_STATE_CACHEABLE_NOT    |    VM_STATE_GUARDED
}
```

4.2.2 串口配置

本课题所使用的串口为 MPC8245 内部嵌入的 DUART1 与 DUART2。MPC8245 内嵌 DUART 编程模式与 PC16550D 串口兼容，所以驱动程序可以通过修改风河公司提供的 ns16550Sio 驱动来实现，另外编写汇编串口输出代码辅助调试。

分别复制/target/src/drv/sio/和/target/h/drv/sio/中的 ns16550Sio.c 和 ns16552Sio.h 到 myMPC8245_3，修改 ns16550Sio.c 中#include "drv/sio/ns16552Sio.h"为#include "ns16552Sio.h" 使用当前目录下的头文件。

修改 sysSerial.c 串口初始化文件，将#include "drv/sio/ns16552Sio.h" 改为 #include "ns16552Sio.h"；修改 sysSerialHwInit 函数中的总线频率 duartFreq = 66000000 确保正确的 66MHz 频率。最后在 Makefile 文件中修改 MACH_EXTRA = ns16550Sio.o 将串口驱动编译到操作系统。

编写图 4.5 所示代码，控制 DUART1 发送数据 0xAA。在调试 BSP 的时候可以将此代码段嵌入到 BSP 相关程序中，程序执行到此处时可从串口 1 通道接收到十六进制数 0xAA，辅助程序员确定系统运行状态。

```
lis     r3,   0xfc00          /*config register ulcr=0x80*/
ori     r3,   r3, 0x4511
li      r0,   0x01
stb     r0,   0(r3)
lis     r3,   0xfc00          /*config register ulcr=0x80*/
ori     r3,   r3, 0x4503
li      r0,   0x80
stb     r0,   0(r3)
li      r0,   0x48            /*0x4501 = DMB, 0x4500 = DLB*/
lis     r3,   0xfc00
ori     r3,   r3, 0x4500
stb     r0,   0(r3)
li      r0,   0x00
lis     r3,   0xfc00
ori     r3,   r3, 0x4501
stb     r0,   0(r3)
lis     r3,   0xfc00          /*config register ulcr=0x03*/
ori     r3,   r3, 0x4503
li      r0,   0x03
stb     r0,   0(r3)
lis     r3,   0xfc00          /*config register umcr=0x02*/
ori     r3,   r3, 0x4504
li      r0,   0x02
stb     r0,   0(r3)
lis     r3,   0xfc00          /*config register ufcr=0x07*/
ori     r3,   r3,   0x4502
li      r0,   0x07
stb     r0,   0(r3)
li      r0,   0xAA
lis     r3,   0xfc00
ori     r3,   r3, 0x4500
stb     r0,   0(r3)
```

图 4.5 DUART1 发送数据 0xAA

4.2.3 TFFS 文件系统配置

TFFS 又称 TrueFFS(True Flash File System)，是 M-Systems 公司具有专利的 Flash 管理软件。它本质上是操作系统与设备之间的一个软件层。TFFS 向操作系统提供全块设备管理功能，对于操作系统就好像是个常规的硬盘设备，与此同时它还提供透明的 Flash 媒体管理功能，其功能结构如图 4.6 所示。

TFFS 主要有以下几个特点[14]：

(1) 延长使用寿命。采用高级动态与静态 wear-leveling 算法极大地延长了设备的使用寿命。

(2) 数据完整性保障。提供错误检测与纠正功能（基于 BCH 码和汉明码算法）和 NAND Flash 坏块管理功能。

(3) 废弃数据块收集。通过废弃数据块的收集机制，TrueFFS 回收不再记录有效数据的数据块。正常的闪存可以从可擦除的数据块恢复成只读数据块，TrueFFS 能够辨识闪存状态，并且在可能的情况下，将一个数据单元中的连续数据块作为一个池来维护。为防止过度的使用此技术，废弃数据块的收集机制是作为

wear-leveling 的一部分来运行的。

(4) 可靠性。实行安全算法以保证在突然掉电情况下的数据完整性；它的算法是基于“先写入后删除”而不是“先删除后写入”。覆盖操作是在分离的数据映射区内实现的。只有当整个写操作完成后，原始数据块才会被收回。块映射信息被更新并存储在闪存中，使用“先写入后删除”算法，即使掉电，也能保证块映射信息的一致性。

(5) 透明性。可以在整个设备上面使用文件系统操作来屏蔽底层硬件操作。

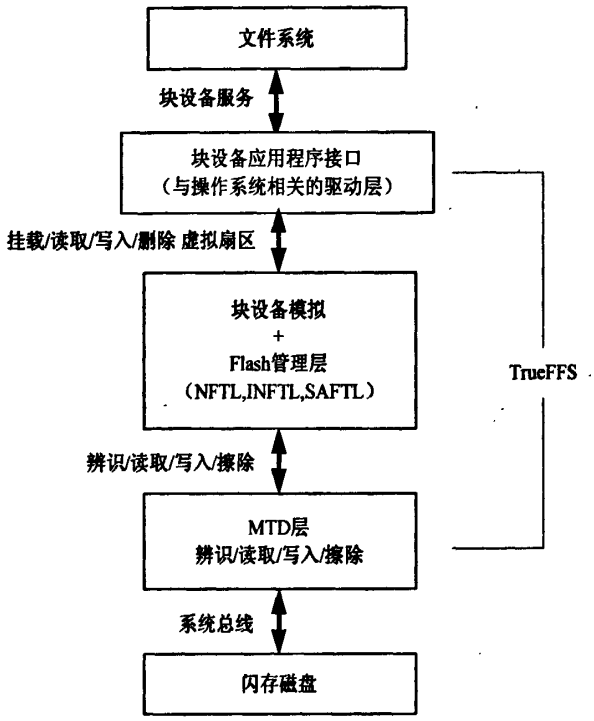

图 4.6 TrueFFS 功能结构

配置过程如下：

首先复制/ target/src/drv/tffs/tffsConfig.c 到 myMPC8245_3 目录下。TFFS 文件系统驱动涉及 5 个文件：sysTffs.c，tffsConfig.c，f2xFlashMem.h，f2xFlashMem.c，f2xFlashMtd.c。

修改 sysTffs.c，该文件包含 socket 驱动：

首先修改#include "tffs/tffsConfig.c"为#include "tffsConfig.c"包含该目录下头文件。

W78M64V 采用的 Flash 编程算法与 AMD (Spansion)的 Flash 编程算法一致[29]，故翻译层选用 INCLUDE_MTD_AMD 并将识别函数指向 f2xFlashMtdIdentify。

```
#define     INCLUDE_TL_FTL      /* FTL translation layer */
#define     INCLUDE_MTD_AMD
#define     amdMTDIdentify      f2xFlashMtdIdentify
```

编写 f2xSetWindow 函数，设定页的基地址与页的大小。由于系统以 4K 页为单位存储，故设定值应为实际值除以 4096，也就是右移 12 位。

```
LOCAL void f2xSetWindow
(
    FLSocket vol
)
{
    vol.window.baseAddress = FLASH64_BASE >> 12;
    flSetWindowSize(&vol, 0x04000000 >> 12);
}
```

编写 flFitInSocketWindow 函数，它返回一片 Flash 的大小。

```
long int flFitInSocketWindow(long int chipSize, int interleaving, long int windowSize)
{
    if (chipSize *interleaving > windowSize)
    /* doesn't fit in socket window */
    {
        int roundedSizeBits;
        /* fit chip in the socket window */
        chipSize = windowSize / interleaving;
        /* round chip size at powers of 2 */
        for (roundedSizeBits = 0; (0x1L << roundedSizeBits) <= chipSize; roundedSizeBits++);
        chipSize = (0x1L << (roundedSizeBits - 1));
    }
    return (chipSize);
}
```

修改 f2xRegister 函数，用于注册 socket 驱动函数。在 f2xRegister 函数中必须设置以下成员：window.baseAddress，cardDetected，VccOn，VccOff，VppOn，VppOff，initSocket，setWindow，setMappingContext，getAndClearCardChangeIndicator，writeProtected，freeSocket。

```
LOCAL void f2xRegister
(
    F2X_GID gid /* group id */
)
```

```
{
    FLSocket vol = flSocketOf(noOfDrives);
    tffsSocket[noOfDrives] = "F2X";
    vol.window.baseAddress = FLASH64_BASE >> 12;
    vol.window.busWidth = 16;
    vol.serialNo = (unsigned)gid;
    vol.cardDetected = f2xCardDetected;
    vol.VccOn = f2xVccOn;
    vol.VccOff = f2xVccOff;
    #ifdef SOCKET_12_VOLTS
        vol.VppOn = f2xVppOn;
        vol.VppOff = f2xVppOff;
    #endif
    vol.initSocket = f2xInitSocket;
    vol.setWindow = f2xSetWindow;
    vol.setMappingContext = f2xSetMappingContext;
    vol.getAndClearCardChangeIndicator = f2xGetAndClearCardChangeIndicator;
    vol.writeProtected = f2xWriteProtected;
    noOfDrives++;
}
```

修改 sysTffsInit 函数：

```
chipBlockSize = 0x10000;/*64k*/
baseAdrs = (char*)TFFS_BASE_ADRS;
regionSize = 0x04000000;
```

添加下面代码依次执行 tffsDrv()初始化 TFFS 文件系统，dosFsInit()准备使用 DOS 文件系统库，usrTffsConfig()挂载整个分区到系统路径/tffs0/下，使 VxWorks 启动后自动挂载 TFFS 文件系统。

```
if (tffsDrv() != OK)
{
    printf("Execute tffsDrv ERROR!\n");
}
if (dosFsInit(NUM_DOSFS_FILES) != OK)
{
    printf("Execute dosFsInit ERROR!\n");
}
```

```
if (usrTffsConfig(0, 0, "/tffs0/") != OK)
{
    printf("Execute usrTffsConfig ERROR!\n");
}
```

4.2.4 其它配置

修改 wrPpmc824x.h，此文件是目标板设备自定义文件，可以定义板上设备地址、中断号、中断优先级等等。修改#define EUMB 0xFC000000 设置 Embedded Utilities Memory Block 地址为 0xFC000000。

修改 m824xAuxClk.c，此文件是系统辅助时钟库文件，为系统提供辅助时钟和时间戳功能。修改 sysAuxClkEnable 函数中 busFreq = 66000000/8，符合系统 66MHz 的总线频率。

修改 sysLib.c，此文件是系统设备初始化程序。修改 sysHwInit 函数中 sysDecClkFrequency = 66000000/DEC_CLK_TO_INC 符合系统 66MHz 总线频率。

修改系统配置文件 config.h。

(1) #undef 所有不必要组件，比如所有有关 NV_RAM、NETWORK 的宏定义。

(2) 添加 TFFS 文件系统所需设置和组件，如下面代码所示

```
#define INCLUDE_TFFS
#define TFFS_BASE_ADRS 0x7C000000
        #ifdef INCLUDE_TFFS
            #define INCLUDE_TFFS_DOSFS
            #define INCLUDE_TFFS_SHOW
            #define INCLUDE_DOSFS
            #define INCLUDE_SHOW_ROUTINES
            #define INCLUDE_TL_FTL
            #define INCLUDE_IO_SYSTEM
            #define INCLUDE_DISK_UTIL
            #define INCLUDE_CBIO
            #define INCLUDE_DISK_CACHE
            #define INCLUDE_DOSFS_CHKDSK
            #define INCLUDE_DOSFS_FAT
            #define INCLUDE_DOSFS_FMT
            #define INCLUDE_DOSFS_MAIN
            #define INCLUDE_DOSFS_DIR_FIXED
```

```
        #define INCLUDE_DOSFS_DIR_VFAT
    #endif
```

定义了 TFFS 文件系统基地址，添加了 DOS 文件系统支持组件和磁盘实用工具组件。

定义串口通道数量和 CONSOLE 通道设置，CONSOLE 通道设为 1 通道，波特率为 57600：

```
#undef NUM_TTY
#define NUM_TTY                 2
#define INCLUDE_WDB
#define INCLUDE_WDB_GOPHER
#undef CONSOLE_TTY
#define CONSOLE_TTY             1
#undef CONSOLE_BAUD_RATE
#define CONSOLE_BAUD_RATE   57600
```

定义启动方式和 WDB 通道参数，WDB 通道也设为通道 1，与 CONSOLE 通道复用，波特率设为 57600；定义 TSFS 与 TFFS 两种启动方式，默认启动方式为 TSFS：

```
/* Startup Type Defination */
#define STARTUP_TSFS        0x01
#define STARTUP_TFFS        0x02
#define STARTUP_TYPE        STARTUP_TSFS
#if(STARTUP_TYPE==STARTUP_TSFS)
    #undef   WDB_TTY_CHANNEL
    #define WDB_TTY_CHANNEL    1
    #undef   WDB_COMM_TYPE
    #define WDB_COMM_TYPE      WDB_COMM_SERIAL
    #undef   WDB_TTY_BAUD
    #define WDB_TTY_BAUD       57600
    #define INCLUDE_TSFS_BOOT
    #undef   INCLUDE_WDB_TSFS
    #define INCLUDE_WDB_TSFS
#elif(STARTUP_TYPE==STARTUP_TFFS)
    #undef   WDB_TTY_CHANNEL
    #define WDB_TTY_CHANNEL    1
    #undef   WDB_COMM_TYPE
```

```
        #define WDB_COMM_TYPE       WDB_COMM_SERIAL
        #define INCLUDE_TSFS_BOOT
        #undef   WDB_TTY_BAUD
        #define WDB_TTY_BAUD        57600
        #undef   INCLUDE_WDB_TSFS
        #define INCLUDE_WDB_TSFS
    #endif
```

定义 DEFAULT_BOOT_LINE 系统默认启动参数：

```
    #if(STARTUP_TYPE==STARTUP_TSFS)
     #define DEFAULT_BOOT_LINE \
     "tsfs=0,0(0,0)host:vxWorks    h=192.168.0.237    e=192.168.0.50    u=vxworks
pw=worksvx f=0x08 tn=MPC8245"
     #elif(STARTUP_TYPE==STARTUP_TFFS)
     #define DEFAULT_BOOT_LINE \
     "tffs=0,0(0,0)host:/tffs0/vxWorks  h=192.168.0.237  e=192.168.0.50  u=vxworks
pw=worksvx f=0x08 tn=MPC8245"
     #endif
```

添加 C++标准函数库支持：

```
    #define INCLUDE_CTORS_DTORS
    #define INCLUDE_CPLUS
    #define INCLUDE_CPLUS_LANG
    #define INCLUDE_CPLUS_COMPLEX_IO
    #define INCLUDE_CPLUS_COMPLEX
    #define INCLUDE_CPLUS_STL
    #define INCLUDE_CPLUS_STRING_IO
    #define INCLUDE_CPLUS_STRING
    #define INCLUDE_CPLUS_IOSTREAMS
    #define INCLUDE_CPLUS_IOSTREAMS_FULL
    #define INCLUDE_CPLUS_DEMANGLER
```

添加 WindView 组件支持：

```
    #define INCLUDE_WINDVIEW
    #define INCLUDE_SEQ_TIMESTAMP
    #define INCLUDE_WVUPLOAD_TSFSSOCK
    #define INCLUDE_WVUPLOAD_FILE
    #define INCLUDE_RBUFF
```

添加符号表等支持：

```
#define SELECT_SYM_TBL_INIT
#define SELECT_COMPILER_INTRINSICS
#define INCLUDE_LOADER
#define INCLUDE_NET_SYM_TBL
#define INCLUDE_SYM_TBL_SYNC
#define INCLUDE_SHELL
#define INCLUDE_SHOW_ROUTINES
#define INCLUDE_STAT_SYM_TBL
#define INCLUDE_SYM_TBL
#define INCLUDE_UNLOADER
#define INCLUDE_MODULE_MANAGER
#define INCLUDE_SYM_TBL_INIT
```

4.2.5 编译生成 bootrom 与 VxWorks 映像

编译生成 bootrom 及 VxWorks 映像有两种方式：命令行方式和 Tornado 工程方式，两种方式各有利弊。命令行方式使用 BSP 内文件创建映像，Tornado 工程方式读取 BSP 内的配置文件生成工程所需文件并进行编译；命令行方式直接体现 BSP 文件修改，而 Tornado 工程方式有可视化的用户配置组件，方便用户设置组件裁剪内核。

(1) 命令行方式：

在 windows 下打开一个控制台，首先需要设置环境变量，进入\host\x86-win32\bin\目录，运行 torVars.bat。

更改路径至 \target\config\myMPC8245_3 下，输入 make bootrom.bin 回车，即可编译生成二进制格式的 bootrom，如图 4.7 所示。

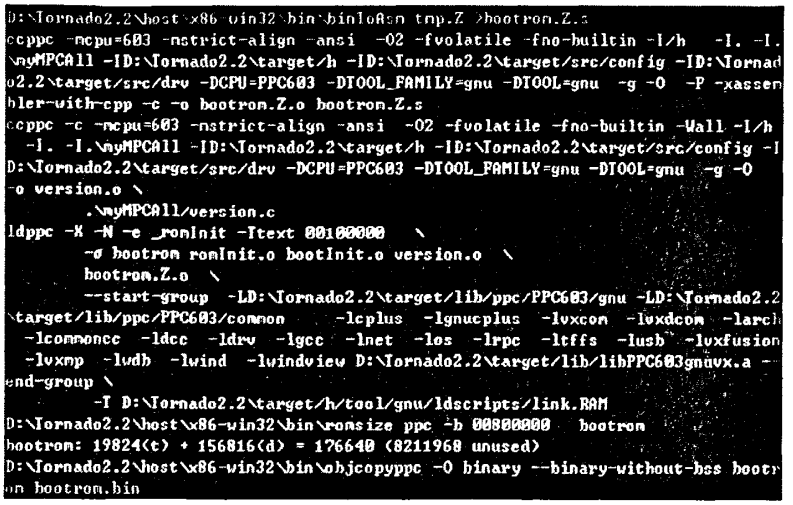

图 4.7 命令行方式编译 bootrom.bin 映像

输入 make vxWorks 回车，即可编译生成 vxWorks 映像及其符号表 vxWorks.sym，如图 4.8 所示。

图 4.8 命令行方式编译 VxWorks 映像

(2) 以 Tornado 工程方式编译 bootrom 与 VxWorks 映像：

打开 Tornado2.2，选择 Create a bootable VxWorks image，如图 4.9 所示。

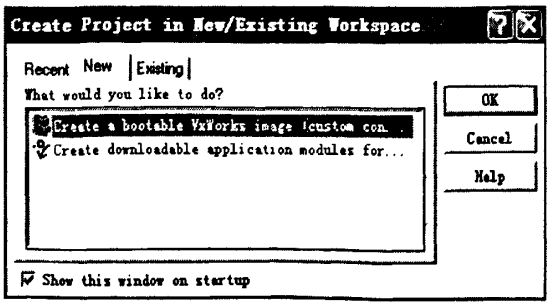

图 4.9 建立新工程选择界面

确定路径与工程名，如图 4.10 所示。

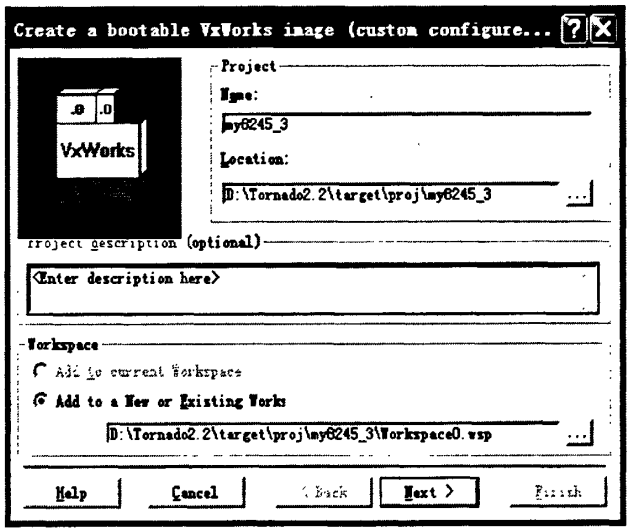

图 4.10 可引导工程建立选项界面

选择 BSP 和编译链接工具，如图 4.11 所示。

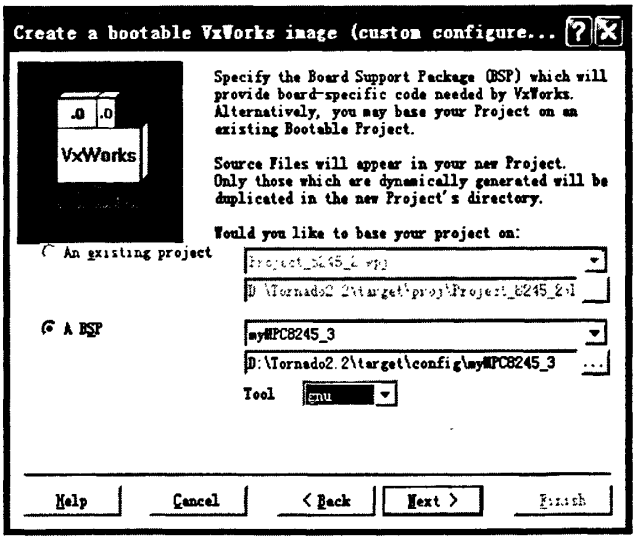

图 4.11 BSP 以及工具链选择界面

点击 Next，直到 Finish。系统会扫描文件依赖性，会弹出错误对话框，如图 4.12 所示。

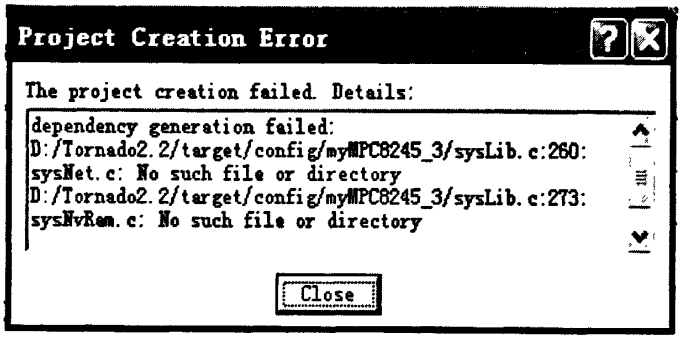

图 4.12 工程文件依赖扫描结果

由于没有使用这两个文件，在 sysLib.c 中注释掉有关 sysNet 和 sysNvRam 的信息或者加入这两个文件都可以解决这个问题，之后即可建立起工程。

选择 VxWorks 标签，在这里增减组件，如图 4.13 所示。所有包含在 config.h 中的组件，工程也会包含进来，所以这里可以不用修改。

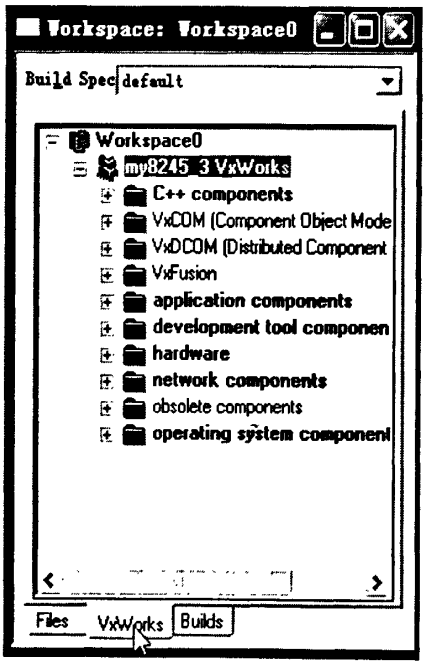

图 4.13 工程组件裁剪选择界面

选择 Builds 标签，如图 4.14 所示。右键点击 default，选择 Build ‘vxWorks’ 即可编译生成 vxWorks 及 vxWorks.sym。

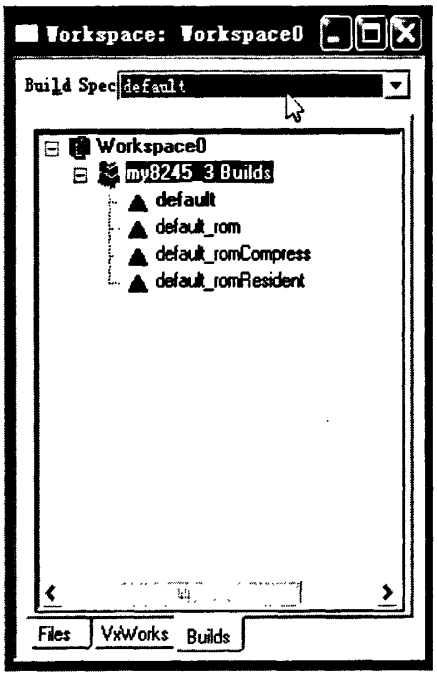

图 4.14 工程编译选择界面

在菜单中选择 Build->Build Boot ROM…

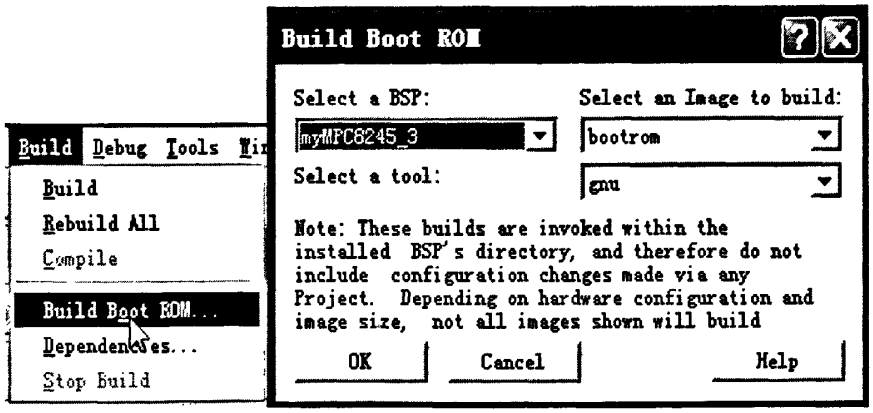

图 4.15 工程编译 bootrom 选择界面

选择 myMPC8245_3，bootrom 和 gnu，点击 OK 即可编译生成 bootrom，如图 4.15 所示。

4.3 系统可靠性设计

为了提升系统的可靠性和稳定性，系统通常需要具有容错设计。而容错最常见的做法是通过冗余技术来实现的。冗余技术就是在系统所需的必备资源以外额外的添加某些资源，当系统的某些部分工作异常时，使用这些额外资源尽量纠正错误确保系统正常运行。冗余技术可以粗分为硬件冗余、软件冗余、编码冗余和

时间冗余四类[21]。

硬件冗余是通过使用追加硬件资源备份来抑止因硬件出错而导致的系统失效。硬件冗余目前已经研究得较为完善，由于使用追加硬件资源，所以具有效率高、容错效果明显的优点。但是这也带来了一些不可避免的缺点，首先可能增大设备体积，在很多情况下不符合特定环境下的要求；其次是灵活性下降，不易对其设计进行动态调整；最后是硬件代价相对较高[2]。硬件冗余的典型例子是三模冗余机制(TMR)，TMR 是把三个完全相同的运算电路的输出与一个比较电路相连，通过对三者的输出采用少数服从多数的表决原则决定最终的输出。

传统的软件容错技术主要是采用多样性的冗余来解决软件本身出现的故障，其核心思想是：对于同一功能的实现可能有多种方法，通过不同的方法得到的结果经过表决机制的判断，得出一个可接受的结果来达到容错的目的。设计多样性的典型技术包括了 N 版本软件设计技术、恢复块技术、N 自检程序设计[22]。软件容错的优点是灵活度高，容错方案可以动态调整与变更，且相比于硬件容错，容错代价较低，缺点是系统复杂性增加、性能略有下降[9]。

信息冗余是指在数据中通过附加冗余的信息以达到故障检测、故障屏蔽和容错的目的。信息冗余的基本方法是通过差错控制编码，在原始数据的信息码元上追加冗余的一些校验码元，这些冗余的码元与信息码元之间通过某种编码算法相互映射，在解码时检验信息码元和校验码元之间的映射关系，一旦传输或存储过程中发生差错，则信息码元和校验码元之间的关系将被破坏，从而可以发现错误或者纠正错误。信息冗余的优点是所增加的冗余度比上两种方法低，可提供故障的自检测、自定位、自纠错能力，缺点是会产生解码延迟，降低系统在正常工作时的运算效率。使用较多的编码方法有奇偶检验码、循环冗余校验码和汉明纠错码[10,23]。

时间冗余是指通过在时间上的冗余来减少硬件冗余的容错方式，其基本思想是重复执行计算以检测故障。航天应用领域实时性要求较高，时间冗余应用较少，它多应用于一些非实时领域，可以减少硬件与软件冗余带来的成本、功耗、体积以及重量等开销。时间冗余根据其重复计算的级别是在指令级还是在程序段级，可被分为指令复执及程序卷回两种。时间冗余可被用来检测瞬时故障、永久故障以及纠错[23]。

4.3.1 SDRAM 硬件检错与纠错

MPC8245 与 SDRAM 数据接口有两种模式：Registered buffer 模式（MPC8245 默认模式）和 In-line buffer 模式，其逻辑框图如图 4.16 与图 4.17 所示。

Registered buffer 模式允许更高的存储器接口频率，但是在读操作时有一个周

期的延迟。这是由于 Registered buffer 模式内部锁存器先将数据锁存，然后再输出，所以会有一个周期的延迟；因为有锁存，处理器核在取锁存数据的同时外部的数据就可以将新的数据放在数据线上，所以只要时钟频率足够高，就可以有更高的存储器接口频率。

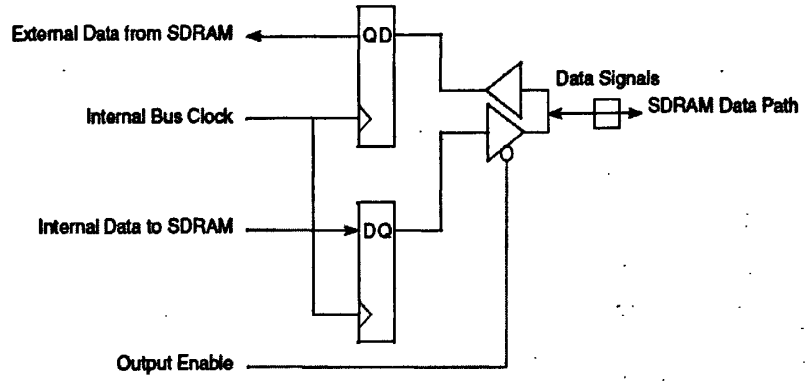

图 4.16 Registered buffer 模式逻辑

In-line buffer 模式可以在处理器核内部总线和外部的 SDRAM 数据总线上进行 ECC 或奇偶校验的产生和检查，相较于注册模式，In-line buffer 模式又有一个时钟的延迟。

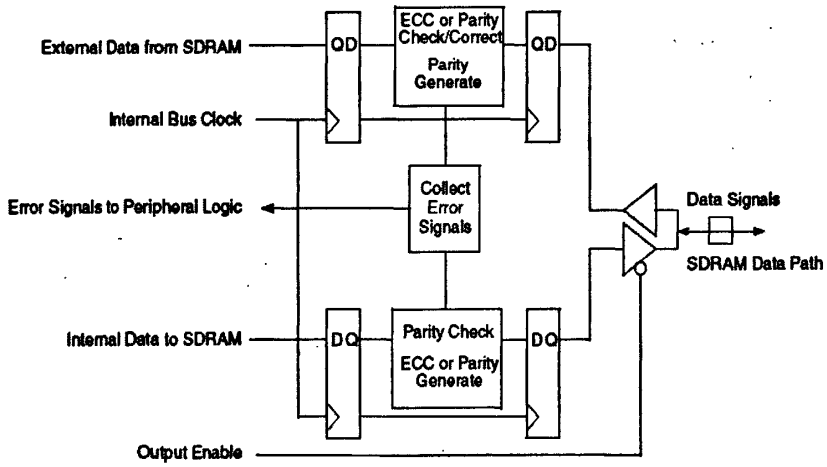

图 4.17 in-line buffer 模式逻辑

MPC8245 的 SDRAM 接口支持普通模式的奇偶校验(Parity)和读-修正-写奇偶校验(RMW Parity)这两种模式。读-修正-写奇偶校验对于小于 64 位的数据写操作，首先会锁住写入数据并读出存放的 64 位数据，对读出数据进行奇偶校验并与写入数据进行合并，算出新的校验值，最后将新得出的值写回存储器。对于奇偶校验的检查和产生，32 位数据总线需要附加 4 位的奇偶校验位，64 位的数据总线需要附加 8 位的奇偶校验位；而 ECC 校验则只对 64 位的数据总线有效。

影响奇偶校验或者 ECC 校验的参数由表 4-1 给出：

表 4-1 校验参数表

Bit Name	Register and Offset	Bit Number in Register
SDRAM_EN	MCCR1 @F0	17
PCKEN	MCCR1 @F0	16
INLINE_WR_EN	MCCR2 @F4	19
INLINE_RD_EN	MCCR2 @F4	18
INLINE_PAR_NOT_ECC	MCCR2 @F4	20
BUF_TYPE[0]	MCCR4 @FC	22
BUF_TYPE[1]	MCCR4 @FC	20
RMW_PAR	MCCR2 @F4	0
Memory parity/ECC enable	ErrEnR1 @C0	2
ECC Multi-bit error enable	ErrEnR2 @C4	3

所以对这些参数不同的设置可以设置不同模式的奇偶校验或者 ECC 校验，全部配置由表 4-2 给出：

表 4-2 校验模式配置表

SDRAM_EN	PCKEN	INLINE_WR_EN	INLINE_RD_EN	INLINE_PAR_NOT_ECC	BUF_TYPE[0]	BUF_TYPE[1]	RMW_PAR	Memory Parity/ECC Enable	ECC Multi-Bit Error Enable	Description
0	0	0	0	0	0	1	0	0	0	Registered, no ECC or parity
0	1	0	0	0	0	1	0	1	0	Registered buffer parity
0	1	0	0	0	0	1	1	1	0	Registered buffer RMW parity
0	0	0	0	0	1	0	0	0	0	In-line, no parity
0	0	1	1	1	1	0	0	1	0	In-line, parity enabled
0	0	1	1	1	1	0	1	1	0	In-line, RMW parity enabled
0	0	0	1	0	1	0	1	1	1	In-line, ECC enabled

明确了错误检查机制后，在系统初始化时设置需要的错误检查模式，即设置上电配置信号线电平或者修改 romInit.s 文件中相应寄存器，便可开启错误检查提高系统可靠性。

4.3.2 RS 编码冗余

在空间环境中，由于闪存Flash容易受到单粒子翻转而改变原有逻辑状态，导致数据出错。虽然TFFS闪存文件系统可以提供一定的容错能力，但是在空间应用

中稍显薄弱。所以本文考虑采取基于滞后校验方法的RS编码对Flash内数据进行检错纠错，进一步加强整个系统的可靠性和稳定性。

1959年1月21日，Irving Reed和Gus Solomon向“Journal of the Society for Industrial and Applied Mathematics” 提交了只有五页纸的论文“olynomial Codes over Certain Finite Fields”。他们在这篇论文中谨慎地提出了构造一类新型纠错编码的方法。然而在随后的四十年里，这种叫Reed-Solomon(RS)纠错编码在CD播放、广播电视、磁体存储、无线通信以及深空通信等等无数个系统中大放异彩。事实上，RS码已经成为了20世纪电信革命密不可分的一部分[24]。

RS 码是 Reed-Solomon 码的简称，是一类对于随机性和突发性错误纠正能力都很强的线性分组码。它是一种扩展的非二进制 BCH 码，也是最大的最小汉明距离可分码（MDS 码）。航天应用中常选用能够纠正 16 个符号的错误（每个符号 8bit）的 RS(255，223)码来作为 RS 校验编码。编码后可以允许错误的数据进入处理器，通过软件来实现对数据的编码和解码。下面介绍参考的 RS 编解码算法[20]：

(1) RS 编码算法及实现

RS码的编码算法按照变化域的不同进行分类，可以分为时域算法和频域算法。本文参考基于时域算法的RS编码，使用软件算法实现基于生成多项式 $g(x)$ 的除法电路。

在RS(n，k)码中，它的最小汉明距离为d，在有限域GF(2^m)上能够纠正t个差错的本原RS码的生成多项式为：

$$g(x)=(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t}) \tag{4-3}$$

设待编码的信息矢量为 $(m_{n-2t-1},m_{n-2t-2},...,m_0)$，即信息多项式 $m(x)$ 为

$$m(x)=m_{n-2t-1}x^{k-1}+m_{n-2t-2}x^{k-2}+\cdots+m_1x+m_0, m_i\in GF(2^m) \tag{4-4}$$

RS 码属于系统码，若码字前面 k 位为信息元，后面 2t 位为校验元，则称之为系统码。码字多项式 $c(x)$ 可由式(4-5)给出，其中 $r(x)$ 是用 $m(x)x^{2t}$ 除以 $g(x)$ 所得的余数，称为校验元多项式。

$$c(x)=m(x)x^{n-k}+r(x)=m(x)x^{n-k}+m(x)x^{2t}\bmod(g(x)) \tag{4-5}$$

式(4-5)中

$$r(x)=r_{n-k-1}x^{n-k-1}+r_{n-k-2}x^{n-k-2}+\cdots+r_1x+r_0 \tag{4-6}$$

多项式 $r(x)$ 相应的系数 $(r_{n-k-1},r_{n-k-2},...,r_0)$ 就是信息元 $(m_{k-1},m_{k-2},...m_1,m_0)$ 的校验元。

以上RS码的时域编码算法，可归结为以下三个步骤[25]：

① 用 x^{n-k} 乘以信息多项式 $m(x)$，得到 $m(x)x^{n-k}$；

② 用生成多项式 $g(x)$ 除 $m(x)x^{n-k}$，可得到余式 $r(x)$，提取其各项系数，得到所要求的校验元；

③ 联合 $m(x)x^{n-k}$ 和 $r(x)$ 即得到码字多项式 $c(x)$。

(2) RS 译码算法及实现

RS码的译码算法比编码算法要复杂的多，RS译码算法主要也可分为两类：时域译码算法和频域译码算法。本文采用时域译码算法来实现应用于该航天系统的RS(255,223)译码器[26]。

① RS译码的基本思想

假设发送码字多项式 $c(x)$ 为：

$$c(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\cdots+c_0 \tag{4-7}$$

接收码字多项式 $r(x)$ 为：

$$r(x)=r_{n-1}x^{n-1}+r_{n-2}x^{n-2}+\cdots+r_0 \tag{4-8}$$

以 $e(x)=e_{n-1}x^{n-1}+e_{n-2}x^{n-2}+\cdots+e_0$ 表示噪声码字，则有

$$r(x)=c(x)+e(x) \tag{4-9}$$

RS译码的基本思想就是从接收多项式 $r(x)$ 确定差错发生的位置和差错位置上的错误值，即求出错误图样 $e(x)$ 的系数 $e_i(i=0,1,...,n-1)$。从 $r(x)$ 中减去 $e(x)$ 就得到译码后的正确码字，也就是 $c(x)=r(x)-e(x)$。

若信道产生错误个数为t个，错误图样 $e(x)$ 则可表示为

$$e(x)=Y_1x_1+Y_2x_2+\cdots+Y_{e-1}x_{e-1}+Y_ex_e=\sum_{i=1}^{e}Y_ix_i \tag{4-10}$$

式(4-10)中是 x_i 第 i 个错误出现的位置，并定义 $x_i=\sigma^i$，其中 Y_i 是错误值。

② RS译码的步骤

a. 由接收多项式 $r(x)$ 计算伴随式分量 $S_j,(j=1,2,\cdots,2t)$；

由RS码的定义知，$g(\alpha^i)=0,(i=1,2,\cdots,2t)$。把 $\alpha^j,(j=1,2,\cdots,2t)$ 代入接收多项式 $r(x)$，得

$$S_j=r(\alpha^j)=r_{n-1}(\alpha^j)^{n-1}+r_{n-2}(\alpha^j)^{n-2}+\cdots+r_0 \tag{4-11}$$

如果 $S_j=0$ 则认为接收无误；倘若 $S_j\neq 0$，则可由 S_j 找出错误图样。

因为

$$r(\alpha^j)=c(\alpha^j)+e(\alpha^j)=e(\alpha^j) \tag{4-12}$$

可以得到

$$S_j=r(\alpha^j)=e(\alpha^j)=e_{n-1}(\alpha^j)^{n-1}+e_{n-2}(\alpha^j)^{n-2}+\cdots+e_0 \tag{4-13}$$

即 $S_j = \sum_{i=1}^{e} Y_i x_i$。

通过式(4-11)可以计算出接收码字的伴随式分量。

b. 采用BM算法，由 S_j 求出错误位置多项式 $\sigma(x)$；

按遍码的纠错能力，可以定义如下 $\sigma(x)$：

$$\sigma(x) = \prod_{i=1}^{t}(1 - xX_i) = \sigma_t x^t + \sigma_{t-1}x^{t-1} + \cdots + \sigma_1 x + 1 \tag{4-14}$$

如能求得 $\sigma(x)$ 的各项系数，其根显然为 $X_t^{-1}, X_{t-1}^{-1}, \cdots, X_1^{-1}$。错误位置 X_i 即为其根的倒数。

通常实际错误数 r 小于 t，即 $\sigma_{r+1} = \sigma_{r+2} \cdots = \sigma_t = 0$，则错误位置多项式可改写为 $\sigma(x) = \sigma_r x^r + \sigma_{r-1}x^{r-1} + \cdots + \sigma_1 x + 1$。

给式(4-14)两边同乘以 $Y_l X_l^{j+r}$，其中 $j = 1,2,\cdots 2t, l = 1,2,\cdots,r$，可得

$$Y_l X_l^{j+r}\sigma(x) = Y_l X_l^{j+r}(\sigma_r x^r + \sigma_{r-1}x^{r-1} + \cdots + \sigma_1 x + 1) \tag{4-15}$$

令式(4-15)中 $x = X_l^{-1}$，X_l^{-1} 由于是 $\sigma(x)$ 的根，可得到

$$\sigma_r S_j + \sigma_{r-1}S_{j+1} + \cdots + \sigma_1 S_{j+r-1} + S_{j+r} = 0 \tag{4-16}$$

c. 用Chien搜索法寻求错误位置；

假设 $r(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \cdots + r_0$，检验第一位 r_{n-1} 是否有误，这相当于检验 $\alpha^{-(n-1)}$ 是否为 $\sigma(x)$ 的根。所以求出 $\sigma(x)$ 以后，首先计算 $\sigma_1\alpha, \sigma_2\alpha^2, \cdots \sigma_t\alpha^t$，检查其累加和是否为-1。若是-1，则说明 r_{n-1} 错误，α^{n-1} 是错误位置数；否则 r_{n-1} 正确[27]。

d. 计算错误值；

得到错误位置后，代入伴随式分量 S_j，求解线形方程组 $S = YX$。

e. 接收多项式减去错误多项式，完成纠错。

本课题中可以先将 VxWorks 操作系统映像进行 RS 编码后放入 TFFS 文件系统所在 Flash，当 bootrom 运行到搬移 VxWorks 时加入 RS 译码程序对已进行 RS 编码的 VxWorks 操作系统映像进行解码后搬移至 RAM 相应地址再启动操作系统。通过这种经过 RS 编码的文件可以进一步加强系统纠错能力，提高系统可靠性。

4.4 本章小结

本章描述了针对本课题 MPC8245 最小系统的 BSP 设计方案，着重描述了系统初始化、串口、TFFS 文件系统以及其它一些系统参数的配置方法，接着介绍了根据 BSP 编译生成 bootrom 和 VxWorks 映像的方法。本章最后给出了系统可靠性设计方案，包括 SDRAM 的 parity/ECC 的配置方法，以及通过 RS 编码冗余增强 Flash 可靠性的方法。

第五章 VxWorks BSP 调试与测试

VxWorks 操作系统的 bootrom 映像编译好了需要烧写到嵌入式系统启动设备中才能启动，本系统启动设备为 8MB Flash，实际上使用了 16MB Flash 的一半空间。启动代码会将镜像代码段和数据段加载至 SDRAM 中运行，所以需要首先调试好 SDRAM 和 Flash 设备。此外由于 Flash 设备中的数据位只能由 1 改写为 0 而无法由 0 改写为 1，写数据之前必须先进行块擦除使该块每一位恢复到 1 的状态。为了减少对 Flash 设备的擦写，延长使用寿命，可以使用 CodeWarrior 软件将 bootrom 映像加载到 SDRAM 中，然后手动修改 PC 指针，从 SDRAM 中映像入口地址运行，开始调试 VxWorks BSP 以及操作系统。

5.1 CodeWarrior 软件介绍

CodeWarrior 是 Metrowerks 公司开发的一款完整的支持多种处理器的集成开发环境。CodeWarrior 包括构建平台和应用所必须的所有主要环境，包含工程管理工具、搜索工具、编辑器、源码浏览器、编译连接系统和调试器等等。它具有下面几个优点[28]：

(1) 跨平台开发

CodeWarrior 软件可以开发适用于多种操作系统的软件，也可在不同主机中开发同一软件工程。CodeWarrior 集成开发环境可以运行于多种主流操作系统中，包括 Windows、Macintosh、Solaris 和 Linux。在不同平台下集成开发环境的界面看上去都是一样的，方便用户跨平台开发。

(2) 多种语言支持

CodeWarrior 集成开发环境支持多种编程语言开发软件，它不仅像 C，C++，Java 这样的高级语言而且也支持多数处理器的内联汇编。

(3) 始终一致的开发环境

在不同架构的处理器之间移植软件再也不用学习使用新的开发工具了，CodeWarrior 集成开发环境可以支持许多常见的桌面和嵌入式处理器家族，包括 X86，PowerPC，MIPS 等等。

(4) 支持插件工具

通过查件工具的添加可以扩展 CodeWarrior 集成开发环境对新增功能服务的兼容性。CodeWarrior 目前支持编译器、链接器、前向链接器、后向链接器、选项面板、版本控制等等插件。插件工具使 CodeWarrior 可以处理不同的语言和支持不

同的处理器。

5.2 Memory 测试与验证

CodeWarrior 提供了硬件诊断(Hardware Diagnostics)功能，可以对地址空间进行读写测试以诊断硬件故障。它提供了 3 项功能，分别是 Memory Read/Write，Scope Loop 以及 Memory Tests。

Memory Read/Write 功能可以对单一地址进行读写，读写位宽可以是 8 位，16 位或者 32 位。利用此工具对 SDRAM 地址范围某地址写入数据，而后再读出判断内存读写是否正常。测试可以配合示波器观察地址总线、数据总线、片选等信号的跳变情况，检查处理器与总线上器件的硬件连接有无问题。通过一系列调试，处理器与 SDRAM、Flash 以及 FPGA 的硬件连接均无问题。

Scope Loop 功能可以连续以某一速率对地址空间某一地址进行读或者写，来测试连续读取以及写入的稳定性。利用此工具对 SDRAM 进行高速读取以及写入，来测试 SDRAM 读写时序是否配置正确。

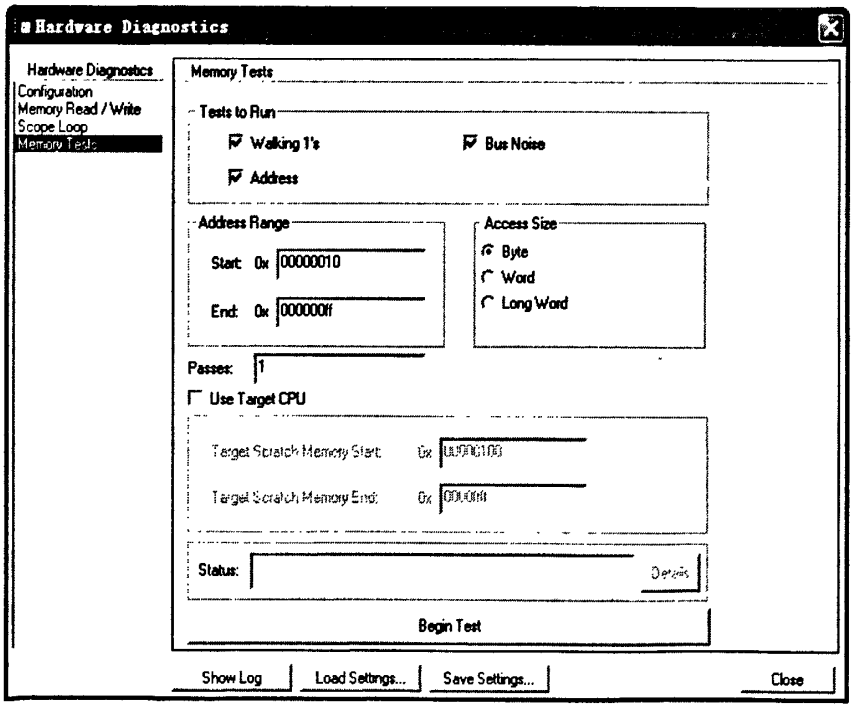

图 5.1 Memory Test 界面

最后一项功能是 Memory Tests，使用它可对存储器地址范围进行三项测试，分别为 Walking 1's，Bus Noise，Address，设置界面如图 5.1 所示。

Walking 1's 测试对内存地址每一位进行写一或写零测试，可检测以下 3 种错误：地址线被拉高或者拉低，数据线被拉高或者拉低，内存长时间不更新导致数

据无法保持。

Bus Noise 通常是由于频繁大量数据存储器访问引起的。Bus Noise 测试强制连续对地址线和数据线交替写 0 和 1 操作来产生噪声，测试读写数据稳定性。

Address 测试。当一块物理内存被多次包含于一个逻辑内存，若发生混迭则会造成物理内存大于实际物理内存的假像，此测试可检测系统是否存在内存混迭。

本课题中的 SDRAM 经过调试后，最终通过了 Memory Test 的测试验证，可以说明 SDRAM 时序正确，与处理器之间数据传输准确无误。

5.3 Flash 擦写

CodeWarrior 提供了 Flash Programmer 程序，可以通过此功能实现它所支持的 Flash 芯片的一些操作，其界面如图 5.2 所示。

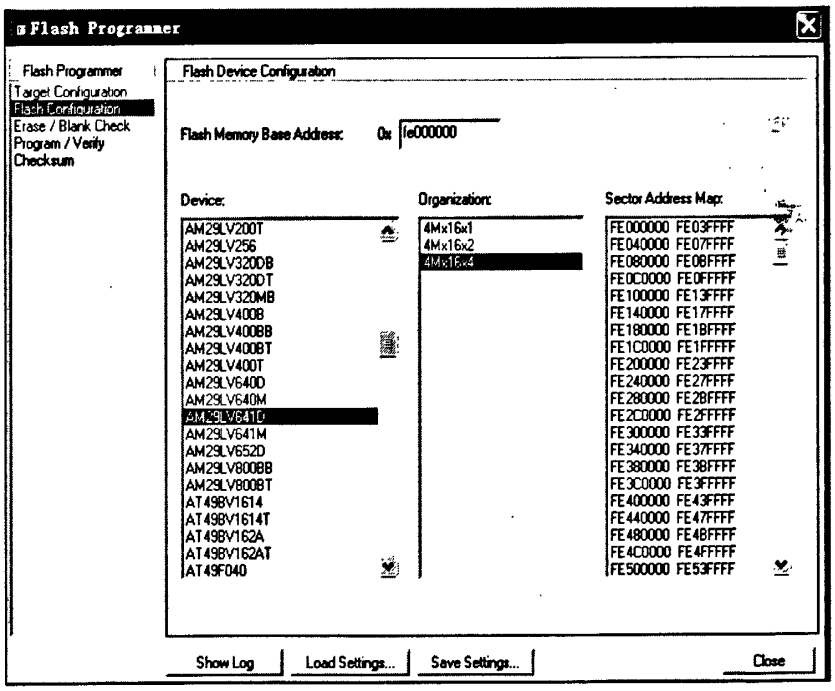

图 5.2 Flash Programmer 界面

Flash Programmer 程序提供了全片/块擦除、编程/验证、计算校验和功能。功能虽然强大，但是提供的设备列表里面没有 W78M64V 这款组织形式的 Flash。虽然通过修改 FPDeviceConfig.xml 也可以自定义 Flash 设备组织形式区块地址映像，为了更好获知 Flash 设备操作状态，本文选择自己编写 Flash 擦写代码。

Flash 擦写代码由 C 语言编写，通过 CodeWarrior 工程生成可执行文件，建立好的工程如图 5.3 所示。

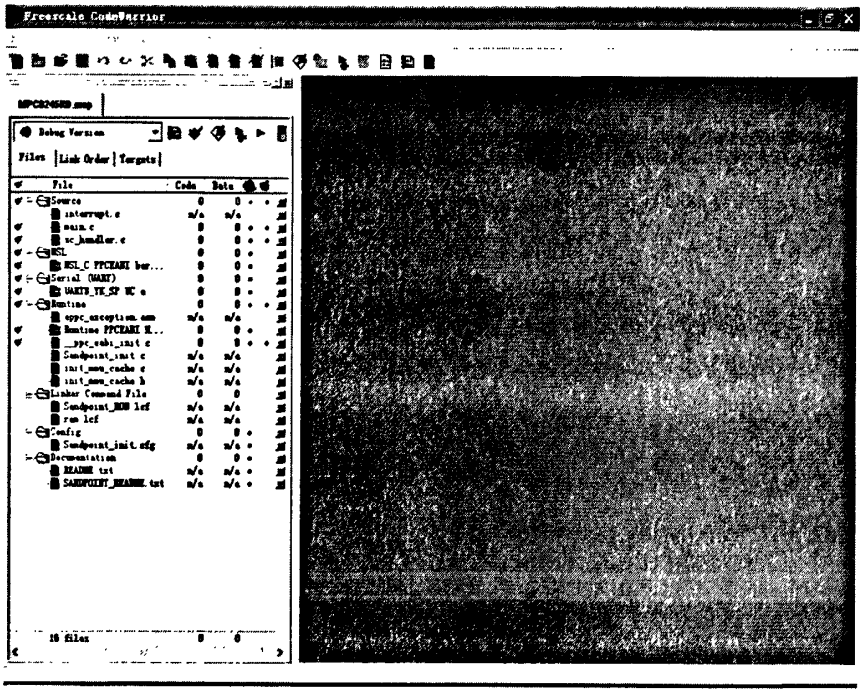

图 5.3 CodeWarrior 工程创建完成界面

工程建立完成后需要对它进行配置，如图 5.4 所示。按下 Alt+F7 进入 Debug Version Setting 选项，在 Debugger 菜单中选择 EPPC Debugger。将 Use Memory Configuration 选项勾掉，不使用 Memory 配置文件；同时需要修改程序安装目录下 PowerPC_EABI_Support\Initialization_Files\Host\Sandpoint_init.cfg 文件。

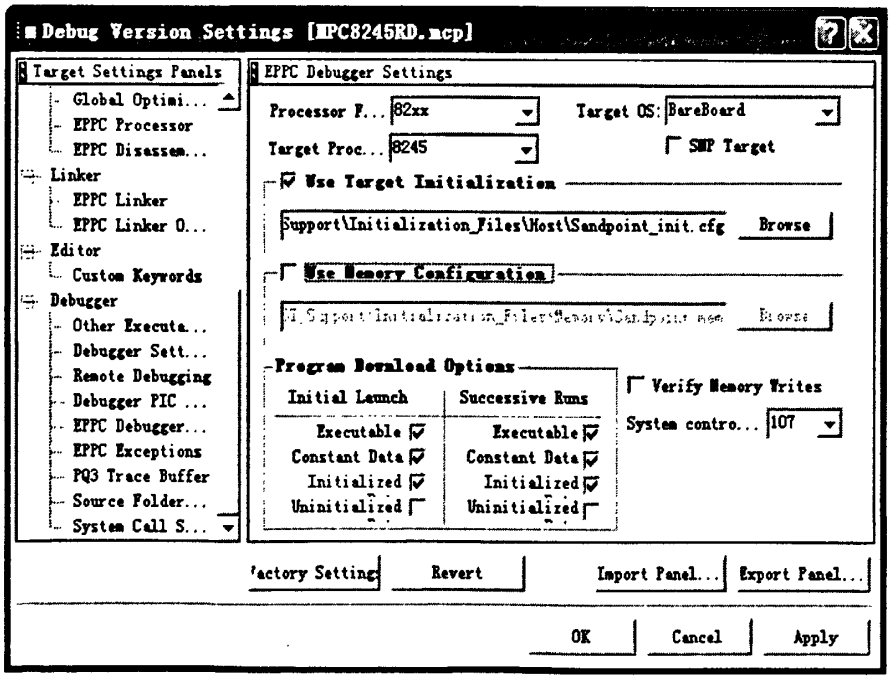

图 5.4 工程 Debug Version Setting 选项界面

SandPoint_init.cfg 是单板初始化文件，可参照 VxWorks BSP 中的 romInit.s 文

件进行修改。需要注意的是 PowerPC 处理器默认为大端模式，所以数据需要按照大端格式来书写。

配置好初始化文件后开始修改 main.c 文件，在其中主要加入 Flash 全片擦除程序和写入程序。

表 5-1 Flash 编程算法

命令	周期数	总线周期											
		一		二		三		四		五		六	
		地址	数据	地址	数据	地址	数据	地址	数据	地址	数据	地址	数据
读取	1	RA	RD										
复位	1	任意	F0										
单片擦除	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
写入	4	555	AA	2AA	55	555	A0	PA	PD				

W72M64V 所采用的 Flash 编程算法与 AMD(Spansion)的 Flash 编程算法一致[29]，从表 5-1 可以看出，单片擦除操作需要 6 个总线周期，分别是：

(1) 向 Flash 偏移地址 0x555 写数据 0xAA；

(2) 向 Flash 偏移地址 0x2AA 写数据 0x55；

(3) 向 Flash 偏移地址 0x555 写数据 0x80；

(4) 向 Flash 偏移地址 0x555 写数据 0xAA；

(5) 向 Flash 偏移地址 0x2AA 写数据 0x55；

(6) 向 Flash 偏移地址 0x555 写数据 0x10。

表 5-1 中 RA 表示需要读取的地址，RD 表示读出的数据，PA 表示需要写入的地址，PD 表示需要写入的数据。数据列中表示的数据为 16 位，在这 16 位 DQ15~DQ0 中，高 8 位 DQ15~DQ8 只在 RD 中有效，在其它操作中其值无关紧要。

对于本系统，Flash 数据总线位宽 64 位，总线地址需要 64 位对齐，所以算出 W72M64V 的偏移地址需要左移三位，如 0x555<<3 得到 0x2AA8。所以对 W72M64V 擦除的第一个 Cycle 就是向偏移地址 0x2AA8 写入数据 0x00AA00AA00AA00AA，后面 5 个周期以此类推。擦除完成后需要对 Flash 进行复位操作，即对任意地址写入 0x00F000F000F000F0。

Flash 写入程序与擦除程序实现原理基本一致，但是每次 64 位写入操作完成后需要检查 DQ7(Data#Polling)位是否置位，置位则表示操作完成，若没有则检查 DQ5(Exceed Timing Limits)位是否置位，若没有置位则表示操作未执行完成，返回继续读取；若 DQ5 置位表示操作完成，再次读取一次 DQ7 判断 Flash 是否有错误，流程如图 5.5 所示。

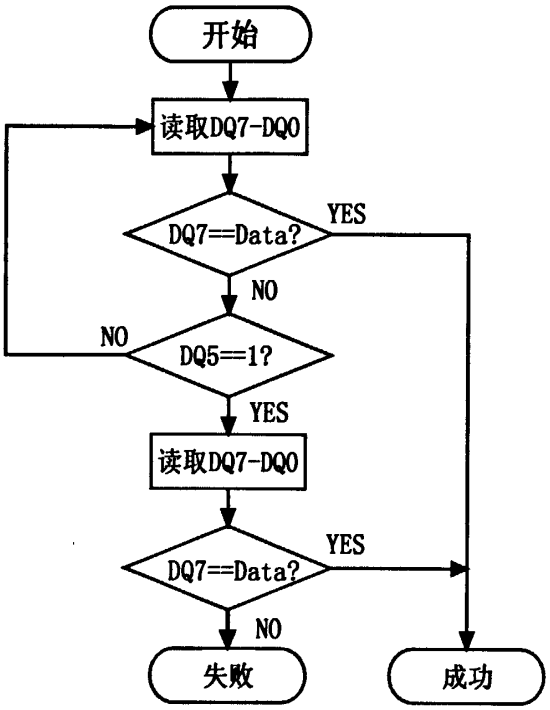

图 5.5 Flash 写入程序流程

写入全部完成后需要对 Flash 复位以返回读取状态。本文所述 Flash 写入程序利用 CodeWarrior 提供的 Fill Memory 功能将 VxWorks 映像先搬移到内存中，而后写入程序会将内存中的这段数据烧写这到 Flash 中特定地址。

执行 Flash 全片擦除时，可以从 CodeWarrior 的 Data 菜单下的 View Memory 窗口中选中 Flash 地址区查看擦除完成情况，当所有单元变成 0xFFFF 后完成擦除。结合本课题，使用 Fill Memory 将 bootrom.bin 二进制映像搬移到内存中，再通过 Flash 写入函数将 bootrom 写入 MPC8245 启动地址 0xFFF00100，写入完成后从 View Memory 窗口可以观察到 Flash 中写入的 bootrom 数据，至此 bootrom 已经成功写入 Flash。

5.4 VxWorks BSP 调试

首先需要在编译 BSP 中加入-g 选项加入调试信息，用 CodeWarrior 软件打开编译生成的 bootrom 映像，软件会自动建立工程，搜索 bootrom 所依赖的文件。某些文件系统找不到，需要手动设定路径。在 Bootrom 已经写入 Flash 后，通过刚才建立好的工程就可以从 Debug 菜单下的 Attach to Process 连接至处理器进行 BSP 的调试了。CodeWarrior 调试功能十分全面，很实用的功能是可以查看和修改处理器的寄存器值，还可以方便的修改内存中的数据和指令，全面掌控系统的状态。经常注意的寄存器有 PC、LR，这两个寄存器可以控制指令的执行流程，PC 指向

当前指令，LR 存放跳转地址，当遇到 blr 指令时会把 LR 的值赋给 PC，程序跳转至新地址执行。为了配合 JTAG 调试，还可以编写串口输出汇编程序，嵌入到 BSP 代码中来输出调试信息。

调试过程中发现系统会跑飞，最终停留到 0x800 处，这里是 MPC8245 浮点异常中断处理向量地址。通过设断点、单步调试等调试方法，发现在中断安装函数 excIntConnect 中会调用系统库函数 blCompute。因为没有源码，调试时该函数为反汇编代码，它会将 RAM_HIGH_ADRS 的值左移 6 位再算术右移 6 位比较是否相同，所以 RAM_HIGH_ADRS 超过 0x01FFFFFF 时，不能安装中断向量，导致后面串口中断时程序跳至 0x500 处。由于没有正确安装中断向量，此处内存为随机数，反汇编出来为一条浮点指令，又因为此时还有没有打开浮点使能，进而会产生浮点错误而跳至 0x800 处，产生了错误循环。将 RAM_HIGH_ADRS 设置为 0x01F00000 即可解决这个问题。

BSP 调试完成后，系统会一直在 VxWorks 下载引导任务中循环，这是由于还没有配置好 VxWorks 的下载通道。下面介绍通过串口通道下载引导 VxWorks 操作系统的方法。

使用串口方式下载引导 VxWorks 操作系统首先需要配置 Target Server，配置界面如图 5.6 所示。

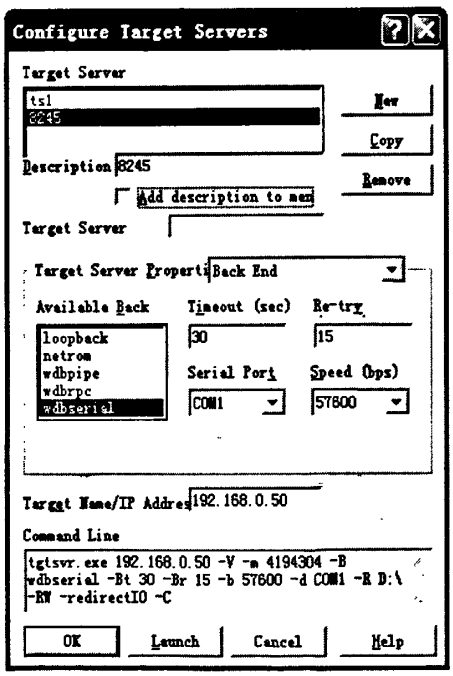

图 5.6 Target Server 配置界面

连接方式选择 wdbserial，串口通道为 1 通道，波特率设为 57600。

在 Memory Cache Size 选项中自定义 Cache 为 4096。

Target Server File System 选项中选择路径为 VxWorks 所在目录。

Console and Redirection 中选中图 5.7 中两项。

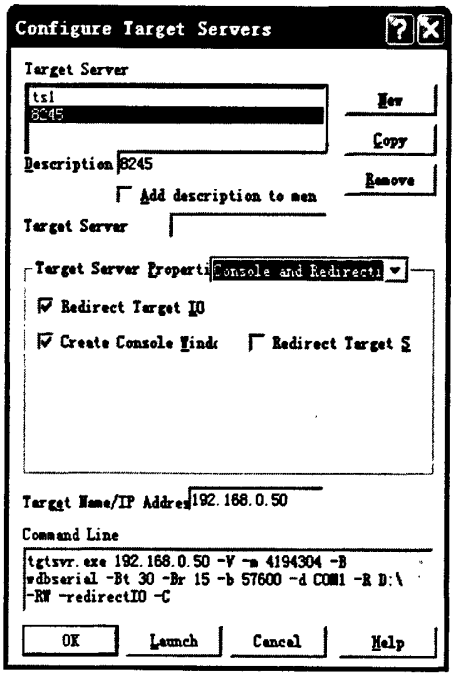

图 5.7 勾选重定向 IO 和创建 Windows 控制台选项

配置好后点击 Launch 启动 Target Server。这时启动目标板，主机会通过 Target Server 与目标板建立连接，重定向控制台中也会显示出 boot 信息，在控制台中键入@后开始通过 TSFS 下载 VxWorks 映像，下载完成后 VxWorks 操作系统启动完成。在控制台中输入 i 并回车可以看到当前系统中运行的任务，进行到这里系统已经可以正常运行，证明 BSP 的引导过程正确无误。

5.5 VxWorks 启动后设置与测试

通过串口下载 VxWorks 映像的方式虽然方便，但是串口的速度过慢，下载过程需要好几分钟。为了加快启动速度，可以将 VxWorks 映像复制到 TFFS 文件系统中，通过 TFFS 启动系统。需要修改 bootrom 使系统能够自动从 TFFS 加载 VxWorks 而无需输入@符号。这一步只需要删除 config.h 中 TFFS 启动选项里面的 #define INCLUDE_TSFS_BOOT 宏定义。这是因为在 bootConfig.h 中，若定义了 #define INCLUDE_TSFS_BOOT 且 CONSOLE 通道与 WDB 通道为同一通道时就会取消自动启动。而后修改 config.h 中#define STARTUP_TYPE STARTUP_TFFS，将系统默认启动类型设置为 TFFS 方式。重新编译好 bootrom 烧写至 Flash 启动地址，重新上电启动 VxWorks。这是会出现无法加载 VxWorks 的错误，这是因为这时 TFFS 文件系统还没有配置，而且系统中也没有 VxWorks 操作系统映像。需要手动输入 TSFS 启动信息从 TSFS 方式启动，按 c 键回车修改启动方式为 TSFS，

按照 config.h 中 DEFAULT_BOOT_LINE 里面 TSFS 启动项的设置填写好启动信息，键入@符号启动系统。

VxWorks 启动后还不能直接使用 TFFS 文件系统，还需要对 W78M64 这片 Flash 进行 TFFS 文件系统初始化才能正常使用，可以通过以下几步操作。

(1) 确保 Flash 已被擦除，若没有擦除，使用自行编写的 Flash 擦除程序擦除全片 Flash，然后重新启动系统。

(2) 使用 tffsDevFormat 0, 0 命令对设备进行格式化。

(3) 使用 usrTffsConfig 0, 0, "/tffs0"将文件系统挂载到/tffs0 路径。

通过这 3 个步骤，TFFS 文件系统配置完成，可以通过访问/tffs0/来访问这块 64MB 的 Flash。

在 Tornado 中打开一个 shell，键入：cp "/tsfs/vxWorks","/tffs0/vxWorks" 将 VxWorks 操作系统映像通过 TSFS 复制到 Flash 中，若希望系统启动能够自动载入符号表，则需要键入：cp "/tsfs/vxWorks.sym","/tffs0/vxWorks.sym" 将 VxWorks 操作系统符号表复制到 Flash 中。这时重新启动目标板，系统仍无法加载，出现 usrTffsConfig Failed。这是由于修改了 sysTffs.c 中的 sysTffsInit 函数，在里面加入了 usrTffsConfig 实现开机自动挂载。但是在 bootConfig.c 中 tffsLoad 函数中 tffsDrv 函数会调用 sysTffsInit 函数，从而调用 usrTffsConfig 函数，如果下面还会再次调用 usrTffsConfig 函数，将导致 usrTffsConfig Failed 错误。解决办法就是删除 tffsLoad 函数里面的 usrTffsConfig 调用，避免重复挂载导致的错误。重新编译后再次从串口启动，连接好 Target Server，在 Shell 中敲入 devs，会看到/tffs0 设备已经挂载。再次将重新编译的 VxWorks 映像以及符号表复制到 Flash 中，重新启动系统，VxWorks 操作系统正常的从 TFFS 启动了。

此时，对系统的配置也就全部完成了。该系统在编写的用户任务控制下可以完满的完成通过 FPGA 处理外部信号，系统稳定性得到验证。设计对空间环境下的辐射有一定的抵抗能力，首先硬件的选型均选用工业级以上，很大程度上增强了系统对总剂量积累效应的抵抗能力。存储设备容易受到单粒子翻转效应的影响，本文所述的 SDRAM 经过硬件奇偶校验与 ECC 可以检查或纠正错误，有效的减少了 SDRAM 出错导致系统异常的机率。使用 TFFS 文件管理系统管理 Flash 可以有效的进行 Flash 磨损控制，延长 Flash 的使用寿命，使用 RS 码对 Flash 中的文件进行编码可以大大减小 Flash 受单粒子翻转效应的影响。设计在一定程度上增强了星载计算机抵抗空间辐射的能力，为系统稳定可靠运行打下了基础。

5.6 本章小结

本章描述了使用 CodeWarrior 调试软件配合示波器对系统进行调试与测试的

方法。介绍了调试完成后对 VxWorks 操作系统启动方式以及 TFFS 文件系统进行配置的方法，并通过对系统硬件的测试与 VxWorks 操作系统的测试，得出整个系统正常运行的结论。

第六章 结束语

6.1 论文工作总结

在对系统 VxWorks BSP 的设计过程中，我首先学习了 VxWorks 嵌入式操作系统以及 PowerPC 处理器的相关知识，查阅了相关手册和帮助，从最底层了解了本系统的核心软硬件。接下来需要了解星载计算机的通常设计方法，了解了国内外提出和使用的设计技术，结合本课题需求加入了增强可靠性的设计和适用于星载的某些特性。最后通过 CodeWarrior 仿真调试软件配合 USB TAP 在线仿真调试器，对硬件系统以及 VxWorks 操作系统的 BSP 进行在线调试，确保了系统的正常运行。现将论文所做的工作总结如下：

(1) 根据课题需求和星载嵌入式系统特点，结合本系统硬件平台，给出了 VxWorks 操作系统 BSP 的设计方案。

(2) 对 VxWorks 操作系统 BSP 进行修改，使操作系统能够正常运行在 PowerPC 最小硬件平台上。

(3) 编写 Flash 擦写程序，通过 CodeWarrior 调试软件实现 Flash 擦写功能。

(4) 配置 TFFS 文件系统，使系统能够使用文件系统管理 Flash。

(5) 配置启动选项，使系统支持从 TSFS 和 TFFS 两种设备启动。

(6) 通过 CodeWarrior 仿真调试软件对系统及 VxWorks BSP 进行调试，确保系统状态正常、运行无误。

在研究本课题的过程中也遇到了许多麻烦，比如 JTAG 连接错误。这个低级错误让课题组耽误了不少时间。由于目标板设计与 USB TAP 的 JTAG 连接线不匹配，导致某两条信号线接反，USB TAP 无法中断和复位处理器，致使调试无法进行。经过课题组耐心的排错，最终解决了这个问题才使课题顺利进行。这样的错误告诫我，在今后的科研和工作中必须保持严谨的态度以及怀疑的精神，减少不必要的麻烦。

6.2 进一步研究展望

伴随着科学技术的发展，嵌入式计算机的处理器也不断的更新换代，本课题所选用的 MPC8245 处理器特性早已不是最新技术，VxWorks 操作系统版本也更新到了 6.8，使之支持更多处理器和更新的特性。但是星载计算机注重的是其稳定性，在保证任务完成的前提下，稳定性比运行频率更为重要。为了保证处理器能够稳

定工作，本课题中 MPC8245 被配置为 264MHz，而不是其标称的 333MHz 或者更高。

由于需求与时间关系本课题还存在其可靠性并不完善的问题，再进一步的研究中需要解决下列问题：

(1) 启动设备可靠性不足。启动设备为 8MB Flash，它易受到单粒子翻转效应导致数据错误，从而无法引导 VxWorks 操作系统。可以将启动设备换为 ROM，它具有更好的可靠性。

(2) 没有设计硬件看门狗，一旦处理器出现异常则会很麻烦。硬件看门狗可以由 FPGA 实现，通过检查处理器错误状态复位处理器来实现异常恢复。

(3) 加入遥感注入程序与自检程序，使得星载计算机可以实现在轨软件更新以及系统自检能力，延长星载计算机的使用寿命。

致谢

首先，我要感谢我的导师许录平教授。许老师渊博的专业知识、严谨求实的治学态度、踏实认真的工作作风和随和待人的态度时刻影响着我，使我受益匪浅。而且，许老师创造了一个宽松有序的工作学习环境，无论是在科研学习方面，还是生活做人方面，许老师都给了我许多无私的教诲，激励着我一直前进。

感谢我的师兄苏哲、张华。研究生阶段他们给予我许多帮助，使我开拓了思路，增长了知识，明白了许多人生道理，是我学习的榜样。

感谢课题组的陈瑞，没有他在硬件方面的调试，本课题不会这么顺利的完成。遇到问题时，我们会互相鼓励、共同探讨来寻求解决方案，与他一起攻关的日子让我难忘。

感谢实验室的左季、李秋生、王德奎、王婷、刘素君、张桂英，尤其是我的舍友冀爱民、刘玉和、董斌。与他们一起的日子使我非常充实，他们给我留下三年美好的回忆。

感谢我的亲人，正是他们在背后默默支持，给我了前进的动力。让我一直坚持着自己的理想，走出自己的道路。

最后我要诚挚的感谢评阅本文的专家和学者们，谢谢你们付出的辛劳！

参考文献

[1] Scarpulla J, Yarbrough A. What Could Go Wrong? The Effects of Ionizing Radiation on Space Electronics. Technical Report 2, CrossLink, 2003

[2] 孙鹏. 星载计算机系统软件容错技术研究. 合肥：中国科学技术大学硕士学位论文. 2007, 2-3

[3] 张雪. 基于MPC8260的微小卫星星载计算机操作系统设计. 北京：中国地质大学硕士学位论文. 2008, 3-4

[4] 王新升，孙汉旭，徐国栋等. 基于ARM处理器的星载计算机系统研究. 北京邮电大学学报，2005，28(4)：23-26

[5] 谢浩，张健. 基于SPARC V8的嵌入式星载计算机. 微计算机信息，2008，24(2-2)：38-40

[6] http://www.qnx.com/products/intl/neutrino_rtos/. QNX Neutrino RTOS Overview, 2010

[7] 段星辉，华建文，代作晓. 基于CAN和μC/OS-II的星载仪器管理系统设计. 微计算机信息，2009，25(4-2)：34-35

[8] 杨雅. 星载嵌入式计算机系统可靠性技术研究. 长沙：国防科学技术大学硕士论文. 2005, 10-11

[9] 陈国林. 星上带容错功能的计算机引导系统的研究和实现. 北京：中国科学院计算技术研究所硕士学位论文. 2005, 3-4

[10] 张钰，正阳明，黄正亮等. 皮卫星星载计算机存储模块的容错结构设计. 宇航学报，2008，29(6)：2057-2061

[11] 李建立. 空间辐射环境下软件实现的硬件故障检测技术研究. 长沙：国防科技大学硕士学位论文. 2008, 1

[12] FreeScale Semiconductor, Inc. MPC8245 Integrated Processor Reference Menual(Rev.3), 2005

[13] http://www.embhelp.com/drew/mypage/VxWorks.htm. VxWorks介绍及编程

[14] http://wenku.baidu.com/view/3ed02d21af45b307e87197bd.html. Tornado和VxWorks嵌入式实时操作系统及其开发环境

[15] 孔祥营，柏桂枝. 嵌入式实时操作系统VxWorks及其开发环境Tornado. 北京：中国电力出版社，2002

[16] 徐少毅，李君龙. 基于VxWorks的BSP概念与开发. 电子产品世界，2002：

26-28

[17] 乔从连. VxWorks 系统的 BSP 概念及启动过程. 舰船电子对抗，2005，28(1)：61-64

[18] 张忠，樊留群. VxWorks 在 S3C2410 上的 BSP 设计. 微型电脑应用，2005，21(10)：16-19

[19] 周雪赞，张宏. 基于 Vxworks 操作系统的 BSP 开发. 工业控制计算机，2004，17(2)：33-34

[20] 王黎容. 天地电子邮件系统的设计. 西安：西安电子科技大学硕士论文. 2010，11-12，27-30

[21] 龚健，杨孟飞. 一种星载控制计算机智能容错方法. 空间控制技术与应用，2008，34(6)：29-33

[22] 刘海峰. 星载软件容错设计及验证技术. 中国电子科学研究院学报，2009，4(3)：313-316

[23] 王霆. 星载并行计算机系统容错设计与分析. 长沙：国防科技大学工程硕士论文. 2008，14-20

[24] 黄勤. Reed-Solomon 码软译码技术研究. 南京：东南大学硕士学位论文. 2007

[25] 韩作生，袁东风. RS 码频域编译码的计算机模拟，通信学报. 1994，15(6)：104-112

[26] Stephen B Wicker，Vijay K Bhargava. Reed-Solomon code and their application. IEEE Press，1994

[27] 叶清贵，刘宇怀. R-S 码纠错算法的软件实现. 华东师范大学学报，2005，(04)：98-101

[28] FreeScale Semiconductor, Inc. CodeWarrior Development Studio IDE 5.7 User's Guide，2006

[29] FreeScale Semiconductor, Inc. Creating an External Flash Algorithm，2006-2008

[30] M.Violante，M.L.Esposti. A low-cost solution for developing reliable Linux-based space computers for on-board data handling. IEEE，2009

[31] Wind River Systems,Inc. VxWorks BSP Developer's Guide,5.5，2002

[32] Wind River Systems,Inc. VxWorks Programmer's Guide,5.5，2002

[33] Wind River Systems,Inc. Tornado User's Guide,2.2. 2002

[34] 周启平，张杨. VxWorks 下设备驱动程序及 BSP 指南开发. 北京：中国电力出版社，2004

[35] 陈智育. VxWorks 程序开发实践. 北京：人民邮电出版社，2004

[36] White Electronic Designs Corp. W78M64V-XSBX Datasheet(Rev.2). 2005

[37] White Electronic Designs Corp. W72M64VK-XBX Datasheet(Rev.0). 2005

[38] White Electronic Designs Corp. WEDPN16M72VR-XB2X Datasheet(Rev.1). 2005

[39] FreeScale Semiconductor, Inc. MPC8245 Integrated Processor Hardware Specifications. 2007

[40] FreeScale Semiconductor, Inc. Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture. 2005

在读期间研究成果

在硕士研究生期间取得的研究成果如下：

参加某研究所无线资源管理器平台研制。

参加某 863 项目 X 射线脉冲星仿真实验平台研制。

撰写《星载 PowerPC 最小系统及 VxWorks BSP 设计》科研技术报告。