摘 要

Nios II 是一种基于 FPGA 开发的软核 CPU，它以代码的形式固化在 FPGA（现场可编程逻辑器件）内。占用一定的片内资源，实现强大的 CPU 功能。相对于常见的硬核 CPU，Nios II 的成本更低，灵活性更高，实现更简易，而且开发更快速。在医疗电子、消费电子、通信产业和其它多种行业中都有很好的应用前景。但由于 Nios II 是 Altera 公司开发的一种新技术，目前世界上只有 Xilinx 公司提供了类同的选择，所以国内外的研究和使用还存在很多空白和未知的领域，从而限制了 Nios II 的广泛应用。

在实现了 Nios II 的基本应用的基础上，本课题主要研究了 Nios II 系统的几大高级用法：多 CPU，多 Master 和高速数据传输。并且有选择地采用了其中的两种多 Master 和高速数据传输，开发了一个完整的 Nios II 系统。实现了 Nios II 控制下的数据传输。

完成本课题，锻炼了 FPGA 全流程开发技能，从模块设计到功能仿真，再到时序验证;也加强了硬件开发能力，从原理图设计到 PCB 设计，以及后期调试。

同时，对于基于软核 CPU 的嵌入式系统，完成了从原理验证到运用于实际开发的过程，最后实现了在医疗电子器械中的产品化。

关键词 软核 CPU；Nios II；USB2.0；红外通讯

Abstract

Nios II is a type of softcore CPU which bases on the development of FPGA.It composed of HDL's codes, occupys some resources inside chips, and realizes power CPU. by contrast with usual CPUs,Nios II has many virtues.

For example, less costs,more flexible,easier realization,and faster speed of development. It has good apect of application in medical electronic、custom electronic、communication、and so on.

Because Nios II is a new tecnology, and only Xilinx co.Ltd has similar product,so it has many blank edge and spect which need be researched and development,and limited Nios II 's application abroad.

Based on basis application of Nios II ,the topic mainly research three advanced application: multi CPU,multi-Master,and high-speed data transfer. Choosing two type of it including multi-Master and high-speed data transfer, I developed whole Nios II system,realized wireless communication in control of Nios II .

I advanced ability in a whole flow of FPGA development from module design to function simulation and timing examination,and strengthened development ability of hardware,from design of theory drawing to PCB,and debug hardware circuit.

At the same time,for emebedded system based softcore CPU,I completed the full course from validate theory to development, realized product design at last.

Keywords softcore CPU, Nios II , IRDA, USB2.0

第1章 绪论

1.1 课题背景

本课题来源于迈瑞生物医疗电子有限公司监护研发部的预研项目。医疗电子产品属于嵌入式应用的一个重要门类，嵌入式系统的核心是 CPU，以往采用的嵌入式 CPU 都是硬核器件，比如摩托罗拉公司的 Coldfire、PowerPC 等。最近两年出现了在 FPGA 片内实现的软核 CPU。硬核 CPU 以硅片的形式提交给用户，而软核 CPU 则以设计文件的形式交付给用户，用户再根据自己的需求将其固化在 FPGA 芯片内这种新出现的技术有着非常广阔的应用前景。

1.1.1 课题的理论背景

软核 CPU 的本质是一段实现特定功能的代码，需要一款合适的 FPGA 作为载体。Nios II 若配置在 Cyclone II 中，只占用 35 美分的逻辑资源，若配置在 Stratix II 中，可以实现超过 200DMIPS 的高性能表现。在 Cyclone II 中也可以达到超过 100DMIPS 的表现，足以与中低密度的 ASIC 形成竞争。Nios II 不同于 16 位指令集的 Nios，它是真正的 32 位软核 CPU，具有 32 位的指令集、32 位的数据路径和 32 位的寻址空间。Nios II 脱胎于 Nios，并分化出了三个版本。一种是 Nios II /f，快速型；第二种是 Nios II /s，标准型；第三种是 Nios II /e，经济型。这三种产品都针对特定的价格和性能范围进行了优化，并都使用同样的指令集架构(ISA)，具有 100%二进制代码兼容性。快速版本的 Nios II 具有最强的性能，其运算能力在 175MHz 时钟频率下可达大约 200 Dhrystone MIPS(DMIPS)。它需要占用中等数量的 FPGA 资源，大约 1180 个逻辑单元。标准版本的性能略低于快速版本，其 175MHz 时钟频率下的运算能力为大约 90 DMIPS，但只需要占用 800 个逻辑单元。经济版本是三种版本中性能最弱的，其运算能力在 190MHz 时钟频率下仅能达到 28 DMIPS，但它所占用的 FPGA 资源最少，只需大约 400 个逻辑单元。可以根据自己设计的特定需求选择合适的软核。如果逻辑资源充裕，那么快速版本显然是首选。Nios II 处理器能通过 Quartus II 开发软件中的 SOPC Builder 系统开发工具添加到设计者的系统中。

RISC(精简指令集计算机)技术的诞生缘于 CISC(复杂指令集计算机)中含有大量效率低下、使用频率过低的复杂指令。指令简单，即优化了寄存器的使用，可腾出大量的寄存器资源用作通用寄存器；每个时钟周期就可完成一条指令，降低了对系统时钟频率的约束；处理器硬件上的结构，因而也可以得到简化[1]。

在集成电路发展初期，电路设计都从器件的物理版图设计入手，后来出现了集成电路单元库，使得集成电路设计从器件级进入逻辑级，这样的设计思路使大批电路和逻辑设计师可以直接参与集成电路设计，极大地推动了 IC 产业的发展。但集成电路不是最终产品，它只有装入整机系统才能发挥它的作用。IC 芯片是通过印刷电路板(PCB)等技术实现整机系统的。尽管 IC 的速度可以很高、功耗可以很小，但由于 PCB 板中 IC 芯片之间的连线延时、PCB 板可靠性以及重量等因素的限制，整机系统的性能受到了很大的限制。随着系统向高速度、低功耗、低电压和多媒体、网络化、移动化的发展，系统对电路的要求越来越高，传统集成电路设计技术已无法满足性能日益提高的整机系统的要求。同时，由于 IC 设计与工艺技术水平提高，集成电路规模越来越大，复杂程度越来越高，已经可以将整个系统集成为一个芯片[2]。

正是在需求牵引和技术推动的双重作用下，出现了将整个系统集成在一个集成电路芯片上的系统芯片(System On Chip，简称 SOC)概念。系统芯片与集成电路的设计思想是不同的，它是微电子设计领域的一场革命。SOC 是从整个系统的角度出发，把处理机制、模型算法、软件(特别是芯片上的操作系统——嵌入式的操作系统)、芯片结构、各层次电路直至器件的设计紧密结合起来，在单个芯片上完成整个系统的功能。它的设计必须从系统行为级开始自顶向下。

很多研究表明，与 IC 组成的系统相比，由于 SOC 设计能够综合并全盘考虑整个系统的各种情况，可以在同样的工艺技术条件下实现更高性能的系统指标。芯片集成系统，主要有三个关键的支持技术：软、硬件的协同设计技术。面向不同系统的软件和硬件的功能划分理论。硬件和软件更加紧密结合不仅是 SOC 的重要特点，也是今后 IT 业发展的一大趋势。IP 模块库问题。IP 模块有三种，软核、固核和硬核。模块界面间的综合分析技术，这主要包括 IP 模块间的胶联逻辑计术和 IP 模块综合分析及其实现技术等[3]。

微电子技术从 IC 向 SOC 转变不仅是一种概念上的突破，同时也是信息技术发展的必然结果，通过以上三个支持技术的创新，必将导致又一次以系统芯片为特色的信息技术上的革命[4]。目前，SOC 技术已经崭露头角，21

世纪将是 SOC 技术真正快速发展的时期。

1.2 本课题研究的目的及意义

基于 Nios II 的嵌入式系统的可应用领域很广，包括网络、无线通信、医疗器械、交通、消费电子、工业控制、军事和航空航天等，这一点已经在第一节的实例阐述中给出了细节的证明。在巨大的市场需求面前，兼之 Nios II 本身的众多优势，使 Nios II 的应用具备了巨大的发展空间[25]。

集成电路是中间产品，必须将它装入到整机上才能发挥其作用，具体地说是通过印刷电路版(PCB)来集成到整机上去的。由于 PCB 板中各种 IC 芯片之间的连线延迟较大，再加上 PCB 板体积大、重量大、可靠性差等原因，使得整机系统的性能及可靠性受到严重影响。随着高性能系统对系统复杂度、处理速度、功耗、功能多样化的要求，在现代信息处理与通信系统如网络、多媒体、移动通信和其它电子系统中迫切需要开发高性能的片上系统[5]。而 Nios II，正是一个优秀的选择。

对嵌入式开发而言，开发基于 Nios II 系统的好处很多。

嵌入 FPGA 中的 CPU 而不是外加 CPU 可以使整个系统布线更为简单，而且 FPGA 芯片管脚设置的灵活性降低了 PCB 布线的复杂性。甚至可能减少所需的 PCB 层数，从而简化系统，降低成本。

基于 Nios II 的系统实现了设计的可复用性，即使是同样一块开发板，根据载入的代码不同，可以实现不同的应用。这也是缩短开发周期、加快产品上市时间的一条捷径。

开始本课题时，Altera 的 Nios II 刚刚发布不过数月，在开发时间上具备优势。再者，如果开发后性价比合理，便可以将其应用在监护仪乃至其它各产品的主控板中。总之，本课题与医疗电子器械行业的实际产品开发紧密相连，并有着远大的应用前景。

1.3 国内外相关技术发展现状

要谈 Nios II，先要说说第一代的 Nios。16 位的软核处理器 Nios 在不到三年的时间内，在全球卖出了超过 1 万 3 千套开发套件，打破了此前任一款 IP 的销售记录，并成为 FPGA 软核处理器的标准，Nios 处理器也因此被 EDN 杂志评为“2003 年 100 个热点产品”之一。Nios 的客户群中包括许多大名鼎鼎的名字：西门子、摩托罗拉、安捷伦、罗克韦尔、飞利浦、IBM 等等。

Nios 处理器被广泛应用于消费类数字显示、数码相机、DVD 播放器、机顶盒和计算机外设等。

1.2.1 国内外开发实例

简要介绍一下国内外的几款 Nios 开发实例。

(1) 无线阅读器 把 Nios 嵌入低成本 FPGACyclone 中，有小型化和低功耗两大特点，执行数据处理和无线接收的功能。

(2) 光多业务节点 这是一款阿尔卡特开发的用于同步数字序列传输的产品。利用 Nios 的定制指令功能和同步多 Master 总线结构(Avalon)，实现了吉比特(Gigabite)速率码流的可靠传输。

(3) ISDN 协议处理器 飞利浦公司把 Nios 处理器运用于综合业务数字网的协议处理器的在线更新。这一应用极大的提高了视频会议和 IP 网关服务的可靠性。

(4) 数码相机中的通用控制处理器 柯达公司在某型号的数码相机中成功的应用了 Nios 处理器，作为 DC 的控制中枢。

(5) 国内 对 Nios 的最高应用可能是某保密项目，该款应用同时使用了八个 Nios 软核，把 Avalon 总线的多 master 功能发挥的淋漓尽致。

可以说，Nios 是相当成功的。对比 Nios，它的第二代 Nios II 处理器更是青出于蓝而胜于蓝。Nios II 具备更强大的灵活性、更高的性能、更低的成本、占用更少的资源，能够满足各种嵌入式应用的性能要求。并且，Altera 提供了更易用的 Nios II 开发套件提供给客户。Altera 曾将其两代软核做了对比，结果显示 Nios II 只用了 50%的逻辑单元就实现了 2 倍于 Nios 的性能。

1.2.2 国内外开发现状

关于 Nios II 的开发现状。国外方面，在 google 上用关键词 Nios II 搜索，只有一些概要性的介绍，并无具体的开发实例报道。也曾经试图通过中国期刊网，以及国内较有影响力的水木清华等高校 BBS 的嵌入式开发版寻找相关资讯，但是少有收获。这也是正常的。Nios II 是 Altera 新近推出的一款通用 32 位软核 RISC CPU。Altera 网站上提供的所有官方开发辅助文档都是 04 年 5 月的第一版，而本人 04 年 6 月就进入迈瑞生物医疗电子有限公司正式启动这个课题。两者的间距时间非常短。可以说，所有打算使用 Nios II 的开发者都处于一个起步阶段。作为 Altera 免费提供的 IP(intellectual

Nios 处理器被广泛应用于消费类数字显示、数码相机、DVD 播放器、机顶盒和计算机外设等。

1.2.1 国内外开发实例

简要介绍一下国内外的几款 Nios 开发实例。

(1) 无线阅读器 把 Nios 嵌入低成本 FPGACyclone 中，有小型化和低功耗两大特点，执行数据处理和无线接收的功能。

(2) 光多业务节点 这是一款阿尔卡特开发的用于同步数字序列传输的产品。利用 Nios 的定制指令功能和同步多 Master 总线结构(Avalon)，实现了吉比特(Gigabite)速率码流的可靠传输。

(3) ISDN 协议处理器 飞利浦公司把 Nios 处理器运用于综合业务数字网的协议处理器的在线更新。这一应用极大的提高了视频会议和 IP 网关服务的可靠性。

(4) 数码相机中的通用控制处理器 柯达公司在某型号的数码相机中成功的应用了 Nios 处理器，作为 DC 的控制中枢。

(5) 国内 对 Nios 的最高应用可能是某保密项目，该款应用同时使用了八个 Nios 软核，把 Avalon 总线的多 master 功能发挥的淋漓尽致。

可以说，Nios 是相当成功的。对比 Nios，它的第二代 Nios II 处理器更是青出于蓝而胜于蓝。Nios II 具备更强大的灵活性、更高的性能、更低的成本、占用更少的资源，能够满足各种嵌入式应用的性能要求。并且，Altera 提供了更易用的 Nios II 开发套件提供给客户。Altera 曾将其两代软核做了对比，结果显示 Nios II 只用了 50%的逻辑单元就实现了 2 倍于 Nios 的性能。

1.2.2 国内外开发现状

关于 Nios II 的开发现状。国外方面，在 google 上用关键词 Nios II 搜索，只有一些概要性的介绍，并无具体的开发实例报道。也曾经试图通过中国期刊网，以及国内较有影响力的水木清华等高校 BBS 的嵌入式开发版寻找相关资讯，但是少有收获。这也是正常的。Nios II 是 Altera 新近推出的一款通用 32 位软核 RISC CPU。Altera 网站上提供的所有官方开发辅助文档都是 04 年 5 月的第一版，而本人 04 年 6 月就进入迈瑞生物医疗电子有限公司正式启动这个课题。两者的间距时间非常短。可以说，所有打算使用 Nios II 的开发者都处于一个起步阶段。作为 Altera 免费提供的 IP(intellectual 的开发者都处于一个起步阶段。作为 Altera 免费提供的 IP(intellectual

Nios 处理器被广泛应用于消费类数字显示、数码相机、DVD 播放器、机顶盒和计算机外设等。

1.2.1 国内外开发实例

简要介绍一下国内外的几款 Nios 开发实例。

(1) 无线阅读器 把 Nios 嵌入低成本 FPGACyclone 中，有小型化和低功耗两大特点，执行数据处理和无线接收的功能。

(2) 光多业务节点 这是一款阿尔卡特开发的用于同步数字序列传输的产品。利用 Nios 的定制指令功能和同步多 Master 总线结构(Avalon)，实现了吉比特(Gigabite)速率码流的可靠传输。

(3) ISDN 协议处理器 飞利浦公司把 Nios 处理器运用于综合业务数字网的协议处理器的在线更新。这一应用极大的提高了视频会议和 IP 网关服务的可靠性。

(4) 数码相机中的通用控制处理器 柯达公司在某型号的数码相机中成功的应用了 Nios 处理器，作为 DC 的控制中枢。

(5) 国内 对 Nios 的最高应用可能是某保密项目，该款应用同时使用了八个 Nios 软核，把 Avalon 总线的多 master 功能发挥的淋漓尽致。

可以说，Nios 是相当成功的。对比 Nios，它的第二代 Nios II 处理器更是青出于蓝而胜于蓝。Nios II 具备更强大的灵活性、更高的性能、更低的成本、占用更少的资源，能够满足各种嵌入式应用的性能要求。并且，Altera 提供了更易用的 Nios II 开发套件提供给客户。Altera 曾将其两代软核做了对比，结果显示 Nios II 只用了 50%的逻辑单元就实现了 2 倍于 Nios 的性能。

1.2.2 国内外开发现状

关于 Nios II 的开发现状。国外方面，在 google 上用关键词 Nios II 搜索，只有一些概要性的介绍，并无具体的开发实例报道。也曾经试图通过中国期刊网，以及国内较有影响力的水木清华等高校 BBS 的嵌入式开发版寻找相关资讯，但是少有收获。这也是正常的。Nios II 是 Altera 新近推出的一款通用 32 位软核 RISC CPU。Altera 网站上提供的所有官方开发辅助文档都是 04 年 5 月的第一版，而本人 04 年 6 月就进入迈瑞生物医疗电子有限公司正式启动这个课题。两者的间距时间非常短。可以说，所有打算使用 Nios II 的开发者都处于一个起步阶段。作为 Altera 免费提供的 IP(intellectual

property)，根据 Altera 自己提供的资料，Nios II 软核在配合 Stratix II 系列 FPGA 的情况下，可以实现超过 200DMIPS 的性能表现。有这样美味的免费大餐可以吃，相信今后会有越来越多的人会加入 Nios II 开发者的行列。

1.3 本课题主要内容

课题的主要内容有二点，挖掘应用和产品开发。

第一点是要实现 Nios II 系统的三大高级应用：高速数据传输、多 Master、多处理器。

高速数据传输，为了发挥 Nios II 系统的全部传输能力，当我们的外设是同步外设时，有必要采用高级传输。Nios II 系统采用两种高级传输模式，带 Latency 的传输和流式传输。根据具体情况加以选用。

多 Master，在 A/D 采样端或是视频输入端等数据流量非常大的位置放置 Master，独立完成指定控制功能，以减轻 CPU 的负担。

多处理器，Nios II 系统的一大特点就是当不同的 Master 访问的 Slaver 不冲突时，数据传输可以并行，这就大大提高了数据的吞吐率。若将这一特性挖掘出来加以利用，可以设计出非常强大的多 CPU 系统。

第二点是要在产品开发中建构基于 Nios II 的嵌入式系统单板。只有在实际开发中运用 Nios II，才能体现它的价值。

最后开发出了一个完整 Nios II 系统，实现对监护仪多路数据的红外通讯方式的编解码和传输控制。该系统承载于 Cyclone II 芯片内，外挂 SDRAM 和 FLASH 两种存储设备，八对红外管支持红外通讯，红外通讯协议的则在 FPGA 片内通过设计模块实现，另一侧是 USB2.0 芯片，采用飞利浦公司的 ISP1583，支持 OTG 功能。

第2章 Nios II 系统的构建

2.1 引言

低成本的 Nios II 软核配合 Altera 公司系列 FPGA 的方案已经在迈瑞生物医疗电子有限公司监护/超声两个部门的新款机型中展开运用，在低档型号中，Nios II 可以取代目前公司所使用的摩托罗拉 coldfire 系列，独当一面，以降低成本；在高端型号中，Nios II 可以辅助主 CPU，在各块硬件单板内作为处理中枢，提高整个系统的效率。

一个基本的 Nios II 系统架构除了一颗可定制的软核 CPU 之外，如同一般的嵌入式系统一样，可以在其中加入存储器设备和一系列必须的外设。

2.2 Nios II 系统总线特性

Avalon 总线是 Altera 公司为 Nios II 量身定做的内部总线。Avalon 总线的花样决定了 Nios 系统的拓扑结构，从硬件工程师的角度看，Nios II 及其外设不过是黑匣子，Avalon 总线才是 Nios II 系统的精髓，掌握了 Avalon 总线，也就掌握了 Nios II。

2.2.1 Avalon 总线模块

Altera 的 FPGA 内部并没有“总线”这种结构，没有双向信号线，也没有三态门，也就是说，PCB 上的总线不可能在 FPGA 内部实现。Avalon 总线其实是一个模块，如图 2-1。逻辑上我们仍然把它看成总线，这样我们可以按传统的总线思维模式来构建整个 Nios 系统。

从图 2-1 中可以看出，Avalon 总线上的设备有两种，Master 和 Slaver。Nios II CPU 是最常用最典型的 Master，而一般的外设都是 Slaver。所有的 Avalon 总线传输都是在一个 Master 和一个 Slaver 之间进行的，Master 产生地址和控制信号。

CPU 有指令和数据两个 Master，图 2-1 中的指令 Master 独占指令存储器，而数据 Master 与其他 Master 共享数据存储器。传统的总线在 FPGA 内部被分为读总线和写总线，CPU 的写总线直接连到每一个 Slaver，而 Slaver

的输出必须经过 MUX 才能连接到 CPU 的读总线。

DMA 控制器是由两个 Master 和一个 Slaver 组成的。Slaver 包括 DMA 寄存器，由 CPU 来配置。读 Master 从 Slaver 中读取数据，通过内部的 FIFO，由写 Master 写到另一个 Slaver。CPU 和 DMA 有可能争夺总线，Avalon 总线模块中有仲裁功能，缺省的情况下，Avalon 总线按 1:1 来分配时间片，我们也可以改变每一个 Master 占用的份额。站在软件的角度，我们感觉不到 Avalon 总线的这种调度操作。

准确地说，Avalon 总线是一个 Fabric，它借鉴了通讯行业“交换”的概念。图 2-1 中，当 CPU 通过数据 Master 访问 Data Memory 的同时，DMA 控制器可以从以太网控制器中读取数据并写到 SDRAM。也就是说，当不同的 Master 访问的 Slaver 不冲突时，数据传输可以并行，这就大大提高了数据的吞吐率[28]。

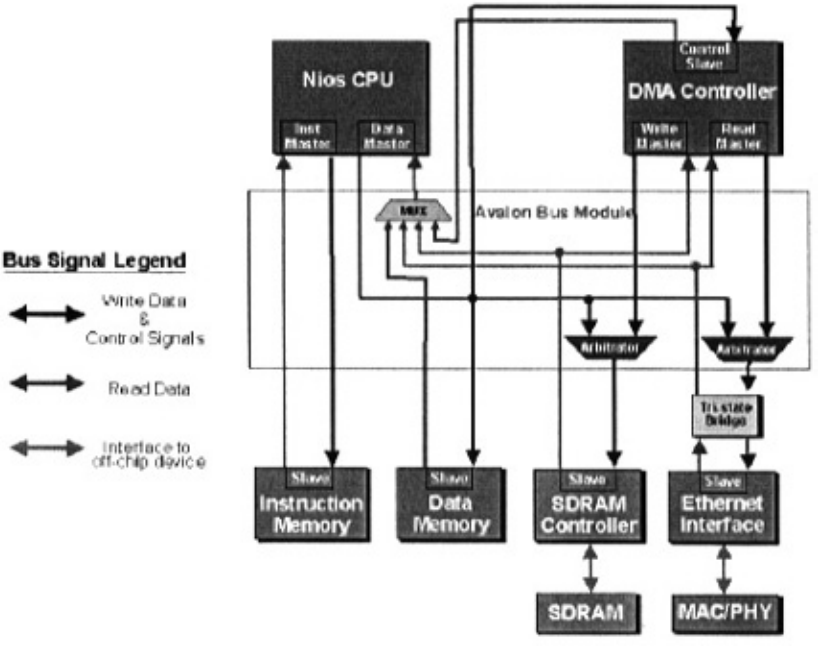


图 2-1 Nios II 基本架构

Fig.2-1 Basis constructure of Nios II

对 FPGA 工程师而言，当我们说把某某设备挂到 Avalon 总线上，实际上是将该模块与 Avalon 总线模块接口。

每一次 Avalon 传输都是在一个 Master 和一个 Slaver 之间进行的。Avalon 总线模块在 Master 和 Slaver 之间起到桥梁的作用。Avalon 总线具有自动适配 Master 和 Slaver 时序的功能。当快速 Master(如图 2-1 中的 CPU)访问慢速 Slaver(如图 2-1 中的以太网控制器)时，Avalon 总线会自动加入等待周期。Avalon 总线具有自动适配 Master 和 Slaver 数据宽度的功能。当 32 位 Master 访问 16 位 Slaver 时，Avalon 总线会在 Slaver 一侧将访问周期拆成 2 个，通过两次对 Slaver 的访问来完成 Master 的一个访问周期。Avalon 总线的这种特性使 FPGA 工程师不必关心 Master 经过 Avalon 总线到 Slaver(或相反)的传输过程，只需要设计 Master 或 Slaver 与 Avalon 总线的接口时序。

在基本的 Avalon Slaver 传输中，Avalon Slaver 有两种，寄存器型和存储器型。寄存器型 Slaver 数据宽度可以是 1~32，按 32 位编址，没有字节允许信号。存储器型 Slaver 数据宽度只能是 8,16 和 32 位，按字节编址，有字节允许信号。

对 Avalon Slaver 而言，接口信号和时间参数决定了它与 Avalon 总线的接口时序。基本的 Avalon Slaver 的接口有如下信号：Address：地址线的宽度决定 Avalon Slaver 占用地址空间的大小。Readdata：读数据。Read：读允许，对片内外设，读允许信号是可选的。Writedata：写数据。Write：写允许。Int：中断，可选。Chipselect：片选，对片内外设，片选信号是可选的。当有片选信号时，读写允许不参与地址译码，否则，读写允许参与地址译码。Waitrequest：等待，用于可变等待周期的外设。Byteenable：字节允许，用于存储器型外设。除了数据和地址信号外，控制信号都可以选择高电平有效或是低电平有效。

对于不同的外设，接口可以不同。比如对于只写外设，就没有读允许和读数据。根据外设的功能省掉不需要的接口，可以减少对逻辑资源的需求，同时还可以提高整个系统的性能。

基本的 Avalon 时间参数有 setup_time，wait_state 和 hold_time。这三个参数读操作和写操作时可以不同。时间参数可以用时间表示，也可以用时钟周期数表示。当用时间表示时，自动按保守的原则转换成时钟周期数。

图 2-2 是最简单的 Avalon 读传输。需要说明的是 Avalon 总线是同步总线，所以的操作都是在时钟的上升沿进行的。在 A 时刻，时钟的上升沿产生地址，经延时在 B 处出现，片选经地址译码在 C 处出现，外设得到地址和片选，在 D 处输出数据，Avalon 总线在下一个时钟的上升沿 E 采样数据。这是一个零等待周期的 Avalon Slaver 读传输的时序，setup_time = 0，

wait_state = 0，hold_time = 0。

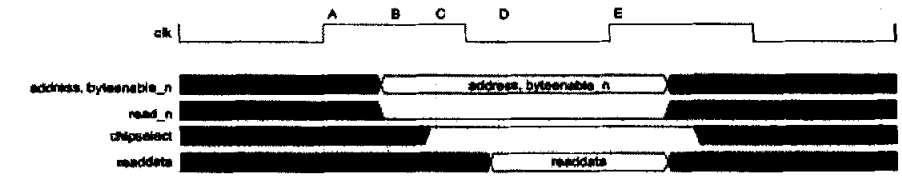


图 2-2 Avalon 读传输

Fig.2-2 Avalon read transfers

图 2-3 是有一个等待周期的 Avalon 读传输。我们可以看到，数据在第二个时钟的上升沿产生，Avalon 总线在第三个上升沿采样数据，地址，读和片选维持两个周期。参数设置为 setup_time = 0，wait_state = 1，hold_time = 0。

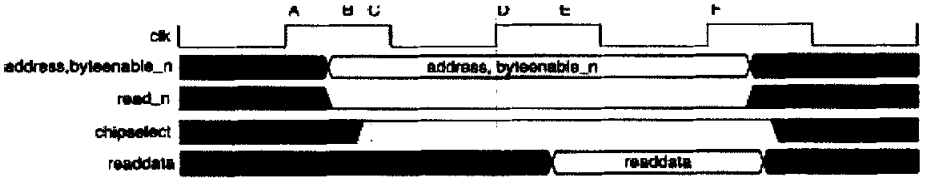


图 2-3 有一个等待周期的 Avalon 读传输

Fig.2-3 Nios II read transfers which has a circle of waiting

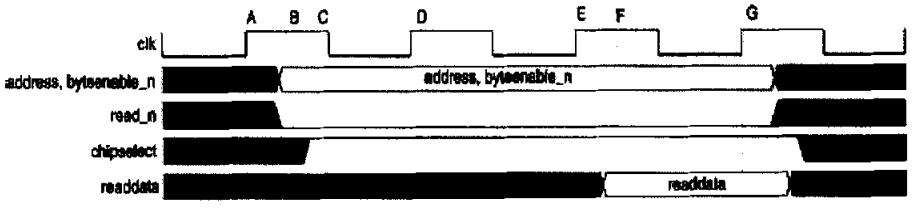


图 2-4 有两个等待周期的 Avalon 读传输

Fig.2-4 Nios II read transfers which has two circles of waiting

当然，wait_state 可以根据需要进行设置。图 2-4 就是具有两个等待周期的 Avalon 读传输。

图 2-5 是具有可变等待周期的 Avalon 读传输。接口信号比前几种形式多了一个 waitrequest 信号，Avalon 总线在采样到 waitrequest 信号无效时采样数据，地址，读和片选维持到 waitrequest 无效。在这种情况下，setup_time = 0，wait_state 没有意义，hold_time = 0

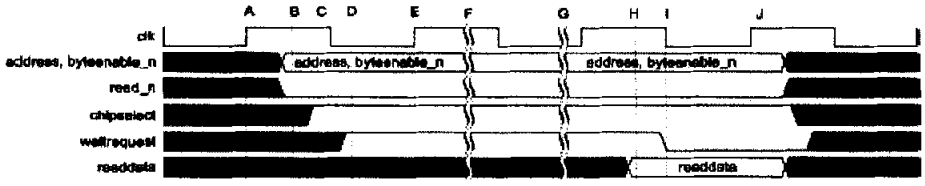


图 2-5 可变等待周期的 Avalon 读传输

Fig.2-5 Nios II read transfers which can change circles of waiting

图 2-6 是带有一个地址建立周期的 Avalon 读传输。Setup_time = 1，wait_state = 1，hold_time = 0。地址建立一周期后，读信号才给出。

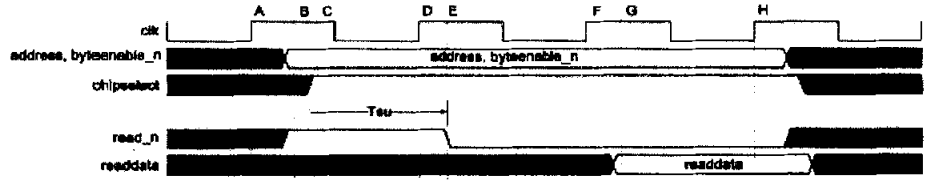


图 2-6 有一个地址建立周期的 Avalon 读传输

Fig.2-6 Nios II read transfers which has one circles of building

图 2-7 是最简单的 Avalon 写传输。数据，地址，写和片选在同一个时钟沿给出，外设在下一个时钟沿采样数据。Setup_time = 0，wait_state = 0，hold_time = 0。

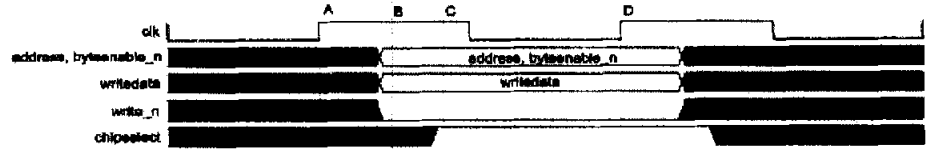


图 2-7 基本的 Avalon 写传输

Fig.2-7 the basis of Nios II write transfers

图 2-8 是带有一个等待周期的 Avalon 写传输。Setup_time = 0，wait_state = 1，hold_time = 0。数据，地址，写和片选都延长了一周期。N 个等待周期的情况，只需要改变 wait_state 参数。

图 2-9 是带有可变等待周期的 Avalon 写传输，接口信号多了 waitrequest 信号，数据，地址，写和片选维持到 waitrequest 无效。Setup_time = 0，wait_state 无意义，hold_time = 0。

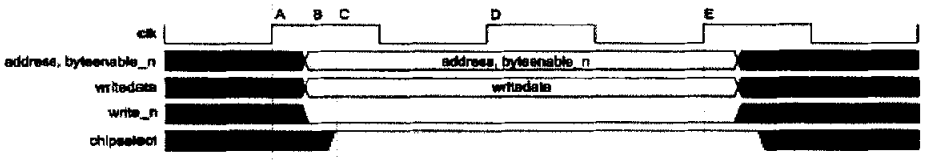


图 2-8 有一个等待周期的 Avalon 写传输

Fig.2-8 Nios II write transfers which has one circles of waiting

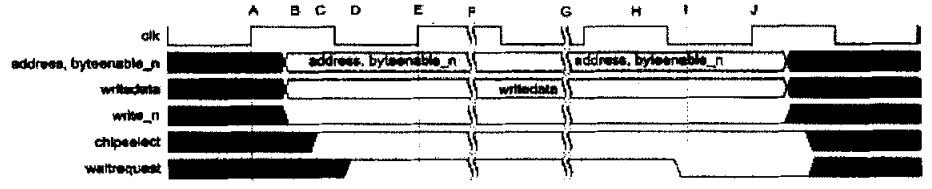


图 2-9 可变等待周期的 Avalon 写传输

Fig.2-9 Nios II ,write transfers which can change circles of waiting

图 2-10 是带有建立时间和等待时间的写传输。典型的兼容异部时序的做法，为了保证异步时序的可靠性，在写信号的前后留有一周期的建立/保持时间。Setup_time = 1，wait_state = 0，hold_time = 1。

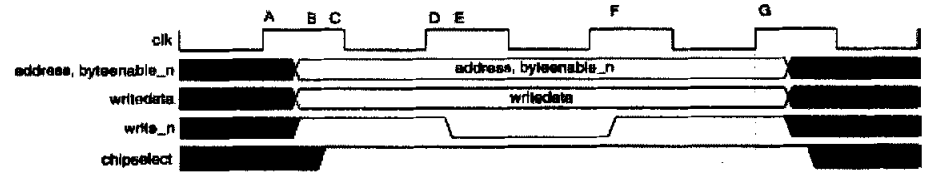


图 2-10 带建立、等待周期的 Avalon 写传输

Fig.2-10 Nios II write transfers which has one circles of setup and waiting

三个时间参数的配合，可以适配几乎所有异步时序的外设。图 2-10(零等待写)的时序其实就是最基本的同步 Slaver 写时序。

基本 Avalon Slaver 是按异步时序设计的，往往不能发挥 Avalon 总线的全部传输能力。当我们的外设是同步外设时，有必要采用高级 Avalon Slaver 传输。

上一节提到，零等待写的时序是最基本的同步 Slaver 写时序。同步 Slaver 读时序是不能通过基本 Avalon Slaver 传输的，只有通过高级 Avalon Slaver

传输。

下面介绍一下带 Latency 的读传输。Latency 和 wait 是两个不同的概念。前面的描述中可以看到，wait 的引入将使地址和控制信号延长至读数据正确采样。而 Latency 是同步逻辑特有的概念，熟悉同步存储器的选手都知道，同步存储器的数据滞后地址 1~2 周期(1 周期称为 flow-through 时序，2 周期称为 pipeline 时序)输出，这里的 1~2 周期就是 latency。以 pipeline 为例，地址和控制信号不是维持 2 周期，而是每个周期都可以改变，数据线连续输出 2 周期前的地址对应的数据。

图 2-11 是一个带有 2 个 latency 和可变等待周期的读传输。在 C 处，Avalon 总线采样到 waitrequest 无效后，两个周期后得到数据。Address2 和 address3 的数据在两个周期后连续得到。这是一个比较复杂的例子，既有 latency，又有 wait。实际设计中更常用的是只有 latency 的外设，比如同步 SRAM，FIFO。本例中需要设置一个新的参数 latency = 2。

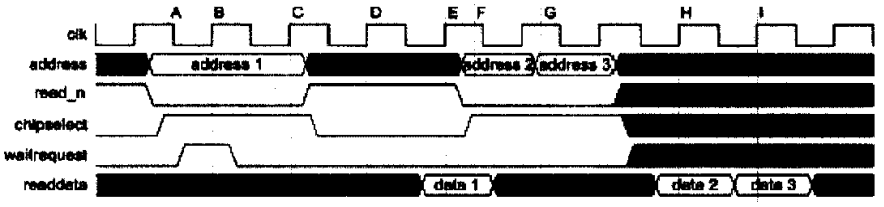


图 2-11 有可变等待周期和两个 Latency 的 Avalon 读传输

Fig.2-11 Nios II read transfers which can change circles of waiting and two Latency

带 Latency 的读传输在同步读操作中的应用极广，它可以完全发挥出 Avalon 总线的性能。

流式传输也是重要的高级传输方式。流式传输是一种类似 PCI 突发访问的传输方式。这种方式更适合批量数据的传送。和带 Latency 的读传输一样，流式传输也可以发挥 Avalon 总线的全部性能。图 2-12 是流式读传输。接口多了两个信号，dataavailable 和 endofpacket。这两个信号是流式读传输的标志。Avalon 总线只给出起始地址，在传输的过程中，地址不变化。Slaver 每个周期输出一个数据，同时输出 dataavailable 信号表示数据有效。在输出最后一个数据时，Slaver 给出 endofpacket 信号，Avalon 总线在收到 endofpacket 信号后，结束传输。和 PCI 总线类似，Slaver 可以通过使 dataavailable 无效来插入等待周期。

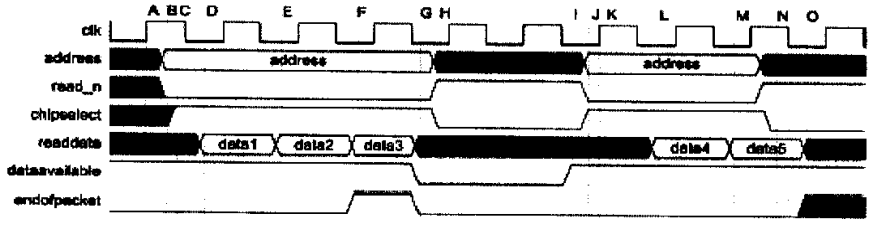


图 2-12 流式读传输

Fig.2-12 Nios II read transfers of flowing

图 2-13 是流式写传输。接口信号多了 readyfordata 和 endofpacket。这两个信号是流式写传输的标志。和流式读传输一样，Avalon 总线只给出起始地址，在传输的过程中，地址不变化。Avalon 总线连续输出数据，当 Avalon 总线检测到 endofpacket 有效时，表示一旦 readyfordata 无效，传输就结束。和 PCI 总线类似，Slaver 可以通过使 readyfordata 无效来插入等待周期。

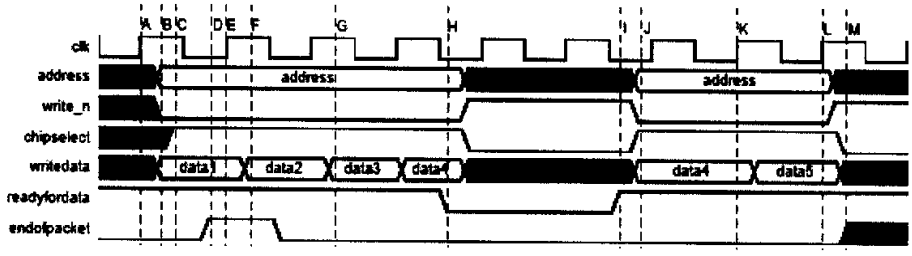


图 2-13 流式写传输

Fig.2-13 Nios II write transfers of flowing

常见的 Avalon Master 包括 CPU 和 DMA 控制器。使用 Avalon Master 外设可以完成批量数据的不经过 CPU 的传输，从而减轻 CPU 的负担，提升系统的处理能力。举个例子，小尺寸的 LCD 控制器就可以不用 VRAM，直接在 SDRAM 中开辟一块区域作为 VRAM，用 Avalon Master 不经过 CPU 直接从 SDRAM 中读取颜色数据输出到 LCD。

前面提到，Avalon 总线自动适配 Master 和 Slaver 的时序，我们在设计 Master 时，就可以不关心 Slaver 的时序。也就是说，我们用 Avalon Master 读 SDRAM 时，可以完全无视 SDRAM 控制器的存在，只把 SDRAM 当成逻辑上的存储器。

使用 Avalon Master，还可以充分利用 Avalon 总线的 Fabric，也就是“交

换"的特性，使整个 Nios 系统的操作并行化，提高系统的效率。

图 2-14 给出的是 Master 基本读时序。Master 没有时间参数。Waitrequest 信号是必须有的，Master 的地址和控制线维持到 Slaver 通过 Avalon 总线送来的 waitrequest 信号无效，并在 waitrequest 信号无效时采样数据。

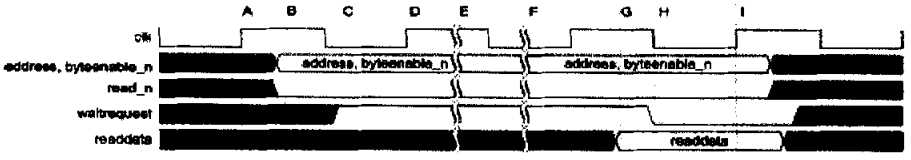


图 2-14 主设备基本读传输

Fig.2-14 Master read transfers

图 2-15 给出的是 Master 基本写时序。同样 waitrequest 信号是必须有的，Master 的地址，数据和控制线都将维持到 Slaver 通过 Avalon 总线送来的 waitrequest 信号无效。

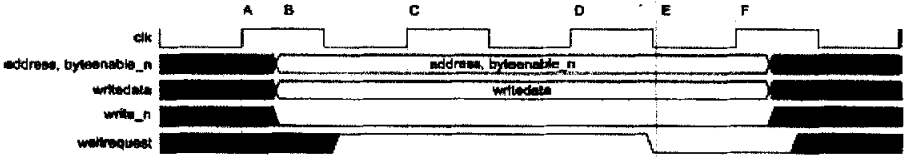


图 2-15 主设备基本写传输

Fig.2-15 Master write transfers

与 Slaver 对应，Master 也有带 Latency 的读传输和流式传输。下面介绍一下带 Latency 的读传输，如图 2-16 所示。Master 没有 latency 参数。接口多了两个信号 readdatavalid 和 flush。Avalon 总线通过 waitrequest 信号来使 Master 插入等待周期，Master 检测到 waitrequest 信号有效时，必须维持地址和读信号不变。

Avalon 总线通过 readdatavalid 信号来通知 Master 数据已经可用，latency 是可变的，Master 必须按 readdatavalid 信号来决定数据对应的地址。Master 可以通过 flush 信号来中止传输，不再继续等待未收到的数据而开始下一个访问周期，如图 2-16 的 J，Master 不再等待未收到的 data3，发 address4 开始下一个访问周期。

再介绍一下流式主传输。图 2-17 所示是流式 Master 传输。A-H 是写传输，I-J 是读传输。

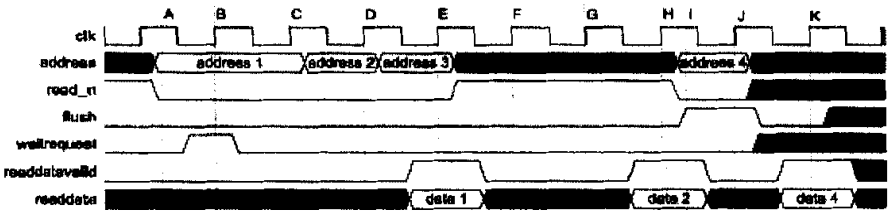


图 2-16 Latency 读传输

Fig.2-16 Read transfers of latency

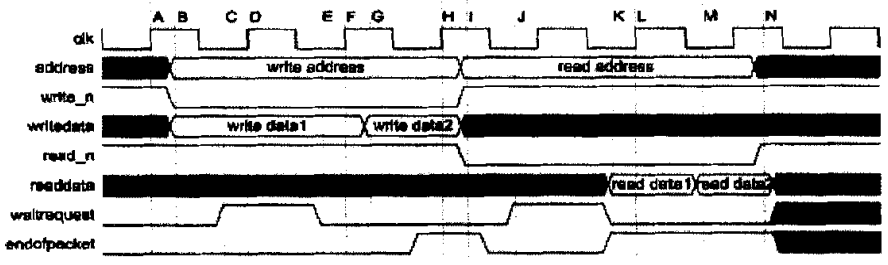


图 2-17 流式主传输

Fig.2-17 Master transfers of flowing

Endofpacket 是流式 Master 输出的标志。在流式传输过程中，Master 可以只输出起始地址。Avalon 总线可以通过使 waitrequest 无效来使 Master 插入等待周期，如 C-E，J-K。写传输过程中，waitrequest 无效时，Master 每个周期输出一个数据，直到 endofpacket 有效。读传输过程中，waitrequest 无效时，Master 每个周期采样一个数据，直到 endofpacket 有效。图 2-17 中，为什么 L 处检测到 endofpacket 有效还要继续读数据呢？Avalon 规范解释道：根据应用 Master 拥有 endofpacket 信号的解释权，在图 2-17 中，Master 将 endofpacket 解释为可以再读一个周期，这是一种特殊的应用。一般情况下，我们还是把 endofpacket 解释为指示访问结束。

2.2.2 Avalon 三态桥和片外设备

通过 Avalon Tri Bridge，Avalon 总线可以访问片外设备。片外设备可以看作逻辑上的 Avalon Slaver，时序与 Avalon Slaver 相同。

片外设备的数据线是双向的，这是 Nios II 外设可以作为片外设备并挂

在 Tri Bridge 上的必要条件。片外设备的接口与基本 Slaver 是一样的，但片选必须有，除了片选外，其他信号都可以共享。我们可以只向片外引一组数据，地址，读，写信号，通过多个片选信号来控制多个片外设备。这与传统的 CPU 是一样的。

2.3 Nios II 系统基本应用

硬件配置如图 2-18 所示。

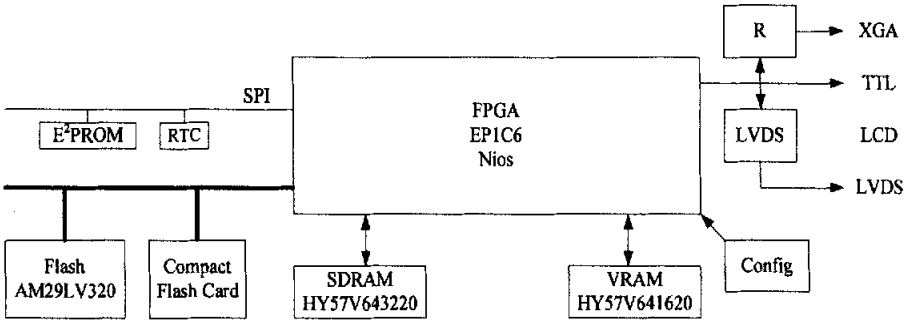


图 2-18 硬件架构

Fig.2-18 Hardware construction

使用的是基于 Nios II 的开发版，硬件配置如图 2-18 所示。本设计的主要目的是验证 Nios II 系统的正常工作能力，所以未使用显示控制部分的板上资源。

Nios II 嵌入 FPGA 中，FPGA 的型号为 EP1C6Q240C8。本板的 Nios II 配置包括下列内容：Nios II CPU，本板的 Nios II 工作在 75MHz，处理能力约 75MIPS。Nios 的时钟由 25MHz 外部晶振经 FPGA 内部锁相环三倍频提供。SDRAM 控制器，支持 8MB 的 SDRAM。Tri-Bridge，三态桥提供内部总线与外部三态总线的适配，外部三态总线上接 Flash。

CPU 使用 75MHz 的时钟。外接晶振为 25MHz，通过 PLL0 变换到 75MHz。SDRAM 使用与 CPU 相同的时钟，由 PLL0 输出。复位电路必须保证 PLL 输出时钟正常前系统一直处于复位状态。计数器使用晶振直接产生的 25MHz 时钟，复位信号保持时间大于 20 微秒。足够保证 PLL 能在这段时间内输出时钟正常。

FPGA 内嵌 Nios II 系统总体框图见图 2-19。

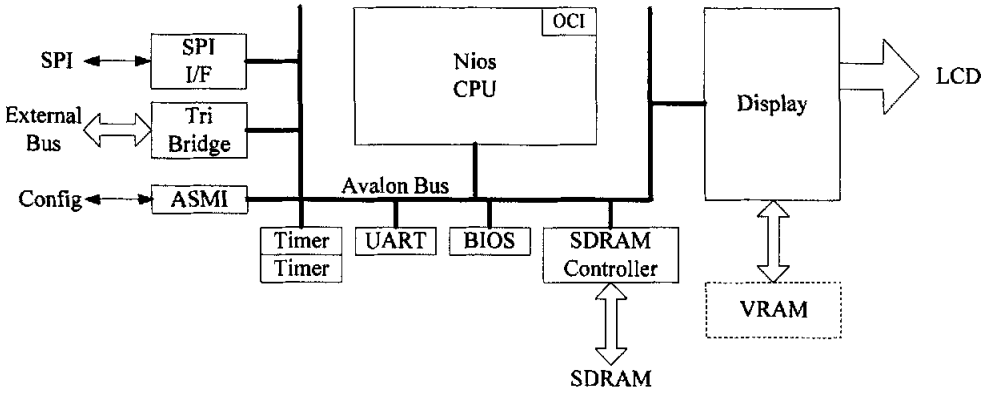


图 2-19 FPGA 架构

Fig.2-19 FPGA construction

本 Nios II 系统完成如下功能。Nios II 作为中央处理器完成 CPU 功能，SDRAM 为 CPU 提供片外存储器空间。Timer 则完成所需的定时器功能。片内一共设置了两个定时器。三态总线桥将 Nios II 的 Avalon 总线转换成三态总线，从而把外部 Flash 接入系统，Flash 中存储程序。ASMI 接口上挂接了一块配置 Flash，其中存储的是配置 FPGA 片内逻辑所用的 HDL 源码。LCD 显示控制逻辑控制显示单元完成显示驱动。

2.3.1 Nios II CPU 配置

Nios II CPU 为 32bit CPU，256 个 32 位寄存器，用 LE 做指令译码，支持中断。根据软件的需要和资源状况，可以考虑用硬件实现乘法器。由于内部存储器资源紧张，本系统不使用指令和数据 Cache。BIOS 为 2kByte 的 RAM。FPGA 配置完成后，BIOS 的内容为监控程序。BIOS 的地址为 0x01601000~0x016017FF。监控程序只占 1.5kByte 空间。在 BIOS 的高端 0x01601700~0x016017FF 放中断向量。

Flash 容量 4MByte，16 位数据接口。地址为 0x01800000~0x01BFFFFF，保留地址 0x01C00000~0x01FFFFF，以便今后扩展到 8MByte。SDRAM 为 8MByte，32 位接口。配合 HY57V643220，我们设置 CL=2，trfc=70ns,trp=20ns,trcd=20ns,tac=6ns。SDRAM 地址 0x000000~0x7FFFFF。保留 0x00800000~0x00FFFFFF，以便以后扩展到 16MByte。SPI Master 工作在 1MHz，数据宽度为 8 位，MSB 在前，LSB 在后，产生两个片选，cs0 接

E^2PROM，cs1 接 RTC。Clock Polarity 和 Clock Phase 都为 1。SPI 的地址为 0x01601900~0x0160191F。中断号为 18。

本系统使用了两个 Timer，都是只产生中断的简单定时器，Timer0 定时周期为 2ms，Timer1 定时周期为 4ms。Timer0 的地址为 0x001601920~0x00160193F，中断号为 16。Timer1 的地址为 0x001601940~0x00160195F，中断号为 17。

UART0 为调试用串口，波特率固定为 115200，误差<0.01%，8 位数据位，1 位停止位，无校验。UART0 的地址为 0x01601960~0x0160197F，中断号为 19。

Active Serial Memory Interface 可以在 FPGA 配置完成后，将配置芯片的剩余空间当作通用 Memory，或用于在线更新 FPGA 配置文件。ASMI 的地址为 0x01601980~0x0160199F，中断号为 20。

Tri Bridge 提供 Avalon 总线与外部三态总线的适配。Tri Bridge 对软件是透明的，不占用地址空间。

P04A 芯片组的 Avalon 总线分三部分。指令总线上有 CPU，BIOS，SDRAM 和 Tri Bridge，也就是说，程序只能在 BIOS，SDRAM 和 Tri Bridge(上的 Flash)中运行。数据总线上有 CPU，BIOS，SDRAM，SPI，Timer，UART，ASMI，显示逻辑等设备，CPU 可以通过指令访问这些设备。Tri 总线上有 Flash，CPU 通过 Tri Bridge 来访问 Flash。

2.4 Nios II 高级应用

低成本的 Nios II 软核配合 FPGA 的方案不仅在价格上有吸引力，而且厂商提供了许多便利的开发工具和已经集成好了的 IP Core。这就能大大加快开发进度，提前了新产品面世的时间。对于企业，开发基于 Nios II 架构的系统具有重要的现实意义。而 Nios II 的高级应用在这个基础上更进一步，以非常巧妙地设计和简洁的逻辑描述，实现系统的高效运转。

2.4.1 高速数据传输

为了发挥 Nios II 系统的全部传输能力，当外设是同步外设时，有必要采用高级传输。Nios II 系统可以使用带有 Latency 的读传输。采用如下方案，在 RAM 中运行程序，搬移 SDRAM 中的数据。整个系统的结构非常简洁，与基本配置的差异只是 Nios II 系统内多放了一个 RAM，而 FPGA 外围接口

类同。系统硬件配置如下。

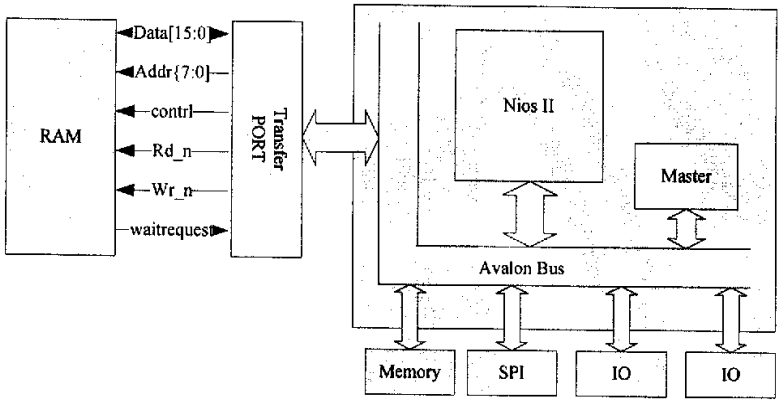


图 2-20 高速数据传输架构

Fig.2-20 High data speed transfers construction

介绍一下带延迟的读传输。同步存储器的数据可以滞后地址 1~2 周期输出，这里的 1~2 周期就是 latency。以延迟 2 周期的读传输为例，地址和控制信号不是维持 2 周期，而是每个周期都可以改变，数据线连续输出 2 周期前的地址对应的数据。这就大大加快了数据读取速度。

介绍一下接口设计。SOPC Builder 生成 Nios II 系统时生成 ptf 文件，存放 Nios II 系统的硬件相关属性。其中的“Read_Latency”项的取值定义读延迟属性，一般的取值有三种 0、1、2。对于我们的 SDRAM，其读延迟设为 2，再依靠与 avalon 总线与外设的接口信号读使能、地址线、waitrequest 信号配合，完成带延迟的读传输。

在接口设计上，需要考虑与带延迟读传输所匹配的信号类型。Latency 和 wait 是两个不同的概念。前面的描述中可以看到，wait 的引入将使地址和控制信号延长至读数据正确采样。而 Latency 是同步逻辑特有的概念，熟悉同步存储器的选手都知道，同步存储器的数据滞后地址 1~2 周期(1 周期称为 flow-through 时序，2 周期称为 pipeline 时序)输出，这里的 1~2 周期就是 latency。以 pipeline 为例，地址和控制信号不是维持 2 周期，而是每个周期都可以改变，数据线连续输出 2 周期前的地址对应的数据。

图 2-21 是一个带有 2 个 latency 和可变等待周期的读传输。在 C 处，Avalon 总线采样到 waitrequest 无效后，两个周期后得到数据。Address2 和 address3 的数据在两个周期后连续得到。

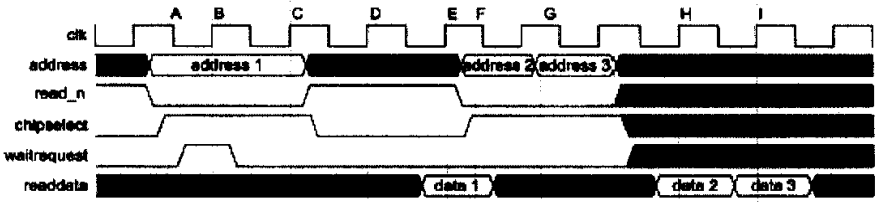


图 2-21 带延迟的读传输时序

Fig.2-21 Read transfers with latency

带 Latency 的读传输在同步读操作中的应用极广，它可以完全发挥出 Avalon 总线的性能。

2.4.2 多 CPU 设计

Nios II 系统的一大特点就是当不同的 Master 访问的 Slaver 不冲突时，数据传输可以并行，这就大大提高了数据的吞吐率。若将这一特性挖掘出来加以利用，可以设计出非常强大的多 CPU 系统。

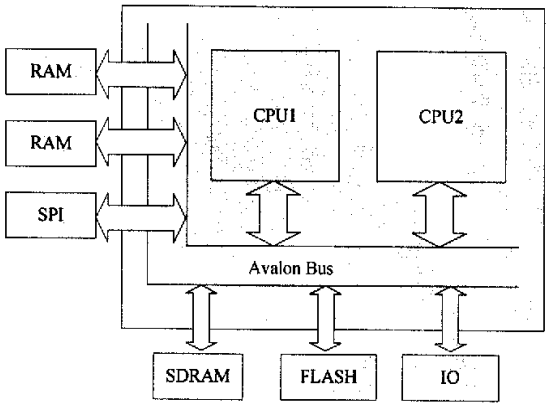


图 2-22 多 CPU 系统

Fig.2-22 Multi-CPU system

配置结构如上图所示。除了片外已有的存储器外设，对两个 CPU 各分配一个 4Kbits 大小的 RAM。用这两个 RAM 来验证 CPU 独立工作的能力。在这个 Nios II 系统内，不分主副 CPU，方便测试。在系统设置完成后，在 Nios II IDE 里可以完成软件测试。

2.4.3 多 Master 设计

Nios II 系统的一大特点就是当不同的 Master 访问的 Slaver 不冲突时，数据传输可以并行，这就大大提高了数据的吞吐率。若将这一特性挖掘出来加以利用，可以设计出非常强大的多 Master 系统。

设计了两个 Master 的处理结构。系统配置如图 2-23。CPU 自身是一个 Master 设备，红外读写逻辑是另一个 Master 设备，总线上挂着 Flash、Sdram 和 USB2.0 芯片。本系统通过红外管以及 USB 端口和其他硬件单板通讯。

Master 设备不需要缓存，也不需要 CPU 中断的介入。只要有合适的总线与之匹配，它可以操作从设备。本设计中的主要从设备是 Sdram，红外管把收到的红外线信号转换成电信号，发送给 CPLD，CPLD 内部解码，再存入 Sdram。当数据量积累到一定程度的时候，再通知 CPU 取走 Sdram 内的数据。如果给单一红外管以及其匹配 CPLD 分配的 Sdram 存储空间越大，CPU 的负荷也就越小，但是相应的，数据上传的时间间隔也会增大。这是一对矛盾，需要权衡处理。

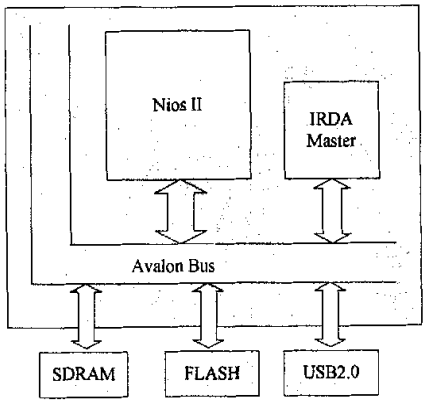


图 2-23 多 Master 系统

Fig.2-23 Multi-Master system

2.5 本章小结

本章对 Nios II 系统做了非常完整的描述，从 CPU 到各种形式的外设，组合成一个完整的基于 Nios II 的片上系统。Avalon 总线是 Nios II 系统的精髓所在，对此进行了详尽的阐述。掌握了 Avalon 总线，对于基本的 Nios II

架构也就掌握了大半。最后，从基本配置的 Nios II 系统开始设计，到几种复杂的 Nios II 应用，都一一做了设计说明。Nios II 的高级应用开发是本章仅次于 Avalon 总线机制的第二个重点。高级应用开发是建立在基本应用的基础上的，高速数据传输优于传统的异步传输，效率提升了 2 倍以上。而多 CPU 概念则把系统的集成度大大提高了，多个 CPU 协同工作的系统，显然在性能上有了巨大的飞跃。第三种高级应用是多 Master，Master 概念在一定程度上是 CPU 概念的延伸。挂在 Avalon 总线上的 Master 其可以绕开 CPU 独立操作 Slave 设备的特性，有效地减轻了 CPU 的负担。

第3章 Nios II在红外通讯中的应用

3.1 引言

对 Nios II 系统特性的深入掌握，是为了开发出更复杂更高校的片上系统。本章阐述的就是这一重要的成果。Nios II 子系统扩展 8 个串口和 1 个中速红外线接口，通过串行红外线(SIR)编解码逻辑的转换，实现本板与另一块单板的通信，进而完成对参数模块的操控以及数据传输。在另一头，Nios II 控制 USB2.0 芯片，把下位机的参数数据上传，并下行控制命令。

图 3-1 给出了整个系统的框图。本单板属于监护仪前端生物小信号放大电路与主控板之间衔接的单板。由于人体生物信号多样，所以对应的采集模块种类也很多。通过该块单板可以把前段的多路参数信号整合成一股数据流，通过 USB2.0 芯片 1583 上传给主控板。

FPGA 片内外挂 8 个低速红外模块，1 个中速红外模块。在遵守红外通讯规范的前提下，自定通讯协议，通过红外对管实现与下位机的通讯。红外模块作为 FPGA 片内外设，由硬件描述语言实现，然后通过自己设计的接口信号挂在 Avalon 总线上，总线侧信号类同异步串口。另一侧则根据约定的波特率收发红外数据。模块内部进行编解码，并提供缓存机制。

各个参数模块的数据上传速率都不一样，对应的红外模块内部也设计了独特的缓存适配机制。以保证不同速率的模块数据上传都能得到足够的缓存。同时，支持 ID 识别模式，即监护仪上任何一个槽位都对于不同的模块都具备识别能力，通过模块 ID 应答机制来实现。模块上电之后，即开始不断的发送 ID，上位机在接收到 ID 信息之后，对其进行解读，同时给模块一个应答信号，告诉模块连接已经完成。至此，模块停止发送 ID，模块和上位机同时切换到约定的波特率开始通讯。系统的连接至此完成。

该系统同时配置了 SDRAM、Flash 两类外设存储器。SDRAM 通过片内的 SDRAM 控制器挂在 Avalon 总线上。Flash 则通过外部三态桥接入系统。Avalon 总线对外部三态总线上设备的识别有其特殊性，比如 Flash，根据我所选用的 Intel 的 8Mbit 容量。在硬件电路连接上，就必须把其最低位地址线接高一位，也就是忽略内部的第零位地址。SPI 口线上挂一个 EEPROM，可供存储一些小批量的重要数据。

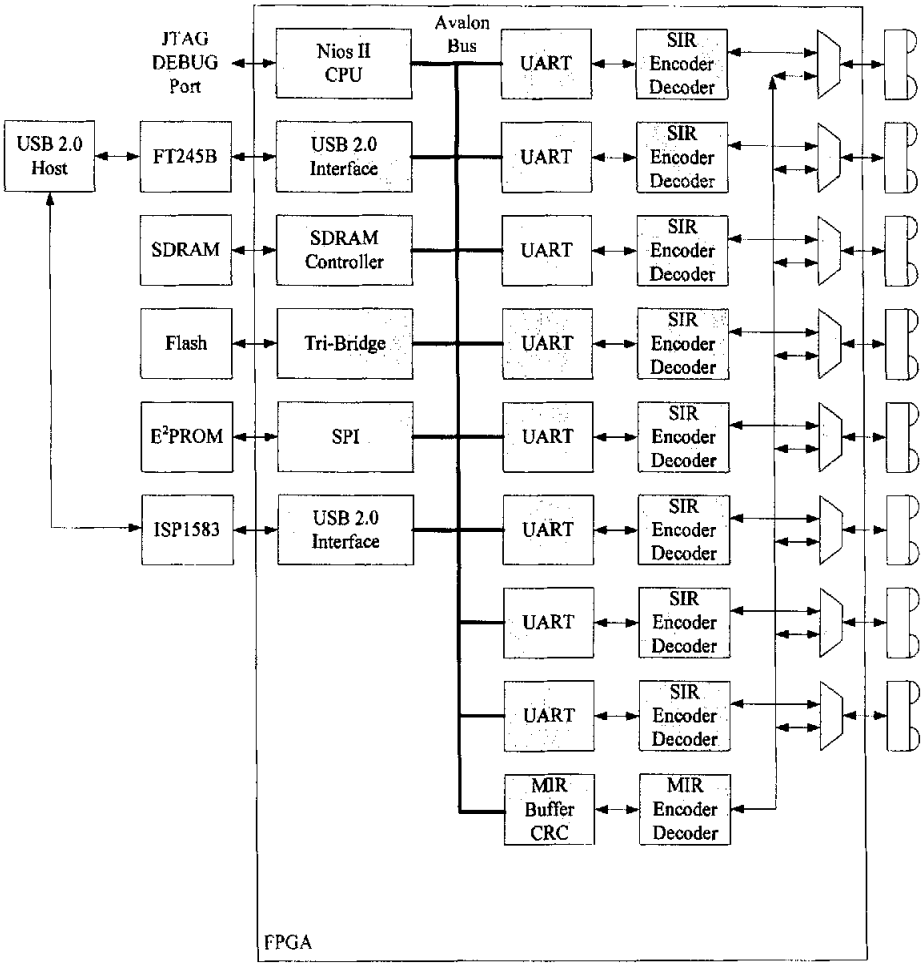


图 3-1 单板系统

Fig.3-1 Hardware system

3.2 本板内 Nios II 系统的配置设计

Nios II 子系统就相当于一个内含 CPU 和若干外设、存储器的单片解决方案。软核 CPU，片上外设，片上存储器，以及片外存储器的接口都集成在一片 FPGA 之中。本 Nios II 子系统由 Nios II CPU、USB2.0 Device、Flash、SDRAM、外接 E2PROM 的 SPI 以及 Altera 专用的主动串行配置芯片(ASMI)

组成。其中 USB2.0 实现接口板与主机的通信，Nios II CPU 则完成 USB 协议到模块协议的转换。

在 Nios II 子系统中，除 USB2.0 之外的所有的外 0 设都用 SOPC Builder 产生。SOPC Builder 产生外设的硬件代码的同时，还产生外设的 Memory Map 和驱动程序，换句话说，Nios 外设的驱动程序是经过验证的，编写软件时所看到的只是一系列函数，这就大大简化了软件开发，所以我们说 Nios II 平台是嵌入式系统软硬件一体化的解决方案。下图 3-2 对本系统内的配置做了图示说明。

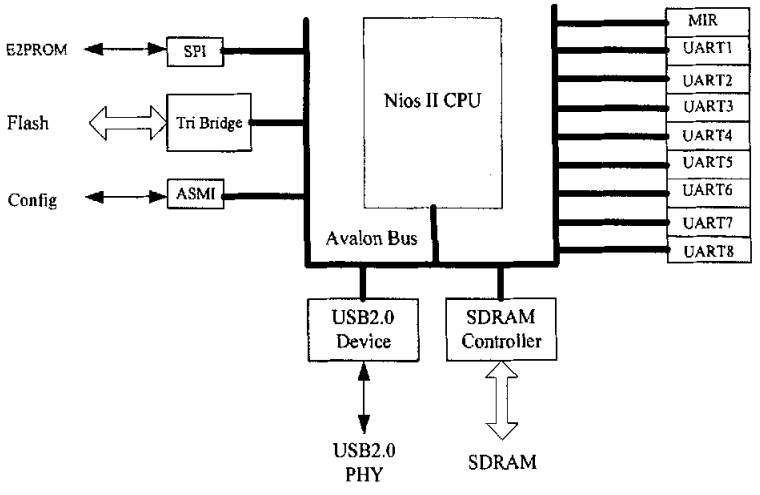


图 3-2 Nios II 系统

Fig.3-2 Nios II system

从 CPU 开始说起，Nios II CPU 的指令集、数据宽度和地址宽度皆为 32 位，是真正的 32 位软核 CPU。由于 Nios II 由 VHDL 代码实现，所以使用 Nios II 本身不需任何成本(当然，占用的 FPGA 逻辑资源还是需要花钱买的)，而且能够完成相当复杂的控制功能，可以说是一个性价比极优的解决方案。Nios II CPU 含 32 个通用寄存器，32 个外部中断源，这是最基本的配置。

Nios II 采用 Tri Bridge 将内部 Avalon 总线转换成传统的双向总线。Flash 通过三态桥连接到 Avalon 总线上，从而挂接到 Nios II 系统内。

Flash 选用 Intel 的 28F800C3，16 位，容量为 8Mbit，电路上实现对同系列芯片中 16/32/64Mbit 型号的向上兼容。如图 3 封装所示，把管脚 9、10、15 置为 NC 即本板方案，如想改用 16Mbit 的同系列 Flash，只需把管脚 15

置为地址输入即可。32/64Mbit 的兼容方案类推。

该款芯片内含 128 个 32K 字长的存储块，其中第一个(也可以是最后一个，根据 Flash 型号而定)块被划分成 8 个 4K 字长的 Boot 块。有三种封装可选，48 脚 TSOP 封装、64 脚 BGA 封装和 48 脚 TSOP 封装。

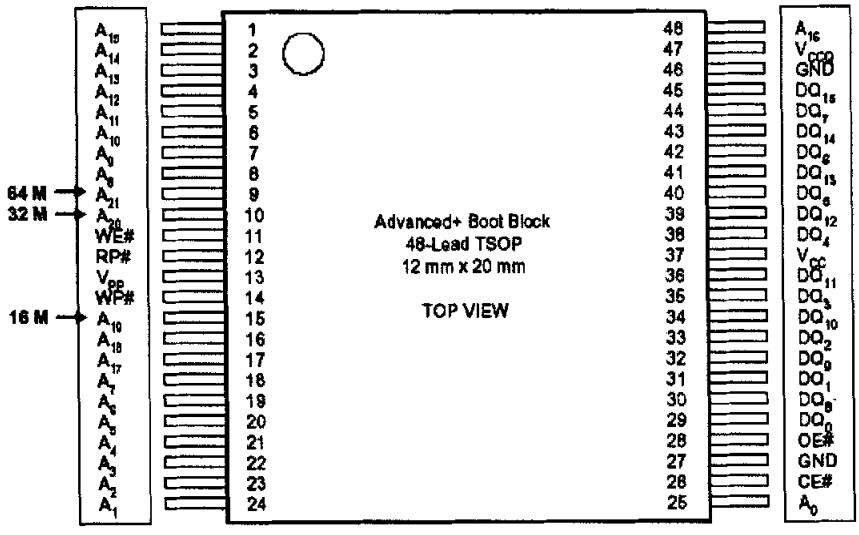


图 3-3 28F800C3

Fig.3-3 28F800C3

下表 3-1 给出了 Flash 芯片硬件连线的设计以及在各种工作状态下，各控制管脚的信号状况。

芯片总线分为五种工作状态，重启，禁用，待命，读，写。由四个控制信号 CE#、WE#、RP# 、OE#完成管理。为了防止错误的写/擦除操作，建议对 VCC 和 VCCQ 同时上电，VCC 和 VCCQ 必须同时掉电。建议在 VCC 到达 VCC 最小值之后，在对 VPP 上电。而且，VPP 必须在 VCC 掉电之前掉电。如果 VCCQ 和 VPP 未连在 VCC 上，那么在给 VCCQ 和 VPP 供电之前，VCC 必须已经达到 VCC 最小值。在 VCC 达到 VCC 最小值前，芯片输入不应该被赋值。当 VPP 电压在 1.65V～3.6V 之间的时候，所有编程/擦除电流通过 VCC 引入。如果驱动 VPP，则电压必须高于 1.65V。也就是说，VPP 必须保持在 1.65V 之上，以执行 flash 修改。如果 VPP 被连到 12V 电压源，则系统直接从 VPP 引入编程/擦除电流。

表 3-1 Flash 管脚功能及设计连接

Table 3-1 Flash datasheet and link design

信号	类型	名称/功能
A[18:0]	输入	存储器地址输入。
DQ[15:0]	输入/输出	数据输入/输出。写周期期间输入数据和指令，读周期期间输出数据。当 CE#和 WE#使能期间，可对命令用户接口输入管理命令
CE#	输入	片选使能，低电平有效
OE#	输入	输出使能，低电平有效
RP#	输入	重启/掉电，低电平有效。当RP#为低电平时，芯片处于重启/深度掉电模式。此时输出管脚为高阻态。当RP#为高电平时，设备处于正常工作模式。当 RP# 从低变为高时，芯片重启所有存储模块，默认处于批处理读模式
WE#	输入	写使能，低电平有效。地址和数据在WE#脉 冲 的上升沿被锁存
WP#	输入	写保护，低电平有效
VPP	输入	编程/擦除电压源，为加速编程/擦除操作提供12V电压。此管脚不能悬空
VCC	输入	芯片电压源
VCCQ	输入	输出电压源。如果工作电压在VCC范围内，这个管脚可以直接连VCC

电源供电状况变化只能发生在 RP#为低期间。RP#建议连接到系统 CPU 的 reset 信号，以保证正确的 CPU/flash 初始化以完成系统重启。在每个 VCC 和 GND 之间，以及 VPP 和 VSS 之间都必须由一个 0.1uF 的陶瓷电容，以保证退耦。

本板的 Flash 进行了兼容设计，可以在不改变 PCB 封装的前提下兼容

INTEL 和 AMD 两家共五种系列的 Flash，但同时只能焊接一种 FLASH，兼容的型号如下表 3-2 所示。

表 3-2 Flash 兼容设计

Table 3-2 Flash compatible design

厂家/型号	容量	封装	电源电压
Intel RC28FxxxJ3 系列	32Mbits～256Mbits	64-Ball Easy BGA	2.7~3.6V
Intel RC28FxxxP30 系列	64Mbits~256Mbits	64-Ball Easy BGA	V_{CC}: 1.7~2.0V V_{CCQ} : 1.7~3.6V
AMD S29GlxxxAxxFA 系列	16Mbits~64Mbits	64-ball fortified BGA	2.7~3.6V
AMD S29GlxxxNxxFA 系列	128Mbits~512Mbits	64-ball fortified BGA	2.7~3.6V
AMD S29GlxxxMxxFA 系列	32Mbits~256Mbits	64-ball fortified BGA	2.7~3.6V

AMD 和 Intel 公司的 Flash 性能稳定，供货量足，较适合运用于大批量生产中。同时，在单一 PCB 内兼容多种 Flash 也降低了制造系统对物料的依赖。兼容通过若干个零欧姆的电阻地焊接无否实现。

以上五种系列的 Flash 数据宽度都为 16bits，芯片容量覆盖了 16/32/64/128/256Mbits。它们的 IO 电压都为 3.3V。其中 Intel RC28FxxxP30 系列的内核电压为 1.8V，其余为 3.3V。用磁珠焊接与否选择 Flash 的内核电压。

表 3-3 给出了 SDRAM 的管脚功能和工作状态。已经连接硬件电路时需要注意的事项。片上系统中，在 Flash 之外，SDRAM 是另一类重要的存储设备。

通过 SDRAM 控制器，Nios II 就拥有大容量的内存支持，以弥补 FPGA 片内存储器资源的不足。对于每一个参数模块，SDRAM 都分配了四个相互独立的存储器空间供其使用。设计中采用了双缓冲的概念，当某一块存储空间被写满等待取数时，系统会自动启用第二块存储器空间存数。这种机制有效地保证了数据传输的连贯性和可靠性。四块存储器空间中，两块为参数模块提供写数据的双缓冲，另两块则作为参数模块读数据的双缓冲使用。

表 3-3 SDRAM 管脚功能及设计连接

Table 3-3 SDRAM datasheet and link design

管脚	管脚名	功能描述
CLK	时钟	系统时钟输入，所有其它输入都在该时钟的上升沿被导入芯片
CKE	时钟使能	在芯片推出工作状态时控制内部时钟信号，此时芯片会处于如下三种工作状态之一，掉电、挂起以及自刷新
CS_n	片选	使能/禁用除去 CLK、CKE 以及 DQM 之外的所有输入
BA0、BA1	Bank 地址	RAS_n 激活期间选择被激活的 bank 在 CAS_n 激活期间选择被读写的 bank
A0～A10	地址	行地址：RA0～RA10，列地址：CA0～CA7，自动预充电标志位：A10
RAS_n RAS_n WE_n	行地址选通，列地址选通，写使能	RAS_n，CAS_n 和 WE_n 配合完成操作
DQM0~3	数据输入/输出屏蔽	在读模式下控制输出缓存，在写模式下屏蔽输入数据
DQ0～ DQ31	数据输入/输出	数据传输管脚
VDD/ VSS	电压源/地	为内部电路及输入缓存提供电源
VDDQ/ VSSQ	数据输出电压源/地	为输出缓存的工电压
NC	不连接	不连接

通过配置 SDRAM 控制器的参数，Nios II 可以支持多种 SDRAM。本方案中 SDRAM 选用 HY57V643220C，32 位，容量为 8MB。同时，也通过类似于 Flash 的兼容设计，实现对若干款 SDRAM 芯片从容量到厂商的兼容。该款芯片提供时钟下降沿完成所有同步工作操作，而所有输入输出则在时钟输入的上升沿同步。所有输入输出电压电平符合 LVTTL 标准。封装为 86 脚 TSOP。根据时序参数的要求，在开发嵌入式系统的时候需要遵守这些时

序约束。由于 Nios II 系统相对于其它 SOC 开发上的便捷性，所以在 Altera 的开发工具 Quartus II 里自带的 SOPC 中即可完成全部操作。

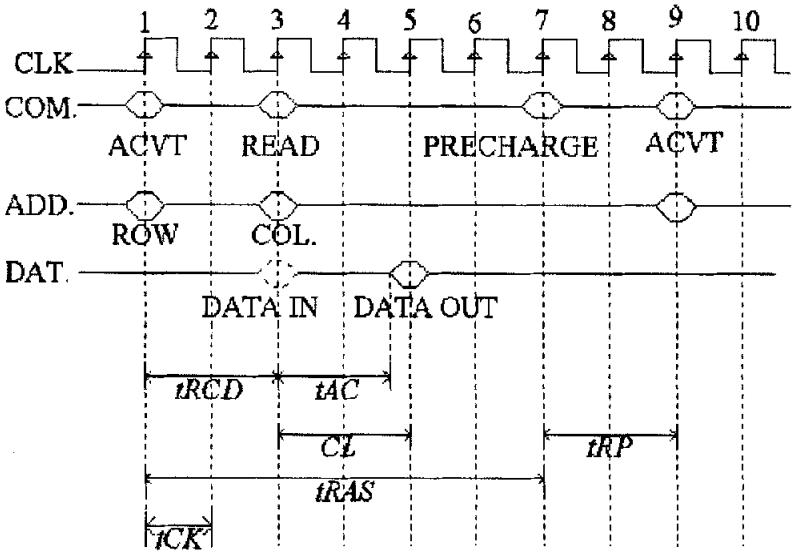


图 3-4 SDRAM 时序图

Fig.3-4 SDRAM timing

表 3-4 对具体的时序参数一一做了说明。时钟是 SDRAM 读写操作最重要的一个度量依据，所有的控制信号都在时钟沿起作用。

表 3-4 时序参数

Table 3-4 Timing parameter

定时参数	描述	参数	要求
TCK	时钟周期	—	MIN
TRCD	ACTIVE到READ/WRITE的延时时间,相当于传统DRAM的/RAS到/CAS的延时时间	2 CLK	MIN
TAC	READ到数据建立时间	—	MIN
CL	读延迟时间CAS LATENCY	2,3 CLK	MIN
TRAS	ACTIVE到PRECHARGE时间	—	MIN
TRP	PRECHARGE周期	—	MIN
TREF	刷新周期	—	MAX
TRC	CBR刷新一行所需时间	—	MIN

Nios II 提供一个功能很强的 UART。有如下特点。波特率可以固定，也可以通过软件配置。固定波特率可以节约资源。数据位可以是 7,8,9，停止位可以是 1,2。可以配置成无校验，奇校验和偶校验。同时也支持 DMA。

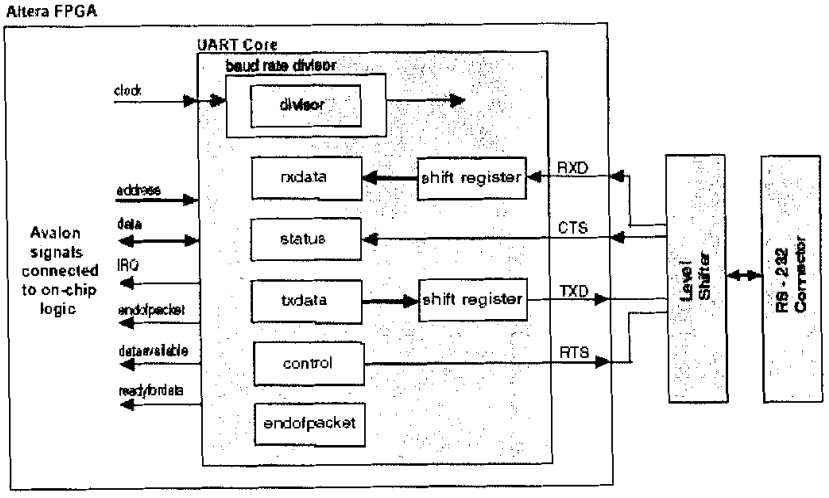


图 3-5 UART 原理图

Fig.3-5 SDRAM timing

本系统共配置 8 个 UART 口，UART1~UART8 都作为 SIR 通讯接口。虽然 SOPC 提供的 UART Core 功能非常好用，但由于向 UART 传输数据的各个参数模块数据格式的特殊性，所以仍需要在 SOPC 中把各个 UART 设置为 User Interface，也就是说，需要自己完成各个 UART 模块的 HDL 代码，然后再融入整个 Nios II 系统之中。MIR，中速红外线接口，使用自己的一套编解码逻辑和发送接收逻辑，包括 buffer 和 CRC。MIR 模块是 SIR 的有效补充，在必要的情况下，我们可以根据需要把八个 SIR 接口中的任意一个设置为 MIR，从而提升该接口的传输速率。但需要注意的是，每次最多只可以升级一个 SIR 接口。我们会在 FPGA 的硬件机制上对此进行保护性限制，不给软件犯错的机会。

ASMI 是 Cyclone II (Cyclone)系列独有的外设。由于 CPU 就是 FPGA，不可能使用软件来配置 FPGA，我们使用专用的配置芯片 EPCS4 来配置。此外，由于 Altera 公司提供的配置芯片价格过于昂贵，通过 SPI 型 Flash 取而代之可以节省大量成本，特别是在大批量生产的时候。SPI 接口，用于控制 SPI 外设，本板的 SPI 外设是 EEPROM。

EEPROM 选用 Microchip 公司的 25AA640，8 位，容量为 8KB。电压 1.8-5.5V，最大时钟频率为 1MHz。封装有 8 脚 PDIP、SOIC 和 TSSOP 三种可选。选用 SOIC。封装见图 3-6。

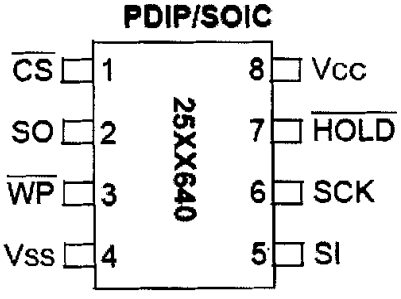


图 3-6 EEPROM

Fig.3-6 EEPROM

在设计 EEPROM 电路的时候，有一些功能管脚介于可用与不用之间，比如 WP 管脚，HOLD 管脚，都可以设置为无效状态，以简化硬件设计。当然，连到 FPGA 的管脚上也是可行的。

表 3-5 EEPROM 管脚功能及设计连接

Table 3-5 EEPROM datasheet and link design

管脚名	信号类型	功能
CS_n	输入	片选信号，低电平有效
SO	输出	串行数据输出，读芯片时，数据在串行时钟的下降沿被移至该管脚
WP_n	写保护	写保护，低电平有效
Vss	地	地
SI	输入	串行输入，包括指令、地址和数据。串行时钟的上升沿锁存输入
SCK	输入	串行时钟输入。为保持 CPU 与 EEPROM 之间的同步，时钟的下降沿更新管脚数据，上升沿时则进行读写
HOLD_n	输入	重传数据，若不使用该管脚时置为高即可

3.3 本章小结

本章主要围绕 Nios II 阐述了片上通讯传输系统的设计，对各主要器件的使用和设计进行了由内部状态到外部连接的设计说明。片上设备基本上可以分成三大类，核心 CPU 和存储器，以及其它功能外设。在本系统中，分别对应 Nios II，Flash、SDRAM、EEPROM 等，USB 芯片、参数上传逻辑。本章对这些芯片的硬件实现做了非常详细的说明。由于 USB2.0 器件硬件设计及其驱动开发是一个难点，所以放到下一章单独说明。

第4章 USB2.0 接口设计

4.1 引言

USB2.0 芯片实现红外通讯单板与主控板的通信。USB2.0 接口采用飞利浦公司 1583 方案。ISP1583 是一款性能非常优越的 USB Device 芯片。在 USB 协议中，把器件分成两类，Host 和 Device，Host 是主的一方，而 Device 则是从设备的一方。本硬件单板上需要实现 USB 从设备，从而把参数模块的数据通过 USB Device 传给上位机。

4.2 USB2.0 Device 实现

采用飞利浦公司的 ISP1583。该款芯片集成了串行接口引擎(SIE)、PIE、8K 字节的 FIFO 存储器、数据收发器和 3.3v 的电压调整器以及自带 PLL 的 12MHz 晶振。支持高速 USB2.0 传输，并且有 OTG 功能。理论最高速率可达 480Mbps。并有 7 个输入 endpoint，7 个输出 endpoint 和 2 个控制 endpoint。考虑到 ISP1583 的驱动已有一定基础，所以使用这款芯片可以缩短开发周期。

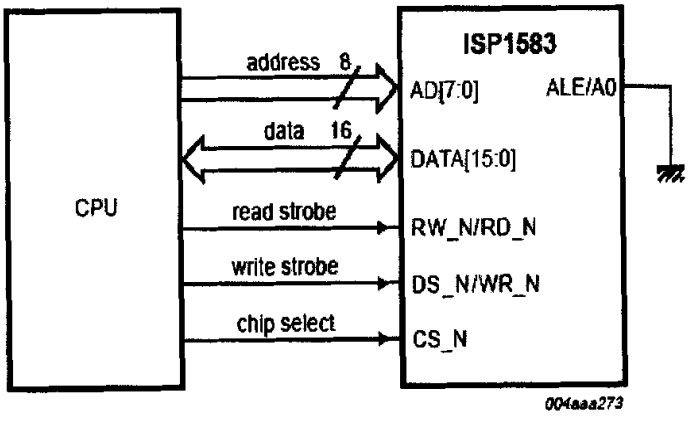


图 4-1 Nios II 与 ISP1583 间的通信

Fig.4-1 Nios II communicates with ISP1583

可通过内部上电复位和低电压复位电路复位，也可通过软件复位。工作电压为 3.3V，封装为 HVQFN64。高速的 DMA 接口，以及完全自治的多结构 DMA 操作。

表 4-1 ISP1583 管脚功能及设计

Table 4-1 ISP1583 datasheet and link design

管脚名	管脚编号	信号类型	功能描述
AGND	1、5	模拟	模拟地
RPU	2	模拟	USB D+线的外部上拉电阻连接端，通过一个1.5KΩ的电阻与VCCA(3.3)相连
DP	3	模拟	USB D+连接(模拟)
DM	4	模拟	USB D- 连接(模拟)
RREF	6	模拟	连接外部偏置电阻；通过一个12.0KΩ(±1%)的电阻接地
RESET_n	7	输入	异步复位输入端；滞后的TTL 电平；5v 的最大承受电压；为低电平时产生一个异步复位信号；与VCC(3.3V) 相连可产生上电复位(内部的POR 电路)
EOT	8	输入	DMA从模式中作为传输终止标志，若不使用，连一个10千欧电阻接到VCC(I/O)。本设计中可以作为外部EOT输入，强行中止本次DMA传输
DREQ	9	输出	本设计中连接FPGA管脚，作为DMA的握手信号输出
DACK	10	输入	本设计中连接FPGA管脚，作为DMA的握手信号输入
DIOR	11	输入	本设计中连接FPGA管脚，作为DMA控制器的读使能信号输入
DIOW	12	输入	本设计中连接FPGA管脚，作为DMA控制器的写使能信号输入
DGND	13、35	地	数字地
INTRQ	14	输入	来自ATA外设的中断请求输入，使用10千欧电阻下拉输入管脚

READY	15	输出	通用处理器工作模式：读信号(READY；输出)引脚为低电平表明ISP1583正在处理上次的命令或数据，还没有准备好接收新的命令或数据；引脚为高电平则表明ISP1583已经为下一次微处理器的读和写做好了准备
INT	16	输出	中断输出；可通过内部寄存器设置完成高/低电平有效，边沿/电平触发的选择
DA2	17	输出	不使用，悬空
CS_n	18	输入	片选输入；TTL；5V的最大承受电压
RD_n	19	输入	读使能输入
WR_n	20	输入	写使能输入
CS0_n	21	输出	不使用，悬空
CS1_n	22	输出	不使用，悬空
AD[7..0]	31、30、29、28、27、25、24、23	输入/输出	8位地址线，数据双向传送端；推挽输出；5ns 的斜率控制；TTL；5V 的最大承受电压
VCCIO	26、41、54	电压源	IO端口电压源
VCC(1V8)	32、56	输出	电压调整器输出(1.8V+/_0.15V)；有内部调整器输出电压，这个电压不能驱动外部设备，32脚使用0.1uF电容去耦。56脚使用4.7uF和0.1uF去耦
NC	33	不连接	不连接
MODE1	34	输入	ALE/A0 的功能选择(仅在分割总线工作模式下)；0-ALE 功能(地址锁存使能)；1-A0 功能(地址/数据指示)输入引脚；TTL 电平；最大可承受5v 电压。注：本设计采用通用处理器工作模式，此管脚连到VCCIO
ALE/A0	36	输入	本设计不使用该输入，接地。
DATA [15..0]	53、52、51、50、	输入/输出	16位双向数据线。数据双向传送端；推挽输出；5ns 的斜率控制；TTL；5V 的最大承

	49、48、47、46、45、44、43、42、40、39、38、37		受电压
VBUS	55	模拟	USB总线电压感应输入——用于检测host是否连上，若VBUS未连上，RPU管脚和DP管脚在大约4ns内内部断开。VBUS脉冲输出——OTG模式，本设计不使用OTG模式。连接1uF电解电容和1Mohm下拉电阻到地
XTAL2	57	输出	晶体振荡器输出(12MHZ)；连接一个基本的并联振荡电路；当XTAL1 连接一个外部时钟源时该管脚悬空
XTAL1	58	输入	晶体振荡器输入(12MHZ)；连接一个基本的并联振荡电路或一个外部时钟源(此时XTAL2 悬空)
MODE0/DA1	60	输入/输出	上电时：输入端，用来选择通用处理器工作模式下的读/写功能。0-Motorola风格：管脚26是R/W_n,管脚27是DS_n。1-8051风格：管脚26是RD_n,管脚27是WR_n。正常操作：地址输出，用来选择ATA/ATAPI 设备的任务文件寄存器。本设计中，采用8051类型的读写选通类型，即在上电时把MODE0设为1。上电完成后不使用
VCC(3V3)	61	输入	输入电压，为内部稳压器供电
BUS_CONF/DA0	62	输入/输出	上电时：选择总线结构0- 分割总线模式；AD[7:0]为多路复用的8 位地址/数据总线，DATA[15:0]为单独的DMA 数据总线1— 通用处理器工作模式；AD[7:0]为单独的8 位地址线，DATA[15:0]为16位的处理器数据总线。DMA 多路复用到DATA[15:0]的处理器

			数据总线上。正常操作：地址输出，用来选择ATA/ATAPI 设备的任务文件寄存器。本设计中，采用通用处理器工作模式，即在上电时把BUS_CONF设为1。上电完成后不使用
WAKEUP	63	输入	当此管脚输入高电平时，芯片不会进入suspend状态。唤醒输入(边沿触发)；输入一个由低到高的电平跳变信号将系统从"挂起"状态唤醒。当不使用时，用10千欧电阻下拉到地。输入端：滞后的TTL 电平； 5V 的最大承受电压
SUSPEND	64	输出	挂起状态标志输出，作为降功耗应用的电源开关控制输出或者作为上电应用的CPU重启信号。COMS输出，8mA驱动
GND	Exposed die pad	—	PCB布线期间连接到DGND

表 4-1 有如下注意事项。引脚符号的后面符号为_n，表示低电平有效。所有的输出端和 I/O 口都能够提供 4mA 的电流。在所有的供电引脚上增加一个去耦电容(0.1uF)。为了得到良好的 EMI 性能，可再增加一个 0.01uF 的电容，并联到 0.1uF 的电容两端。

DMA 总线为三态，直到执行一个 DMA 命令。控制信号不是三态的。在 Vcc(I/O)=3.3 V 时，最大承受电压为 5V。

4.3 ISP1583 芯片使用设计说明

由 Nios II 完成对 ISP1583 芯片的预配置，最主要的工作模式选择由对管脚 BUS_CONF 和 MODE0 的设置完成。ISP1583 内部含有两种总线结构配置，本设计选用通用处理器工作模式(BUS_CONF=1)。AD[7:0]：8 位地址总线(选择目标寄存器)；DATA[15:0]：16 位数据总线(处理器和 DMA 共享)；控制信号：RD_n 和 WR_n，CS_n。

DMA 接口：DATA[15:0]为数据总线，DIOR 和 DIOW 读/写选通信号。对本方案来说，由于选用了通用处理器工作模式，所以只能选用通用 DMA 的从机工作模式。通用 DMA 主机工作模式和 ATA 模式都与本设计无关。

ISP1583 和外部存储器或外部设备之间的高带宽的数据传输是通过集成的 DMA 控制器来控制完成的。通过"写"对应的 DMA 寄存器来配置 DMA 接口。ISP1583 含有一个 8K 字节的内部 FIFO 存储器，供所有正在使用的 USB 端点共享。

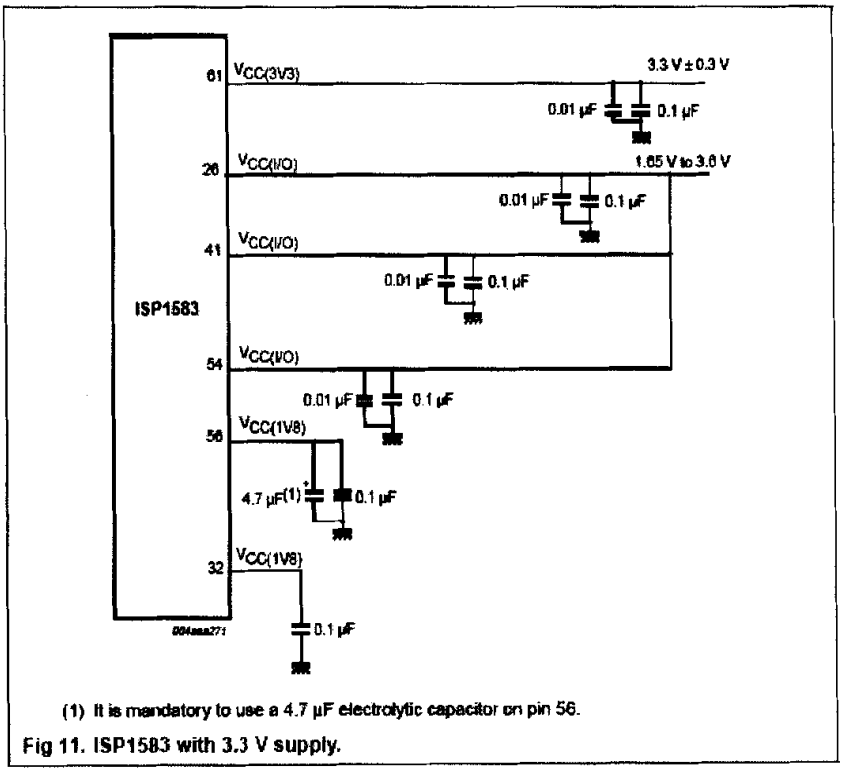


图 4-2 ISP1583 供电系统

Fig.4-2 ISP1583 with 3.3V supply

ISP1583 含有 7 个 IN 端点，7 个 OUT 端点和 2 个控制端点，它们的固定长度是 64K 字节。7 个 IN 和 7 个 OUT 端点可以单独使用或禁止。这些端点的端点类型(中断，同步或批量)和信息包大小根据实际需要进行设置。利用双缓冲器的配置来增加这些数据端点的数据吞吐量。由于本款芯片支持 USB 全速/高速两种工作模式，所以我们需要进行相关设计。当设备和主控制器已经在 USB 电缆两端连接好了，ISP1583 器件就默认为全速(FS)状态直至主控制器发出一个总线复位信号。考虑到所有模块同时进行最大速度传输

的情况下，USB 全速码流速率已够用，所以传速本方案中采用全速工作模式。

设备与 USB 的连接是通过一个 1.5KΩ 的上拉电阻将 DP 线(对于全速 USB 器件)置为高来实现的。在 ISP1583 中，RPU 和 VCC(3V3)之间连接了一个 1.5KΩ 的外部上拉电阻。上拉电阻再经 RPU 与 DP 线相连，此时方式寄存器的 SOFTCT 被置位，从而实现了软件连接。在一个硬件复位发生后，上拉电阻默认为断开(SOFTCT=0)。USB 总线复位时，SOFTCT 位的值仍保持不变。

ISP1583 的供电电压为 3.3V。利用 VCC=3.3V 的电压对 ISP1583 供电时，集成的电压调整器就给内部逻辑电路和 USB 收发器提供 1.8V 的电压。具体电路的连接如图 4-2 所示。

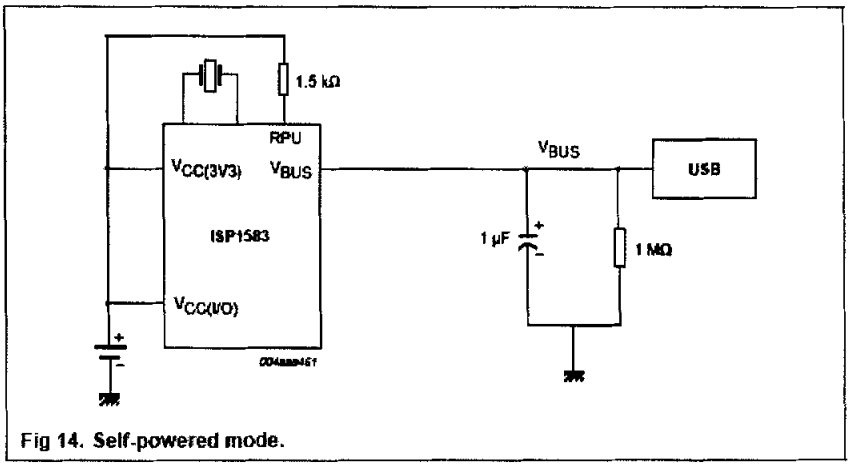


Fig 14. Self-powered mode.

图 4-3 ISP1583 自供电模式

Fig.4-3 ISP1583 with self-powered mode

ISP1583 提供两种可选的供电模式，分别为自供电模式和总线供电模式。在本单板内采用自供电模式，即必须自己给芯片供电而不依赖 USB 总线的供电能力。

图 4-3 对此供电模式的电路系统连接给了详细说明。采用一个 1uF 的电容和一个兆欧的电阻对 USB 总线进行接地，完成滤波。同时 VCC(I/O)和 VCC(3V3)管脚都需要滤波接地。

ISP1583 的时序约束也需要加强关注。本设计中选用了通用处理器模式，

也即有 8 位数据总线、16 位地址总线，以及相对应的控制信号。此模式下的 DMA 传输在数据总线上通过数据/地址复用实现。不同于一般的分立总线模式。同时，还需要在摩托罗拉读写模式和 8051 读写模式间进行选择。这可以根据个人偏好。本方案中选用 8051 读写控制信号方式。即采用读使能信号和写使能信号去配合时序。摩托罗拉控制方式与此稍有差异，但是达到的目的都是类似的。都需要保证数据的可靠传输。

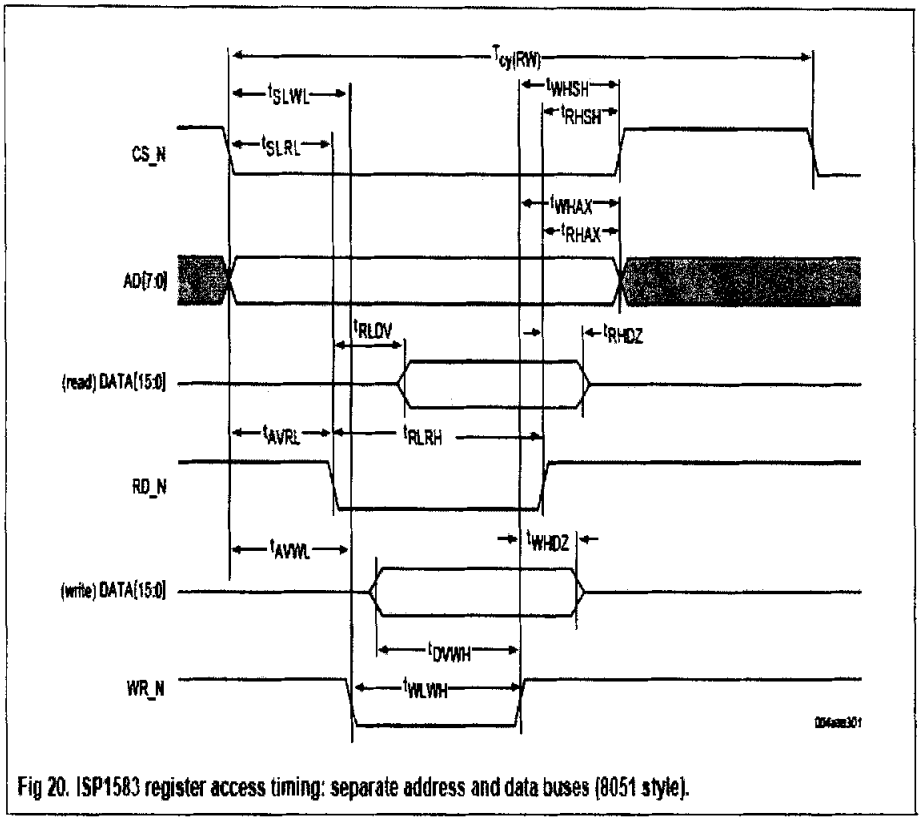


Fig 20. ISP1583 register access timing: separate address and data buses (8051 style).

图 4-4 通用处理器读写时序

Fig.4-4 General processor read/write timing

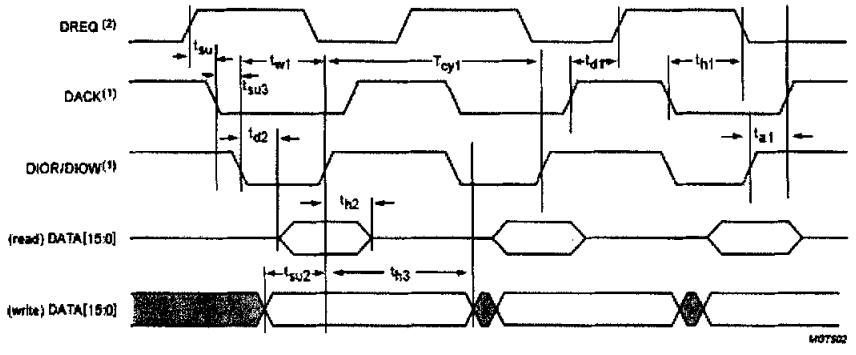
对于读数据时序，tAVRL＋tRLDV 如果为 Yns，而时钟为 Zns，那么就必须在这段时间放 s 个时钟周期，使 s 个时钟周期的总时长大于 Y，以保证地址/片选信号给出之后到数据有效所需的延时能够被满足。对于写数据时序，tDVWH 即为数据建立时间，tWHDZ 为数据保持时间。都需要满足。

表 4-2 通用处理器时序参数

Table 4-2 General processer timing parameter

（ns）

符号	参数	最小	最大
读	—	—	—
tRLRH	RD_n低脉冲宽度	>tRLDV	—
tAVRL	RD_n 变低前地址建立时间	0	—
tRHAX	RD_n 变高后地址保持时间	0	—
tRLDV	RD_n 低到数据有效的延时	—	26
tRHDZ	RD_n 高到数据输出三态的延时	0	15
tRHSH	RD_n 高到 CS_n 高的延时	0	—
tSLRL	CS_n为低到RD_n为低时间	2	—
tWLWH	WR_n低脉冲宽度	15	—
tAVWL	WR_n 变低前的地址建立时间	0	—
tWHAX	WR_n 变高电后的地址保持时间	0	—
tDVWH	WR_n 变高前的数据建立时间	11	—
tWHDZ	WR_n 变高后的数据保持时间	5	—
tWHSH	WR_n高到CS_n高的延时	0	—
tSLWL	CS_n为低到WR_n为低的延时	2	—
Tcy(RW)	读/写周期	50	—
tRDY1	最后一次访问的 READY 高到 RD_n/WR_n 高的时间	—	91



DREQ is asserted for every transfer.
Data strobes: DIOR (read), DACK (write).
(1) Programmable polarity: shown as active LOW.
(2) Programmable polarity: shown as active HIGH.
Fig 31. GDMA slave mode timing: DIOR (master) or DACK (slave).

图 4-5 通用 DMA 从机方式时序

Fig.4-5 General DMA slave mode timing

此外，写的地址建立时间 tAVWL 也需要保证，而对于读的地址建立时间 tAVRL 一般都能满足，所以无需特别关注。通过设置 DMA 配置寄存器中的相关位可以设置 DMA 传输中的 BURST 和 MODE 值，决定设置如下，BURST=01H,MODE=00H。在此种设置下，每发生一次传输，DREQ 就有效一次。数据选通信号为：DIOR(读)，DIOW(写)。

对于读时序，td2(选通有效后的读数据有效延时)必须满足，为 20ns。对于写时序，tsu2(选通无效前的写数据建立时间)为写数据建立时间，th3(选通无效后的写数据保持时间)为写数据保持时间，都必须满足。表 4-3 给出了 DMA 时序参数。

表 4-3 DMA 时序参数

Table 4-3 DMA timing parameter

(ns)

符号	参数	最小	最大
Tcy1	读/写周期	75	—
tsu1	第一个 DACK 有效前的 DREQ 建立时间	10	—

td1	最后的选通无效后 DREQ 有效的延时	33.33	—
th1	最后的选通有效后 DREQ 有效保持时间	0	53
tw1	DIOR/DIOW 脉冲宽度	39	600
tw2	DIOR/DIOW 恢复时间	36	—
td2	选通有效后的读数据有效延时	—	20
th2	选通无效后的读数据保持时间	—	5
tsu2	选通无效前的写数据建立时间	10	—
tsu3	DIOR/DIOW 有效前的 DACK 建立时间	0	—
ta1	DIOR/DIOW 无效到 DACK 无效的时间	0	30

4.4 USB 驱动程序设计方案

驱动程序完全按照 WDM 分层结构设计，响应操作系统中 I/O 管理器，PNP 管理器，电源管理器的 IRP 请求，实现 USB 设备的即插即用管理，设备及系统电源管理以及应用程序对设备的各种操作。

图 4-6 是驱动程序总体结构框图，在系统加载驱动程序时，首先从入口函数获得注册表信息；驱动程序入口函数包括在入口模块中，其中还有对各种 IRP(根据 IRP 主码)所指派的各自的处理函数，这些处理函数分别在相应的 PNP 模块，电源管理模块，I/O 控制模块，数据传输模块中实现。针对程序的调试以及出现的错误，异常则在调试及异常模块中处理[24]。

驱动程序按各模块功能介绍如下。第一个是驱动程序入口模块。

驱动程序入口模块通过调用 DriverEntry()函数来初始化驱动程序本身；将要处理的 IRP 按主功能码 IRP_MJ_XXX 分派给各自的处理函数；也要指定 Unload()驱动程序卸载函数，由于在 DriverEntry()中并不分配任何资源，所以 Unload()并不需要进行任何操作。

添加设备函数 AddDevice()：在设备插入时，系统 PNP 管理器为每个设备调用 AddDevice()函数，用来加载和初始化设备。指定 IRP 串行处理 StartIO()函数：当多个 IRP 产生时，需要采取串行处理策略。

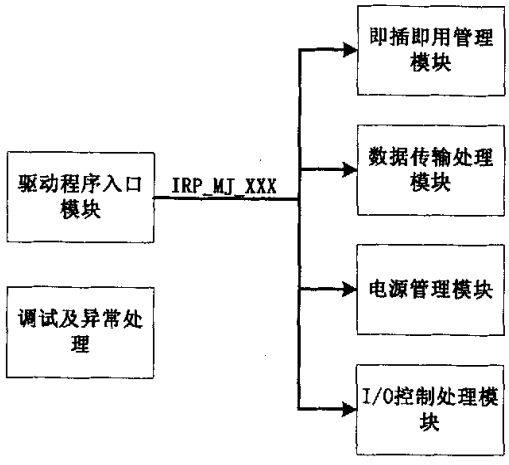


图 4-6 驱动程序结构

Fig.4-6 Drive program construction

说明一下功能。驱动程序入口模块通过调用 DriverEntry()函数来初始化驱动程序本身；将要处理的 IRP 按主功能码 IRP_MJ_XXX 分派给各自的处理函数；也要指定 Unload()驱动程序卸载函数，由于在 DriverEntry()中并不分配任何资源，所以 Unload()并不需要进行任何操作[24]。

添加设备函数 AddDevice()：在设备插入时，系统 PNP 管理器为每个设备调用 AddDevice()函数，用来加载和初始化设备。指定 IRP 串行处理 StartIO()函数：当多个 IRP 产生时，需要采取串行处理策略。见驱动程序入口模块框图 4-7。

第二个是即插即用管理模块，下面对这个模块做说明。PnP 管理器使用 IRP 来指导启动，停止和删除设备，所有即插即用 IRP 的主功能代码都是 IRP_MJ_PNP。自动识别已安装的设备 ；硬件资源的动态重分配；自动加载正确的驱动程序；提供使得在硬件环境发生变化时，驱动程序和用户模式代码得到通知的机制。

说明一下功能，PnP 管理器使用 IRP 来指导启动，停止和删除设备，所有即插即用 IRP 的主功能代码都是 IRP_MJ_PNP。

自动识别已安装的设备，硬件资源的动态重分配，然后自动加载正确的驱动程序。最后提供在硬件环境发生变化时，驱动程序和用户模式代码得到通知的机制。见模块设计框图 4-8。

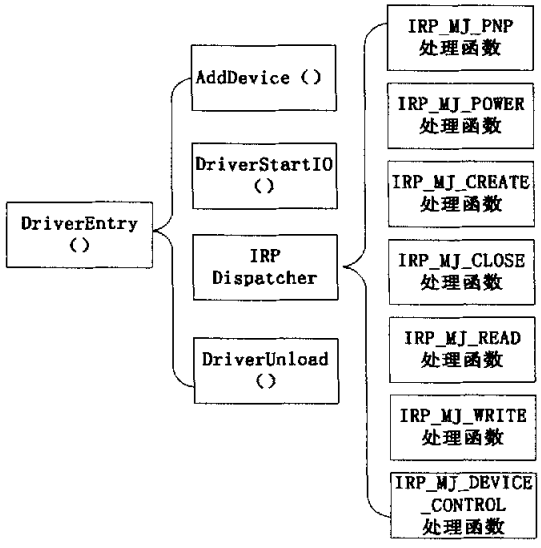


图 4-7 驱动程序入口模块框图

Fig.4-7 Drive program entry construction

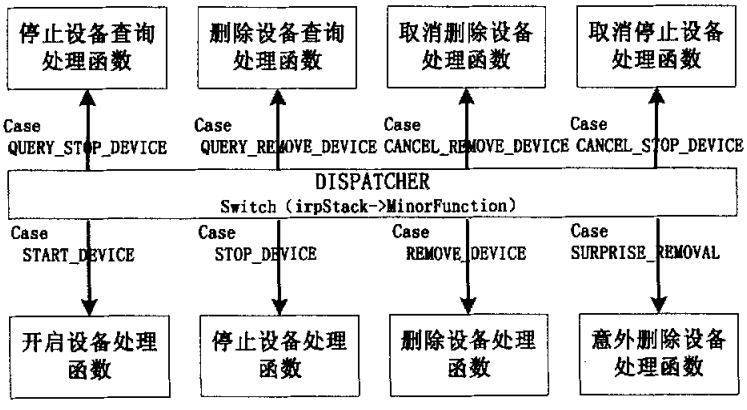


图 4-8 即插即用管理模块框图

Fig.4-8 PNP management construction

第三个是电源管理模块。电源管理器通过处理 IRP_MJ_POWER 请求发送给驱动程序来请求电源操作。在此模块中可以实现：制定一个新的电源状态；可以查询设备或者系统的电源状态；对设备或者系统电源状态作出更改，也可以查询查询在电源状态方面的更改是否可行。

作一下功能说明。电源管理器通过处理 IRP_MJ_POWER 请求发送给驱动程序来请求电源操作。一个请求可以制定一个新的电源状态，或者查询在电源状态方面的更改是否可行。

所有 WDM 驱动都支持以下三种电源管理 IRP 次功能码，IRP_MN_QUERY_POWER：查询电源状态；IRP_MN_SET_POWER：设置电源状态；IRP_WAIT_WAKE：唤醒[24]。

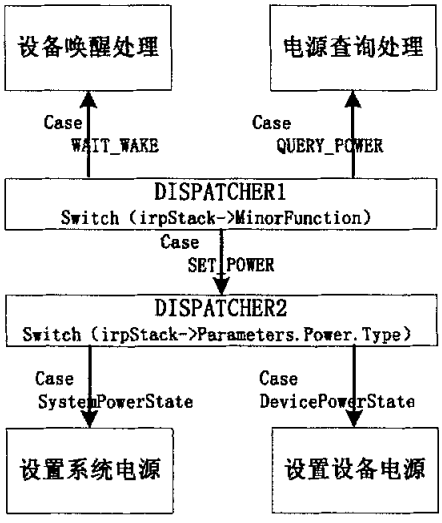


图 4-9 电源管理模块框图

Fig.4-9 Power management construction

在驱动程序中，设备电源状态有三种，所对应的名称如下：D0 - PowerDevice D0；D1 - PowerDevice D1；D2 - PowerDevice D2；D3 - PowerDevice D3。电源管理模块见图 4-9。

第四个是数据传输处理模块，响应应用程序产生的主码为 IRP_NJ_READ 和 IRP_MJ_WRITE，实现主机对 USB 设备的读写。针对 BULK 传输模式，读写存在细微差别。可以用一个函数来实现；但是根据一个 IRP 是否需要分为两个以上的传输事务，则应区别对待：SINGLE_READWRITE 单独读写：IRP 可以在一个传输事务中完成；STAGED_READWRITE 分段读写：IRP 较大，分为多个块完成。

下面作一下功能说明。数据处理传输模块主要是对主功能码为 IRP_MJ_WRITE 和 IRP_MJ_READ 的 IRP 进行处理，由于 BULK 类型传输

读写差别很小所以可以用一个函数实现，但是一个 IRP 是否需要分多个传输事务完成，在实现上是有差别的。所以针对 bulk 传输要区分：SINGLE_READWRITE()单独 BULK 读写操作；STAGED_READWRITE()分段 BULK 读写操作[24]。图 4-10 就是单独 BULK 读写操作的实现框图。

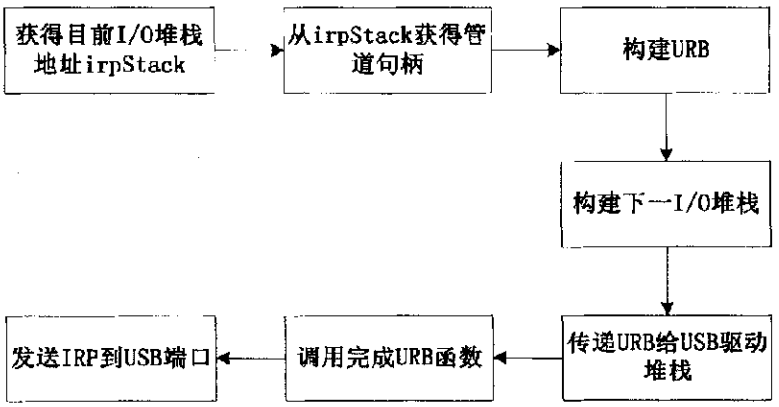


图 4-10 单个读写操作

Fig.4-10 Single read/write()

URB 函数的工作过程见图 4-11 所示。

第五个为 I/O 控制处理模块，主码为 IRP_MJ_DEVICE_CONTROL 的 IRP 在 I/O 控制模块中处理，主要完成应用程序对设备的 I/O 控制。程序根据 IO 堆栈单元的 Parameters.DeviceIoControl.IoControlCode 来决定采取什么操作，IO 控制码 0X0800 之后的分给用户使用，可以自定义用户命令[24]。

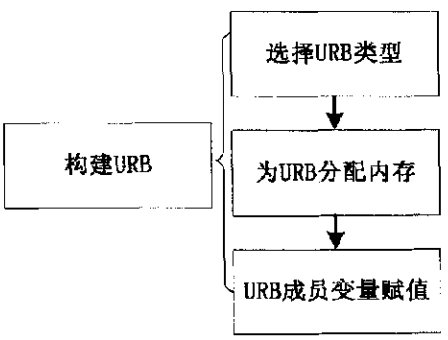


图 4-11 构建 URB

Fig.4-11 URB construction

做一下功能说明，IO/控制处理模块主要是对 I/O 请求 IRP(主码为 IRP_MJ_DEVICE_CONTROL)的不同 IO 控制命令码进行处理函数分派 (IRP_MJ_DEVICE_CONTROL DISPATCHER)，如图 4-10 所示。

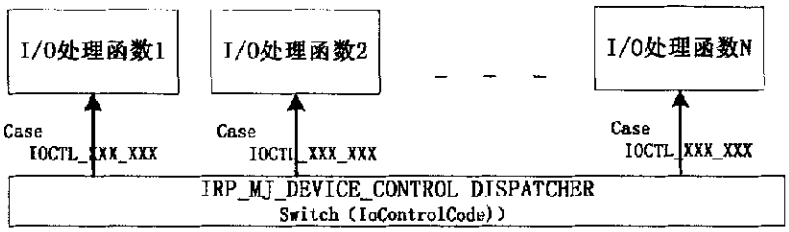


图 4-12 I/O 控制处理流程框图

Fig.4-12 I/O control construction

4.5 本章小结

本章对 Nios II 系统架构中较为重要的一部分 USB2.0 芯片 ISP1583 的芯片设计和内部驱动函数做了较为详细的设计说明。ISP1583 作为一款 USB 从芯片，带有 OTG 功能，如果进一步挖掘，还可以开发出更精彩的应用。USB 驱动开发是一个费时费力的难点，本章对 USB 驱动程序的各个子模块和其内部实现都做了较为详细的说明。

本课题设计的 USB 口完成了汇集前端参数模块数据流的任务，取得了较为理想的开发效果，并在实际产品中获得了应用。

结 论

CPU 在嵌入式系统开发中占据着极其重要的位置，但是在消费电子、通信业等产业开发中都存在一体化能力差，成本高，开发速度慢等问题，为了解决这些问题，引入了软核 CPU 的概念。可以说这种 CPU 是可用厂商集成环境控制和再定义的 CPU。选用不同的代码模块就可以实现不同的功能，而且可以升级更新。本课题的主要任务就是设计一个系统平台实现 Nios II 系统，开发其三大高级应用，并且在具体的产品开发中获得应用。

由于嵌入式系统要解决软硬件一体化集成的问题，对单板体积、系统功耗都有很高的要求，所以，要精心选择软硬件平台。本方案的 Nios II 系统平台主要是基于 FPGA 进行嵌入式开发的高效平台，这是一种高速、低成本的新颖平台。该设计的硬件采用 Cyclone II 为核心的系统平台来实现。FPGA 是一种高速、可编程、实现简单的芯片。它的系统主频和集成度逐步提高，故在这种硬件平台上实现嵌入式系统具有可行性。

论文取得了下列研究成果：

(1) 深入研究了基于 Nios II 的嵌入式系统架构，详细分析了 Nios II 的嵌入式设计流程，提出了高速数据传输应用，说明与目前的异步数据传输之间相比较的优势所在。

(2) 提出了 Nios II 架构下多 CPU 和多 Master 实现的具体设计以及实现方法，并且通过开发板进行了验证。

(3) 对 Nios II 架构进行了具体产品应用开发，获得成功，目前迈瑞公司的新监护仪产品硬件架构中有四块以上单板采用了 Nios II 架构，实现了低成本高效的嵌入式方案。课题在理论应用和设计方法学上都有重大突破，其中的多个子命题已经提交了专利申请，包括多 Master 设计、模块间自适应切换等。

嵌入式系统是通信、航天、医疗电子等关键行业科研开发中必不可少的一部分，Nios II 系统降低了开发成本，提升了系统效率，解决了目前嵌入式系统中存在的部分瓶颈问题，实现这一系统平台对嵌入式系统应用有着一定的理论和实践意义。

参考文献

1 王效平, 刘捷臣. 集成电路进入片上系统时代. 微处理机, 2000,2:1~5.

2 徐善锋, 初秀琴等. 21 世纪微电子芯片设计技术发展方向. 微电子学, 2001,31(5):313~316

3 夏宇闻. 片上系统设计与 EDA. 单片机与嵌入式系统应用, 2002,1:16~23

4 魏少军. SOC 设计方法学. 电子产品世界, 2001,6:15~18

5 林学龙. SOC 技术水平以及发展趋势. 单片机与嵌入式系统应用, 2002,4:20~22

6 吉利久. SoC 的技术支持及嵌入式系统设计. 单片机与嵌入式系统应用, 2002,4:13~17

7 于宗光. 开发 SOC 面临的问题及发展展望. 微电子技术, 2001,29(2): 36~41

8 孟宪元. 可编程器件实现片上系统. 测控技术, 2000,19(10):31~35

9 Kevin Skahill. 可编程逻辑系统的 VHDL 设计技术. 朱明程. 东南大学出版社. 2002:27~56

10 王金明, 杨吉斌. 数字系统系统设计与 VerilogHDL. 电子工业出版社, 2002:35~51

11 徐志军, 徐光辉. CPLD/FPGA 的开发与应用. 电子工业出版社, 2002:101~120

12 李景华, 杜玉远. 可编程逻辑器件与 EDA 技术. 东北大学出版社, 2002:85~96

13 林敏, 方颖力. VHDL 数字系统设计与高层次综合. 电子工业出版社, 2002:32~45

14 袁俊泉. VerilogHDL 数字系统设计及其应用. 西安电子科技大学出版社, 2002:85~104

15 张明. VerilogHDL 实用教程. 电子科技大学出版社, 1999:65~90

16 潘松, 王国栋. VHDL 使用教程. 电子科技大学出版社, 2000:101~120

17 姚力. Nios 平台简介. 研发特刊. 2003:5~20

18 钱晓捷, 陈涛. 16/32 位微机原理、汇编语言及接口技术. 机械工业出版社, 2001:65~80

19 王志华,邓仰东. 数字集成系统的结构化设计与提高. 清华大学出版社,

2000:120~138
20 杨之廉. 大规模集成电路设计方法导论. 清华大学出版社, 1998:135~140
21 王成儒, 李英伟. USB2.0 原理与工程开发. 国防工业出版社, 2004:120~138
22 陈国通. 数字通信. 哈尔滨工业大学出版社, 2002:101~110
23 孙斌, 贾杏池. 软硬件系统可测试性设计的研究. 电子测量测试, 2000,12:15~20
24 王哲. 小平台嵌入式 USB 驱动开发. 西安交通大学硕士论文. 2005：8-20
25 An Chengquan, Zhou Tingxian. A method of generating chaotic spread spectrum sequences based on multilevel quantification. 2003 IEEE International Conference on Acoustics, speech, and Signal Processing. 2003,4：7~13
26 L Nguycn. Self-Encoded Spread Spectrum and Multiple Access Communication. Spread Spectrum Technique and Application ,2002 IEEE sixth International Symposium.2000,2：2~12
27 R F Omondroyd. PN code synchronizers for direct sequence spread spectrum system a comparative evaluation. IEEE Colloquium on New Synchronisation Techniques for Radio System.1995,11：4~11
28 A. Kostic, S. Seetharaman. Digital Signal Processors in Cellular Radio Communication. IEEE Communication Magazine.1997, 35(12): 22~35
29 J. Mitola. Software Radio Arichitecture: A Mathematical Perspective. IEEE Journal on Selected Areas in Communication. 1999, 17(4):514~538
30 P. G. Cook, W. Bonser. Architectural Overview of the Speakeasy System. IEEE Journal on Selected Areas in Communication. 1999, 17(4): 650~661
31 J. Mitola. Technical Challenges in the Globalization of Software Radio. IEEE Communication Magazine. 1999, 37(2): 84~89
32 K.P. Lim, A.B. Premkumar. A modular approach to the computation of convolution sum using distributed arithmetic principles. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. 1999, 46(1): 92~96
33 T. S. Chang, C.W. Jen. Hardware-efficient implementations for discrete function transforms using LUT-based FPGAs. Computers and Digital Techniques, IEE Proceedings. 1999, 146(6): 311~315

34 T. S. Chang, C.W. Jen. Hardware-efficient pipelined programmable FIR filter design. Computers and Digital Techniques, IEE Proceedings. 2001, 148(6): 228~230

35 Sunjung Park, Joonho Lim, Soo-Ik Chae. Discrete-time cellular neural networks using distributed arithmetic. Electronics Letters. 1995, 31(21): 1851~1852

36 V.P. Dimri. Design of the unimodular shaping filter. IEEE Transactions on Signal Processing. 1991, 39(18): 1889~1991

37 Seo How Low, Yong Ching Li. A new approach to synthesize sharp 2D half-band filters. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. 1999, 46(8): 1106~1110

38 B.R. Murphy, I. Watanabe. Digital shaping filters for reducing machine vibration. IEEE Transactions on Robotics and Automation. 1993, 8(2): 285~289

39 Lacky .R .J, Upmal .D .W. Speakeasy: The Military Software Radio.IEEE Communication Magazine. 1995, (5): 2023~2027

40 F. Monteiro, S. Philip, A. Dandache. A New Processor Architecture Dedicated to Digital Modem Applications. Circuits and Systems. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on, 1998: 494 ~497

41 A. Azel, M. Zhili, N. Youssef. Optimized DSP Implementation of GMSK Software Modem for GSM transceiver. Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st, 2000: 2573 ~2577

42 Song Wu, XiaoLin Lu, W. Y. Chen. TI DSP Implementation of a Medium Speed DSL (MDSL) for Multimedia Applications. Acoustics, Speech and signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, 1998: 3149 ~3152

43 S. Chow, B. Gagnon, N. Serinken. DSP Implementation of a Multitone FSK Modem for HF Radio Data Systems. HF Radio Systems and Techniques, Seventh International Conference on (Conf. Publ. No. 441), 1997: 125 ~129

44 R. Link, V. C. M. Leung. Design of a DSP-based code-phase-shift Keying Modem for Wireless Local Area Network Applications. Vehicular Technology Conference, 1997. IEEE 47th, 1997: 1069 ~1073

45 H. Johansson, L. Wanhammar. High-speed recursive digital filters based on the frequency-response masking approach. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. 2000, 47(1): 50~53

46 F. Harris. On the Design of Pre-equalized Square-root Raised Cosine Matched

Filters for DSP based Digital Receivers. Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on, 1993: 1291 ~1295

47 B. Koblents, P. J. McLane. Asynchronous Timing Recovery in DSP based PSK Modems. Signals, Systems and Computers, 1992. 1992 Conference Record of The Twenty-Sixth Asilomar Conference on, 1992: 632 –641

哈尔滨工业大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《基于 Nios II 的通用嵌入式系统平台》，是本人在导师指导下，在哈尔滨工业大学攻读硕士学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签字 丁能 日期：2005 年 12 月 8 日

哈尔滨工业大学硕士学位论文使用授权书

《基于 Nios II 的通用嵌入式系统平台》系本人在哈尔滨工业大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归哈尔滨工业大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解哈尔滨工业大学关于保存、使用学位论文的规定，同意学校保留并向有关部门送交论文的复印件和电子版本，允许论文被查阅和借阅。本人授权哈尔滨工业大学，可以采用影印、缩印或其他复制手段保存论文，可以公布论文的全部或部分内容。

保 密□，在 年解密后适用本授权书。

本学位论文属于

不保密☑。

(请在以上相应方框内打"√")

作者签名：丁能 日期：2005 年 12 月 8 日

导师签名：张晓东 日期：2005 年 12 月 8 日

致 谢

张钦宇老师是我的硕士导师，他严谨的工作和治学态度，开阔的学术思路，使我受益菲浅。

本次毕业设计得到了迈瑞公司诸多工程师的指导、合作与帮助。系统工程师姚力是我的企业导师，他对我的指导尤多，从工作态度到学习方法、具体开发进行了全面地点拨，对我今后的学习和工作产生了重要的影响。

来自西安交大的周毅同学是我同组的实习生，在项目中合作较多，一起克服了许多技术难题。

在此首先向张钦宇老师和姚力老师致以诚挚的谢意。此外对迈瑞公司监护资源部硬件组的万力励，刘彬,陈巍，黄洋等工程师对我的热心帮助表示由衷的感谢。