分类号 TP31 学号 04040086
UDC

工学硕士学位论文

基于 FPGA 的系统设计和应用研究

硕士生 姓名 刘 凯

学 科 领 域 信息与通信工程

研 究 方 向 嵌入式系统设计

指 导 教 师 徐 晖 教授

指 导 教 师 徐 欣 副教授

国防科学技术大学研究生院

二〇〇五年十一月

The Research on System Design and Application Based on FPGA

Candidate: Liu Kai
Advisor: Prof. XuHui
Assoc. Prof. XuXin

A thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
in INFORMATION & COMMUNICATION ENGINEERING
Graduate School of National University of Defense Technology
Changsha, Hunan, P.R.China
November, 2005

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：基于 FPGA 的系统设计和应用研究

学位论文作者签名：刘凯　　日期：2005 年 12 月 1 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于 FPGA 的系统设计和应用研究

学位论文作者签名：刘凯　　日期：2005 年 12 月 1 日

作者指导教师签名：徐欣　　日期：2005 年 12 月 1 日

目 录

表 目 录

图 目 录

摘 要

在现在的数字系统设计中，FPGA 发挥着越来越重要的作用。本文首先分析了 FPGA 的内部结构特点，对 FPGA 设计技术进行了详细阐述，针对现代数字系统的特点，研究分析了 FPGA 的设计流程，提出了一种优化设计时序的 FPGA 应用系统设计流程。然后针对传统电路设计技术在航天惯测系统中的局限性，本文引入了 IP 复用设计技术，研究了在 FPGA 内进行 IP 设计应遵循的原则和方法，并使用 FPGA 设计了一种航天惯测系统。本文最后对基于 FPGA 的微处理器 IP—MicroBlaze 的结构性能进行了分析，并基于 MicroBlaze 设计了一种 CPCI 的动态可重构数据采集系统。

主题词：FPGA SOPC IP 复用 微处理器

ABSTRACT

FPGA plays a more and more important role in modern digital system design. In this thesis, first, the architecture characteristic of FPGA is analyzed, and FPGA design techniques are discussed, a FPGA design flow for optimizing timing is advanced. Secondly, to the question that traditional design method has a bottleneck for spaceflight inertial navigation system, the IP reuse technology is introduced, the basic principles and the coding guidelines about design for reuse is introduced, a spaceflight inertial navigation system is designed based on FPGA. Finally, the architecture of an embedded microprocessor IP-MicroBlaze is analyzed, a dynamic reconfigurable DAQ system based on CPCI in FPGA is implemented used MicroBlaze.

Key Words: FPGA SOPC IP Reuse MicroBlaze

第一章 绪论

在现在的数字系统设计中，FPGA 发挥着越来越重要的作用。从简单的接口电路设计到复杂的状态机，甚至片上系统设计，FPGA 所扮演的角色已经不容忽视。FPGA 的可编程特性带来了电路设计的灵活性，缩短了产品的上市时间，其性能已完全能够与 ASIC 媲美，而且由于 FPGA 的逐步普及，其性能价格比足已与 ASIC 抗衡。因此 FPGA 在现代数字系统设计领域将发挥越来越重要的作用。

1.1 FPGA 的特点与发展趋势

FPGA 是英文 Field Programmable Gate Array 的缩写，即现场可编程门阵列，它是在 PAL、GAL、EPLD 等可编程器件的基础上进一步发展的产物。它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。

与 ASIC 相比，可编程逻辑器件研制周期较短，先期 NRE 费用较低，也没有最少订购数量的限制，所有这一切简化了库存管理。随着每个门电路成本的降低和每个器件中门电路数量的增加，可编程逻辑器件正在大举打入传统的门阵列领域，并已有少量的打入了标准单元 ASIC 的领域。系统设计师和制造商们开始探索和利用"系统可重编程性" 的功能，其目的是为了在最终系统交付给用户使用后仍可纠正其错误和进行功能升级，或是为了实现"可重新配置计算"，亦即用固定数量的逻辑门电路来实现时分复用多功能。

片上 RAM 和单芯片 ASIC/可编程逻辑混合器件的数量越来越多，以及摩尔定律预测的集成度发展趋势，正在促进有效门电路数量的爆炸性增长。这些因素最终将使人们期盼已久的单片系统成为现实。为了在仍然能满足产品上市时间要求的时间内充分利用硅片的资源，许多设计师正在从传统的低级状态机和原理图输入综合转向高级语言，比如 VHDL 和 Verilog，甚至转向 C 语言等传统软件语言。

FPGA 的逻辑门数已达 1 千万，内核速度达到 400MHz，能提供高达 11Gbps 的芯片间通信速度，随着工艺微缩，当前的 FPGA 已经能够以内嵌 DSP 核心的方式在诸多高端应用中实现传统 DSP 的工作，且可编程特性将大幅压低成本，并加快设计周期。现在的 FPGA 芯片中不只是包含可编程逻辑功能模块、可编程输入输出模块和可编程内部互连资源等基本的资源，还集成了存储器（Block RAM 和 Disturbed RAM）、数字时钟管理单元（完成分频/倍频、数字锁相和延迟功能的 DLL 和 DCM）、算术运算单元（乘法器、加法器）以及特殊功能模块（MAC、

微处理器等硬 IP 核）等更丰富的资源，与过去 FPGA 仅仅用作胶合逻辑不同，现在 FPGA 已经被用来实现主要系统功能。而与此同时，它仍然保持着非常合理的成本，因此，与 ASIC 和定制 IC 相比，FPGA 是一种更具有吸引力的选择。FPGA 的发展趋势主要体现在以下几个方面：[1]

1、向更高密度、更大容量的千万门系统级方向迈进

2、向低成本、低电压、微功耗、微封装和绿色化发展

3、IP 资源复用理念将得到普遍认同并成为主要设计方式

4、MCU、DSP、MPU 等嵌入式处理器 IP 将成为 FPGA 应用的核心

随着处理器以 IP 的形式嵌入到 FPGA 中，ASIC 和 FPGA 之间的界限将越来越模糊，未来的某些电路版上可能只有这两部分电路：模拟部分（包括电源）和一块 FPGA 芯片，最多还有一些大容量的存储器。这一切表明，可编程片上系统（System on a Programmable Chip：SOPC）正在成为 FPGA 最为重要的发展方向[2]。

1.2 SOPC 的技术特点

可编程片上系统（SOPC）是一种特殊的数字电子系统：首先它是片上系统（SOC，System on Chip），即由单个芯片完成整个系统的主要逻辑功能；其次，它是可编程系统，具有灵活的设计方式，可裁减、可扩充、可升级，并具备软硬件在系统可编程的功能。

SOPC 结合了 SOC 和 FPGA 各自的优点，一般具备以下基本特征[2]：

1、至少包含一个以上的嵌入式处理器 IP Core；

2、具有小容量片内高速 RAM 资源；

3、丰富的 IP Core 资源可供灵活选择；

4、足够的片上可编程逻辑资源；

5、处理器调试接口和 FPGA 编程接口共用或并存；

6、可能包含部分可编程模拟电路；

7、单芯片、低功耗、微封装；

SOPC 设计技术实际上涵盖了嵌入式系统设计技术的全部内容，除了以处理器和实时多任务操作系统（RTOS）为中心的软件设计技术、以 PCB 和信号完整性分析为基础的高速电路设计技术以外，SOPC 还涉及目前已引起普遍关注的软硬件协同设计技术。由于 SOPC 的主要逻辑设计是在可编程逻辑器件内部进行，而 BGA 封装已被广泛应用在微封装领域中，传统的调试设备，如：逻辑分析仪和数字示波器，已很难进行直接测试分析，因此，必将对以仿真技术为基础的软硬件协同设计技术提出更高的要求。同时，新的调试技术也已不断涌现出来，如 Xilinx 公司的片内逻辑分析仪 Chip Scope ILA 就是一种价廉物美的片内实时调试工具；而

在应对复杂设计方面，诸如 Xilinx 公司的 System Generator for DSP 就是一个利用可编程硬件逻辑实现数字信号处理算法的强大辅助工具。

由于芯片设计的复杂性和产品面市时间对于保证终端市场的成功率至关重要，设计师不断寻求缩短设计周期的方法，以及更有效的设计方式。随着我们步入系统级芯片时代，利用 IP 内核和可编程逻辑进行设计复用显得日趋重要。IP 资源复用（IP Reuse）是指在集成电路设计过程中，通过继承、共享或购买所需的智力产权内核，然后再利用 EDA 工具进行设计、综合和验证，从而加速芯片设计过程，降低开发风险。IP Reuse 已逐渐成为现代集成电路设计的重要手段，在日新月异的各种应用需求面前，超大规模集成电路设计时代正步入一个 IP 整合的时代[3]。

1.3 设计复用和嵌入式微处理器 IP

可重用 IP 大量应用在 SOPC 的设计之中，基于应用需求、规范协议和行业标准的不同，IP Core 的内容也是千差万别的。除了购买使用现有的 IP 资源外，设计者可能还需要自己进行 IP 的设计，一般地，为了使 IP Core 易于访问和易于集成，并具有良好的复用性，其设计必须严格按照“设计复用方法学（Reuse Methodology）”的要求，按照一定的规范和准则进行设计[3]。

在 SOPC 中最核心的部分就是嵌入式微处理器 IP 的设计与使用。当前有多种为嵌入式系统而设计的微处理器 IP，如 ARM、Motorola 和 MIPS 等芯片公司设计的处理器系列。其中 ARM 公司设计的 ARM 系列 IP 核应用最为广泛，目前世界上大部分嵌入式系统芯片都采用 ARM 核，这些处理器主要为 ASIC 而设计的。为了实现 SOPC 的设计技术，FPGA 生产厂商也推出了基于 FPGA 的微处理器 IP 核，比较著名的有 Xlinx 公司的 MicroBlaze、PowerPC 和 Altera 公司的 Nios 等。

下面对两种较为著名的微处理器 IP core 进行简单的介绍。

1. NIOS

Nios 处理器是一个流水线通用 RISC 微处理器，采用 16 位指令集，16/32 位数据通道，5 级流水线技术，平均一个时钟周期处理一条指令，性能达 50MIPS。其基本特征如下[4]：

- ✧ 大容量窗口寄存器堆。最大可以实现 512 个内部通用寄存器，编译程序运用这些内部寄存器可以加快子程序的调用和局部变量的存取。
- ✧ 简单完备的指令集。32 位和 6 位的 Nios 系统都使用 16 位宽的指令，16 位宽的指令减少了代码长度和指令存储区的带宽。
- ✧ 强大的寻址模式。Nios 指令集包括装载和存储指令，这样，编译器就可以加快结构体和局部变量的访问速度。
- ✧ 可扩展性强。用户可以直接把定制逻辑集成入 Nios 算术逻辑单元（ALU）

内，并且自动生成包含 C 语言和汇编语言下访问定制指令硬件的宏指令的软件开发工具包（SDK）。

- ✧ 硬件辅助功能。用户可以利用硬件的优点去另创一些指令，从而提高系统的性能。它可以有 5 条用户定制指令，用户可以把复杂的工作用一条指令来完成。

2. MicroBlaze

MicroBlaze 是一个非常简化，但具有较高性能的软处理器内核，可以在性价比很高的 Spartan II(-E)系列 FPGA 上实现，系统时钟频率高可达 75MHZ，仅占用约 500 个 Slice 资源，稍多于 10 万门 FPGA 容量的三分之一。MicroBlaze 是一个专门为 Xilinx FPGA 优化的 RISC 嵌入式软处理器，符合 IBM Core Connect 标准，能够与 PPC405 系统无缝连接，它具备以下基本特征[5]：

- ✧ 32 个 32bit 通用寄存器
- ✧ 硬件乘法器(仅限 Virtex II 系列)
- ✧ 32bit 地址总线和 32bit 数据总线
- ✧ 三操作数 32bit 指令字，两种寻址模式
- ✧ 独立的片内程序 32bit 总线和数据总线
- ✧ 片内总线遵循 OPB(On-chip Peripheral Bus)标准
- ✧ 通过 LMB(Local Memory Bus)访问片内 Block RAM

1.4 课题所做的工作和论文的安排

课题在深入探索 FPGA 结构的基础上，详细阐述了 FPGA 的设计技术，对 FPGA 设计流程做了细致的研究，提出了一种符合现代数字系统设计需求、优化设计时序的 FPGA 设计流程。课题使用 IP 复用设计技术，突破传统电路设计方法在航天惯测系统中局限性，设计了一种基于 FPGA 的航天惯测系统。课题研究了如何应用微处理器 IP 在 FPGA 内构建嵌入式系统及其应用，在此基础上，设计了一种基于处理器 IP 的可重构数据采集系统。

第二章首先分析了现代 FPGA 的内部结构特点，然后对 FPGA 设计技术进行了详细阐述，最后针对现代数字系统的特点，研究分析了 FPGA 的设计流程，提出了一种优化设计时序的 FPGA 应用系统设计流程。

第三章针对传统电路设计技术在航天惯测系统中的局限性，引入了 IP 复用设计技术，研究了在 FPGA 内进行 IP 设计应遵循的原则和方法，并使用 FPGA 设计了一种航天惯测系统。

第四章对基于 FPGA 的微处理器 IP—MicroBlaze 的结构性能进行了分析，研究了 MicroBlaze 的开发和使用，并设计了一种基于 MicroBlaze 的 CPCI 的动态可

重构数据采集系统。

第二章 FPGA 的结构和应用设计技术

本章以 Xilinx Virtex-II 系列 FPGA 为例，深入分析了 FPGA 结构特点，在结构分析的基础上，详细阐述了 FPGA 设计技术，提出了一种符合现代数字系统设计、优化设计时序的 Xilinx FPGA 设计流程。

2.1 FPGA 的结构研究

作为FPGA的设计者，对FPGA内部结构的深入分析，有助于更好的发挥FPGA性能，提高 FPGA 使用效率，使自己的 FPGA 设计程序更适合于在 FPGA 上实现。本节以 Xilinx Virtex-II 系列 FPGA 为例，深入分析了 FPGA 结构。

Vitex-II 系列 FPGA 的内部结构如图 2.1 所示，它主要由可配置逻辑（CLB）、用户可编程 I/O（IOB）、BlockRAM、数字时钟管理模块（DCM）、数字阻抗匹配模块（DCI）和硬件乘法器等组成。其中 CLB 用于实现 FPGA 的绝大部分逻辑功能；IOB 用于提供封装管脚与内部逻辑之间的接口；BlockRAM 用于实现 FPGA 内部的随机存取，它可配置为 RAM、双口 RAM、FIFO 等随机存储器；DCM 用于提供灵活的时钟管理功能；硬件乘法器用于提高 FPGA 的数字信号处理能力[6]。

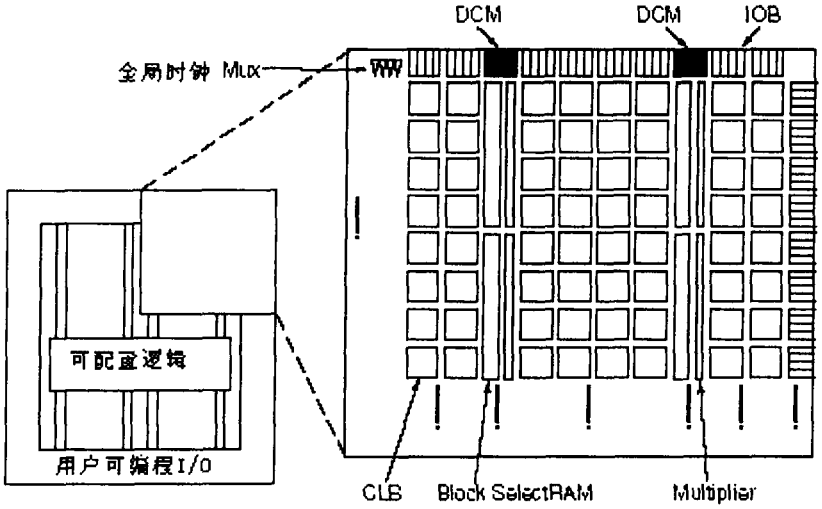

图 2.1 Virtex-II 系列 FPGA 内部结构示意图

2.1.1 CLB

在 Virtex-II 系列 FPGA 中，CLB 模块由 4 个相同的 Slice 和附加逻辑电路构成，用于实现组合逻辑和复杂时序逻辑。Slice 的结构如图 2.2 所示，每个 Slice 由两个 4 输入函数发生器、进位逻辑、算数逻辑、存储逻辑和函数复用选择器组成。算数逻辑包括 1 个异或门（XORG）、1 个专用与门（MULT_AND），异或门可以实

现两个 Slice 的 2bit 全加操作，专用与门可用于提高乘法器的效率。进位逻辑有专用进位信号和函数复用发生器（MUXC）组成，共同实现快速的算数加减法操作。

在 CLB 模块中：4 输入函数发生器可以用于实现 4 输入 LUT、分布式 RAM 或 16bit 移位寄存器；存储逻辑可配置为 D 触发器或锁存器；进位逻辑包括两条快速进位链，用于提高 CLB 模块的处理速度；算数逻辑包括一个异或门和一个用于加速乘法运算的专用与门[6]。

在 Virtex-II 系列 FPGA 中，每个 CLB 模块既可以配置为分布式 RAM，也可以配置为分布式 ROM，分布式 ROM 可以通过设置初始化数据来实现。表 2.1 给出了 CLB 中的 LUT 资源配置为分布式存储器的的对应关系。

Virtex-II 系列 FPGA 中的每个 CLB 模块包括多种类型的复用器（4 个 MUXF5、2 个 MUXF6、1 个 MUXF7 和 1 个 MUFX8）。通过使用这些复用器，每个 CLB 不仅可以实现 5 输入 LUT、6 输入 LUT、7 输入 LUT、8 输入 LUT 和 9 输入 LUT，还可以实现 128 位移位寄存器，从而提高了 Virtex-II 系列 FPGA 内部的资源利用率[6]。

表 2.1 Virtex-II 分布式 RAM 与 LUT 资源使用对照表

类型	容量	资源占用
单端口分布式 RAM	16×1bit	1 个 LUT
单端口分布式 ROM	16×1bit	1 个 LUT
双端口分布式 RAM	16×1bit	2 个 LUT
单端口分布式 RAM	32×1bit	2 个 LUT
单端口分布式 ROM	32×1bit	2 个 LUT
双端口分布式 RAM	32×1bit	4 个 LUT
单端口分布式 RAM	64×1bit	4 个 LUT
单端口分布式 ROM	64×1bit	4 个 LUT
双端口分布式 RAM	64×1bit	8 个 LUT
单端口分布式 RAM	128×1bit	8 个 LUT
单端口分布式 ROM	128×1bit	8 个 LUT
双端口分布式 RAM	128×1bit	16 个 LUT
单端口分布式 RAM	256×1bit	16 个 LUT
单端口分布式 ROM	256×1bit	16 个 LUT
双端口分布式 RAM	256×1bit	32 个 LUT

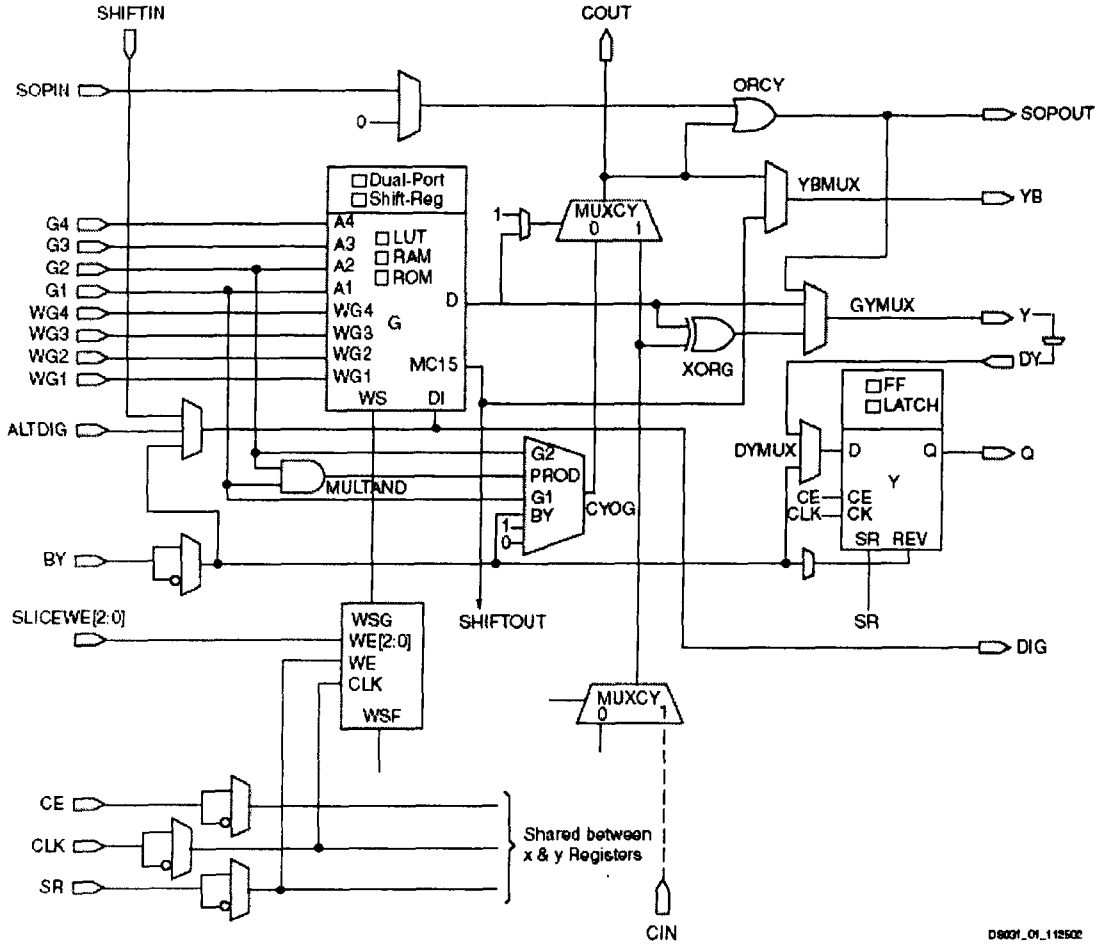

图 2.2 Virtex-II 系列 FPGA 中的 Slice 结构示意图

2.1.2 IOB

IOB 用于提供 FPGA 内部逻辑与外部封装管脚之间的接口。如图 2.3 所示，Virtex-II 系列 FPGA 的 IOB 含有 6 个触发器，它们不仅可以单独配置为 D 触发器或锁存器，还可以成对实现 DDR（Double-Date-Rate）输入和输出[6]。

外部信号既可以经过IOB模块的存储单元进入FPGA内部，也可以引用IOBUF直接输入到 FPGA 内部。当外部信号经过 IOB 模块的存储单元进入 FPGA 内部时，其保持时间的要求可以降低。通常，外部输入信号经过 IOB 模块的存储单元进入 FPGA 内部，其保持时间默认为 0。

在 Virtex-II 系列 FPGA 中，根据当前使用的 I/O 接口标准不同，需要设置不同的接口电压 VCCO 和参考电压 VREF。在 Virtex-II 系列 FPGA 中，高速差分信号得到了广泛的支持，表 2.2 给出了差分信号 I/O 接口所对应的接口电压 VCCO 和参考电压 VREF。I/O 管脚分布在 8 个 Bank 中，每个 Bank 的 VCCO 必须保持一致，

不同的 VCCO 允许有不同的 VCCO 电压。

在 Virtex-II 系列 FPGA 中，同一 Bank 中的 I/O 接口标准应保持兼容，但不同 Bank 之间的 I/O 接口标准不要求兼容。由于同一 Bank 中的 VCCO 必须保持一致，因此 VCCO 电压相同是输出接口标准兼容的必要条件。

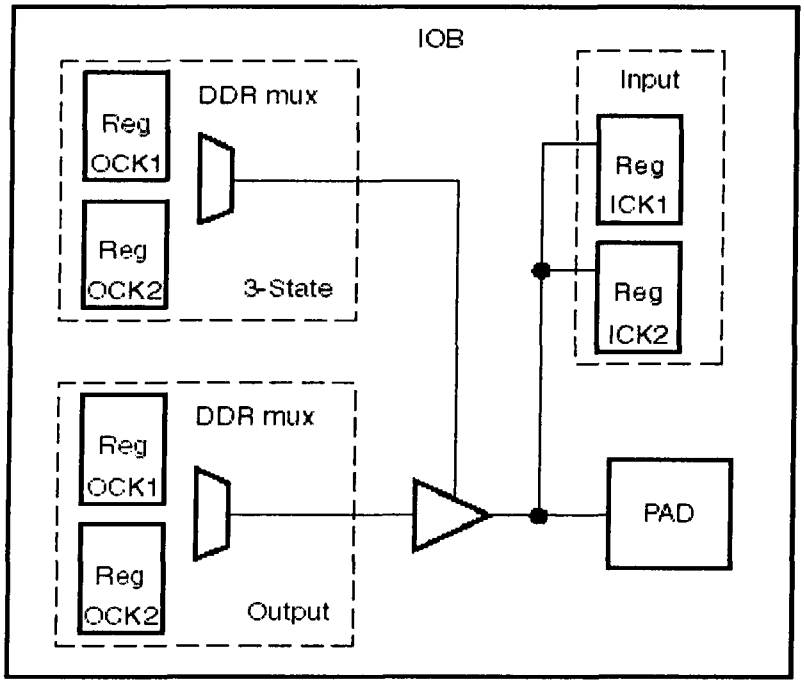

图 2.3 Virtex-II 系列 FPGA IOB 结构示意图

表 2.2 差分信号 I/O 接口标准电压要求

I/O Standard	Output V_{CCO}	Input V_{CCO}	Input V_{REF}	Output V_{OD}
LVPECL_33	3.3	N/R(1)	N/R	490 mV to 1.22V
LDT_25	2.5	N/R	N/R	0.430 - 0.670
LVDS_33	3.3	N/R	N/R	0.250 - 0.400
LVDS_25	2.5	N/R	N/R	0.250 - 0.400
LVDSEXT_33	3.3	N/R	N/R	0.330 - 0.700
LVDSEXT_25	2.5	N/R	N/R	0.330 - 0.700
BLVDS_25	2.5	N/R	N/R	0.250 - 0.450
ULVDS_25	2.5	N/R	N/R	0.430 - 0.670

2.1.3 Block RAM

在 Virtex-II 系列 FPGA 中，BlockRAM 的单位容量是 18Kbit。如图 2.4 和图 2.5 所示，每一个 BlockRAM 均可配置为单端口或双端口 RAM。利用 Core Generator，还可将 BlockRAM 资源配置为双端口 FIFO[6]。

值得注意的是，当 Block RAM 配置为双端口存储器时，其每一个端口的深度

和数据宽度可以是不同的，可选的双端口存储深度和数据宽度对应关系如表 2.3 所示。

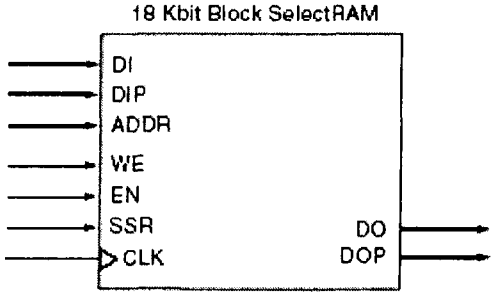

图 2.4 单端口 BlockRAM 示意图

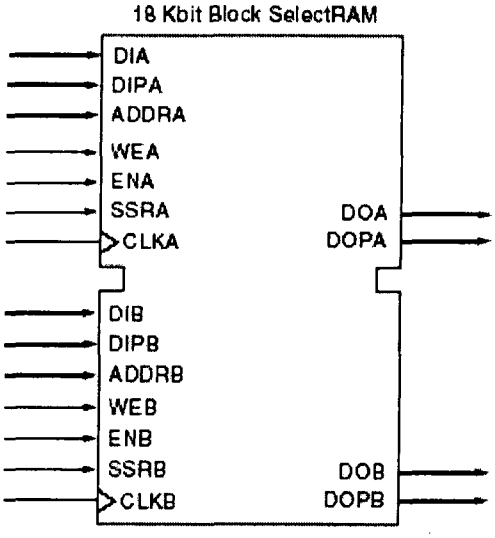

图 2.5 双端口 BlockRAM 示意图

表 2.3 双端口 RAM 每个端口存储深度和数据宽度对照表

Port A	16K x 1	16K x 1	16K x 1	16K x 1	16K x 1	16K x 1
Port B	16K x 1	8K x 2	4K x 4	2K x 9	1K x 18	512 x 36
Port A	8K x 2	8K x 2	8K x 2	8K x 2	8K x 2	
Port B	8K x 2	4K x 4	2K x 9	1K x 18	512 x 36	
Port A	4K x 4	4K x 4	4K x 4	4K x 4		
Port B	4K x 4	2K x 9	1K x 18	512 x 36		
Port A	2K x 9	2K x 9	2K x 9			
Port B	2K x 9	1K x 18	512 x 36			
Port A	1K x 18	1K x 18				
Port B	1K x 18	512 x 36				
Port A	512 x 36					
Port B	512 x 36					

2.1.4 DCM

为了适应大规模 FPGA 设计中多时钟、高速度、低延时的要求，Virtex-II 系列 FPGA 提供 16 个全局时钟缓冲器，支持 16 个全局时钟网络，每个象限最多可支持 8 个全局时钟网络。根据具体器件型号不同，Virtex-II 系列 FPGA 中包含 4～12 个 DCM，具体对应关系如表 2.4 所示。Virtex-II 系列 FPGA 中的 DCM 的标准原型如图 2.6 所示[6]。

表 2.4 Virtex-II 器件与 DCM 数量对照表

Device	Columns	DCMs
XC2V40	2	4
XC2V80	2	4
XC2V250	4	8
XC2V500	4	8
XC2V1000	4	8
XC2V1500	4	8
XC2V2000	4	8
XC2V3000	6	12
XC2V4000	6	12
XC2V6000	6	12
XC2V8000	6	12

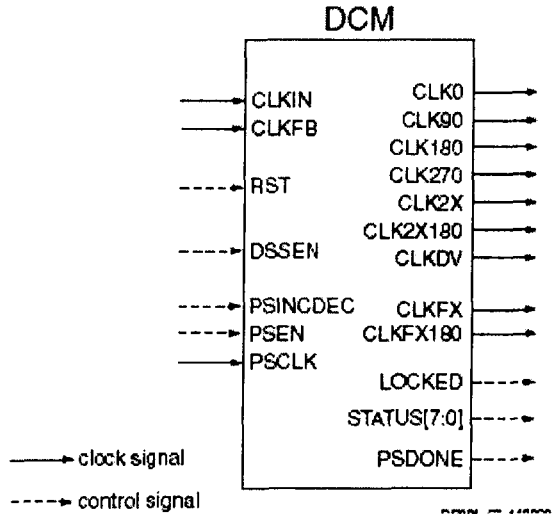

图 2.6 DCM 标准原型示意图

2.1.5 DCI

随着 FPGA 设计速度的不断提高，型号完整性问题越来越突出。为了保证高速信号的信号完整性，通常需要在 PCB 上进行阻抗匹配，以减小信号的反射和振荡。尽管大量的匹配电阻保证了信号的完整性，但也增加了 PCB 板的布线复杂度和成本。通过使用 DCI（数控阻抗匹配技术），可以在 Virtex-II 系列 FPGA 内部实现阻抗匹配，从而大幅度减少匹配电阻的数量，提高板级系统的稳定性，并降低设计复杂度和减少设计成本。

在 Virtex-II 系列 FPGA 中，可以使用 DCI 进行阻抗匹配的接口标准包括

LVTTL、LVCOMS、SSTL3-I、SSTL3-II、SSTL2-I、SSTL2-II、HSTL-I/II/III/IV、GTL 和 GTLP。

值得注意的是，在 Virtex-II 系列 FPGA 中，每个 I/O Bank 的 DCI 设置必须保持一致，但不同 Bank 的 DCI 设置可以不同。如图 2.7 所示，通过设置 VRN 和 VRP 管脚的外接电阻，可以实现不同阻值的阻抗匹配。

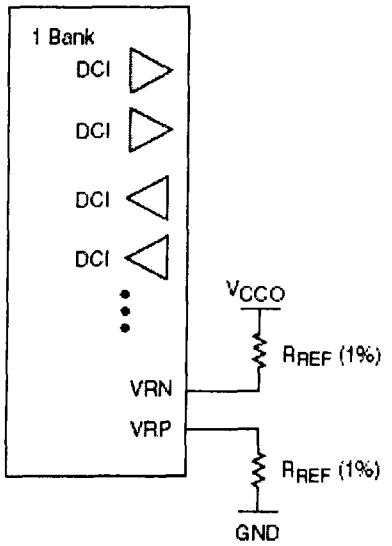

图 2.7 Virtex-II DCI 连接示意图

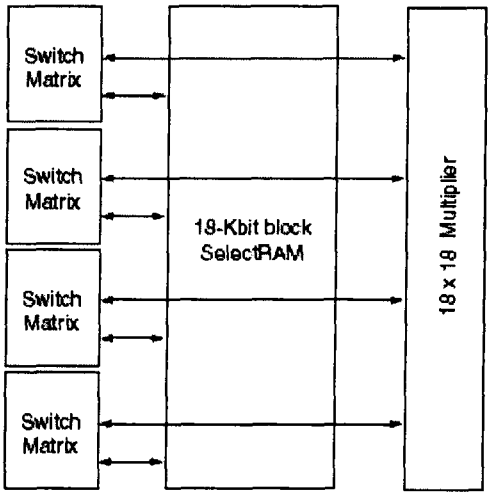

图 2.8 Virtex-II 乘法器结构示意图

2.1.6 乘法器

在 Virtex-II 系列 FPGA 中，乘法器模块支持 18×18bit 的有符号乘法。如图 2.8 所示。乘法器模块不仅可以通过交换矩阵（Switch Matrix）与 18Kbit 的 BlockRAM 配合使用，也可以单独使用。在 Virtex-II 系列 FPGA 中，乘法器模块的物理分布与 BlockRAM 的物理分布一致，利用这样的结构，读取、相乘、累加操作以及 DSP 滤波器结构设计都将变得异常快速和高效率[6]。

2.2 FPGA 应用设计技术

从上节对 FPGA 内部的结构分析来看，FPGA 的时序逻辑资源异常丰富，不同于 CPLD 等其他可编程逻辑器件；因此，对 FPGA 而言，必须有一套有效利用内部丰富时序逻辑资源的设计技术，而不同于以往的一般的 PLD 器件的设计技术。

FPGA 在开发阶段具有安全、方便、可随时修改设计等不可替代的优点，在电子系统中采用 FPGA 可以极大的提升硬件系统设计的灵活性，可靠性，以及提高硬件开发的速度和降低系统的成本。FPGA 的固有优点使其得到越来越广泛的应用，FPGA 设计技术也被越来越多的设计人员所掌握。实际中使用最佳的系统设计方法可以在很大程度上改善 FPGA 应用中所出现的问题，全面提高 FPGA 设计的

性能，达到性能和面积的最佳优化。

2.2.1 层次化设计技术

采用层次化设计技术的系统由一般由几个顶层模块组成，每个顶层模块又由几个小模块构成，以此类推，没有采用层次化结构的设计我们称为平板(plane)设计。层次化设计中的模块，可以是原理图描述的结构图，也可以是 HDL 语言描述的实体。

采用层次化设计对系统模块的划分非常重要，不合理的模块划分，导致系统设计的不合理，致使系统性能下降，这样的系统甚至比平板设计的系统效果更差。

一般的层次化模块划分是根据设计中不同的逻辑类型来划分，如图 2.9。

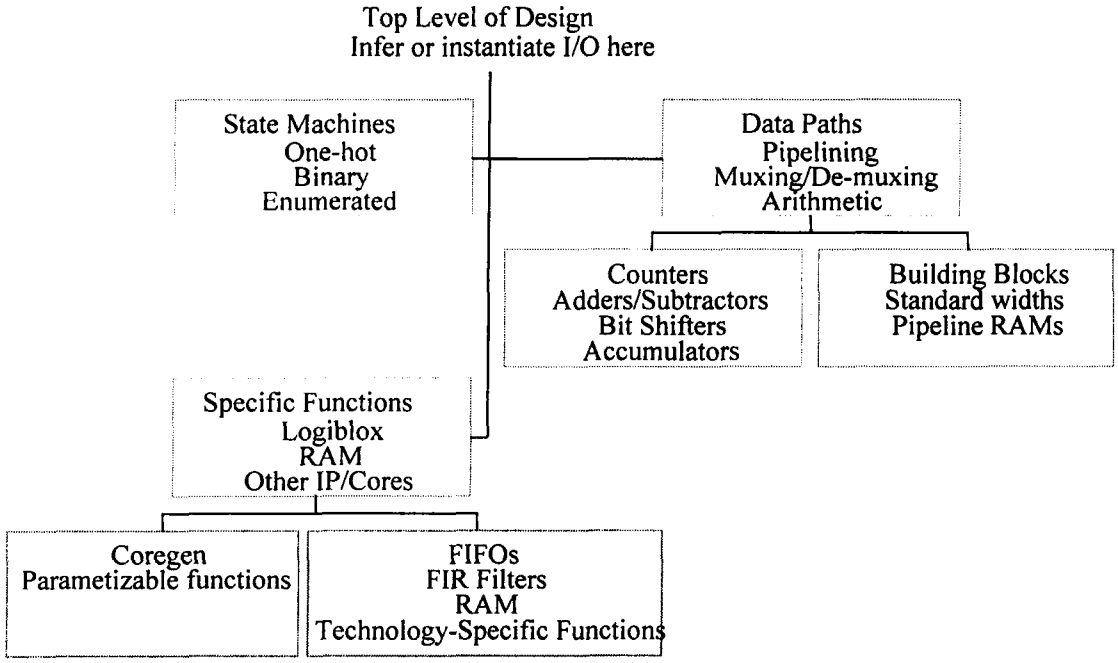

图 2.9 层次化模块划分

采用层次化结构设计的优点主要体现在两个方面：

增加设计的可读性

增加设计复用的可能性

2.2.2 同步设计技术

所有的时序电路都具有一个共同的性质－如果要使电路正确的工作就必须严格执行预先明确定义好的开关事件的顺序。如果不这样做，也许会把错误的数据写入存储元件，导致操作出错。同步系统设计方法，即采用全局分布的周期性同步信号（即全局时钟信号）使系统中的所有存储单元同时更新，是执行这一次序

的有效的非常普遍的方法。电路的功能性是通过对时钟信号的产生以及它们在遍布整个芯片的存储单元上的分布实行某些严格的限定来保证的，违背这些限定常常会使功能出错。时序电路设计的另一个极端是非同步（异步）设计方法，它由于不需要全局时钟因而完成避免了时钟的不确定性问题[7]。

异步信号可以在任何时候随意变化，并且它们不服从任何本地时钟。因此，把这些随意的变化映射到一个同步的数据流中并不容易。通过检测这些变化将等待时间引入与本地时钟同步的数据流中就可以产生异步信号。然而一个更加自然的处理异步信号的方法就是去掉本地时钟并采用自定时的异步设计方法，模块之间的通信由握手协议控制，由它来保证正确的操作次序。这种方法采用的握手协议增加了电路的复杂性以及通信开销，从而影响了性能。

对于静态同步设计，当满足以下两个条件时，我们说这个系统是同步的:

1. 每个边缘敏感部件的时钟输入是一次时钟输入的某个函数；并且仍是像一次时钟那样的时钟信号。

2. 所有存储元件(包括计数器) 都是边缘敏感的，在系统中没有电平敏感存储元件。

我们对 FPGA 的同步设计理解为所有的状态改变都由一个主时钟触发，而对具体的电路形式表现为所有的触发器的时钟端都接在同一个主时钟上。一个系统的功能模块在内部可以是局部异步的,但是在模块间必须是全局同步的。CPU 是一个同步设计的典型实例，就是所有电路都与一个系统主时钟同步，主时钟是系统的心脏，尽管在与慢速的外设传送数据时需要插入等待周期，但它的输入输出理论上仍然是主时钟同步驱动的。相比异步设计来说同步设计有许多的优点，但在 FPGA 中实现电路的同步设计需要考虑多个方面的因素。

首先是选取主时钟。数字电路中，时钟是整个电路最重要、最特殊的信号。首先，系统内大部分器件的动作都是在时钟的跳变沿上进行，这就要求时钟信号时延差要非常小，否则就可能造成时序逻辑状态出错；第二，时钟信号通常是频率最高的信号；第三，时钟信号通常是负载最重的信号，所以要合理分配负载。出于这样的考虑，在 FPGA 这类可编程器件内部一般都设有数量不等的专门用于系统时钟驱动的全局时钟网络。这类网络的特点是，一是负载能力特别强，任何一个全局时钟驱动线都可以驱动芯片内部的触发器；二是时延差特别小；三是时钟信号波形畸变小，工作可靠性好。因此，在 FPGA 设计中最好的时钟方案是：由专用的全局时钟输入引脚驱动单个主时钟去控制设计项目中的每一个触发器。同步设计时，全局时钟输入一般都接在器件的时钟端，否则会使其性能受到影响。对于需要多时钟的时序电路，最好选用一个频率是它们的时钟频率公倍数的高频主时钟[7]。

其次各个功能模块要使用统一的复位电路。在使用带时钟的触发器、计数器

等有复位端的库器件时，一般应尽量使用有同步复位的器件。注意复位时保证各个器件都能复位，以避免某些寄存器的初始状态不确定而引起系统工作不可靠。

第三是尽量在一般的时序电路采用同步电路。尽可能使用同步计数器、并行进位加法器、同步移位寄存器等同步器件，避免使用异步触发器和异步 RS 触发器；尽量减少或不使用逻辑门控制时钟信号，也不要用系统主时钟参与逻辑运算来产生控制信号,因为它会在时钟和数据之间产生较多的扭曲；避免用触发器的输出作为另一触发器的时钟；译码器的输出不能直接接在器件的时钟端和复位端；不要将异步数据接在同一时钟触发的多个触发器上；避免将异步数据接在同一时钟打入的寄存器上。在设计中电路的每一部工作时序都要留有足够的余量。

下面我们结合以上原则介绍一些实现同步设计的实例。首先我们介绍一下毛刺信号的产生。

1. 毛刺信号的产生

信号在 FPGA 器件内部通过连线和逻辑单元时，都有一定的延时。延时的大小与连线的长短和逻辑单元的数目有关，同时还受器件的制作工艺、工作电压、温度等条件的影响。信号的高低电平转换也许要一定的过渡时间。由于以上因素的影响，多路信号的电平变化时，在信号变化的瞬间，组合逻辑的输出状态不确定，往往会出现一些不正确的尖峰信号，这些尖峰信号被称为“毛刺”。许多逻辑电路产生的小的寄生信号，也能成为毛刺信号。这些无法预见的毛刺信号可通过设计来传播并产生不需要的时钟脉冲。应该明确的是，任何组合电路都可能是潜在的毛刺信号发生器，而时钟端口、清零和置位端口对毛刺信号十分敏感，任何一点毛刺都可能会使系统出错。

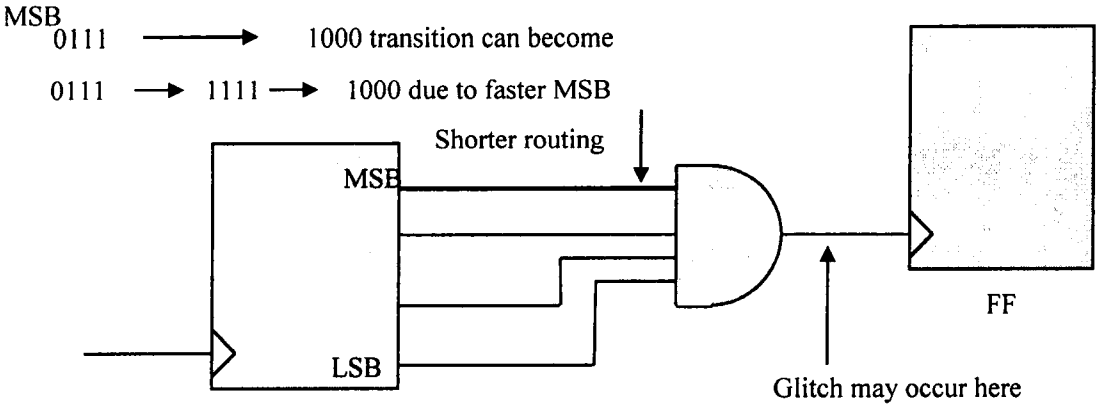

图 2.10 毛刺信号产生的误触发

如图 2.10 所示，当计数器从 0111 增加到 1000 时，由于数据最高位变化较快，数据的变化有个中间状态，即数据首先从 0111 变成 1111 然后变成 1000，在这样四输入与门的输出端产生了一个毛刺尖冲，造成触发器的误触发。

2. 对于一般组合逻辑的同步设计

对于 D 触发器来说，只要毛刺不出现在时钟的上升沿并且满足数据的建立和保持时间，就不会对系统造成危害，因此可以认为 D 的输入端对毛刺信号不敏感。基于以上认识，在输出信号的保持时间内，用 D 触发器来读取一般组合逻辑的输出信号就成为一种行之有效的方法。在实践中，可用工作于高速时钟的 D 触发器对组合逻辑的输出信号进行采样使其强行与时钟同步。这里要求系统时钟的频率相对很高，一个时钟的采样时延不会引起系统的错误。图 2.11 说明了这种方法。

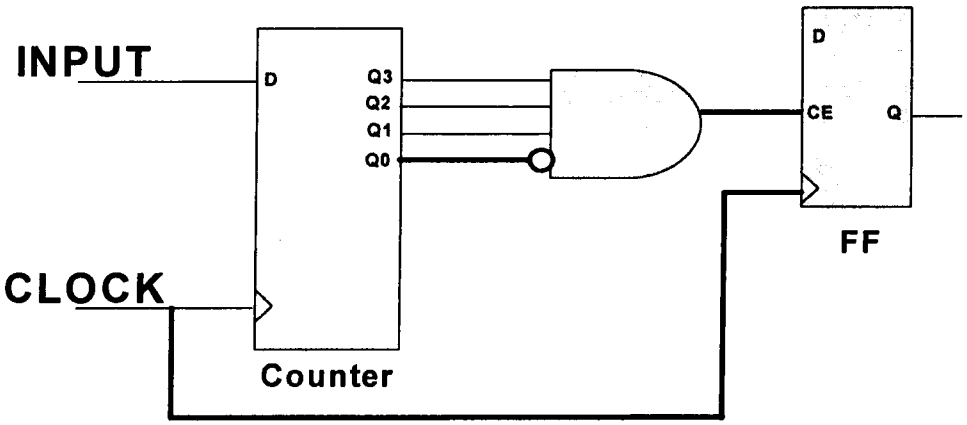

图 2.11 使用 D 触发器消除毛刺

另一种利用采样来消除毛刺实现同步的方法是用一定宽度的高电平脉冲与输出信号做逻辑与运算，由此获取输出信号的电平值。图 2.12 说明了这种方法。

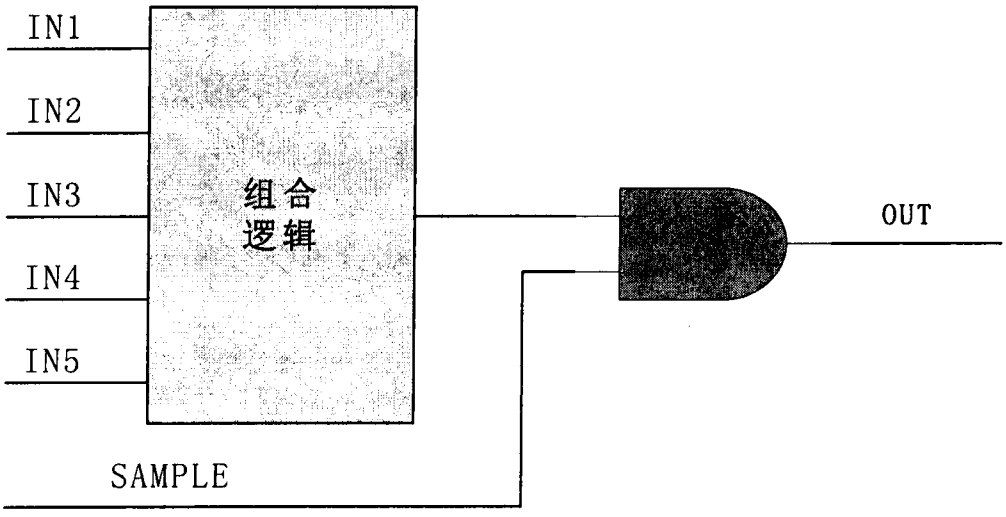

图 2.12 使用与门消除毛刺

3. 对于使能信号的合理使用

在 FPGA 同步设计时，合理使用使能信号可以得到很好的设计。对于不同源的多时钟输入处理问题，通过合理选用使能信号，我们可以将它转化为单时钟多路并行处理的问题。例如在处理 4 路一次群信号时，输入是 4 路带有帧定时信号的

不同源的一次群信号，时钟速率都是 2.048MHz。输入的每个一次群都包括数据、时钟和帧三个信号。由于输入的 4 个一次群是不同源的，存在着 4 个时钟信号。

在用 FPGA 处理时，用一个比输入时钟频率高数倍(32.768MHz) 的本地时钟 CLOCK 作为工作主时钟，对输入时钟 CLK0 通过如图 2.13 所示的数字单稳电路进行采样，产生使能信号 CE0。

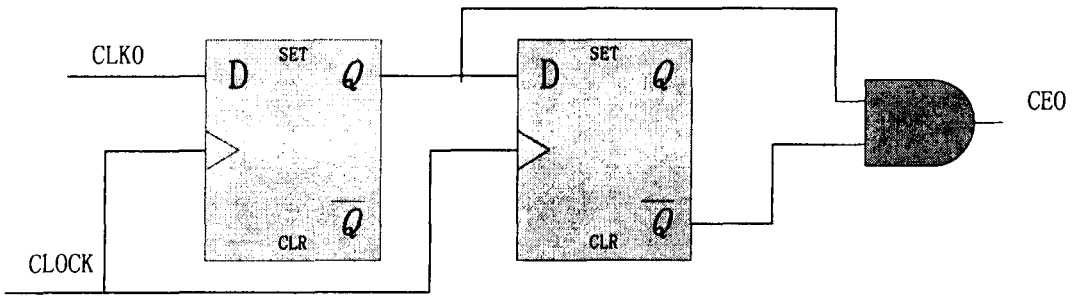

图 2.13 数字单稳态电路

用 AHDL 语言描述为:

```
SUBDES IGN CE_ GEN
 (
    CLOCK, CL K0: IN PU T;
    CE0: OU TPU T;
 )
VAR IABL E
    D1, D2 : DFF;
BEGIN
    D1. (CL K, D) = (CLOCK, CL K0) ;
    D2. (CL K, D) = (CLOCK, D1. Q ) ;
    CE0= D1. Q & (! D2. Q ) ;
END;
```

对输入时钟的每一个周期，单稳电路输出一个在输入时钟上升沿附近的、宽度为一个本地时钟周期的使能脉冲 CE0。在 FPGA 芯片内，所有的 D 触发器都可以定义为有一个使能端，同样，由 D 触发器构成的计数器和锁存器等时序电路单元也都有使能端。本地时钟在由该输入时钟产生的使能信号的控制下，对该数据进行处理，这样就可以实现用高速率的时钟在低速率的使能信号控制下，对低速率的信号进行处理。在 FPGA 内，将输入的 4 个一次群的时钟通过 4 个数字单稳电路，产生 4 个使能信号,在同一本地时钟的驱动下，按 4 路并行输入数据来处理。

采用这种设计思路带来的好处主要是系统工作稳定，布线转换成功率高，设备调试周期短。其不足之处是系统工作频率增高，资源使用较多，电路设计更复杂。但对 FPGA 实现来说，工作频率可以工作在较高的频率上；资源使用虽然较多，

但采用统一全局时钟驱动后，系统的资源利用率也可以提高，代价并不大。而且，由于使用了高倍时钟作为工作主时钟，可以对一些模块进行时分复用。我们在WCD2MA 移动台接收的 FPGA 实现过程中，采用这种思想设计实现了一种时分数字匹配滤波器，它可以对匹配滤波器进行时分复用，同时完成对多个信道的匹配相关运算，大大节省了硬件资源。实际上这种方法代价最大的是设计复杂性的增加，对于设计人员来讲，要非常清楚各个功能模块的控制时序关系，需要一定的设计技巧。

由此可见，在进行 FPGA 同步设计时，用逻辑门产生使能信号来控制数据而不是用时钟来控制是一种较好的设计。

从上面的一些实例可以看出，在 FPGA 设计过程中，如何合理的使用时钟是需要最优先考虑的，许多设计中出现的问题都与使用时钟不合理有关。当然复位等其他全局信号的使用也很重要，另外在电路每一步工作时序上都要留有足够的余量。

2.3 一种优化设计时序的 FPGA 应用设计流程

本章第一节对 FPGA 结构的研究可知，在 FPGA 内部时序逻辑资源十分丰富，不同于 CPLD 等其他以组合逻辑资源为主的可编程逻辑器件；相应的 FPGA 的设计技术不同于其他的可编程逻辑器件，采用层次化设计技术和同步设计技术能够更好的发挥 FPGA 器件的潜能，提高 FPGA 应用系统的性能。但是使用同步设计技术时，如果按照一般的 FPGA 设计流程，即编写 FPGA 设计代码→综合→翻译→布局→布线→下载，如图 2.14 所示[8]。

图 2.14 一般的 FPGA 设计流程

对于一些普通、低速的同步逻辑设计来说基本可达到和满足设计要求，但对于高速、高性能、高密度及多时钟信号控制的同步逻辑设计，是远远不够的，同时也不能充分发挥 FPGA 结构的优点，造成对 FPGA 器件资源的浪费。本节根据 FPGA 的结构功能特点和先进的 FPGA 设计技术，提出一种优化时序的 FPGA 设计流程。

2.3.1 方法概述

与其它的设计流程不同，优化时序的 FPGA 设计流程，主要在 FPGA 的同步设计过程中，加入时序约束，提高 FPGA 设计的性能，达到设计速度要求。

图 2.15 优化设计时序的 FPGA 设计流程

如图 2.15 所示，在 FPGA 的设计实现过程中，编写代码的同时，给 FPGA 设

计中的同步设计添加时序约束，综合、布局、布线后，查看设计软件的布线后静态时序分析报告，检查 FPGA 的设计是否满足时序约束中所定义的时序条件，如果不满足，首先检查设计是否合理，若不合理，重新编写设计代码；如果设计合理，那么提高布局布线软件的努力级别（Effort Level），重新布局布线，再次查看设计软件的布线后静态时序分析报告，检查 FPGA 的设计是否满足时序约束中所定义的时序条件，若还不满足，则采用多周期和伪路径约束，重新布局布线，如此反复，直到设计满足时序约束要求，采用的方法包括，添加特定路径约束，使用多重布局布线，使用底层设计工具调整布线，甚至重新编写设计代码，最终达到设计的时序约束。

由此可见设计过程中的时序约束的添加，显得非常重要，约束既不能太严，以致超过 FPGA 结构本身的速度极限，也不能太松，以致达不到提高设计性能的目的。

2.3.2 时序约束

随着系统工作速度的不断提高，对逻辑器件的工作频率和处理速度的要求也越来越高。器件处理速度的提高，意味着逻辑设计，对时钟信号的要求和处理变得更严格，需要通过附加约束来控制逻辑的综合、映射、布局、布线，以减少逻辑和布线的传输延时，从而提高工作频率。

随着晶体管尺寸的降低，门的速度越来越快，限制电路性能提高的主要因素不再是开关速度，而是互连延迟。时钟频率越高，互连线作为扇出负载引起的延迟在整个时序预算中所占的比例越大。深亚微米设计范围中互连延迟占总延迟的60%～70%，因此精确地计算这部分延迟在芯片设计中是十分重要的。时序约束主要用来控制路径端点间的延时路径。路径端点可以是引脚、寄存器、锁存器和存储器。附加的时序约束不可能以任何方式优化设计和改变设计网表，只能够改善设计的布局和布线[7]。

时序约束主要包括周期约束和偏移约束以及路径约束，如图 2.16 所示；除了图中所示的四种约束以外，对于多时钟域的复杂设计，需要对时钟域内的同步元件进行分组约束。对一些特定的路径，需要专门的约束，通过附加约束条件，可以使综合布线工具调整映射和布局算法，以使设计达到时序要求。在进行时序约束时，一般分为两个步骤：1、确定需要约束的路径端点；2、定义延时的长度。约束的添加可以用文本编辑的方式描写 UCF 文件，也可以使用专门的约束编辑器生产约束文件。

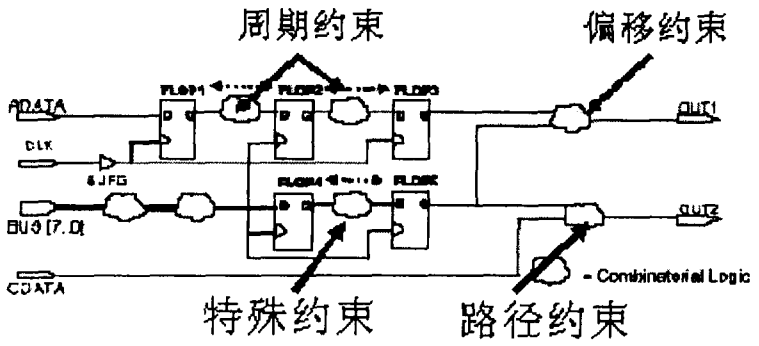

图 2.16 约束的分类

周期（PERIOD）约束是一个基本时序和综合约束，它附加在时钟网线上。时序分析工具根据 PERIOD 约束检查时钟域内所有同步元件的时序是否满足要求；同时 PERIOD 约束会自动处理寄存器时钟端的反相问题，即如果相邻同步元件时钟相位相反，那么它们之间的延时将被默认限制为 PERIOD 约束值的一半。周期约束的作用范围如图 2.16 所示。通常，逻辑器件内部所能达到的最高运行频率，取决于同步元件本身的建立保持时间及同步元件的逻辑和布线延时。

偏移（OFFSET）约束可以为综合、实现工具指出输入数据到达的时刻，或者输出数据的稳定时刻。从而在综合、实现过程中调整布局布线过程，使正在开发的 FPGA 输入建立时间以及下一级输入的建立时间满足要求。偏移约束规定了外部时钟和数据输入输出引脚之间的时序关系，只用于与 PAD 相连的信号，不能用于内部信号，如图 2.16 所示[8]。

2.3.4 应用实例

下面以 Xilinx 公司的 8 位微处理器 PicoBlaze 在 FPGA 上的实现为实例来说明优化时序的 FPGA 设计流程在实际设计中的应用，以及它对提高 FPGA 设计性能的作用。

PicoBlaze 8 位微处理器是 Xilinx 公司为 Virtex™ 系列 FPGA、Spartan®-II 系列 FPGA 和 CoolRunner-II 系列 CPLD 器件设计的嵌入式专用微处理器模块，解决了常量编码可编程状态机 constant (K) Coded Programmable State Machine (KCPSM) 的问题[9]。

PicoBlaze 处理器模块，可适用于进行数据的处理，在复杂性高，而对于实时性要求不高的状态机的一些应用中，其表现更为出色。PicoBlaze 示意图如图 2.17 所示。

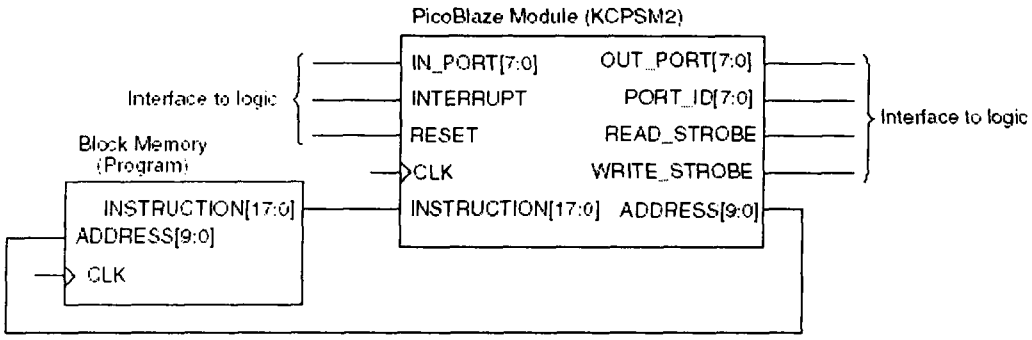

图 2.17 PicoBlaze 模块示意图

应用 ISE6.2 设计工具，使用 Spartan®-II 系列 XC2S300E 芯片，新建一个工程，在工程中添加 PicoBlaze 设计文件 KCPSM.vhdl，如图 2.18 所示。

图 2.18 在工程中添加 PicoBlaze 设计文件

双击操作栏下面的 Create Time Constraints，打开时序编辑器，给系统时钟添加周期约束如下：

NET "clk" TNM_NET = "clk";

TIMESPEC "TS_clk" = PERIOD "clk" 50 MHz HIGH 50 %;

综合布线后，查看布局布线后静态时序报告，报告如下：

Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 20 nS HIGH 50.000000 % ;

676 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)

Minimum period is 9.826ns.

由此得出，在 50M 系统时钟周期约束下，PicoBlaze 设计程序能运行在 9.826 微秒周期的时钟下，也就是 110MHz。

将周期约束改为：

NET "clk" TNM_NET = "clk";

TIMESPEC "TS_clk" = PERIOD "clk" 110 MHz HIGH 50 %;

综合布线后，查看布局布线后静态时序报告，报告如下：

Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 9.091 nS HIGH

50.000000 % ;

676 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)

Minimum period is 8.448ns.

那么现在 PicoBlaze 能够运行在 118M 时钟下；

若将周期约束更改为：

NET "clk" TNM_NET = "clk";

TIMESPEC "TS_clk" = PERIOD "clk" 120 MHz HIGH 50 %;

得出静态时序报告如下：

Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 8.333 nS HIGH 50.000000 % ;

676 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)

Minimum period is 8.144ns.

PicoBlaze 的性能提高到了 122MHz。

同样将周期约束更改为：

NET "clk" TNM_NET = "clk";

TIMESPEC "TS_clk" = PERIOD "clk" 120 MHz HIGH 50 %;

得出静态时序报告如下：

Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 7.692 nS HIGH 50.000000 % ;

676 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)

Minimum period is 7.644ns.

这样采用时序约束的方法，将 PicoBlaze 处理器的系统主时钟频率提高到了 130MHz，按照 PicoBlaze 每两个时钟周期运行一条指令的结构特点，PicoBlaze 处理器的性能提高到了每秒 65 百万条指令（65MIPS）。

2.4 本章小结

本章首先以 Xilinx Virtex-II 系列 FPGA 为例，详细分析了 FPGA 内部结构特点，接着针对 FPGA 的结构特点，阐述了 FPGA 的设计技术，在此基础上，提出了一种优化设计时序的 FPGA 设计流程，并且以一个实例论证了这种 FPGA 设计流程，对 FPGA 系统性能的提高有重要的促进作用。

第三章　基于 FPGA 的 IP 复用技术及其应用

在航天惯测系统设计中，总系统留给电路系统的空间很小，同时对电路系统的总重量有十分苛刻的要求，而且为满足实际需求，电路系统升级换代速度很快；传统的以 ASIC 为核心的电路设计技术，在惯测系统应用中，有很大的局限性。

本章针对传统电路设计的局限性，引入了 IP 复用设计技术，缩小电路空间，减少系统重量，用可复用的特点以提高系统的可升级能力，设计了一种基于 IP 复用技术的航天惯测系统。

3.1　传统电路设计技术的局限性

在集成电路（IC）发展初期，电路都从器件的物理版图设计入手，后来出现了 IC 单元库（Cell-Lib），使用 IC 设计从器件级进入到逻辑级，这样的设计思路使大批电路和逻辑设计师可以直接参与 IC 设计，极大的推动了 IC 产业的发展。但 IC 不是最终产品，它只有被装入整机系统才能发挥它的作用。IC 是通过印制电路板（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 之间的连线延迟，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 IC 设计技术已经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不断提高，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。目前已经可以在一个芯片上集成 108-109 个晶体管，而且随着集成电路制造技术的发展，21 世纪的微电子技术将从目前的 3G（G=109）时代逐步发展到 3T（T=1012）时代，即存储容量由 G 位发展到 T 位，IC 器件的速度由 GHz 发展到 THz，数据传输速率由 Gbps 到 Tbps。

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 IC 芯片上的芯片系统（System On Chip，简称 SOC）概念。

所谓片上系统就是指在单个硅芯片上实现信号采集、转换、存储、处理和 I/O 等功能，或者说在单个硅芯片上集成了数字电路、模拟电路、信号采集和转换电路、存储器、微处理器 MPU.微控制器 MCU、数字信号处理器 DSP 等，实现一个系统的功能。片上系统能够在单个硅芯片上实现高层次的系统集成，但同时也对硅芯片的设计提出了巨大的挑战。因为当前的设计工具和 IC 设计方法不能完全胜任片上系统的设计。在芯片设计复杂度迅速增加的同时，熟练设计人员的增长很有限，而对设计周期的要求越来越高[10]。

传统设计技术，需要对不同系统中相同功能的模块进行重复设计，片上系统设计效率不高，设计周期过长。传统设计技术对设计人员要求很高，系统越来越复杂，设计一个系统需要的设计人员，越来越多，而熟练设计人员的增长有限。传统设计技术延迟了产品上市周期，对复杂的系统成本越来越高。传统设计技术已经越来越不能满足系统复杂程度的提高和产品上市时间限制的要求。

3.2 IP 复用技术及其优势

设计复用——就是使用预先设计和预先验证的核——是当前 SoC 和 SOPC 设计的基础。如果以一个一定规模的开发小组，在限定的时间内，以较小的代价，开发出质量较高的超大规模的芯片，这也是唯一正确的方法。对于设计者来说，挑战不是来自于是否采用复用技术，而是如何很好的使用这个技术。在目前的 SoC 芯片设计中，设计流程正在进行两个方面的主要变化[10]：

从瀑布式模型向螺旋式模型转变；

从“自顶向下（top-down）”设计方法向“自顶向下（top-down）” 与“自底向上（bottom-up）”综合的设计方法转变。

3.2.1 瀑布式模型与螺旋式模型

瀑布式模型的设计方法，在项目开发中，从一个阶段到下一个阶段是按照顺序进行的，两个阶段之间没有信息的交互。其开发流程如图 3.1 所示。以开发图形芯片为例，首先由图形学专家开发出算法，然后交给设计小组开发 RTL 级模型；通过功能验证后，再由设计小组或另外的小组进行综合，映射到门级网表，进行时序仿真；满足时序要求后，由物理设计小组对设计进行布局和布线；最后，设计出原型芯片，交由软件小组进行软件调试。

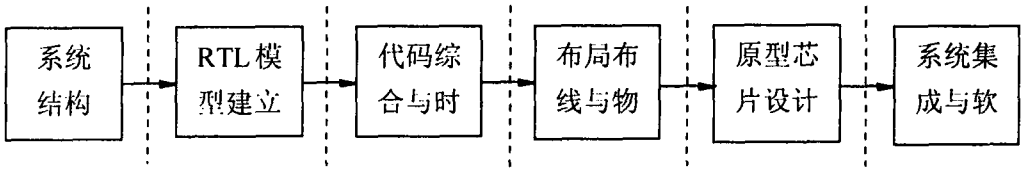

图 3.1 瀑布式模型的设计流程

由于这种设计方法顺序性比较强，前一步骤完成之前，后一步骤就无法进行。虽然软件设计在硬件设计开始之后就可以进行，但只有当样片做出来之后，才可以进行软件调试，软件和硬件设计也是按顺序进行的。

这种设计流程在进行 10 万门以下芯片设计是比较有效的。在进行规模更大的深亚微米芯片设计时，瀑布型设计流程就不适用了。大型系统设计要求硬件和软件设计要同步进行，以确保系统功能的正确性。物理层设计要在设计的初期就要

加以考虑，以确保设计能达到预计的性能。另外随着复杂度的增加，芯片尺寸的减小，以及市场竞争的压力，芯片设计人员更多的采用“螺旋式开发模型”的新方法。这种模式，要求设计人员在多个方面同时进行设计，同步向前进展。其主要特征有：硬件和软件并行、协同开展；模块的综合和验证并行进行；布局和布线在综合的过程中进行；有计划的反复贯穿于整个设计过程[11]。

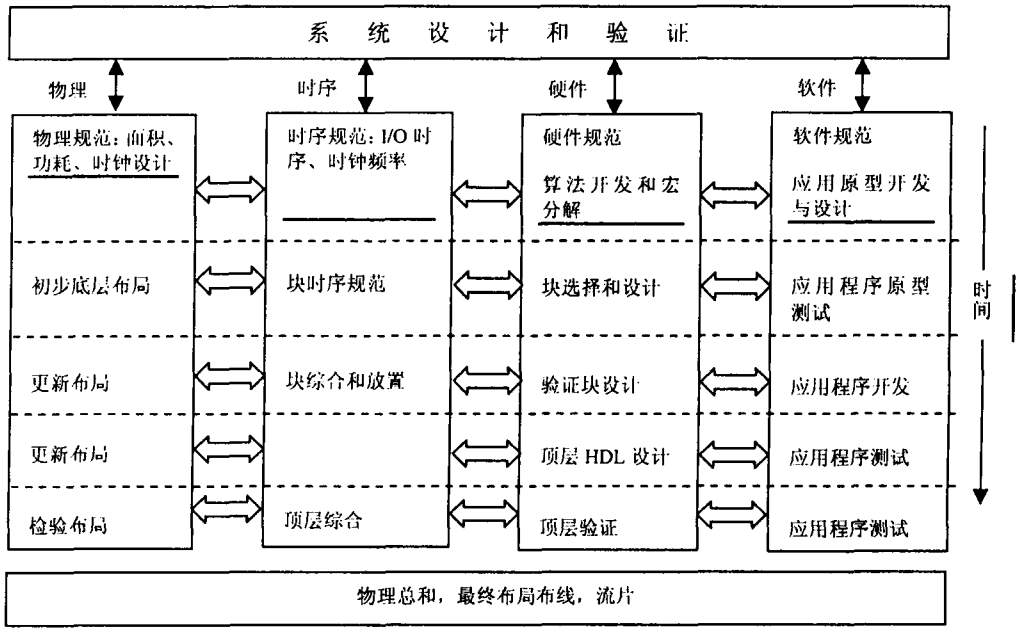

图 3.2 螺旋式模型设计流程

3.2.2 自顶向下设计和自底向上设计

经典的自顶向下设计方法是一个渐进的过程，从设计方案开始，逐步分解细化，直到最终完成系统集成和验证。但象瀑布模型的系统开发一样，自顶向下设计方法也是一个理想化的思路，因为它要假定定义的底层模块能够设计实现，如果无法实现，设计方案就要改变，设计流程就要反复。因此，开发人员往往混合使用“自顶向下”和“自底向上”的设计方法。在精确定义系统和模块时，预先设计具有严格时序特性的底层模块，可以为其提供参考。对可重复使用的模块预先设计和验证，有助于提高系统设计的效率[11]。

3.2.3 设计复用是 SOC 的要求

基于 SoC 概念的设计复杂性，为设计者带来巨大的挑战。市场的压力对设计的时效性要求越来越严，而芯片的设计是否成功关键要看芯片的性能、面积和功率等方面是否能达到严格的要求。深亚微米技术的采用和芯片复杂度的提高，使得时序收敛和验证更加困难。具有不同方面设计专长的设计小组往往比较分散，难以技术共享；而由于设计方法、工具和规则的改变，使得以前的设计不能得到

继承和重复使用。个别的设计小组采用了基于模块化的设计，重复使用一些经过预先设计和验证过的模块。但是如果复用的模块没有经过为复用而做的简明化设计，即使将其集成到设计之中也很难提高系统的性能。因此一个高性能的可复用的模块，必须设计简明，易于集成。SoC 面临的挑战需要设计复用技术， 遵循严格的标准是设计复用技术的关键。

3.3 IP 设计的基本准则

具有良好设计的 IP 是成功进行 SoC 设计的关键，因此 IP 设计要遵循一定的设计准则。其中最基本的几个前提如下：

规范——严格按照规范设计，这样的系统具有可升级性，可继承性，易于系统集成；

简单——设计越简单，就越容易分析、验证，达到时序收敛；

局部——时序和验证中的问题局部化，就容易发现和解决问题，减少开发时间，提高质量。

只有按照一定的编码规则编写的 HDL 代码才具有较好的可读性，易于修改并且具有较强的可复用性，同时也可获得较高的综合性能和仿真效果。下面简要的介绍一些编码的基本方法与规则。

3.3.1 基本的编码习惯

常用命名约定。一般要为设计制订一个命名约定，并在整个设计中要统一。一般原则是：信号、变量和端口名用小写字母，常量和自定义类型用大写字母；用有意义的词进行命名；时钟信号和复位信号命名要一致；低有效信号要用下划线加小写字母作标志；多位总线要用一致的位顺序；信号与端口相连时最要用相同的名称等[11]。

在每个源代码文件头部要加上一个文件头，里面有一些注册信息，包括：版权信息、文件名、作者、模块的功能描述和主要特征、创建时间、修改历史纪录（包括修改时间、修改任何修改内容）。

适当的使用注释，解释进程、函数和子类型声明的功能；对端口、信号和变量进行注释；注释要在被注释段之前，用独立的注释行。

不要使用 HDL 保留字为代码中的元素命名；不要在 Verilog 代码中使用 VHDL 保留字，也不要在 VHDL 代码中使用 Verilog 保留字。

按照一定的逻辑顺序声明端口，并在整个设计中保持一致；使用完整的端口和属性映射。

尽可能使用函数；使用循环和数组提高代码的可读性。

对进程和模块使用有意义的标识进行命名。

3.3.2 可移植性编码

代码的编写必须具有很好的可移植性，只使用 IEEE 标准库中的数据类型，与采用的技术无关；兼容各种仿真工具；并且可以很容易的在 VHDL 和 Verilog 两种语言之间转化。

时钟和复位信号结构要尽量简单、易懂，尽量使用单一的全局时钟和时钟的单个边沿触发控制。

3.3.3 可综合的编码

可综合的编码还要满足可测性、高性能、简单的静态时序特性，门级电路性能要与 RTL 级编码相匹配。如避免使用电平锁存器；避免组合反馈；列出所有的敏感信号；用 case 语句代替嵌套的 if-else 语句等。

对编码进行良好的综合隔离，获得的综合效果好，编译快速，可以用简单的综合策略获得时序特性。主要采用的编码措施有：对每个子模块的输出加上寄存器；将相关的组合逻辑放在同一个模块中；将不同设计目的的模块分离；避免异步逻辑；避免多时钟周期路径；不要在顶层例化门级电路逻辑等。

严格按照编码规则进行代码设计编写，可读性、可移植性强，查找修改错误快速，综合性能较好，可以起到事半功倍的效果，为进行 IP 设计打下好的基础。

3.4 一种基于 IP 复用技术的航天惯测系统设计

航天惯测系统要求对惯组导航陀螺的温度进行测试，对陀螺的测向脉冲进行计数，同时通过 UART 将监测到的陀螺信息，发送给 DSP 处理。

本系统设计中，在 FPGA 内部用 IP 复用设计技术完成对陀螺参数的测量，同时在 FPGA 设计实现 UART 传输核，实现数据的传输。

3.4.1 系统结构

系统结构框图如图 3.3 所示。

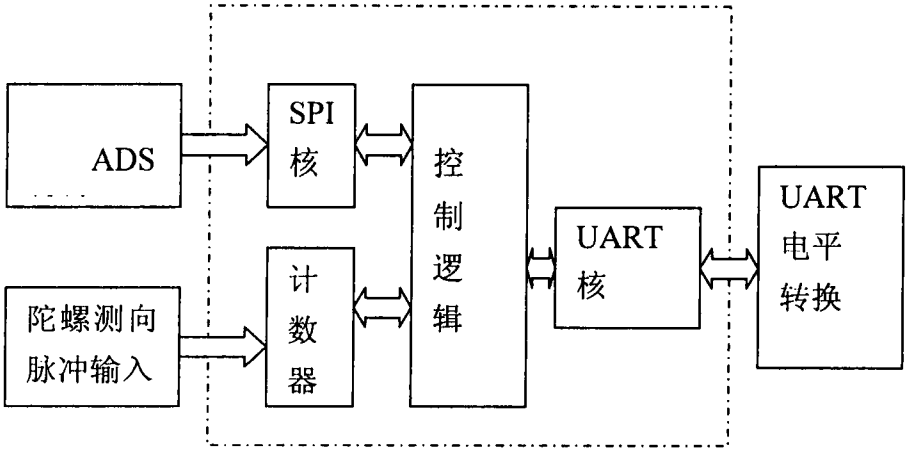

图 3.3 惯测系统结构

虚线框内为 FPGA 内部设计，ADS1210 为 SPI 结构的 AD 芯片，陀螺的温度信息转化为电压信号，输入到 ADS1210 的模拟输入端，SPI 核完成对 ADS1210 的控制将温度信息从 ADS1210 读出来；计数器对陀螺的测向脉冲进行计数；SPI 和计数器的数据通过 UART 核将信息发送给 DSP 处理。计数器设计比较简单，下面将对 SPI 和 UART 核的设计进行详细描述。

3.4.2 SPI 核设计

1、设计方案

SPI(Serial Peripheral Interface--串行外设接口)总线系统是一种同步串行外设接口，它可以使处理器与各种外围设备以串行方式进行通信以交换信息。 SPI 总线系统可直接与各个厂家生产的多种标准外围器件直接接口，SPI 接口一般使用 4 根线：串行时钟（SPICLK）、主机输入/从机输出数据线 SPIIN、主机

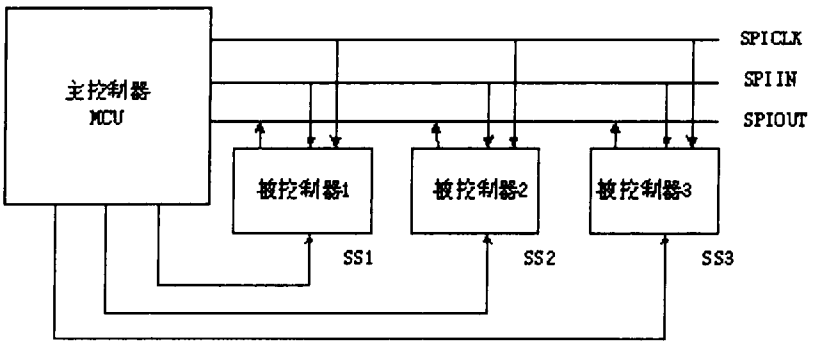

图 3.4 SPI 总线的组成

输出/从机输入数据线 SPIOUT 和低电平有效的从机选择线 SS。由于 SPI 系统总线一共只需 3～4 位数据线和控制即可实现与具有 SPI 总线接口功能的各种 I/O

器件进行接口，而扩展并行总线则需要 8 根数据线、8～16 位地址线、2～3 根控制线，因此，采用 SPI 总线接口可以简化电路设计，节省很多常规电路中的接口器件和 I/O 口线，提高设计的可靠性[12]。

在本设计中，我们用 IP 核的设计理念，把 PicoBlaze 8 位处理器作为核心，设计了一个灵活可编程的 SPI 软核，根据不同的要求适当修改 PicoBlaze 程序，可以将软核应用在不同的场合，同时可以增加应用系统接口器件的种类，提高应用系统的性能。

2、SPI 通信时序

图 3.5 是 SPI 的通信时序，SPI 工作在“主机－无延时上升沿“模式下，数据发送是在 SPICLK 信号的上升沿，数据接受是在 SPICLK 的下降沿。

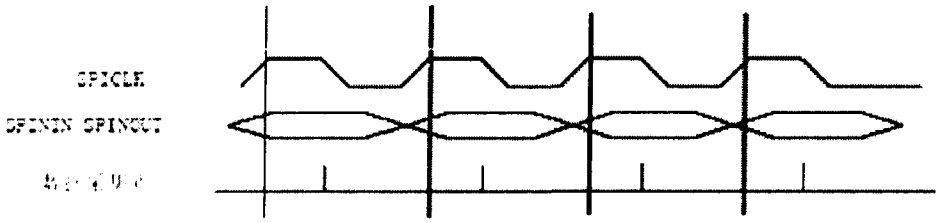

图 3.5 SPI 通信时序

3、SPI 通信实现方案

根据 SPI 的时序和 PicoBlaze 处理器输入输出端口的特性，我们设计 SPI 核的原理框图如图 3.6 所示；PicoBlaze 外面添加三个 D 触发器，触发器的输出分别作为 SPI 总线的 SPICLK、SPIIN 和 SPIOUT 信号，PicoBlaze 的读写信号和地址译码信号相与作为触发器的 CLK 输入，对 ID_PORT 输出的地址信号译码，使不同的触发器时钟端有效，用程序控制读或是写来模拟 SPI 的时序。

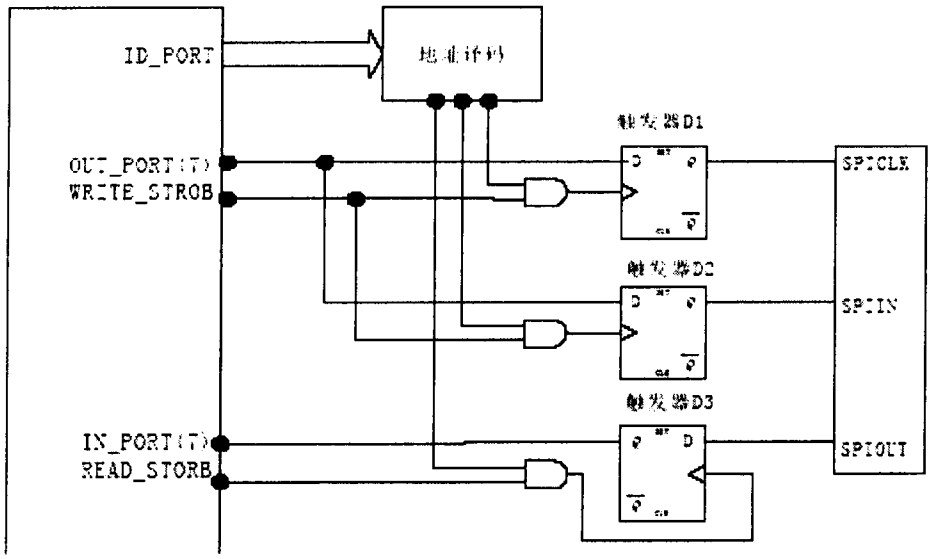

图 3.6 SPI 硬件框图

比如对触发器 D 1 重复写 1 和 0，就可以模拟 SPI 的时钟，因为 D1 的输入端 D 连接 OUTPORT(7)，所以，把寄存器 sX 的第 7 位 sX（7）置 1 或置 0，pp 写位 D1 的地址，使用汇编语言 OUTPUT sX,pp，就完成了对 SPICLK 的置 1 或置 0，同理对 D 2 写 1 和 0 就是模拟数据位的 1 和 0，从 D 3 读到的数据就是 SPIIN 上的数据。

4、模拟 SPI 读写的子程序

针对上节中描述下原理框图，编写 PicoBlaze 程序，控制 PicoBlaze 的输入输出端口，就可以对 SPI 总线上的器件进行访问了。以下是 PicoBlaze 通过输入和输出端口模拟 SPI 的读写一个字节的程序，程序中触发器 D1 对应于 PicoBlaze 的地址 E5，D2 对应于 E2，D3 对应于 E1，读和写的数据事先存在寄存器 S0 中。

读子程序为 ：

```
read_byte: LOAD s6, 08 ; //读一个字节子程序
           LOAD s5, 00 ; //写 0
           LOAD sf, FF ; //写 1
           LOAD se, E5 ; //SPICLK 地址
           LOAD sd, E2 ; //SPIIN 地址
read_byte1: OUTPUT sf, (se) ; //SPICLK 变 1
            LOAD sb,00
            INPUT sA, (sd) ; //读入数据
            SL0 sA ;
            SLA s0 ; //S0 移位
            OUTPUT s5, (se) ;
             LOAD s9,00
```

```
                LOAD s9,00
                SUB s6, 01 ;
                JUMP NZ, read_byte1 ;
                RETURN
```

写子程序为

```
write_byte:   LOAD s6, 08 ; //写一个字节子程序
              LOAD s5, 00 ; //写 0
              LOAD sf, FF ; //写 1
              LOAD sd, E5 ; //SPICLK 地址
              LOAD se, E1 ; //SPIOUT 地址
write_byte1: OUTPUT sf, (sd) ;
              OUTPUT s0, (se)
              LOAD sa,00
              LOAD sa,00
              LOAD sa,00;
              OUTPUT s5, (sd) ;
              SL0 s0 ;
              SUB s6, 01 ;
              JUMP NZ, write_byte1 ;
              RETURN;
```

5、配置寄存器设置

作为通用的 IP 核，处理器通过访问寄存器来参与控制核的工作。在本设计中，我们在图 3.6 SPI 核结构 PicoBlaze 处理器外部添加 RAM，将 RAM 作为外部处理器和 PicoBlaze 指令和数据交换的通道。这样的设计有很大的灵活性，因为不同的外设可能有不同的寄存器个数和类型，只需根据需要修改 RAM 的大小和属性等，就可以适应不同外设的需求。我们使用设计的 IP 核用来访问 SPI 接口的 AD 芯片 ADS1210 芯片时，ADS1210 有 8 位指令寄存器，32 位命令寄存器、24 位数据寄存器、24 位偏置校准寄存器和 24 位满刻度校准寄存器。所以我们采用的 RAM 的是 16x8 bit 的 RAM，RAM 的配置如下表。

表 2.5 针对 ADS1210 的寄存器设置

0	INSR	ADS1210 指令寄存器
1	COM1	ADS1210 命令寄存器 1
2	COM2	ADS1210 命令寄存器 2
3	COM3	ADS1210 命令寄存器 3
4	COM4	ADS1210 命令寄存器 4
5	OCR0	ADS1210 偏置校准寄存器 1
6	OCR1	ADS1210 偏置校准寄存器 2
7	OCR2	ADS1210 偏置校准寄存器 3
8	D0	ADS1210 数据寄存器 1
9	D1	ADS1210 数据寄存器 2

A	D2	ADS1210 数据寄存器 23
B	FCR0	ADS1210 满刻度校准寄存器 1
C	FCR1	ADS1210 满刻度校准寄存器 2
D	FCR2	ADS1210 满刻度校准寄存器 3
E	未用	
F	未用	

6、SPI 核设计实现

综上所述，我们设计的基于 PicoBlaze 的 SPI 核的整体结构如图 3.7 所示，根据不同的场合 SPI 外设的不同特性，适当的修改 RAM 寄存器和程序，就可以访问不同的 SPI 外设，这样的设计比一般通用的 SPI 核具有更大的灵活性，用户可根据自己的需求增减核的配置，节省系统资源，体现了 SOPC 的设计思想。

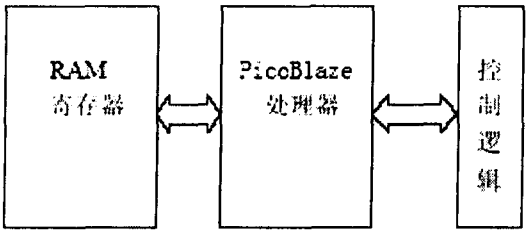

图 3.7 SPI 核结构

7、性能分析

设计实现使用 ISE6.2i，综合工具使用 XST，使用芯片为 Virtex 系列的 30 万门 FPGA 芯片 xcv300-6，综合实现的 RTL 级电路图如图 3.8 所示。

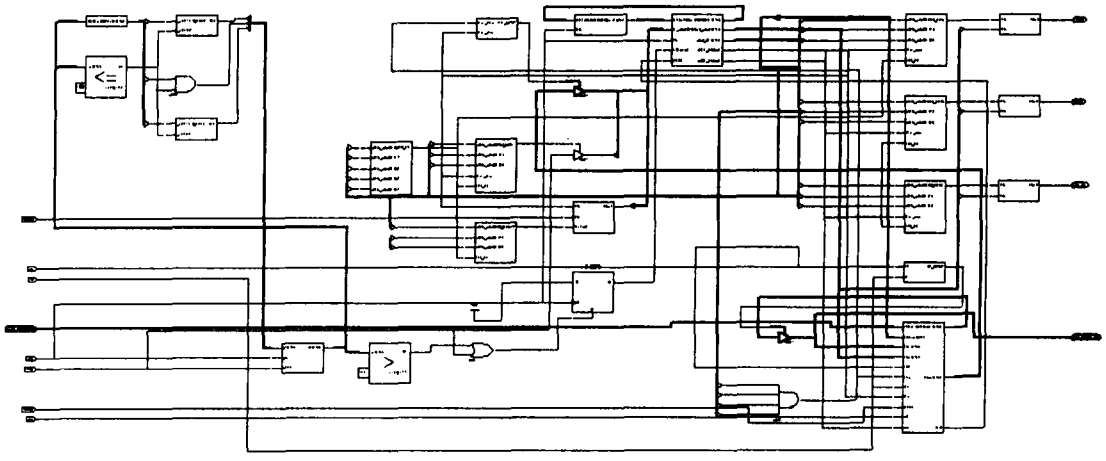

图 3.8 SPI 的 RTL 电路图

资源占用情况如下：

Number of Slices: 185 out of 3072 6%

Number of Slice Flip Flops:	200	out of	6144	3%
Number of 4 input LUTs:	229	out of	6144	3%
Number of bonded IOBs:	21	out of	264	7%
Number of TBUFs:	26	out of	3072	0%
Number of BRAMs:	1	out of	16	6%
Number of GCLKs:	1	out of	4	25%

占用了 185 个小区。在 30 万门的芯片中实现，占用 6%的资源。综合结果的最大时钟频率可达 47MHz。

3.4.3 UART 核设计

1、设计需求分析

航天惯测系统中根据上位机 DSP 的要求，UART 核需能完成一般 UART 传输，要求校验位可设置，奇偶校验可设置，中断可设置，波特率可设置，要求接收发送 FIFO 为 80 字节，并且发送数据时发送字节之间的时间间隔可控制，精度为 1us。

根据设计需求，给 UART 核设计了一组寄存器，以完成对核的控制，同时通过寄存器将核的状态提供给外部控制器。寄存器配置如表 2.6。

表 2.6 UART 核寄存器分配

寄存器名	地址	访问类型
接收 FIFO	000	读
发送 FIFO	000	写
控制寄存器	001	写
状态寄存器	010	读
FIFO 控制寄存器	011	写
发送间隔寄存器（高字节）	100	写
发送间隔寄存器（低字节）	101	写
波特率寄存器（高字节）	110	写
波特率寄存器（低字节）	111	写

接收寄存器和发送寄存器的 FIFO 地址相同，用读写信号区分。

控制寄存器的配置如表 2.7。

表 2.7 控制寄存器配置

位序	名称	缺省值	含义
7	保留	0	未使用
6	发送 FIFO 空中断	1	1—使能；0—未使能
5	接收 FIFO 有数据中断	1	1—使能；0—未使能
4	校验类型	1	1—有校验；0—无校验

3	校验选择	1	1—奇校验；0—偶校验
2	保留	0	未使用
1-0	字长度选择	11	

表 2.8 状态寄存器配置

位序	名称	含义
7	接收 FIFO 有无数据	1—有数据；0—无数据
6	接收 FIFO 是否充满	1—满；0—未满
5	发送 FIFO 空状态	1—空；0—不空
4	发送 FIFO 满状态	1—满；0—未满
3	中断使能状态	1—使能；0—未使能
2	溢出	1—溢出；0—未溢出
1	帧错误	1—出错；0—未出错
0	校验错误	1—出错；0—未出错

表 2.9 FIFO 控制寄存器配置

位序	名称	缺省值	含义
7-5	接收 FIFO 触发中断字节数	000	000—1 字节；001—17 字节 010—33 字节；011—49 字节 100—65 字节；
4-3	保留	00	未使用
2	复位发送 FIFO	1	1—复位；0—不复位
1	复位接收 FIFO	1	1—复位；0—不复位
0	保留	0	未使用

发送间隔寄存器用来配置向串口发送数据字节间时间间隔。

波特率分频比寄存器用来配置 UART 的波特率，分频比寄存器的高低字节分别保存分频比的高低字节。默认值为 27（系统时钟为 50MHz，串口的波特率为 115200Hz）。

UART 的传输机理比较简单，可以通过移位送出来完成，但是移位送出容易产生毛刺，要求可以设置有无的时候，发送每个字节实际发送的数据位数是变化的，并不好控制，需要设计一些特殊的结构，保证每次输出的数据正确[13]。

波特率的产生可以通过分频计数器来完成，发送自己的间隔控制，也可以通过控制波特率的产生来完成。

80 字节的 FIFO 可以用 FPGA 内部的 SRL16E 资源设计。

2、UART 的帧格式

UART 的帧格式如图所示。

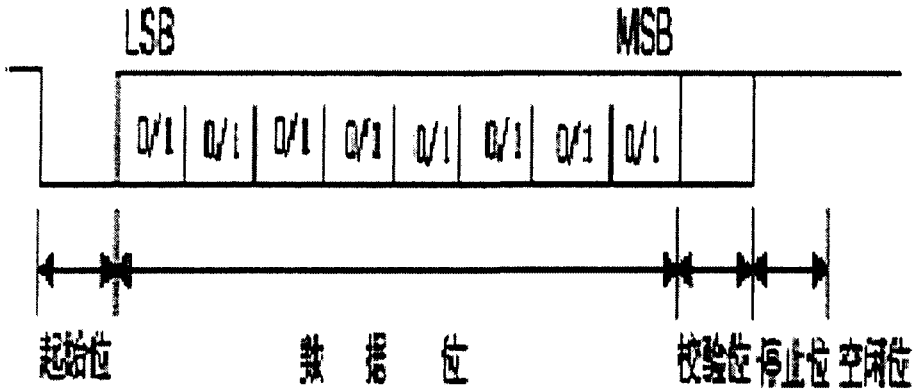

图 3.9 UART 帧格式

图 3.9 示,UART 数据帧包括线路空闲状态（idle，高电平）、起始位(start bit，低电平)、5～8 位数据位(data bits)、校验位(parity bit，可选)和停止位(stop bit，位数可为 1、1.5、2 位)。这种格式是由起始位和停止位来实现字符的同步。 UART 内部的配置寄存器，可以配置数据位数、是否有校验位和校验的类型、停止位的位数等设置[14]。

3、UART 核结构

根据设计需求，UART 核主要有由数据总线接口、控制逻辑、波特率发生器、发送部分和接收部分等组成。功能包括微处理器接口，控制寄存器、状态寄存器、FIFO 控制寄存器、波特率发生器、波特率控制寄存器、发送 FIFO、发送移位寄存器、帧产生、奇偶校验、并转串、数据接收 FIFO、接收移位寄存器等[14]。

图 3.10 为 UART 核的具体结构。

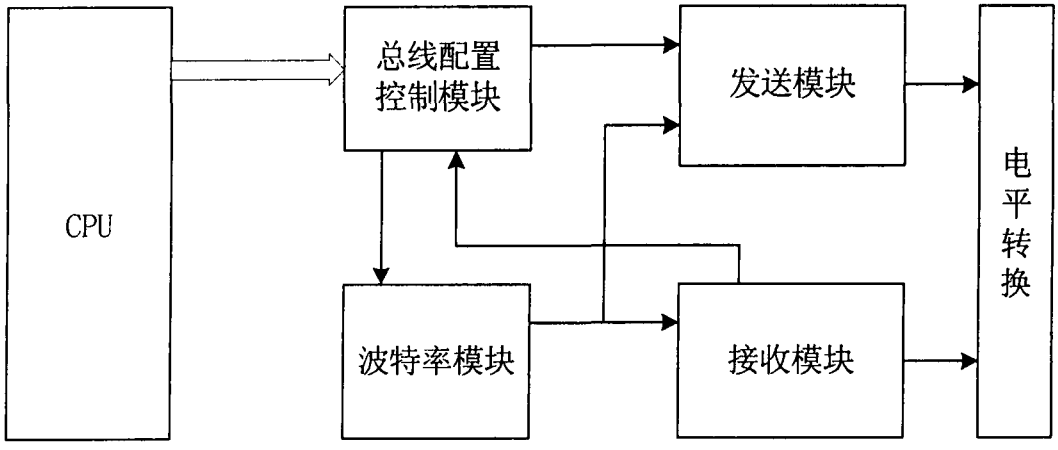

图 3.10 UART 核结构

4、UART 核的设计实现

波特率产生模块

波特率产生模块如图 3.11 所示。

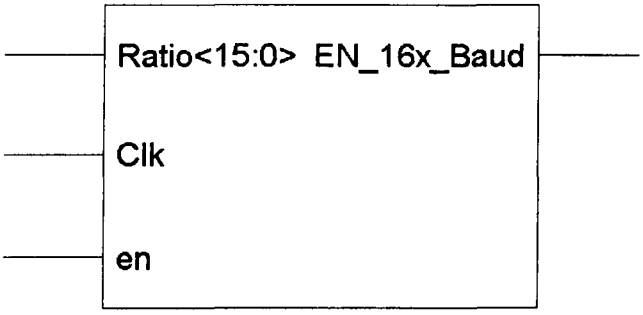

图 3.11 波特率产生模块

Ratio 端口从波特率寄存器读取 16 位数据，作为时钟分频系数，产生 16 倍于波特率的 EN_16x_Baud 信号，内部 VHDL 描述如下：

```
Counter : process (Clk) is
  begin   -- process Counter
    if Clk'event and Clk = '1' then        -- rising clock edge
     if en='0' then
      if (Count = 0) then
        Count          <= Ratio-1;
        Baud <= '1';
      else
        Count          <= Count - 1;
        Baud <= '0';
      end if;
 else
   Count<=Count;
   Baud<=Baud;
 end if;
   end if;
 end process Counter;
  EN_16x_Baud<=Baud;
```

波特率发生器的 RTL 原理图如 3.12。

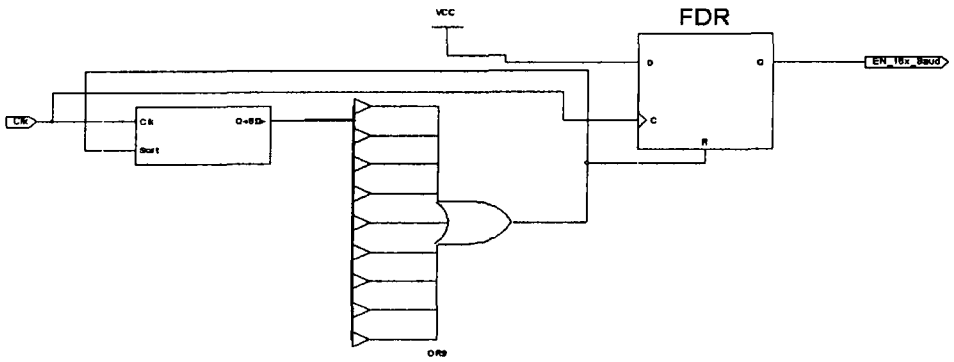

图 3.12 波特率发生器 RTL 原理图

发送模块设计

发送模块内部包含 80 字节 FIFO，外部控制器经过控制模块，把发送数据写入

发送 FIFO，发送模块的内部逻辑，从 FIFO 读出数据，经过并串转换，把并行数据转化成为串行数据，并判断是否带奇偶校验位，计算出校验位后，按波特率将数据串行发送出去，发送模块还根据 FIFO 状态，将 FIFO 满或空信号，发送给控制模块产生中断，输出给 CPU。

UART 数据发送使用数据选择器取代移位寄存器，保证数据和波特率配合的正确。数据发送开始时刻，从内部 FIFO 读取一个数据到寄存器，通过选择器组合来选择当前时刻要发送的数据某个 bit 位，同时把这个 bit 位放到发送数据端口，具体代码如下：

```
-------------------------------------------------------------------------------
 -- Select which bit within the data word to transmit
 -------------------------------------------------------------------------------
 -- Need special treatment for inserting the parity bit because of parity generation
 Parity_Bit_Insertion : process (parity, select_Parity, fifo_DOut) is
 begin   -- process Parity_Bit_Insertion
   data_to_transfer <= fifo_DOut;
   if (select_Parity = '1') then
     data_to_transfer(C_DATA_BITS-1) <= parity;
   end if;
 end process Parity_Bit_Insertion;

 mux_01 <= data_to_transfer(1) when mux_sel(2) = '1' else data_to_transfer(0);
 mux_23 <= data_to_transfer(3) when mux_sel(2) = '1' else data_to_transfer(2);

 Data_Bits_Is_5 : if (C_DATA_BITS = 5) generate
   mux_45 <= data_to_transfer(4);
   mux_67 <= '0';
 end generate Data_Bits_Is_5;

 Data_Bits_Is_6 : if (C_DATA_BITS = 6) generate
   mux_45 <= data_to_transfer(5) when mux_sel(2) = '1' else data_to_transfer(4);
   mux_67 <= '0';
 end generate Data_Bits_Is_6;

 Data_Bits_Is_7 : if (C_DATA_BITS = 7) generate
   mux_45 <= data_to_transfer(5) when mux_sel(2) = '1' else data_to_transfer(4);
   mux_67 <= data_to_transfer(6);
 end generate Data_Bits_Is_7;

 Data_Bits_Is_8 : if (C_DATA_BITS = 8) generate
   mux_45 <= data_to_transfer(5) when mux_sel(2) = '1' else data_to_transfer(4);
   mux_67 <= data_to_transfer(7) when mux_sel(2) = '1' else data_to_transfer(6);
 end generate Data_Bits_Is_8;

 MUX_F5_1 : MUXF5
   port map (
     O  => mux_0123,
     I0 => mux_01,
```

```
      I1 => mux_23,
      S  => mux_sel(1));

  MUX_F5_2 : MUXF5
    port map (
      O  => mux_4567,
      I0 => mux_45,
      I1 => mux_67,
      S  => mux_sel(1));

  MUXF6_I : MUXF6
    port map (
      O  => mux_out,
      I0 => mux_0123,
      I1 => mux_4567,
      S  => mux_sel(0));
  Serial_Data_DFF : process (Clk, Reset) is
  begin   -- process Serial_Data_DFF
    if Reset = '1' then                    -- asynchronous reset (active high)
      serial_Data <= '0';
    elsif Clk'event and Clk = '1' then     -- rising clock edge
      if select_Parity='0' then            --判断是否为校验位
      serial_Data <= mux_Out;                  --发送数据
else
 serial_Data<=Parity;                      --发送校验位
end if;
    end if;
  end process Serial_Data_DFF;
```

校验产生代码如下

```
-------------------------------------------------------------------------------
  -- Parity handling
  -------------------------------------------------------------------------------
  Using_Parity : if (C_USE_PARITY = 1) generate

    Using_Odd_Parity : if (C_ODD_PARITY = 1) generate
      Parity_Bit : FDSE
        -- pragma translate_off
        generic map (
          INIT => '1')
        -- pragma translate_on
        port map (
          Q  => Parity,
          C  => Clk,
          CE => tx_Data_Enable,
          D  => calc_Parity,
          S  => tx_Start);
    end generate Using_Odd_Parity;

    Using_Even_Parity : if (C_ODD_PARITY = 0) generate
      Parity_Bit : FDRE
        -- pragma translate_off
```

```
        generic map (
          INIT => '0')
        -- pragma translate_on
        port map (
          Q   => Parity,
          C   => Clk,
          CE => tx_Data_Enable,
          D   => calc_Parity,
          R   => tx_Start);
    end generate Using_Even_Parity;

    calc_Parity <= parity xor serial_data;

    tx_Run_DFF : process (Clk, Reset) is
    begin   -- process tx_Run_DFF
      if Reset = '1' then                    -- asynchronous reset (active high)
        tx_Run1 <= '0';
      elsif Clk'event and Clk = '1' then   -- rising clock edge
        if (tx_Data_Enable = '1') then
          tx_Run1 <= tx_DataBits;
        end if;
      end if;
    end process tx_Run_DFF;

    tx_Run <= tx_Run1 or tx_DataBits;

    Select_Parity_DFF : process (Clk, Reset) is
    begin   -- process Select_Parity_DFF
      if Reset = '1' then                    -- asynchronous reset (active high)
        select_Parity <= '0';
      elsif Clk'event and Clk = '1' then   -- rising clock edge
        if (tx_Data_Enable = '1') then
          select_Parity <= mux_sel_is_zero;
        end if;
      end if;
    end process Select_Parity_DFF;
  end generate Using_Parity;

  No_Parity : if (C_USE_PARITY = 0) generate
    tx_Run          <= tx_DataBits;
    select_Parity <= '0';
  end generate No_Parity;
```

发送模块内部 80 字节 FIFO 比较特别，它在信号上直接和 DSP 接口，DSP 为异步读写，而发送模块从 FIFO 读取数据采用的是同步读写，基于此 FIFO 的特殊性，使用 FPGA 内部 SRL16E 资源设计此 FIFO，RTL 原理图如图 3.13 示。

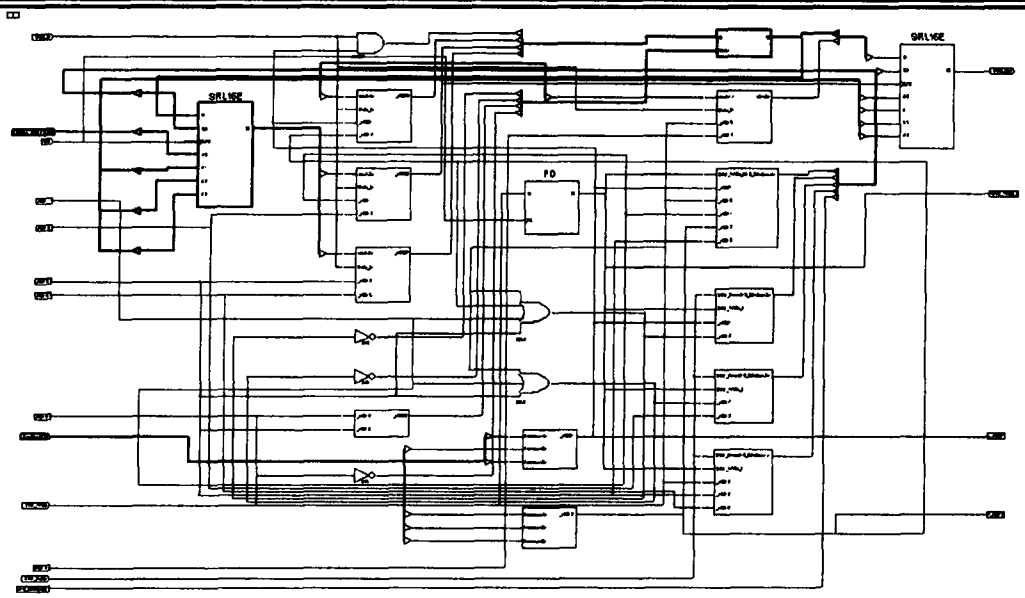

图 3.13 FIFO 的 RTL 原理图

源代码如下：

```
sig_choice : process(Address)
begin
  case Address(6 downto 4) is
  when "000" =>
      srl16_Choice(0) <= SRL_Write_i;
      srl16_Choice(1) <= SRL_Write_i;
      srl16_Choice(2) <= SRL_Write_i;
      srl16_Choice(3) <= SRL_Write_i;
      srl16_Choice(4) <= SRL_Write_i;
      din(0) <= Data_In;
      Address_SRL(3 downto 0) <= Address(3 downto 0);
      Address_SRL(19 downto 4) <= (others=>'0');
   when "001" =>
      srl16_Choice(0) <= SRL_Read;
      srl16_Choice(1) <= SRL_Write_i;
      srl16_Choice(2) <= SRL_Write_i;
      srl16_Choice(3) <= SRL_Write_i;
      srl16_Choice(4) <= SRL_Write_i;
      din(0) <= qout(1);
      din(1) <= Data_In;
      Address_SRL(3 downto 0) <= "1111";
      Address_SRL(7 downto 4) <= Address(3 downto 0);
      Address_SRL(19 downto 8) <= (others=>'0');
   when "010" =>
      srl16_Choice(0) <= SRL_Read;
      srl16_Choice(1) <= SRL_Read;
      srl16_Choice(2) <= SRL_Write_i;
      srl16_Choice(3) <= SRL_Write_i;
      srl16_Choice(4) <= SRL_Write_i;
      din(0) <= qout(1);
      din(1) <= qout(2);
```

```
        din(2) <= Data_In;
        Address_SRL(7 downto 0) <= (others=>'1');
        Address_SRL(11 downto 8) <= Address(3 downto 0);
        Address_SRL(19 downto 12) <= (others=>'0');
      when "011" =>
        srl16_Choice(0) <= SRL_Read;
        srl16_Choice(1) <= SRL_Read;
        srl16_Choice(2) <= SRL_Read;
        srl16_Choice(3) <= SRL_Write_i;
        srl16_Choice(4) <= SRL_Write_i;
        din(0) <= qout(1);
        din(1) <= qout(2);
        din(2) <= qout(3);
        din(3) <= Data_In;
        Address_SRL(11 downto 0) <= (others=>'1');
        Address_SRL(15 downto 12) <= Address(3 downto 0);
        Address_SRL(19 downto 16) <= (others=>'0');
      when "100" =>
        srl16_Choice(0) <= SRL_Read;
        srl16_Choice(1) <= SRL_Read;
        srl16_Choice(2) <= SRL_Read;
        srl16_Choice(3) <= SRL_Read;
        srl16_Choice(4) <= SRL_Write_i;
        din(0) <= qout(1);
        din(1) <= qout(2);
        din(2) <= qout(3);
        din(3) <= qout(4);
        din(4) <= Data_In;
        Address_SRL(15 downto 0) <= (others=>'1');
        Address_SRL(19 downto 16) <= Address(3 downto 0);
      when others =>
        srl16_Choice(4 downto 0) <= "00000";
        din <= "00000";
        Address_SRL(19 downto 0) <= (others=>'0');
end case;
end process;

srl80_gen : for i in 0 to 4 generate
begin
  gen_first : if (i=0) generate
begin
  SRL16E_L : SRL16E
    -- pragma translate_off
    generic map (
      INIT => x"0000")
    -- pragma translate_on
    port map (
      CE  => srl16_Choice(0),
      D   => din(0),--qout(1),
      Clk => Clk,
      A0  => Address_SRL(0),
```

```
            A1   => Address_SRL(1),
            A2   => Address_SRL(2),
            A3   => Address_SRL(3),
            Q    => Data_Out);
      end generate gen_first;

   gen_mid : if (i>0 and i<4) generate
   begin
   SRL16E_I : SRL16E
         -- pragma translate_off
         generic map (
            INIT => x"0000")
         -- pragma translate_on
         port map (
            CE   => srl16_Choice(i),
            D    => din(i),--qout(i+1),
            Clk => Clk,
            A0   => Address_SRL(4*i+0),
            A1   => Address_SRL(4*i+1),
            A2   => Address_SRL(4*i+2),
            A3   => Address_SRL(4*i+3),
            Q    => qout(i));
      end generate gen_mid;

   gen_last : if (i=4) generate
   begin
      SRL16E_H : SRL16E
         -- pragma translate_off
         generic map (
            INIT => x"0000")
         -- pragma translate_on
         port map (
            CE   => srl16_Choice(4),
            D    => din(4),--Data_In,
            Clk => Clk,
            A0   => Address_SRL(16),
            A1   => Address_SRL(17),
            A2   => Address_SRL(18),
            A3   => Address_SRL(19),
            Q    => qout(4));
      end generate gen_last;
   end generate srl80_gen;
```

接收模块设计

串行数据帧和接收时钟是异步的，发送来的数据由逻辑 1 变为逻辑 0 可以视为一个数据帧的开始。接收模块先要捕捉起始位，确定 RXD 输入由 1 到 0，然后在每个波特率信号有效时采样接收数据，移位输入接收移位寄存器，并根据校验位判断数据是否正确，若数据正确将数据写入内部 80 字节 FIFO，若不正确产生校验错中断，发送给控制模块，FIFO 内数据个数和中断触发寄存器个数相同时，产

生触发中断，发送给控制模块。

接收模块如图 3.14 示，接收模块还产生帧错误，Overrun 错误等中断信号。FIFO 有数据信号等给控制模块，写入状态字，供 CPU 读取。

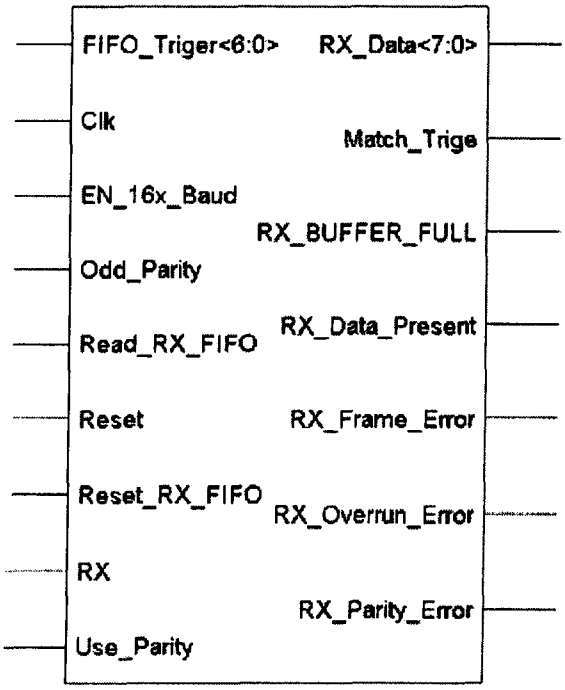

图 3.14 接收模块

5、性能分析

设计实现使用 ISE6.2i，综合工具使用 XST，使用芯片为 Virtex 系列的 30 万门 FPGA 芯片 xcv300-6，综合实现的 RTL 级电路图如图 3.15 所示。

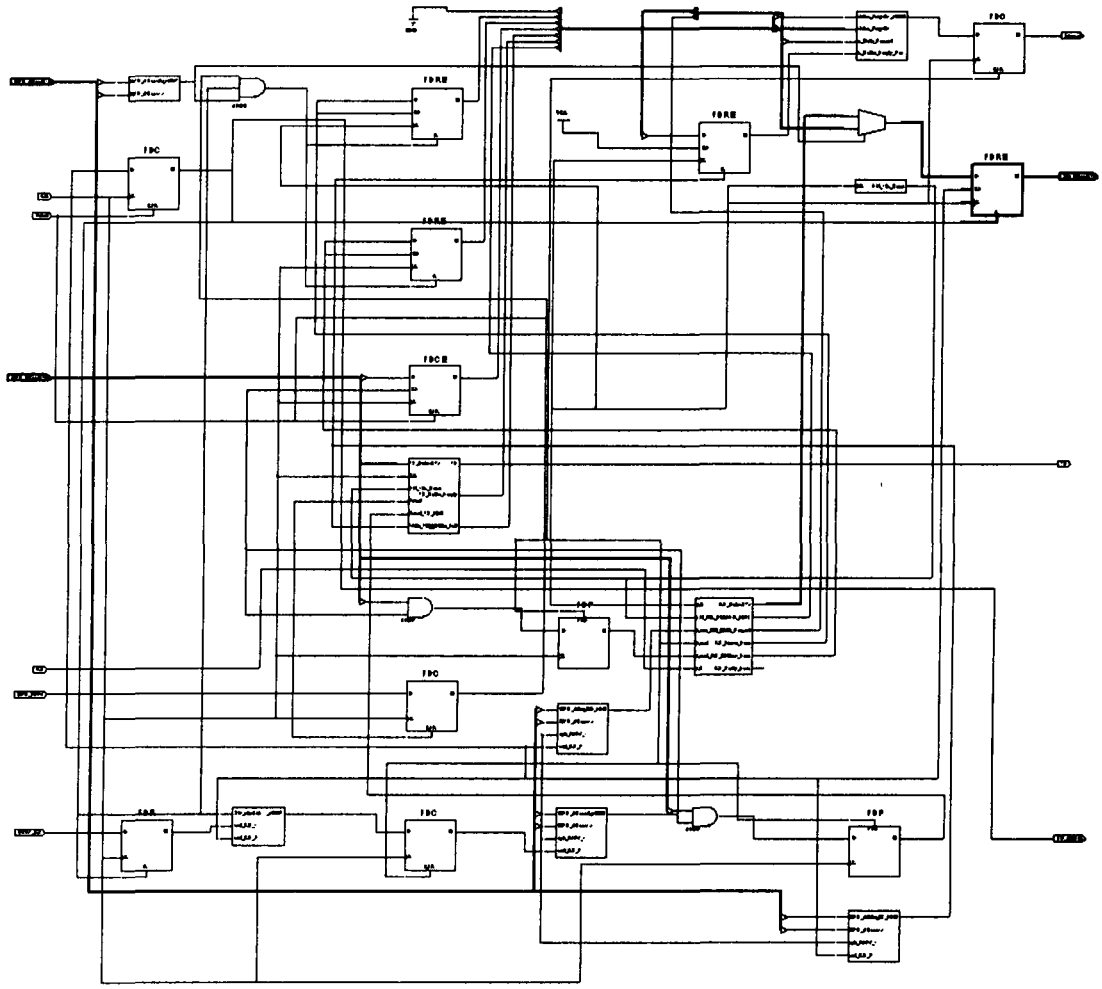

图 3.15 UART 的 RTL 原理图

资源占用情况如下：

Number of Slices:	55	out of	3072	1%
Number of Slice Flip Flops:	63	out of	6144	1%
Number of 4 input LUTs:	93	out of	6144	1%
Number of bonded IOBs:	82	out of	264	31%
Number of GCLKs:	1	out of	4	25%

仅占用了 55 个小区。在 30 万门的芯片中实现，占用 1%的资源。综合结果的最大时钟频率可达 117MHz。

2.5 本章小结

本章针对传统电路设计的局限性，设计了一种基于 IP 复用技术的航天惯测系统，此系统解决了传统惯测系统占用空间大，重量重的缺点，而且系统采用的 IP 复用设计技术，对将来系统更新升级提供了方便；

此系统的原理样机，如图 3.16 所示。此系统目前已在航天某型号惯测系统中

运行，运行情况良好，得到了用户的充分肯定。

图 3.16 航天惯测系统原理样机

第四章　处理器 IP 的 FPGA 实现和应用

本章通过对 Xilinx 的 MicroBlaze IP 结构的分析，对其开发使用进行研究；在此基础上，设计了一种基于 MicroBlaze 的 CPCI 动态可重构数据采集系统。

4.1　微处理器 IP 的设计原理和及其优势

当今的设计工程师按照以往的惯例，完成符合性能和特性竞争要求的产品设计，但是同时也面临着更紧迫的预算和市场份额减少的压力。过去，采用 ASIC 或者合适的 ASSP 技术就可以实现满足高级性能需求、最具成本效益的大批量产品设计。ASIC 技术曾经是实现最佳性能表现的最低成本途径，但是缺乏设计灵活性，而且产品面市时间较长，开发风险很高。另一方面，ASSP 在产品及时面市和保证开发成功上风险较小，但仍旧没有可编程逻辑的灵活性。

现成的 ASSP 和微控制器具有逐渐过时的缺点。多数硬件嵌入式处理器随着时间的推移而逐渐过时，并最终被淘汰。结果，采用这种处理器的制造商不得不重新设计硬件，并重新编写软件，耗费了大量的时间和成本。由于采用了新的处理器系列，这种重新设计甚至要求一套全新的指令集，导致更多的时间和资金投入。软核嵌入式处理器由于可以在未来的 FPGA 系列中继续使用，因此有助于改变这种系统逐渐过时的缺点，并确保终端用户的软件代码与多代硬件兼容。

FPGA 和处理器软核，实现了最具成本效益的 SOPC 解决方案。FPGA 和处理器软核使设计人员能够构建与中等密度 ASIC 和 ASSP 有效竞争的 SOPC 设计。这样，设计人员可以在大量的新应用中充分利用可编程逻辑的灵活性、产品及时面市以及成本优势来进行设计，而这些应用在以前完全被 ASIC 和 ASSP 技术所统治。比如 Altera 公司的 Nios II/Cyclone II 设计组合能够以每逻辑门$0.35 的成本实现完整的嵌入式处理系统，其性能超过 100 DMIPS。Nios II 系列包括：高性能内核（Nios II/f，“快速”）；低成本内核（Nios II/e，“经济”）；性能/成本均衡的内核（Nios II/s，“标准”）。这三个内核共享一套相同的 32 比特指令集体系（ISA），与二进制代码百分之百兼容。它们具有单个相同的免版税许可，增强了设计灵活性，同时将成本降到最小。此外，所有内核都可以使用用户指令，使关键软件子程序在 Nios II 嵌入式处理器的操作控制下，能够在 FPGA 中运行。这些指令在硬件中运行时需要占有多个时钟周期，而在 Nios II 处理器应用中，最少只占用一个时钟周期，大大提高了系统性能和数据吞吐量[15]。

FPGA 制造商认识到用户，特别是在消费类市场上，面临产品及时面市和产品较短生命周期的压力，因此，他们开发了整套工具——包括处理器软核集成设计环境（IDE）和 FPGA 综合布线开发软件，帮助用户加速硬件和软件的开发，实现

完整的基于可编程逻辑的 SOPC 解决方案。与此同时，具有微处理器软核许可的用户将收到含有软处理器内核和一套软件工具的开发工具包，用于在 FPGA 中进行处理器设计。

4.2 MicroBlaze 处理器 IP 的设计实现技术

4.2.1 MicroBlaze 的功能与结构

MicroBlaze 是 32 位微处理器。采用 RISC 指令集，Harvard 体系结构，分离的 32 位指令和 32 位数据总线，以全速执行指令；可以访问片内存储器和片外存储器；在 VII 以上系列芯片实现时使用硬件乘法器，因此有着比较高的性能。在 Spartan-IIE 系列芯片上实现时，约占用 500 Slices[6]。

其系统结构如图 4.1 所示：

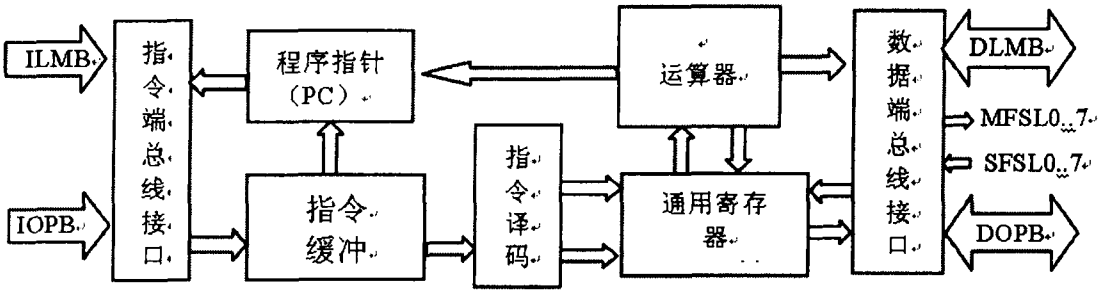

图 4.1 MicroBlaze 系统功能框图

MicroBlaze 有着与一般的通用微处理器类似的结构，同时又有着自己的特点：

指令系统：MicroBlaze 的指令字为 32 位，其指令又分为 A 型和 B 型，A 型指令有两个源寄存器操作数和一个目的寄存器操作数；B 型指令有一个目的寄存器、一个源寄存器和一个 16 位的立即数（通过在 B 型指令前加一个 IMM 指令可将其扩展到 32 位）；按功能划分有算术运算、逻辑运算、分支、存/取和特殊指令等。

寄存器：MicroBlaze 是一个完全正交的体系结构，有 32 个通用寄存器和两个特殊功能寄存器。

32 个通用寄存器定义为 R0-R31，其中 R0 的值总为 0，任何对其的写操作均无效。特殊功能寄存器有程序计数指针 PC（Program Counter）、状态标志寄存器 MSR（Machine State Register）等。特殊寄存器 PC 指令指针，存储的是下一条指令的地址。状态标志寄存器 MSR 中保存进位标志、中断使能、总线锁定、缓冲标志和 FSL 出错标志等标志位。可通过 MFS 指令读取状态寄存器，MTS 指令写状态字。

指令和数据缓冲：MicroBlaze 可以选择配置指令和数据缓冲。当从 LMB 总线地址以外的区域执行程序或读取数据时，使用指令和数据缓冲可以提高系统性能。

对缓冲可以配置大小和控制打开与关闭。缓冲的实现不需要特殊的存储器控制逻辑，占用的资源较少（小于 20 个 LUT）。

流水线：MicroBlaze 流水线为并行流水线，分三级：取指，译码和执行。

通常，每一级均在一个时钟周期内完成，完成一条指令需要三个时钟周期。并行流水线的每一级在每一个时钟周期都是活动的，三条指令在流水线的不同级中在同时运行。可以做到在每一个时钟周期有一条指令执行完成。

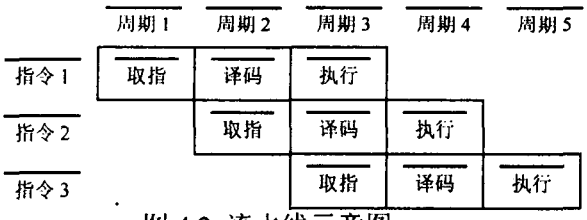

图 4.2 流水线示意图

存储结构：MicroBlaze 可以用字节（8 位）、半字（16 位）和字（32 位）三种数据宽度对存储器进行访问。MicroBlaze 是 Big-Endian 格式的处理器，使用 Big-Endian 格式的地址。图 4.3 为地址访问时地址格式和位、字节标识的转换，其中缩写词的含义如下[16]：

MSByte：最高标志字节；　LSByte：最低标志字节；

MSBit：最高标志位；　LSBit：最低标志位。

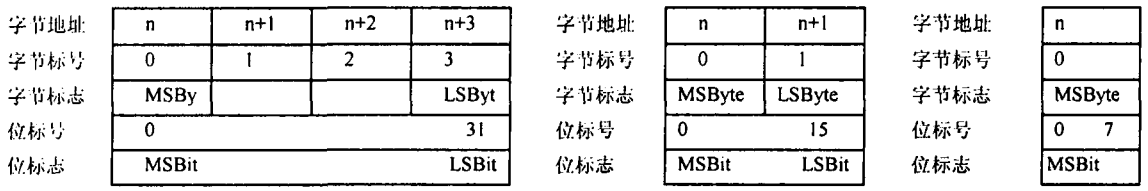

图 4.3　Big Endian 数据类型

中断控制：MicroBlaze 可以响应软件和硬件中断，进行异常处理。MicroBlaze 有一个外部中断输入端口，通过外加控制逻辑，可以扩展外部中断，能够响应多个中断。

快速单工连接接口（FSL）：MicroBlaze 中还包含了 8 个 FSL 输入接口和 8 个 FSL 输出接口。FSL 通道是专用的单方向点到点数据流接口，32 位数据宽度。

调试接口：MicroBlaze 在硬件上还支持基于 JTAG 软件调试工具的调试接口（称为 BDM Background Debug Mode Debuggers），类似 Xilinx 的微处理器调试（XMD）工具。调试接口连接到“微处理器调试模块（MDM）”IP 核上，MDM 再与 FPGA 的 JTAG 端口相联，就可以应用软件调试工具通过 FPGA 的 JTAG 接口来调试处理器系统。一个 MDM 可以支持调试多个 MicroBlaze。

总线接口：MicroBlaze 采用 Harvard 体系结构，具有分离的数据总线和地址总线。这两种总线又有局部存储器总线（Local Memory Bus　LMB）和 IBM 的片上

总线（On-chip Peripheral Bus　OPB）两类。LMB 总线可提供对片上双端口 Block RAM 的单周期的快速访问。OPB 总线接口既可提供对片上外设、存储器的访问，又可对片外的外设和存储芯片进行访问。

MicroBlaze 的总线接口有六种配置模式[17]：

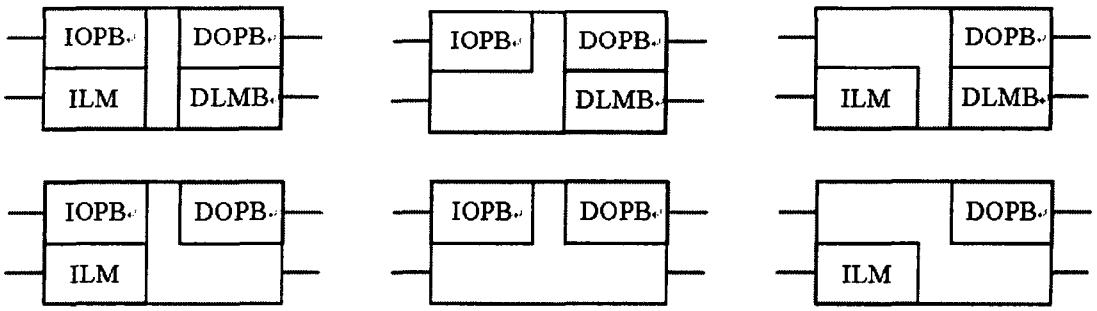

图 4.4　MicroBlaze 系统总线接口形式

其中：

DOPB：OPB 数据接口；DLMB：LMB 数据接口；

IOPB：OPB 指令接口；ILMB：LMB 指令接口；

根据代码的长度和数据量的大小，以及是否需要对内部块 RAM 的快速访问，来选择最佳配置模式。

通过对 MicroBlaze 的分析研究，可以看出 MicroBlaze 是一个真正的 32 位处理器：32 位的指令字，32 位的指令和数据总线，所以处理器有较强的寻址能力；而流水线的设计和指令、数据缓冲的使用，使得处理器有着较快处理速度。在 Spartan IIE 系列芯片上实现时，时钟频率可达 50MHz 以上。

4.2.2　MicroBlaze 的开发环境

应用 EDK（Embedded Development Kit）开发套件，可以进行 MicroBlaze IP 的开发。工具包中集成了硬件平台产生器、软件平台产生器、仿真模型生成器、软件编译器和软件调试工具等。EDK 中提供一个集成开发环境 XPS（Xilinx Platform Studio），在其中可以使用系统提供的所有工具，完成嵌入式系统开发的整个流程。EDK 中还带有一些外设接口的 IP core，如 LMB、OPB 总线接口，外部存储控制器，SDRAM 控制器，UART，中断控制器，定时器等，利用这些资源，可以构建一个较为完善的嵌入式微处理器系统[17]。

嵌入式软件设计工具的结构如图 4.5 所示。所有的设计工具都集成在一个平台之上，可以在这一个环境下完成系统设计的整个过程。

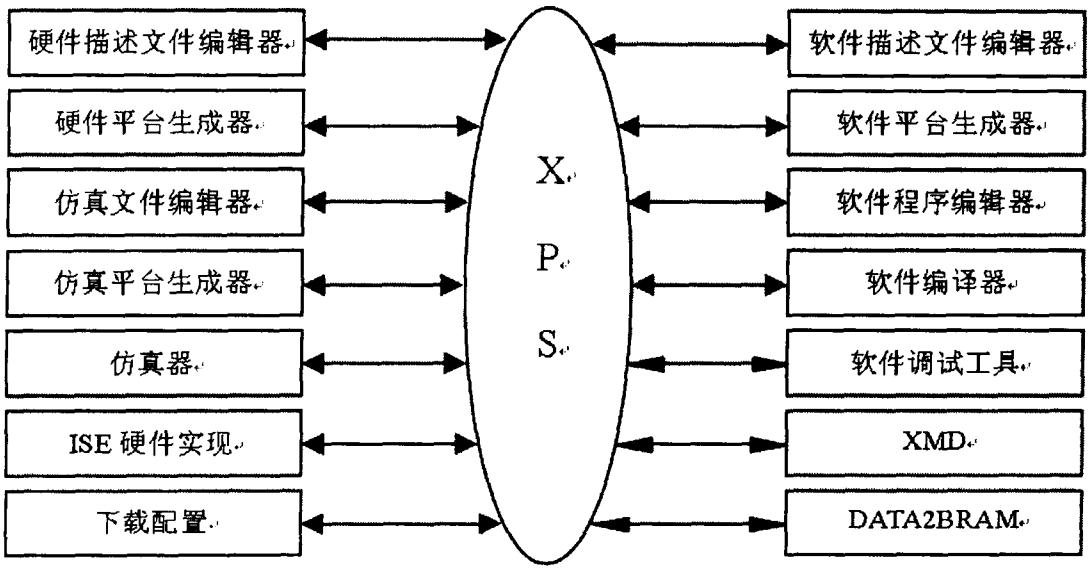

图 4.5 开发套件的组织结构

EDK 中包含以下主要设计工具：

Xilinx Platform Studio（XPS）。XPS 提供一个图形化的硬件和软件设计界面，还可以进行源程序编辑和工程管理。在 XPS 内可以管理整个设计流程，包括硬件流程和软件流程。

Platform Generator（平台产生器）。定制嵌入式处理器系统的硬件环境，产生系统的 HDL 描述文件和网表文件。

Simulation Model Generator（仿真模型产生器），仿真模型产生器根据 MVS 文件针对硬件配置的不同产生相应的仿真模型，使用者可以定义不同的仿真工具和 HDL 语言。

Library Generator（库产生器），库产生器使用 MSS 文件，为嵌入式系统配置库、设备驱动、文件系统和中断等。

GNU Compiler Tools（GNU 编译器），XPS 调用 GNU 编译器对应用程序进行编译链接，生成可执行文件。

另外还有软件调试工具 XMD，当硬件系统构建完成并在目标芯片中运行后，可利用 XMD 将程序下载到系统中进行调试[17]。

4.2.3 MicroBlaze 的开发流程

在 FPGA 上设计的嵌入式系统的层次结构如图 4.6 所示。其中，在 FPGA 提供的硬件资源上开发 IP ，或利用已开发的 IP 资源构建嵌入式系统，是硬件开发部分；开发 IP 的设备驱动（Driver）、应用接口（API）和应用层（算法）属软件开发内容。对于软、硬件开发，都可以在 Xilinx 提供的嵌入式系统开发套件 EDK 中完成。

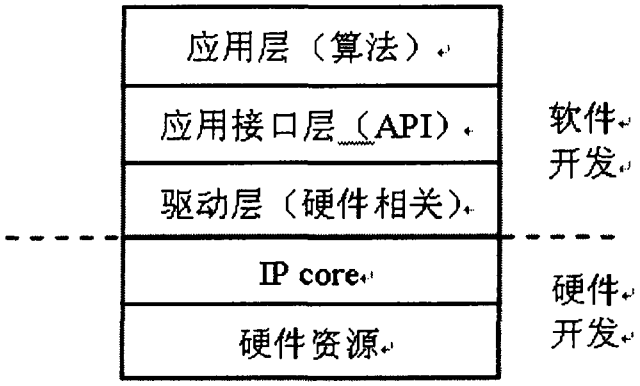

图 4.6 嵌入式系统开发层次结构

典型的嵌入式系统的开发流程包括以下几步[10]：

硬件平台的设计产生；

硬件平台的验证（仿真）；

软件平台的设计产生；

应用软件的设计；

软件调试。

硬件平台是在 MHS（Microprocessor Hardware Specification）文件中定义的，在文件中定义系统结构、外围设备和嵌入式微处理器。同时还定义了系统中设备的相互连接、地址映射和每个设备的参数配置。硬件平台产生器根据 MHS 文件产生硬件的网表文件（.ngc，.edif 文件）和顶层 HDL 描述文件。

硬件开发就是将所需要的 IP，包括处理器、总线接口、外设计接口、存储器接口（部分 IP 资源 EDK 中已提供）等所需要的 IP 资源，按照一定规则进行连接，并进行相应的参数设置，对其进行综合实现。

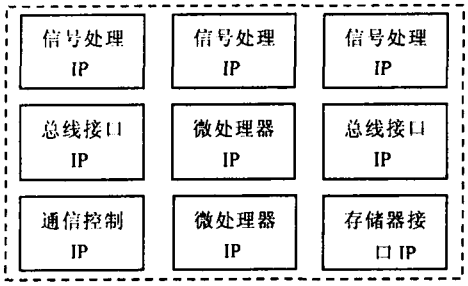

图 4.7 嵌入式系统的组织结构

验证平台是在硬件平台基础上产生的，在 MVS（Microprocessor Verification Specification）文件中定义系统硬件的仿真模型，利用仿真模型产生器产生仿真文件。再用仿真工具对硬件进行仿真。

软件平台在 MSS（Microprocessor Software Specification）文件中定义。在 MSS 文件中主要定义外设和处理器的用户配置参数，驱动程序和库文件、标准输入输

出、中断控制和其它相关的软件特性。

EDK 中提供的 IP 均有相应的设备驱动和应用接口，使用者只需利用相应的函数库，编写自己的应用软件和算法程序。对于用户自己开发的 IP ，就需要自己编写相应的驱动和接口函数。软件设计流程如图 4.8 所示。

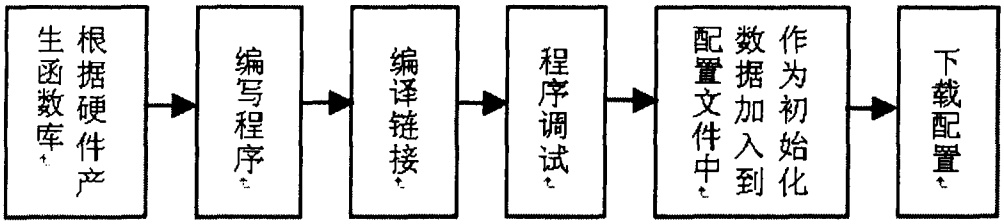

图 4.8 软件开发流程

用户在硬件平台和软件平台上编写自己的应用程序。可以用高级语言，如 C 或 C++，或者用汇编语言。再用 GNU 编译工具编译连接生成可执行的 ELF 文件。利用 XMD 和 GDB 工具进行软件调试。

4.3 一种基于 MicroBlaze 的 CPCI 动态可重构数据采集系统设计

现代工业生产和科学研究对数据采集和处理的要求越来越高，在雷达瞬时信号测量、图像处理等一些高速高精度的测量中，都需要用到高速高精度的数据采集和处理。基于 PCI 总线的高速数据采集卡，以其高带宽、高精度、高性能的特性，在高速数据采集和处理的场合，应用越来越广泛。一般的 PCI 板卡，存在以下缺点：安装麻烦、不支持热插拔、不同的应用场合需要设计相应的板卡，以满足应用需求，增加重复设计的风险。

CompactPCI（简称 cPCI）是国际 PICMG 协会于 1994 提出来的一种总线接口标准。CPCI 抛弃了 PC 工控机的金手指式互连方式，改用 2mm 密度的针孔连接器，具有气密性、防腐性，进一步提高了系统的可靠性，并增加了负载能力，保证了 99.999%的高可靠度。 cPCI 具有抗振性能好、开放性强、高可用性、高可靠性、可热插拔等优点。本章将提出一种动态可重构的 CPCI 数据采集处理系统构架，支持热插拔，通过动态重构硬件和软件适应不同的数据采集和处理要求[18]。

4.3.1 系统工作原理

根据设计需求，本系统是一种基于 CPCI 总线、FPGA 和 DSP 的动态可重构软硬件采集系统（重构过程由 PC 软件控制实现）。系统重构包括软件和硬件的重构。系统通过 CPCI 总线和主机交换程序和指令，接受重构指令和数据，将不同用途的 FPGA 算法 bit 流文件、DSP 程序文件和 PC 应用程序保存在 PC 机中，当采集系统

改变任务时，只需将相应用途的 FPGA 算法 bit 流文件或 DSP 程序文件从 PC 文件系统加载到 FPGA 或 DSP 中，调用相应用途的 PC 软件，即完成整个采集系统的任务转换。这种动态可重构方式的采集系统较比传统的完成特定功能的非重构采集系统而言，具有任务可转换、设备可复用、配置灵活、运行速度高等特点[19]。

4.3.2 系统可重构硬件设计

CPCI 总线动态可重构采集系统由主控制器、PCI 控制器、FPGA 及其配置存储器、DSP 及其配置存储器、A/D 前端五大部分构成。如图 4.9 所示。

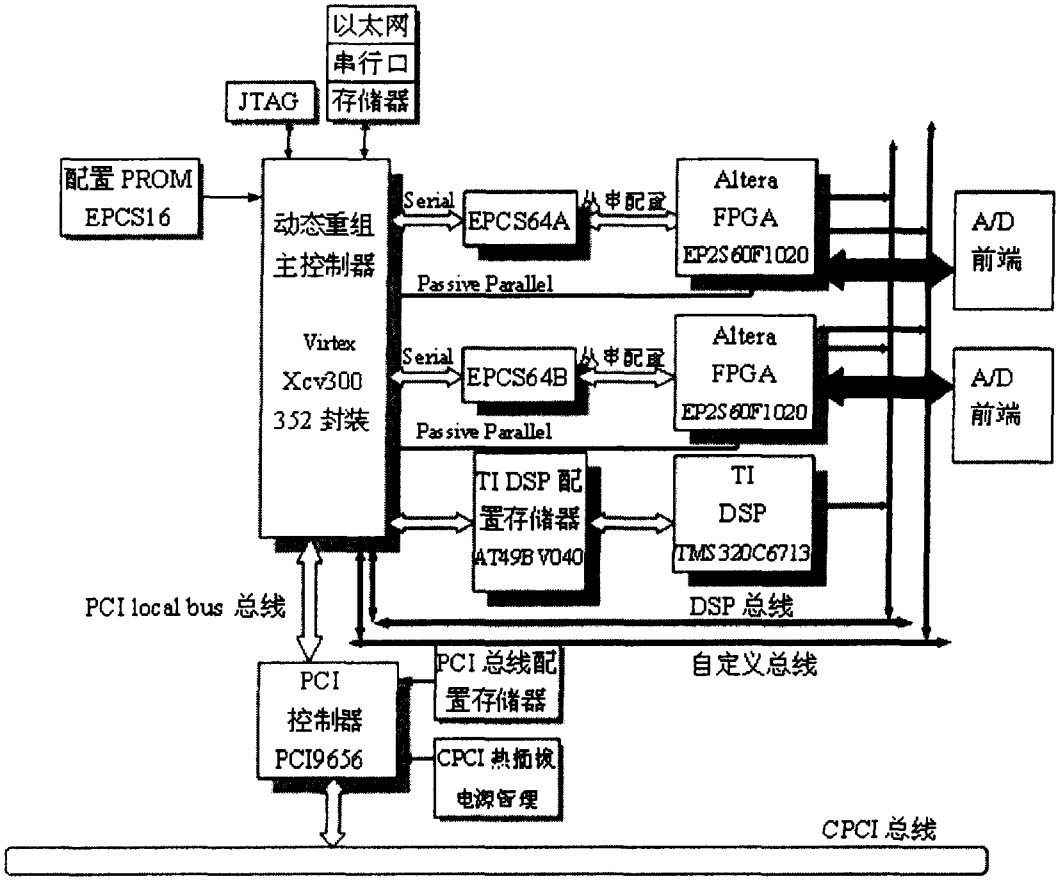

图 4.9 基于 CPCI 总线的动态可重构采集系统硬件框图

主控制器是实现可重构功能的关键部分，它是连接信号采集处理部分和 CPCI 总线数据交换的通道；主控制器与 PCI 控制器通过 PCI local bus 总线交换指令和数据，通过 DSP 总线和自定义的总线从 FPGA 和 DSP 获得采集和处理的数据。在系统重构模式下，它通过 CPCI 控制器接受主机重构指令和数据，对 FPGA 的配置存储器串行在线编程，对 DSP 的程序存储器进行在线编程，同时也可以直接对 FPGA 进行在线并行加载，完成系统的硬件重构；在系统正常工作模式下，主控制

器把从采集部分获得的实时采集数据通过 CPCI 总线传输给主机。基于 SOPC 的设计思想，主控制器使用 MicroBlaze 软处理器 IP Core 及外围逻辑实现编程实现。

CPCI 控制器采用 PLX 公司的 PCI9656，这是一种先进的 64bit/66MHZ PCI 总线控制器，支持热插拔和电源管理,支持直接读写和两个 DMA 通道,Jungo 公司提供了高速的数据传输驱动和热插拔驱动程序。主控制器可直接操作 PCI9656，也可以作为 PCI9656 的从设备，被它访问。CPCI 的热插拔电源管理部分我们采用了凌特电源芯片 LTC1643L，这是一个 PCI 热插拔专用芯片，支持过流过压保护，有 5V(voltage)、3V、±12V 输出[20]。

FPGA 可根据设计需求选用容量合适的芯片。我们根据需求采用的 Altera EP2S60，其配置存储器用的是 EPCS64。配置存储器可以由主控制器在线并行加载。考虑到 EPCS64 容量比较大，串行可能需要一段时间，在对实时性要求比较高的场合，不能满足快速重构系统，快速切换采集系统功能的要求，我们的设计主控制器可以对 FPGA 进行并行加载，减小系统切换时间损耗[21]。

DSP 采用了 TI 公司的 TMS320C6713, TMS320C6713 是 T I 公司在 TMS320C6711 的基础上推出的 C6000 系列新一代浮点 DSP 芯片，它是目前为止 C6000 系列 DSP 芯片中性能最高的一种。TMS320C6713 可在 255M H z 的时钟频率下实现 1800MIPS / 1350MFLOPS 的定点和浮点运算，因而可极大地满足通信、雷达、数字电视等高科技领域对信号处理实时性的要求。其配置 FLASH 我们采用的是 AT49VB040[22]。

A/D 采用 TI 公司的模数转换芯片 ADS5102，该芯片是一款 4 通道 65Mbps－10Bit 高速 ADC 芯片，具有以下几个特点:差分信号输入,1.8V(voltage)低电压模拟数字供电,兼容 1.8V、3.3V 逻辑电平,105mW(milliwatt)工作功耗,336uW(microwatt)功耗的省电模式;ADS5102 采用低功率 CMOS 设计，具有 10 位分辨率，工作电压为 1.8V，具有内部、外部参考两种模式，可以满足不同电平的输入信号，模拟输入信号采用差分方式，从而具有较高的共模抑制比。输出数字信号，采用 10 位并行输出方式[23]。

4.3.3 系统可重构软件设计

- ✧ PC 软件开发平台：Windows 2000/NT、Jungo PCI 驱动和热插拔驱动、Visual C++;
- ✧ FPGA 开发平台：Altera 的 QautusII（含 SOPC builder 和 DSP builder）XILINX 的 ISE 6.3
- ✧ DSP 开发平台：TI CCS2.2＋

系统软件的工作流程如图 4.10 所示，首先在系统上电或板卡热插入后， PC

端检测到板卡并加载 PCI 驱动程序，主控制器进行上电配置，FPGA 和 DSP 处于复位保持状态。在 PC 程序与主控制器的配合下，对 FPGA 和 DSP 进行任意顺序的配置，直至完成配置；然后主控制器释放 DSP 与 FPGA 并使其开始正常工作；接着利用 PC 程序与 DSP 软件进行数据包交换的方式对 PCI 总线及板卡进行自检，

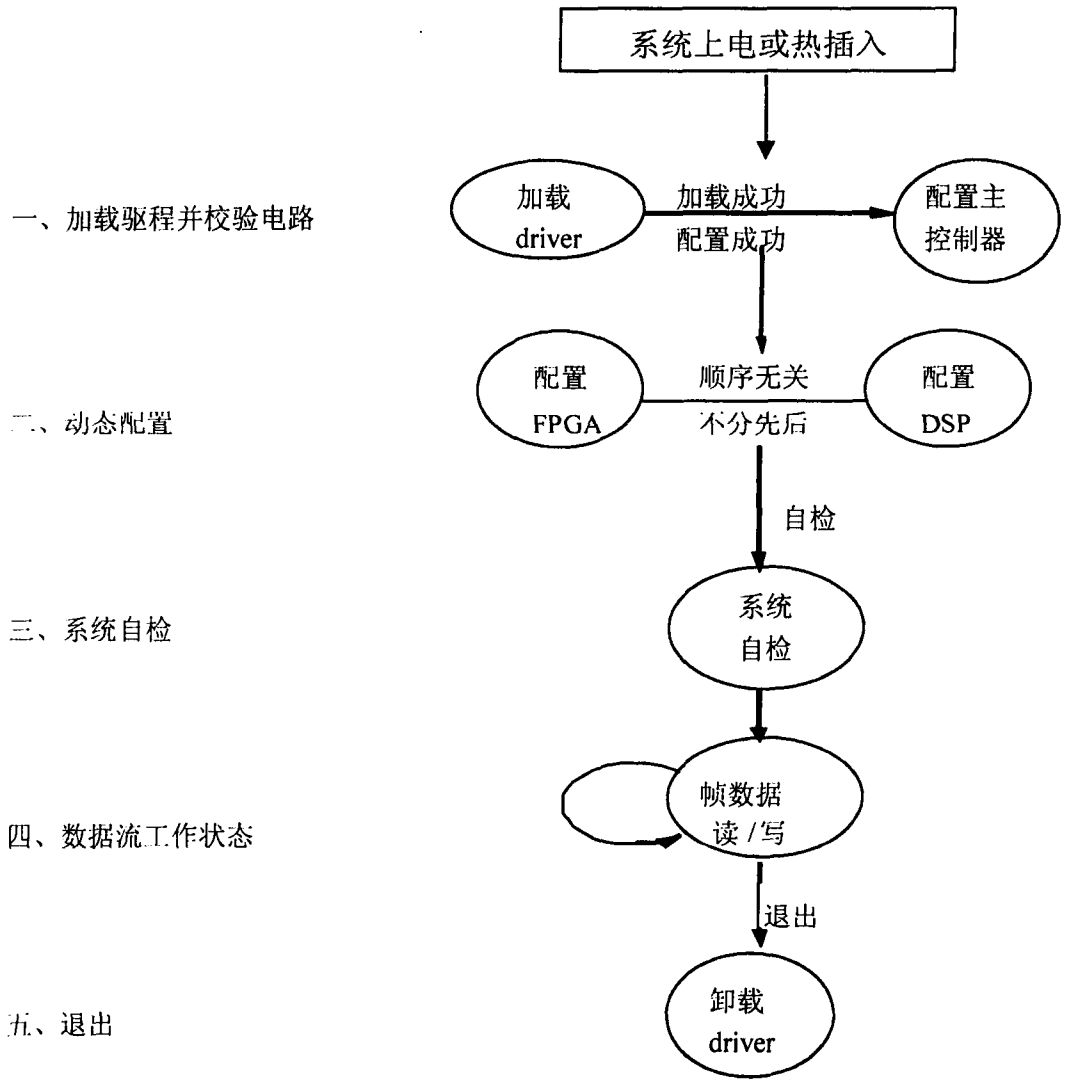

图 4.10 系统软件工作流程

报告自检状态。自检正常后，进入正常工作状态，直至正常工作结束。当板卡热拔出或系统关闭时，卸载 PCI 驱动程序，释放所有资源，正常退出。

PCI 驱动程序和动态重构程序具有有完备的异常处理机制和丰富的错误报告信息，便于开发、调试和分析。

随着新一代超大规模可编程逻辑器件性能的进一步提高，动态可重构技术有了长足发展，在卫星，航天等技术领域的应用也越来越频繁。基于 CPCI 总线动态

可重构技术将以其设计灵活、高性能、高可靠性、极大降低项目研发成本和缩短项目研发周期等技术优势，在通信、工业控制等技术领域得到广泛的应用[12]。

4.4 小结

本章详细分析了 Xilinx 的 MicroBlaze IP 结构，对其开发使用进行了研究；在此基础上，设计了一种基于 MicroBlaze 的 CPCI 动态可重构数据采集系统。

本章设计的基于 MicroBlaze 的 CPCI 动态可重构数据采集系统，在高速数据采集和处理领域得到了实际应用，为数据采集系统的发展提供了新的思路。

结 束 语

本文首先以 Xilinx Virtex-II 系列 FPGA 为例，深入分析了 FPGA 结构特点，在此基础上，详细阐述了 FPGA 设计技术，提出了一种符合现代数字系统设计的、优化设计时序的 Xilinx FPGA 设计流程。然后针对传统电路设计技术在航天惯测系统设计中的局限性，引入了 IP 可复用设计技术，设计了一种基于 FPGA 和 IP 复用技术的航天惯测系统。最后分析了 Xilinx 的 MicroBlaze IP 结构，对其开发使用进行研究，在此基础上，设计了一种基于 MicroBlaze 的 CPCI 动态可重构数据采集系统。

随着 FPGA 结构的不断完善，在单片 FPGA 内部能够完成的工作越来越多，现在已经能够在 FPGA 内嵌入 DSP 处理器，基于 FPGA 的系统设计技术将越来越丰富，在将来的系统设计中，将出现 FPGA 和一些外围的模拟电路组成整个电路系统。因此，基于 FPGA 的系统设计技术是一个不断延伸的、发展的研究方向。

以 FPGA 为核心的 SOPC 设计正在蓬勃发展，将来可以在以 MicroBlaze 构建的系统上架设实时操作系统，进行多任务的管理和控制，实现更高层次的应用。相信基于 FPGA 的系统设计将有更多更高层的应用前景。

致 谢

在论文的完成之际，我要向我的导师徐晖教授致以深深的敬意和诚挚的感谢。徐老师渊博的知识、严谨的治学作风对我产生巨大的影响，使我终生受益。在课题上给予我大量的指导，论文的完成倾注着徐老师的心血。

徐欣副教授在我的课题研究和平时的工作中对我进行了大量而具体的指导，他的丰富的知识和经验给我巨大的帮助，帮助我解决了许多在工作中遇到的具体问题，论文的完成也倾注着徐老师的巨大心血，在此表示衷心的感谢！

我还要感谢周一宇教授，周老师不仅对我的工作给予了指导，周老师的渊博的知识和敏锐的思维，对我的工作也有很大的启发。姜文利教授、卢启中副教授在我的课题研究和平时的工作中对我进行了大量的指导，在此表示衷心的感谢！

另外，感谢黄知涛、邓新浦、吴京等老师，他们也都给我许多具体的帮助。同时，感谢于红旗博士、柳征博士、李俊博士、许丹博士、徐海源博士、谢凯博士、王建东博士、石磊硕士等师兄，他们也都给我细致的帮助。同时，实验室的喻小虎硕士、孙恩元硕士、张建硕士、吴佳硕士、郭加力硕士等同学，两年多来朝夕相处，建立了深厚的友谊，学业上也共同进步，对他们给我的帮助，在这里也一并表示感谢。

最后，我要深深的感谢我的父母和朋友，是他们的理解与支持促使我不断进步，学业的完成也是我对他们的最大回报。

参考文献

[1] 徐欣，温广翔，可编程片上系统设计技术，“2003 嵌入式世界研讨暨展示会论文集”，2003.9

[2] 潘松，黄续业，曾旒，SOPC 技术实用教程，清华大学出版社，2005.3

[3] Michael Keating, Pierre Bricaud, “Reuse Methodology Manual”, Third Edition, Kluwer Academic Publishers,2002

[4] Altera, “Nios 3.0 CPU Data Sheet” Altera Inc., 2003.

[5] Xilinx，“MicroBlaze Processor Reference Guide” EDK (v3.1.2 EA) November 13, 2002

[6] 徐欣，于红旗，卢启中，易凡，基于 FPGA 的嵌入式系统设计，电子工业出版社，2004.10

[7] 周润德等译 Jan M.Rabaey， Anantha Chandrakasan Borivoje Nikolic 著 数字集成电路－电路、系统与设计（第二版） 电子工业出版社，2004.9

[8] 孙航 Xilinx 可编程逻辑器件的高级应用与设计技巧，电子工业出版社 2004.8

[9] Xilinx Inc. PicoBlaze Processor IP Guide 2003

[10] 彭澄廉 挑战 SOC-基于 NIOS 的 SOPC 设计与实践 清华大学出版社 2004.8

[11] 林敏，方颖立，VHDL 数字系统设计与高层次综合 电子工业出版社，2002.1

[12] 高谷刚，罗春，可复用 SPI 模块 IP 核的设计与验证 单片机与嵌入式系统与应用 2005.4

[13] 井新宇，用 FPGA/CPLD 设计 UART，今日电子，2004.1

[14] 廖日坤等，CPLD/FPGA 嵌入式应用技术开发白金手册，中国电力出版社，2005.10

[15] 薛以辉，孙广富，常青，基于 FPGA μP IP 的嵌入式系统，电子产品世界，2003 年第 9 期

[16] 李庆诚，张杰，汤建军， FSL 总线 IP 核及其在 MicroBlaze 系统中的应用，单片机与嵌入式系统应用，2005.6

[17] Xilinx，“Embedded System Tools Guide” EDK (v3.1.2 EA) November 13, 2002

[18] 叶建华，陆农春，黄 凌，基于 CPCI 总线的动态可重构系统。电迅技术，2004 第六期

[19] 刘凯，徐晖，徐欣 基于 CPCI 的动态可可重构数据采集系统。微处理机，

2005 第五期

[20] PLX Technology, PCI 9656 Data Book r0.90. www.plx.com , 2000 .

[21] Altera ,Digital Library. www.altera.com ,2003.

[22] Texas Instruments, TMS320C6713 User S Guide. www.ti.com ,1997.

[23] Texas Instruments, ADS5102 Data Sheet, www.ti.com ,1997.

作者在学期间取得的学术成果

[1] 刘凯，徐欣，徐晖，基于 CPCI 的动态可重构数据采集系统 微处理机 2005 年，第五期