摘要

为了满足现代电子领域对于大容量、高速度存储介质的需要，DDR SDRAM存储介质需要更完善的接口控制模块和更便利的使用方式。本论文所构建的模块系统是一种行之有效的使用手段，已应用于中兴北研所多个传输类项目之中。

本文首先介绍了选题背景、国内外研究现状和论文的主要工作，接着对DDR SDRAM存储设备的原理和发展过程进行简介，并介绍了DDR SDRAM的接口时序，分析了其在系统中的位置、功能和作用，在此基础上提出了设计方案规划。之后着重叙述了基于Stratix-II GX系列FPGA的DDR2接口的FIFO工程设计，对于主控核心单元、数据输入单元和数据缓存单元进行了单独的模块化分析，并且对主要模块进行了功能仿真，归纳问题。接着通过使用chipscope软件平台进行在线调试，分析问题并提出相关关键技术问题及解决方法。

通过本系统模块的开发和调试，实现了一种FIFO特性的存储介质接口装置，便捷了对复杂时序接口的大容量、高速存储介质的应用。

关键词： DDR SDRAM， FPGA， FIFO

Abstract

In order to meet the requirement of high-capacity and high-speed storage medium in electronics, DDR SDRAM has been used more and more, and it needs better interface and more convenient way to use. In this paper, building a modular system is an effective way. It has been used in some projects of ZTE Beijing Corporation.

This article first introduces the background of the topics, including domestic and foreign related research, then introduces the principle and the development process of DDR SDRAM and interface, and analyzes its position and function in the system. Then article introduces the implementation of FIFO character based on Stratix-II GX FPGA, and analyzes the main unit, data entry and data cache unit. Through the software platform chipscope, on-line debugging works for analyzing problems and summarizing some technology issues and solutions.

Through the development and debugging of this system, I realize a transmission character with FIFO feature, and make a convenient interface to the complexity of the timing of high-capacity, high-speed storage medium applications.

Key words: DDR SDRAM, FPGA, FIFO

西安电子科技大学

学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个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中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技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做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名：李原　　日期 2009.1.10

西安电子科技大学

关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。

（保密的论文在解密后遵守此规定）

本学位论文属于保密，在___年解密后适用本授权书。

本人签名：李原　　日期 2009.1.10

导师签名：　　日期 2009.1.10

第一章 绪论

1.1 选题背景

如今，在计算机、通讯、信息处理及其它电子领域内，对高速大容量的存储介质需求是越来越大。在这种势头下，近年来 DDR、DDR2、DDR3 以及 QDR 等高速大容量存储介质迅速发展。

目前，中兴通讯北京研究所的业务类单板已经普遍采用 FPGA 作为业务处理的核心器件，而且随着处理业务越来越复杂，高速度、大容量的 DDR SDRAM 系列存储器也成为许多单板的必选[1]。

DDR SDRAM 即双倍数据率同步动态随机访问存储器，他的优点是存储容量大、成本低、接口成熟，而且并行突发访问时，可以达到较高的访问速率。随着支持时钟频率的升高，依次有 DDR、DDR2、DDR3，三个标准，接口时序基本一致，电气特性相差比较大。

然而，这类存储介质的应用受到一些因素的限制，目前业界对信号的处理方式普遍还是采用时钟单边沿触发，而这类存储介质为了提高数据传送，一般都采用时钟双沿触发的方式。其次，这类存储介质接口时序复杂，数据传送时不可避免的会浪费一些时钟周期。即使采用厂商提供的 IP CORE，在与 FPGA 内部逻辑接口互连时一般也不能实现直接对接。

为此，需要设计一种 FIFO 特性的存储介质接口方法和装置。从而可充分利用现有的技术解决方案对上述问题进行处理，通过引入本装置，实现存储介质具有 FIFO 特性的接口。

这种具有 FIFO 特性的黑盒接口的优势在于以下几点：(1)接口信号非常简单，和 FIFO 接口类似。从而提高了操作性能，减少了管理成本。(2)支持多通道系统，灵活提供用户所需的多通道信息交流。

本课题来源于中兴通讯公司传输事业部 M800 项目组，目的在于开发一种 FIFO 传输特性的存储介质接口方法，实现业务信息交流的更加便捷与快速。

1.2 国内外研究现状

在目前需求量最大的计算机领域，AMD Optero 处理器内核实际上集成了单通道 128-bits 的 DDR 内存控制单元，Athlon 64 处理器内核集成了单通道 64-bitsDDR 内存控制单元。Opter 内核当中的内存总线位宽实际上是 144-bits，其

中 128-bits 用来进行数据传输，16-bits 用来进行 DDR333 状态下的 ECC 错误校验。AMD 一度将这种单通道 128-bits 的内存总线架构称之为“双通道”，AMD 的意思是这种内存带宽在 5.3GB/s 的单通道 128-bits 的内存总线架构，等效于 2 条 Athlon 64 单通道 64-bits 带宽在 2.67GB/s 的内存总线架构[2]。AMD 未来用于移动平台的 Turion 64 处理器将内建 DDR2 内存控制器，以进一步提升整体性能， 同时还会有双核和 VT 技术，而稍微低端一点的 Mobile Athlon 64 也将会内建 DDR2 内存控制器，但不会有双核心的版本，而未来最低端的 Mobile Sempron 也将内建 DDR2 内存控制器，同时支持 64bit 处理能力，这将大幅提升下一代 AMD 移动平台的内存带宽和整体性能。

在嵌入式系统领域，Alter 与 Northwest Logic 宣布为 Alter 的高密度 Stratix II 与 Stratix II GX FPGA，提供经过硬件验证的 667-Mbps DDR2 SDRAM 接口，这个接口结合了 Alter 的自动校准 DDR2 PHY 与 Northwest Logic 的全功能 DDR2 SDRAM 控制器核心，在最高的内存传输量时，可大幅地简化 DDR2 SDRAM 的接口设计。

Alter 的 DDR2 PHY 已经过最佳化，以便在处理、电压与温度变动过程中提供卓越的性能，它可支持完整的技术文件、软件与工具、IP 核心、展示版、特性描述报告与仿真模型等组合，都可用于协助设计师成功地进行 Altera FPGA 到 DDR2 SDRAM 的接口设计。Altera 的自动校准 DDR2 PHY 现在已可透过初期客户合作计划来供应，客户可以联系他的业务代表来参与这个计划。

Northwest Logic 的 DDR2 SDRAM 控制器核心是高效能、易于使用的内存控制器核心产品系列的一部分，能够支持 DDR2、DDR、可携式 DDR、SDR、可携式 SDR SDRAM，以及可减少延迟的 DRAM II (RLDRAM II)内存，DDR2 SDRAM 控制器核心可透过使用请求重新排序、记忆库管理与预看(look-ahead)处理，来提供高总线效率。

Northwest Logic 也提供错误校正码(Error Correction Code，ECC)、读取-修改-写入(Read-Modify-Write)，以及多埠前端(Multi-Port Front-End)附加模块，以进一步地简化使用者的设计工作。IP 核心支持最高的内存频率速率，只需要最少的逻辑闸数，并搭配完整的文件与验证套装[3]。

中兴通讯开发的传输设备是针对城域网、金融网、政府网和军网、企业网等市场需求而开发的产品，可向用户提供更加安全、可控、可管理的高性能传输解决方案[4]。中兴北京研究所的业务传输类单板的核心器件是 FPGA，为了处理复杂高速的业务信息，需要使用大容量，高速度的 DDR 存储器件，为此，需要设计适合于业务类单板的具有 FIFO 特性的 DDR 系列接口模块[5]。

1.3 本论文主要研究工作

本人首先根据实际的项目需求，分析了在已有的硬件环境上开发 FIFO 特性 DDR 接口模块的可行性，提出了功能设计要求和设计方案，实现了接口开发和仿真调试。并且在新建项目的硬件环境下开发了 FIFO 特性的 DDR2 接口控制模块。

1.3.1 本文的主要研究工作

在开发接口控制模块的过程中，首先需要了解 DDR 存储器件的原理以及在设备中的位置和作用，然后需要分析业务功能需求和自身开发的硬件结构，确立需要设计开发的功能，最后根据实际需求进行开发。

本文所做的主要工作：

1、进行接口模块的需求和功能分析。

2、研究 DDR 存储介质以及相关 IP CORE 原理。

3、根据实际需要，对接口模块实现功能进行了硬件抽象层的描述，并给出了设计和实现的解决方案。

4、对功能模块进行编码驱动开发。

5、对功能模块进行仿真，调试。

6、分析和解决关键技术问题。

1.3.2 本文的组织结构

本文的组织结构如下：

第一章 根据中兴通讯传输产品对 DDR SDRAM 器件的需求现状，介绍选题背景、国内外研究现状和论文的主要工作。

第二章 首先对 DDR SDRAM 存储设备的原理和发展过程进行简介。然后介绍了 DDR SDRAM 的接口描述，对其在系统中的位置、功能和作用进行了分析，提出了设计中的功能要点。并在此基础上提出了系统的设计规划。

第三章 对基于 Stratix-II GX 系列 FPGA 的 DDR2 接口的 FIFO 设计提出具体实施方案，对于主控核心单元、数据输入单元和数据缓存单元进行了单独的模块化分析和实现方案，并且对主要模块进行了功能仿真，归纳问题。

第四章 通过使用 chipscope 软件平台进行在线调试，分析问题并提出相关关键技术问题及解决方法。

第五章 全文总结以及工作展望。

第二章 DDR SDRAM 原理以及系统设计方案

2.1 DDR SDRAM 的发展过程

2.1.1 内存体系结构发展历程

作为 PC 不可缺少的重要核心部件——内存，它伴随着 DIY 硬件走过了多年历程。从 286 时代的 30pin SIMM 内存、486 时代的 72pin SIMM 内存，到 Pentium 时代的 EDO DRAM 内存、PII 时代的 SDRAM 内存，到 P4 时代的 DDR 内存和目前 9X5 平台的 DDR2 内存。内存从规格、技术、总线带宽等不断更新换代。不过我们有理由相信，内存的更新换代可谓万变不离其宗，其目的在于提高内存的带宽，以满足 CPU 不断攀升的带宽要求、避免成为高速 CPU 运算的瓶颈。

在 80286 主板发布之前，内存并没有被世人所重视，这个时候的内存是直接固化在主板上，而且容量只有 64 ～256KB，对于当时 PC 所运行的工作程序来说，这种内存的性能以及容量足以满足当时软件程序的处理需要。不过随着软件程序和新一代 80286 硬件平台的出现，程序和硬件对内存性能提出了更高要求，为了提高速度并扩大容量，内存必须以独立的封装形式出现，因而诞生了我们所提到的“内存条”概念。

在 80286 主板刚推出的时候，内存条采用了 SIMM（Single In-lineMemory Modules，单边接触内存模组）接口，容量为 30pin、256kb，必须是由 8 片数据位和 1 片校验位组成 1 个 bank，正因如此，我们见到的 30pin SIMM 一般是四条一起使用。自 1982 年 PC 进入民用市场一直到现在，搭配 80286 处理器的 30pin SIMM 内存是内存领域的开山鼻祖。

自 Intel Celeron 系列以及 AMD K6 处理器以及相关的主板芯片组推出后，EDO DRAM 内存性能再也无法满足需要了，内存技术必须彻底得到个革新才能满足新一代 CPU 架构的需求，此时内存开始进入比较经典的 SDRAM 时代[6]。

第一代 SDRAM 内存为 PC66 规范，但很快由于 Intel 和 AMD 的频率之争将 CPU 外频提升到了 100MHz，所以 PC66 内存很快就被 PC100 内存取代，接着 133MHz 外频的 PIII 以及 K7 时代的来临，PC133 规范也以相同的方式进一步提升 SDRAM 的整体性能，带宽提高到 1GB/sec 以上。由于 SDRAM 的带宽为 64bit，正好对应 CPU 的 64bit 数据总线宽度，因此它只需要一条内存便可工作，便捷性进一步提高。在性能方面，由于其输入输出信号保持与系统外频同步，因此速度明显超越 EDO 内存。

不可否认的是，SDRAM 内存由早期的 66MHz，到后来的 100MHz、133MHz，

尽管没能彻底解决内存带宽的瓶颈问题，但此时 CPU 超频已经成为 DIY 用户永恒的话题，所以不少用户将品牌好的 PC100 品牌内存超频到 133MHz 使用以获得 CPU 超频成功，值得一提的是，为了方便一些超频用户需求，市场上出现了一些 PC150、PC166 规范的内存。

DDR 是一种继 SDRAM 后产生的 内存技术，DDR，英文原意为“Double Data Rate”，顾名思义，就是双数据传输模式。之所以称其为“双”，也就意味着有“单”，我们日常所使用的 SDRAM 都是“单数据传输模式”。DDR SDRAM 最早是由三星公司于 1996 年提出，由日本电气、三菱、富士通、东芝、日立、德州仪器、三星及现代等八家公司协议订立的内存规格，并得到了 AMD、VIA 与 SiS 等主要芯片组厂商的支持。

这种内存的特性是在一个内存时钟周期中，在一个方波上升沿时进行一次操作（读或写），而 DDR 则引用了一种新的设计，其在一个内存时钟周期中，在方波上升沿时进行一次操作，在方波的下降沿时也做一次操作，之所以在一个时钟周期中，DDR 则可以完成 SDRAM 两个周期才能完成的任务，所以理论上同速率的 DDR 内存与 SDR 内存相比，性能要超出一倍，可以简单理解为 100MHZ DDR=200MHZ SDR。

SDRAM 在一个时钟周期内只传输一次数据，它是在时钟的上升期进行数据传输；而 DDR 内存则是一个时钟周期内传输两次次数据，它能够在时钟的上升期和下降期各传输一次数据，因此称为双倍速率同步动态随机存储器。DDR 内存可以在与 SDRAM 相同的总线频率下达到更高的数据传输率。

与 SDRAM 相比：DDR 运用了更先进的同步电路，使指定地址、数据的输送和输出主要步骤既独立执行，又保持与 CPU 完全同步；DDR 使用了 DLL(Delay Locked Loop，延时锁定回路提供一个数据滤波信号)技术，当数据有效时，存储控制器可使用这个数据滤波信号来精确定位数据，每 16 次输出一次，并重新同步来自不同存储器模块的数据。DDL 本质上不需要提高时钟频率就能加倍提高 SDRAM 的速度，它允许在时钟脉冲的上升沿和下降沿读出数据，因而其速度是标准 SDRA 的两倍。

从外形体积上 DDR 与 SDRAM 相比差别并不大，他们具有同样的尺寸和同样的针脚距离。但 DDR 为 184 针脚，比 SDRAM 多出了 16 个针脚，主要包含了新的控制、时钟、电源和接地等信号。DDR 内存采用的是支持 2.5V 电压的 SSTL2 标准，而不是 SDRAM 使用的 3.3V 电压的 LVTTL 标准[7]。

2.1.2 DDR SDRAM 的体系结构

随着支持时钟频率的升高，依次有 DDR、DDR2、DDR3，三个标准，接口时序基本一致，电气特性相差比较大。

DDR内存采用184线结构，DDR内存不向后兼容SDRAM，要求专为DDR设计的主板与系统。

DDR2 内存将是现有 DDR1 内存的换代产品，它们的工作时钟预计将为400MHz或更高（包括现代在内的多家内存商表示不会推出DDR2 400的内存产品）。从JEDEC组织者阐述的DDR2标准来看，针对PC等市场的DDR-II内存将拥有400-、533、667MHz等不同的时钟频率。

高端的DDR2内存将拥有800-、1000MHz两种频率。DDR2内存将采用200-、220-、240-针脚的FBGA封装形式。最初的DDR2内存将采用0.13微米的生产工艺，内存颗粒的电压为1.8V，容量密度为512MB。 DDR2将采用和DDR1内存一样的指令，但是新技术将使DDR2内存拥有4到8路脉冲的宽度。DDR2将融入CAS、OCD、ODT等新性能指标和中断指令。DDR2标准还提供了4位、8位512MB内存1KB的寻址设置，以及16位512MB内存2KB的寻址设置。

DDR2内存标准还包括了4位预取数（pre-fetch of 4 bits）性能，DDR1技术的预取数位只有2位。

DDR3 的市场导入时间预计为 2006 年，最高数据传输速度标准将达到1600Mbps。不过，就具体的设计来看，DDR3与DDR2的基础架构并没有本质的不同。从某种角度讲，DDR3是为了解决DDR2发展所面临的限制而催生的产物。

由于 DDR2 的数据传输频率发展到 800MHz 时，其内核工作频率已经达到200MHz，因此再向上提升较为困难，这就需要采用新的技术来保证速度的可持续发展性。另一方面，也是由于速度提高的缘故，内存的地址/命令与控制总线需要有全新的拓朴结构，而且业界也要求内存要具有更低的能耗，所以，DDR3 要满足的需求就是：更高的外部数据传输率；更先进的地址/命令与控制总线的拓朴架构；在保证性能的同时将能耗进一步降低[8]。

2.2 DDR SDRAM 的原理

2.2.1 DDR SDRAM 概述

DDR SDRAM是一种采用双沿触发结构的SDRAM，双沿触发结构本质上是一种2n预取结构，在输入/输出管脚上，每个时钟周期传输两个字节。对于一次DDR SDRAM访问操作（读操作、写操作），在芯片内部，一个时钟周期执行一次位宽为2n的数据传输；在DDR SDRAM的管脚上，每半个时钟周期执行一次n位宽的数据传输。因此，DDR SDRAM的总线带宽为：工作频率×数据总线宽度×2。

DDR SDRAM 具有如下特点[9]：

1. 采用双沿触发结构，每个时钟周期传输两个数据；
2. 相向数据触发信号（DQS）与数据一起传输，用于接收侧锁存数据；
3. 读操作时，DQS 与数据边沿对齐，写操作时，DQS 与数据中心对齐；
4. 差分时钟（CK、CK#）输入；
5. DLL 将 DQ 和 DQS 的跳变与 CK 的调变对齐；
6. 在时钟 CK 的上升沿接收命令字，数据和数据屏蔽位的接收，参考 DQS 的上升沿和下降沿；
7. 内部分为 4 个 bank；
8. 写操作时，支持写数据屏蔽功能；
9. 突发程度可设，支持的突发长度为 2，4，8；
10. CAS 延迟为 2 或者 2.5，DDR 400 的 CAS 延迟还可设为 3；
11. 每次突发操作支持 AUTO PRECHARGE 功能；
12. 支持 Auto Refresh 和 Self Refresh 两种刷新模式；

2.2.2 DDR 接口管脚说明

DDR 接口管脚如表 2.1 所示：

表 2.1 DDR 芯片接口表

名称	方向	有效电平	位宽(bit)	说明
CK，CK#	输入	时钟	1	时钟信号，CK、CK#是差分时钟输入，所有的地址和控制输入信号都在 CK 的上升沿和 CK#下降沿的交叉处被锁存；输出数据以 CK 和 CK#的交叉口为参考数据（双沿输出）。
CKE （CKE0） （CKE1）	输入	高	1	时钟使能：CKE 为“高”，使能内部时钟信号、输入缓存、输出驱动；CKE 为“低”，DRAM 进行 PRECHARGE、POWER_DOWN、SELF REFRESH、ACTIVE POWER_DOWN 操作。对于 POWER_DOWN 进入和退出操作；SELF REFRESH 进入操作，CKE 是同步信号；对于退出 SELF REFRESH 操作，CKE 是异步信号。在读/写操作的过程中，CKE 必须一直保持为高。

CS# （CS0#） （CS1#）	输入	低	1	片选信号。当 CS#为高时，所有的命令都是无效的。
RAS#，CAS# WE#	输入	低		命令信号输入，RAS#、CAS#、WE#和 CS#一起确定输入的命令。
DM （LDM） （UDM）	输入			输入数据屏蔽位。在写操作过程中，当 DM 为高时，输入数据将会被屏蔽。DM 在 DQS 的双沿进行采样。对 16bit 位宽芯片，LDM 对应 DQ0-DQ7；UDM 对应 DQ8-DQ15。在读操作过程中，DM 可以为高、低或者悬空状态。
BA0，BA1	输入	总线	2	Bank 地址输入，BA0 和 BA1 确定对哪个 Bank 执行 ACTIVE、READ、WRITE、PRECHARGE 操作
A0-A13	输入	总线	14	地址输入。为 ACTIVE 命令，提供行地址；为读/写命令提供列地址和 AUTO PRECHARGE 位。A10 只在 precharge 命令时被采样，当 A10 为“低”时，对一个 bank 进行 PRECHARGE 操作；当 A10 为“高”时，对所有 bank 进行 PRECHARGE 操作。如果只对一个 bank 进行 PRECHARGE 操作，则 BA0，BA1 确定进行操作的 bank。
DQ	双向	总线		数据总线
DQS （LDQS） （UDQS）	双向			数据总线触发，读操作时，DQS 为输出，写操作时，DQS 为输入。读操作时，DQS 与数据数据边沿对齐；写操作时，DQS 与写入数据中心对齐。
NC				无连接管脚
VDDQ				DQ 工作电压
VSSQ				DQ 工作地
VDD				供电电压，
VSS				地
VREF		输入		SSTL_2 参考电压

2.2.3 DDR 上电初始化过程

DDR SDRAM 使用前需要初始化。其流程图如图 2.1 所示：

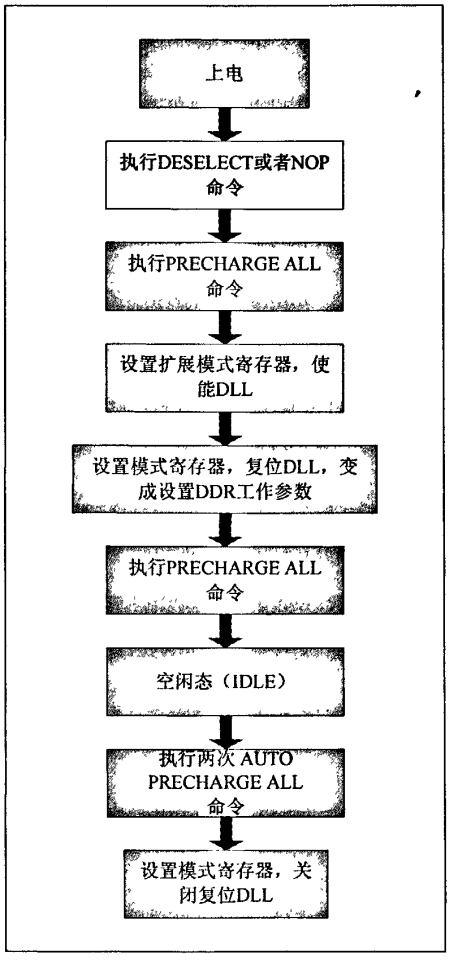


图 2.1 DDR SDRAM 初始化流程图

在供电电压、参考电压、工作时钟稳定之后，DDR SDRAM 必须等待 200us 才能执行各种操作。等待 200us 之后，CKE 必须为高，DDR SDRAM 需要执行 DESELECT 或者 NOP 操作，执行完 NOP 操作后，DDR SDRAM 执行 PRECHARGE ALL 操作。

扩展模式寄存器设置主要是使能 DLL；模式寄存器设置主要是复位 DLL，设置 DDR SDRAM 工作参数，在复位 DLL 之后，需要等待 200 个时钟周期，才能执行 PRECHARGE ALL 命令。进行完上述操作之后，DDR SDRAM 的所有 bank 处于空闲态（IDLE）。

处于空闲态后，DDR SDRAM 必须执行两次 AUTO PRECHARGE 操作，然后，设置模式寄存器，关闭 DLL 复位功能，执行完上述操作之后，DDR SDRAM 已经可以进行读/写等正常操作[10]。

2.2.4 DDR 状态控制图

DDR SDRAM 控制器简化状态转换图如图 2.2 所示。

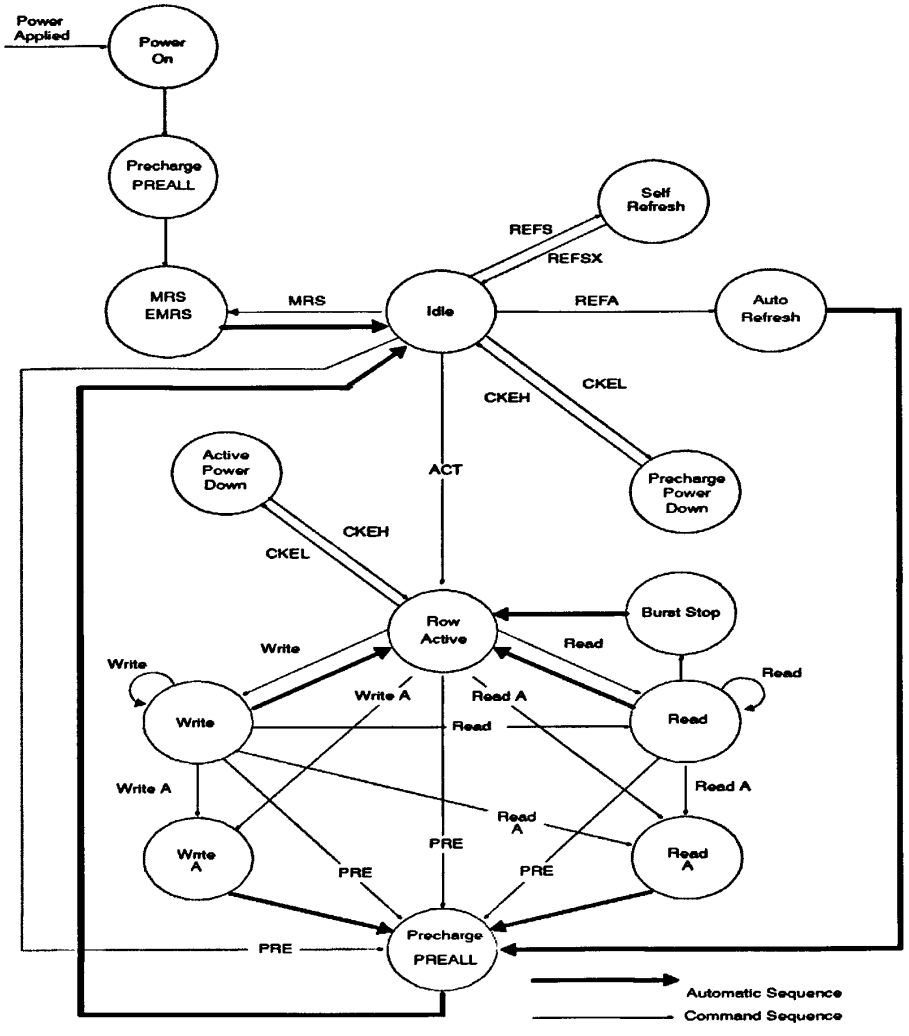


图 2.2 DDR SDRAM 控制器简化状态转换图[11]

PREALL = Precharge All Banks		CKEL	= Enter Power Down
MRS	= Mode Register Set	CKEH	= Exit Power Down
EMRS	= Extended Mode Register Set	ACT	= Active
REFS	= Enter Self Refresh	Wrtie A	= Write with Autoprecharge
REFSX	= Exit Self Refrensh	Read A	= Read with Autoprecharge

2.3 接口模块开发方案

2.3.1 接口模块需求

目前我们应用中，FPGA 内部通常缓存数据时，都是通过 FIFO 实现。对于有多个通道接入的单板，可能同时需要多个并行 FIFO 缓存数据，而这些 FIFO 对应的存储器则是外部的 DDR SDRAM。功能如下[12]：

1. 实现与 DDR2 IP Core 用户接口的对接时序
2. 实现与逻辑的 FIFO 接口
3. 最多支持将 DDR2 存储器划分为 8 个独立的存储空间，实现 8 个独立的 FIFO
4. DDR2 读写效率高于 75%（即有效的数据吞吐量达到时钟频率的 1.5 倍）

2.3.2 接口模块设计规划

本系统提供了一种实现无时隙数据传输的存储介质接口设计方法及其装置，通过把这类存储介质接口时序映射为 FIFO 接口时序，实现了无时隙的数据交流传输。

为解决功能要求，系统提供了如下的解决方法。该方法和装置包括以下几个部分：数据缓存单元，状态监控单元，核心控制单元，输入数据组装单元，输出数据组装单元，存储介质控制单元。系统框图如图 2.3 所示[13]：

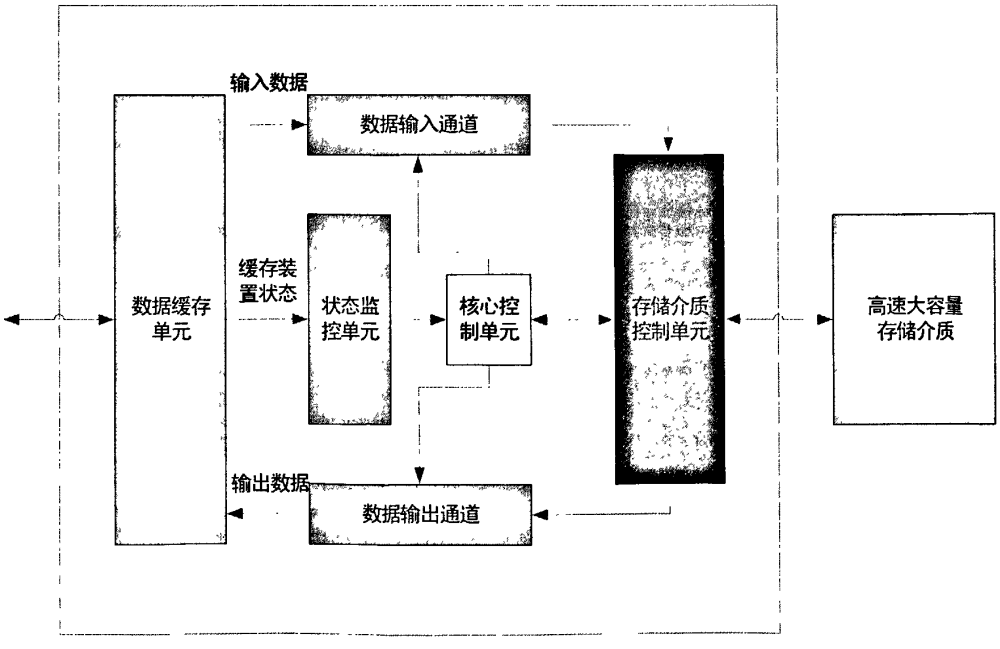


图 2.3 系统框图

数据缓存单元：输入的数据宽度及时钟频率可能与系统内部不一致，需要用缓存单元进行转换。本单元对外接口主要包括输入输出的数据接口、使能控制接口以及数据缓存单元的状态信号。

状态监控单元：通过采集数据缓存装置的状态，进行处理并给核心控制单元发送相关信息。

核心控制单元：采集状态信息，分别对数据输入及输出通道进行操作，并与存储介质控制单元进行信息交互。

数据输入通道：采集数据缓存单元数据，并匹配成存储介质控制单元的接口时序。

数据输出通道：接收存储介质控制单元数据，并匹配到数据缓存单元中。

存储介质控制单元：映射存储介质接口时序，并与核心模块进行信息交流，完成对存储介质的数据交互。

2.3.3 软件实施方案

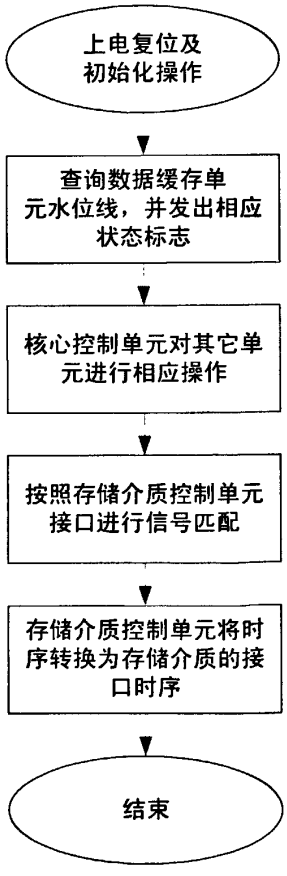


图 2.4 软件流程图

由软件流程图 2.4 所示，上电复位之后，存储介质控制单元需要对存储介质进行初始化操作，配置时钟单元，然后等待初始化完成。

在初始化完成后，状态监控单元查询数据缓存单元状态信息，并计算存储介质的状态信息，当满足输入或输出的条件后将产生相应状态标志。将状态标志传送到核心控制单元。

核心控制单元在接收到状态监控单元提供的状态标志后，会触发相关操作。

各单元在核心控制单元的控制下相互配合，与存储介质控制单元进行信息交流，完成数据的输入输出，地址信号以及使能控制信号的匹配。

存储介质控制单元需要把出其它单元传送的信息转换成存储介质能识别的接口时序。与存储介质进行数据交互。

第三章 基于StratixIIGX系列FPGA的DDR2接口的FIFO工程设计

3.1 系统概述

3.1.1 系统结构

系统设计要求主要包括四点：(1)实现与DDR2 IP Core用户接口的对接时序；(2)实现与逻辑的FIFO接口；(3)最多支持将DDR2存储器划分为8个独立的存储空间，实现8个独立的FIFO；(4)DDR2读写效率高于75%(即有效的数据吞吐量达到时钟频率的1.5倍)。

为了达到设计要求，本设计实现了DDR用户接口与多个FIFO接口之间的时序转换，基本设计框图如图3.1所示。

DDR2_FIFO模块主要由状态控制模块(ddr2_state_ctrl)，地址产生模块(ddr2_addr_gen)，写方向数据选择控制(wr_data_sel)，读方向数据选择控制(rd_data_sel)等几个模块组成。

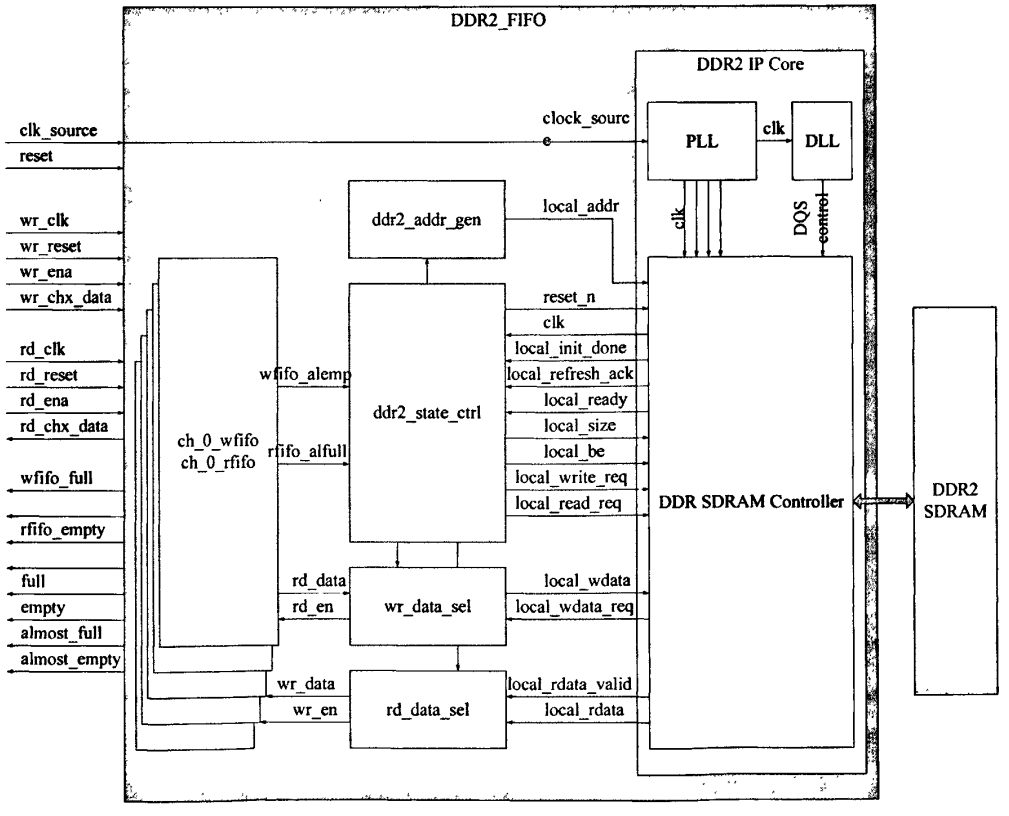


图3.1 DDR2 FIFO接口实现框图

3.1.2 Altera DDR2 IP Core 简介

Altera 提供了 DDR2 接口的 IP Core（DDR2 SDRAM Controller），用以实现 DDR 数据采样及内存读写状态控制。其接口时序如下图 3.2 所示[14]：

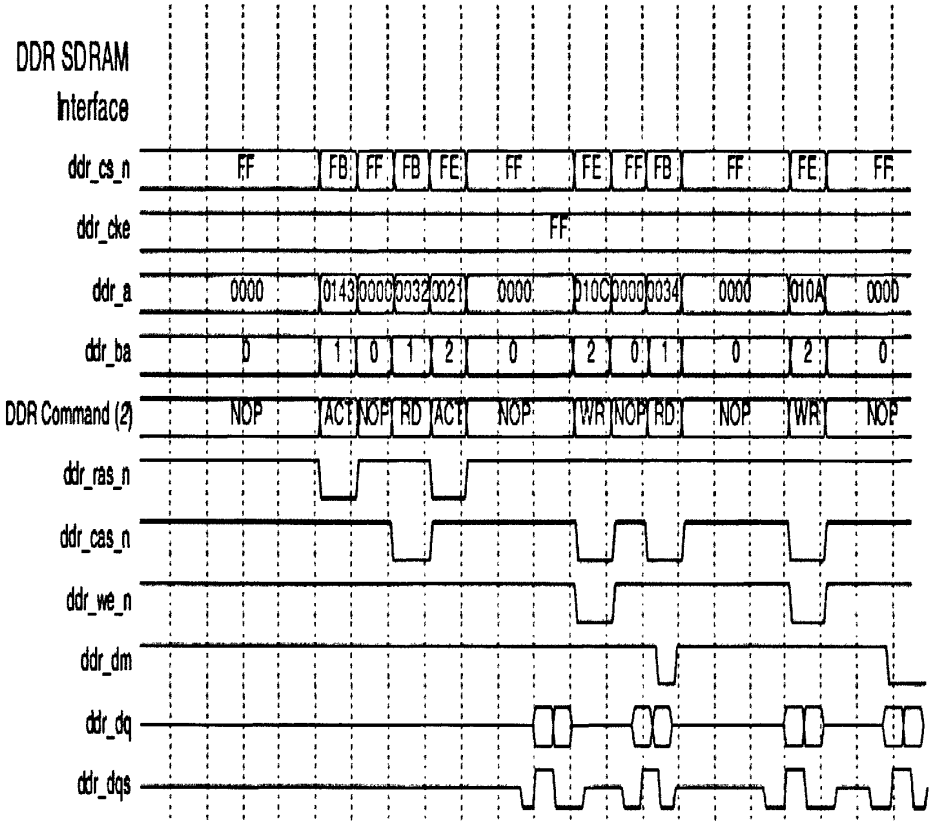


图 3.2 IP CORE 接口时序图

关于 IP CORE 的一些说明如下：

1. 生成 DDR2 SDRAM Controller IP Core 时，一些参数设置如表 3.1 所示：

表 3.1 IP CORE 设置参数表[15]

名称	设置值	说明
Clock Speed	200	DDR2 接口时钟频率 建议至少 200MHz，以提高数据的吞吐量。
Data bus width	72	数据位宽，根据具体硬件确定。FCA 为 72bit
Use dedicated PLL	False	是否使用专用的 PLL 输出时钟采样数据，如果选择此项，会多使用一个 PLL。

Local interface	Native	选择使用Native模式
ODT Setting	50ohm	使用片内匹配电阻
Memory DLL enable	True	使用DLL产生不同相位时钟
Use feedback clock	False	是否使用外部反馈时钟校准，要根据实际电路来确定。一般选False
User controlled refresh	False	由控制器来完成刷新操作，用户不必手动控制。

2. DDR2 IP CORE接口信号，如表3.2所示：

表3.2 DDR2 IP CORE接口信号表

端口名称	方向	位宽(bit)	有效电平	含义
clk	输入	1	时钟	系统时钟
addrcmd_clk	输入	1	时钟	地址及命令寄存器同步时钟，默认使用系统时钟。
clk_to_sdram	输出	3	时钟	输出到DDR2的时钟正，位宽由实际原理图确定，SAU单板为3。
clk_to_sdram_n	输出	3	时钟	输出到DRR2的时钟负，位宽由实际原理图确定，SAU单板为3
ddr2_a[12:0]	输出	13		DDR2地址位宽
ddr2_ba[1:0]	输出	2		DDR2列地址
ddr2_cas_n	输出	1	低	DDR2列片选
ddr2_cke	输出	1		DDR2时钟有效
ddr2_cs_n	输出	1	低	DDR2片选，SAU单板位宽为1
ddr2_dm[7:0]	输出	8		DDR2数据写Mask
ddr2_dq[71:0]	双向	72		DDR2接口数据
ddr2_dqs[8:0]	双向	9		DDR2接口数据锁存
ddr2_odt	输出	1		ODT使能信号
ddr2_ras_n	输出	1	低	DDR2行片选
ddr2_we_n	输出	1	低	DDR2写有效信号
dqs_delay_ctrl[5:0]	输入	6		DQS延时控制信号
dqsupdate	输入	1		控制DLL输出DQS的延时更新

local_addr[23:0]	输入	24		用户接口地址（bank+raw+col-1）
local_be[17:0]	输入	18		写数据 byte 有效信号
local_init_done	输出	1		初始化完成标志
local_rdata[143:0]	输出	144		控制器的输出数据
local_rdata_valid	输出	1		输出数据有效信号
local_rdvalid_in_n	输出	1	低	早期版本 ipcore 读数据有效信号，不使用
local_read_req	输入	1		读请求
local_ready	输出	1		控制器可接受读写请求的标志
local_refresh_ack	输出	1		表示控制器完成一次刷新操作
local_size[1:0]	输入	2		突发读写长度，DDR2 固定为 2
local_wdata[143:0]	输入	144		给控制器的输入数据
local_wdata_req	输出	1		控制器写数据请求标志，要在下一周期给出数据。
local_write_req	输入	1		写请求
reset_n	输入	1	低	系统复位
resynch_clk	输入	1	时钟	将读数据从 DQS 时钟域同步到系统时钟域的同步时钟。通常情况下，可以使用系统时钟。
stratix_dll_control	输出	1		DLL 的 DQS 延时刷新控制信号
write_clk	输入	1	时钟	写时钟，与写数据中心对齐。

3. PLL 接口信号，如表 3.3 所示：

表 3.3 PLL 接口信号表

端口名称	方向	位宽 (bit)	有效电平	含义
inclk0	输入	1	时钟	PLL 输入时钟，实际频率可能低于 DDR 接口时钟和系统时钟的频率
c0	输出	1	时钟	PLL 输出 0 度时钟，系统时钟
c1	输出	1	时钟	PLL 输出 270 度时钟，写时钟
c2	输出	1	时钟	PLL 输出用于读同步的时钟，若使用则多一个全局时钟，通常为 0 度（即系统时钟）。

c3	输出	1	时钟	PLL 输出，专用的后同步码时钟。不使用
locked	输出	1	高	PLL 锁定指示

4. DLL 接口信号，如表 3.4 所示：

表 3.4 DLL 接口信号表

端口名称	方向	位宽 (bit)	有效电平	含义
addnsub	输入	1		接低
reset_n	输入	1	低	DLL 复位
clk	输入	1	时钟	DLL 输入时钟，应该为布线延时反馈的时钟或系统时钟。
delayctrlout	输出	6		控制输入 DQS 信号的延时
dqsupdate	输出	1		DLL 输出 DQS 的延时更新
stratix_dll_control	输出	1		DLL 输出 DQS 的延时更新
offset	输入	6		

使用 IP CORE 生成工具，设置需要的参数，即可生成上述三个子模块。其中 PLL 模块是用来产生模块间需要的各种时钟；DLL 模块是为了实现系统对 DQS 信号的延时；DDR SDRAM Controller 模块实现了系统与 DDR 芯片的信号时序转换以及对 DDR 芯片的控制。通过把这三个子模块封装在一起，合成 DDR2 IP CORE 模块，从而能够方便使用[16]。

3.1.3 状态控制模块（ddr2_state_ctrl）

本模块主要实现与 DDR2 Core 之间的时序转换，控制执行读或写操作，并产生通道选择信号。

1. 本模块接口信号，如表 3.5 所示：

表 3.5 ddr2_state_ctrl 模块接口信号表

端口名称	方向	位宽 (bit)	有效电平	含义
clk	输入	1	时钟	ddr2 core 系统时钟输出

reset	输入	1	高	全局异步复位信号
ddr_reset	输入	1	高	外部输入，复位 ddr2 内存及相关控制逻辑
ch_reset	输入	8		通道复位
ch_reset_req	输出	8		通道复位响应
local_be[17:0]	输出	18		写数据 byte 有效信号，始终有效，置全“1”
local_size[1:0]	输出	2		突发读写长度，DDR2 固定为 2
local_init_done	输入	1	高	初始化完成标志
local_ready	输入	1	高	控制器可接受读写请求的标志
local_read_req	输出	1	高	读请求
local_write_req	输出	1	高	写请求
local_refresh_ack	输入	1		表示控制器完成一次刷新操作
write_addr_en	输出	1		写请求提前一周期信号，用于写地址产生
read_addr_en	输出	1		读请求提前一周期信号，用于写地址产生
local_wdata_req	输入	1	高	控制器写数据请求标志，要在下一周期给出数据
local_rdata_valid	输入	1		控制器输出数据有效信号
ch_sel[7:0]	输出	8	高	通道选择信号，表示当前对某个通道进行读或写操作
wr_ch_sel[7:0]	输出	8	高	数据通道选择信号
wfifo_almost_empty[7:0]	输入	8	高	写 fifo 将空标志
rfifo_almost_full [7:0]	输入	8	高	读 fifo 将满标志
almost_full[7:0]	输入	8	高	ddr 通道将满标志
almost_empty[7:0]	输入	8	高	ddr 通道将空标志

2. 本模块实现方法，如图 3.3 所示：

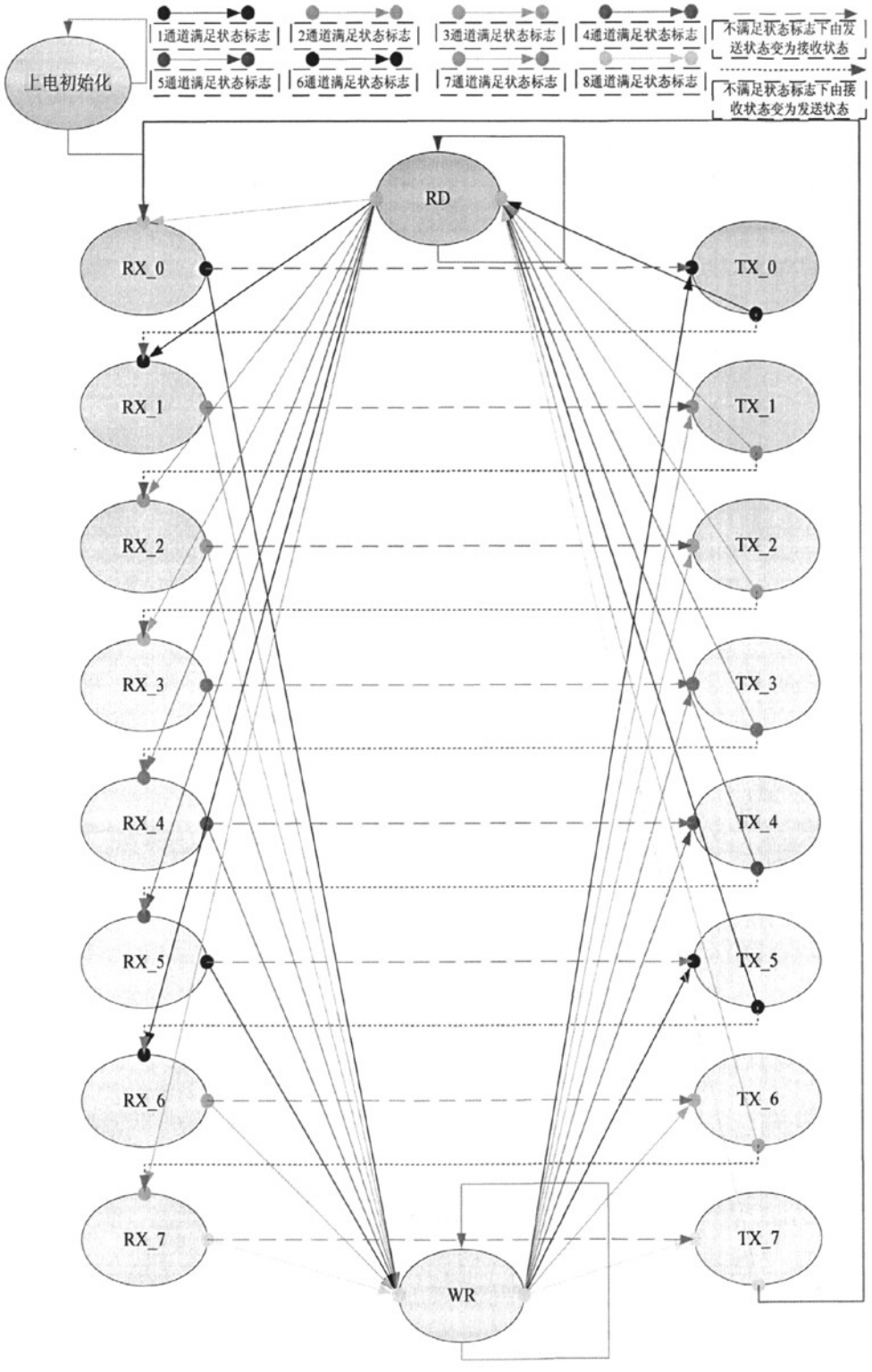


图 3.3 核心模块状态转换图

本模块的实现原理如下：主控状态机通过轮循察看各通道，当发现数据缓存

单元数据量大于一定值而且相应的 DDR 通道内可存储的数据空间小于一定值时，触发此通道的写操作；当发现数据缓存单元内可读取的数据量小于一定值而且相应的 DDR 通道内可读取的数据量大于一定值时，触发此通道的读取操作。通过上图所示的轮询方式，保证各个通道始终处在一种动态平衡的状态，只要外界信息正常，数据信息就不会溢出，也不会被读空。

3.1.4 地址产生模块（ddr2_addr_gen）

本模块实现两个功能，其一是产生 DDR IP CORE 所需要的地址信号，其二是产生 ddr 将空或将满的标志信号，用于 ddr2_state_ctrl 模块进行通道转换。

1. 模块接口信号，如表 3.6 所示：

表 3.6 ddr2_addr_gen 模块接口信号表

端口名称	方向	位宽 (bit)	有效电平	含义
clk	输入	1	时钟	ddr2 core 系统时钟输出
reset	输入	1	高	全局异步复位信号
ch_reset	输入	8		通道复位
local_ready	输入	1		控制器可接受读写请求的标志
write_addr_en	输入	1		写请求提前一周期信号，用于写地址产生
read_addr_en	输入	1		读请求提前一周期信号，用于写地址产生
ch_sel[7:0]	输入	8	高	通道选择信号，表示当前对某个通道进行读或写操作
local_addr[22:0]	输出	23		输入 IP CORE 的 2 位 bank 地址 + 13 位行地址 + 8 位列地址
full[7:0]	输出	8		ddr 各通道满标志
empty[7:0]	输出	8		ddr 各通道空标志
almost_full[7:0]	输出	8		ddr 各通道将满标志
almost_empty[7:0]	输出	8		ddr 各通道将空标志

2. 实现方法

通过调用一个生成地址信号的子模块，在 ch_sel[7:0]信号的选择下，实现对 8 个通道的地址调配。子模块接口信号，如表 3.7 所示：

表3.7 地址内部子模块接口信号表

端口名称	方向	位宽(bit)	有效电平	含义
clk	输入	1	时钟	ddr2 core 系统时钟输出
reset	输入	1	高	全局异步复位信号
ch_sel_en	输入	1	高	通道选择使能信号
write_addr_en	输入	1		写请求提前一周期信号，用于写地址产生
read_addr_en	输入	1		读请求提前一周期信号，用于写地址产生
full	输出	1		某一通道写满标志
empty	输出	1		某一通道读空标志
almost_full	输出	1		某一通道写将满标志
almost_empty	输出	1		某一通道读将空标志
addr[22:0]	输出	23		某一通道的地址

本模块在write_addr_en信号的驱动下，配合ch_sel_en信号的选择作用，输出本通道的地址信号，并随时向外界报告本通道内部的数据量空满状态。

3.1.5 写方向数据选择（wr_data_sel）和 读方向数据选择（rd_data_sel）

这两个模块实现FIFO模块与DDR IP CORE模块间的数据交换。

1. 写方向数据选择接口信号，如表3.8所示：

表3.8 wr_data_sel模块接口信号表

端口名称	方向	位宽(bit)	有效电平	含义
clk	输入	1	时钟	ddr2 core 系统时钟输出
reset	输入	1	高	全局异步复位信号
ch_reset	输入	8		通道复位
wr_ch_sel[7:0]	输入	8	高	数据通道选择信号
local_wdata_req	输入	1	高	控制器写数据请求标志，要在下一周期给出数据
local_wdata[143:0]	输出	144		写入控制器的数据
wbuff_rd_en [7:0]	输出	8		wr_data_sel模块发送的读使能信号
wbuff_rd_data [143:0]	输入	144		发送到wr_data_sel模块的数据信号

为了满足 local_wdata_req 信号要求，本模块采用了一种流水线的处理方式，即先从数据缓冲单元内取出两个数据，分别存于两层寄存器内，并且为了避免通道选择所带来的时序紧张，设置了第三层寄存器。采用流水线处理，通过不同使能信号的组合来驱动寄存器间的传输，实现了接口时序要求。

2. 读方向数据选择接口信号，如表 3.9 所示：

表 3.9 rd_data_sel 模块接口信号表

端口名称	方向	位宽(bit)	有效电平	含义
clk	输入	1	时钟	ddr2 core 系统时钟输出
reset	输入	1	高	全局异步复位信号
rd_ch_sel[7:0]	输入	8	高	数据通道选择信号
local_rdata_valid	输入	1		控制器输出数据有效信号
local_rdata[143:0]	输入	144		控制器的读出数据
rbuff_wr_en [7:0]	输出	8		rd_data_sel 模块发送的写使能信号
rbuff_wr_data [143:0]	输出	144		rd_data_sel 模块发出的数据信号

3.1.6 数据缓存单元

本模块主要完成对外的数据缓存和交换功能，并产生 fifo 状态标志，用于 ddr2_state_ctrl 模块进行通道转换。

其接口信号，如表 3.10 所示：

表 3.10

端口名称	方向	位宽(bit)	有效电平	含义
clk	输入	1	时钟	ddr2 core 系统时钟输出
reset	输入	1	高	全局异步复位信号
wr_clk	输入	1		外部写时钟
rd_clk	输入	1		外部读时钟
ch_reset	输入	8		通道复位
wr_ena	输入	8		外界写使能
wr_chx_data[143:0]	输入	144		X 通道输入数据
rd_ena	输入	1		外界读使能
rd_chx_data [143:0]	输出	144		X 通道输出数据

wfifo _full	输出	8	高	写 fifo 写满标志
rfifo _empty	输出	8	高	读 fifo 读空标志
wfifo_almost_empty	输出	8	高	写 fifo 将读空标志
rfifo_almost_full	输出	8	高	读 fifo 将写满标志
wbuff_rd_en	输入	8′		wr_data_sel 模块发送的读使能信号
rbuff_wr_en	输入	8		rd_data_sel 模块发送的写使能信号
wbuff_rd_data [143:0]	输出	144		发送到wr_data_sel模块的数据信号
rbuff_wr_data [143:0]	输入	144		rd_data_sel 模块发出的数据信号

本模块通过内置的16个FIFO核，实现外界与系统间交换数据时的缓冲作用，同时时刻向外界报告内置FIFO的空满状态，避免FIFO的溢出或读空。

3.2 系统模块仿真

通过对ddr2_fifo模块及其子模块进行仿真，检测了相关信号的时序逻辑，验证设计方案的可行性。仿真体系如表3.11所示：

表3.11 系统仿真体系表[17]

顶层	第一层	第二层	第三层	……	模块功能
test.v	ddr2_fifo.v	data_control.v	fifo_144_36.v		
			fifo_36_144.v		
		ddr2_state_ctrl.v			
		ddr2_addr_gen.v	addr_gen.v		
		wr_data_sel.v			
		rd_data_sel.v			
		ddr2_ip_core.v	ddr2ip.v		
			ddr_pll_stratixii.v		
			ddr2_auk_ddr_dll.v		
	data_gen.v				

3.2.1 激励模块介绍

test_tb激励模块用于最顶层模块的仿真。

ddr2_ip_core_tb激励模块包含ddr2的仿真文件，通过给IPCORE相应的使能

及数据，检测 ddr2 一侧的时序是否正常。

ddr2_state_ctrl_tb 激励模块中通过模仿 IPCORE 的接口信号输出，检测 ddr2_state_ctrl 模块的状态转换和使能信号输出。

ddr2_addr_gen_tb 激励模块主要用来检测 ddr2_addr_gen 模块的地址信号产生是否正确。

3.2.2 顶层模块仿真情况

由于 data_gen 模块是 testbench 类型的模块，能够发送，接收及检测结果，所以 test_tb 激励模块只需发送时钟和复位。

1. 第一项仿真功能是检测是否读到的数据和写入数据不一致，实际结果发现读写数据正常。作为检测信号的 led_sig[7:0]是通过读取数据和写入数据进行比较，若数据不正确则对应的通道位为 1。下图 3.4 所示，led_sig 只有在复位一瞬间才可能导致比较的数据不一致，说明读写数据正常。

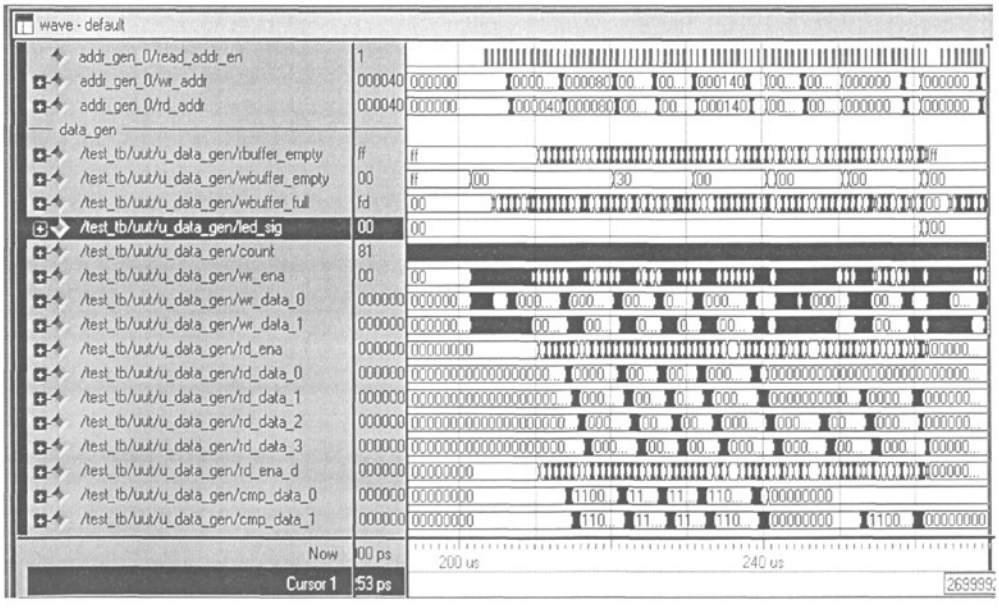


图 3.4 顶层模块数据检测图

2. 第二项仿真功能是检测写通道模块是否运行正确。下图 3.5 所示，写使能的相关信号按照设计要求进行变化，在第一次或复位后对某一通道的 wfifo 进行读时，首先读出 4 个数据放入相应寄存器中，满足设计的时序要求。

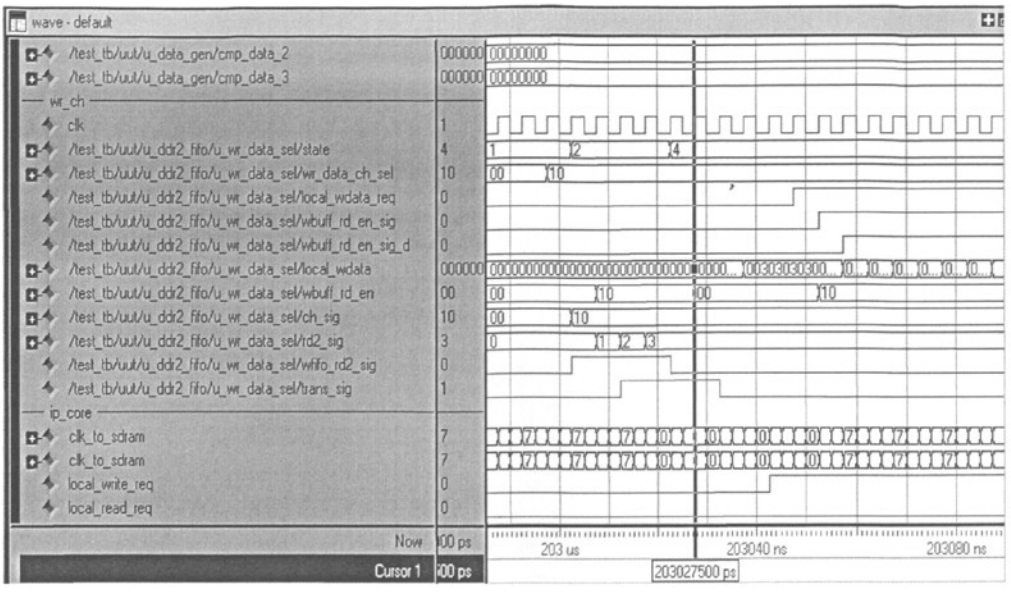


图 3.5 数据写入模块仿真图

3. 第三项仿真功能是模拟接口输入数据，用来检测 ddr2_ip_core.v 接口信号时序是否正确。由图 3.6 所示，IP CORE 接口信号时序配合正常。

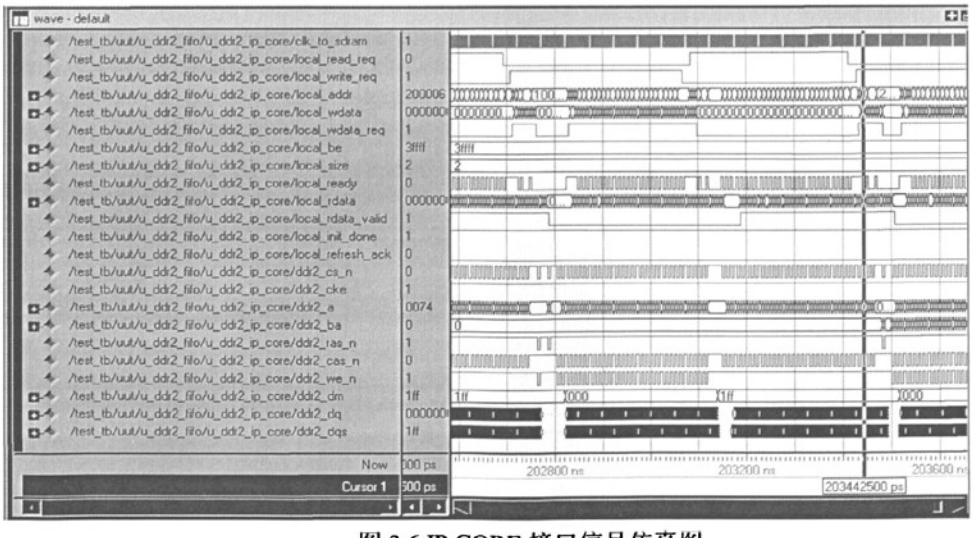


图 3.6 IP CORE 接口信号仿真图

3.2.3 ddr2_ip_core 模块仿真情况

测试模块发送读写使能及相关信号给 IPCORE，并引用 ddr2.v 文件作为 DDR SDRAM 的模拟仿真模块，检测读写时序。

本激励文件的仿真功能是检测在连续读写的情况下 IPCORE 接口时序。下图 3.7 所示，local_wdata_req 和 local_rdata_valid 信号有可能重叠，但只要把读写通道分开，并不影响 DDR 侧的接口时序。

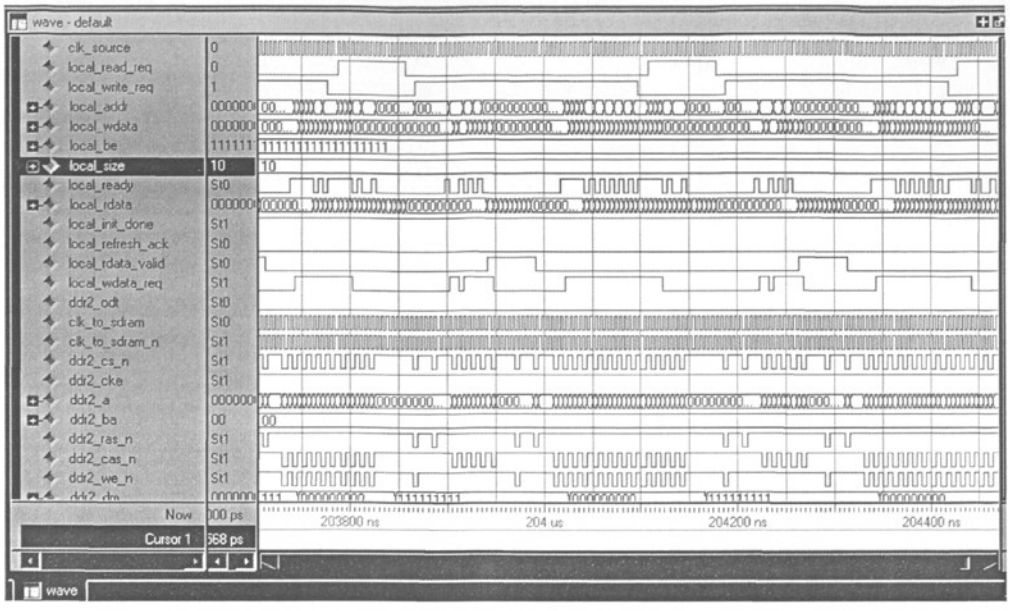


图 3.7 IP CORE 单独仿真图

3.2.4 ddr2_state_ctrl 子模块仿真情况

激励文件通过模拟 IPCORE 接口输出信号，和 ddr2_state_ctrl 模块进行交流，检测模块内信号是否运转正常。

本激励文件仿真功能是检测通道信号是否按要求进行变化。要求各接口信号能够相互配合，达到设计的要求。由下图 3.8 所示，通道信号严格按照设计进行变化，分别在读写计数器达到要求后才开启读写通道的转换。

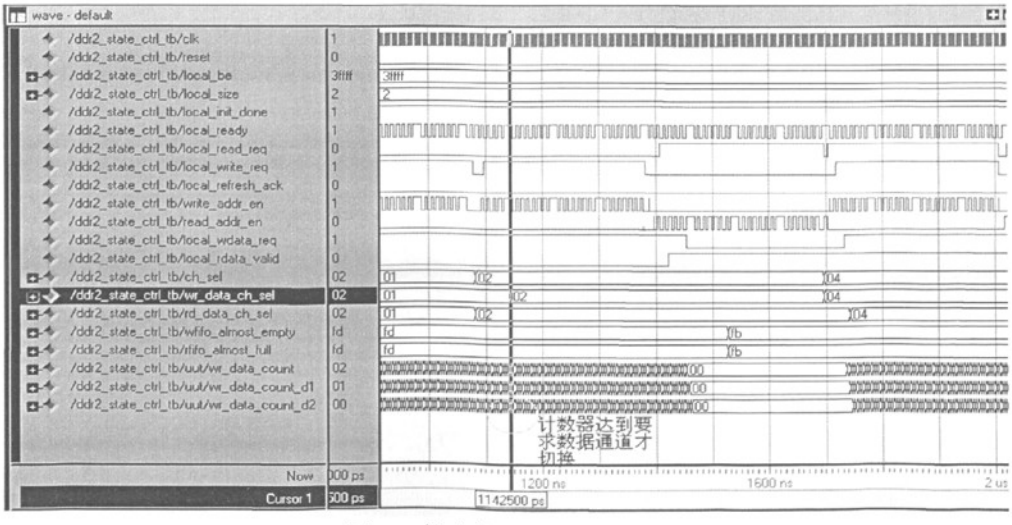


图 3.8 核心控制模块仿真图

3.2.5 仿真情况总结

通过在modelsim中调用wave_ch2.do, wave_ch4.do及wave_ch8.do文件，读者可直接看到二通道，四通道及八通道的仿真结果，较为方便。通过仿真，发现一次包含刷新情况的读写操作，其读写效率为69%。在流量100%时为避免FIFO溢出，对于200M的系统时钟，二通道读写时钟不能大于138M。由于ddr2内存容量较大，对ddr2某一通道若写满的情况没有仿真到，需要考虑检测方法。

第四章 chipscope 软件平台下的在线调试与关键技术

4.1 chipscope 在线调试平台

4.1.1 chipscope 软件简介

Chipscope 是 Xilinx 公司推出的一种实时在线调试逻辑分析工具，通过它，可以观察 FPGA 内部信号的波形图象。与 Modelsim 不同，它是观察真实的内部硬件资源信号，从而更加可信。对于系统稳定性与功能验证具有重要的作用[18]。

ChipScope 按照功能的不同主要分为了三种应用：

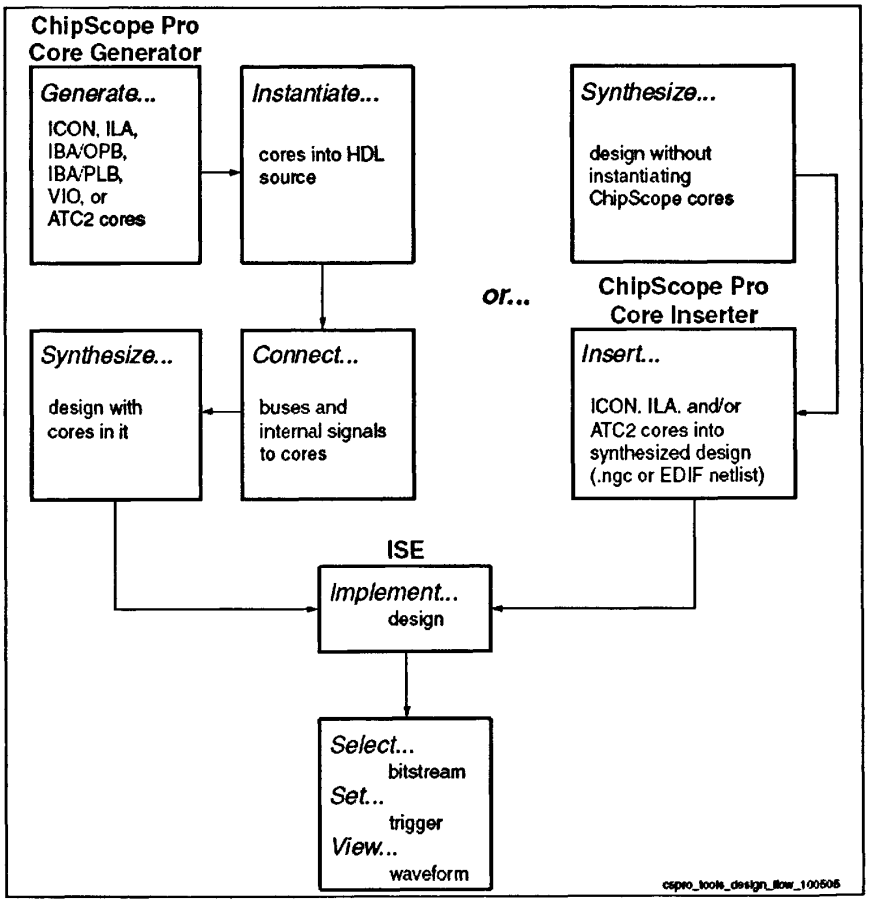


图 4.1 ChipScope 使用流程

ChipScope Pro Core Generator：生成 edn 形式的各类 core: ICON、ILA、ILA/ATC、IBA/OPB、IBA/PLB、VIO、ATC2。其中 ICON（Integrated Controller core）是必须的，它负责控制其余的 Core，并且与 PC 机通过 JTAG 通信。其余各 Core 是特定功能的嵌入式逻辑分析仪，其中 ILA(Integrated Logic Analyzer core)是常用的，使用它不需要额外的设备。ILA 的作用是根据设置的触发条件，将需

要观测的信号存入 FPGA 内的 BlockRAM 中。生成了各类 Core 以后，用户可以在 HDL 代码中对它们进行例化，将待观测的各信号与对应的 ILA 端口相连，然后将各 ILA core 与 ICON 相连，这样得到的代码就具有了嵌入式逻辑分析仪的功能了。

ChipScope Pro Core Inserter：Core Generator 用在 HDL 流程中，当用户需要不修改 HDL 代码而使用 ChipScope 的功能时，需要用到 Core Inserter。它使用户能够通过图形界面，将需要的 Core 插入综合后的网表文件中，实现与 Core Gen+HDL 例化相同的功能。

ChipScope Pro Analyzer：运行在监控 PC 上的人机接口软件。通过它，用户可以在线设置触发条件，并将满足要求的观测信号数据收集上来，进行图形化显示或者存储为文件供后续处理。

通过图 4.1 我们可以清楚地看到，ChipScope 的使用主要是两个方向，一个是将 ChipScope 生成的 HDL 源文件加载到自身的工程中，然后完成整个综合布局布线，最后通过 ChipScope Pro Analyzer 分析；另一个是将综合出来的网表通过向其中插入监控信息，然后布局布线，通过 ChipScope Pro Analyzer 分析[19]。

4.1.2 生成正确的 ChipScope 文件

一般来说，我们主要是第二种应用，通过对综合的网表文件信号的监控，来达到最终的目的。所以，下文主要针对的是这种应用展开说明。

我们首先需要生成一个 ISE 可以正确识别的 ChipScope 文件，并且正确关联。下文介绍一种简易的生成方法。

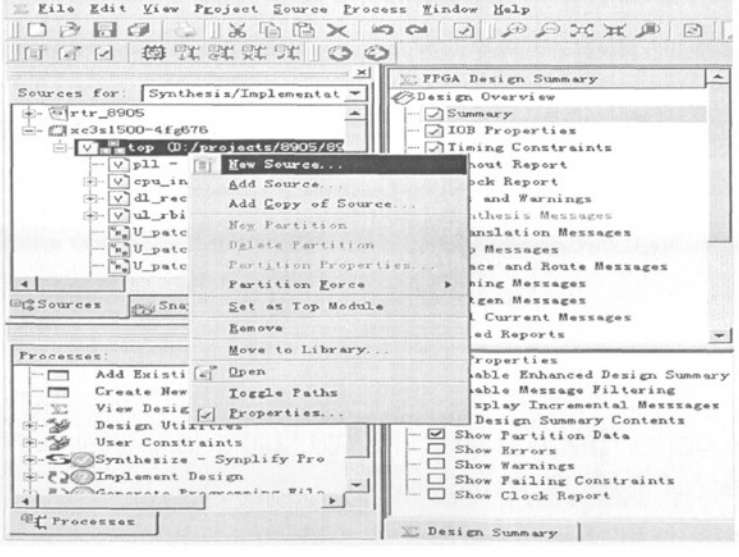


图 4.2 ChipScope 文件生成

如图 4.2 所示，我们在 ISE 试图的工程框内右键选择 new source，这样就可以弹出图 的界面。

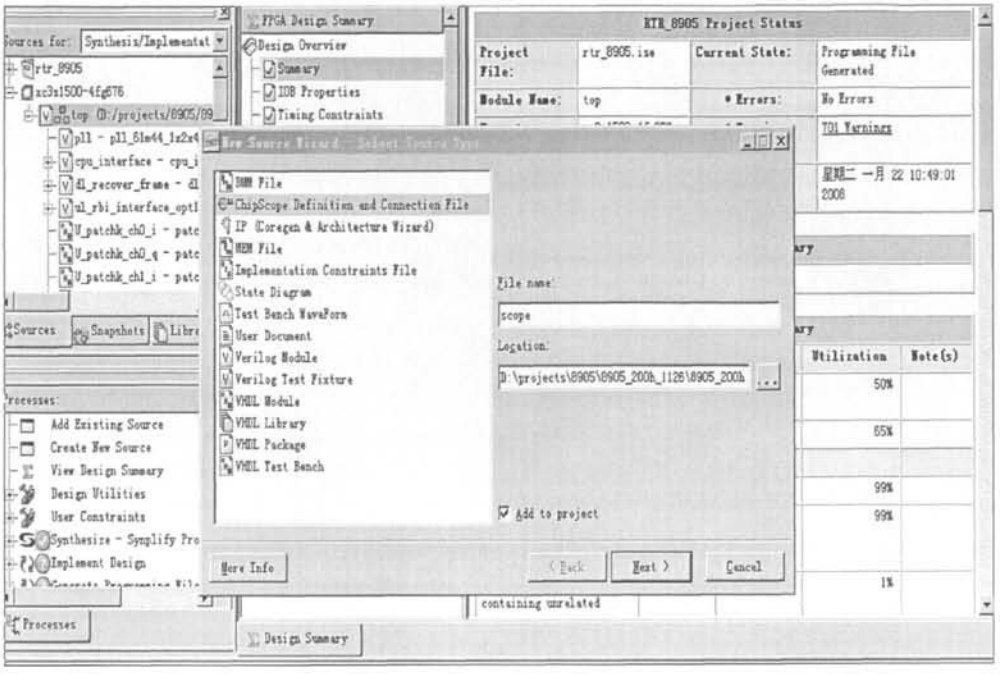


图 4.3 ChipScope 文件属性

在图 4.3 的视图中，我们需要选中左边的 ChipScope Definition and Connnection File 选项，在右边的对话框中填入文件的名称和所处的位置，在最后的 Add to project 复选框中，建议选上，这样就可以自动关联到所在的工程。这样一路选择 next，一直到最后的 finish，就完成了一个 ChipScope 的生成与关联。

4.1.3 正确的配置 ChipScope 相关选项

如果正确的加载了 ChipScope 文件后，在文件树的视图中，我们将看到图 4.4 中所示的图样。

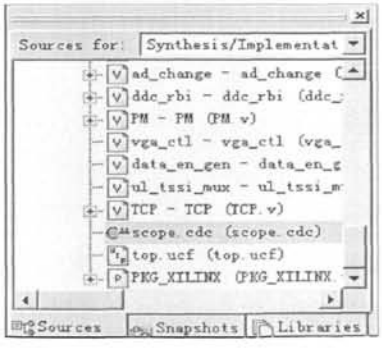


图 4.4 ChipScope 生成文件图示

双击我们生成的后缀名为 cdc 的 ChipScope 文件，我们将进入 ChipScope 监控设置的界面当中，如下图 4.5 所示。

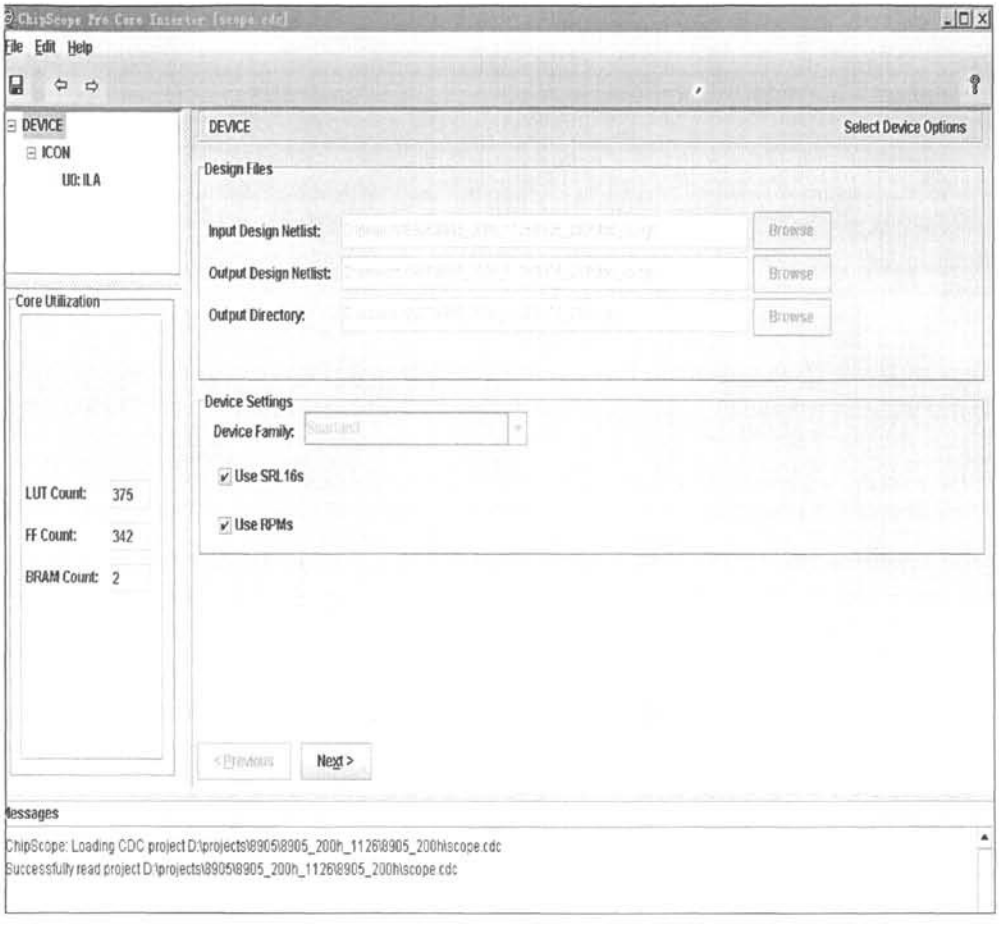


图 4.5 ChipScope 设置界面（一）

如图 4.5 所示，图示的界面中 Input Design Netlist、Output Design Netlist、Output Directory，分别代表了所在工程的输入输出网表文件，以及输出目录。如果我们按照本文的方法进行操作的话，这些都是由 ISE 直接关联好了，无需我们再做修改。

如果是单独调用 ChipScope Pro Core Inserter 进行设置的话，这些就需要我们手动填写，包括 Device Family 类型。

在图中的 Use SRL 16s 选项和 Use RPMs 选项的意义代表我们是否使用 Xilinx 的内部优化选项，其中 Use SRL 16s 代表我们是否使用 Xlinx 内部的 SRL 16 器件，这个器件的使用可以降低整个 ChipScope 的资源占用，所以默认应该使用。

Use RPMs 选中代表我们在综合的时候，在资源允许的情况下将 ChipScope 平均分配到 FPGA 片内的 Slice 上，而不是集中在某几个 Slice，这样不利于布线，

默认的选项是使用，而当资源严重不足的时候，此选项将不起作用。

在图 4.6 中左边显示了 ChipScope 的资源占用情况，以供我们作为参考。

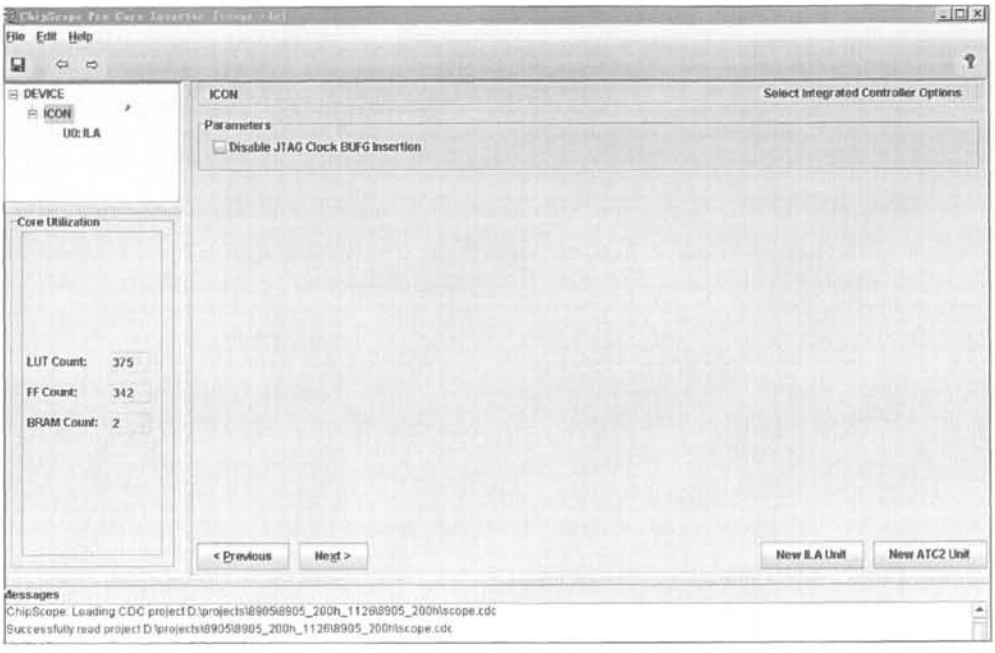


图 4.6 ChipScope 设置界面（二）

在图 4.5 的界面中，点击 next 将进入图 4.6 的设置界面，在本届面主要有一个复选框 Disable JTAG Clock BUFG Insertion。这个复选框的作用是是否将 ChipScope 的时钟通过 BUFG 上时钟树。这个在时钟资源紧张的情况下，可以选择让它不上时钟树，但是这样做的结果会大大影响到数据测量的准确性，所以在一般情况下，都是需要将时钟上时钟树的。

在本图的右下角有两个选择项，New ILA Unit 可以让我们在增加一个新的 ILA 单元，这个单元就是我们通常所用的一个 ChipScope 资源。而 ATC2 单元，主要用于接在外部的新一代的逻辑分析仪上，所以一般情况下是不使用的。

图 4.7 ChipScope 设置界面（三）

在图 4.6 点击 next 将进入图 4.7 界面，从这开始进入 ChipScope 设置的重点。首先，在 Number of Input Trigger Ports 里面我们可以选择需要跟踪的数据端口的数量，其中每个端口最多能够跟踪 256 个信号，所以我们可以根据我们实际工程需要跟踪的信号数量和个人喜好来确定需要几个端口。

在每个数据端口的内部，都有几个可供参考的设置选项。其中 Trigger Width，是设置跟踪数据的个数，它的宽度是按照 bit 进行计算的。右边的 Match Type 用于设置跟踪数据的类型，其中有以下几种数据类型：Basic，Basic w/edges，Extend，Basic w/edges，Range，Range/wedges。对于 Basic 数据类型来说，他的数据值有 0，1，x 三种可供选择。而后缀带 w/edges 的数据类型，它的数据类型有 0，1，x，R，F，B。其中 R 代表了上升延，F 代表了下降延，B 代表了任意跳变。前缀为 Basic 的数据类型，它的可触发条件只有＝和<>两种，而前缀为 Extend 的类型还多出<,<=,>,>=4 种比较运算。前缀为 Range 的类型还多出了 out of range 和 in range 两种类型。对我们平时调试而言，Basic 类型已经可以满足我们的需要，而且它的资源占用是最少的；而 w/edges 类型的数据也使用较多，一般用于使能信号的抓取，我们可以观察它们的上升或者下降延，我们应根据实际需要来设置所需要的数据类型。

Match Units 选项是用于我们设置出发条件的组合条件，它的设置最大为 16，也就是说最多能够跟踪 16 个条件的组合触发。一般我们都是每次通过一种条件出发，所以默认为 1，这个的设置也会很大程度的影响资源的占用。

Counter Width 用于设定特定的时间段内我们跟踪符合条件的信号的脉冲数量，如果我们需要对这种信息进行统计，就可以设置计数器的宽度。一般情况下不使用就将其设置为 Disable。

Enable Trigger Sequencer 是设置我们是否跟踪状态机的状态，默认是选中的。右边的 Max Number of Sequencer Levels 用于设定状态机跳转的状态的数量。

最后的 Enable Storage Qualification 用于设定是否进行存储优化，默认的为选中。

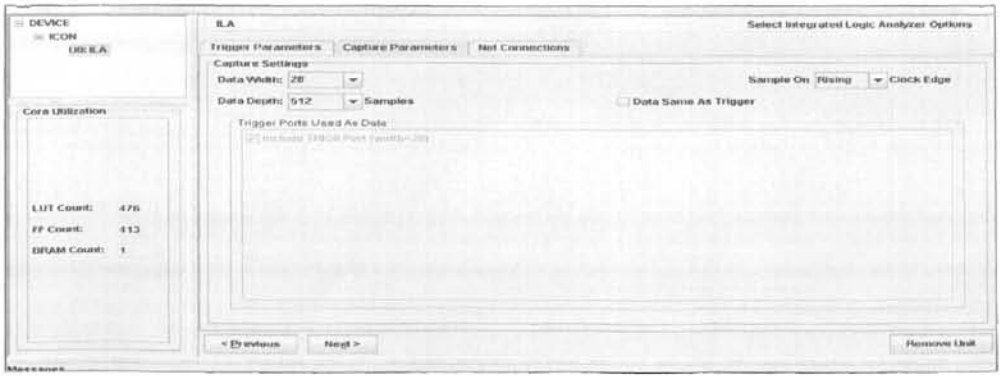


图 4.8 ChipScope 设置界面（四）

在图 4.7 的界面点击 next，我们将进入图 4.8 的界面，在这个界面下，我们主要设置需要捕获的数据信息类型。这个捕获的数据主要通过片内 ram 来进行捕获，所以我们可以设置数据的宽度和深度来决定消耗多少 ram 资源，同样设置我们数据是通过时钟的上升延还是下降延进行触发。默认的是数据的宽度与触发源的宽度保持一致。

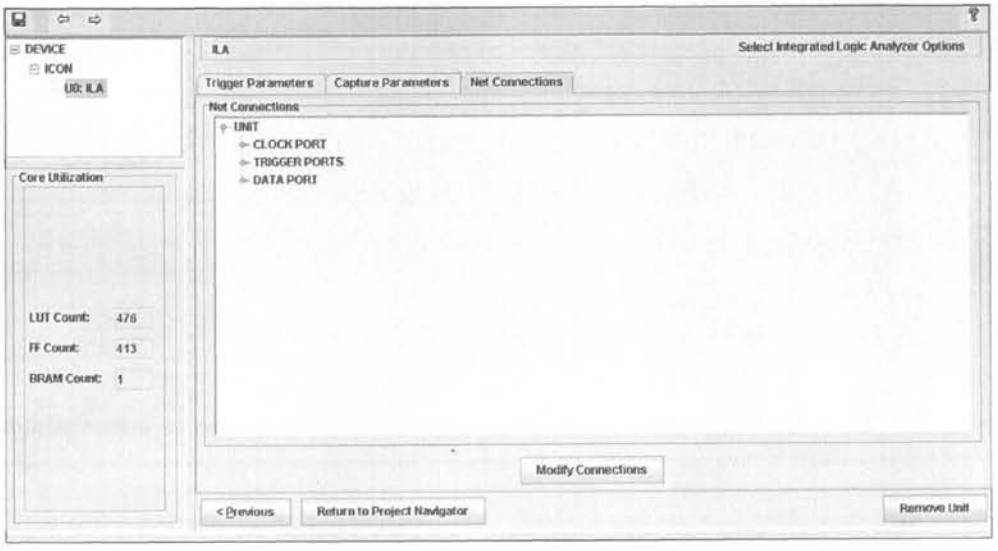


图 4.9 ChipScope 设置界面（四）

在图 4.8 中点击 next 我们将进入图 4.9 的界面。在本届面中我们可以通过选择 Modify Connection 来确定具体需要跟踪的信号，当所有的时钟和数据都设置好后，我们可以选择 Return to Project Navigator 来返回 ISE 界面，进行后续的布局布线。

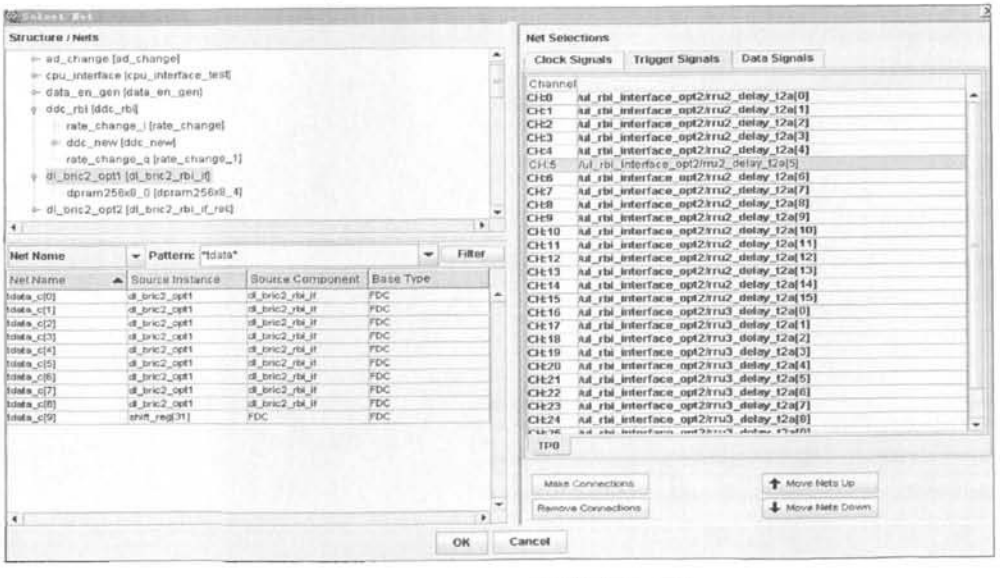


图 4.10 ChipScope 设置界面（五）

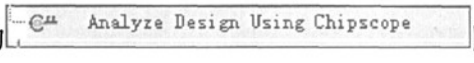
在图 4.8 中，我们选择 Modify Connection 则会进入图 4.10 的界面，在这个界面下我们将选择出我们需要跟踪的信号。⊞这种标志代表这个模块还有下级模块没有展开，我们首先需要所跟踪的信号在代码中的哪个层次，然后在 Pattern 对话框中输入我们需要查找的信号名称，然后点击 Filter 进行过滤。在输入的信号名称我们需要在其两端加*来进行约束，因为整个工程已经经过了综合，所以信号名称不一定会完全匹配，一般我们选择相似的信号即可。如果信号名在综合的过程中变得无法识别，我们可以在 HDL 代码中在信号定义的那句话加入以下信息：

```
reg     [7:0]       tdata_tmp/* synthesis syn_keep = 1 */;
```

在找到我们需要跟踪的信号后，我们点击 Make Connections 来使其加入右边的对话框中，当我们不需要某个信号，也可以通过 Remove Connections 来移出，一直到将全部的信号选择完全后，点击 ok 退回到图 4.8 的界面，完成整个信号的选取过程。

至此我们已经正确的生成了 ChipScope 文件，并将其加入到工程当中。

4.1.4 分析 ChipScope 信号

双击 ISE 工程下的 

图标，我们就可以进入 ChipScope 的调试界面了。见图 4.11 所示：

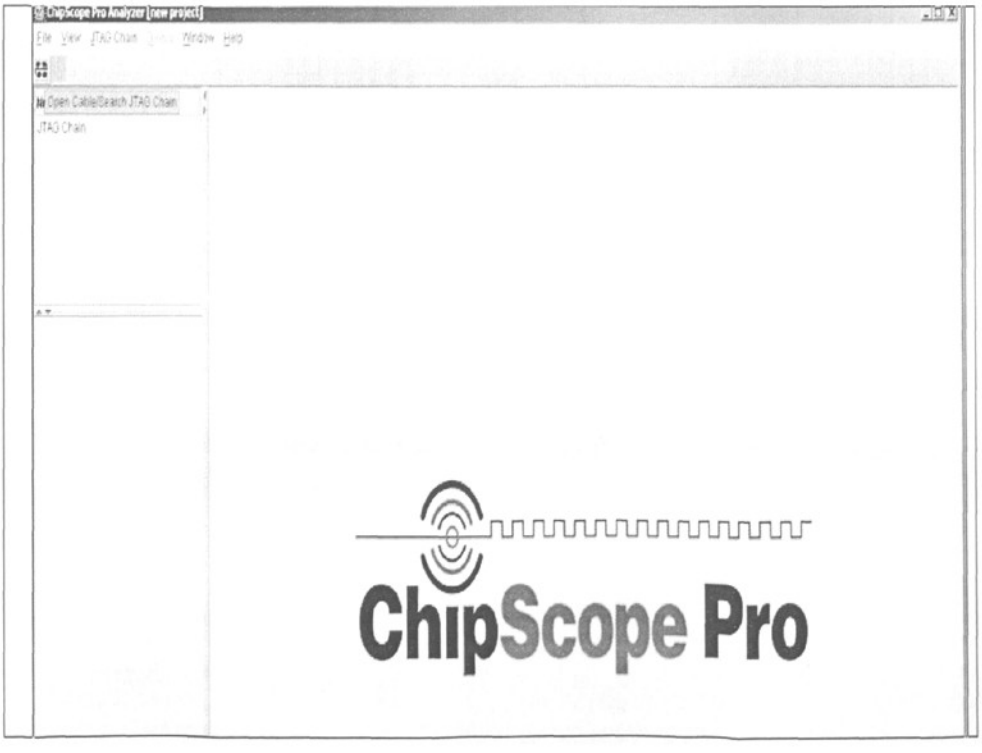


图 4.11　ChipScope 调试界面（一）

在此界面中，我们单击图标，就可以让 ChipScope 进行并行电缆的搜索，一旦搜索到所使用的下载电缆，那么将会弹出图 4.12 所示内容。

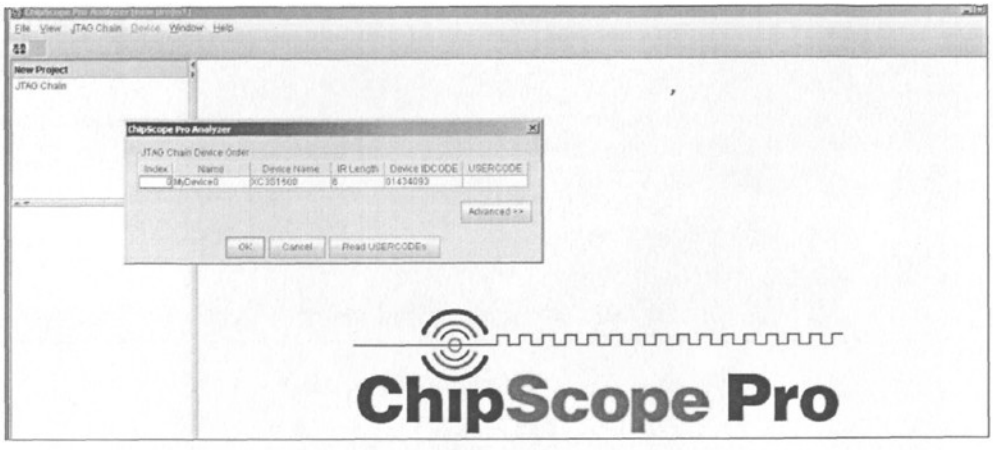


图 4.12 ChipScope 调试界面（二）

在图 4.12 界面中，我们核对 FPGA 的相关信息，看是否与我们使用的一致，如果一致点击 OK，进入图 4.13 界面。

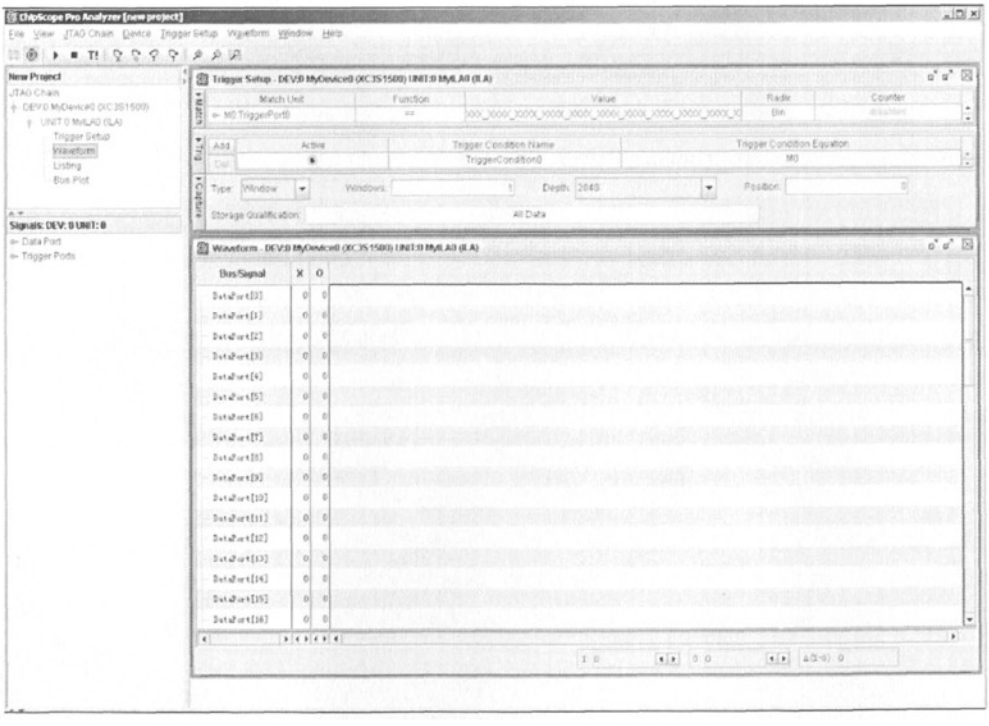


图 4.13 ChipScope 调试界面（三）

图 4.13 就是调试的主页面了，首先我们需要对信号的名称进行导入，见图 4.14。

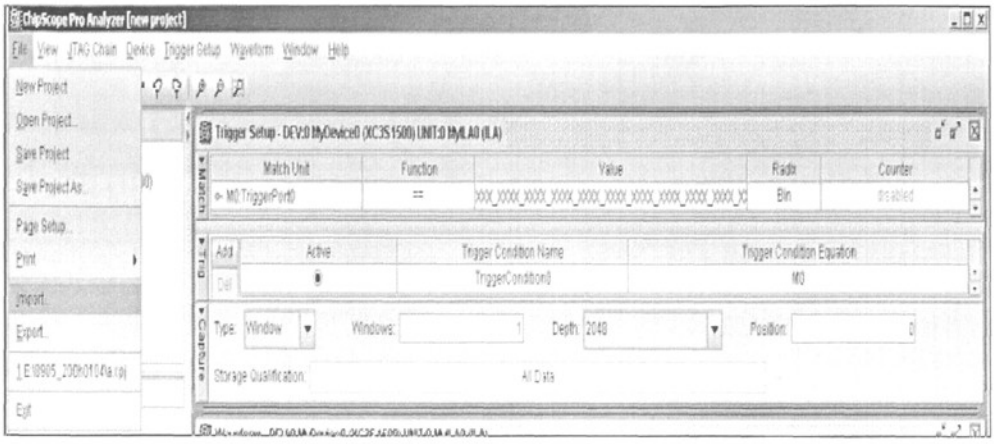


图 4.14 ChipScope 调试界面（四）

我们选择 File 目录下的 Import 选项，则会进入图 4.15 的界面。

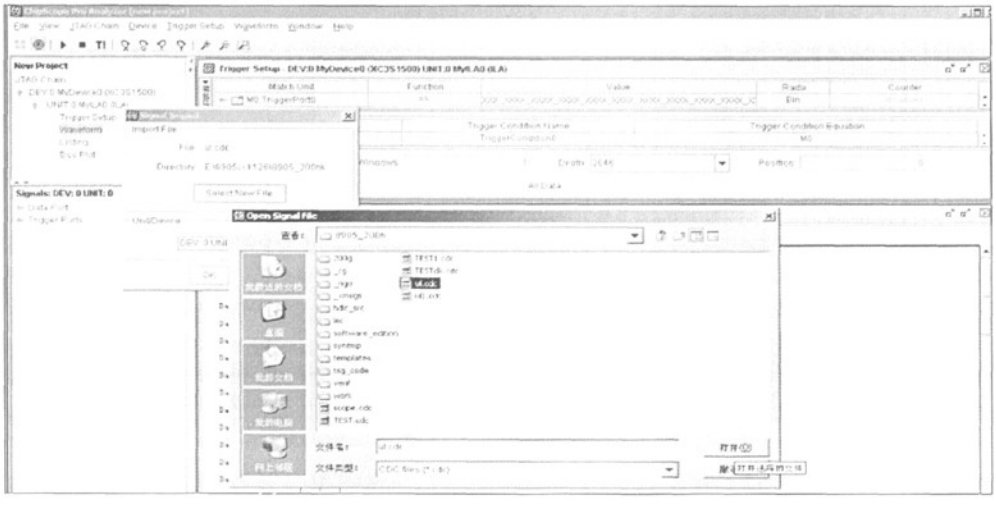


图 4.15 ChipScope 调试界面（五）

在图 4.15 中，我们选择 Select New File，来制定相关 cdc 文件的位置，选中后 ok，我们将进入图 4.16 的界面当中。

图 4.16 ChipScope 调试界面（六）

在图 4.16 中我们看到，所有的信号都已经被命名，这时候我们可以将相关的总线信号进行合并，具体的做法如图所示：首先选定总线的所有信号，然后右键 Add to Bus =>New Bus，这样我们就可以将其相关的总线信号进行合并进行监控。具体如图 4.17 所示。

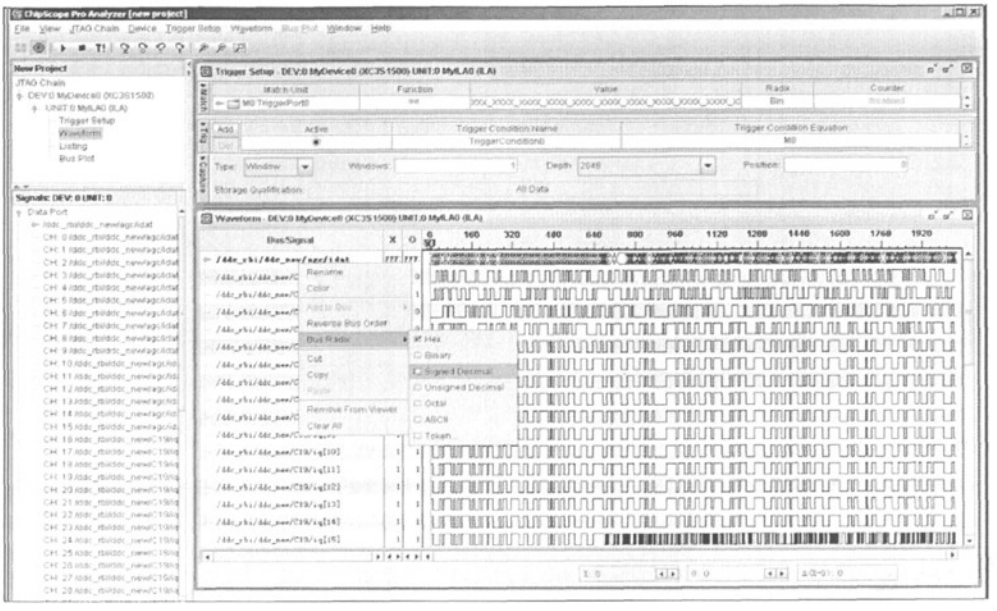


图 4.17 ChipScope 调试界面（七）

如图 4.17 所示，我们进一步可以设置总线数据类型，也是右键菜单，从上倒下依次为 16 进制、2 进制、有符号 10 进制、无符号 10 进制、8 进制、ASCII 码、Token 码。其中前面 6 种进制是我们常见进制，最后的 Token 码是用于跟踪状态机的特殊码，我们可以定义状态机种任意的状态为一种可识别的数据，具体都是人为定义。因为只要我们心里清楚每个状态具体对应的数据应该为什么，就无需在定义 Token 码的类型，所以这个在具体的调试过程中用处不是很大，在这里就不做详细说明了。

图 4.18 ChipScope 调试界面（八）

如图 4.18 所示，此图为触发条件的设置，我们可以设置自己需要触发的条件，具体某些数据类型能设置那些触发条件，前面都有介绍这里就不详细解释了。

在 Position 对话框里面我们可以选择触发的条件在整个波形所处的位置，它是以时钟波为单位的，设置好这些条件后我们可以点击工具栏中的▶按钮，进行数据跟踪。当满足所设置的触发条件后，整个 ChipScope 将自动停止下来。这时候我们将见到图 4.19 的界面。

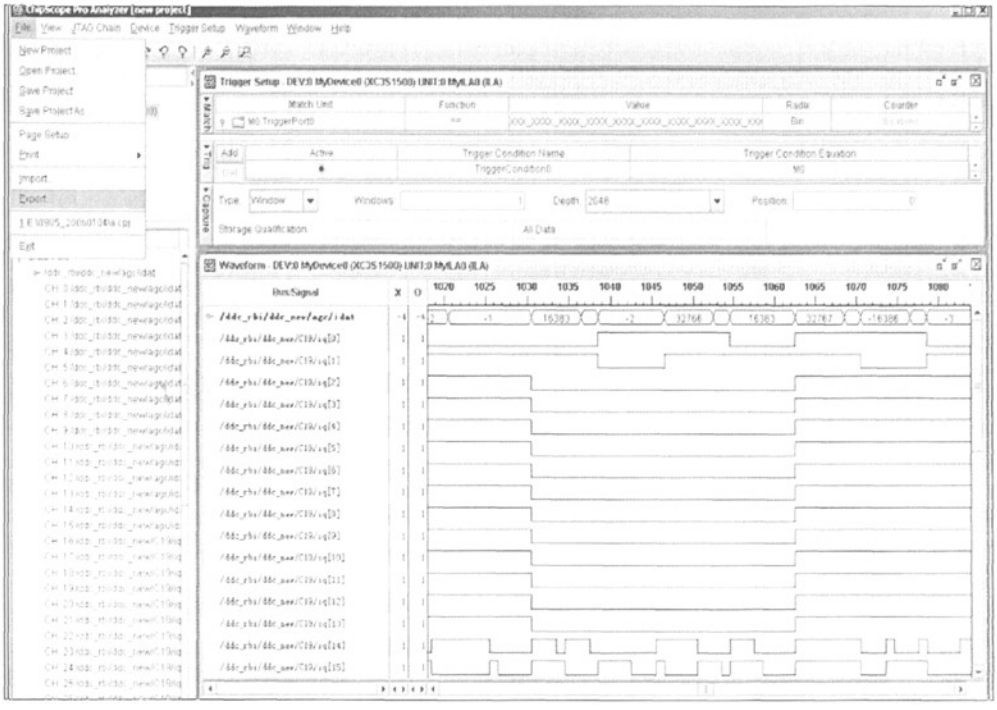


图 4.19 ChipScope 调试界面（九）

如果我们需要将波形数据进行倒出，我们就可以选择 File 目录下的 Export 选项进行操作，具体如图 4.20 所示。

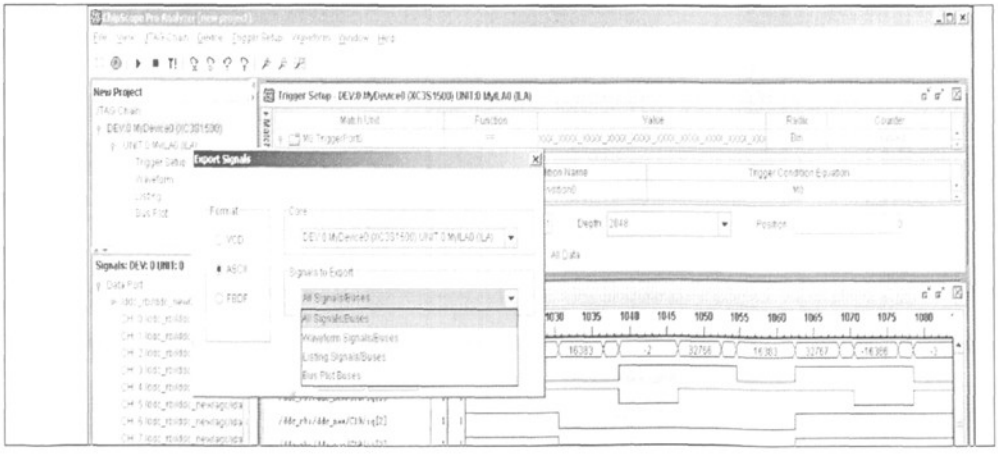


图 4.20 ChipScope 调试界面（十）

如图 4.20 所示，Format 选项代表了能够倒出数据的类型。

VCD 类型的数据主要用于将波形导入 Modelsim 中观看，因为 ChipScope 中的波形与 ModelSim 的波形基本一致，所以在实际应用中这个数据类型应用不是很多。

ASCII 数据类型主要是将所有的数据以 ASCII 码的格式导入到文本文件中使用，这种数据类型应用较多，我们可以将到处的数据在导入到 Matlab 中仿真，查看我们的数据是否是否达到了设计的要求，这样可以大大缩短问题分析的周期，在实际应用中应用较多[20]。

4.2 模块在线调试分析

在掌握了 chipscope 软件平台的使用方法后，通过在线调试对系统模块进行调试分析。其中以下问题需要注意：

1. 因可综合的测试模块内部逻辑不合适，会导致数据缓存单元读写溢出问题,如下图 4.21 所示：

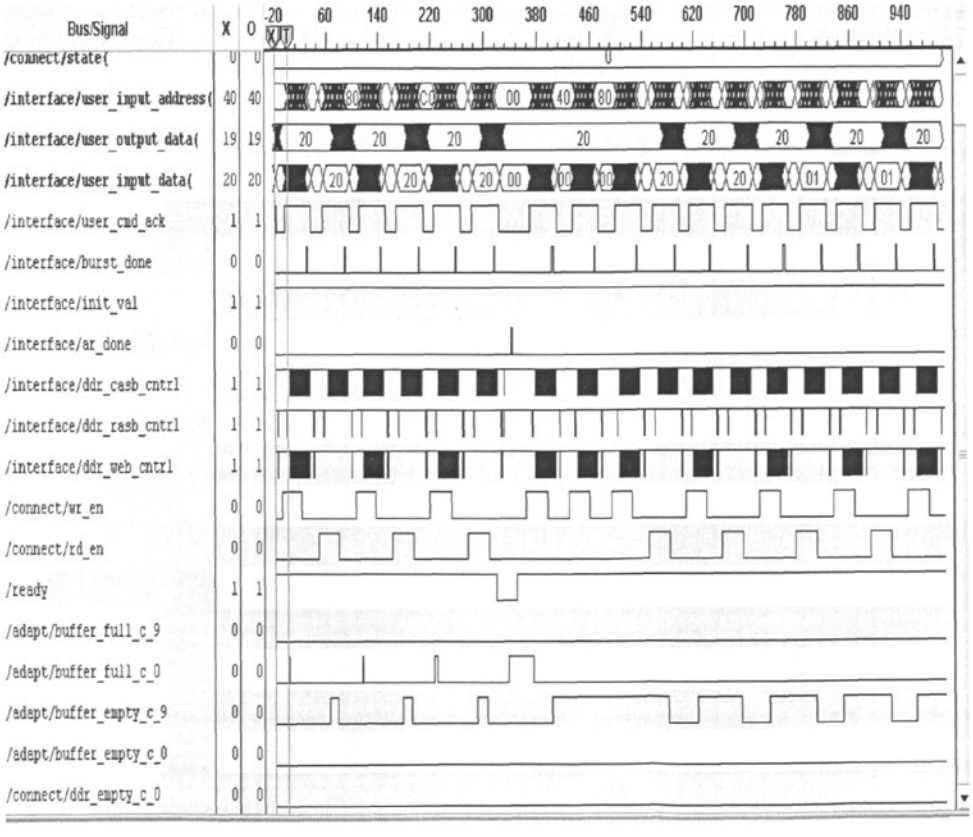


图 4.21 数据缓存单元溢出图

因状态机安排不够恰当，会导致状态跳转时出现异常情况，如下图 4.22 所示：

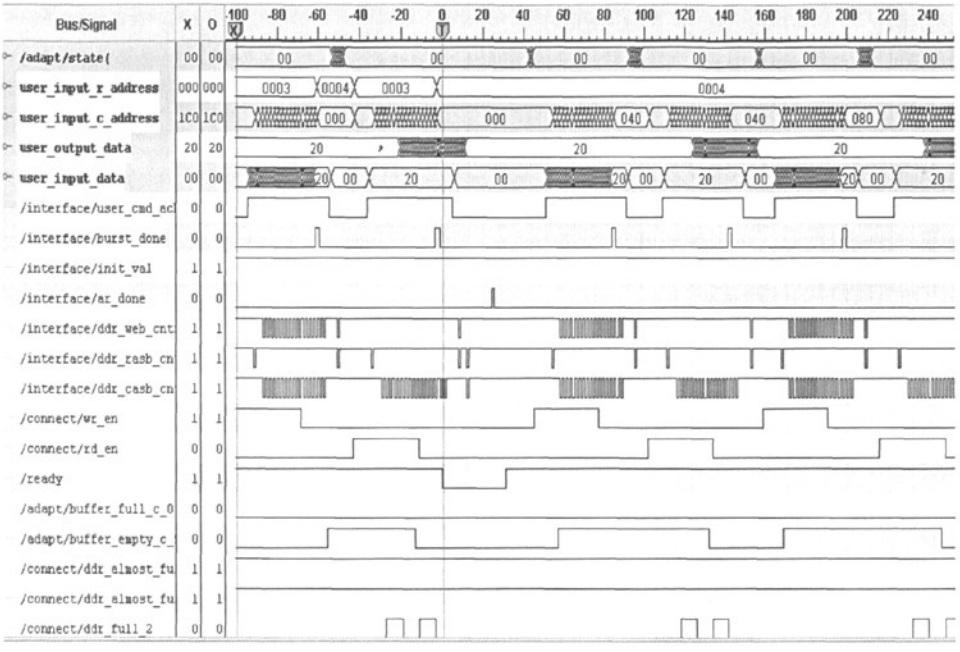


图 4.22 状态跳转异常

2. 因各通道复位信号逻辑关系处理不准确，导致 FIFO 溢出的情况，如下图 4.23 所示：

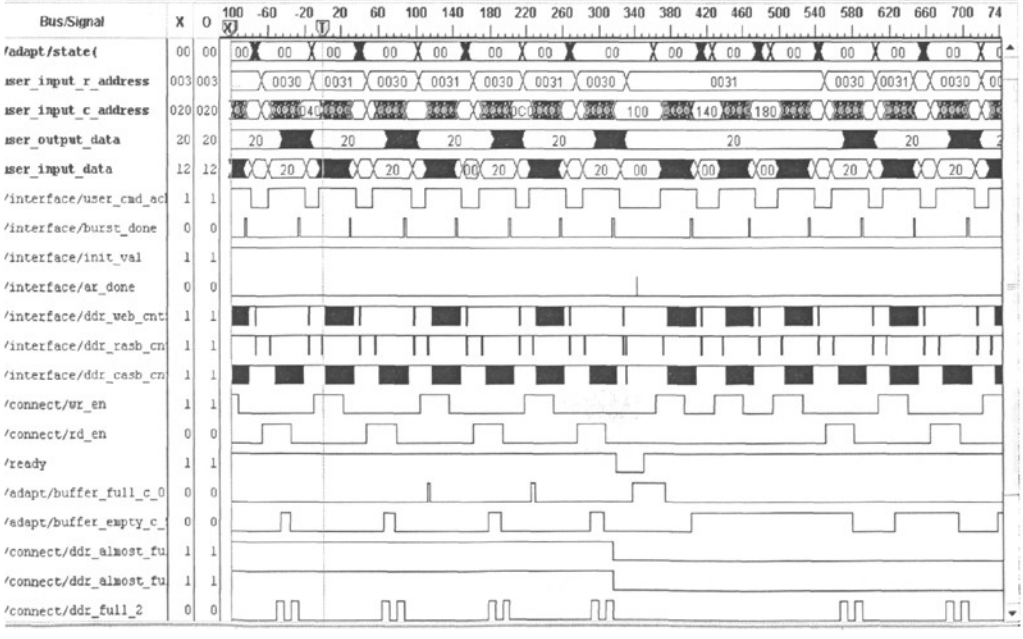


图 4.23 FIFO 溢出图

以上问题的产生，经分析主要由模块内部一部分逻辑安排不合适导致。通过反复更正与总结，得到一些关键技术问题及经验[21]。

4.3 关键技术及总结

4.3.1 流水线处理技术

流水线处理技术主要用于时隙紧张和数据位宽较大的逻辑，主要目的在于满足逻辑时序要求以及减少组合逻辑的级数，从而减少时序方面的压力[22]。例如本系统模块中的数据写入单元就采用了这一技术。如下图 4.24 所示，按照设计要求进行变化，在第一次或复位后对某一通道的 wfifo 进行读时，首先读出 4 个数据放入相应寄存器中，通过流水线的操作方式，缓解时序紧张，并且满足设计的时序要求。

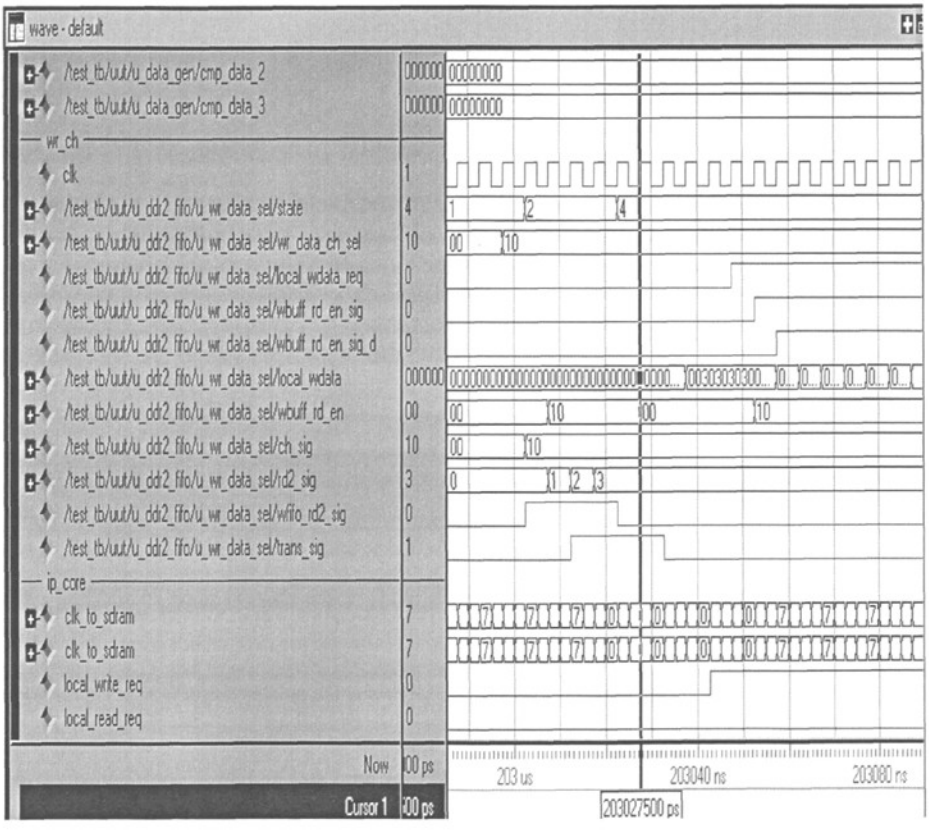


图 4.24 数据写入单元执行机制仿真图

而实现这一过程的流水线体系如下图 4.25 所示：

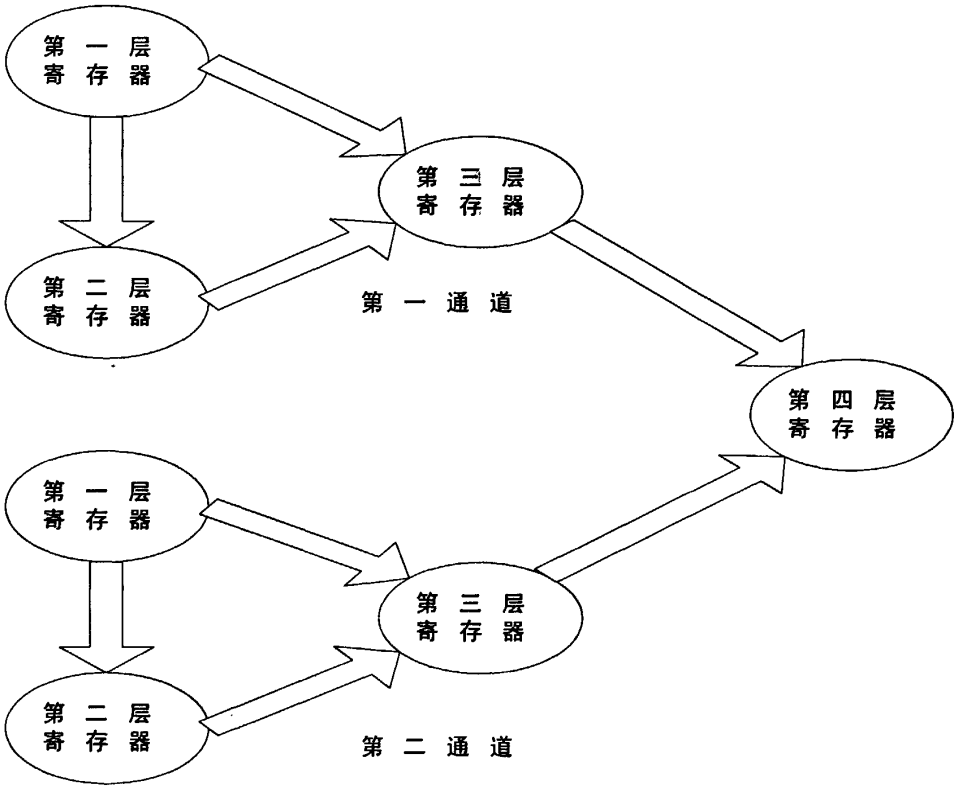


图 4.25 流水线机制图

如上图 4.25 所示的两通道系统，需要四层寄存器满足要求。数据首先从 FIFO 进入第一层寄存器，在条件一下再进入第二层寄存器，在一般情况下写入第三层寄存器，第二层寄存器的数据在条件二下写入第三层寄存器，由一层和二层进入第三层寄存器需要条件三仲裁。之后在条件四下，由第三层进入最终的输出层，第四层寄存器[23]。

这种做法，主要是因为若从 FIFO 读取的数据直接进入第四层寄存器的时隙不够，所以需要设定特定条件，先从 FIFO 预读取数据存入第一层寄存器。而且由于分通道的原因，再次对时隙要求提出了挑战，必须先从 FIFO 内部读取两个数据存入寄存器。而在一般情况下，FIFO 中的数据和最终写入数据从输入端输出端来看，是一致的，只是时间上会延迟两拍，所以又要保证数据在一般情况下的透传特性。这样的情况一般采用状态机执行，分不同情况完成相应得操作，同时需要逻辑仲裁。

用流水线处理方式的另一个原因是减少组合逻辑级数[24]。因为传送的数据是上百位的，所以应尽量减少组合逻辑的含量，减少对时序要求的冲击。通过流水线处理方式，各层寄存器间都是由时钟驱动，即用纯时序逻辑代码写成。尽量避免了时序问题的发生。

4.3.2 通道复位抗干扰技术

在本系统中，由于有多个通道相互独立，为了保证各个通道的复位信号不会对其他通道造成影响，本设计采用了一种抗干扰的技术。

首先将各通道的外部复位信号都要经过核心控制模块进行处理，用处理之后的复位信号在合适的时机触发相应的通道。具体操作过程如下：

1. 先将各个通道的复位信号转换为相应的等待信号，置高。
2. 各个等待信号在核心状态机的每个通道的写入状态触发通道复位，这里的重点是在一个通道的写入状态只能触发其他通道的复位。
3. 当完成通道复位且响应读写操作都已完成时，关闭等待信号，置低。

通过这样的处理方式，避免了本通道的复位导致其他正常通道的读写时序，避免了逻辑混乱。

4.3.3 状态机相关问题

建立模块是为了实现需要的功能，而实现功能很多时候都要用到状态机，状态机是模块的核心[25]。

对于复杂的模块，经常会被划分成为实现不同功能的小模块，顶层模块有顶层的状态机，下层模块可能需要建立自己的状态机。和盖房子一样，模块实际上就是被这样大大小小的状态机支撑起的高楼。楼无论有多高，都有主梁，所以每个模块都需要这样一个主状态机。状态机是一种逻辑思想，状态机构建的好坏直接影响着模块功能的发挥与稳定。状态机建好了，模块的雏形就定了。

既然状态机实现的是特定功能，我们一般都会按功能划分状态，这是比较便利的方法，但此时我们必须把涉及的情况都想到，并建立对应的状态。当涉及的状况相对复杂，我们不能面面俱到的时候，按照天然的轴线进行构建是合适的切入点[26]。

按时间轴建立状态机：我们的状态很多时候是以时间为走向的。比方说从模块开始运行的时间算起，某段时间内，由于外部激励的影响导致模块应该完成相应的动作，那么这段时间就可以建立一个模块；而在另一段时间，外部激励没有影响的情况下模块需要完成另外的动作，则成为又一个状态。这种状态机，经常是为了满足某一段时间内的特殊动作而建立的。例如本系统的数据写入单元中就用到了这一方式。

按空间轴建立状态机：当遇到很多特定的区域相互作用时，可方便的按空间不同而划分出不同的状态。比如在不同通道转换时，每一个通道的动作可以建立需要的状态。需要考虑的是，看能否把不同通道的某个特定动作合为一个状态，从而简化代码。例如本系统中的核心控制模块就用到这一机制。下图 4.26 中为只

有两个通道的状态机建立机制：

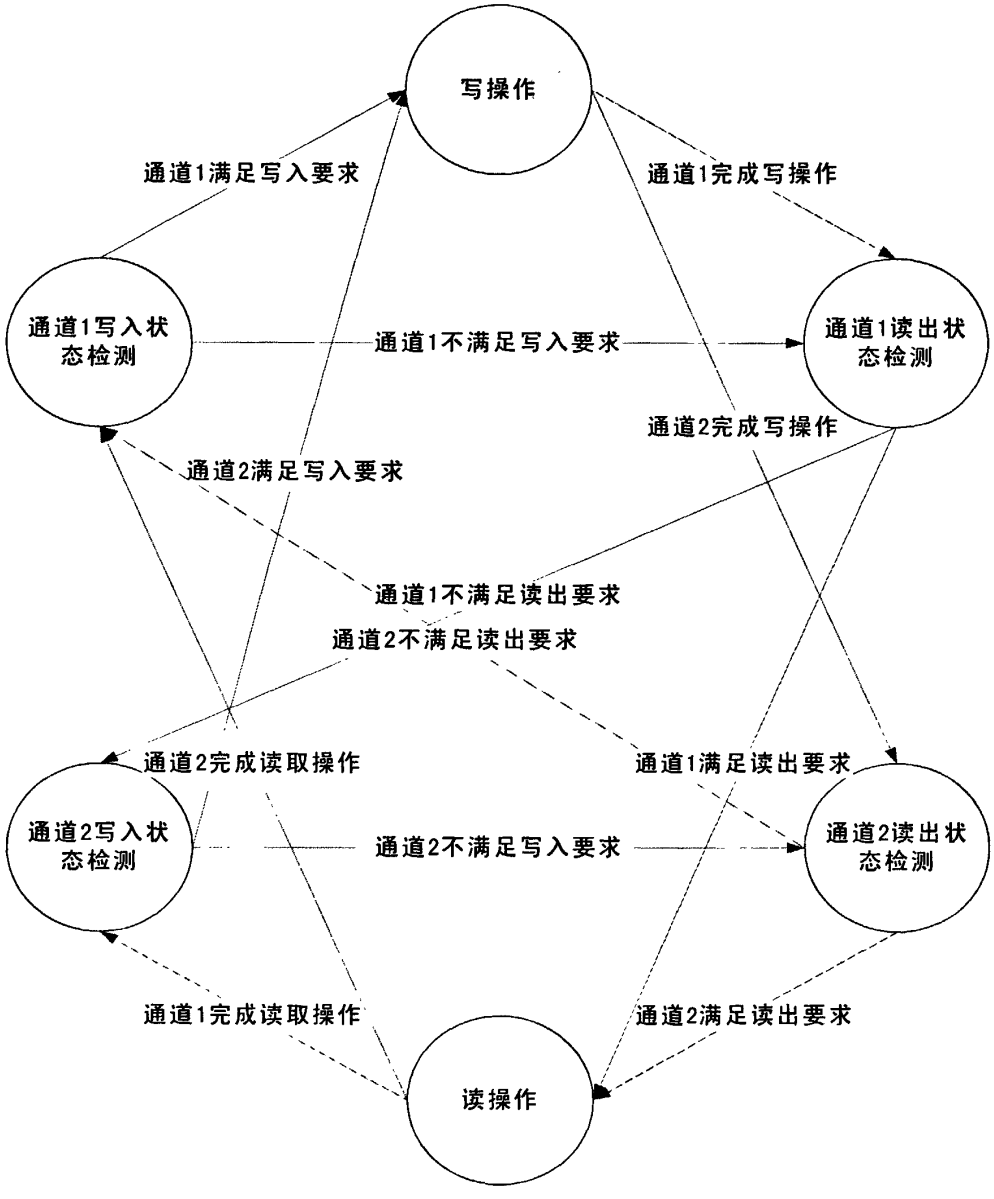


图 4.26 两通道状态图[27]

本章总结：

通过对上述关键问题的解决，实现了硬件调试误码率保持为 0，软件运行平稳。以下为整合后的系统所占资源，如图 4.27 所示：

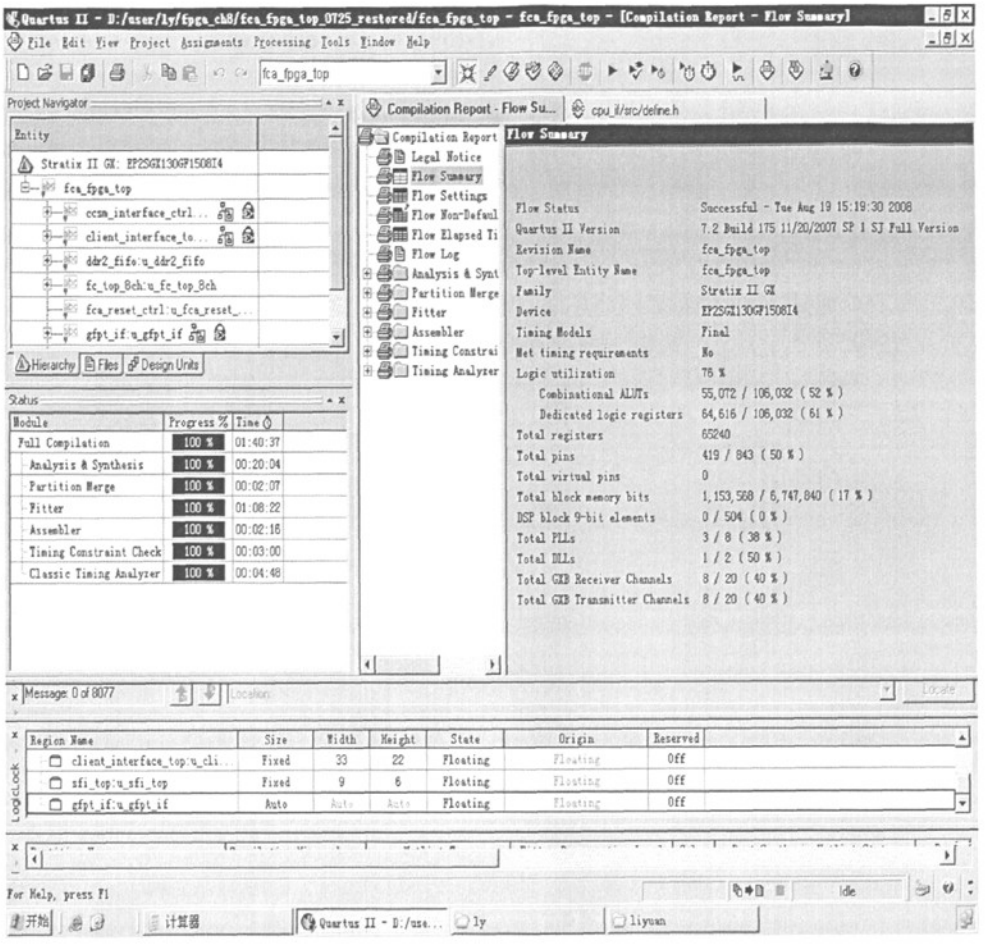


图 4.27 系统所占资源图

由图中可以看出，系统所占资源与选用芯片匹配。并且发现时序分析报告没有错误，说明系统达到时序要求。在仪表调试后发现系统功能正常并且运行稳定。属于一次成功的设计。

第五章 结束语

（1） 本文所做工作

在开发接口控制模块的过程中，首先需要了解 DDR 存储器件的原理以及在设备中的位置和作用，然后需要分析业务功能需求和自身开发的硬件结构，确立需要设计开发的功能，最后根据实际需求进行开发[28]。

本文所做的主要工作：

1、进行接口模块的需求和功能分析。

2、研究 DDR 存储介质以及相关 IP CORE 原理。

3、根据实际需要，对接口模块实现功能进行了硬件抽象层的描述，并给出了设计和实现的解决方案。

4、对功能模块进行编码驱动开发。

5、对功能模块进行仿真，调试。

7、分析和解决关键技术问题。

本文通过对接口功能需求进行分析，然后根据功能特性进行开发。这种具有 FIFO 传输特性的接口模块的研发具有一定代表性，在理论和实践方面都具有重要的意义[29]。

本文对于接口模块的调试结果表明：接口装置的开发满足需求的时序功能设计要求。分别解决了在原有硬件结构基础上的模块开发和新建硬件基础上的开发，实现了无时隙传输特性接口解决方案，奠定了后续在接口方案研究上的基础。

（2）进一步的研究工作

对于本接口的设计开发过程，本文内容主要是根据业务需求进行功能实现，对于接口和其他业务的兼容性问题并未作详细介绍。

就接口模块的功能而言，本文并没有将接口支持的功能完全开发，还需进一步阅读手册和观察市场上同类产品的新功能，在必要的时候提出需求，对接口功能进一步完善[30]。

综上所述，在接口装置的开发中，我们需要开展更加深入的工作，紧紧跟随国际先进水平，吸收国外先进技术，争取技术创新。

致谢

研究生的整个阶段，时光如梭，转瞬即逝。在论文即将完成之际，回想这篇论文，不论是那个方面，都已渗透出心血，可谓字里行间充满了感激。我的导师一直是我的良师益友，给了我非常的关心。从论文的选题，到课题的研究，再到最后的撰写、修改和定稿无不凝聚着马老师的关心，关注论文的每一个细节，力求严谨。他渊博的知识、严谨的治学态度、积极进取的精神和谦逊务实的工作作风都时时刻刻影响着我。在学习中，马老师悉心指导，循循善诱，使我的学术水平和实际动手能力都有了很大的提高；在工作中，马老师不辞辛劳，以身作则，为我作出了极好的榜样。从他身上，我们学到了许多书本上没有的东西，这些都将使我终身受益。

在论文即将完成之际，向尊敬的马老师表示衷心的感谢！

衷心感谢在我论文的撰写中的朋友，感谢曾经指导我完成开发接口卡的领导和同事，感谢瞿战、安靖、宋晓鹏、耿健、刘洋等。感谢陪同我的室友牛年增、胡俊杰、吴善永，对开发中的问题给了不少意见。感谢在给了我论文帮助修改的许加星、刘昌华、徐根同学。

感谢我的父母家人，给我无限的关心和支持，是他们给了我无尽的爱和无穷的动力。

参考文献

[1] 安靖. GEM 8 单板详细介绍. ZTE, 2006

[2] http://www.gdbear.com.cn "AMD 集成 DDR 内存控制器"

[3] http://www.hope.com.tw "Altera 推出 DDR2 SDRAM 解决方案"

[4] 耿健. 锁相环技术. ZTE, 2007

[5]张秀平，钟奇，刘成. 基于 FPGA 设计数字锁项环. 河海大学学报. 2007, 9, 21(3). 51-54

[6] http://article.pchome.net "内存发展历史"

[7] http://baike.baidu.com "DDR SDRAM 简介"

[8] 姜文波，李文成，王伟. 计算机科学计算体系. 北京：国防工业出版社，2006.1. 22-35

[9] 宋晓鹏. DDR SDRAM 时序简介. ZTE, 2006.2

[10] Qu Zhan. Synthesizable 266MBits/s DDR SDRAM Controller. ZTE Beijing Corporation, 2002

[11] DDR and DDR2 SDRAM Controller Compiler User Guide. Datasheet. ALTERA. 2006

[12] 宋晓鹏. 基于 StratixIIGX 系列 FPGA 的 DDR2 接口的 FIFO 设计方案. ZTE, 2008.7.28

[13] 宋晓鹏,李原. 一种实现无时隙传输特性的存储介质接口装置. ZTE, 2008.8-21

[14] Qu Zhan. Some experience about double data rate ram. ZTE Beijing Corporation, 2002

[15] 李原，宋晓鹏，刘洋. 基于 Altera 芯片的 DDR2 IP CORE 使用说明. ZTE, 2008.5-10

[16] 耿健等. FPGA 设计指导准则. ZTE, 2005

[17] 李原. 可编程器件仿真报告_156042. ZTE, 2008.7-28

[18] SubodhGupta, JasonAnderson. 使用 ISE 设计工具优化 FPGA 的功耗. Xilinx 公司设计软件部. 2000

[19]王义军，谢兵，张小刚. 可编程逻辑器件开发系统 ISE 的使用. 东北电力学院学报. 2005,8,25(4). 16-21

[20] 瞿战等. ISE 使用说明. ZTE, 2000

[21] Zhou Xiaojun. Manage your design by using HDL Designer Series. ZTE Beijing

Corporation. 2003

[22] 李原.可编程器件调试报告_156042. 2008,7-28

[23] 耿健等. 时钟设计指南. ZTE, 2005.5

[24]李萌，侯亚辉. 一种基于 Quartus 集成综合器的 FPGA 面积优化方法. 中国传媒大学学报. 2007,9,14(3). 57-61

[25]陈勇，王召. 有限状态机的建模与优化设计. 重庆工业学院学报. 2007,5,21(5). 55-59

[26]王志，石江宏，周剑扬. 同步数字复接的设计及其 FPGA 实现. 厦门大学学报. 2005,4,4(3). 70-74

[27] 李原. FPGA 设计中的一些经验总结. ZTE, 2008.5-22

[28]张晶，李佳妍，魏凤歧. 教学型 CPU 的设计与实现. 内蒙古大学学报. 2007,10,2(6). 87-90

[29]曾橡萍，赵海全. ISE 集成开发环境下基于 FPGA 的数字设计. 成都：成都信息工程学院硕士论文，2006

[30] Zhou Xiaojun. Advanced FPGA Design - Architecture, Implementation, and Optimization. ZTE Beijing Corporation, 2000