摘 要

可重构计算技术结合了通用处理器(General-Purpose Processor，GPP)和专用集成电路（Application Specific Integrated Circuits，ASIC）两者的优点，能够提供硬件的高效性和软件的可编程性，是当前热门的研究课题之一。动态部分可重构技术是可重构计算技术的最新进展之一。该技术能够在可重构系统正常工作的情况下，配置其中部分可重构资源，使得一部分任务的执行能够与另一部分任务的配置同时进行，具有节约硬件资源和增强系统灵活性的优点。

支持部分可重构的可编程逻辑器件是实现动态部分可重构技术的保证。目前采用最广泛的是基于 SRAM 实现的 FPGA（Field Programmable Gate Array），具有反复多次编程的优点。通过给 FPGA 加载不同的配置数据，就可以执行不同的硬件功能。其代表作是 Xilinx 公司 Virtex-II Pro 系列 FPGA。

虽然 Xilinx 公司对于 Virtex-II Pro 系列 FPGA 的动态部分重构技术提供了相当丰富的文档，但是动态部分重构技术比较新，还缺乏公认成熟可靠的设计流程，在一定程度上制约了动态部分可重构系统的开发。本文选择在 Xilinx Virtex-II Pro XC2VP30 平台上利用动态部分重构技术实现一个过程级动态部分可重构系统，对如何利用内部配置访问通道(Internal Configuration Access Port，ICAP)对 OPB(On-Chip Peripheral Bus)总线上的加密 IP(Intellectual Property)模块进行动态重构作了详细的介绍，主要完成的研究工作包括：

(1) 建立了一个可靠的，模块化的动态部分重构设计流程，包括建立初始硬件平台、静态与重构模块的划分与设计、模块生成与系统组装等步骤。

(2) 利用基于 Slice 的总线宏解决了静态模块与重构模块通信的关键问题。针对传统的基于 TBUF 的总线宏在通信效率和信号控制方面存在的缺陷，本文根据系统重构的需要实现了基于 Slice 的总线宏，实验表明，基于 Slice 的总线宏能更加有效的控制重构操作的执行。

(3) 按照本文提出的模块化设计思路完成了一个动态部分可重构系统的设计与实现，利用 XC2VP30 FPGA 的动态部分重构特性实现系统的运行时重构。实验表明系统实现了硬件资源的分时复用，提高了系统资源利用率。

关键词：动态部分重构；可编程逻辑器件；Virtex-II Pro XC2VP30 FPGA；OPB 总线；IP 模块；基于 Slice 的总线宏；模块化设计

Abstract

Reconfigurable computing technology combines the advantages of GPP (General-Purpose Processor) and ASIC(Application Specific Integrated Circuits), providing the hardware efficiency and software programmability in one platform. It is one of the hot topics of current computer research. As the most recent development of reconfigurable computing technology, dynamic partial reconfiguration can reconfigure a part of the reconfigurable logic device while other parts of the system continue to operate, enabling the parallel execution of running tasks and reconfigurations. It can efficiently utilize the reconfigurable resource and improve flexibility of the reconfigurable computing system.

Programmable logic devices with partial reconfigurable feature are the guarantee to the dynamic partial reconfiguration technology. Currently, the most widely used device is the SRAM-based FPGA(Field Programmable Gate Array), capable of repeatedly reprogramming. By loading different configuration data to FPGA, different hardware functionalities can be executed. Xilinx Virtex-II Pro Series FPGA is one of the key programmable devices supporting denamic partial reconfiguration.

Xilinx Inc. has provided a rich set of documents for the dynamic partial reconfiguration technology realized on the Xilinx Virtex-II Pro Series FPGA. However, since the dynamic partial reconfiguration technology is fairly new, there is hardly a well proven design flow for reliable dynamic partial reconfigurable system development, which in some extent, slows down the popularization of the dynamic partial reconfiguration technology.

The paper introduces how to use the ICAP (Internal Configuration Access Port) to reconfigure the encryption IP (Intellectual Property) modules on the OPB (On-chip Peripheral Bus), and implement the process level dynamic reconfigurable system on Xilinx Virtex-II Pro XC2VP30 FPGA platform in detail. The completed research work includes:

(1) Setup a reliable module-based dynamic partial reconfiguration design method, including the establishment of the initial hardware platform, the partitioning and design of static and reconfigurable module, module generation and system assembly.

(2) Use the slice-based bus macro to solve the key problem of the communication between static modules and reconfigurable ones. Considering the defects in efficiency

of communication and signal control provided by the TBUF-based bus macro, the paper realizes the slice-based bus macro according to the system reconfiguration requirement. Experiment shows that the slice-based bus macro is more effective to control the operation of reconfiguration.

(3) Following the proposed module-based design method, the paper completes the design and implementation of a dynamic partial reconfigurable sytem. The sytem achieves run-time reconfiguration on the XC2VP30 FPGA device. Experiments show that the system allows multiple design modules to time-share physical resources, improving the utility of FPGA hardware resources.

Key Words: Dynamic partial reconfiguration; Programmable logic devices; Virtex-II Pro XC2VP30 FPGA; OPB bus; IP module; Slice-based bus macro; Module-based design

插图索引

附表索引

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：赵远宁　　日期：2008 年 5 月 25 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本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和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密☑。

(请在以上相应方框内打"✓")

作者签名：赵远宁　　日期：2008 年 5 月 25 日

导师签名：[illegible]　　日期：08 年 5 月 25 日

第1章 绪 论

当前，传统计算模式的实现方式主要有ASIC和通用处理器两种。ASIC主要特点是由于为特定计算任务专门设计，因此能做到设计精确，可得到极高的运算速度及效率，设计电路的约束基本上是在物理层次的实现上，但其最大缺陷是在完成制造以后不能进行修改，一旦遇到不同的工作环境，就需要对ASIC重新设计，往往导致ASIC的设计成本高，灵活性差。通用处理器则恰恰相反，一般拥有一套属于自己的指令集。在执行任务之前都是将任务抽象为指令集，选择其中的指令依某种算法构成一个新的指令序列，就成了完成特定计算任务的软件，也就是通过用编程语言来实现不同的功能。这样通过修改软件便可达到改变系统功能的目的，对硬件无需做任何改动。虽然通用处理器拥有很高的灵活性，但是它的可编程性是以牺牲系统的性能和速度为代价换来的，常常导致执行效率低下，这是因为微处理在运算时，按预先设定好的指令序列逐条执行，而每条指令的执行一般都要经过“读取指令→译码→执行→存储→写回”五个步骤，这不仅增加了系统处理的时间，也使系统的复杂性增加。从上述的简要分析可以获知，当前主流计算模式的实现存在的主要问题是：处理器计算模式能够灵活地实现各种计算任务，但是在性能上存在缺陷；ASIC计算模式虽然性能较高，却不能够灵活地应对计算任务的改变。为了有效地解决计算效率和灵活性两者的权衡问题，自从上个世纪九十年代末至今，可重构计算技术逐渐成为研究的热点.

1.1 可重构计算技术概述

1.1.1 可重构计算技术定义

可重构计算技术是指数字系统制造完成以后，其硬件结构可以根据需要重新配置的技术，简单的说就是利用现场可编程门阵列FPGA来实现特殊功能的计算任务[1]。

早在 20 世纪 60 年代末，美国加利福尼亚大学的 GeraidEstrin 就提出了重构计算的概念，并研制了原型系统。该系统由非柔性但可编程的处理器和柔性的由程序控制重构的数字逻辑部件两部分组成。该系统其硬件和软件尽管抽象层次不高但均可编程重构。由于当时实现技术尚不完善，故 Estrin 研制的系统只是其理论设计的粗略近似。但这种结构奠定了以后可重构计算系统的核心基础[2]。

目前，能够被广泛接受的定义是由加州大学伯克利分校的Andre Dehon和John Wawrzynek提出的，他们通过与其它计算系统的组织结构比较的基础上，在广义

上给出了可重构计算系统应该具有的两个关键特点[3]：1)能够在硬件器件制造后针对计算任务进行定制；2)能够为计算任务提供大量的可以定制的执行空间。这两个关键特点能有效的弥补传统的ASIC模式不具备制造后可改动的能力以及通用处理器模式不能够为操作提供专用数据通路的缺陷，阐明了可重构计算系统必须同时具备灵活性和高效性等特征。

同时，Katherine Compton和Scott Hauck对当前研究的主流可重构计算系统进行了更具体的定义：可重构计算系统是结合了硬件编程能力的系统。其中，硬件编程是指通过改变一系列物理控制点的方式对硬件应该如何工作进行定制；控制点的改变将导致硬件的执行功效发生相应的改变[4][5]。

1.1.2 可重构计算体系结构

可重构计算系统的一种体系结构是由一个或者多个微处理器与其它一些专用处理模块一起集成到可重构逻辑模块里面；专用处理模块通过共享总线或者一些特殊的内部网络连接和微处理器核通信；微处理器和专用处理模块可能用到一些片上内存作为局部缓存；另外，由于可编程逻辑器件能够支持动态的重构，那么上述专用处理模块能够被动态的改变，所以需要相应的控制和管理组件来处理动态添加或删除的专用处理模块，以及重构之后内部的通信等问题。这种紧密耦合的体系结构图如图 1.1 中所示，其中专用硬件模块可以是 IP 核，也可以是自行设计的硬件模块，用虚线框表示说明专用处理模块可动态改变。

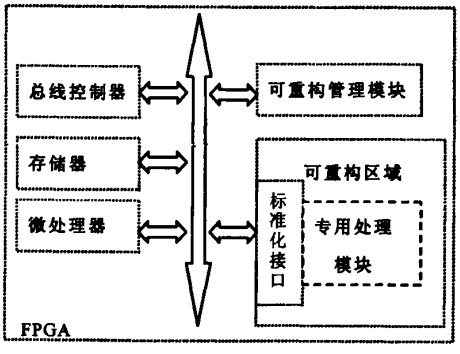


图 1.1 系统结构图

1.1.3 可重构计算技术应用

尽管可重构计算的概念早在 70 年代就已经提出，但由于没有理想的硬件条件，这方面的研究没有取得很大突破。90 年代以来，随着大规模集成电路的迅速发展，可编程逻辑器件的出现为可重构计算的研究提供了条件，陆续实现了一些可重构计算的应用系统。

在各 FPGA 制造商相继推出的一系列支持部分重构的 FPGA 器件中，早期的 FPGA 结构中只包含少量的逻辑块，通常是由细粒度的可编程逻辑块通过走线和

可编程开关相连而组成的。但随着现代工艺的进步，现在的FPGA集成度得到了明显的提高，甚至达到了数百万门的单片FPGA芯片的新水平。这些技术的高速发展为以前只是用于实现简单的逻辑功能和原形系统设计的可重构逻辑器件能够逐步占领计算系统的核心地位提供了基本支持。

可重构计算利用了可编程器件可多次重复配置逻辑状态的特性，能以较少的硬件资源实现较复杂的逻辑电路功能，在提高系统执行速度的同时又显著地降低系统成本。尤其在对执行速度和灵活性要求比较高的场合，例如集成电路的计算机辅助设计[6]、大数运算[7]、目标识别[8]、字符模式匹配[9]、数据压缩[10]、嵌入式系统等方面都有着广泛的应用前景。

1.2 部分重构技术概述

可重构计算技术根据重构过程的行为差异可分为静态重构和动态重构两类[11]。

1.2.1 静态重构

静态重构也称全部重构，是最简单也是当前用途最广泛的利用可重构逻辑器件实现计算任务的技术。它的特点是每个计算任务都需要一个涵盖整个器件的配置文件。在开始执行计算任务前，这个配置文件一次性全部编程到可编程逻辑器件的逻辑资源上。一旦系统开始运行，那么在系统的整个生命周期内，可编程逻辑器件上的配置必须保持静态而不发生任何改变。

实际上，静态重构的执行过程非常类似于ASIC计算模式：在计算任务执行的整个过程中，它们都不会对硬件做任何改变。而相比较ASIC计算模式，静态重构的优势在于能够多次复用同一器件实现不同的计算任务，减少了任务实现的代价。

1.2.2 动态重构

动态重构在系统运行之前对FPGA进行配置，在系统运行的过程中根据需要可重配置FPGA，使之具有新的逻辑功能，从而实现系统重构。这样，系统所能支持的配置只依赖于保存配置信息的存储器容量的大小。

动态重构按照系统重构的粒度可分为动态全局重构和动态部分重构。

（1）动态全局重构

动态全局可重构在进行重构时，需对整个FPGA芯片进行重新配置，在重配置的过程中芯片停止工作，系统旧的逻辑功能失去，新的逻辑功能尚未建立，系统的逻辑功能在时间轴上断裂，系统的功能无法动态连续。如图1.2所示为动态全局可重构的工作流程。

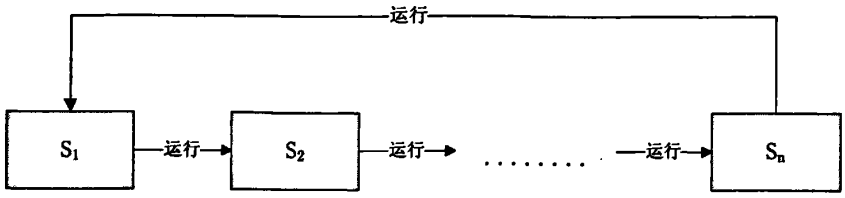


图 1.2 动态全局可重构系统

（2）动态部分重构

动态部分重构在进行重构时，由于选择性地对可编程逻辑器件上的部分资源进行重新配置，而不会影响到器件上的其它资源，所以在重配置的过程中芯片仍然工作。系统在建立新的逻辑功能的过程中，未被重配置部分的逻辑功能仍然正常，即系统的逻辑功能在时间轴上是动态连续的。目前，对动态部分重构研究使用的FPGA主要是Xilinx公司的支持动态部分重构的Virtex/Virtex-II/Virtex-II Pro/Virtex 4/Virtex 5等系列。本文使用的实验平台就是Virtex-II Pro系列的产品。如图1.3所示为动态部分重构工作流程。

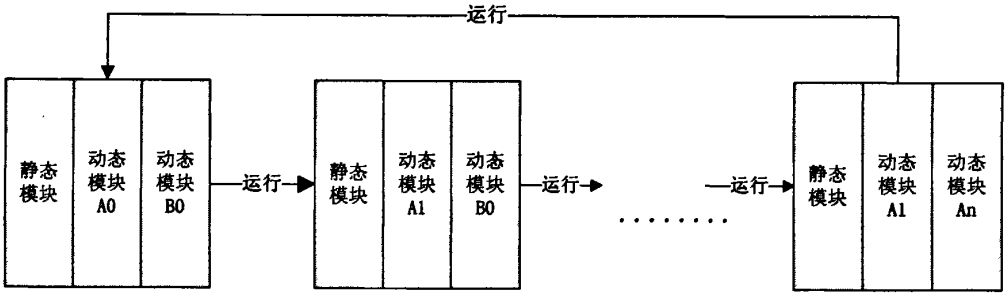


图 1.3 动态部分可重构系统

基于FPGA的动态部分重构是通过将一个纯空间的数字逻辑系统化解为在时间空间上混合构建的数字逻辑系统。这种新型的数字逻辑系统从时间轴和外部看上去，和原来整体功能一样；但从资源利用来看，由于可以动态地重复利用资源，资源的利用率将成倍地提高，实现的数字逻辑系统规模受硬件资源的限制相对要小得多[5]。

1.3 本文主要工作

目前国内外对于动态部分可重构计算技术的研究十分活跃，有大量的研究工作和论文集中讨论这一问题,所提出的编程模型可基本分为两类:指令级和进(线)程级。指令级和进（线）程级编程模型各有优劣。

指令级编程模型的主要优点在于完全不用设计人员干预，编译器、操作系统都可以不变，由专门硬件完成，对于设计人员而言开发效率很高。但由于需要额外的专用微处理器进行在线反汇编、综合和布局布线，硬件开销比较大。另一方

面指令级编程模型粒度目前局限在基本块内，结构也局限于单循环结构，性能提升效果有限。另外，虽然设计人员只需提供软件实现方案，而把硬件综合交给专门软硬件执行，但这样未能利用已有的硬件设计资源，这在目前第三方 IP 核十分丰富的情况下[12][13][14][15]，显得有些浪费。

进（线）程级编程模型优点主要在于把软件和硬件任务都当作进程或线程，由操作系统统一管理，可以支持现有硬件设计资源，便于集成开发。进（线）程编程模型可转换为进（线）程动态调度问题，利用和扩展操作系统进（线）程管理功能而实现。同时，进（线）程通信、同步也可利用操作系统提供的相应机制实现，较为灵活方便。但相对指令级编程模型而言，进（线）程调度、通信、同步基本由软件完成，时间开销较大。另外，目前的进（线）程级编程模型研究中，软件和硬件进（线）程对设计人员是可见的，程序员创建一个软件或硬件进（线）程的同时，实际已暗示了相应功能的软硬件划分，虽然有可能增加专门代码和电路来进行软硬件进（线）程迁移[16][17]，但需要同时修改操作系统和底层硬件以提供支持，并非一个理想的方案。

针对指令级和进（线）程级编程模型的不足，提出了过程级编程模型。

过程级编程模型主要为设计人员提供统一的软硬件过程描述方法。这通过软硬件协同函数库的形式来体现。其中关键的问题是硬件设计资源，如 IP 核、自行设计的电路模块等的封装。

软件过程或函数的描述形式与当前软件程序中的过程或函数是相同的，无需修改。硬件设计资源则需根据其实现的功能，封装为软件的过程或函数形式。考虑到在可重构片上系统中，硬件模块一般作为微处理器的加速器，并通过总线连接到微处理器和存储器。硬件模块的配置文件一般是从硬件描述通过硬件设计综合工具获得，并保存到非易失性存储器中，以便需要运行时加载配置到可编程器件资源上。加载配置过程由硬件配置管理器来负责。这样，一个硬件模块的实现可以分为两部分，一部分是配置文件，实现硬件模块的基本功能，一部分是软件代码，实现微处理器对硬件模块功能的调用。因此，硬件模块的封装需要保证硬件的接口过程或函数通过正常的编译流程后生成相应的代码，访问对应的硬件配置文件和接口。

本文在 Virtex-II Pro XC2VP30 FPGA 实现了一个过程级动态部分可重构系统，通过使用内部配置访问通道 ICAP 对 OPB 总线上的加密 IP 核进行重构配置。

系统借鉴了模块化设计技术的思路，对模块化设计过程中涉及的模块划分、模块通信和硬件实现三个关键问题进行了研究。

模块划分阶段，为了部分重构系统的易实现性，将系统划分为重构模块和静态模块两部分。重构模块在运行时，根据系统需求的变化在静态模块的 PowerPC 控制下做功能切换。由于 XC2VP30 FPGA 只支持一维重构，因此对重构模块硬件

资源的分配需作特殊处理，需为模块进行合理的区域约束。

重构模块和静态模块间的通信是动态部分重构系统的关键，实验中，使用了基于 Slice 的总线宏。基于 Slice 的总线宏提供了更大的信号通信量，最重要的，由于增加了信号控制机制，能更加有效的控制重构阶段来自重构区域的无序信号，保证系统的运行时重构。同时，基于 Slice 的总线宏在 FPGA 片内的放置对于整个系统的模块设计实现阶段至关重要，易产生信号布线越界的现象，需用户对总线宏进行约束。

经过初始预算、模块激活、模块合并和配置数据生成阶段得到系统的硬件实现，可实现系统的运行时重构，有效的提高系统资源利用率。

1.4 本文组织结构

本论文共分四章，其组织结构为：

第1章：绪论。论述本论文的相关研究背景。首先系统地介绍了可重构计算技术的定义、体系结构以及应用。然后通过对可重构计算的比较，得出动态部分重构技术是可重构计算技术的重要发展趋势。

第2章：可重构计算技术基础。首先介绍了实现动态局部重构系统的Virtex-II XC2VP30 FPGA硬件开发平台，特别是系统中所用到的硬件组件PowerPC405、ICAP、System ACE。通过分析，具有混合粒度的支持部分重构的器件将是未来可重构逻辑器件的重要发展方向。

第3章：基于模块化设计的部分重构方法。介绍了动态局部重构系统的设计技术以及各自的优缺点。分析了Xilinx基于模块化的部分重构设计方法的关键技术、设计流程等。

第4章：过程级动态部分重构系统设计与实现。提出过程级动态部分重构系统的设计方法，并以系统在加密领域为例进行系统的设计与实现.首先简述过程级动态部分重构系统的概念，介绍了与传统动态部分重构系统的差别；然后分析过程级动态部分重构系统设计过程中需要涉及的关键技术；通过在现有的基于可重构逻辑器件的嵌入式系统开发流程上增加了对基于模块的部分重构技术的支持，针对系统的设计需求进行系统部件的选型与设计，解决了过程级动态重构系统所涉及的关键问题;通过加密IP模块系统的实现验证了本论文所提设计方法的有效性，也体现出了过程级动态部分重构系统具有的性能优势。

第2章 可重构计算技术基础

2.1 可重构逻辑器件的基本技术

2.1.1 可重构逻辑器件的编程原理

从上章内容易知，可重构计算技术需要依赖于硬件器件的可编程重构特性，以便在计算过程中能够将硬件器件定制为计算任务所需的功能部件。因此，具有可重构能力的可编程逻辑器件是可重构计算技术的基础。这是因为可重构计算的关键在于电路结构可以动态改变，这只有通过采用合适的可编程逻辑器件来实现这一功能。

常见的可编程逻辑器件有：可编程阵列逻辑(Programmable Array Logic，PAL)、复杂可编程逻辑器件(Complex Programmable Logic Device，CPLD)、现场可编程门阵列(Field Programmable Gate Array，FPGA)和动态现场可编程门阵列(Dynamically Field Programmable Gate Array，DFPGA)。上述各种器件都可满足可重构计算中关于可变硬件的条件，但实际应用中，除了FPGA外其他器件很少在可重构系统中使用，这是因为FPGA具有更高的集成度和更大的灵活性，能够高效地实现非常复杂的电路逻辑，甚至在单片器件中配置计算系统的全部硬件部件。

FPGA 的编程技术主要有反熔丝、Flash 以及 SRAM 三类[18]：

（1）反熔丝 FPGA 由专用的编程器根据设计所给出的数据文件，对其内部的反熔丝阵列进行烧录，从而达到逻辑功能的实现；具有速度快、功耗低、安全性好等优点，但其有致命的弱点，即只具有一次性编程(One Time Programmable，OTP)能力，因此不适合可重构计算技术的研究。

（2）基于 Flash 的 FPGA 是一种比较新的技术，它集成了 SRAM 和 Flash 两类存储结构。SRAM 用于器件正常工作时对系统进行控制，Flash 则在掉电后用来保存配置信息，因此不需要片外的配置芯片，充分发挥了 Flash 的非易失性和可重配置性，但配置速度不够理想。

（3）基于SRAM的FPGA中的各个配置点都连接着SRAM存储位，FPGA的配置过程就是对这些SRAM存储位的编程过程，如图2.1所示[4]：

（a）可编程逻辑位的示意图　　（b）可编程互连的示意图

图 2.1 基于 SRAM 的 FPGA 编程原理

在图2.1(a)中,FPGA片内的每个可重构逻辑位对应着一个SRAM存储位:由四个晶体管组成的两个交叉耦合的反相器，可具有两个稳定状态0和1。FPGA最基本的逻辑单元就是利用数个这样的可重构逻辑位构成的。在图2.1(b)中，FPGA上的互连布线资源是通过使用通门(passgate)结构来配置的:如果可编程的SRAM存储位P为1，则布线资源#1与#2连通；否则就不连通。FPGA中纵横交错的各种布线线段进行连接，可以灵活地改变各基本逻辑单元间的互连关系，组成更为复杂的计算电路。

FPGA 配置过程中，上电时要将配置数据写入片内 SRAM 中，配置完成后才可进入工作状态。掉电后 SRAM 中的配置数据会丢失，FPGA 内部逻辑关系也随之消失。因此，基于 SRAM 的 FPGA 可反复使用，具有可重编程能力，而且配置速度快，迄今为止也是可重构计算领域应用最广泛的。

2.1.2 可重构逻辑器件的分类

按照不同的分类标准，可以将可重构逻辑器件分为不同的类型，下面介绍一些常用的分类标准[19]。

（1）粒度

粒度是指系统中可重构成分（Reconfigurable Processing Unit，RPU）的操作数的宽度。RPU是功能可配置的逻辑块，其内部连接关系是可重构的。在一个细粒度的可重构体系结构中，RPU中的处理元素通常是逻辑门、触发器、查找表等，它们进行位级操作，实现一个有限状态机的布尔函数。而在粗粒度的可重构体系结构中，RPU中的处理元素可能包括复杂的功能单元，如算术逻辑单元、乘法器等，它们一般处理字级的操作。如果一个可重构体系结构中包括粗粒度和细粒度两种RPU，则称其为混合粒度的。

（2）编程深度

编程深度是指存储在RPU中的配置程序或文件的数量。一个RPU可能含有单个配置文件或多个配置文件。对于单配置文件系统，只有一个配置文件驻留在RPU内，因此RPU的功能局限于当前装载的配置文件。而在多配置文件系统中，同时

有多个配置文件驻留在RPU内，这使得可以通过切换配置文件很方便地实现不同的功能，而不必从外部重新装载配置文件。

2.2 实验环境介绍

本文研究加密模块的动态部分重构技术实现所使用的硬件平台是Xilinx公司提供的XUP Virtex-II Pro开发系统，其中主芯片是Xilinx Virtex-II Pro系列的XC2VP30型号FPGA。

2.2.1 Xilinx Virtex-II Pro FPGA

Xilinx Virtex-II Pro FPGA 是基于 SRAM 编程技术的可编程逻辑器件，具有先进的体系结构和多样的配置方式。下面将简要介绍其体系结构、配置方式以及配置原理。

1．体系结构

Virtex-II Pro FPGA 内部结构如图 2.2 所示[20]：

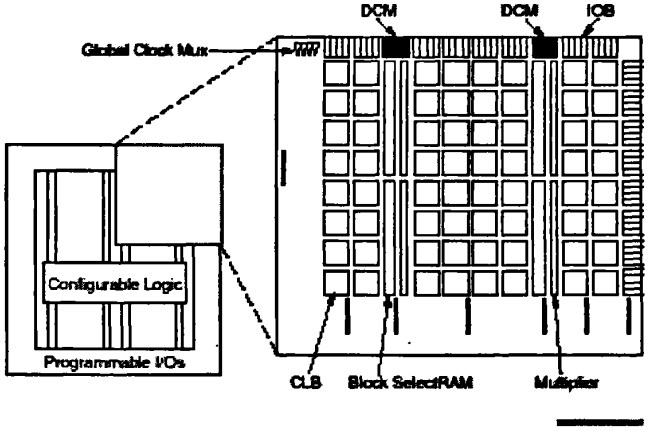


图 2. 2 Virtex-II FPGA 的内部结构

其包含可编程功能模块(Configuration Logic Blocks，CLB)、输入输出模块(Input/Output Blocks，IOB)、时钟管理模块(Digital Clock Manager，DCM)、嵌入块式 RAM、丰富的布线资源和内嵌专用硬件模块。

（1）CLB

CLB 是 FPGA 的核心，可用于实现组合逻辑和复杂时序逻辑，其结构如图 2.3 所示[20]：

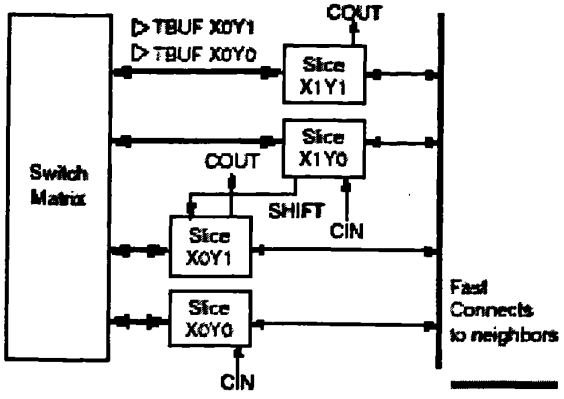


图 2. 3 CLB 结构

每个 CLB 包括 4 个 slice、2 个三态缓冲(TBUF)，以及内部的快速互连机制与开关矩阵(Switch Matrix)相连并最终连接 FPGA 上的布线资源。CLB 中的每个 slice 包括两个函数发生器、两个存储逻辑、算术逻辑、进位逻辑和函数复用选择器，其结构如图 2.4 所示[20]。

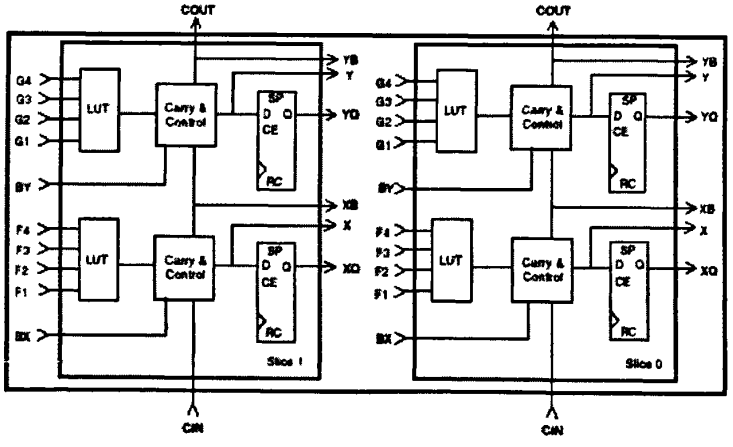


图 2. 4 slice

函数发生器采用了基于 SRAM 存储器的 4 输入查找表(Look Up Table，LUT)结构，能够根据输入的地址在相应的存储空间中读取数值以完成计算功能。系统的重构配置过程中，对 LUT 的重构就是对 SRAM 存储器重新编程的过程；两个存储单元是由边沿触发的 D 触发器或者是电平敏感的锁存器构成，主要被用于实现时序逻辑；算术逻辑包括 1 个异或门和 1 个专用与门。异或门可实现两个 slice 的 2bit 全加操作，专用与门则用于提高乘法器的效率；进位逻辑由专用进位信号和函数复用发生器组成，以实现快速的算术加减法操作。

CLB 中的三态缓冲主要用于驱动水平方向上的布线资源，构成专用的固定通讯通道，在动态局部重构系统中可用来实现基于 TBUF 的总线宏，以提供重构模块和其他模块的通信连接。

（2）IOB

IOB 是 FPGA 与外界电路的接口部分，完成不同电气特性下对输入/输出信号的驱动与匹配要求，其结构如图 2.5 所示[20]：

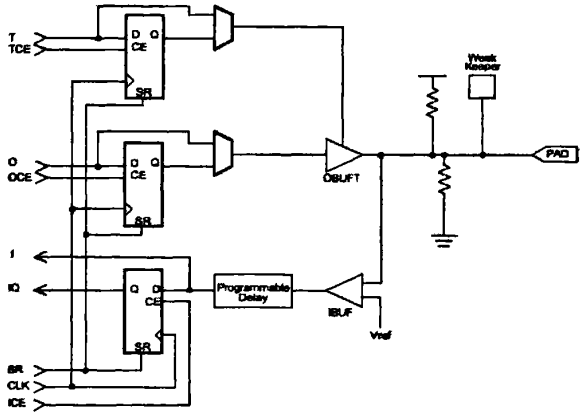


图 2. 5 IOB 内部结构

外部输入信号可以通过 IOB 的存储单元输入到 FPGA 的内部，也可以直接输入到 FPGA 内部。当外部输入信号经过 IOB 的存储单元输入到 FPGA 内部时，其保持时间(Hold Time)通常默认为 0。

FPGA 内部的 IOB 按组(Bank)分类，每组都能够独立支持不同的接口标准，由其接口电压 VCCO 决定。

（3）DCM

数字时钟管理器 DCM 提供了大量有效的时钟管理功能，包括无扭曲时钟信号生成、频率合成和时钟移相。DCM 使用全数字延时线，产生高精度的时钟相位和频率控制。

（4）嵌入式块 RAM

块 RAM 可被配置成单端口 RAM、双端口 RAM、内容地址存储器以及 FIFO(First In First Out)等常用存储结构。

单片 RAM 的容量是 18K 比特，即位宽为 18 比特、深度为 1024。用户根据需要改变单片 RAM 的位宽和深度时要遵循修改后的容量不能大于 18k 比特以及位宽最大不能超过 36 比特。当将多片块 RAM 级联起来形成更大的 RAM 时，只受限芯片内块 RAM 的数量而不是前述两条原则。

（5）布线资源

布线资源连通 FPGA 内部所有单元，根据工艺、长度、宽度和分布位置的不同常分为 4 类[21]：第一类是全局布线资源，用于片内全局时钟和全局复位/置位的布线；第二类是长线资源，用来完成芯片 Bank 间的高速信号和第二全局时钟信号的布线；第三类是断线资源，用于完成基本逻辑单元之间的逻辑互连和布线；第四类是分布式的布线资源，用于专有时钟、复位等控制信号线。

在实际运用中，布线是在布局布线器的自动控制下根据输入逻辑网表的拓扑结构和约束条件选择布线资源来连通各个模块单元，不需要用户直接选择所需布线资源。

（6）内嵌专用硬核

为了提高 FPGA 性能，FPGA 内部集成了一些专用硬核。譬如：为了提高 FPGA 的乘法速度，片内集成了专用乘法器(Multiplier)；特别是集成了 PowerPC 系列的硬微处理器，在系统级设计工具 Xilinx EDK(Embedded Development Kit)和 Platform Studio 支持下提供片上系统(System on chip)的开发。

2．配置方式

Xilinx Virtex-II Pro 系列 FPGA 支持 5 种配置方式[22]：从串模式(Slave Serial Mode)、主串模式(Master Serial Mode)、从并模式(Slave SelectMAP Mode)、主并模式(Master SelectMAP Mode)和边界扫描模式 JTAG(Boundary-Scan Mode)。其中，主从模式的区别在于配置时钟是否由 FPGA 提供：主模式，FPGA 的配置时钟由 FPGA 提供；从模式，FPGA 的配置时钟由微控制器或者微处理器提供。串并模式的区别在于每个配置时钟周期内传递的配置数据的宽度是 1bit 还是 8bit。

（1）从串模式：其实现方式是读取串行 PROM(Programmable Read-Only Memory)里的配置数据流，通过串行模式实现 FPGA 的在线配置。从串配置模式的配置时钟 CCLK 由外接的微控制器或者微处理器提供。从串配置模式可以使用通用的并行、串行 PROM。从串配置模式支持 Xilinx 全系列 FPGA。

（2）主串模式：其实现方式是读取串行 PROM 里的配置数据流，通过串行模式实现 FPGA 的在线配置。主串配置模式的配置时钟 CCLK 由 FPGA 产生。主串配置模式必须使用 XILINX 公司专用的 PROM。主串配置模式支持 Xilinx 全系列 FPGA。

（3）从并模式：其实现方式是读取并行 PROM 里的配置数据流，通过并行模式实现 FPGA 的在线配置。从并配置模式的配置时钟 CCLK 由外接的微控制器或者微处理器提供。从并配置模式可以使用通用的并行、串行 PROM。从并配置模式支持 XILINX 全系列 FPGA。

（4）主并模式：其实现方式是读取并行 PROM 里的配置数据流，通过并行模式实现 FPGA 的在线配置。主并配置模式的配置时钟 CCLK 由 FPGA 产生。主并配置模式必须使用 XILINX 公司专用的 PROM。主并配置模式支持 XILINX 全系列 FPGA。

（5）边界扫描 JTAG 模式：边界扫描 JTAG 模式是基于 IEEE1149.1 和 IEEE1532 的下载配置模式。其通过 TDI(数据输入)、TDO(数据输出)、TMS(测试模式)、TCK〔测试时钟)四根信号线通过串行模式实现 FPGA 的下载配置。边界扫描 JTAG 模式常需要微控制器或者微处理器控制 FPGA 的配置进程，一般采用

PC 机。边界扫描 JTAG 模式支持 XILINX 全系列 FPGA。

FPGA 的配置模式由配置模式管脚 M0、M1、M2 的高低电平决定。FPGA 配置模式设定如表 2.1 所示：

表 2. 1 FPGA 配置模式参数设定

配置模式	M0	M1	M2	时钟 CCLK	Data Width	Dout
Slave Serial	1	1	1	CCLK Input	1	Yes
Master Serial	0	0	0	CCLK Output	1	Yes
Slave SelectMAP	0	1	1	CCLK Input	8	No
Master SelectMAP	1	1	0	CCLK Output	8	No
JTAG	1	0	1	TCK input	1	No

3．配置原理

Virtex-II Pro FPGA 上所有可重构逻辑资源的配置数据都保存在配置缓存单元(Configuration Memory Cells)中，它们定义了 LUT 的内容、各种内部连线和 IOB 的电平特性等信息。

配置数据是以帧(Frame)的形式存在的，每一帧的宽度是 1bit，高度与 FPGA 等高。配置数据帧是 FPGA 对配置信息进行读写的基本单位，因此所有的配置操作也是以一个完整的帧为单位。

Virtex-II Pro FPGA 内部的所有帧分成六类：IOB、IOI、CLB、GCLK、BlockRAM 和 BlockRAM Interconnect[23]。图 2.6 描述了 Virtex-II Pro FPGA 内部帧的分布以及它们的分类信息[23]。

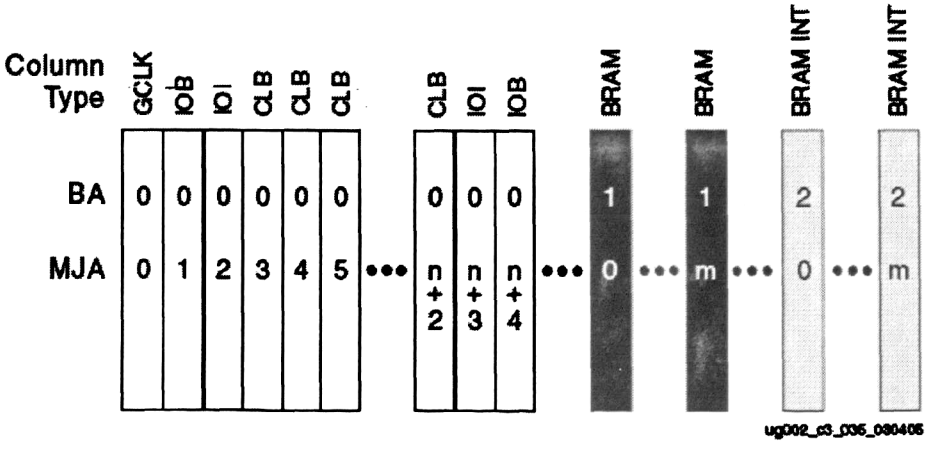


图 2. 6 Virtex-II Pro 内部帧结构和分类

从图 2.6 易知，具有越多逻辑资源数量的 FPGA 相应地具有更多的配置数据帧数量，每帧中包含的配置数据量也就越大。

配置数据帧还可以被组装成更大的配置数据列(Column)单元，每个配置数据列对应于 FPGA 一整列逻辑资源的配置信息，包括 IOB、CLB、BlockRAM 等。

对于可下载到 FPGA 片内的比特数据流(Bitstream)文件其实就是是以配置数据帧为基础存储配置信息，只是会在这些配置数据帧头部加上一些额外的信息，然后形成一个个配置包。因此 Bitstream 可以看成是这些配置包的集合，用户通过 JTAG、并行或者串行配置模式之一下载 Bitstream 文件就可以改变 FPGA 的逻辑功能，实现动态部分重构。

2.2.2 XUP Virtex-II Pro 开发系统

XUP Virtex-II Pro 开发系统集合 Xilinx Virtex-II Pro 系列的 XC2VP30 型号 FPGA 以及环绕 FPGA 的各种外设组件，非常适合于复杂应用系统的设计，如图 2.7 所示[24]：

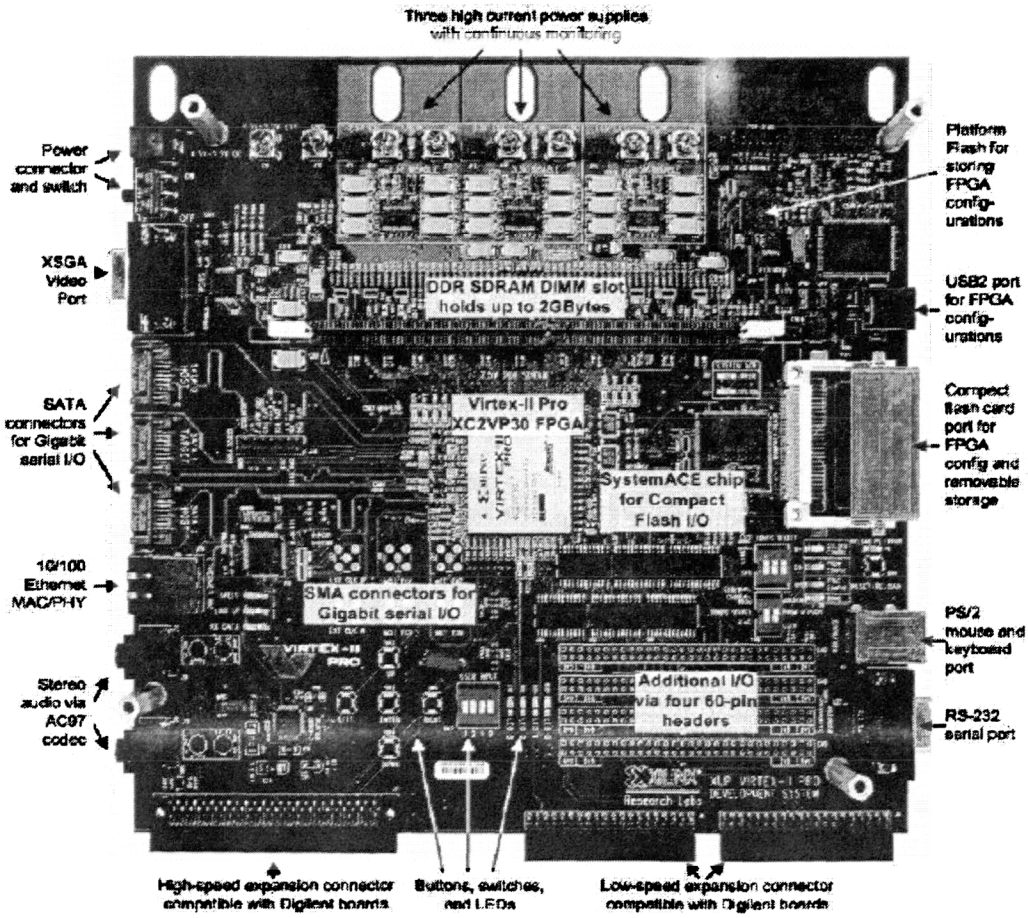


图 2. 7 Virtex-II Pro 开发板

XC2VP30 FPGA 是 Xilinx 在 2002 年推出的产品，其采用集成 RocketI/O 和 PowerPC 所采用的高水平系统集成技术，主要用于基于 IP 核和传统模块的设计。它先进的设计模式使可编程技术的使用模式从逻辑器件层次提升到系统级，用户可以在系统结构一级利用可编程性所提供的优点来构建动态部分重构系统。同时，它可以采用由上位机控制的基于 USB 技术实现的边界扫描配置模式或者脱离上位机控制的从板上 Flash 获得配置数据的并行主模式进行重构。其片内的各项资源具体情况见表 2.2：

表 2. 2 XC2VP30 片内资源

属性	slice	队列数目	分布式RAM	乘法器	BRAM	DCM	处理器核	MGT
值	13969	80×46	428Kb	136	2448Kb	8	2	8

开发板上集成的外设器件有[24]：

- 一片集成了两个PowerPC 405硬核微处理器的Virtex-II Pro XC2VP30 FPGA
- 一个支持高达2G的DDR SDRAM的插槽
- 一个System ACE控制器和CF卡连接器，用于FPGA的配置或者存储数据
- 一个下载配置数据的USB接口
- 系统可编程配置存储器PROM支持高速SelectMAP的方式配置方式
- 通过片上模式选择开关，支持“Golden”和“User”两种配置比特流
- 一个10/100M 以太网接口
- 一个RS-232 DB9串口
- 两个PS-2串口
- 四个连接到Virtex-II Pro输入输出管脚的LED
- 四个连接到Virtex-II Pro输入输出管脚的switches
- 五个连接到Virtex-II Pro输入输出管脚的push buttons
- 六个扩展连接器连接到80个Virtex-II Pro输入输出管脚，提供over-voltage保护
- 高速扩展连接器连接到40个Virtex-II Pro输入输出管脚，可应用在微分或者单独终端中
- 一个AC-97音频解码器，支持speaker/headphone输出以及线性输出
- 麦克风和线性音频输入
- 板上XSGA输出，70Hz的刷新率下可达到1200×1600
- 三个串行ATA端口，两个主机端口，一个目标端口
- 支持用户可供时钟的板外扩展MGT连接
- 支持100MHz系统时钟，75MHz SATA时钟
- 支持用户可供时钟
- 一个电池插座
- 开机及PowerPC 405复位电路

2.2.3 平台重要组件

本文实现的加密模块在动态部分重构系统中，是挂载在OPB总线上的IP核。在系统对加密模块进行重构的过程中，通过微处理器PowerPC从外部存储器

Compact Flash读取部分配置文件，然后利用ICAP将部分配置数据下载到FPGA中来完成重构模块间的功能置换。因此我们这里简单介绍与本文实验有密切关系的PowerPC405硬微处理器、内部配置方位通道ICAP和System ACE。

1．PowerPC405 硬微处理器

对于 XC2VP30 FPGA，Xilinx 公司支持两款 32 位的嵌入式处理器内核。一种是 PowerPC405 嵌入式处理器硬核(这是业界目前唯一的嵌入式硬核)，另一种是 MicroBlaze 嵌入式处理器软核。硬核的好处是能够提供更快的数据处理能力，而软核则具有更好的灵活性，在目标器件中可以进行任意配置。由于硬核在速度和资源上具有优势，因此本文采用了 PowerPC405 硬微处理器。

PowerPC405 硬微处理器是 IBM 和 Xilinx 公司合作，利用 IP(IP Immersion)植入技术，嵌入在 Virtex-II Pro 系列 FPGA 中的高性能 32 位 RISC 处理器。这种植入方法允许硬 IP 核心分布在 Virtex-II Pro 结构中的任何位置，同时可保持与周围逻辑阵列的平滑集成。IP 植入技术将核心中的所有高速总线与可编程结构直接密切耦合，从而获得了比同样的分立处理器高得多的系统级性能。图 2.8 所示为 PowerPC405 处理器模块的结构图[25]，有关 PowerPC 处理器的详细内容请看参考文档[25][26]。

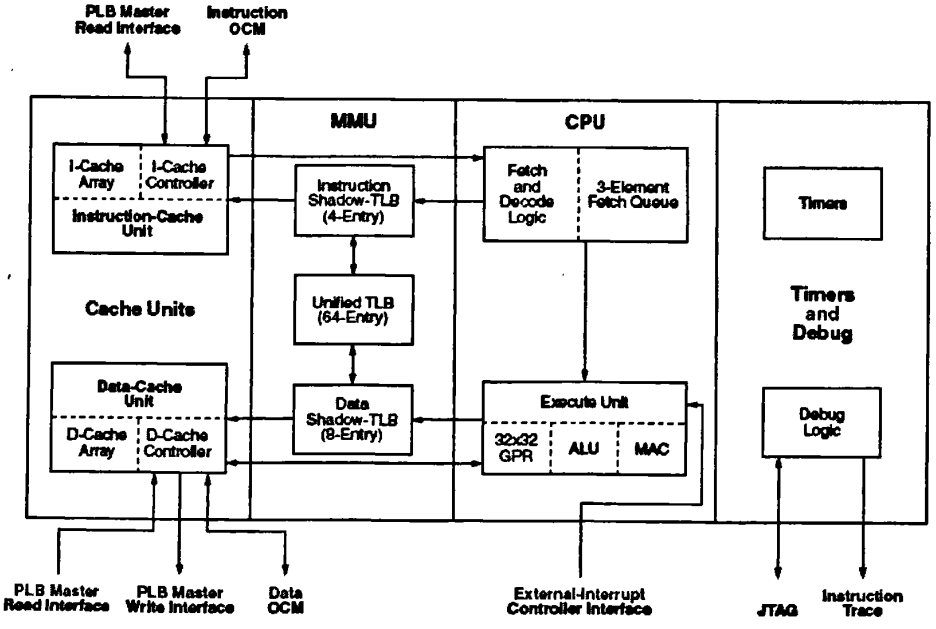


图 2. 8 PPC405 IP Core 结构框图

该处理器内核具备以下特征[25][26]：

- 32 位、Harvard 结构，300MHZ 以上工作频率
- 支持 IBM Core Connect 总线标准
- 符合 PowerPC UISA 标准

- 低功耗：0.9mW/MHZ
- 硬件乘法和除法单元
- 32 个 32bit 通用寄存器
- 16KB 双端口程序缓存
- 16KB 双端口数据缓存
- 内存管理单元(MMU)支持
- 独立的调试和跟踪接口

2．ICAP

传统的可重构计算系统中用于控制重构操作的处理器一般位于可编程逻辑器件之外，因此它只能够通过器件的外部配置端口(如 JTAG 端口、SelectMAP 端口)通过使用编程电缆(Programming Cable)对配置数据进行读出与写入。但这种配置方式受到系统外设性能的影响，效率较低，而且只能够进行全器件或者以列为单位的重构操作。

为了弥补上述不足，Xilinx Virtex-II 系列以后的 FPGA 都提供了 ICAP，这使得 FPGA 中内嵌的处理器能够直接在可编程逻辑器件内部对其配置数据进行操作。ICAP 在 FPGA 内部是以硬件原语的方式存在，允许用户实现 FPGA 的配置以及配置数据的回读。ICAP 使用的协议与 SelectMAP 的从协议类似，如图 2.9 所示：

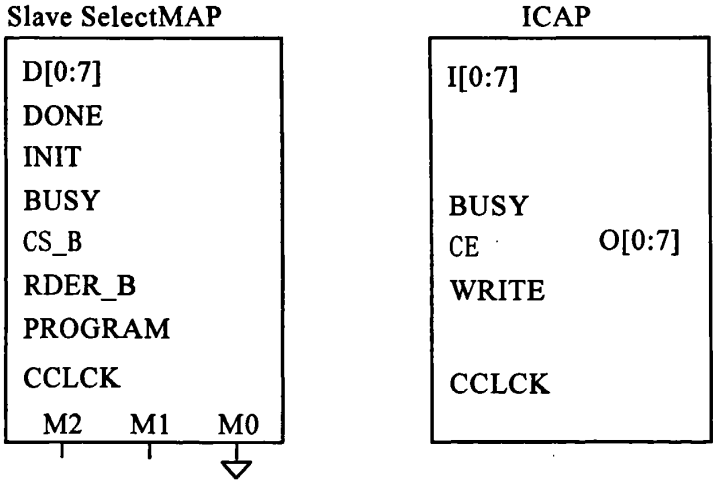


图 2. 9 Slave SelectMAP 模式和 ICAP 模式

ICAP 的接口与 Slave SelectMAP 接口的对照如表 2.3 所示。它们的主要区别是 ICAP 是内部连接可访问，SelectMAP 则需通过 FPGA 外部引脚来控制。

表 2. 3 ICAP 端口以及对应 SelectMAP 端口

ICAP		对应SelectMAP端口
信号名称	信号方向	
I[7:0]	Input	写模式下的D[0:7]
CLK	Input	CCLK
CE	Input	CS
WRITE	Input	RDWR_B
O[7:0]	Output	读模式下的D[0:7]
BUSY	Output	BUSY

使用 ICAP 要遵循以下原则[23]：

（1）FPGA 完成配置后，ICAP 和 SelectMAP 不能同时使用。

- 如果配置文件中 PERSIST 位被设置为 1，选择 SelectMAP 方式完成配置后，信号 DONE 呈高电平，ICAP 将处于不工作的状态。
- 如果配置文件中 PERSIST 位被设置为 0，选择 SelectMAP 方式完成配置后，SelectMAP 的 D[0:7]、RDWR_B、CS_B、BUSY 和 INIT_B 端口将呈用户可用 I/O 状态，因此此时 ICAP 处工作状态。

（2）当系统中同时存在 JTAG 接口和 ICAP 接口时候必须注意：

- 在 ICAP 执行配置或者回读配置数据操作时，JTAG CFG_IN、CFG_OUT、JSTART、JSHUTDOWN 指令不能直接送 FPGA。
- 通过 JTAG 接口完成配置或者回读配置数据操作后，在进行任何 ICAP 操作以前配置逻辑需保持同步；反过来，使用 ICAP 接口完成配置或者回读配置数据操作后，在使用 JTAG 接口进行任何操作以前配置逻辑也需保持同步。

3．System ACE

从上文易知，由于本文实现的动态局部重构系统中 ICAP 使用的限制性，对于系统初始配置文件的下载方式要采用 PROM 方式。在初始配置文件比较小的情况下，PROM 方式能满足用户的需求，但数据规模较大的初始配置文件时，仅利用片上有限的 PROM 资源是远远不够的。因此实验中用到了系统级的 System ACE 解决方案。

实验中，System ACE 使用 Flash 存储卡保存部分配置数据，通过 System ACE 控制器把数据写入到 HWICAP IP 核的缓冲存储空间中。

System ACE CF 的核心是 System ACE CF 存储设备和 System ACE 控制器芯片[21]。System ACE CF 存储设备包括 Xilinx 的 ACE Flash 卡或者其他厂家的 Compact Flash 卡。

System ACE CF 控制器芯片提供了存储单元和 FPGA 器件之间的接口，PC 和存储器的标准 JTAG 接口。控制器芯片默认的配置模式是通过边界扫描的方式将数据配置到 FPGA 中，其主要特点有[21]：

- 支持 Xilinx 所有 FPGA 系列芯片的配置
- 以最小的 PC 板空间实现多达 8Gb 的配置数据
- 可实现高达 152Mbps 的配置速率
- 利用嵌入微处理器的 FPGA 进行系统调节
- 可管理多个比特流，并根据需要对其进行激活
- 包含处理器核初始化
- 包含内置式微处理器接口，可以直接调整 FPGA 配置，释放设计资源
- System ACE 技术能简化大型 FPGA 系统的配置方案，令用户将精力主要集中在系统性能的提高核开发时间的缩短。

2.3 小结

本章首先简要的介绍了可重构逻辑器件的基本技术，包括可重构逻辑器件的编程原理以及分类。针对所要实现的加密动态部分重构系统，详细的分析了 Virtex-II Pro XC2VP30 实验开发平台，并对本文使用的 PowerPC405、ICAP 和 System ACE 组件作了重点阐述。

第3章 基于模块化设计的部分重构方法

3.1 部分重构的设计技术

实现动态部分重构存在两种设计方法。一种是基于差分设计(Difference-based design)，另一种就是基于模块设计(Module-based design)[27]。

基于差异的设计技术通常适用于重构前后的部分配置数据文件间差别不大的情况。其原理就是对部分配置数据中的差异部分所对应的可重构逻辑资源进行重新配置，配置完成之后就可以生成只包含差异信息的配置文件，从而大大减少重构的数据规模。显而易见，比起整个配置文件，下载这种配置文件必然节约大量的时间，有效减少配置时间对系统运行效率的影响。

基于差异的设计技术的实现主要是通过工具 Xilinx FPGA Editor 人为的修改 FPGA 内部多种的配置信息来完成。在已经完成布局布线的 FPGA 资源图上，FPGA Editor 对三类资源进行修改：1)设计中使用的 I/O 配置信息；2)BRAM 中的存储内容和写模式；3)查找表(LUT)实现功能的逻辑表达式。

尽管基于差异的设计技术在配置数据规模和下载效率方面有一定的优势，但是这种设计技术存在以下缺点：1)不能修改与布线资源相关的配置信息。这是由于 FPGA Editor 属于开发工具链后端的工具，在对布线资源的配置信息修改后，无法进行相应的时序分析工作，容易导致信号冲突。2)只适合不太复杂的电路设计。这是由于配置内容的修改都是通过人工来完成的，在整个 FPGA 设计流程中来说，是相对比较底层的。当用户需要对电路进行修改时，就需要对相应的逻辑功能非常熟悉。一旦电路比较复杂，人工的修改配置信息就很容易发生配置错误，易导致电路的损坏。3)可供改动的空间小，面临具有较高复杂度的设计时应用难度很大。

基于模块的设计技术提供的是体系结构级的解决方案，适用于复杂系统的设计。

相对差分设计技术而言，模块设计技术具有并行设计和实现各模块，各模块间可最大程度上互不干扰，提高系统开发效率；各个模块有独立的优化目标；改变某些模块功能时，不影响其他模块的设计和实现，保证设计的可靠性和稳定性等优点。

除了能有效提高系统的开发效率外，模块设计运用在动态部分可重构系统的实现中有明显的优势。这是因为硬件设计存在着“与生俱来”的模块化特性，例如硬件语言 HDL 程序本身就是由不同层次的模块(module)组成的，在应用模块化设

计技术进行划分时可以有明确的分界。

动态部分重构系统设计中，第一步是系统的顶层设计。在顶层设计过程中，要有意识的把系统分成多个子模块，同时还要加上适当的约束条件，比如模块的大小，布局位置等。第二步是系统的子模块划分。在子模块划分时，依据系统内部各模块的互连关系和功能结构以及实际的应用需求，确定各个子模块在重构过程中具有的属性是固定还是可重构。对于按照何种方式进行模块划分，也一直是研究的重点。同时，考虑到部分重构系统的易实现性，用户需遵循以下原则：1)尽量将彼此间存在密切通讯关系的划分在同一个模块中 2)尽量减少重构模块的数量(最好为一个)，有助于降低设计难度和简化子模块之间的通信。

3.2 模块化设计流程

经过合理的模块划分后，就可以开始进入硬件的模块化设计阶段，同时通过加载应用程序来完成完整的系统，其详细流程图如图 3.1 所示：

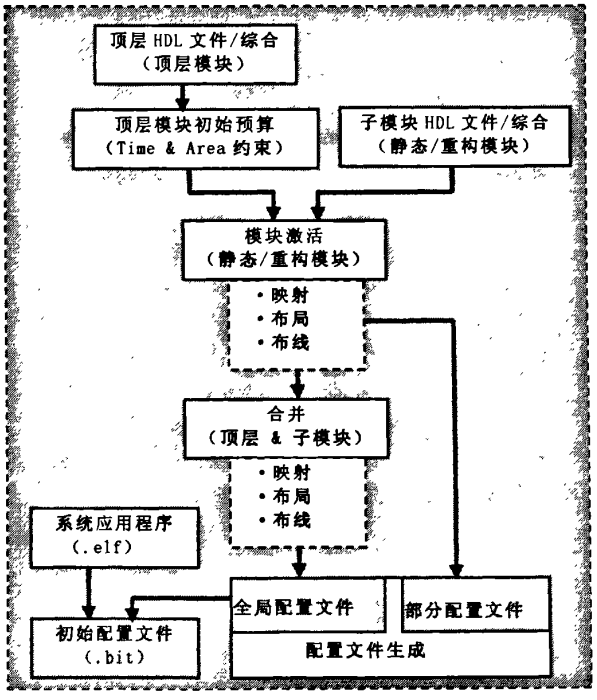


图 3.1 系统实现流程图

图中，模块 HDL 文件以及综合阶段是根据划分的模块，用户分别用 HDL 语言表述各个模块，并使用综合工具进行综合，为模块设计的实现阶段做准备；初始预算对系统进行全局约束，确定用户设定的约束条件是否满足硬件要求；模块激活阶段实现静态模块和重构模块的布线工作；合并阶段将单独实现的静态模块和重构模块合并成一个完整的系统，使系统能最终在 FPGA 中运行；配置文件阶段生成系统所需的各类配置数据文件。

3.2.1 模块 HDL 设计和综合

模块 HDL 设计和综合阶段注重概念设计，而不是具体的功能描述，通常采用硬件描述语言 VHDL 作为顶层模块、固定模块和重构模块的设计入口，并可利用 Xilinx ISE、Synplify Pro、LeonardoSpectrum 等综合工具进行综合，生成各模块用于布局布线的网表文件和相应的约束。综合效果直接影响设计的性能和逻辑门的利用效率。

此阶段必须要在模块激活阶段之前完成。与综合有关的详细文档，用户可以参考文档[28]。

图 3.2 所示为模块 HDL 设计和综合的实现流程[28]：

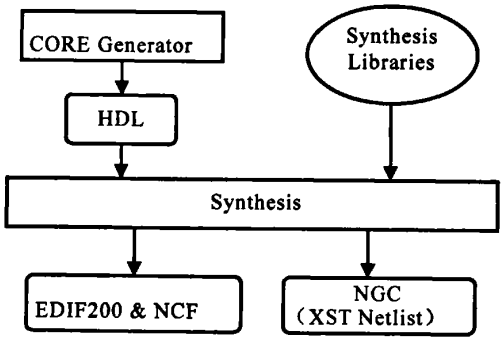


图 3.2 模块 HDL 实现和综合流程

在各模块的综合过程中，用户需遵循以下要求：

- 设计中的各模块必须只包含唯一的网表文件。为了减少设计中各模块的相关性，用户应尽量为各模块建立单独的实现目录，避免各模块边界逻辑的优化。
- 固定模块和重构模块的综合属性有别于顶层模块，这是由于固定模块和重构模块的输入/输出并不是整个设计的外部接口，所以在综合过程中应该禁止固定模块和重构模块插入 I/0 端口属性选项，而仅在综合顶层模块时才插入 I/O 端口。
- 顶层模块 HDL 实现中，子模块的引用名称必须与对应模块的实现文件名保持一致。如果名字不符，则在后续阶段 NGDBuild 就不能将子模块的网表文件嵌入到顶层模块网表文件中。
- 同样，顶层模块 HDL 中，底层子模块一般需以黑盒的方式存在，以避免初始预算阶段产生错误。

3.2.2 初始预算

初始阶段的主要完成对设计进行全局区域布局、约束每个子模块的规模和区域、定位每个模块的输入/输出和对设计进行全局时序约束。初始预算过程中，用户必须要为设计的整体进行位置布局，只有布局合理，才能够在最大程度上体现

模块设计的优势；反之，如果布局不合理则会导致在后面各阶段实现失败，需用户重新返回到初始预算阶段重新开始，从而大大降低了系统开发效率。

初始预算阶段主要完成四种任务：布局设计的全局逻辑、对每个模块的大小和在目标板上的位置进行详尽的约束、对每个模块的输入输出端口进行约束以及进行全局时序约束。

在利用Xilinx开发工具进行模块设计时，该阶段使用的工具是NGDBuild，它利用顶层模块综合生成的网表文件作为输入数据，产生Xilinx的硬件原语网表文件（.NGD）。该文件中所有子模块的实例（Instance）被表述为未扩展的模块，虽然NGD文件不能被映射到芯片上，但是可以使用约束编辑器（Constraints Editor）、引脚与区域约束编辑器（PACE）、Xilinx布局规划器（Floorplanner）等工具打开。使用Constraints Editor 可以约束模块的时序，使用PACE 可以锁定引脚位置，使用Floorplanner 可以对每个子模块进行区域约束。

所有全局逻辑的区域约束在整个设计中至关重要，因此用户必须要对顶层逻辑进行合理的约束，区域约束适合于以下四种逻辑：顶层I/O端口约束、顶层逻辑约束（此项约束主要是对全局时钟资源、三态缓冲器、触发器和查找表等资源进行约束）、每个模块的区域约束（规定每个模块在FPGA上所占用的位置和大小）、伪逻辑(Pseudo logic)区域约束（伪逻辑以“黑盒”形式存在，在系统实际实现时，这些伪逻辑就被实际的连接取代了）。

初始预算阶段的流程如图 3.3 所示[28]：

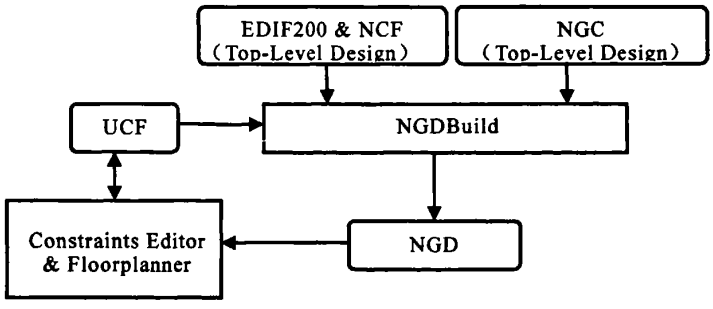


图 3. 3 初始预算阶段流程图

3.2.3 模块激活

本阶段，利用子模块的网表文件与初始预算阶段产生的顶层模块的实现文件NGD，将顶层模块中的子模块进行扩展性实现。也就是说，初始预算阶段中作为“黑盒”处理的子模块，将被具体的IP核功能单元嵌入到顶层模块中。

子模块的激活实现通过独立的的实现目录来单独进行，而不能直接在顶层模块的实现目录中进行。激活阶段的具体实现方式为：激活模式下，NGDBuild工具以顶层模块实现文件NGD、约束文件UCF和当前子模块的网表文件作为输入，生

成嵌入子模块逻辑的NGD文件和布线文件NCD。该NGD文件中仅当前子模块被“激活”，也就是说仅有当前子模块被填充入具体的功能实体，而其他子模块仍然以“黑盒”形式存在。

完成布局布线后，为了便于合并阶段的实现，用户可以利用pimcreate命令是将各个子模块对应的物理实现文件集中拷贝到指定文件目录/pims中。

模块激活阶段的流程如图3.4所示[28]：

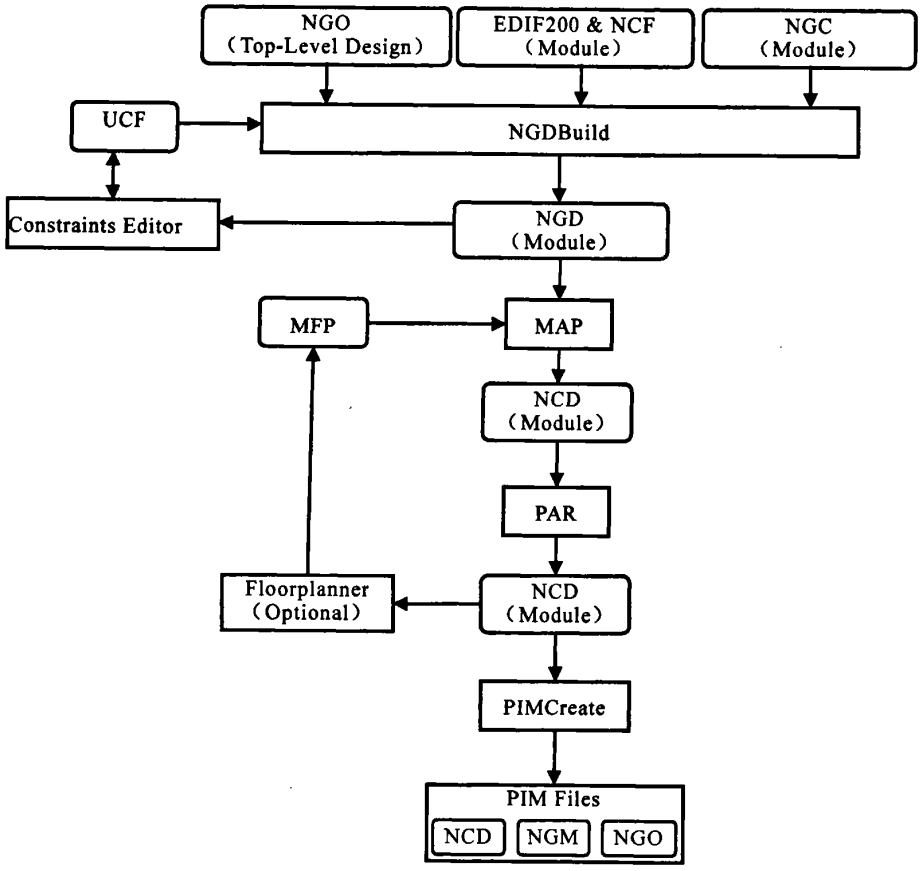


图 3.4 激活阶段流程图

3.2.4 合并阶段

本阶段分别将组成系统的子模块有机地结合起来，完成整个设计的实现。通过实验最终的布线图易知，布线工具复制每个子模块所对应的布线资源，通过硬宏总线宏来连接，每个子模块内部电路逻辑并没有改变。

NGDBuild工具读取顶层设计的NGO文件、约束文件和组成系统的子模块所对应的物理实现，以便完成整个系统的布局布线。同时，在最后的合并阶段，系统会自动优化掉子模块之间没有连接的信号，提高系统的整体性能。其具体的流程如图3.5所示[28]：

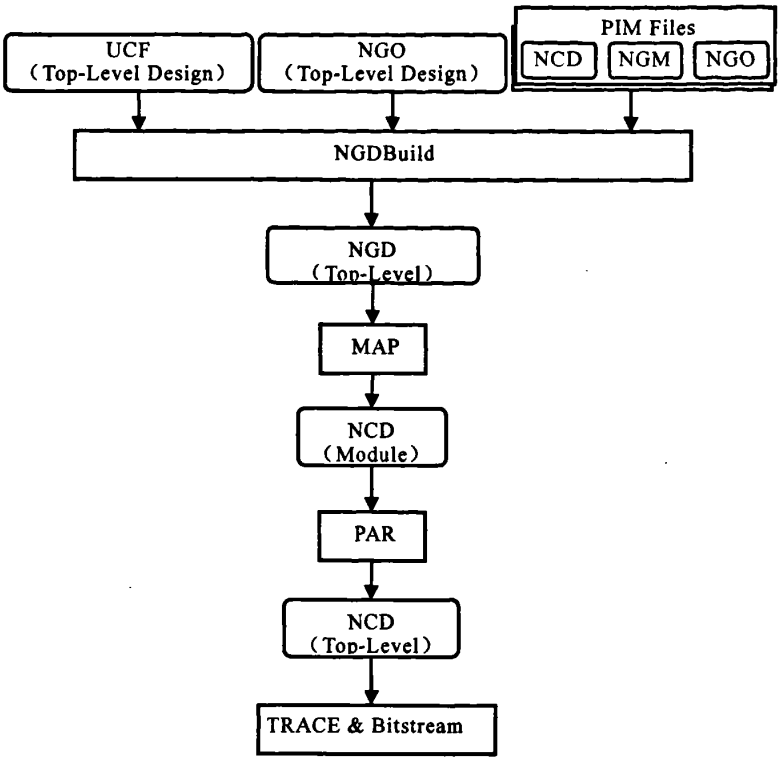


图 3.5 合并阶段流程

3.2.5 配置数据的生成

系统完成整体布局布线后，还需将布线文件转化为流格式的文件才能使 FPGA 能够运行。Xilinx 流文件的生成是通过 BitGen 工具来实现。BitGen 读取布线文件.ncd，生成后缀为.bit 的二进制配置流文件。配置流文件包含了所选 FPGA 芯片和 FPGA 内部所有逻辑以及逻辑连接关系的配置信息，其流程如图 3.6 所示[23]：

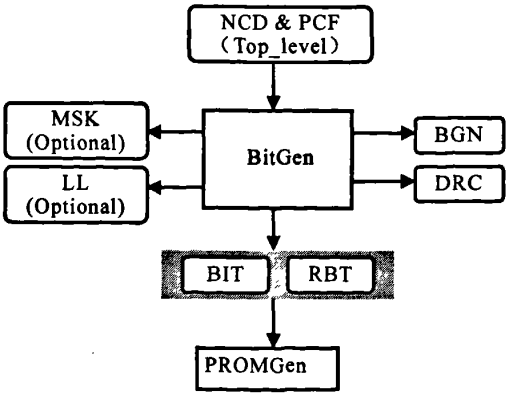


图 3.6 配置流文件流程

动态局部重构系统中，第一次下载到 FPGA 中运行的必然是一个系统的完整配置信息，只是在后期的运行过程中重构模块可被反复替换，以实现系统的不同工作。

3.2.6 配置数据的转化及下载

本文中的动态局部重构系统是通过使用内部访问接口 ICAP 来完成配置的。ICAP 接口在配置管脚设置 M2:M1:M0 为 101 的情况下是无效的[29]，常用的 JTAG 下载方式的配置管脚设置 M2：M1：M0 正好为 101，为此，实验中不可将初始配置文件直接通过 JTAG 下载到开发板中。

本文实验中所使用的 Virtex－II PRO 开发板仅支持 JTAG 和 PROM 两种用户配置模式。为了实现动态局部重构系统，必须先利用 Xilinx 的工具软件 iMPACT 将 BIT 文件转化为 MCS 文件，然后通过 JTAG 下载存储到开发板上的 EPROM 内，在系统加电的时候，系统自动将配置文件下载到 FPGA 中并开始运行。重构时，直接调用 Flash 中的局部配置文件即可实现重构操作。

PROM 配置模式的流程如图 3.7 所示[23]：

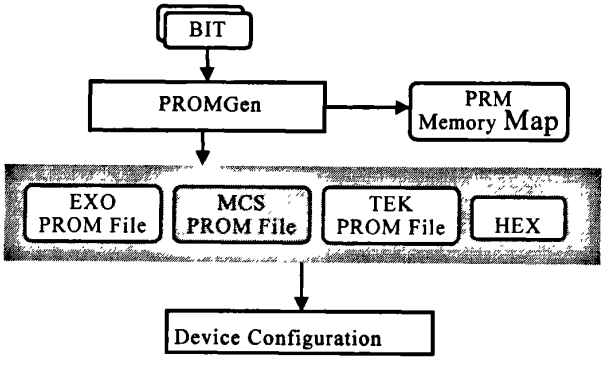


图 3. 7 PROM 配置模式流程

3.3 小结

本章首先简要介绍了现有的部分重构设计方法并分析它们的优缺点。借鉴模块化设计技术，重点介绍了部分重构设计的基础——Xilinx 模块设计方法，分析了部分重构模块设计的相关问题和流程。

第4章 过程级动态部分重构系统设计与实现

4.1 过程级动态部分重构系统

本文实现的过程级动态部分重构系统是利用可重构逻辑器件内嵌的控制单元对系统的重构过程进行控制，在系统运行时生成或修改器件的配置数据文件，以达到系统功能切换的目的。与传统的动态部分重构系统相比，具有以下的特点：1)内嵌的控制单元不受部分重构过程的影响，增强了系统的可靠性。2)通过器件内的配置数据访问端口从可重构逻辑器件内部访问器件的配置数据，缩短了重构过程的时间延迟，减少了系统为板级重构准备的相关外设。3)可重构逻辑器件内嵌的处理器是系统重构配置的控制单元，对不同的系统部件进行管理，可提高计算功效。

在过程级动态部分重构系统的设计与实现过程中，以加密算法为例开展了研究，这是因为加密算法使用了大量适合于采用硬件实现的位运算。过程级动态部分重构系统能够在执行过程中将加解密功能依据应用需求做及时的切换，目的是为了避免加解密功能在可重构逻辑器件上共存导致的资源占用量增加以及尽量减少重构过程的高昂时间代价。

过程级动态部分重构系统的设计需要解决以下几个关键问题：

1）支持运行时重构的设计方法。虽然Xilinx EDK开发流程为基于可重构逻辑器件的嵌入式系统设计提供了很好的支持，但是在系统中，现有开发流程不能够保证通讯通道不受重构过程的影响，因此需要对其进行改进。

2）模块划分。基于模块的部分重构技术具有较多的设计约束，在借鉴Xilinx模块化设计方法学的基础上，改进了基于模块的部分重构设计方法。系统中，重构模块被重构时，静态模块仍旧处于工作状态，这常常会引起模块间处理数据的不同步而导致算法执行错误。因此，用户需合理的对系统进行模块划分，同时对各模块的特殊约束进行绑定。

3）通信通道设计。重构模块和其它模块间的通讯通道必须采用总线宏构建。

4.2 过程级动态部分重构系统的关键技术研究

4.2.1 系统开发流程

和所有的基于可重构逻辑器件的嵌入式系统一样，自重构计算系统的开发流程非常复杂：首先在可重构逻辑器件上集成系统所需部件，然后针对此硬件平台进行系统应用的软件设计，最后将生成的硬件配置文件和软件程序分别加载到可

重构逻辑器件和系统内存中。

Xilinx 公司为可重构系统设计提供了完整的集成开发工具 EDK，它支持硬件和软件协同设计的能力，可完成逻辑综合、布局布线和配置数据文件生成等工作，能有效的缩短了设计周期和减少了设计人员的工作量。硬件设计主要从 Xilinx IP 核库中选取系统所需的 IP 核，包括软硬微处理器、总线、外围设备和各种接口控制器等，对它们进行信号连接以及设置相应的约束。在 IP 核连接完成后，系统会自动生成硬件结构描述文件 MHS(Microprocessor Hardware Specification)，PlatGen 工具根据 MHS 文件信息产生系统的硬件工程文件，并对 IP 核进行综合以产生硬件配置文件，下载到 FPGA 中运行。在硬件设计的同时，软件设计根据系统中所选择的 IP 核自动添加对应的硬件设备驱动程序和操作系统产生系统软件描述文件 MSS(Microprocessor Software Specification)，LibGen 工具根据 MSS 文件信息生成系统软件库以及操作系统的 BSP 文件。用户就是根据 LibGen 提供的库来编写系统的应用程序，编译生成可执行文件下载到内存中运行。

针对过程级动态部分重构系统在加密中的应用，Xilinx EDK 利用 BaseSystem Builder Wizard 来快速构建系统[30]。首先设定参考时钟频率、处理器时钟和总线时钟，然后选取系统所需的外设，串行通信模块 RS232、通用输入输出模块(GPIO)、BRAM 等，其中 plb_bram_if_cntrl 外设是必选的，该控制器完成对 BlockRAM 的控制，保持 CPU 处于一个确定的状态。

在 Xilinx EDK 为系统添加必要的 IP 核后，根据加密的实际需求，使加密算法以协处理器的方式通过总线技术与处理器互连，其接口设计必须满足相关总线协议的要求。系统设计完成硬件 IP 核选型与互连后，Xilinx EDK 将为系统中的各个外设划定相应的地址空间，以便处理器对相关外设进行驱动与访问，地址具体分配如表 4.1 所示：

表 4.1 系统地址分配

IP 核	总线类型	地址空间	地址大小(字节)
plb2opb	PLB	0x00000000~0x7fffffff	2G
plb_bram_if_cntlr_1	PLB	0xfffe0000~0xffffffff	128K
opb_hwicap_0	OPB	0x40200000~0x40200fff	4K
SysACE_CompactFlash	OPB	0x41800000~0x4180007f	128
RS232_Uart_1	OPB	0x40600000~0x406000ff	256
leds_4Bit	OPB	0x40000000~0x400001ff	512
opb_des	OPB	0x7A200000~0x7A2000ff	256
opb_aes	OPB	0x60000000~0x600000ff	256

需注意，除了 plb_bram_if_cntlr_1 外，其他 IP 核的地址空间不能超过 plb2opb 所确定的系统地址空间范围。

同时，由于系统中用到了 System ACE 配置方案，用户还需选择 Xilinx 公司提供的系统软件模块 xilfatfs[31]，它主要对外存 Compact Flash 的读写进行控制。如图 4.1 所示：

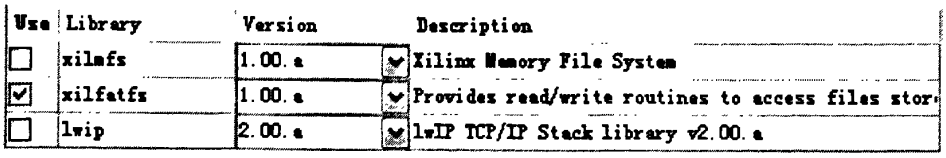


图 4. 1 EDK 系统软件模块 xilfatfs

通过 EDK 的软硬件设计，最终将生成了本文所设计的一个示例动态可重构计算系统的结构图，如图 4.2 所示，该系统是 IP 核的有效集成。

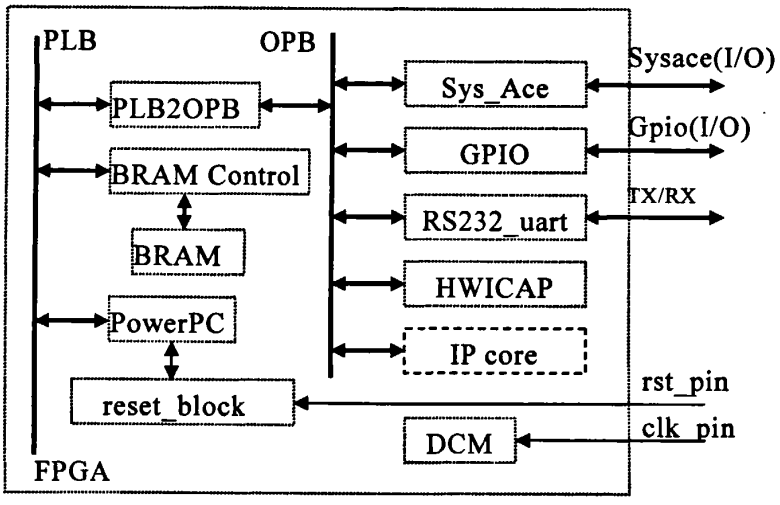


图 4. 2 EDK 生成的具体结构图

在EDK生成的结构图中，对应图1.1的系统结构图，其中PowerPC为系统图中所示微处理器单元，存储器单元则包括BRAM以及BRAM控制器；SYS_ACE用于管理重构过程及重构模块的配置数据；GPIO属于系统图的可重构管理模块，负责组件的控制和管理；最后我们所说的专用硬件处理模块在示例中就是IP核，它可被动态改变。

4.2.2 模块划分

Xilinx EDK的嵌入式设计开发流程虽然能够在系统设计阶段通过重新配置器件上的可重构逻辑资源来构建系统的硬件平台，但是它不能够对系统的运行时重构设计提供足够的支持。因此，在过程级动态部分重构系统的设计开发中，需要在Xilinx EDK开发流程的基础上进行设计方法的改进。

过程级动态部分重构系统在加密算法的应用中，所有的系统硬件部件都直接配置

在可重构逻辑器件内。根据在系统中承担的不同功能，这些部件被划分为静态和重构两部分。其中，静态部分包括处理器以及其它一些相关外设，主要负责系统管理以及输入输出等工作，这部分部件在系统运行过程中始终保留在可重构逻辑器件上不做任何改变；重构部分是加密算法核，在运行时根据系统需求的变化在固定部分的控制下做功能切换。

实验中，为了整个系统的健壮性，需为整个系统提供顶层模块。顶层模块就包含重构模块和静态模块两个子模块。

顶层模块的作用主要是连接各个子模块，子模块才是功能实体。顶层模块中包含数字时钟管理器 DCM、全局时钟多路缓冲器(BUFG)、IOB(输入/输出)逻辑、重构模块实例、静态模块实例、总线宏的实例以及临时信号的声明。顶层模块的实现是后续设计阶段的基础，直接关系着整个系统设计的成功与否。

（1）DCM 和 BUFG 属于全局逻辑，为整个系统提供统一的时钟逻辑，独立于各个子模块，因此需单独置于顶层模块中。

DCM 需要用户自行在顶层模块中实例化，实例化的参数要与 Xilinx EDK 中设定的数据一致。而不能直接使用 Xilinx EDK 工具自动添加的 DCM 模块所生成的网表文件(Netlist)，这是因为 Xilinx EDK 工具会默认为 DCM 模块添加了相应的 BUFG，如果直接使用网表文件，将与用户自行添加的 BUFG 产生重复；如果用户不自行添加 BUFG，则用户对系统默认添加的 BUFG 的位置无法进行约束。

DCM 一般和 BUFG 配合使用，使用方法是将其输出 clk_1x 接在 BUFG 的输入引脚上，BUFG 的输出引脚反馈回来接在 DCM 的反馈时钟脚 CLKFB 的同时，也使时钟信号分配到全局时钟网络，这样获得的时间的延迟将是最小的，驱动也最强。Virtex-II Pro FPGA 全局时钟多路缓冲器如图 4.3 所示[23]：

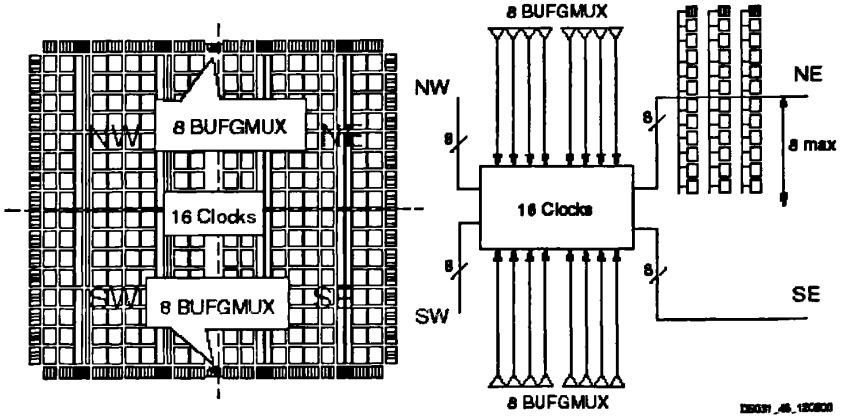


图 4. 3 Virtex-II Pro FPGA 全局时钟多路缓冲器分布

（2）IOB 逻辑应放置寄存器，确保逻辑组内的所有关键路径当进行逻辑优化时，不会在跨越模块边界的地方产生问题。

（3）静态模块、重构模块和总线宏的实例以“黑盒”的方式存在，即在模块声

明中只定义模块的端口和端口方向、模块间互连的信号以及模块与 I/O 端口间互连的信号。

对于图 4.2 所示系统，对应的顶层模块设计如图 4.4 所示：

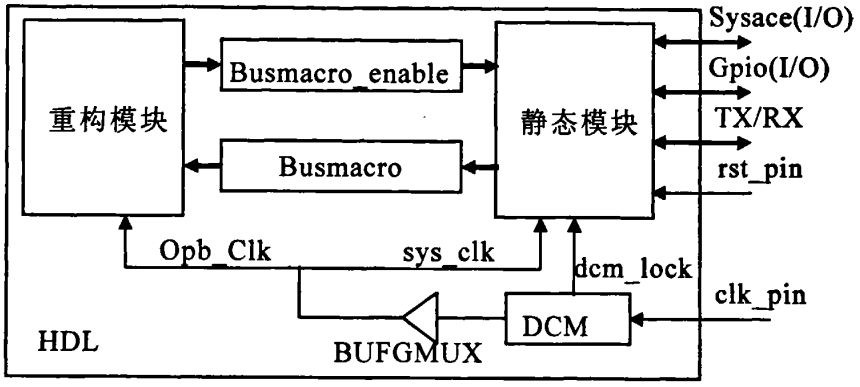


图 4.4 系统模块设计布局

图 4.4 显示了系统划分为静态模块和重构模块的结构图。静态模块和重构模块之间的信号，除了时钟信号，必须通过总线宏。这是因为重构模块在配置过程中，FPGA 重构区域片上逻辑发生改变，模块边界信号不再对齐，无法完成信号的连接。为了保证重构前后信号连接的完整性，必须保证连接重构模块端口信号的路径固定不变[32]。Xilinx 在 Virtex II Pro 系列 FPGA 上专门提供了总线宏，以实现连接路径固定，但连接功能可配置的模块间互连。模块间不同方向的信号传输需要不同结构的总线宏，尤其重构模块向静态模块传输信号的总线宏，需要控制总线连接和断开的使能端口，我们称这种形式的总线宏为使能总线宏。这是由于当重构模块不存在或者正在配置过程中，重构模块所占区域会发射出状态不稳定的信号，这些信号进入静态模块中，易导致局部重构失败。因此，利用带使能信号的总线宏在重构前断开重构模块与静态模块重构通信，重构后再恢复通信。

4.2.2.1 重构模块

重构模块是指在 FPGA 设计中被重新配置的部分，且重新配置的过程中剩余的部分仍保持持续的工作状态。重构模块是动态部分可重构系统设计中的重点和难点，主要涉及到重构模块的端口定义、重构模块资源的利用以及重构模块可置换的特性。

（1）重构模块端口定义

本文是对 OPB 总线上的 IP 核进行重构，端口信号遵循 OPB 总线的标准，因此，重构模块与静态模块之间的信号就是对应 IP 核的 OPB 总线端口信号。如图 4.5 所示：

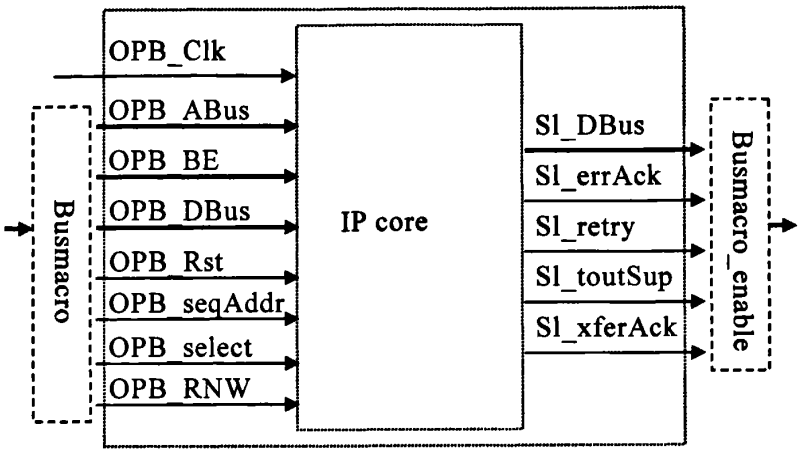


图 4.5 重构模块的结构

图 4.5 易见，除了时钟信号，重构模块和静态模块间的信号通信都应通过总线宏实现。

（2）重构模块资源分配

根据重构模块的特点，重构区域的硬件资源是所有的重构模块共享的，因此如何有效分配FPGA资源满足重构模块需求是用户需解决的问题。

本文实验中所使用的 Virtex-II Pro XC2VP30 FPGA 的物理结构决定了对重构模块资源分配必须按照如下规则[27]：

- 重构模块的宽度必须是 4 slice 的倍数，至少要占 4 slice 单位资源，最大可以达到整个设备的宽度。
- 重构模块的放置必须以 4 slice 为边界，从最左边开始可以放置的位置为 x=0,4,8, ...slice。
- 重构模块宽度范围内所有的逻辑资源都属于重构模块，包括 TBUF、slices、BlockRAMs 、Multipliers、IOBs 和所有的布线资源。
- 时钟逻辑(BUFGMUX，DCM)总是和重构模块分开的，因为时钟有单独的比特流帧。
- 如果重构模块的高度是整个设备的高度，则片上顶部和低部 IOB 资源属于重构模块。
- 重构模块如果占居 FPGA 片内左右两端资源，则最左边或者最右边的所有 IOB 资源也属于重构模块。
- 重构模块边界不能改变，所占区域和位置固定不变。同时，重构模块和其他模块通过总线宏来实现通信，总线宏 在模块配置信息改变的时候保持不变。
- 为了降低动态部分重构设计的复杂性，应尽量控制重构模块的数目。理想情况下，系统中只有一个重构模块。重构模块的最大数目是整个设备

宽度的 slice 数除 4。

- 重构过程中，固定模块不能依赖可重构模块的状态，同时为保证重构配置操作的正确性，模块通信中显式的握手逻辑是必须的。

重构模块在 FPGA 片上的具体分配如图 4.6 所示：

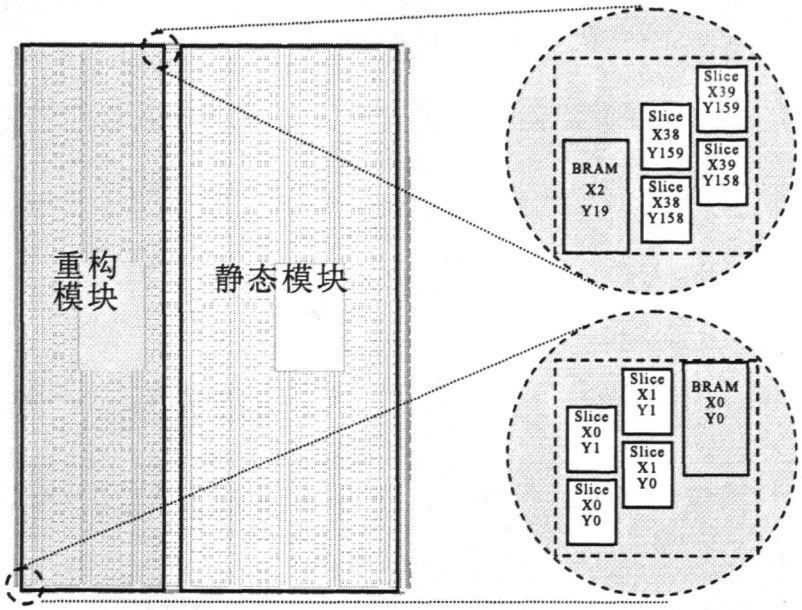


图 4.6 重构模块所占资源分配示意图

（3）重构模块的可置换性

在重构过程中，由于所有的重构模块都可共享重构资源并可与静态模块进行通信，重构模块与静态模块之间的通讯端口必须保证不会发生改变，所以同一可重构模块的各个版本的命名与端口定义必须保持一致，见图 4.7：

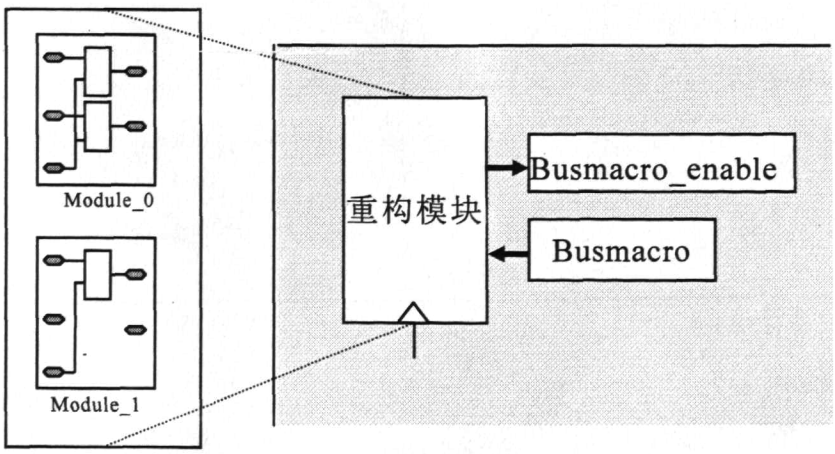


图 4.7 重构模块端口一致性示意图

4.2.2.2 静态模块

静态模块是系统设计中的固定逻辑模块。在重构过程中，模块功能不改变，并全程负责控制重构操作的完成。本文的静态模块结构如图 4.8 所示：

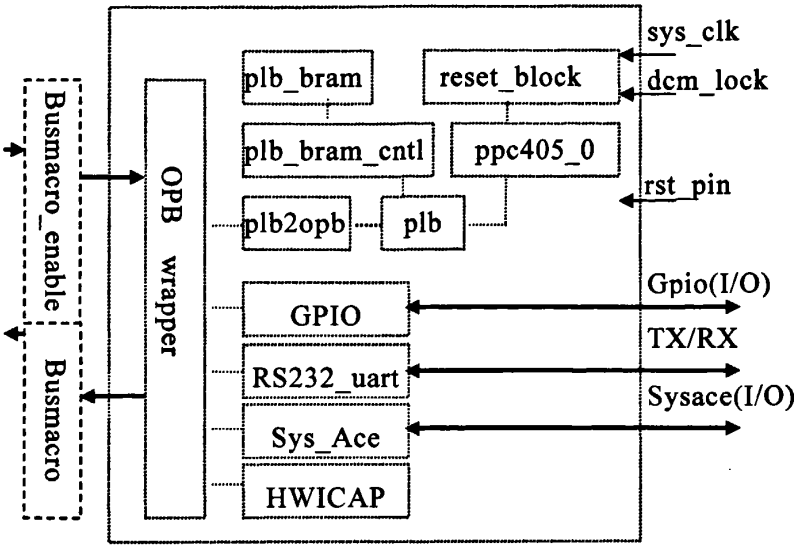


图 4.8 静态模块的结构

图 4.8 中，系统在 PowerPC 微处理器(ppc405_0)的控制下，利用 ICAP(HWICAP)向重构单元写入配置文件来完成重构操作。

静态模块为了实现与重构模块的通信，需在静态模块中添加与重构模块对应的端口信息，以保证模块间的信号完整性，端口信息见表 4.2：

表 4.2 静态模块中与重构模块对应的端口信息

输出信号	输入信号
OPB_Rst	Sl_DBus[0..31]
OPB_ABus[0..31]	Sl_errAck
OPB_BE[0..31]	Sl_retry
OPB_DBus[0..31]	Sl_toutSup
OPB_RNW	Sl_xferAck
OPB_select	
OPB_seqAddr	

其中，OPB_开头的是静态模块向重构模块发送的信号，Sl_是重构模块返回给静态模块的信号。

由于全局时钟资源(DCM 和 BUFG)被放置在顶层模块中，因此，静态模块还需为模块内部的各 IP 核提供统一的时钟端口信号。

```
sys_clk  : in std_logic;
dcm_lock : in std_logic;
```

其中，dcm_lock 是 reset_block 的驱动信号。

4.2.2.3 系统特殊约束

FPGA 的设计过程中，约束文件是进行资源分配的依据。通过附加约束可以有效控制逻辑的综合、映射、布局和布线，减少逻辑和布线延时，提高系统工作频率；同时也可以根据 FPGA 设计平台上静态时序分析工具获得映射或布局布线后的时序分析报告，从而对设计的性能做出评估。

在动态部分可重构系统的开发过程中，约束文件主要应用在模块设计中的初始预算阶段。其中包括时序约束的设定、I/O 资源的管脚绑定、各个子模块以及总线宏的布局限制。如果在后续的模块设计实现中发现硬件资源不足或者发生冲突，则说明初始预算阶段资源分配有误，需要对用户约束文件进行修改。

对于约束文件中时序约束和 I/O 资源的管脚的设定，与在 Xilinx EDK 创建的系统有直接联系。附录 B 显示了在时钟频率为 100HZ 的情况下，包含 GPIO、Sysace 和相应的约束。图 4.9 所示为部分端口在片内的具体布局。

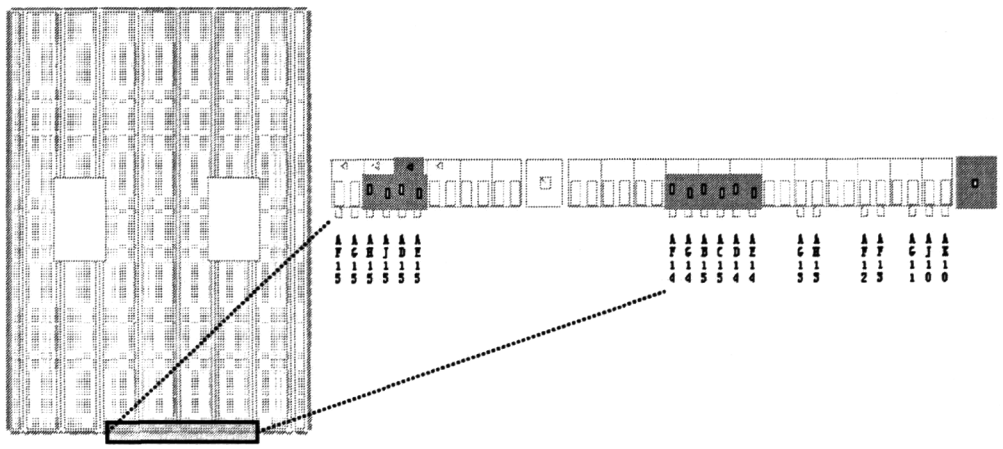


图 4.9 片内部分 I/O 管脚的绑定端口

尽管 Xilinx EDK 在系统创建过程中，通过基本的时序约束已经为系统内部各外设器件自动分配了管脚。但考虑到在动态局部重构系统模块划分阶段，对系统所占资源又重新进行了分配。因此，除了一些专用管脚外[33]，用户还需通过工具 FPGA floorplanner 以及 Xilinx PACE 来适当的修改外设器件的部分管脚，以便更加有效的利用 FPGA 片内资源。

用户通过 FPGA floorplanner 以及 Xilinx PACE 分配管脚需考虑以下的几个方面：

（1）FPGA 所承载逻辑的信号流向

Virtex-II Pro XC2VP30 FPGA 外部的管脚数量虽然很丰富，但如果管脚的分配不合理，那么有可能在系统布线时出现大量的交叉的信号线，给布线的实现带来很大的困难，甚至走不通；或者是即便顺利完成布线，也有可能由于外部的延时过大而不满足时序方面的要求，所以在管脚分配前需对 FPGA 内部有一定的了

解。Virtex-II Pro XC2VP30 FPGA 内部分成 8 个区域，每个区域中可用的 I/O 管脚数量各不相同，见图 4.10。

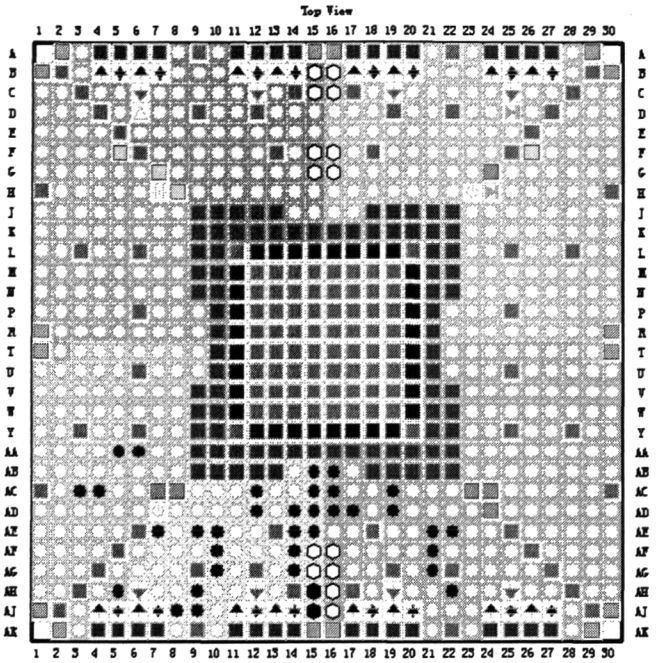


图 4. 10 Virtex-II Pro XC2VP30 FPGA 内部 Bank 分布图

图中易见 Virtex-II Pro XC2VP30 FPGA 内部 BANK 的分配情况和每个 BANK 中所支持的 I/O 标准。实验中，根据 FPGA 中内部 BANK 的分配的情况，同时结合系统中外部器件的信号的流向，就可以按照连线最短的原则将对应的信号分配到与外部器件连线最近的 BANK 中，完成信号的分配。

（2）掌握所选 FPGA 每个 BANK 所支持的 I/O 标准

从图 4.10 中可以看出 FPGA 内部的每个 BANK 所支持的 I/O 的标准不尽相同，因此，在分配管脚时要将支持相同标准的管脚都集中到一个 BANK 中，这是因为 FPGA 中同一个 BANK 一般不同时支持两种 I/O 标准，当然也有例外，这就需要用户查阅相关 I/O 标准所要求的工作条件。

除了上述的在 Xilinx EDK 阶段自动确定的约束外，动态局部重构的设计过程中，用户还必须手动为系统中的子模块定义相应的区域约束、时钟资源 DCM 及 BUFG 的绑定约束和总线宏的 LOC 约束。

1）系统中子模块的区域约束是在 FPGA 上规划各个模块的实现区域，通过物理布局布线约束，完成模块化设计。本文使用的约束包括 AREA GROUP，AREA GROUP RANGE，MODE 约束[34]。

AREA GROUP，AREA GROUP RANGE 和 MODE 3 种约束，重构模块和静态模块都必须具有。AREA GROUP 是为了避免重构模块和静态模块之间产生共有逻辑。AREA GROUP RANGE 则是定义重构模块和静态模块的硬件资源区域，包括 slice 和 RAM 块两种。

其语法格式为：

AREA_GROUP "AG_PRregionA" RANGE =SLICE_(minX)(minY):

SLICE_(maxX)(maxY);

对于实验平台 XC2VP30，minX 必须满足 4*n(n=0,1...39) slice，(maxX-minX+1) 需满足 4*n(n=1,2...40)。

AREA_GROUP "AG_PRregionA" RANGE =RAMB16_(minX)(minY):

RAMB16(maxX)(maxY);

参数(minx,minY)，(maxX,maxY)的取值取决与 slice 参数的取值，必须将 slice 范围内所有的 RAM 块资源都包含在内。

对于定义了 AREA GROUP，AREA GROUP RANGE 的模块，必须同时定义 MODE 约束，特别需要将各个模块占用的资源区域标识为“RECONFIG”属性。

其语法格式为：

AREA_GROUP "AG_PRregionA" MODE=RECONFIG;

2）在顶层模块 HDL 实现中，对于用户自行添加的时钟资源 DCM 实例以及 BUFG，需设置特定的 LOC 约束。

在分配时钟资源的过程中，时钟资源的多少导致分配的策略差别很大，这就需要查阅相应的 FPGA 文档看哪些时钟分别能到达哪些区域。通常情况下，一般的时钟都是差分时钟，如果所用的不是差分时钟就需要注意 P 端与 N 端一般不能同时分配给不同的时钟信号。

Xilinx Virtex-II Pro 系列的 FPGA 中，成对的时钟如果是同时采用那么就不能同时到达相同的区域，因为到达相同区域的时钟线只有一根，结构如图 4.11 示[20]：

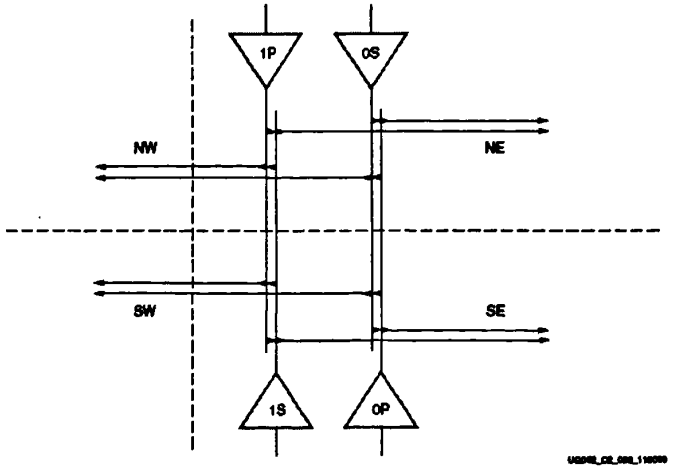


图 4. 11 BUFG#P and BUFG#S 连接结构

因此在时钟较少时最好不要同时使用成对的 P 和 S，而只选择 P 或者是 N，这样就能有效避免出现冲突的情况。

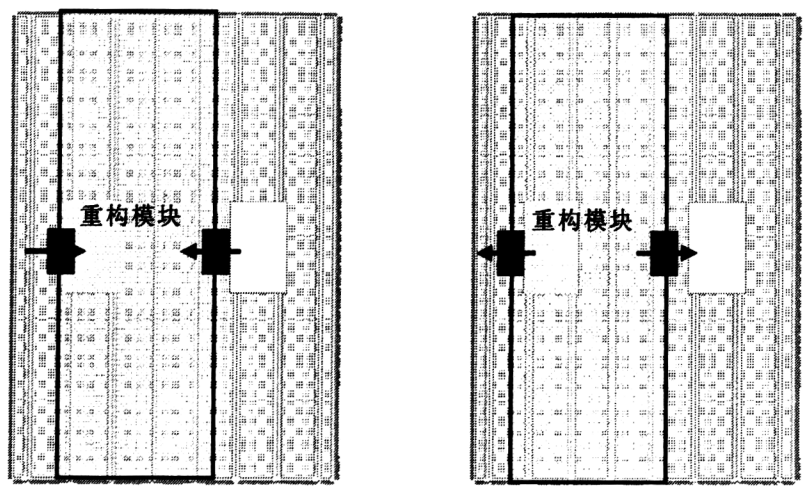
本文实验中所使用的 Virtex-II Pro XC2VP30 FPGA，片内包含 8 个 DCM 时钟

资源，上下边界分别 4 个；BUFG 资源处在上下边界的中间部分。

DCM 时钟资源和 BUFG 的 LOC 约束语法为：

```
INST "dcm_0" LOC = "DCM_X2Y0" ;
INST "bufg_1" LOC = "BUFGMUX1S"  ;
INST "ibufg_0" LOC = "BUFGMUX6P"  ;
```

3）总线宏的 LOC 约束至关重要，不仅需设定适当的位置以连接重构模块和静态模块满足通信需求，而且要保证总线宏的对齐以实现动态重构。另外，Virtex-II Pro 系列 FPGA 片内资源的分布决定了总线宏的位置只可放置在重构模块的左右边，不可放在上面和下面，见图 4.12。



a) 进入重构模块信号的总线宏　　b) 离开重构模块信号的总线宏

图 4. 12 总线宏的放置

总线宏的放置不仅决定系统重构的成功，对于系统的时间性能也有一定的影响。

对于实验平台 XC2VP30，基于 Slice 的总线宏 LOC 约束的语法为：

```
INST "bus_macro" LOC = "SLICE_X aYb";
```

在总线宏满足同时连接重构模块和静态模块条件下，a 和 b 需为偶数，且小于重构区域的范围。

4.2.3 模块通信

模块设计中，模块间的通信至关重要。系统中，不同模块间的通信方式不一样。静态模块间的通信可以通过简单的直接路由来实现。但对于重构模块与静态模块或者重构模块间的通信则需要固定的布线资源，原因在前文 4.1.2 节中已经解释，即总线宏。

总线宏实际上是一个预先布线好的用来确定精确布线轨道的硬件宏，在重构模块的配置信息改变的时候保持不变。对于每一个不同的设计来说，它是绝对固

定不变的。

基于 FPGA 的动态部分重构系统开发中，使用的总线宏主要有基于三态缓冲的总线宏和基于 Slice 的总线宏。

4.2.3.1 基于 TBUF 的总线宏

基于 TBUF 的总线宏主要使用 FPGA 内部的 TBUF 和 TBUFLINE 资源来实现。在 Virtex-II Pro XC2VP30 FPGA 中，每隔两行 slice 资源有专门连接 TBUF 的四条 TBUFLINE 的路由资源。每条 TUBFLINE 水平排列，连接若干个 TBUF，且不用作一般的路由资源，因此可以作为模块间通信的固定通路，其结构图如图 4.13 所示：

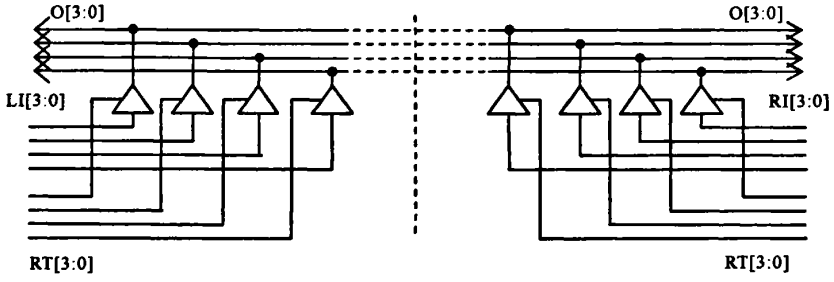


图 4. 13 基于 TBUF 的总线宏结构

图 4.13 所示的是基于 TBUF 的 4-bits 总线宏。其中，LT[3:0]和 LI[3:0]是其左端的方向控制端口和数据输入端口，RT[3:0]和 RI[3:0]是其右端的方向控制端口和数据输入端口，O[3:0] 则是数据输出端口。信号的传递方向只可以从左向右或者从右向左，由总线宏的左右控制端 LT[3:0]和 RT[3:0]来控制。控制信号高低由对应的 TBUF 的三态使能端口来决定。使用总线宏进行通信时，LT[3:0]和 RT[3:0]的值必须相反，或者同时无效。例如，在控制信号高有效条件下，当总线宏从左向右传递数据时，LT 为 1，RT 为 0，反之亦然；如果总线宏不传递数据，则两个控制信号应同时为 0，总线宏输出高阻态。

基于 TBUF 的总线宏没有固定的格式。根据实际需要，可分别实现 1-bit、2-bits 和 3-bits 的总线宏，并且可对总线宏的长度进行拉伸，以便总线宏能够适应远近不同模块间的通信。有时为了简化系统的设计，也可以将多个总线宏合并成一个文件，而不是简单的形成一个宏，因此，约束总线宏的位置时，仍然必须单独约束每个宏。

对于基于 TBUF 的总线宏，其 LOC 约束语法为：

```
INST "bus_macro" LOC = "TBUF_XaYb" ;
```

总线宏满足跨越重构模块和静态模块条件下，a 和 b 需为偶数，且小于重构区域的范围。

基于 TBUF 的总线宏具有结构简单、实现方便等优点，Virtex-II Pro 系列 FPGA 传统的动态部分重构系统开发中，就是使用基于 TBUF 的总线宏来实现模块通信的[35][36][37][38]。但是，它信号传输量小，同时它只能控制总线宏信号的传输方向，并不能控制信号的断开和连接，因此必须在静态模块中添加特定的控制模块，阻止配置过程中来自重构模块的无序信号，只接受完成配置后来自重构模块的信号。

4.2.3.2 基于 Slice 的总线宏

基于 Slice 的总线宏利用了 FPGA 内部的预留连线来实现模块之间通信的通路。这些预留连线是专门用来进行静态路由的，即使这些连线没有被设计所使用，重构模块也应该避免使用到这些连线资源。

相比基于 TBUF 的总线宏而言，基于 Slice 的总线宏有很多优点[36]：1)制造商新推出的支持动态可重构系列的 FPGA 器件中不再拥有 TBUF 资源，限制了基于 TBUF 的总线宏的使用。2)基于 Slice 的总线宏提供了一个更大的通信接口，在模块间的通信可以传送更多的信号，甚至还可以嵌套基于 Slice 的总线宏，满足模块通信的同时还避免了 FPGA 内部资源的浪费。3)在制造商新推出的 FPGA 中(Xilinx 公司的 Virtex-4\Virtex－5 系列 FPGA)，用户通过适当的设计，基于 Slice 的总线宏的放置不再局限于模块边界处，可以嵌入到重构模块内部。4)最重要的，基于 Slice 的总线宏增加了一个防止信号离开模块的机制，因为在模块重构过程中从重构模块发射出的信号的状态是不稳定的。

本文实验中采用基于 Slice 的总线宏，通过增加专门的使能信号来控制信号的传输。实验结果表明，与基于三态缓冲的总线宏比较，它能够更好的控制重构过程中来自重构区域的状态不稳定信号进入静态模块，有效的保证了重构的顺利完成。

本文中使用了基于 Slice 总线宏的两种形式，这里着重介绍带使能信号的总线宏。在 Virtex－II Pro 系列 FPGA 中，仅支持水平方向的总线宏，即总线宏的方向只能从左至右或者从右至左。同时，由于包含在静态模块中完成配置工作的内部配置访问通道(ICAP)位于 XC2VP30 的右下方[23][29]，静态模块需约束在 FPGA 的右边。因此，重构模块通向静态模块信号的总线宏结构如图 4.14 所示[39][40]：

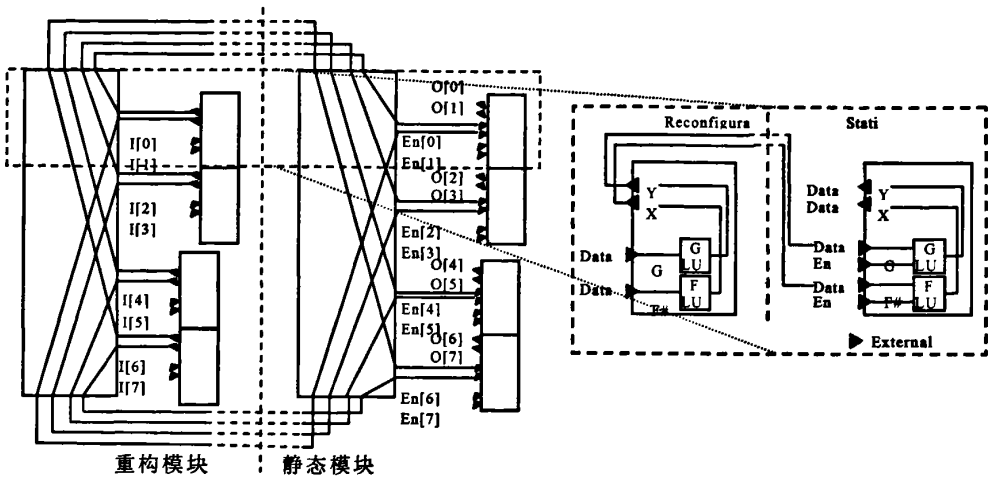


图 4.14 带使能信号的基于 Slice 的总线宏结构

图中所示的是基于 Slice 的 8-bits 总线宏。其中，I[7:0]和 O[7:0]分别是信号的输入端口和输出端口，En[7:0]是控制端口，控制通过总线宏信号的连接与断开，处于静态模块中，保证重构模块在配置过程中，重构区域发射的信号不通过总线宏影响静态模块的运行。使用总线宏进行通信时，某些位不需要时，在控制端口相应位赋值 1，即可切断重构模块向静态模块传输的信号。

在顶层模块中，图 4.14 所示总线宏的 HDL 实现如图 4.15 所示：

```
component busmacro _l2r _enable is
port (
  input0 : in std_logic;      input1 : in std_logic;
  input2 : in std_logic;      input3 : in std_logic;
  input4 : in std_logic;      input5 : in std_logic;
  input6 : in std_logic;      input7 : in std_logic;
  enable0 : in std_logic;     enable1 : in std_logic;
  enable2 : in std_logic;     enable3 : in std_logic;
  enable4 : in std_logic;     enable5 : in std_logic;
  enable6 : in std_logic;     enable7 : in std_logic;
  output0 : out std_logic;    output1 : out std_logic;
  output2 : out std_logic;    output3 : out std_logic;
  output4 : out std_logic;    output5 : out std_logic;
  output6 : out std_logic;    output7 : out std_logic
);
end component;
```

图 4.15 使能信号总线宏的 HDL

用户只需将重构模块通向静态模块方向的信号以及指定的使能信号赋予相应的端口即可。

基于 slice 的总线宏 LOC 约束的语法为：

INST "bus_macro" LOC = "SLICE_X aYb" ；

总线宏满足跨越重构模块和静态模块条件下，a 和 b 需为偶数，且小于重构区域的范围。

4.2.3.3 基于 Slice 的总线宏的实现

在应用中，用户往往需要根据系统实际需求设计满足自身要求的总线宏。下面以基于 Slice 的总线宏为例，简要介绍总线宏的实现方法。

总线宏的实现主要通过 Xilinx FPGA Editor 工具来生成。其基本步骤是：

1．在 FPGA Editor 中新建 hard macro 文件，生成一个后缀为.nmc 的文件。

2．根据实际需求，确定总线宏的跨度和所需的 slice 数目。

3．具有连接关系的 slice 间确定本地连线资源，并添加与之对应的内部输入输出端口。

4．对 slice 片内资源进行连线，并设置参考 slice。

5．根据总线宏结构，将相对应信号端口、本地连线资源通过路由连接，形成通路。

6.运行指令 Xdl –ncd2xdl busmacro_name 将生成的.nmc 文件转化为.xdl 文件，见附录 B。

生成的.xdl 文件包含总线宏中每个 slice 资源利用和 slice 间连线的描述信息，其语法形式为：

```
# module <name> <inst_name> ;                    总线宏文件名
# port <name> <inst_name> <inst_pin>;            总线宏外部端口定义
# instance ...;                                  总线宏内部 slice 实例
# net ...;                                       总线宏内部连线
#   endmodule <name>;                            总线宏解说
```

（1）总线宏外部端口定义

总线宏的外部端口定义需用户手动添加。根据用户在 FPGA Editor 中设定的端口实例，详细标志端口名、端口所属 slice 名以及端口的信号传输方向。

```
port "enable0" "slice4" "G2";      port "enable1" "slice4" "F2";
port "input0" "slice0" "G1";       port "input1" "slice0" "F1";
port "output0" "slice4" "Y";       port "output1" "slice4" "X";
```

（2）总线宏内部 slice 实例

总线宏内部 slice 实例描述了每个 slice 资源利用信息见图 4.16：

```
inst "slice4" "SLICE",placed R54C12 SLICE_X23Y53  ,
cfg  "  BXINV::#OFF  BXOUTUSED::#OFF  BYINV::#OFF
BYINVOUTUSED::#OFF  BYOUTUSED::#OFF  CEINV::#OFF
CLKINV::#OFF  COUTUSED::#OFF  CY0F::#OFF  CY0G::#OFF
CYINIT::#OFF  CYSELF::#OFF  CYSELG::#OFF  DIF_MUX::#OFF
DIGUSED::#OFF  DIG_MUX::#OFF  DXMUX::#OFF  DYMUX::#OFF
F:slice4.F:#LUT:D=A1*A2  F5USED::#OFF  FFX::#OFF
FFX_INIT_ATTR::#OFF  FFX_SR_ATTR::#OFF  FFY::#OFF
FFY_INIT_ATTR::#OFF  FFY_SR_ATTR::#OFF  FXMUX::F
FXUSED::#OFF  F_ATTR::#OFF  G:slice4.G:#LUT:D=A1*A2  GYMUX::G
G_ATTR::#OFF  REVUSED::#OFF
SHIFTOUTUSED::#OFFSLICEWE0USED::#OFF  CEWE1USED::#OFF
SLICEWE2USED::#OFF  SOPEXTSEL::#OFF  SOPOUTUSED::#OFF
SRFFMUX::#OFFSRINV::#OFFSYNC_ATTR::#OFF  WF1USED::#OFF
WF2USED::#OFF  WF3USED::#OFFWF4USED::#OFF  G1USED::#OFF
WG2USED::#OFF  WG3USED::#OFFWG4USED::#OFF  BMUX::#OFF
XUSED::0 YBMUX::#OFF YBUSED::#OFF YUSED::0 " ;
```

图 4.16 slice 资源利用信息

图 4.16 易见，slice 资源利用信息的基本形式为 componentname::#parameter[40]。所有的组件名对应于图 4.17 所示 slice 内部各资源名称[20]。参数只要不设定为#OFF 就表明相应的组件被用户使用。图 4.16 表示用户使用了一个 slice 内部的两个查找表 F、G：查找表 F 的端口 A1 和 A2 被作为输入信号端口，其对应的输出信号端口为 Y；查找表 G 的端口 A1 和 A2 被作为输入信号端口，其对应的输出信号端口为 Y。

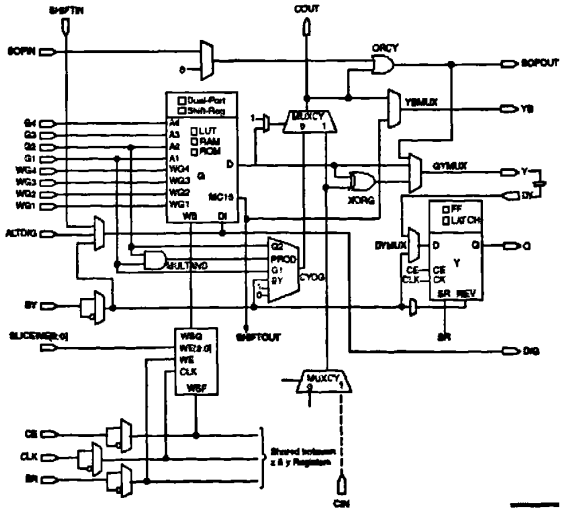


图 4. 17 half-slice 结构图

在带有使能信号的基于 Slice 的总线宏实现过程中，对于连接使能信号的 slice 的资源利用信息，用户需要在.xdl 文件按照语法显示的标志出相应的输入/输出端口名。

（3）总线宏内部连线

总线宏内部连线显示总线宏中 slice 间连线所占用的布线资源。

7. 修改.xdl 文件后，再利用指令 Xdl –xdl2ncd busmacro_name，即可产生相应的 Slice 总线宏。

4.2.3.4 基于 TBUF 的总线宏和基于 Slice 的总线宏的实验比较

在本文实验中，分别测试了 TBUF 的总线宏和基于 Slice 的总线宏。通过实验结果的比较，基于 Slice 的总线宏在端口信号赋值和总线宏控制方面明显优于基于 TBUF 的总线宏，主要有以下两个原因：

（1）对于用户设计的 IP 模块，综合工具经常会“优化”掉一些它认为是“冗余”的逻辑，造成 IP 模块端口信号不完整性。不同的 IP 模块被“优化”掉的信号往往不同。在实验中，用基于 TBUF 的总线宏来进行通信时，为了实现局部重构，必须将总线宏上对应“优化”掉信号的数据位设置为无信号状态。对于不同的 IP 模块，根据“优化”程度，则需设置不同的总线宏信号连接。这对于局部动态重构要求 IP 模块的端口信号相同的条件不再满足，将导致局部重构失败。而基于 Slice 的总线宏则很好的解决了这个问题，不管 Slice 总线宏上对应的信号是否被“优化”，都将不影响 IP 模块向 OPB 总线传输信号，有效的实现局部重构。

（2）重构模块配置的成功与否对局部重构系统起决定性作用。这是因为重构模块在配置过程中，模块所占区域会通过总线宏向静态模块发射状态不稳定的信号，这些不稳定的信号往往会导致重构模块配置中断，更严重则会使整个系统崩溃。在实验中，基于 TBUF 的总线宏是通过方向控制信号来实现传输信号，因此静态模块是不能控制来自重构模块方向的信号。为了实现局部重构，必须在静态模块中添加特定的控制模块，只接受完成配置后来自重构模块的信号。该控制模块虽能控制配置过程中的无状态信号，但不稳定。在实际实验中，经常导致内部配置访问通道(ICAP)向 FPGA 写入配置数据流时中途中断，导致系统崩溃。而基于 Slice 的总线宏在内部设置了专门的控制信号 en[0:7]来控制总线宏的断开和连接，能有效的控制配置过程中的无状态信号，同时，仅需在静态模块中设置相应的信号控制位即可。

4.3 过程级动态部分重构系统的设计

4.3.1 系统硬件设计

Xilinx EDK提供了包括处理器和大量外设在内的丰富的硬件IP资源供用户使

用，在加密系统的设计中首先需要选择合适的处理器IP核、各类系统外设IP核等资源用于构建基础的硬件平台，然后定制专用的加密算法核插入到基础硬件平台的设计中。

目前，Xilinx EDK 系统的开发是基于 IP 核的。IP 核具有良好可重用性和可更改性，能大大减轻设计者的负担。

Xilinx EDK 提供了面向处理器内部总线 PLB(Processor Local Bus)、片上外围总线 OPB 和快速单工连接(Fast Simplex Link，FSL)的三种 IP 核的便捷实现方式。其中，PLB 和 OPB 属于 IBM 公司开发的芯片内部总线通讯结构 CoreConnect[41]，适用于 PowerPC 硬核微处理器。PLB 主要提供了一个高带宽、低延迟、高性能的处理器内部总线[42]；OPB 则用于连接具有不同的总线宽度及时序要求的外设和内存[43][44]。FSL 则是 MicroBlaze 软核处理器特有的总线，能够为处理器与其它的系统部件之间的通讯提供高速连接[45]。

本文实验中，系统只应用基于寄存器实现的 OPB 总线 IP 核，其结构如图 4.18 所示[46]：

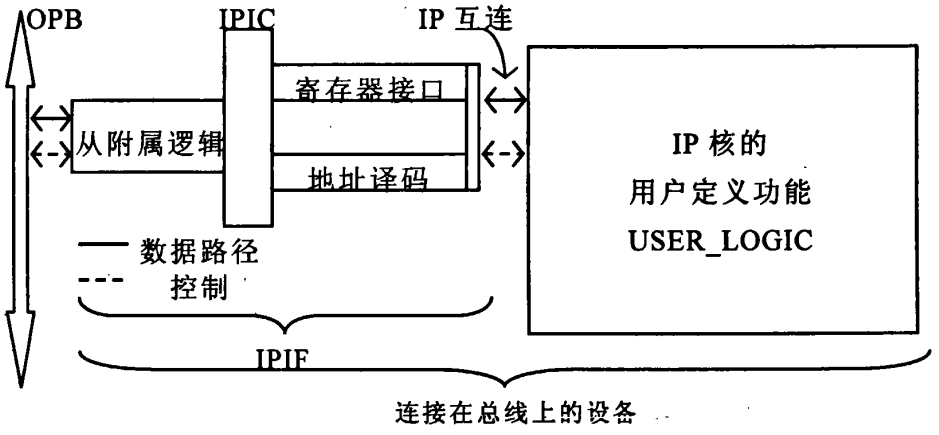


图 4.18 仅使用寄存器访问的设备通过 IPIF 连接 OPB 总线

图中易见，IP 核是通过 OPB IPIF 模块与 OPB 总线连接[46]。用户创建自己定义的 IP 核时，只需要改变 USER_LOGIC 模块即可，不需要关心 IP 核与 OPB 总线接口设计。IPIF 模块还提供 FIFO、DMA 等多种逻辑，用户可以根据需要定制自己需要的 IPIF 接口。

下面将介绍动态部分可重构系统中使用的挂载在 OPB 总线上的 IP 核结构：

（1）OPB HWICAP

OPB HWICAP IP 核是 Xilinx 公司将 ICAP 原语封装成能挂载 OPB 总线下的形式，如图 4.19 所示：

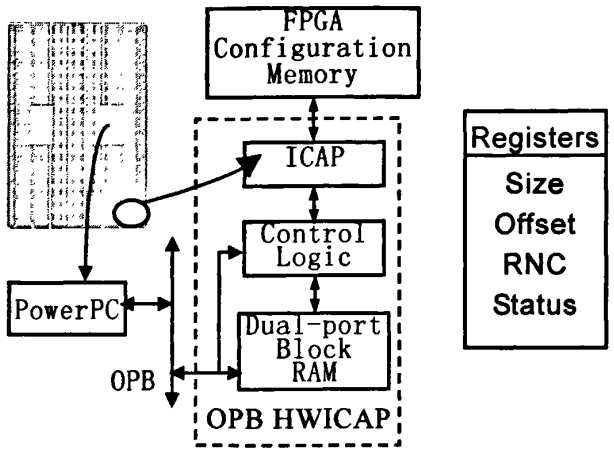


图 4. 19 OPB HWICAP 示意图

图中易见，HWICAP 由 ICAP 原语、控制逻辑和存储资源三部分组成。ICAP 原语既能够对 FPGA 进行配置也能对 FPGA 片上逻辑进行回读。控制逻辑则负责 ICAP 的读写状态以及 ICAP 和 BlockRAM 之间的数据传输。HWICAP 模块内部的存储资源包含 BlockRAM 和寄存器。BlockRAM 的大小为 16K bit，用来保存配置和回读阶段的临时数据，有效提高 ICAP 配置和回读速度。寄存器有 4 个，大小都是 32bit，分别为 Size、BRAM Offset、RNC 和 Status。Size 主要记录 BlockRAM 和 ICAP 原语之间传输的字节数；BRAM Offset 指示 ICAP 原语从 BlockRAM 读取或者写入数据时偏移地址；RNC(Readback not Configure)则控制 ICAP 原语的工作状态，当 RNC＝1 时 ICAP 从配置缓存中读取数据保存在 BlockRAM 中，RNC＝0 时 ICAP 将 BlockRAM 中的数据写入到片上配置缓存中；Status 显示 HWICAP 的工作状态，包含成功 Done 位、失败 Error 位等。其他详情请见文档[29]。

HWICAP 工作原理是：配置状态下，在 PowerPC 硬微处理器的控制下，将部分配置文件以 16K bit 为单位写入到 HWICAP 内部的 BlockRAM 中，然后驱动 ICAP 原语将 BlockRAM 中的数据写入到配置缓存中，依次循环直至部分配置文件全部写完为止，可通过 Status 寄存器验证操作是否成功；回读操作与配置操作正好相反。

（2）数据加密标准 DES 算法

DES 是上一代的加密标准，是当前使用最普遍的加密算法之一[47]。

DES 算法的入口参数有三个：Key、Data 和 Mode。其中 Key 长度为 8 个字节共 64 位，是 DES 算法的密钥；Data 也为 8 个字节 64 位，是被加密或被解密的数据；Mode 为则 DES 的加密或解密工作模式。

DES 算法的工作方式是：如 Mode 为加密，则用 Key 去把数据 Data 进行加密，生成 Data 的密码形式(64 位)作为 DES 的输出结果；如 Mode 为解密，则用 Key 去把密码形式的数据 Data 解密，还原为 Data 的明码形式(64 位)作为 DES 的输出结果。

DES 是对称加密，因此，解密和加密的过程一样，密钥也一样。

实验中，DES 模块是挂载在 OPB 总线上基于寄存器实现方式的 IP 核，使用了 9 个 32 位的寄存器。其端口定义如图 4.20 所示：

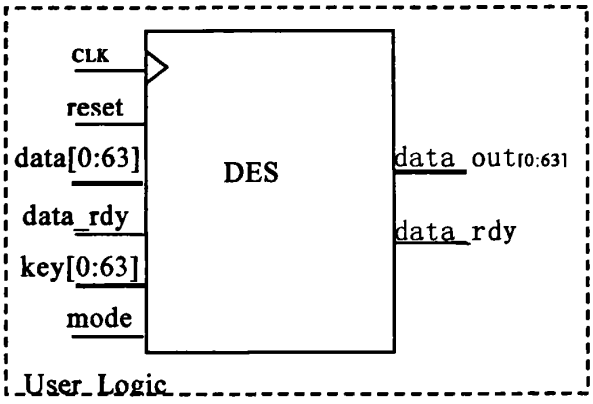


图 4. 20 DES 模块的 User_Logic 部分

图中，data[0:63]为待加密或解密的输入数据，对应寄存器为 R3^R4；data_rdy 表示输入数据是否有效(有效/无效＝1/0)，对应寄存器 R0；key[0:63]为密钥，对应寄存器为 R5^R6；mode 为加密或者解密模式(加密/解密＝0/1)，对应寄存器为 R1；data_out[0:63]为已加密或解密的输出数据，对应寄存器为 R7^R8；data_rdy 表示输出数据是否有效(有效/无效＝1/0)，对应寄存器为 R2。

（3）高级加密标准 AES 算法

AES 是下一代的加密算法标准，具有速度快、安全级别高等优点[47]。目前 AES 标准的一个实现是 Rijndael 算法。

AES 算法输入 128 位数据，密钥长度也是 128 位，要进行 10 轮的加密。每一轮都需要一个与输入分组具有相同长度的扩展密钥的参与。AES 进行一次加密的过程包括四个不同的阶段，由一个混淆和三个代换组成：S-盒变换，是一种字节代换，用算法的 S-盒完成分组中的按字节代换；行变换，也叫行移位，是一个简单的置换；列变换，也叫列混淆，是一个利用在域 GF(28)上的算术特性代换；轮密匙加，利用当前分组和扩展密码的一部分进行按位异或运算。

本文实现的 AES 加密模块也是挂载在 OPB 总线上基于寄存器实现方式的 IP 核，使用了 14 个 32 位的寄存器。其端口定义如图 4.21 所示：

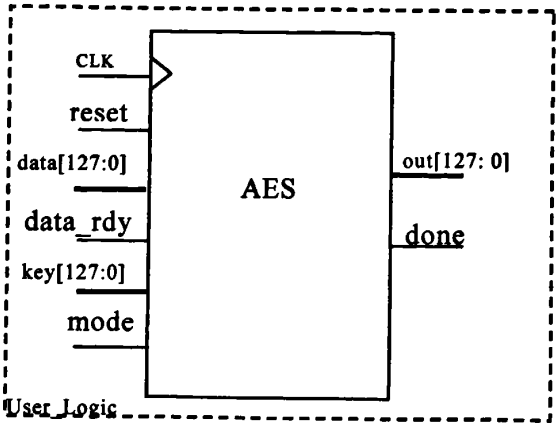


图 4. 21 AES 加密模块的 User_Logic 部分

图中，data[127:0]为待加密的输入数据，对应寄存器为 R1^R2^R3^R4；data_rdy 表示输入数据是否有效(有效/无效＝1/0)，对应寄存器 R0；key[127:0]为密钥，对应寄存器为 R5^R6^R7^R8；data_out[127:0]为已加密或解密的输出数据，对应寄存器为 R9^R10^R11^R12；mode 为加密或者解密模式(加密/解密＝0/1)，对应寄存器为 R13。

4.3.2 系统实现流程

在利用 Xilinx EDK 获得初始的系统时会自动产生相应的文件目录，目录中包含了动态部分可重构系统的实现过程中不同阶段所需要的初始物理实现文件。本文所创建系统的具体的文件目录如图 4.22 所示：

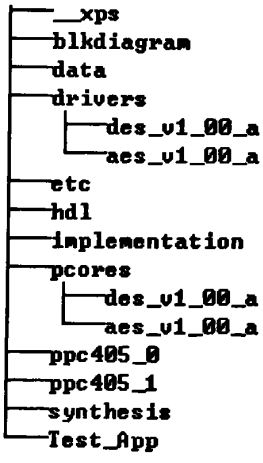


图 4. 22 Xilinx EDK 生成系统结构图

图中，data 目录是 Xilinx EDK 根据系统中的时钟资源和各外设所对应的 I/O 资源自动生成的系统约束文件；pcores 和 drivers 目录则分别对应用户添加 IP 核的硬件实现和驱动程序，用户在应用程序中通过直接调用驱动程序来驱动配置在 FPGA 片上的 IP 核；hdl 和 synthesis 目录存放了 Xilinx EDK 综合阶段各 IP 核所

产生的硬件描述语言(Hardware Description Language，HDL)文件以及对应文件；implementation 目录包含系统中 IP 核的网表文件以及片上存储器映射文件(BlockRAM Memory Map，BMM)；ppc405_0 和 Test_App 目录则保存了应用程序以及系统所调用的库文件。

虽然遵照图 3.1 所示的设计流程，但由于动态局部重构流程中涉及到的文件比较多，实现起来比较繁杂。为了便于系统的实现，用户在模块设计的具体实现之前，很有必要建立合适的文件结构来简化过程。本文根据模块设计流程采用如图 4.23 所示的文件结构，通过实验证明，该目录结构能够很好的组织和管理各阶段的设计和实现文件。

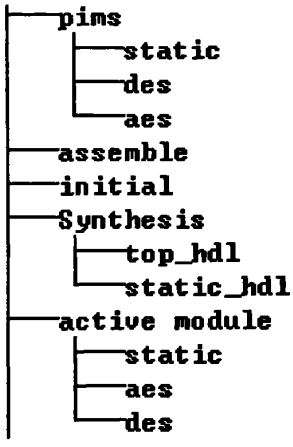


图 4.23 重构系统实现文件结构

其中，pims 目录是为了有利于合并阶段的完成而添加的，包括顶层模块、固定模块和重构模块的已完成布局布线的物理实现。其他各文件目录分别对应模块设计流程的各个阶段。

1．初始预算阶段

初始预算阶段命令行如下：

ngdbuild -p xc2vp30ff896-7 -uc top.ucf -modular initial top.ngc

其中，选项-p 指定系统的目标平台，包括型号、模式等信息；-uc 选项指定系统的约束文件，为系统内部组件提供特定的资源，其他选项详细内容可参看文档[28]。

2．模块激活阶段

模块激活阶段完成各个子模块布局布线，命令行如下：

ngdbuild -p xc2vp30ff896-7 –uc *top*.ucf -modular module –active *module_name*.ngc .. /initial/*top*.ngc *module _name*.ngd
map *module_name*.ngd
par –w *module_name*.ncd *module_name _routed*.ncd
trce *module_name _routed*.ncd
pimcreate -ncd *module_name _routed*.ncd ../pims

其中，map 命令对设计进行映射，由于所有的模块设计信息都编码在输入文件 NGD 中，因此命令中不再需要其他特别的参数；par 命令实现设计的布局布线，"-w"选项表示可以覆盖已经存在的 NCD 文件，该命令同样也不需要特殊的参数。trce 命令能对实现的设计进行时序检查。上述各命令行的选项并不唯一，用户可以根据自己的需求选择适当的选项来生成所需的文件，详细内容可参看文档[28]。

布局布线后，用户须用FPGA Editor工具[49]来检查各个子模块的布线情况，保证除了通过总线宏的信号跨越模块边界外，其他信号不能有任何越界现象。如发生越界，必须重新布局布线，即返回初始预算阶段，修改约束文件，通过延长总线宏长度或者总线宏位置来避免。

3．合并阶段

合并阶段实现系统整体布局布线，具体命令行如下：

ngdbuild -p xc2vp30ff896-7 –uc *top.ucf* -bm *system_sutb.bmm* -modular assemble -pimpath ../pims –use_pim *static* –use_pim *ipcore1 top*.ngc

map *top.ngd*

par -w *top.ncd top _routed.ncd*

trce *top _routed.ncd*

其中-bm 选项指定系统片上存储器映射文件；-pimpath 则指示了子模块物理实现所存储的文件目录；选项-use_pim 指定组成系统各模块的文件。其他选项同激活阶段的作用相同，详细内容可参看文档[28]。

命令执行完毕后，生成的系统完整布线文件可供 BitGen 工具生成可配置的流文件。

布局布线后，用户须用 FPGA Editor 工具来检查整个系统的最终布线文件 top_routed.ngc。布线图中，除了总线宏跨越模块边界外，其他信号是否发现了越界现象。如发生越界需返回到初始预算阶段修改约束文件，重新对整个系统进行布局布线。检查布线文件更重要的是要确认总线宏位置是否对齐，以保证模块通信正常。

4．配置文件的生成

本文实验中，动态局部重构系统的实现涉及三类配置文件：系统的硬件全局配置文件、重构模块的部分配置文件和包含了软件和硬件信息的初始配置文件。

（1）系统的硬件全局配置文件由合并阶段生成的系统布线文件转化而来，可直接下载到 FPGA 中运行，但由于没有添加应用程序，因此用户不可以运用系统的功能。

实验中，生成系统硬件全局配置文件命令行如下：

bitgen -d -g startupclk: jtagclk *top_routed*.ncd *hardware*.bit

选项"-d"为设置不进行 DRC(设计规则)检查，其他选项详细内容可参看文档

[28]。

（2）重构模块的部分配置文件由模块激活阶段，重构模块产生的布线文件生成。系统正是通过写入不同 IP 核对应的配置文件来改变系统的功能。

实验中，生成部分配置文件命令行如下：

bitgen -d -b -w -g activereconfig: Yes *module_name _routed*.ncd *partial*.bit

为了部分配置文件能正常工作，该命令中必须包含"-g activereconfig Yes"选项，确保部分配置文件在配置过程中，其余部分可以仍然保持工作状态。其他选项详细内容可参看文档[28]。

（3）初始配置文件是 FPGA 上电后自动加载的文件，由处理器运行的软件跟硬件全局配置文件生成，用户通过软件可以控制重构的实现与否。

实验中，生成初始配置文件命令行如下：

data2mem -bd executable.elf –bm *partial _bd*.bmm -bt *hardware*.bit -o b *download*.bit

选项-bd、-bm、-bt 和-o 分别指定紧随其后的已编译应用程序名、已映射的片上存储器文件名、硬件全局配置文件名以及初始配置文件名。其他选项详细内容可参看文档[28]。

4.4 实验结果分析

实验使用的硬件平台是由 Xilinx 公司大学计划提供的 XUP Virtex-II Pro XC2VP30 实验板。平台的软件开发环境是 Xilinx Platform Studio 9.1i 和 ISE 9.1。

实验以加密算法 128 位分组对称加密算法 AES 和 DES 为动态重构应用研究对象，将它们以基于寄存器的实现方式封装成 IP 模块，并挂载在 OPB 总线上。用户通过调用 IP 模块对应的硬件函数[50][51][52]即可实现不同的加密功能，对应的硬件函数分别为 OPB_DES()和 OPB_AES()。当应用程序调用硬件函数时，系统先检查对应的 IP 模块是否已配置在 FPGA 中，如存在则直接运行；否则系统先从 Flash 卡中读取相应的部分配置比特流文件到片内缓冲区，通过配置控制器 ICAP 完成重构配置后再实现硬件函数功能。

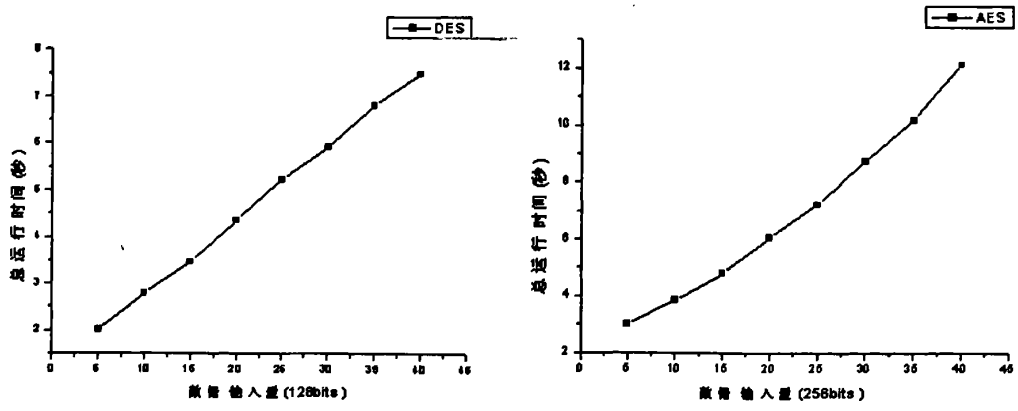
通过最终的实验，表 4.3 显示了在处理器时钟为 100MHz 情况下，各模块所占用的 XC2VP30 FPGA 的片内资源情况：

表 4.3 系统资源消耗

资源类型	资源数						
	AES		DES		静态模块		片内资源
	Total	%	Total	%	Total	%	
SLICEs	8,594	62	4,694	34	1,271	9	13,696
Slice Flip Flops	1,448	5	5,475	19	1,301	4	27,392
4 input LUTs	15,671	57	4,575	16	1,116	4	27,392
PPC405s	0	0	0	0	1	50	2
RAMB16s	0	0	0	0	65	47	136
ICAP	0	0	0	0	1	100	1

从表中可以看到，AES 占用了芯片 62%的资源，DES 占用 34%，静态模块占用 9%，其总资源需求量超过了 XC2VP30 FPGA 可提供的资源总数，所以如果想使得系统中同时运行 AES 和 DES 两个 IP 模块，不采用动态重构技术是无法在单片的 XC2VP30 FPGA 芯片上实现的。

重构配置过程中，AES 和 DES 在静态模块控制下利用重构区域的资源来实现功能切换，使用较少的资源完成了更多的功能模块，提高了资源的利用率。当然，占用资源少是以一定的时间开销为代价的。这个时间开销主要是动态可重构模块的配置时间。对于待处理数据量小，运行时间不长的应用，重构配置时间很可能会显著增加系统的整体运行时间；但是当待计算数据量很大，系统运行时间较长时，重构配置的时间将远小于重构模块的运行时间。



a）DES 模块总运行时间和数据输入量的关系　　b）AES 模块总运行时间和数据输入量的关系

图 4.24 加密模块总运行时间和数据输入量的关系图

图 4.24 中分别显示了加密模块 DES 和 AES 总运行时间随数据输入量的变化关系。其中，模块的总运行时间是模块的重构配置时间和模块运行时间之和；数据输入量则是加密模块执行一次运算所需的信号量大小，DES 每个输入数据大小

为 64bits，AES 每个输入数据大小为 128bits，由于每个加密模块都有待加密数据和密钥两个输入信号量，因此，模块一次接受的数据量分别为 128bits 和 256bits。

加密模块 DES 和 AES 的重构配置时间是常量，在实验中，XC2VP30 FPGA 上，DES 的配置需时为 1.22 秒，AES 的配置需时为 2.14 秒。因此，根据图 4.24 易知，随着加密模块的执行次数增加，相应的总运行时间也随之增加，加密模块重构配置开销在系统的总的运行时间比重变小，大大降低配置开销对系统的整体性能的影响。

4.5 小结

本章在现有的基于可重构逻辑器件的嵌入式系统开发流程上增加了对基于模块的部分重构技术的支持。针对系统的设计需求进行系统部件的选型与设计，解决了过程级动态重构系统所涉及的关键问题。通过加密IP模块系统的实现验证了本论文所提设计方法的有效性，也体现出了过程级动态部分重构系统具有的性能优势。

结论和展望

可重构计算是当前计算机系统结构领域的研究热点。动态部分可重构技术作为可重构计算技术重要发展趋势，在性能和灵活性等方面具有很大的优势。当前的主流可编程逻辑器件已经对动态部分重构技术提供了充分的支持，但是相关设计方法的缺失以及动态部分重构系统实现的复杂性，制约了动态部分可重构技术的应用。

本文选择在 Xilinx Virtex-II Pro XC2VP30 平台上利用 FPGA 局部重构技术实现了一个过程级动态部分可重构系统，对如何利用 ICAP 对 OPB 总线上的加密 IP 模块进行动态重构进行了研究。与现有的设计方法相比，本论文提出的基于模块的部分重构设计方法主要具有以下优势：以硬件描述语言作为设计入口，无需关注硬件细节，降低了设计门槛、设计流程能够普遍适用于多款 Xilinx FPGA，具有跨器件系列的可复用性、充分利用 CAD 工具，提高了设计效率，保证了设计质量。

在系统的开发过程中，借鉴了模块化设计技术的思路，对模块划分、模块通信和系统硬件实现三个关键问题进行了研究。模块划分阶段，根据应用的功能分析手工将系统划分为重构模块和静态模块，并对模块的端口定义和模块资源进行了分析。重构模块和静态模块间的通信是通过基于 Slice 的总线宏来完成，在通信效率和信号控制方面优于传统的基于 TBUF 的总线宏。经过初始预算、模块激活、模块合并和配置数据生成步骤生成系统的硬件实现，实现了系统的运行时重构。实验表明系统实现了硬件资源的分时复用，有效的提高系统资源利用率。

本论文需要开展的进一步研究工作包括：

1)完善部分重构设计方法。本论文提出的基于模块的部分重构设计方法虽然能有效的实现动态部分可重构系统，但在实践中显得不灵活，主要表现在：模块大小和位置必须预先确定，在运行过程中不能更改；模块区域的 IOBs 资源只能被所属模块使用，不同模块对 IOB 需求的不同造成整体 IOB 资源充足的情况下局部 IOB 资源不足。

2)缩短重构开销。本文的重构开销包括从外部存储器读入配置数据和将配置数据通过 ICAP 下载到 FPGA 片内两部分。其中，外部存储器读入配置数据占总开销的大部分，这部分开销的缩短将大大提高系统运行效率。同时 ICAP 在 FPGA 内部操作配置数据是采用“读出－修改－写回”的机制，目前读出阶段的效率比较低，因此用户还需采用预配置等措施解决。

3)总线宏的设置。Xilinx XC2VP30 FPGA在模块边界上提供的总线宏资源有

限，虽然在数量上满足模块之间的通讯需要，但是因为总线宏单元的分布过于密集，信号线经常会发生越界行为导致系统布局布线失败，影响重构系统的开发效率。

参考文献

[1] 覃祥菊,朱明程,张太镒等. FPGA动态可重构技术原理及实现方法分析.电子器件,2004,27(2):277-282

[2] 段然,樊晓桠,高德远等.可重构计算技术及其发展趋势.计算机应用研究,2004,21(8):14-17

[3] Andre DeHon,John Wawrzynek. Reconfigurable Computing:What,Why,and Implications for Design Automation. In:Proceedings of the 36th ACM/IEEE conference on Design automation. New Orleans,1999,610-615

[4] Katherine Compton,Scott Hauck. Reconfigurable Computing:a Survey of Systems and Software. ACM Computing Surveys,2002,34 (2):171-210

[5] 周学功.可重构嵌入式系统样机平台与操作系统研究:[复旦大学博士学位论文].上海:复旦大学,2007,26-38

[6] Zhong P,Martonosi M,Ashar P. FPGA-based SAT solver architecture with near-zero systhesis and layout overhead. IEE Proceedings Computers & Digital Techniques, 2000,147(3):135-141

[7] Alexandre F. Tenca, Milos D. Ercegovac. A variable long-precision arithmetic unit design for reconfigurable coprocessor architectures. In:Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines. Napa,1998,216-225

[8] Rencher M, Hutchings B.L. Automated target recognition on SPLASH2. In:IEEE Symp on Field-Programmable Custom Computing Machines. New York, 1997,192-200

[9] Weinhardt M, Luk W. Pipeline vectorization for reconfigurable systems. In: Proceedings of IEEE Symp on Field-Programmable Custom Computing Machines. New York,1997, 24-32

[10] Scott Hauck,William D. Wilson. Run length compression techniques for FPGA configurations. In:Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines. Washington DC, 1999, 284-287

[11] Brad Hutchings,Michael J. Wirthlin. Implementation Approaches for Reconfigurable Logic Applications. In: Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications.London,1995,419-428

[12] OPENCORES.ORG. OPENCORES Projects.http://www.opencores.org,2007-09-1

[13] Design and Reuse (D&R). IP Cores. http://www.design-reuse.com,2007-09-01

[14] Virtual Component Exchange(VCX). IP Catalog.http://www.thevcx.com,2007-09

[15] 国家 IP 核库. IP 核库分类. http://www.csip.org.cn,2007-09-01

[16] V. Nollet, P. Coene, D. Verkest,et al. Designing an Operating System for a Heterogeneous Reconfigurable SoC. In:Proceedings of. the RAW'03 workshop. Nice,2003.174-180

[17] Erik Anderson,Jason Agron,Wesley Peck,et al.Enabling a Uniform Programming Model Across the Software/Hardware Boundary. In:Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. New York . New York 2006. 89-98

[18] 郑友泉.数字和 DSP 系统平台级设计和实现.http://www.eefocus.com/html/07-08/49220908316528.shtml,2008-01-20

[19] 曲英杰,王泌,王昭顺.可重构体系结构的特征及应用.计算机工程与应用.2001,17(44):19-21

[20] Xilinx Inc.Virtex-II Pro and Virtex-II Pro X Platform FPGAs:Complete DataSheet.http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_profpgas/index.htm,2005-09-15

[21] 田耘.FPGA 开发实用教程.http://www.eefocus.com/ html/08-03/415520110334ALD8.shtml,2008-05-09

[22] Xilinx Inc. Virtex FPGA Series Configuration and Readback. http://www.xilinx.com/support, 2005-03-11

[23] Xilinx Inc.Virtex-II Pro and Virtex-II Pro X FPGA User Guide. http://www.xilinx.com/support,2005-03-23

[24] Xilinx Inc.Xilinx University Program Virtex-II Pro Development System. http://www.xilinx.com/support,2005-03-08

[25] Xilinx Inc.PowerPC 405 Processor Block Reference Guide. http://www.xilinx.com/support,2005-06-20

[26] Xilinx Inc.PowerPC Processor Reference Guide. http://www.xilinx.com/ support, 2003-09-02

[27] Xilinx Inc.Two Flows for Partial Reconfiguration: Module Based or Difference Based. http://www.xilinx.com/support,2005-05-01

[28] Xilinx Inc. Development System Reference Guide. http://www.xilinx.com/support, 2005-03, 75-140

[29] Xilinx Inc.Product Specification:OPB HWICAP. http://www.xilinx.com/ support,

2005-04-04

[30] Xilinx Inc.A Hands-On Guide to EffectiveEmbedded System Design.http://www.xilinx.com/ise/embedded/edk.htm,2007-01-01

[31] Xilinx Inc.LibXil FATFile System(FATfs) EDK 9.1i. http://www.xilinx.com/support,2007-01-01

[32] Kiran Bondalapati,Viktor K. Prasanna. Reconfigurable Computing Systems. In:Proceedings of the IEEE 2002. New York,2002,1201-1217

[33] Xilinx Inc.EXTERNAL_REV_C_SCHEMATICS.http://www.xilinx.com/ support, 2005-03-08

[34] Xilinx Inc.Constraints Guide.http://www.xilinx.com/support,2006-05-12

[35] Gregory M. A Module-Based Dynamic Partial Reconfiguration Tutorial. http:// icwww.epfl.ch/~gmermoud/,2004-11-1

[36] Jens Thorvinger. Dynamic Partial Reconfiguration of FPGA for Computational Hardware Support:[dissertation]. Copenhagen:Univ. of Lund, 2004, 40-61

[37] Blodget B,Bobda C,Huebner M,et al. Partial and Dynamically Reconfiguration of Xilinx Virtex-II FPGAs. In:Proceedings of International conference on field-programmable logic and applications No14. Antwerp,2004,801-810

[38] Sedcole P,Blodget B,Becher T,et al. Modular Dynamic Reconfiguration in Virtex FPGAs. Journal of IEE Proceedings-Computers and Digital Techniques, 2006,153(3):157-164

[39] Xilinx Inc.Early Access Partial Reconfiguration User Guide For ISE 8.1.01i UG208 (v1.1).http://www.xilinx.com/support/prealounge/protected/ index.htm, 2006-03-06

[40] Christopher Claus,Bin Zhang,Michael H¨ubne,et al. An XDL-based busmacro generator for customizable communication interfaces for dynamically and partially reconfigurable systems. In:Proceedings of Workshop on Reconfigurable Computing Education at ISVLSI 2007. Porto Alegre, 2007,68-75

[41] IBM Inc. The CoreConnect™ Bus Architecture. http:// www.chips.ibm.com /products, 2006-07-12

[42] Xilinx Inc.PLB Usage in Xilinx FPGAs. http://www.xilinx.com/ support, 2005-09-11

[43] Xilinx Inc.OPB Usage in Xiliax FPGAs. http://www.xilinx.com/ support, 2005-09-11

[44] Firefighters. On-Chip Peripheral Bus Architecture Specifications. http://www.edacn.net/bbs,2005-5

[45] Xilinx Inc.Fast Simplex Link(FSL)Bus(v 2.00a). http://www.xilinx.com/ support, 2005-09-11

[46] Xilinx Inc. OPB IPIF Architecture. http://www.xilinx.com/ support,2004-08

[47] Hemanth,Seanlee. IP cores for DES and AES. http://www.opencores.org, 2007-09-10

[48] Nicholas Peter Sedcole. Reconfigurable Platform-Based Design in FPGAs for Video Image Processing:[dissertation]. London:Univ. of London,2006,149-154

[49] Xilinx Inc. FPGA Editor Guide. http://www.xilinx.com/support, 2006-10

[50] Xilinx Inc. Device Driver Programmer Guide v1.2. http://www.xilinx.com/ support, 2002-07-21

[51] Wang Wei,Wu Qiang,Xie Wei. Hardware-Software Co-design for Dynamic Reconfigurable Computing with Collaborative Supports of Architecture and Operating System. In: Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design. Melbourne, 2007,275-279

[52] Qiang Wu,Wei Xie,Wei Wang. An Architecture and Programming Framework for Dynamic Reconfigurable Computing Systems. In: Proceedings of 9th JCIS 2006. Atlantis,2006,599-602

致　谢

在湖南大学三年的研究生学习阶段，使我的知识结构得以拓展，理论视野得以开阔，综合素质得以提升。在这毕业即将离开学校之际，我要感谢许多人：

首先，我要感谢我的导师吴强副教授。从论文的选题到结束，所有的研究工作都是在吴老师的悉心指导下完成。三年的研究生期间，吴老师在学术上给了我很多宝贵的指导，生活上给了我无微不至的关怀，工作上给予了我充分的支持，他的人格魅力深深的影响了我；同时，吴老师严谨的治学态度、精益求精的工作作风令我印象深刻，相信在我以后的工作和生活中将终生受用。

特别感谢师姐王伟、同窗邹祎在实验以及论文完成阶段给我的帮助，使我能够顺利完成毕业各方面的准备工作。

同时也感谢整个项目组成员彭蔓蔓老师、刘彦博士以及师弟黄瑜臣、袁虎和张维等同学，他们使我对整个项目有了更深的理解。

感谢我的室友赵孝敏、李贤鹏、何军球和梁华林在生活上给予我的帮助。

感谢在湖南大学研究生求学期间给予我无私帮助的所有老师、同学和朋友们，正是因为他们，我的求学时光才如此美好。

最后，我要特别感谢我的亲人，感谢他们在物质上的资助和精神上的鼓励。

附录 A 读研期间发表学术论文和参与科研项目

1．学术论文

[1] 赵远宁,吴强,邹祎.基于Virtex-II Pro系列FPGA的动态部分可重构系统设计与实现.中国科技论文在线（已录用）

2．科研项目

[1] 国家高科技研究计划（863）：面向可重构片上系统的过程级动态软硬件划分研究，项目编号：No.2007AA01Z104

附录 B 系统部分约束文件

```
//全局逻辑数字时钟管理器 DCM 和全局时钟多路缓冲器 BUFG 的 LOC 约束
INST "dcm_0" LOC = "DCM_X2Y0" ;
INST "bufg_1" LOC = "BUFGMUX1S"  ;
INST "ibufg_0" LOC = "BUFGMUX6P"  ;
//静态模块区域约束
INST "static" AREA_GROUP = "AG_static";
AREA_GROUP " AG_static "  MODE=RECONFIG;
AREA_GROUP " AG_static " RANGE = SLICE_X36Y159:SLICE_X91Y0 ;
AREA_GROUP " AG_static " RANGE = RAMB16_X3Y1: RAMB16_X7Y19 ;
AREA_GROUP " AG_static " RANGE = MULT18X18_X3Y1: MULT18X18_X7Y19 ;
//重构模块区域约束
INST "RPM" AREA_GROUP = "AG_RPM " ;
AREA_GROUP " AG_RPM " MODE = RECONFIG;
AREA_GROUP " AG_RPM " RANGE = SLICE_X0Y159:SLICE_X35Y0 ;
AREA_GROUP " AG_RPM " RANGE = RAMB16_X0Y0:RAMB16_X2Y19 ;
AREA_GROUP " AG_RPM " RANGE = MULT18X18_X0Y0:MULT18X18_X2Y19 ;
//基于 Slice 总线宏约束文件
INST "bus_macro13" LOC = "SLICE_X28Y108" ;
INST "bus_macro12" LOC = "SLICE_X28Y106" ;
INST "bus_macro11" LOC = "SLICE_X28Y104" ;
INST "bus_macro10" LOC = "SLICE_X28Y102" ;
INST "bus_macro9" LOC = "SLICE_X28Y100" ;
INST "bus_macro8" LOC = "SLICE_X28Y42" ;
INST "bus_macro7" LOC = "SLICE_X28Y38" ;
INST "bus_macro6" LOC = "SLICE_X28Y34" ;
INST "bus_macro5" LOC = "SLICE_X28Y30" ;
INST "bus_macro4" LOC = "SLICE_X28Y26" ;
INST "bus_macro3" LOC = "SLICE_X28Y22" ;
INST "bus_macro2" LOC = "SLICE_X28Y18" ;
INST "bus_macro1" LOC = "SLICE_X28Y14" ;
INST "bus_macro0" LOC = "SLICE_X28Y12" ;
```

附录 C 基于 Slice 的总线宏 XDL 语言实现

//带使能机制的基于 Slice 的总线宏(信号传输方向从左到右，信号传输量为 2bit，跨度为 5CLB)

```
module "E:\newest\busmacro make\slice_l2r_enable_10" "slice3" , cfg "_SYSTEM_MACRO::FALSE" ;
port "enable0" "slice4" "G2";
port "enable1" "slice4" "F2";
port "input0" "slice0" "G1";
port "input1" "slice0" "F1";
port "output0" "slice4" "Y";
port "output1" "slice4" "X";
inst "slice0" "SLICE",placed R54C3 SLICE_X5Y53  ,
cfg " BXINV::#OFF BXOUTUSED::#OFF BYINV::#OFF BYINVOUTUSED::#OFF BYOUTUSED::#OFF
CEINV::#OFF CLKINV::#OFF COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF CYSELF::#OFF
CYSELG::#OFF DIF_MUX::#OFF DIGUSED::#OFF DIG_MUX::#OFF DXMUX::#OFF DYMUX::#OFF
F:slice0.F:#LUT:D=A1 F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::F FXUSED::#OFF F_ATTR::#OFF
G:slice0.G:#LUT:D=A1 GYMUX::G G_ATTR::#OFF REVUSED::#OFF SHIFTOUTUSED::#OFF
SLICEWE0USED::#OFF SLICEWE1USED::#OFF SLICEWE2USED::#OFF SOPEXTSEL::#OFF
SOPOUTUSED::#OFF SRFFMUX::#OFF SRINV::#OFF SYNC_ATTR::#OFF WF1USED::#OFF
WF2USED::#OFF WF3USED::#OFF WF4USED::#OFF WG1USED::#OFF WG2USED::#OFF
WG3USED::#OFF WG4USED::#OFF XBMUX::#OFF XUSED::0 YBMUX::#OFF YBUSED::#OFF YUSED::0
"  ;
inst "slice4" "SLICE",placed R54C12 SLICE_X23Y53  ,
cfg " BXINV::#OFF BXOUTUSED::#OFF BYINV::#OFF BYINVOUTUSED::#OFF BYOUTUSED::#OFF
CEINV::#OFF CLKINV::#OFF COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF CYSELF::#OFF
CYSELG::#OFF DIF_MUX::#OFF DIGUSED::#OFF DIG_MUX::#OFF DXMUX::#OFF DYMUX::#OFF
F:slice4.F:#LUT:D=A1*A2 F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::F FXUSED::#OFF F_ATTR::#OFF
G:slice4.G:#LUT:D=A1*A2 GYMUX::G G_ATTR::#OFF REVUSED::#OFF SHIFTOUTUSED::#OFF
SLICEWE0USED::#OFF SLICEWE1USED::#OFF SLICEWE2USED::#OFF SOPEXTSEL::#OFF
SOPOUTUSED::#OFF SRFFMUX::#OFF SRINV::#OFF SYNC_ATTR::#OFF WF1USED::#OFF
WF2USED::#OFF WF3USED::#OFF WF4USED::#OFF WG1USED::#OFF WG2USED::#OFF
```

```
WG3USED::#OFF WG4USED::#OFF XBMUX::#OFF XUSED::0 YBMUX::#OFF YBUSED::#OFF YUSED::0
" ;
   net "input1" ,
      inpin "slice0" F1 ,    ;
   net "input0" ,
      inpin "slice0" G1 ,    ;
   net "$NET_25" ,
      outpin "slice0" X ,
      inpin "slice4" F1 ,
      pip BRAMR54C2 E6END2 -> E6BEG2 ,
      pip R54C12 BY0 -> F1_B3 ,
      pip R54C12 F1_B3 -> F1_B_PINWIRE3 ,
      pip R54C12 W2END2 -> BY0 ,
      pip R54C14 E6END2 -> W2BEG2 ,
      pip R54C3 X3 -> E6BEG2 ,     ;
   net "$NET_24" ,
      outpin "slice0" Y ,
      inpin "slice4" G1 ,
      pip BRAMR54C2 E6END8 -> E6BEG6 ,
      pip R54C12 BY1 -> G1_B3 ,
      pip R54C12 G1_B3 -> G1_B_PINWIRE3 ,
      pip R54C12 W2END6 -> BY1 ,
      pip R54C14 E6END6 -> W2BEG6 ,
      pip R54C3 Y3 -> E6BEG8 ,       ;
   net "enable1" ,       inpin "slice4" F2 ,     ;
   net "enable0" ,       inpin "slice4" G2 ,     ;
   net "output1" ,       outpin "slice4" X ,     ;
   net "output0" ,       outpin "slice4" Y ,     ;
;
endmodule "E:\newest\busmacro make\slice_l2r_enable_10" ;
```

附录 D 系统片上存储器文件引用层次

/*片上存储器属于静态模块层次，层次名以顶层模块中静态模块的引用名 static 标志，片上存储器大小为 0xFFFE0000:0xFFFEFFFF (64 KB)*/

//未映射的片上存储器文件示例

```
ADDRESS_MAP ppc405_0 PPC405 100
ADDRESS_SPACE plb_bram_if_cntlr_1_bram_combined RAMB16 [0xFFFE0000:0xFFFEFFFF]
BUS_BLOCK
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_0 [63:62];
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_1 [61:60];
……
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_30 [3:2];
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_31 [1:0];
END_BUS_BLOCK;
END_ADDRESS_SPACE;
END_ADDRESS_MAP;
```

//已映射的片上存储器文件示例

```
ADDRESS_MAP ppc405_0 PPC405 100
ADDRESS_SPACE plb_bram_if_cntlr_1_bram_combined RAMB16 [0xFFFE0000:0xFFFEFFFF]
BUS_BLOCK
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_0 [63:62] PLACED = X4Y7;
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_1 [61:60] PLACED = X5Y6;
……
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_30 [3:2] PLACED = X7Y14;
static/plb_bram_if_cntlr_1_bram/plb_bram_if_cntlr_1_bram/ramb16_s2_s2_31 [1:0] PLACED = X6Y14;
END_BUS_BLOCK;
END_ADDRESS_SPACE;
END_ADDRESS_MAP;
```

附录 E 常见问题

//Xilinx EDK 阶段问题

1、问题：Analyzing file test/executable.elf...

WARNING:MDT - Elf file test/executable.elf does not reside completely within BRAM memory of processor ppc405_0.

WARNING:MDT - The sections of ELF residing outside BRAMs must be initialized separately using a debugger, a bootloader, or an ACE file

ERROR:MDT - Memory overlap detected between various program headers for processor ppc405_0

make: *** [implementation/download.bit] Error 1

解决方案：用户没有为应用程序的正常运行提供足够的片上存储器，在 Xilinx EDK 图形界面上增大片上存储器的地址空间。

//模块激活阶段问题

1、问题：ERROR:NgdBuild:647 - instance '<instance_name>' of module '<module_name>' has not yet been defined.

ERROR:NgdBuild:559 - Cannot find active block '<module_name>'

解决方案：此类错误在整个模块实现阶段都可能出现。主要是由于用户没有提供系统中所需的网表文件。用户只需复制相应的网表文件到目录下即可。

2、问题：ERROR:Place -

The design has objects that are constrained to a set of sites. The number of objects constrained (<bigger number>) is too large for the number of sites in this set (<smaller number>) .

COMPGRP: <PRM_instance_name>.SLICE

The following preference(s) in the PCF file are responsible for the failures COMPGRP "<PRM_module_name>.SLICE" LOCATE = SITE "<AG range constraint>" LEVEL 4 ;

解决方案：这是因为用户在初始阶段对重构模块和静态模块所设置的 AREA GROUP, AREA GROUP RANGE 区域约束中，为模块提供的资源不够造成的。用户只需返回初始预算阶段修改约束文件为模块提供足够的资源即可。

//模块合并阶段问题

1、问题：Processing BMM file

ERROR:NgdBuild:927 - Failed to process BMM file system_stub.bmm

Checking expanded design ...

ERROR:NgdBuild:604 - logical block 'system_0' with type 'system' could not be resolved. A pin name misspelling can cause this, a missing edif or ngc file, or the misspelling of a type name. Symbol 'system' is not supported in target 'virtex2p'.

解决方案：在静态模块的激活阶段，pimcreate 不能自动将静态模块的网表文件复制到../pims/static 文件夹内,因此在合并阶段涉及静态模块的属性因素会出现错误。用户将对应文件拷贝至相应文件夹即可。

//重构配置阶段问题

1、问题：系统运行正常，ICAP 的驱动程序执行正常，但 ICAP 执行完毕后，并没有发生重构操作。

解决方案：一般由于配置模式管脚 M0、M1、M2 设置为 1：0：1 所造成，用户需改变管脚的设置。可利用 iMPACT 工具将流文件转化为 MCS 文件，然后再通过 JTAG 下载存储到开发板上的 EPROM 内，在系统加电运行后，ICAP 能正常工作。

2、问题：系统配置过程中经常只能下载一部分配置文件，随即系统崩溃。

解决方案：这是由于在重构配置中，重构区域的无序信号通过总线宏进入了静态模块，影响了整个系统的时序，说明总线宏的信号控制机制不稳定。用户需要重新设计和实现总线宏，尤其对信号控制位。

3、问题：系统配置能顺利完成，但配置后串口显示端口出现乱码。

解决方案：这是由于模块实现阶段某些信号的布线发生了越界行为，用户需要再一次重新执行系统的模块实现。在模块激活阶段一定要检查信号的越界行为，如发生，需返回到初始预算阶段对约束文件进行修改，可通过改变总线宏的放置来避免。