摘 要

软硬件协同设计使嵌入式系统的软件和硬件设计互相协同、并行实现，有利于尽早发现错误、降低成本和提高系统性能，而软硬件任务划分是嵌入式协同设计的重要环节，对系统的后续设计实现及系统整体性能都有较大的影响。软硬件划分是在满足系统各项约束条件的前提下，为目标系统各个功能任务确定具体的实现方式，以达到各项约束最佳的软硬件分配方案。

可重构计算以其较高的灵活性和系统性能目前广泛应用于多个领域。随着以FPGA为代表的可重构硬件的发展，动态可重构系统已经成为了嵌入式系统的一个发展趋势。动态可重构系统不仅在系统结构上有别于传统的嵌入式系统，而且其任务执行方式也发生了根本性的变化，最大的特点是可以时分复用的使用可重构单元以硬件方式执行任务，极大的提高了系统性能。但是同时也带来了一些新的问题，如重构时间过长、可重构硬件资源布局和配置文件如何预读取等。由于动态可重构系统的诸多新特点新问题，传统的软硬件划分方法已经不再适用。因此本文在对软硬件划分和动态可重构系统特点研究的基础上，提出了基于权重动态可变的免疫算法的软硬件划分方法，并结合调度算法进行评估。这种算法能较好的适应动态可重构环境，使动态可重构系统的软硬件任务划分更加合理，能够充分发挥可重构单元的高速性，达到系统性能的最优化。

关键词：动态可重构系统，软硬件划分，免疫算法，任务调度

Abstract

Hardware/software co-design enable parallel implementation for the software and hardware design of embedded system and it also conduces to early detection of errors, costs reduction and system performance improvement. And hardware/software partitioning is one of the most important aspects of embedded system co-design. The result of HW/SW portioning has a greater impact to follow-up design and overall system performance. HW/SW portioning algorithm firstly must meet the demands of the target system, then it assigns each task of target system to software or hardware computing unit in order to achieve the best system performance.

Because of its high flexibility and system performance, Reconfigurable Computing currently is widely used in many fields. With the development of reconfigurable hardware, such as FPGA, Dynamically Reconfigurable System has become a developing trend of embedded system. Dynamic Reconfigurable System is not only different from the traditional embedded systems in system structure, but also its patterns of the task performance have undergone a fundamental change. It can use the reconfigurable hardware to run the tasks in TDM way to speed up the system. Meanwhile, it brings some new problems, such as reconfiguration latency, the layout mode of reconfigurable hardware resources, how to prefetch the configuration files and so on. For the sake of the new features and problems of Dynamically Reconfigurable System, the traditional methods of HW/SW partitioning are no longer adaptable. Therefore, in this paper, based on the sufficient consider of the characteristic of dynamically reconfiguration and HW/SW partitioning, an Immune Algorithm of Alterable Coefficient of Parameters driven HW/SW partitioning method combined with scheduling algorithm is presented to try to partition the tasks of application. This algorithm is better than traditional HW/SW partitioning methods to adapt to the environment of Dynamic Reconfigurable System. It is more rational and efficient, and it can achieve the optimization of system performance.

Key Words: Dynamically Reconfigurable System, HW/SW Partitioning, Immune Algorithm, Task Scheduling

南京邮电大学学位论文原创性声明

本人声明所呈交的学位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的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：卢昭村 日期：2009.4.10

南京邮电大学学位论文使用授权声明

南京邮电大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其它复制手段保存论文。本文电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权南京邮电大学研究生部办理。

研究生签名：卢昭村 导师签名：[illegible] 日期：2009.4.10

第一章 引言

1.1 课题背景

1.1.1 嵌入式系统的特点

随着嵌入式技术的发展，嵌入式系统被广泛应用于国防电子、数字家庭、工业自动化、汽车电子、医学科技、消费电子、无线通讯、电力系统等国民经济的主要行业。嵌入式系统被定义为[1]：以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统，对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。一个嵌入式系统就是一个硬件和软件的集合体，它包括硬件和软件两部分。硬件包括嵌入式处理器、存储系统及外部接口。嵌入式软件主要包括两大部分，嵌入式操作系统和应用软件。相对于通用计算机系统，嵌入式系统具有许多自身的特点[1,2]，主要有以下几个方面：

(1) 专用性强：通常用在特定的领域，为特定的用户群设计。软件系统和硬件紧密结合，针对不同的应用常常要做很大的改动，甚至废弃整个系统并重新进行设计。

(2) 有实时性约束：嵌入式系统往往对时间的要求非常严格，能快速响应外部事件，常用在一些对实时性要求很高的场合，如果不能及时响应，系统将出现重大错误。虽然现在有些嵌入式系统的应用对实时性的要求不是很高，但实时性的要求也始终是衡量系统性能的一个重要指标。

(3) 成本要求严格：成本指的是系统成本，包括软硬件成本，设计开发成本和开发周期成本等多个方面，要求设计者在进行开发时必须综合考虑，在各种约束中找到满足系统需求描述的折中方案。

(4) 要求低功耗：由于许多嵌入式系统用在小型应用系统中，且要求长时间不间断工作，因此功耗也是衡量嵌入式系统性能的一项重要指标。功耗约束影响了系统设计决策的许多方面，包括处理器选择、内存体系结构的设计等，还有可能决定软件是用汇编语言编写还是C语言编写。

(5) 可靠性要求：由于一些嵌入式系统用于关键性产品，承担高危险及高保密要求的应用，所以可靠性的要求也是设计者必须考虑的因素。嵌入式CPU大多工作在为特定用户群设计的系统中，通常具有功耗低、体积小、集成度高等特点，要求嵌入式系统设计的小型化，高可靠性。

(6) 需要专门的开发工具和方法：由于嵌入式系统包括了硬件和软件，其开发的方法

也不同于普通的 PC 机。针对嵌入式系统已经出现了很多设计方法，如先硬件后软件、先软件后硬件和软硬件协同设计等。嵌入式调试已发展出了很多支持嵌入式系统开发过程的专用工具套件。

广泛的应用范围以及上述这些特点，使嵌入式系统的复杂度日益增加，设计的难度也越来越大。针对这些问题，前人做了很多探索，并提出了一些有效的解决方法，如在设计中使用软硬件协同设计，采用基于 FPGA 的可重构系统满足系统性能的各项要求等。

1.1.2 软硬件协同设计技术的发展状况

在传统的嵌入式系统设计方法中，将硬件和软件划分为两个独立的部分，先硬件后软件，分别按照各自的设计流程来完成。这导致了软硬件开发工作被割裂开，工程师之间缺乏有效的沟通。如果在软硬件集成时，系统实现不能满足开发性能指标，将需要对软件与硬件反复修改、反复试验。而硬件功能的调整会导致成本的增加和开发周期的延长。在实际设计中，这种开发方法不能对整个系统性能做出较好的优化，已经不能适应如今规模和复杂度都越来越大的应用。

针对传统设计方法缺陷，人们提出了软硬件协同设计的思想。软硬件协同设计方法学的研究始于 20 世纪 90 年代初期，第一届 International Workshop on Hardware/Software Co-design 会议于 1993 年召开，它标志着软硬件协同设计方法学的研究正式展开，软硬件协同设计领域正式确立。与传统的嵌入式系统设计方法不同，它把软件设计和硬件设计工作作为一个整体而不是把软件和硬件分开。软硬件协同设计强调软件和硬件设计开发的并行性和相互反馈，克服了传统方法中把软件和硬件分开设计所带来的种种弊端，协调软件和硬件之间的制约关系，为硬件和软件的协同描述、验证和综合提供一种集成环境，达到了系统高效工作的目的，是目前嵌入式系统设计中广泛采用的一种方法。软硬件协同设计的设计流程如图 1.1：

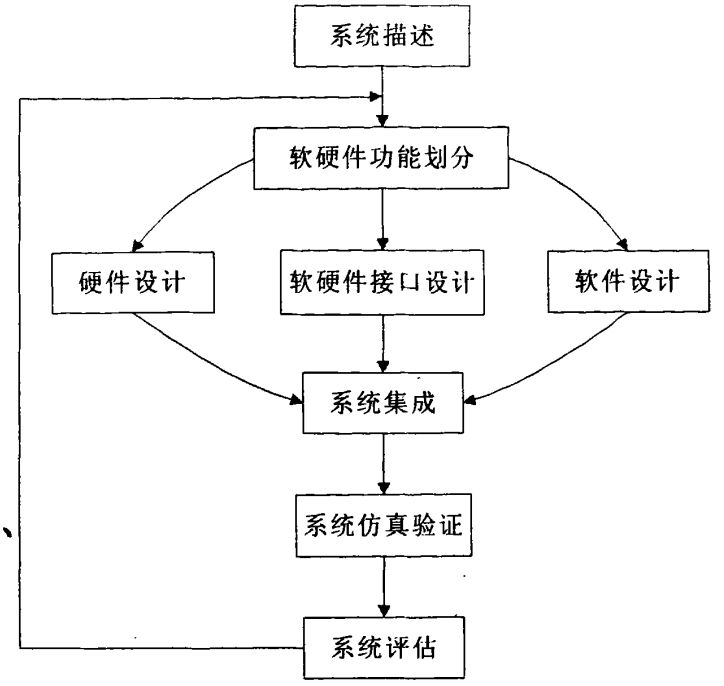

图 1.1 软硬件协同设计流程图

软硬件协同设计在软件及硬件实现之前，就从统一描述的系统功能开始，将硬件完成的功能作全盘考虑并均衡，在设计空间搜索技术的支持下，设计出不同的软硬件体系结构并进行评估，最终找到较理想的目标系统软硬件体系结构，然后使用软硬件划分技术进行划分，产生一个最佳的软件、硬件划分方案来满足性能、成本、功耗、开发周期等一系列设计约束限制。其核心问题是沟通软件设计和硬件设计，避免系统设计中相互依赖、互相影响的这两部分过早独立。在开发过程中强调硬件设计师和软件设计师相互协作，并行地设计系统的硬件和软件。其设计过程充分体现了软硬件的协同性，在功能划分时就对现有的软硬件资源进行充分的考虑，在软硬件功能的设计和仿真验证过程中，软件和硬件相互支持，这就使得软硬件功能模块能够在设计开发的早期互相结合协调，从而及早发现问题解决问题[3]，避免了开发后期对系统的反复修改。通过软硬件协同设计可以有效的缩短开发周期，降低成本，实现软硬件系统性能的优化，更好的满足设计要求。

软硬件协同设计具有以下几个特征[4,5]：

(1) 在系统层设计中，使用统一的描述和工具对软件和硬件进行集成开发，并具有跨越软硬件界面进行系统优化的能力；

(2) 提供软硬件协同仿真的能力，可进行全系统设计验证；

(3) 支持多领域专家进行合作开发与研究。

软硬件协同设计从软件和硬件设计工作紧密结合的需要中演化而来。目前软硬件协同

设计方法学研究还处于发展阶段，许多技术仍未成熟和实用化，是目前一个十分活跃的研究领域，加强对软硬件协同设计的研究具有重大的现实意义，将能够为嵌入式系统设计带来革命性的变化。

软硬件协同设计方法主要有下面几个关键技术[6,7]：

(1) 系统描述

包括功能和非功能性的需求描述。用一种或多种系统级描述语言对所要设计的嵌入式系统的功能和性能进行全面描述，建立系统软硬件模型。模型可以用非正式语言，或是自然语言来手工完成，也可以用 EDA 工具实现。手工描述不具体也不够准确，虽然简单但却会使设计复杂化，在设计时一定要考虑充分，而用 EDA 实现则会好一点。

(2) 软硬件划分

把系统划分成为软件模块和硬件模块。软硬件划分是协同设计方法中的一个关键组成部分。随着嵌入式系统的复杂程度和规模的不断扩大，手工的划分方法已不能胜任，为此人们提出了很多划分算法如 GA，SA，TS，PACE，Kernighan/Lin，MIBS，双向约束搜索法，分级群聚法，基于簇的规划法，整数规划法等[8,9,10,11]，以解决软硬件划分的问题。软硬件划分的内容在下面的章节中将具体展开叙述，它是开展动态可重构系统软硬件划分研究的重要基础。

(3) 软硬件综合

包括硬件综合，软件代码的生成和接口的生成。硬件综合由硬件高层综合、逻辑综合、版图综合等几个不同的阶段构成。现在，中低层的逻辑综合和版图综合已较成熟，硬件高层综合虽然技术成熟度不如前两者，但是目前也已经有成果投入使用。软件综合包括软件高级综合、编译、汇编等几个阶段即代码的生成。由于传统的软件综合技术尚不成熟，所以嵌入式系统的软件综合技术还处于发展阶段。但是嵌入式软件综合较之传统的软件综合有更多的限制，可以引入更多领域的知识，因此软件综合技术有着广大的发展前景。接口综合包括硬件构件添加锁存器、FIFO、地址译码器、为软件构件添加输入/输出、同步操作等。

(4) 软硬件协同仿真和验证

对系统的功能和限制进行协同仿真和验证。仿真和验证是检验系统设计正确性的过程，用来对设计结果的正确性和优化程度进行评估，以达到避免在系统实现过程中发现问题再进行反复修改的目的。软硬件协同仿真可以分为系统级协同仿真、行为级协同仿真、RTL(Register Transfer Level)级协同仿真和门级协同仿真。系统级协同仿真用于验证算法的正确性和评估系统的整体性能，它往往侧重于对总线操作进行模拟。门级协同仿真完整的

模拟软硬件实际运行过程，但是仿真的速度会随着设计规模的增大急剧下降。一般来说，抽象层次越高，反映的细节越少，模拟的速度越快。软硬件协同仿真通常是直接在模拟的硬件上运行软件，即微处理器通常与别的硬件在同一层次上被模拟。软硬件协同验证技术把软件和硬件的模拟联系起来协同进行，软件的集成与调试在设计循环初期通过一个虚拟样机进行，不需要等待硬件完成。应该注意的是，在系统仿真验证过程中，虽然模拟可以改进和加速设计工作，但是软硬件工作的环境很难和实际目标系统完全一样，软硬件交互的方式和效果也有很大不同，因此仿真验证的有效性是有限的。

1.1.3 软硬件划分

在整个软硬件协同设计流程中，软硬件划分算法是其中的一个关键技术。其主要任务是在满足各项设计约束指标的条件下，把系统功能划分到目标结构中的软件和硬件部分上，是确定系统的哪部分功能用软件实现，哪部分功能用硬件实现的过程，已成为软硬件协同设计的重要研究内容之一[12]。统计数据表明，包括软硬件划分在内的最初10%的系统级设计过程对于最终的设计质量和成本有80%的影响，因此系统设计对软硬件划分的质量要求十分严格。

软硬件划分主要包括以下三个方面，这三个方面既相互独立，又相互依赖：

(1) 处理单元的确定

确定要用到的软件和硬件处理单元的类型和数量,如CPU类型和FPGA的型号等，它也是设计约束之一；

(2) 任务分配

把系统中的任务分配到目标结构中的处理单元和资源上执行，并满足系统在性能和成本方面的约束。这是软硬件划分中最关键的一步，有了划分结果后才能对划分方案进行调度和评估。

(3) 任务调度

确定指派在各个处理单元上任务的执行顺序和开始时间，以满足系统任务之间的控制和数据依赖关系，并优化系统性能。任务调度不仅是评估软硬件划分方案优劣的必要步骤，也是进一步提升系统性能的有效手段。

软件实现的特点是灵活、成本低；而硬件实现的特点速度快，但是成本高。在满足系统设计要求的同时，如何兼顾系统的性能和成本，达到性能和成本的最佳结合，是软硬件划分要解决的问题。划分结果的好坏直接影响着系统的性能。从理论上来说，每一个应用

系统，都存在一个适合于该系统的硬件和软件功能的最佳组合。应该从应用系统的需求出发，依据一定的指导原则对硬件和软件的功能进行分析和合理的划分，从而使系统的整体性能、运行时间、功耗、存储容量等达到最佳状态。

传统的软硬件划分是手工进行的，随着嵌入式系统结构变的更加复杂，系统规模的不断扩大，开发时间要求更加紧迫，传统的方法已不再可行，单凭设计人员的经验进行划分很难保证满足设计约束指标，并得到最优或近似最优的结果。这种情况下，对系统级自动化设计工具的需求日益紧迫,近年来自动划分算法得到了长足的发展。软硬件划分问题的求解，就是在一定的系统模型基础上，对于解空间进行搜索的过程，为系统的每个功能模块(节点)选择一种合适的实现方式(软件或硬件)，在最大化系统性能的条件下，使系统成本达到最小。一个好的划分算法可以在较快的时间内得出较优的划分方案。一个优秀的划分方案必须考虑系统的整体性能、运行时间、功耗、存储容量等，属于多约束条件下的优化组合问题。

1.1.4 可重构计算的发展

可重构是在软件的控制下，利用可重用的资源重构或重组成另一个计算平台以适应不同的应用需求。可重构计算系统提供了一种介于通用计算机和专用计算系统之间的计算手段。通用计算机是面向通用应用领域进行设计的，其设计目标主要是能够灵活地处理不同的计算任务，这使得通用计算机具有很大的灵活性和应用范围，但对某些特殊的应用领域却难以取得高性能。专用集成电路是针对某种特殊应用设计的计算系统，对于特定的计算任务，专用计算系统表现出极高的性能，但无法处理特定计算任务之外的其它任务。通用计算机和专用计算系统代表了两种极端的计算手段，通用计算机具有最大的灵活性和低性能，专用计算系统具有最高的性能和最差的灵活性。而现在有许多应用需求既要求较高的性能，又需要一定的灵活性。这些应用需求导致了可重构计算的产生。可重构计算的主要目标是希望通过硬件可编程，来适应计算任务的需求，以期达到最佳性能。可重构计算的概念最早可以追溯到上世纪60年代。可重构计算大致经过了三个发展时期[13]：

(1) 早期的可编程模拟计算机，大量运算放大器、比较器、乘法器和无源元件通过一块块插线板和接插线连接起来，使用者可以实现一种网络，该网络的所有节点电压遵从一组微分方程。这样模拟计算机就变成了微分方程求解器，并具有可重构性。这是可重构计算系统的雏形。

(2) 可重构计算真正向灵活流畅迈出的第一步是嵌入式数字计算机的出现。系统特性由RAM中的软件来定义，实现起来非常简单方便。在安装时甚至是工作时，可以通过改变

系统的操作来响应数据改变,此过程实际上只是加载不同应用程序的过程。

(3) 基于静态存贮器(SRAM)的大规模现场可编程门阵列(FPGA)出现以后，才真正对可重构计算展开深入研究。研究人员发现通过对这种器件所拥有的逻辑单元与互连结构进行改变，可以创建出适合芯片要求的任意逻辑网表，很快选定FPGA实验时间配置的可重构性,创建用于特殊算法的硬线连接数字网络。

可重构的目的是为了解决硬件结构与应用的不匹配，可重构的基础是可重用资源。可重构计算具有以下一些优点[13,30]:

(1) 可重构计算能够有效的提高系统性能。可重构系统本质上是用硬件来实现系统功能，因此可以显著提高系统运行速度;

(2) 可重构计算技术能够通过动态改变可重构硬件的配置来灵活满足多种功能需求。可重构的特性使得同一可重构逻辑器件能够满足不同的设计需求，这一点是传统的专用集成电路计算模式不能够达到的。

(3) 可重构计算可以有效的加速产品开发。可重构器件厂商会配合不同的可重构逻辑器件提供相应的开发工具和流程，同时还会提供大量参考设计和IP核以减少设计者的重复劳动并提高设计的可靠性。

(4) 使用可重构计算技术能够极大的降低系统成本。在开发中利用可重构硬件比使用专用硬件电路更能节省成本。另外，通过在运行时使用时域划分的概念复用可重构硬件可以进一步降低系统的成本。

可重构系统就是指用可重构的硬件实现可重构计算的系统。由于可重构系统的这些优点，目前可重构技术被广泛的应用于高速数字滤波器、硬件演化计算、多媒体计算、图象处理、数据加密、容错系统等领域。动态可重构系统已经被广泛应用于实际生产和生活中，如何提高动态可重构系统性能已经成为了目前研究的一大热点。现在专门用于动态可重构系统任务划分的方法还不多，软硬件任务的划分问题又是影响整个系统性能的关键。所以改进优化任务划分的算法不仅可以使动态可重构技术不断发展和完善，更能使动态可重构计算系统的应用越来越广泛。

1.1.5 FPGA 技术的发展

随着微电子技术的发展，系统设计师希望专用集成电路(ASIC)的设计周期尽可能短，最好是在实验室里就能设计出合适的ASIC芯片，并且立即投入实际应用之中，因而出现了现场可编程逻辑器件(FPLD)，其中应用最广泛的当属现场可编程门阵列(FPGA)和复杂可

编程逻辑器件(CPLD)。现场可编程门阵列(FPGA)是在PAL、GAL、EPLD等可编程器件的基础上进一步发展的产物,它是作为ASIC领域中的一种半定制电路而出现的,既解决了定制电路的不足,又克服了原有可编程器件门电路数有限的缺点。

FPGA内部包括可配置逻辑模块(CLB,Configurable Logic Block)、输入输出模块(IOB,Input Output Block)和内部连线(Interconnect)三个部分[14,15]。当今的FPGA支持很多I/O标准,实现了I/O支持的灵活性。可配置逻辑块是FPGA内的基本逻辑单元。实际数量和特性会依器件的不同而不同,但是每个CLB都包含一个由4或6个输入、一些选型电路(多路复用器等)和触发器组成的可配置开关矩阵。开关矩阵是高度灵活的,可以进行配置以便处理组合逻辑、移位寄存器或RAM。CLB提供了逻辑性能,内部连线在CLB和I/O之间发送信号。目前有几种布线方法,从专门实现CLB互连到快速水平和垂直长线,再到实现时钟与其它全局信号的低歪斜发送的器件。随着技术的发展,目前已经可以使用设计软件自动进行内部连线,这样就极大地降低了设计复杂度。在系统存储方面,大多数FPGA均提供嵌入式Block RAM存储器,这可以在设计中实现片上存储器。对于高密度的器件,时钟分配是一个大问题。在系统级设计中,时钟脉冲相位差过大,就会限制系统的性能,在每个时钟周期内损失若干纳秒。新一代的FPGA中有独立的延时锁相环,允许内、外时钟进行同步来解决这个问题和消除系统时钟脉冲相位差。

FPGA的具体结构为CLB排成阵列位于芯片内部,在芯片四周是可编程的I/O,连接功能块和I/O的可编程互连线均匀分布于阵列的行与列之间。按照编程的方式和功能块的结构可分为SRAM查找表型和反熔丝多路开关型两类。前者在掉电后配置丢失,但可以实现系统内再编程、系统运行器件再编程和网络远程配置等特性,后者一次编程后不可再编程,掉电后配置不丢失。

FPGA具有编程方式简便、可靠性高、功能强大、开发便捷和开发周期短的优点。它的迅速发展为可重构计算技术的发展提供了必要的硬件支持。作为实现可重构计算技术的关键器件,FPGA在过去很长一段时间内存在计算密度低、配置时间长、封装复杂、高密度芯片的静态功耗和尺寸大等问题,在性能上不及ASIC,灵活性不如处理器。如今,FPGA经过十几年的发展,现场可编辑逻辑器件从最初的1200个可用门,发展到现在数百万可用门,并能嵌入处理器软核、乘法器以及大量的片上存储器,I/O管脚数也得到很大程度的增加。性能和灵活性的不断提升与价格和功耗的不断降低,已经使FPGA能够满足可重构计算对硬件的要求,这也是最近几年可重构计算得到长足发展的一个重要原因。

1.2 论文的组织结构

论文对可重构系统以及软硬件划分方法进行了研究，并在此基础上提出了一种针对可重构系统的软硬件划分算法，并进行了仿真实验与比较。论文中的各章节安排如下：

第一章：引言。介绍了嵌入式系统的特点，分析了软硬件协同设计技术目前的发展状况，软硬件划分的基本概念和要求等，最后介绍了可重构系统的发展状况以及论文的组织结构。

第二章：软硬件划分的相关技术。介绍软硬件划分中的一些基本概念，阐述了划分过程中的一些问题。

第三章：可重构嵌入式系统的研究。介绍可重构系统的关键技术，分析了影响系统性能的各个问题，以及应该在软硬件划分时需要注意的因素等。

第四章：基于权重动态可变免疫算法的软硬件划分算法和动态可重构系统调度。提出了针对动态可重构系统的软硬件划分算法和与评估划分结果相结合的任务调度方法，并用实验验证了算法的优越性。

第五章：总结与展望。主要对全文进行了总结，并提出了未来需要进一步深入研究的工作。

第二章 嵌入式系统软硬件划分技术

2.1 软硬件划分的概念、意义和基本原则

系统划分主要有两种方法：结构划分和功能划分[16]。结构划分先从结构上来实现目标系统，然后再对其进行划分，只能应用于硬件设计。功能划分先将系统功能分为若干个不可再分的块，再把这些功能块划分到系统组件上，通过软件或硬件来实现每个组件的功能。随着系统组件的数量越来越多，结构划分已不能适应实际的要求，因此功能划分更适合于用来划分复杂的目标系统，本文所指的软硬件划分即为软硬件功能的划分。

嵌入式系统中的功能任务模块大多都有两种基本的实现方法:软件实现和硬件实现。一般来说，硬件比软件能提供更好的性能，而软件更容易开发和修改，灵活性更强、成本比硬件更低。软硬件任务划分是将系统任务描述分解成为与实现有关的软件部分和硬件部分，确定系统的软硬件界面，并分配到相应的软件和硬件上，决定如何在软硬件各部件之间优化系统配置的过程。

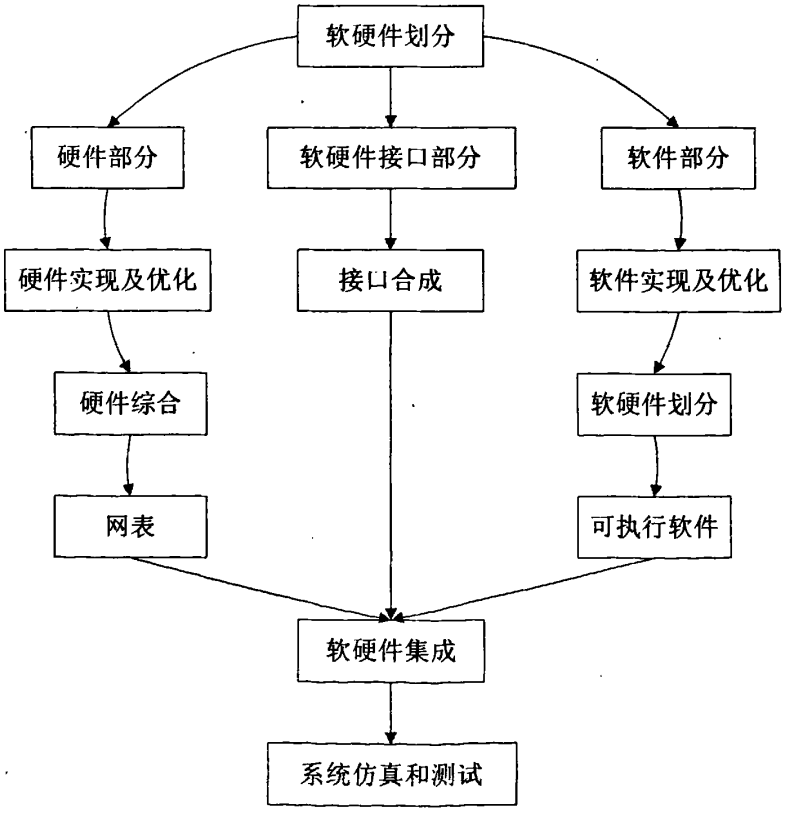

图 2.1 软硬件划分流程图

软硬件划分的基本流程如图 2.1 所示。嵌入式系统的硬件和软件之间有着相互联系、

相互影响和制约的密切关系。虽然很多时候，计算机中的大部分功能既可以用硬件实现，也可以用软件实现，软硬件之间的界限并不十分清晰，但是它们之间的性能差异很大。因而有的功能适合用硬件实现，另外一些则适合用软件实现。这就需要对目标系统进行分析，综合考虑各种功能性和非功能性约束，找到目标系统软硬件设计最佳平衡点。

在多数情况下，采用硬件特别是大规模集成电路，往往可以简化设计工作，通过操作的并行性和硬件本身较快的速度使系统性能得到显著提高，但使用硬件的成本较高，缺乏灵活性，而且硬件使用得越多开发风险越大，这是因为修改硬件较之修改软件的成本和时间都要大得多。用软件来实现部分硬件功能的优点是可以降低系统成本，减小体积和功耗，提高系统灵活性等，但缺点是运行速度较低，另外复杂的软件其开发和维护成本也相对较高。所以在嵌入式系统的设计中必须从系统的各项约束出发来对软硬件功能进行划分。基本原则是：对实时性有要求且成本不敏感的系统尽量用硬件实现；关注成本且产量较大的系统尽量通过软件实现；在软件实现无法满足系统性能要求时采用硬件实现；处理器和专用硬件或可重构硬件并用，可以同时提高处理速度和系统的灵活性。

2.2 软硬件划分的步骤和方法

软硬件划分是嵌入式系统软硬件协同设计的最重要的问题之一，一般有以下六个步骤[16]，划分结果的好坏将直接影响着后续的开发进程和整个系统的成本及性能，是协同设计中必须解决好的一个重要问题。

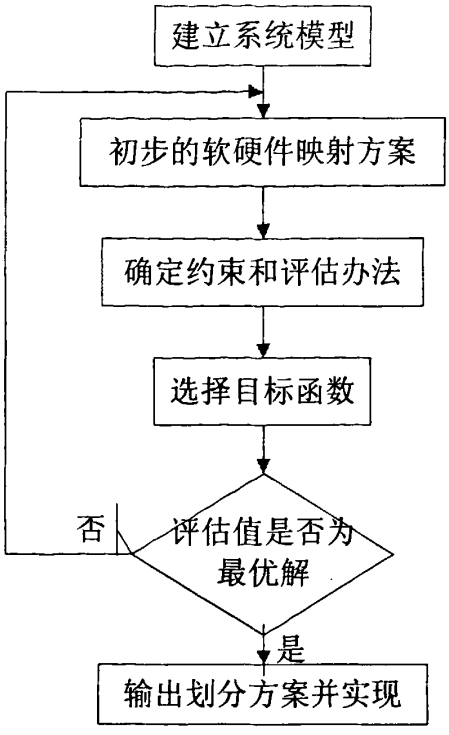

图 2.2 软硬件划分步骤

(1) 根据系统需求，建立一个可以用来表达整个系统的模型。该模型从用户的角度对系统进行描述，包括系统要实现的功能以及性能、功耗、成本等非功能性的指标，是系统整体的功能分配图，在这一步还没有确定系统功能的哪些部分由软件实现，哪些由硬件实现。

(2) 根据系统模型，确定目标系统架构，包括软硬件子系统的划分和通信机制，定义系统实现的目标平台的软硬件映射，如所使用的 CPU、内存容量、总线、操作系统等，得到初步的软硬件划分结果，但这还不是最终的输出。

(3) 根据系统需求，明确系统设计所必须满足的各项约束指标，确定对系统设计进行评估的办法和准则，为评价第二步得到的初步划分结果做准备。

(4) 根据系统各项约束和评估准则选择合适的目标函数。目标函数的选择对划分结果的评价具有极大的影响，因此在选择时必须综合各种因素进行考虑。

(5) 迭代求解，确定是否是最优或次优的划分方案。在这个过程中不断改进系统的软硬件划分，直到满足终止条件。也可以在过程中改变系统的评估标准、约束指标和目标函数，但一定要满足目标系统的需求。

(6) 根据输出的软硬件划分方案进行设计并实现。

按软硬件功能划分的方法有很多种，如一种常用的4层分割法。它按每项功能把一个系

统分成4个等级，从上到下依次是功能程序层、流程控制层、基本块层和指令层。对于一个结构和功能比较简单的系统来说，软硬件划分的方案数量较少，可以根据设计者的经验，手工把系统功能划分为软件实现或硬件实现。但是随着嵌入式系统规模的增大，软硬件模块的数量也随之增多，软硬件划分问题的求解规模也不断扩大，这时再仅仅依靠设计者的经验已经很难得到一个合理的软硬件划分方案。因此，研究自动的软硬件划分方法，在嵌入式软硬件协同设计中的重要性日益突出。Rolf Ernst[17]等人提出了面向软件的划分方法，主要思想是把系统所有可以用软件实现的功能都以软件方式实现，然后再逐步把操作从软件转移到硬件实现，直到满足目标系统要求。Gupta和De Micheli[18]在1993年提出了面向硬件的划分方法，思想是把系统的功能尽量都以硬件的方式实现，然后再逐步把操作从软件转移到硬件实现。Frank Vahid[19]等人1996年提出了一种适于对软硬件功能部件进行进程级描述的划分模型SLIF(Specification level intermediate format)。Ta-Chen Lin[20]等人1998年提出了基于缓冲区大小的划分方法。他们的划分算法考虑了以往算法所忽略的两个方面：数据依赖和系统延迟。X. Hu[21]在1999年提出了人工干预下的自动划分方法。这种划分方法的核心是设计空间的可视化(不是枚举所有可能的设计配置)。这种设计空间的表示方法为设计者提供了一个很直观的形式来标明那些划分更好。Jorg Henkel[22]于1999年提出一种面向低功耗的软硬件划分方法。M. Meerwein[23]于2000年提出基于IP重用的软硬件划分方法。

这些划分方法在其使用环境和目标系统应用领域内都取得了较好的试验结果，但它们都不是通用的自动化软硬件任务划分方法，只适用于某个具体领域和环境。因此，在目标系统做软硬件划分时，还应该根据系统的具体情况选择划分算法。本文就是在对动态可重构系统功能实现特点和应用环境研究的基础上，改进并优化免疫算法，并提出一种调度策略以更好更客观的评估算法的优劣，最后找到一种能够有效提高动态可重构系统性能的任务划分方法，可以很好地应用于嵌入式系统软硬件协同设计中。

2.3 软硬件划分中的问题

为了便于理解和对比各种不同的划分技术，可以将整个划分问题分为[24]：系统描述抽象级别、粒度、系统组件的分配、度量和评估、目标和接近函数、划分算法、输出，以及控制流程和设计者的参与等八个基本问题。

2.3.1 系统描述

软硬件划分是在系统描述基础上进行的，系统描述是软硬件划分六个步骤中的第一

步。在系统功能软硬件划分之前，嵌入式系统的功能描述是一种有利于划分的统一的中间表示格式。利用系统描述，软硬件功能的折衷及系统功能在软件与硬件之间的转换可以很方便地进行。系统描述的最终结果是生成软硬件划分所需要的节点相关信息，包括元件库、节点映射和设计约束等。各种嵌入式系统的结构和功能千差万别，要抽象出一个通用的系统描述模型，是非常困难也是不可能的。

在进行系统描述时，首先应给出系统所要实现功能的规格描述。建立的系统功能描述应该与该功能的实现方式无关，即不能偏向于特定的结构，也不能倾向于用软件或是硬件来实现。同时这种系统功能概念级的描述应是可执行的，以便对系统功能进行模拟验证。此描述根据系统需求和系统定义描述出系统要实现的功能以及性能、功耗、芯片面积等非功能性的指标。系统描述把系统功能分解成各个子功能，描述出实现这些功能的算法，然后建立起可执行的系统功能模型。通过系统功能模型的仿真，分析完善系统的功能设计。在对实际系统进行描述的时候，没有必要也不可能把所有的有关因素都考虑进去，应该根据问题的特征选择目标系统各项主要指标，力求在满足一定准确度的前提下，尽可能采用简洁的模型描述系统。

各种划分技术一般通过概念模型的抽象级别来定义划分的输入。比较粗略的可以将抽象级别定义为系统描述中低复杂度结构对象数量的度量，抽象级别越低，说明低复杂度结构对象越多。多种抽象级别的描述，由高到低大致有如下几种：任务数据流图、任务级、算法级数据流图、带数据通道的有限状态机、寄存器传输级、结构级等。不同的抽象级别代表了设计中不同的中间实现体。由于通常的设计都从较高的抽象级别开始，再往里加入结构细节信息形成较低的抽象层次。因此，划分输入的抽象级别的不同也能代表划分之间已完成设计的多少。

目前常用于描述系统的模型有控制数据流图(CDFG)，VHDL，无环数据流图(Acyclic DFG)，带数据通道的有限状态机(DFSM)等。其中CDFG是一种异构模型，结合了流程图CFG和数据流图DFG两种模型的优点，可以在一个表示中既可以显式表示数据相关性，又可以显式表示系统的控制顺序。由于嵌入式系统的实现流程是一个数据流的输入输出过程，整个系统可以分为一定粒度大小的多个任务来完成。系统的控制数据流图(CDFG)是一个有向无环图，它能够很好地表示这一过程，所以本文就以这种模型作系统描述。

2.3.2 划分粒度

划分的粒度也是软硬件划分中的一个重要问题。所谓粒度，就是每个对象中所含描述

规模的度量。粗度量意味着每个对象中含有大量描述，而细粒度则表示对象中仅含有少量描述。划分的粒度主要有任务(Task)，操作(Operation)，实体(Entity)，基本调度单元(Basic Scheduling Blocks)以及分层等。划分时可以使用固定的粒度也可以使用动态变化的粒度。显然，对输入分解的粒度越细，就能获得越多的对象和越多的划分方案，也就更有可能获得较优的结果。但是如果划分的粒度过小也会遇到一些问题。首先细粒度分解不可避免的会导致划分对象增多，划分算法将需要更多的时间进行计算，在广泛使用自动划分算法和计算机速度不断增加的今天，这已经不是很大的问题。其次，设计者不容易辨识细粒度对象，因此也就难于支持手动交互。再次，细粒度划分的结果对设计者来说不易于理解，对其进行手动设计或修改都较困难，对系统的评估也较困难。

2.3.3 系统组件分配

系统组件分配是软硬件划分步骤中的第二步。所谓系统组件分配就是从系统实现允许的软硬件组件中选择部分组件来构造系统，以满足目标系统需求的过程。可以用各种不同的组件分配组合来实现目标系统，各种组合之间的不同在于实现系统的性能和开销各不相同。分配可以手工或使用自动划分算法来完成，它必须指定功能对象分配到的系统组件类型。

2.3.4 划分结果的评估

要对不同的软硬件划分结果进行优劣比较，最直接的方法就是以某种方式将软硬件划分结果量化，然后通过量化结果的比较得出较优的划分结果，大多数现有的软硬件划分方法都是这么做的。因此，对划分结果做出尽可能精确的评估对于自动化软硬件任务划分方法取得成功至关重要。

在嵌入式系统中，主要的系统开销包括时间、成本和功耗等。时间指的是系统从开始运行到结束的时间，在可重构系统中包括任务的运行时间和可重构硬件的配置时间。系统成本由软件成本和硬件成本组成。软件成本主要指处理器和存储器成本以及软件开发、修改和维护所需的成本。硬件成本主要由一定工艺条件下的硬件面积决定，与占用的硬件面积成正比。需要注意的是在可重构系统中，当几个任务模块都使用可重构硬件实现时，在运行不发生冲突的情况下，他们可以共享可重构硬件。因此，把所有用硬件实现的任务所需实现面积之和作为系统总硬件面积可能会对硬件面积做出错误的判断，从而影响对软硬件划分结果的评估，这在设计可重构系统的划分算法时必须加以考虑。功耗包括静态功耗和

动态功耗两部分。静态功耗是指系统在空闲状态下的功耗，包括存储器功耗，总线和时钟功耗等。动态功耗是指系统在运行时产生的功耗，这个功耗包括处理单元执行任务时产生的功耗和通信通道的功耗，与处理单元的类型、运行的任务以及传输的数据量有关。一般来说，把使用硬件实现的任务转移到软件实现有助于减少系统功耗。

要对嵌入式系统的开销进行量化评估并不是很容易。这是由于所有开销都是根据实现功能对象的结构(或软件)来建立的，而在划分中这些实现并不存在。要计算软硬件划分结果的开销，有下面两种方法。

(1) 是根据软硬件划分的结果创建详细的实现，这种方法的优点是能得到精确的量化值，但是针对每一个划分结果都进行细节的实现将需要花费大量的时间和成本，在实际开发中是不可接受的。

(2) 创建粗略的实现，忽略对系统开销不大的一些细节。粗实现包含了设计中主要的寄存器传输级元件，但不涉及其中布线或逻辑优化等细节。通过这种粗实现来计算系统开销的量化值就叫做评估。评估的速度和精确度是互相矛盾的。为了达到较快的速度，必然会忽略粗实现中的一些细节；反之为了达到较高的精度则必须考虑这些细节。对于每种系统开销，都有不同的实现模型，在不同程度上对速度和精度进行折衷。

系统开销量化的本质是将需要同时进行硬件和软件等多目标优化的系统分配通过规定不同优化目标之间的相对关系将问题简化为单目标的优化问题。因为在软硬件系统协同设计时常常需要考虑多种不同的开销，如时间、成本和功耗等，而这些开销在系统实现中往往是相互矛盾的，为了定义划分结果的好坏，需要将他们合并到一个值，因此需要定义一个目标函数，目标函数应该返回一个自然数，以评价这个划分结果的质量。由于不同的目标系统所关注的各种开销的重要性不同，将多种开销组合成一个值也是一个重要的问题。多数方法是使用加权和目标函数，即用权重来表示各个度量的相对重要性，将度量值和其权重的乘积和作为目标函数的函数值。在本论文中也是采用这种加权的方法,具体的函数形式在后面的章节中给出。

2.3.5 软硬件划分算法

软硬件划分的主要工作就是在满足设计约束的条件下，为系统的每个功能任务确定它的实现方式，对受约束的系统开销进行优化，它实际上是一个组合优化的问题。另外，功能任务执行的软硬件分配和调度问题交织在一起互相影响，使问题的解空间非常巨大。为了达到系统成本的最优，确定系统中的每个节点到底采用哪一种实现方式，是一个全局优

化问题，从算法角度讲，也是一个NP完全问题。若节点数为N，每个节点有M种实现方案，则应用系统的划分方案就多达M^N种。

划分算法将每一个功能对象映射到一个组上，每组代表一个系统组件。从理论上来说，算法产生的划分结果能使目标函数获得最小成本。形式化的划分问题定义如下[24]：

定义：划分问题

给定一个目标集合$O=\{o_1,o_2,\cdots o_n\}$，求划分$P=\{p_1,p_2,\cdots p_m\}$，使其满足

$p_1 \cup p_2 \cup \cdots p_m = O$，$\forall i \neq j$，$p_i \cap p_j = \phi$，并使目标函数Object(P)成本最小。

划分算法通常可以分为两类，即构造性算法和迭代算法。构造性算法用全部对象分组来形成一个完全划分，并使用接近度量来指导对象的分组，从而获得较好的划分结果。构造性算法的计算时间主要用于构造这些数量较少的划分。迭代算法对已有的划分结果进行修改，并使用目标函数来对每个划分结果进行估计，产生较构造更为精确的估计值。迭代算法又可分为两大类:贪婪算法和爬山算法。贪婪算法只接受能降低成本的移动，其结果是可能会陷入局部最优。爬山算法通过使用随机函数允许向成本增加的方向移动，结果是能够避免局部最小化。各种迭代算法的不同之处在于它们修改划分的方式和它们接受或拒绝某个修改的方式。在实际使用中，经常需要将构造性算法和迭代算法结合起来使用。

在现有的软硬件划分算法中，每种算法都各有其自身特点，可应用于一定的目标系统软硬件划分中。但是由于可重构技术的发展和实用化程度越来越高，尤其是动态可重构系统的出现已经使嵌入式系统的结构发生了巨大的改变，任务执行的方式也随之改变，它使任务的执行更加高效，极大的提高整个系统的性能。而现有的软硬件划分方法大多没有考虑到动态可重构对整个应用执行方式的改变和对系统性能的影响以及可重构本身的特性。因此传统的软硬件划分方法已经不再适应。

2.3.6 输出

输出对应着软硬件划分步骤的第六步。每种划分技术都必须定义其输出的表示格式和可能应用。格式可以是一个列表，指示每个功能对象映射到了哪个系统组件上，也可以是另一种新的输入描述，含有系统组件的结构对象，并通过功能对象的映射关系来定义组件的功能。输出的可能应用描述了输出信息在后续设计过程中所扮演的角色。例如输出可以作为功能描述提供给设计者，来完成每个组件的具体实现，也可以作为综合工具的输入，甚至给综合工具提供启发式信息，来帮助综合工具选取信息中的有用部分。在本论文中输

出确定了功能任务对象到系统组件的软硬件映射。

2.3.7 控制流程和设计者的参与

软硬件划分的过程需要控制划分的顺序，其中某些划分步骤需要执行多次。划分算法是一个迭代求解的过程。

在实际划分过程中有时需要设计者的参与，设计者参与软硬件划分过程的方式有两种：第一种是指示，它是指设计者可以手工进行系统结构组件的选择、功能任务的映射或更改评估的标准等工作。第二种是反馈，是指当前设计提供给设计者的信息，比如一个图可以表示对象间的连线数目，直方图可以表示设计约束的不满足程度。

2.4 本章小结

软硬件划分是嵌入式软硬件协同设计的一个关键步骤。划分的结果直接影响着整个系统的性能和成本。本章首先介绍了软硬件划分问题的概念、划分的意义和基本原则。指出了系统任务分别用软件实现或硬件实现各自所具有的优缺点，为满足系统设计要求，需要在软件和硬件之间做出折衷。然后分析了软硬件划分的六个步骤和前人就划分问题所提出的一些有效的方法。最后叙述了划分中遇到的一些基本问题：系统描述、粒度、系统组件分配、划分结果的评估和目标函数、划分算法、输出以及控制流程和设计者的参与等。

第三章 可重构嵌入式系统

3.1 可重构系统的概念和发展

随着嵌入式系统技术的发展和应用范围的不断扩大，一些数据密集型任务如通信协议、安全和多媒体等对嵌入式系统性能的要求越来越高。而传统的以 CPU 为核心，以软件方式实现为主的嵌入式系统已不能满足目标系统的需求，为此后来提出 ASIC 的实现方法。这种方法虽然用硬件加速了任务的处理速度，提高了系统的并行性，但是 ASIC 的设计时间长、灵活性差、成本高，开发风险大，不利于应用产品的开发。在这种情况下，提出了可重构计算的概念，而 FPGA 的迅速发展也为可重构计算技术的发展提供了必要的硬件支持。可重构计算(Reconfigurable Computing) 技术是指在软件的控制下，利用系统中的可重用资源(如 FPGA 等可重构逻辑器件)，根据应用的需要重新构造一个新的计算平台，达到接近专用硬件设计的高性能[25,26]。可重构计算是一种介于 CPU 通用计算和 ASIC 专用计算系统之间的计算手段。它将系统需求中计算量大、有规则的数据访问模式、控制简单的那部分计算密集型任务放到可重构硬件中执行，而把那些计算量相对较少、数据访问模式比较随机、控制比较复杂的任务放到 CPU 中执行。用可重构硬件如 FPGA 实现的可重构计算即为可重构系统，与 CPU 和 ASIC 等传统的计算系统相比，可重构系统具有以下优点[27]，这些优点使可重构系统的应用更加具有针对性。

(1) 在性能上可以达到或接近定制的硬连线硬件，同时还具有很大的灵活性；

(2) 系统中硬件错误的更正和功能升级非常方便；

(3) 对于动态可重构系统可以在运行时随时改变系统内部结构和功能，也就是说动态可重构系统可以在运行时响应外部数据的变化；

(4) 可以使多种不同的功能共享同一个硬件平台，从而减少对系统硬件资源的需求，提高硬件的利用率。

可重构计算系统把通用计算系统和ASIC系统各自的优点结合在一起，折中了系统的性能和灵活性，在保持系统灵活性的基础上最大化系统运算速度，是目前嵌入式系统的发展趋势。可重构系统的发展大致可分为三个阶段[28]：

(1) 第一阶段：用执行串行操作的CPU和可以执行并行操作的商用化FPGA共同组成可重构系统。FPGA和CPU之间通常由外部总线连接，互连方式不灵活，总线容易形成通信瓶颈。系统的初始化配置要在系统启动之前完成，配置时间长，且大多数不支持动态部分重构。

(2) 第二阶段：把细粒度可编程逻辑和动态重构概念引入可重构系统，系统重构通过上下文切换或部分重构方式来实现。而且，以设备等效门数测量的芯片复杂度的提高使得可编程片上系统的实现成为可能，它将处理器、内存和可编程逻辑集成在一起，减小了微处理器和FPGA之间的通信瓶颈。

(3) 第三阶段：出现了具有动态部分重构和硬件虚拟化能力的面向数据流处理(多媒体)的系统。可重构系统逐步向需要复杂的计算操作和数据吞吐量的多媒体应用前进。还使用流水线技术有效地隐藏了系统的重构开销，使得可重构系统取得了更好的性能。

3.2 可重构系统的技术特征

3.2.1 系统结构

可重构系统的一个好处是可以提高目标系统性能，与传统的计算系统相比，可重构系统实现了真正的并发计算，使其在解决计算密集型应用时具有强大的优势。因为使用可重构硬件来实现系统功能，往往在几个时钟周期内就可以执行原来CPU要花费几百个周期才能执行完成的代码，有效的加快了系统执行速度。除了性能加速外，可重构系统还具有很大的灵活性，可以根据实际计算的需要设计定制计算逻辑，这种定制形式能够根据需要随时改变芯片上的计算逻辑。可重构系统还能显著降低目标系统的成本和功耗。

可重构系统一般由CPU(通用处理器)、存储器、可重构单元(RPU)、配置文件存储器等几部分组成。其中CPU负责用于控制和处理通用的计算任务以及控制可重构单元的配置调度，根据系统运行情况加载配置文件到RPU中；RPU负责执行专用领域且计算量大的任务，在某些系统中CPU与RPU集成在同一块芯片上，而在另外一些系统中，CPU与RPU分别位于不同的芯片上；配置文件存储器用来保存可重构单元的配置数据。系统结构如图3.1：

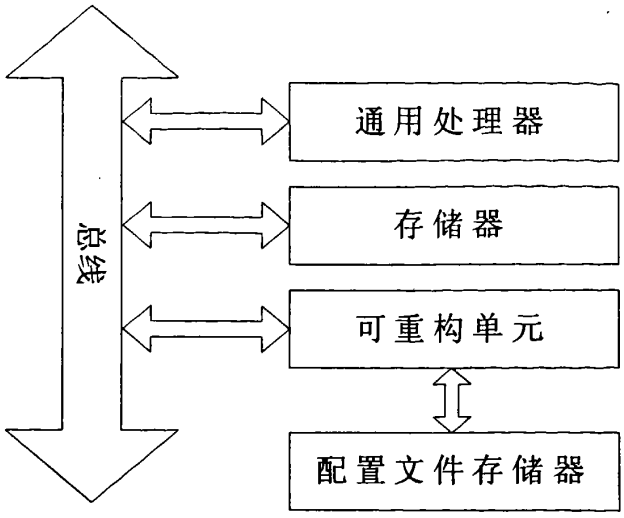

图3.1 可重构系统结构图

其中可重构单元和通用处理器的耦合方式，可以分为以下的四种：

(1) 可重构单元通过接I/O接口和通用处理器相连，作为一个单独的处理单元存在，这是最松散的连接方式。

(2) 作为通用处理器附加的处理单元，和主处理器之间通过内存总线和标准的通信原语支持。

(3) 作为协处理器，可以执行较大粒度的运算，通用处理器较少干预协处理器，通常只为协处理器准备好数据和环境，协处理器完成计算后把结果返回给通用处理器。

(4) 把可重构单元嵌入通用处理器中作为一个功能可变的处理单元，提供功能可以定制的指令，通用处理器的流水线和寄存器对可重构单元都可见，这是最紧密的一种耦合方式。

3.2.2 可重构系统的分类

按照不同的分类标准，可以将可重构系统分为不同的类型。一般的分类标准有划分粒度、系统重构的方式等。其中系统重构方式又包括：静态和动态重构方式、可重构单元的配置方式两种划分方法[28, 32,33,41,42]。

(1) 按粒度划分

粒度是指系统中可重构单元的操作数的宽度，说明了可重构硬件中最小处理单元的复杂程度。可重构系统按粒度划分可分为细粒度可重构系统和粗粒度可重构系统两大类。在一个细粒度的可重构体系结构中，可重构单元通常由逻辑门、触发器、较小的宏单元和查找表等小逻辑单元组成，它们进行位级操作，实现一个有限状态机的布尔函数。常用在数

据宽度可变的计算结构中，缺点是系统重构时间长。粗粒度可重构系统中的可重构单元可能包括完整的功能，如算术逻辑单元、ALU、乘法器等，它们以字宽为单位进行操作。粗粒度可重构系统较之细粒度可重构系统功能更强，速度也更快，但缺点是硬件面积利用率较低。如果一个可重构系统结构中包括粗粒度和细粒度两种可重构单元，则称其为混合粒度的可重构系统。

(2) 静态重构和动态重构

在处理不同应用的过程中，可重构单元可能需要被频繁地重构。重构实际上就是重新装载配置文件的过程。如果装载配置文件的过程必须在系统运行之前完成，在系统运行过程中配置保持不变,配置文件的更换必须在中断程序执行的情况下进行，那么这种重构方式称为静态重构；如果装载配置文件的过程可以在系统运行过程中和程序的执行同时进行，动态的改变可重构单元的配置，那么这种重构方式称为动态重构。由于在应用中，动态重构比静态重构具有更高的灵活性，因此也是目前可重构技术研究的一个主要方向。

(3) 按配置方式划分

1. 单上下文模式

在单上下文模式中，每次加载配置信息都需要对整个可重构硬件进行重写。这样做虽然结构简单，但是即使只修改部分配置信息也必须重写整个可重构单元，重构时间开销大，严重影响了系统的性能。单上下文的可重构系统常常也具有静态重构的特点。

2. 多上下文模式

在多上下文模式中，可重构单元有多套编程点用来存储多个配置文件,可以事先将配置文件下载到编程点中。系统的有效编程点内容决定了可重构单元完成的功能。通过有效编程点的切换即可切换配置信息，从而实现可重构单元功能的切换。与单上下文模型相比，多上下文的可重构单元的优点在于可以更快地切换配置文件，另外，处于不活动状态的上下文可以在系统运行时进行重写而不影响可重构单元的执行，从而隐藏了重构开销，极大地提高了系统的性能。这种模式虽然减少了重构时间，但由于每个上下文存储的都是整个芯片的配置信息，编程点会占用很大的面积,故采用多上下文模式的设计方法会导致芯片面积的迅速增加。

3. 部分重构模式

部分重构系统的可重构单元包含很多可以独立配置的区域(DRU),在系统运行过程中,能够有选择地重写可重构单元上指定区域的逻辑资源，而不是重写整个可重构单元，没有被重写的区域不受影响，整个系统的运行也不会中断[29]。目前,只有Xilinx、Atmel等公司生产的FPGA支持部分可重构。这种重构模式减小了每次配置时被重写的资源范围和数目，

实现了硬件资源的时分复用，能够更加有效地利用逻辑资源，是目前可重构系统发展的方向。对于多配置文件的可重构系统，一般都具有动态重构的特性。本论文所讨论的可重构系统就是具有动态部分重构特性的系统。

3.2.3 重构时间

可重构系统运行时间可以分为任务计算时间和重构时间，任务计算时间是真正有效的运行时间。重构时间是指功能切换时对可重构硬件进行配置重写带来的延时。可重构系统的特点是对可重构硬件进行时分复用，在重构时把原有的配置文件重写为系统功能要求的新配置文件。新配置文件一般存放在配置文件存储器，通过可重构单元的配置接口加载到硬件资源中。在重写时，加载原配置文件的可重构资源区域必须停止执行，然后再把新配置文件写入可重构资源。从停止执行开始到新配置文件重写完成的这段时间，系统的这部分可重构资源处于无任务运行的状态，这段时间就是系统的重构时间开销。

虽然可重构系统能极大的提高系统运行的速度，但是重写配置的时间过长已经成为了制约可重构系统发展的瓶颈。例如，UCLA的ATR[30]系统用在可重构硬件的重构和等待重构上的时间达到了总运行时间的75%，RRANN[31]系统花费的重构时间大约占整个任务执行时间的80%。随着可重构硬件容量的不断增加，重构时间将越来越长，对系统性能的影响也会越来越大。解决好这个问题可以使可重构技术更具有实用性，基于可重构系统平台的应用范围也会越来越广。目前缩短系统重构时间的方法主要有：

(1) 配置流水线

在动态部分可重构系统中，引入流水线的概念，把一个可重构单元区域的重构与另一个可重构单元区域的执行重叠起来，以最大限度的减少重构的时间。这种方法的缺点是需要可重构硬件的支持。

(2) 压缩配置文件

减少重构时传输的数据量也是有效减少重构时间的方法，目前大部分FPGA都已能支持这一特性。

(3) 提高配置时的数据传输率

显然，在配置文件大小一定时，较高的数据传输率将减少重构的时间开销。这也是目前降低重构开销的研究重点之一，现在大部分的FPGA都同时支持串行和并行传输。

(4) 数据差量配置

可重构单元在重构时需要用新配置文件重写原配置文件，数据差量配置即在重构时只

重写新旧配置文件之间不同的部分，这样做的好处是传输的配置数据量和重写的时间都将减少。如果新旧两个配置文件之间的差异较小，使用差量配置的方法能极大的缩短重构的时间。缺点是需要可重构硬件的支持，目前还不是很成熟。

(5) 配置预读取

在特定的功能任务被调度前，先按照某种策略把该任务的配置文件写入可重构单元，当该任务被CPU调度后可以立即开始执行，不用再等待配置文件加载。这种方法是目前研究提高重构效率的主要方法之一，常常和流水线技术一起使用。

(6) 可重用配置的优化

在可重构单元完成计算任务后，不替换已写好的配置文件，等待对该任务的再次调度。对于周期执行的功能任务，用这种方法实现能显著的缩短重构时间。但是当可重构资源不够用时，需要某种调度策略来实现资源替换。

(7) 配置数据共享

先从不同的配置文件中提取出相同的信息，再把这部分信息配置到可重构单元中，从而实现在配置替换时，这部分信息不需要再次重写，可以减小每次重构时重写硬件的面积。缺点是信息的提取没有规律可循，实现起来不容易。

(8) 配置克隆

在可重构单元内部复制那些有规律的配置信息，但是这种方法在实际系统中并不实用。

3.2.4 可重构单元的资源模型

可重构系统的资源模型[32]是指可重构芯片表面计算资源的抽象和运行时对硬件任务所占资源的约束。可重构单元(以FPGA为主)通常是矩形逻辑阵列，功能任务逻辑也被假设为一个矩形区域，在运行过程中进行模块布局时，选择可重构单元中一个矩形子区域作为新功能任务的映射区域。

对于非部分重构系统，任何时候都只有一个功能任务的配置文件处于运行状态，该配置文件占据了可重构单元的全部资源。对于部分重构系统通常有一维和二维两种资源模型[32]：

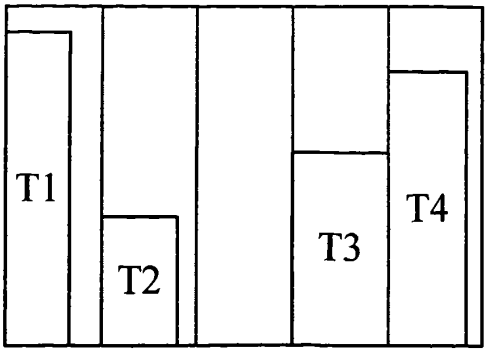

图3.2一维等宽模型

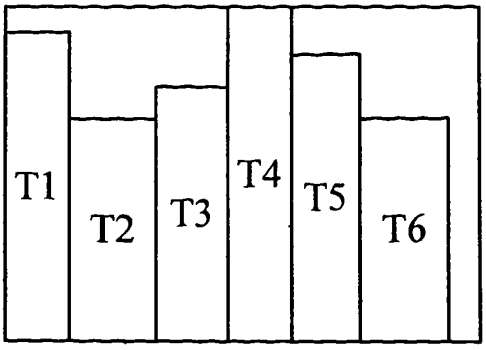

图3.3一维变宽模型

(1) 一维模型。任务映射到可重构单元(FPGA)上时，可被布局在水平方向上的任何位置，但垂直方向不允许任务重叠，占用可重构单元的整列。高度即为FPGA的资源高度(即使任务矩形高度达不到资源矩形高度也将被占用)，宽度可分为等宽(如图3.2)和变宽(如图3.3)两种，图中T1至T6为映射到可重构单元中执行的任务。一维模型配置方法简单，实现时的附加开销也较小，但是缺点是如果配置的任务较小，即任务占用的逻辑矩形高度较小时，将会浪费掉一些可重构资源，造成资源利用率低下。

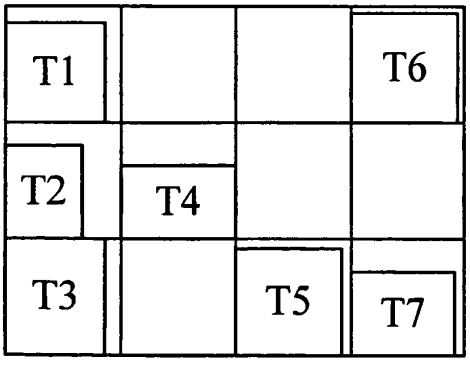

图3.4 重构区域固定的二维模型

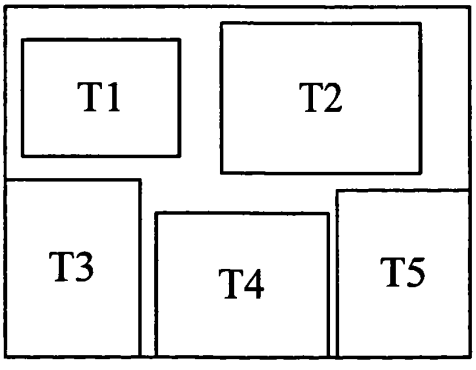

图3.5 重构区域可变的二维模型

(2) 二维模型。允许任务以面积的方式映射到可重构单元的资源中，在水平和垂直两个方向上都没有限制。分为区域尺寸固定(如图3.4)和区域尺寸可变(如图3.5)两种，图中的T1、T2、T3等为映射到可重构单元中执行的任务。相对于一维模型，二维模型能够更有效的利用可重构资源，但是目前技术还不是很成熟，难以应用在实际系统中，而一维模型已经出现了很多实用的商用系统。所以本文所讨论的动态可重构系统也是针对基于一维资源布局模型的系统。

3.2.5 可重构系统的其他问题

除了本章上述的可重构系统特性之外，还有其他的一些相关问题[40]，它们在可重构系

统的设计和应用中也有相当重要的地位，处理好这些问题对于可重构系统的实际应用和发展都具有非常重大的现实意义。

(1) 可重构系统的操作系统。任务分配到CPU或可重配置单元上后，需要由操作系统管理各任务的运行，且可重构系统的结构种类比较多，资源的管理比较复杂。但是现有的操作系统都是针对软件任务的管理进行设计的,没有考虑可重构系统的各种新特性，因此把原有的操作系统照搬到可重构系统中的方法不可行，必须根据可重构系统的特点扩展原有操作系统的功能，使之能够适用于可重构系统中的任务调度和资源管理。

为了使对硬件任务的管理像软件任务一样，必须先对硬件任务的接口进行定义，使它能响应操作系统和用户的调度，接口定义好之后就可以对它进行管理了。在使用时操作系统还需要对配置文件的加载和资源布局模型进行管理。另外，还要注意硬件任务不像软件任务一样在执行时要独占CPU，它们可以并行的执行。对于动态可重构系统，操作系统还要考虑配置流水线和配置预读取等技术以加快可重构单元的重构速度。这些因素在设计用于可重构系统的操作系统时都必须加以考虑，这是一个非常复杂且困难的问题。目前的可重构操作系统还不成熟，有待于进一步的研究。

(2) 编译工具。由于可重构系统的新特性，原来的编译器已不再适用，设计可重构系统的编译工具也是一个很有挑战性的问题。

3.3 可重构系统的软硬件任务划分和调度

嵌入式系统在引入硬件重构的概念后具有了时分复用硬件设备的能力，极大的提高了系统的速度和灵活性。采用可重构技术提高系统性能的同时也随之产生了很多新的特性，这些特性在进行可重构目标系统设计时如果不加以考虑将会极大的影响系统性能，尤其是重构时间过长已经成为系统性能的瓶颈，这些问题处理不好时甚至会使系统性能低于传统的计算系统[33,34]。从上一章的分析中可知软硬件划分在嵌入式协同设计中占用相当重要的作用，是整个设计流程中关键性的一步。而在可重构系统尤其是动态可重构系统设计中引入协同设计思想和其中的软硬件划分方法，将有利于进一步降低系统重构的时间开销和解决其他的可重构设计问题，有效的提高可重构系统性能[35]。

目前，对可重构系统软硬件任务划分算法的研究才刚起步，成果不多，技术还不成熟。总的来说可重系统软硬件划分的问题包括以下两个方面：

(1) 任务分配问题。在可重构系统上运行的功能任务是分配到哪类器件上，以何种方式实现。是分配到CPU以软件方式实现，还是以专用电路方式实现或者分配到可重构单元

以时分复用硬件资源的方式实现。这是软硬件划分问题的核心和主要矛盾，在划分时就应该从可重构系统的技术特征出发，充分考虑各种因素对系统性能的影响，使划分算法更贴近实际目标系统。

(2) 调度和资源模型。任务软硬件映射完成后的调度问题包括对软硬件功能任务的调度和对预读取配置文件的调度两个方面(调度模型如图3.6所示)。资源模型即可重构任务在可重构单元中的布局问题。与传统的嵌入式软硬件划分问题不同，由于采用可重构系统产生了很多新的特性，尤其是重构时间在系统设计时更是不可忽略，在对任务分配结果进行评估时必须考虑这些因素，这样得到的结果才能更客观。较好的任务调度可以有效的隐藏部分重构时间，资源模型则决定了某个任务分配到可重构单元是否可行。因此在任务评估阶段需要分析调度策略和可重构单元的资源模型，它们在很大程度上决定着系统的性能和可行性。

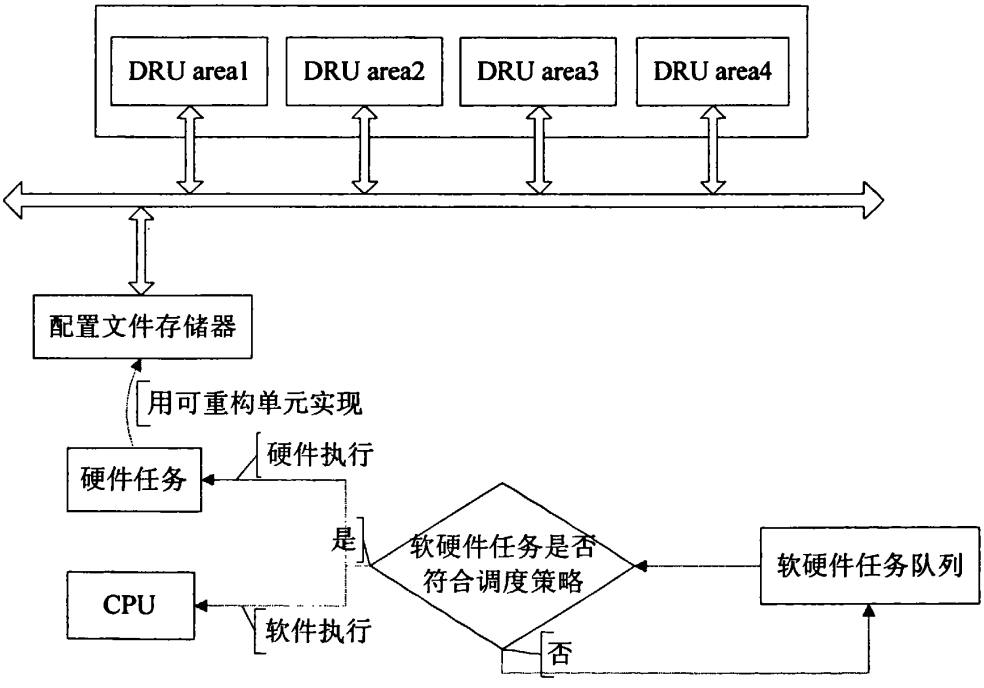

图3.6 可重构系统调度模型

3.4 本章小结

本章首先介绍了可重构系统的概念和发展，可重构系统是提高嵌入式系统性能的一种有效手段，已经成为了嵌入式系统发展的趋势和研究热点。然后详细说明了可重构系统的各种技术特征：它的系统结构，按粒度和重构方式的分类，重构时间开销，资源模型和其

他的一些问题。本文所讨论的可重构系统是一种动态的、具有部分重构特性的可重构系统，它也是可重构系统的发展方向。最后，分析了对可重构系统进行软硬件任务划分的意义和需要重点解决两大问题。

第四章 动态可重构系统的软硬件任务划分

4.1 基于权重动态可变的免疫算法的软硬件划分

动态可重构系统任务的划分就是将系统模型分解成为一系列相关的子任务，再映射到软件执行单元、非可重构硬件单元或可重构单元(FPGA)上，通过优化系统配置来达到最佳的系统综合性能的设计目标。本质上，动态可重构任务的划分算法，就是在一定的系统模型之上，对解空间进行搜索，并最终确定应用的每个任务节点采用哪种实现方式。这是一个全局优化的问题。优化技术是一门以数学为基础，用于求解各种工程问题优化解的应用技术。作为一个重要的科学分支，它一直受到人们的广泛重视，并在诸多工程领域得到迅速推广和应用，如系统控制、人工智能、模式识别、生产调度、VLSI技术、计算机工程等。实现生产过程的最优化，对提高生产效率与效益、节省资源具有重要的作用。因此，优化算法的研究是一个同时具有理论意义和实用价值的重要课题，寻求一种适合于大规模问题并具有自组织、自适应、自学习能力的算法一直是人们研究的目标。所谓优化算法[36]，就是一种搜索过程或规则，它基于某种思想和机制，通过一定的途径或过程来得到满足用户要求的问题的解。

4.1.1 免疫算法中的免疫概念

在人们探索通用优化算法的过程中，大自然尤其是生命科学一直是人们获得灵感的源泉。从上世纪五六十年代开始人类就对生命科学中的各种概念和生理功能进行模仿，借鉴其相关的内容和知识，特别是遗传学方面的理论，提出了很多性能优异的算法，并将其成功应用于工程科学的一些领域，收到了良好的效果。美国MICHIGAN大学的Holland教授于1975年受生物进化论的启发，在对以前的学者提出的遗传概念进行总结的基础上提出了遗传算法(genetic algorithms，GA)。由于遗传算法较之原有的传统算法具有使用方便、鲁棒性强、易于并行处理等优点，因此广泛应用于机器学习、模式识别、组合优化、图像处理、神经网络等领域。

遗传算法的主要任务是设法产出或有助于产生优良的个体成员，且这些成员能够充分表现出解空间中的解，从而使算法效率提高并避免早熟收敛[36]。遗传算法是一种并行、随机和自适应的优化算法，可在全局解空间进行搜索。算法在一定条件下具有全局收敛性，但是在算法实际使用中，还存在许多问题有待进一步研究解决。如在保证种群进化性的基

础上，如何避免退化现象的出现；单调和单峰函数如何在接近最优值时加快收敛等。这些问题主要是由于交叉、变异和选择等操作都是在一定发生概率条件下随机的、没有选择的进行。另外，在实际的应用中常常可以从问题的特征中提取出一些基本的、可对问题的求解起到辅助作用的有益信息，这些信息对加速算法收敛于最优解具有重要的指导作用，但是遗传算法的交叉、变异和选择等操作灵活性较低，难以将提取出的信息用于问题的求解。因此，还需要对遗传算法继续改进以加快寻优速度和改善寻优质量。

免疫算法是一种模仿生物免疫系统的进化算法，是对生物自然科学免疫理论的一种抽象，与遗传算法比起来更能深入的挖掘和利用生命科学的理论和特征。免疫算法将免疫的概念和理论应用于遗传算法，根据引入到遗传算法中免疫概念的不同又可分为[40]：免疫抗体自我调节算法、免疫响应过程算法、免疫疫苗接种算法和免疫记忆算法等几种不同的类型。本文所讨论的免疫算法是指其中的免疫疫苗接种算法，这种算法在保留原算法优良特性的前提下，有选择的利用一些特征信息和设计经验来避免解空间搜索的盲目性，有指导的收敛于最优解，从而抑制原算法中的退化现象使问题求解更具有针对性，显著的提高了收敛速度[36]。这种算法非常适合用于动态可重构系统软硬件划分问题的求解，因为动态可重构系统中有大量的基本且显而易见的信息，如应用中的某些任务用可重构单元执行加速效果明显且占用FPGA资源面积小、一些周期性的硬件任务在需要进行配置文件替换时可少替换甚至不替换等等。

免疫是机体的一种特异性生理反应，通过识别和排除抗原性异物维持机体内环境的稳定[37]。机体的免疫功能是在淋巴细胞、单核细胞和其它有关细胞及其产物相互作用下完成的。免疫系统是机体执行免疫功能的机构，是产生免疫应答的物质基础。免疫细胞具有特异性抗原受体，接受抗原刺激后能活化、增殖和分化，产生特异性免疫应答。随着生物学和医学的发展，生物免疫学的研究也得到了极大的发展，同时在对该学科各方面所表现出的一些现象不断深入认识与理解的基础上，逐渐的把免疫学中的一些原理和概念引入到其他的学科，促进并拓宽了各门学科的发展方向。尤其是在与计算机科学的交叉研究中，通过计算机对免疫系统及其各种机体功能与特征行为进行数学建模，更易于分析和解决各种目标系统的特征及关键问题。在目前的研究中，免疫系统的许多功能特点和作用机理对于工程应用中许多复杂问题的求解都起到了重要的启示和借鉴作用。

为了把免疫的概念引入优化算法，首先需要理解免疫系统的一些基本概念和技术术语。简单的介绍如下[37]：

(1) 抗原(Antigen)

抗原是指一种能刺激人或动物机体产生抗体或致敏淋巴细胞，并能与这些产物在

体内或体外发生特异性反应的物质。抗原的基本能力是免疫原性和反应原性。免疫原性又称为抗原性，是指能够刺激机体形成特异抗体或致敏淋巴细胞的能力。反应原性是指能与由它刺激所产生的抗体或致敏淋巴细胞发生特异性反应。应用到优化算法中的抗原是指所要求解的具体问题。

(2) 疫苗(Vaccine)

疫苗是指为了预防、控制传染病的发生、流行，用于人体预防接种的疫苗类预防性生物制品，包括微生物或其毒素、酶，人或动物的血清、诊断和治疗用的制剂等。在优化算法中，疫苗是指从具体问题中分析提取出的特征信息，这些信息将有助于算法快速的收敛于问题的最优解。

(3) 抗体(Antibody)

机体是指在抗原物质刺激下，由B细胞分化成的浆细胞所产生的、可与相应抗原发生特异性结合反应的免疫球蛋白。对应于优化算法中的抗体是指对疫苗进行处理，转化为求解问题的一种方案后，由该方案得到的所要解集合。

(4) 特异性免疫(Specific Immunity)

特异性免疫又称获得性免疫或适应性免疫，这种免疫只针对一种病原。是获得免疫经后天感染(病愈或无症状的感染)或人工预防接种(菌苗、疫苗、类毒素、免疫球蛋白等)而使机体获得抵抗感染能力。一般是在微生物等抗原物质刺激后才形成的(免疫球蛋白、免疫淋巴细胞)，并能与该抗原起特异性反应。对应于优化算法中的目标免疫(Target Immunity)，是指在个体在进行了遗传操作后，经过一定的判断，个体仅在作用点处发生免疫反应的类型。

(5) 非特异性免疫(Nonspecific Immunity)

非特异性免疫又称天然免疫或固有免疫。它和特异性免疫一样都是人类在漫长进化过程中获得的一种遗传特性，但是非特异性免疫是人一生下来就具有，而特异性免疫需要经历一个过程才能获得。对应于优化算法中的全免疫(Full Immunity)，是指种群中每个个体在交叉、变异操作后，对其每一环节进行一次免疫操作的类型。

(6) 染色体(Chromosome)

染色体是由线性双链DNA分子同蛋白质形成的复合物，真核生物的核基因就分藏在每条染色体中，所以，染色体是基因的载体，也就是遗传信息的载体。在动态可重构系统软硬件任务划分算法中，染色体以编码的形式代表了问题的一个解，染色体的长度即为任务数，其上的基因代表了每一个节点软硬件任务的一次功能分配。

软硬件任务划分的过程即是使用基于疫苗接种的免疫算法求解出问题最优解的过程。

在进化选择中，通过提取疫苗、接种疫苗和免疫选择来指导搜索过程，从而获得更好的优化性能和更快的收敛速度。

提取疫苗是指通过具体问题具体分析来得到一些基本的、显而易见的特征信息或先验知识。先验知识可以是最优个体某些基因的取值范围，也可以是一些基因之间的制约关系。疫苗不是一个成熟或完整的个体，它仅具备最佳个体局部基因位上的可能特征。在选取疫苗时，既可以根据问题的特征信息来制作免疫疫苗，也可以在具体分析的基础上考虑降低原问题的规模，增设一些局部条件来简化问题，用简化后的问题求解规律作为选取疫苗的一种途径。疫苗的正确选择将会对群体的进化产生有益的推动作用，在本文中即指从动态可重构系统软硬件任务划分(抗原)的问题中提取出有用的特征信息(疫苗)。

接种疫苗[36]是指对于个体x，按照先验知识来修改其某些基因位上的基因，使所得个体以较大的概率具有更高的适应度，疫苗所包含的信息量和正确性对算法的性能起着重要的作用。需要注意的是，动态可重构系统中有很多基本的、可加以利用的特征信息，因此可以提取出的疫苗可能有很多个，在接种时可以选取其中最具代表性的一个进行注射，也可以将几种疫苗组合后同时注射。在注射时，从种群$T(x_1,...,x_n)$中，随机的按接种概率$\alpha(1\le\alpha\le n)$取$n_\alpha=\alpha n$个个体进行注射。

免疫选择[36]是指对接种疫苗之后的个体进行检测。若其适应度不如父代，说明在交叉、变异和接种疫苗的过程中出现了退化现象，这时个体将被舍弃并由其父代中所对应的个体替代。如果子代的适应度优于父代，采用退火选择决定哪些个体进入下一代种群，即在当前的子代种群中以概率$P(x_i)=e^{f(x_i)/T_k}\Big/\sum_{i=1}^{n}e^{f(x_i)/T_k}$选择个体$x_i$进入下一代，其父代中相对应的个体将被舍弃，式中$f(x_i)$为$x_i$的适应度，$\{T_k\}$是趋近于0的温度控制序列。

本章在分析免疫算法原理的基础上，把它引入到动态可重构系统的软硬件划分中，并根据动态可重构的特点对划分算法加以改进，充分考虑了对系统性能影响极大的重构时间问题，还有目前能有效减少重构时间的两项技术：流水线和预读取技术对系统性能的影响，以及可重用配置优化的问题。这些因素在软硬件划分中应该加以考虑，否则将会使划分的结果脱离实际系统，得不到划分的最佳方案。

4.1.2 系统模型描述

由于动态可重构系统具有在线实时重构的能力，且能够通过部分重构特性缩短重构时

间和实现硬件的时分复用，这些特点使它同时具备了运算的高速性和功能的灵活性。软硬件划分就是为系统的每个功能任务确定它的实现方式，对于传统的嵌入式系统，功能任务的实现方式不外乎两种，即以CPU为核心的软件实现和以ASIC为核心的硬件实现。可重构系统尤其是动态可重构系统出现后，任务的实现方式发生了巨大的变化，除了传统的两种方式外还可以用可重构硬件完成实现方式的动态改变。在系统设计中，如何在充分发挥动态可重构系统优势的基础上对软硬件任务进行划分是本文研究的主要内容。

根据嵌入式软硬件划分的步骤，首先将应用划分为一定粒度大小的符合系统约束条件的任务，然后用带权的控制数据流图(CDFG)来描述系统。控制数据流图可以看成是由控制流图和数据流图组合而成，可以粗略地分为两个部分[38]——控制部分和数据部分。对于控制部分，主要反映事件及交互关系、响应时间、事件或处理的优先级。对于数据部分，可以看成运算或处理实体，其功能依赖于输入和输出，对每个事件或处理都是平等的，其特性是存储或处理的效率(需要的时间)。而且CDFG不仅能够描述系统硬件逻辑功能，同时能够描述软件流程。它是一个有向无环图，体现了整个系统的执行过程。图中的每个节点V_i对应一个系统任务，$V=\{V_i, 1\le i\le N\}$，N为应用划分出的任务总数。边$e_{ij}=\langle V_i, V_j\rangle \in E$表示任务节点间的数据交换过程和它们的依赖关系，整个图G是节点V与边E的集合，$G=\{V,E\}$。

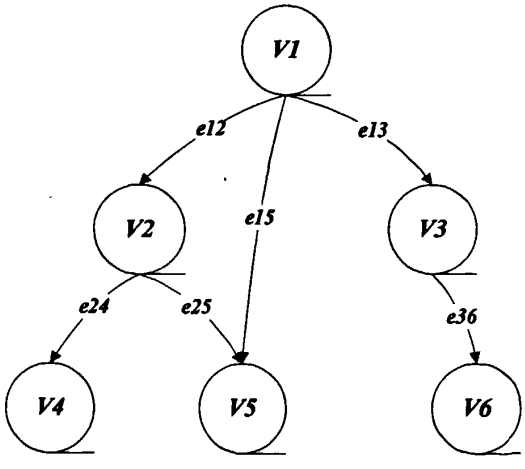

图4.1 描述系统的CDFG图

图4.1中每个任务节点都可以映射到软件域、硬件域(ASIC硬件)或可重构单元(FPGA)上，两个节点间的依赖关系由它们的边表示。任务图中的边都是有向边，图中一个节点任务在开始运行前，它的前趋节点任务必须执行完毕。整个系统的执行时间为结束时间减去开始时间。每个任务节点V_i都有一组与之对应的开销参数VST_i、VAT_i、VFT_i、VRT_i、

VAS_i和VFS_i。VST_i表示任务V_i的软件执行时间；VAT_i表示任务V_i用ASIC实现的执行时间；如果任务用FPGA实现，VFT_i为FPGA上的执行时间；VRT_i为该任务的配置文件加载到可重构单元的时间，即重构时间，其值为0时表示任务用非可重构单元的方式实现；VAS_i指任务用ASIC方式实现时所需的硬件面积，基于现在的技术水平ASIC的面积与成本成正比，因此当任务用ASIC实现时这一项又表示系统成本。VFS_i指当任务用FPGA实现时，任务占用可配置逻辑资源的面积。ASIC实现方式和可重构硬件实现方式并不冲突，因为在目标系统中可能存在某些经常需要周期执行的任务，如果固定使用可重构硬件实现不仅会占用宝贵的重构资源而且也会损失动态重构的灵活性，这时如果增加少量成本用ASIC实现将会使系统在时间和成本的约束条件下找到一个更佳的平衡点。本文加入对ASIC方式的考虑是为了使算法更具灵活性、实用性和拓宽算法应用范围。另外，如果系统约束要求不能使用ASIC方式，在算法中也只需做很小的变动即可实现。

用$P=\{V_0^A, V_1^R, V_2^S.....\}$表示目标系统的一次划分，其中$V_0^A$表示任务$V_0$用ASIC方式实现，$V_1^R$表示任务$V_1$用可重构单元即FPGA实现，$V_2^S$表示任务$V_2$用软件实现。$C^A$表示用ASIC实现的任务集合，$C^A=(V_0^A,...,V_i^A), C^A \subseteq V$；$C^R$表示用FPGA实现的任务集合，$C^R=(V_1^R,...,V_i^R), C^R \subseteq V$；$C^S$表示用软件实现的任务集合，$C^S=(V_2^S,...,V_i^S), C^S \subseteq V$。FPGA的总面积为$A_{FPGA}$，可重构区域数量为$Amt(DRU)$，划分后系统总实现时间为Time，总面积Area，可重构任务占用FPGA的面积为$Area^R$，ASIC面积$Area^A$，$Area = Area^A + Area^R$。

4.1.3 划分目标

动态可重构软硬件划分算法的目标就是在满足设计约束的条件下，使系统运行速度达到最快。系统的性能在很大程度上依赖于划分的结果，一个好的划分可以使系统运行时间、成本等指标达到最佳的平衡点。

一般来说，硬件的运行速度比软件快，软件比硬件灵活性高且成本低。把任务分配到ASIC或可重构单元上执行可以提高系统性能，但如果把过多的任务分配到ASIC会迅速增加系统成本，过多的分配到可重构单元又会导致配置文件频繁重构，从而增加重构时间影响系统性能，如果把任务过多的分配到软件执行，虽然成本降低了，但是由于软件实现效

率低和串行的执行特性将会极大的降低系统运行速度。另外除了性能和成本的约束外，动态可重构系统还有其他的一些约束条件，如可重构单元资源布局、配置文件存储器空间有限等。

在得到一个划分方案后，还需要使用调度策略对划分方案进行评估，检查是否满足时间设计约束。调度即说明系统各功能各自在何时、何处实现，它依赖于划分的结果，同时又为划分操作提供依据。在动态可重构系统中，调度策略除了分配任务执行顺序之外，还涉及预读取操作中配置文件的调度和可重构资源的布局问题。一个优秀的预读取操作调度可以有效的减小重构次数，隐藏重构时间。好的布局策略可以提高可重构单元的利用率，最大化可重构器件的加速作用。

根据上述描述，本章提出的动态可重构系统划分方案的目标是：对一个已知的系统用CDFG模型描述，系统运行实现方式有软件实现、ASIC和可重构单元实现，设计就是要在满足可重构单元面积资源约束条件下，决定目标系统的各个任务是用哪种方式实现，以使系统执行时间最小，成本最低，并且对分配好的节点只有当其所有前驱节点都执行完成时才能开始执行，对于用可重构单元实现的节点还应当在配置文件加载完成后才能开始执行即重构时间不能忽略。用数学方式表示如下：

约束： $\mathrm{Area}^{A}=\sum_{i}\mathrm{VAS}_{i}\le \mathrm{MAX}(\mathrm{Area}^{A})$, 其中$\mathrm{VAS}_{i}\in C^{A}$

使用ASIC实现的面积应小于系统要求的$\mathrm{MAX}(\mathrm{Area}^{A})$，超过这个值将使系统成本超出可接受范围。

目标：(1) 最小时间

$$\mathrm{MIN}(\mathrm{Time}),\ \mathrm{Time}=\sum_{i}(\mathrm{VST}_{i}+\mathrm{VHT}_{i}+\mathrm{VFT}_{i}+\mathrm{VRT}_{i})$$

其中$\mathrm{VST}_{i}\in C^{S},\mathrm{VHT}_{i}\in C^{A},\ C^{R},\mathrm{VFT}_{i}、\ \mathrm{VRT}_{i}\in C^{R}$

(2) 最低成本(面积)

$$\mathrm{MIN}(\mathrm{Area}),\ \mathrm{Area}=\mathrm{Area}^{R}+A_{FPGA}=\sum_{i}\mathrm{VAS}_{i}+A_{FPGA},\ 式中\ \mathrm{VAS}_{i}\in C^{A}$$

4.1.4 权重系数动态可变的目标函数

在传统的软硬件划分算法中，为了满足不同的系统约束条件和优化目标，可通过调整目标函数的开销权重系数来体现对系统性能的不同要求，例如在对系统运行时间有较高要求的设计中可增大时间开销的系数以体现对运行时间的关注程度。但是问题是如何选取合适的开销系数来衡量划分的结果，如果选取得不好，即使划分算法再优秀也难以得到客观

的评价结果，另外，由于可重构系统诸多因素的动态改变，也使得系数的确定难以进行。

目前这些系数多根据经验来确定，不同的设计者根据自己的经验所指定的系数往往差别很大。尤其在动态可重构系统中，引入动态重构技术后，由于重构时间占系统运行时间的比重较大，有时甚至占到了系统运行时间的大半，为了降低重构开销的影响人们提出了一些缩短重构时间的有效方法，如广泛采用的流水线和预读取技术等，它们可以把一个DRU(可重构单元的逻辑区域)的配置与另一个DRU的执行重叠起来或提前把配置文件加载到DRU中，以最大限度的减少重构的时间。这使得问题进一步复杂化，可重构的特性使以往的经验不能被简单的照搬使用，因此运用原有的方法将难以保证在动态可重构软硬件划分中得出最优的划分结果。

为了解决这些问题，本文提出一种与算法划分过程紧密结合在一起的逐步优化的系数确定方法，这种方法比根据人工经验指定的系数更加客观可靠，而且加入了对可重构系统特性的考虑，使得出的动态可重构软硬件任务划分结果更贴近于实际应用系统。我们使用的目标函数为：

$$\begin{aligned}\text{Fuction}(\ ) = \sum_{i=1}^{N}\Big(&(C1,C2,C3,C4,C5)\bullet(VST_i, VAT_i, VFT_i, VRT_i, VAS_i)^{-1} \\ &+ C6\bullet A_{FPGA} + C7\bullet e^{-k\left(\sum VAS_i - MAX(Area^{\wedge})\right)}\Big)\end{aligned} \tag{4-1}$$

系数行中的C1,C2,C3,C4,C5,C6,C7 分别表示节点各开销权重对于应用系统性能影响的相对权重，值越大，说明系统越关注它所对应的开销。针对动态可重构系统的特点，令C4,C5,C6 可以随着任务划分方案动态改变，以体现动态重构对软硬件划分的影响。具体取值如下：

$$C4 = \begin{cases} 0, Card(C^R) \le Amt(DRU) \\ \dfrac{(Card(C^R) - Amt(DRU) - 1)\bullet \sum VRT_i}{(Card(C^R))^2 \bullet VRT_i} + 1 - \dfrac{MAX(VT_{if})}{VRT_i}, Card(C^R) \ge Amt(DRU) \end{cases} \tag{4-2}$$

由于流失线和预读取技术可以有效的缩短重构时间，因此要把这个因素考虑到划分方案中，这样才能得到更加精确的划分结果。使目标函数中的C4 动态改变是为了修正分配到可重构单元执行的任务重构时间对系统性能的影响。考虑以下两种情况：

(1) 软硬件划分方案中分配到可重构单元执行的任务数量$Card(C^R)$小于可重构的逻辑区域数$Amt(DRU)$。这时可以把所有的任务都加载到FPGA中，不需要运行时重构，节省了所有的重构时间。这实际上是一种静态重构的实现方式，这种情况使用C4中的第一项值0，

即系统重构时间为0。

(2) 当划分方案中分配到可重构单元执行的任务数量$\mathrm{Card}(C^R)$大于可重构区域数$\mathrm{Amt}(DRU)$时，这种情况下就必须考虑任务重构时间VRT_i。在采用流水线和预读取技术的前提下，重构配置时可以与重构时间重叠的实际上是该任务前驱节点的最大执行时间$MAX(VT_{if})$（在预读取的调度中有说明）。另外，除了任务自身的重构时间外，如果重构的任务过多将发生需要等待前面的任务重构完成的情况。分配到可重构单元执行的任务越多，发生等待的概率就越大，因此在分配任务到可重构单元执行的过程中还必须考虑其对已分配好的任务的影响。定义等待重构概率$\beta=\frac{(\mathrm{Card}(C^R)-\mathrm{Amt}(DRU)-1)}{\mathrm{Card}(C^R)}$，从式中可看出等待重构的概率$\beta$与$\mathrm{Card}(C^R)$及$\mathrm{Amt}(DRU)$有关，即分配到可重构单元执行的任务越多，可重构区域越少则等待重构的概率就越高，反之概率就越低。等待重构的时间可以取在一次划分结果中所有C^R任务重构时间的平均值$AVG(VRT)=\sum VRT_i/\mathrm{Card}(C^R), VRT_i \in C^R$。因此修正的任务重构时间是指，任务自身的重构时间加上等待重构时间与等待概率之积，它们的和再减去重叠的前驱节点最大执行时间。这个修正时间可以通过使用$C4$中的第二项来体现。

$$C5=\begin{cases}C5,\text{任务用ASIC实现}\\0,\text{任务用可充重构单元或软件实现}\end{cases} \tag{4-3}$$

C5为用ASIC实现的任务面积的开销系数，表示设计需求对ASIC面积(成本)的关注程度。当设计不允许使用ASIC实现，只能采用可重构单元或软件实现时，C5取0。

$$C6=\begin{cases}0,\text{所有任务用ASIC或软件实现}\\1,\text{有任务用可重构单元实现}\end{cases} \tag{4-4}$$

当有任务采用FPGA实现时，不管分配到其上实现的任务有多少个，可重构硬件面积都是固定的，即FPGA的面积A_{FPGA}，C6取值1。如果整个目标系统的任务都没有分配到FPGA上执行，则C6值取0，这时算法变成了一种传统的软硬件划分算法。

目标函数中$C7\bullet e^{-k(\sum VAS_i-MAX(Area^A))}$项表示当$Area^A=\sum VAS_i$（ASIC的总面积或成本）超过系统设计允许的最大值$MAX(Area^A)$时，对适应度指数的惩罚程度。由于使用的是指数型的惩罚函数，因此惩罚将以指数级增长，超过越多则此划分方案的适应度越差。

4.1.5 动态可重构软硬件任务划分算法

动态可重构技术的出现促进了嵌入式系统的发展，其中的软硬件划分方法作为协同设计的关键步骤亟待进一步的研究、探索。动态可重构任务的划分算法，就是在一定的系统模型之上，对解空间进行搜索，最终确定目标系统的每个任务节点采用哪种实现方式，并满足系统的设计约束。本文提出结合调度策略的基于权重系数动态可变的免疫算法对动态可重构系统进行软硬件任务划分，与传统的软硬件划分方法相比，除了加入为适应动态可重构系统特点所做的调整外，还考虑了目前较常用的用来减少重构时间的技术对系统性能的影响，使划分的结果更贴近实际目标系统。算法的几个关键步骤和流程如下：

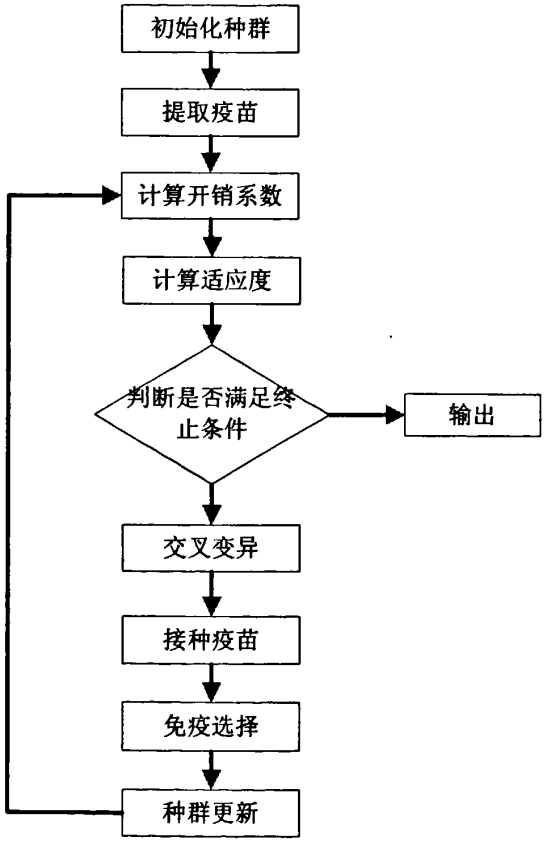

图4.2 权重动态可变的免疫算法流程图

(1) 编码策略选择

将CDFG中的任务节点对应于免疫算法中的基因，基因的值对应该任务的实现方案。任务有3种实现方案，每种方案对应一个节点的划分结果，使用十进制编码方式，这样是为了更好的贴近实际目标系统，采用0表示任务节点V_i用软件实现，1表示用非可重构硬件(ASIC)实现，2表示用可重构单元(FPGA)实现。设图中共有N个节点即N个任务，向量

$T=(V_1,V_2,.....,V_N)$表示算法中的一个染色体。初始种群大小选择$K=2N$，种群太小可能会导致早熟收敛，太大又会增加计算量。

(2) 个体适应度评估

适应度函数用于对个体进行评价，也是优化过程发展的依据。本文采用的适应度函数Fuction( )为上一章提出的权重系数动态可变的目标函数，其中加入了对重构时间和缩短重构时间技术的考虑，函数值越小表示个体代价越低，系统的优化程度越高。个体的评估还结合了动态可重构系统调度算法，目标是找到既符合时间约束又使系统开销最小的软硬件划分结果。

3、 疫苗的提取

首先对目标系统进行分析，搜集特征信息，提取出可利用的基本信息或先验知识，然后将这些信息放进疫苗库中形成较优的解决方案，即最优个体的某些基因的取值范围。在划分算法中疫苗代表了某节点的较优实现方案，如疫苗$(V_i,0)$表示节点i的较优方案是用软件实现；疫苗$(V_j,2)$表示节点j的较优方案是用FPGA实现。

4、 交叉变异操作

交叉操作[36]可以组合出新的个体，在解空间中进行有效搜索，同时保证优秀个体的基因在子代中尽可能的遗传。本文采用多点交叉的方法，首先随机确定两个交叉位置，并交换交叉点之间的片段，其他位置不变。例如，两个父代个体为$T_1=(0,1,2,0,1,0,2,1,0)$，$T_2=(1,2,0,2,0,1,2,0,1)$，若交叉位置为3和7，则后代个体为$T_1=(0,1,0,2,0,1,2,1,0)$，$T_2=(1,2,2,0,1,0,2,0,1)$。变异是为了保持群体多样性，克服早熟收敛使问题陷入局部最优解，算法以0.005的概率进行变异操作。

5、接种疫苗

在交叉变异操作后，随机抽取一些个体注射疫苗，即按照系统的较优方案来修改个体某些基因位上的分量，使个体能以较大的概率获得更高的适应度。例如个体为$T_1=(0,1,2,2,1,0,2,1,0)$，疫苗为$(V_2,0)$，$(V_8,2)$则接种疫苗后的个体为$T_1=(0,0,2,2,1,0,2,2,0)$。接种按接种概率α，随机的取αK个个体进行操作，默认取经验值0.8，也可以依据此方案的优化效果而定。

6、 免疫选择

、首先进行免疫检测，如果接种了疫苗的个体适应度不如父代，则说明发生了退化现象，

这时用父代个体取代对应的子代；反之继续。然后用退火选择选取子代群体中进入下一父代群体的个体[36]。

7、 计算权重系数

根据遗传和免疫算子得出的软硬件划分结果，重新计算目标函数Fuction( )中各个开销权重的系数，使开销系数与各划分方案的实现相关联，适应动态可重构系统的特点，避免了静态估计各开销系数所带来的脱离实际划分结果的问题。

8、 算法终止条件

本文中算法终止的条件是最佳优化值连续L代没有明显的变化或达到最大代数200。当节点数N小于50时， $L = 2N$ ；节点数N大于100时，L=100。

4.2 对预读取和可重用配置优化的调度算法

嵌入式系统任务在执行时需要为每个任务分配开始时间以满足系统的时序要求。调度算法的目标就是在任务集合中选取任务并把它分配到相应的计算单元中执行，并确定任务开始执行的时间以使系统总运行时间最短。调度在满足时序约束的前提下，还应考虑资源利用率、吞吐率等问题。在动态可重构系统中，对于每一种软硬件划分方案，任务实现的方式不同，分配到可重构单元上的任务数量、重构时间、重构顺序与重构时间的重叠情况等都会有所不同，因此在调度算法中还应包含对重构时间和缩短重构时间技术的考虑。调度算法除了作为划分算法的评估手段对划分结果作评价外，还可以实现对重构时间的有效隐藏，进一步提高系统性能。由于动态可重构系统的动态部分重构、重构时间和多种任务执行方式的特点使之与传统的计算系统之间存在很大差别，因此调度算法也必须进行修改以适应动态可重构系统的这些新特性。本文提出的动态可重构系统调度算法包括预读取配置文件调度和任务调度，在实现时需要建立任务就绪队列TRL、预读取配置就绪队列CPL和一个已配置文件列表CRL。资源布局模型采用一维定宽模型，这是因为该模型具有结构简单、容易实现的优点，且目前商用产品对它的支持也最好，基于该模型的调度算法更具有现实意义。

4.2.1 预读取配置文件的调度

动态可重构系统有很多与传统的计算系统不同的地方，它用可重构硬件来同时提高系统的计算速度和灵活性，并由此带来了很多独有的特性。如当系统调度器把任务分配到可

重构单元执行时，要首先把配置文件存储器中的配置文件加载到可重构单元中，然后才能在可重构单元上执行。因此动态可重构系统的调度算法也应体现出这一过程，以实现系统性能的最优化目标。本文提出的动态可重构系统调度算法除了包含传统意义的任务调度外，还包括对配置文件的调度。这样做是为了更好的利用流水线和预读取技术实现重构时间的有效隐藏。一般来说，硬件执行的速度比软件执行更快，在软硬件划分时都要尽可能的把周期任务和软件执行耗时长的任务分配到可重构单元中，虽然预读取技术可以事先把配置文件加载好，但是如果随机的将一些任务的配置文件预读取到可重构单元中，除了增加系统的不确定性外，甚至还可能起到适得其反的作用。因为在目前的技术条件下，重构时间占运行时间的比重较大，如果无规律的频繁换进换出配置文件将浪费大量的系统运行时间，因此需要使用某种调度策略来对配置文件进行预读取操作。

预读取调度需要建立一个预读取配置就绪队列CPL，能够进入队列的配置文件需要满足的条件是该任务的所有前驱节点都已进入任务就绪队列。进入预读取配置就绪队列的配置文件根据调度优先级依次在可重构资源空闲的情况下加载到可重构单元中。这样做是因为可重构硬件资源有限而需要用可重构单元执行的任务数量又较多，随机的预读取配置文件可能会导致没有就绪的任务占据有限的可重构资源，而当有其他任务就绪时又会需要重新进行配置，导致重构次数增多，反而增加了重构时间。预读取配置文件优先级的确定以ASAP(尽可能早调度)算法为基础，结合每个节点的前驱节点任务执行方式及时间计算。优先级计算公式如下：

$$P(file_i) = P - (ASAP_i + MAX(VT_{if})) \tag{4-5}$$

每个用可重构单元执行的节点的$ASAP_i$值根据CDFG系统模型图计算出，其值为从源节点开始到V_i节点的最大路径长度，可以通过静态计算得出。$MAX(VT_{if})$为V_i所有前驱节点的最大执行时间。P为理论最大值，优先级值$P(file_i)$越大，节点V_i配置文件的优先级越高。预读取配置文件调度算法的伪代码如下：

```
ConPreAlm(C^R)
{
    while(V_i ∈ C^R ∩ V_i的前驱节点都已进入任务就绪队列)
    {
        ASAP_i = CalASAP(CDFG);
        MAX(VT_if) = CalVT(CDFG);
        P(file_i) = P - (ASAP_i + MAX(VT_if));
```

```
        i++;
    }
    while(Vacant(FPGA))
    {
        Load(MAX(P(file_i)));
        MAX(P(file_i)) = 0;
        if(V_i ∈ TRL ∩ TaskV_i < M − VRT_i)
        TaskV_i = TaskV_i + VRT_i;
    }
}
```

算法首先计算进入预读取配置就绪队列节点的优先级，即前驱节点都已就绪且以可重构单元执行的节点的优先级，未进入队列的配置文件优先级都初始化为0。算法中使用$MAX(VT_{if})$是因为对于前驱节点执行较快的节点，其进入任务就绪队列的时间也较早，因此需要优先在可重构硬件上配置好，这样做也可以使重叠的重构时间更多一些，最大化系统性能。然后判断可重构单元是否有空闲区域，如果有则加载优先级最高的任务配置文件，再把该文件从预读取配置就绪队列中删除。

4.2.2 可重用配置的优化

在嵌入式系统中有很多需要周期执行的任务，如果这些任务在可重构单元上执行时不需要每次都重新加载配置文件，而重复利用上次该任务执行时配置好的DRU，将极大的缩短任务重构时间。可重用配置优化的最大问题是找到一种判断执行完的配置文件是否需要继续保留的方法，如果该配置文件在接下来的执行中还需要重复使用则保留，反之则删除该配置文件以空出可重构资源给后续的任务。可重用配置优化的伪代码如下：

```
ChkCon ( V_i ∈ C^R )
{
    if(Check(TRL)!=0)
    {
        if(TaskV_i < M − VRT_i)
        TaskV_i = TaskV_i + VRT_i;
        return;
    }
    elseif(Check(CPL)!=0 ∩ TRL中的 C_i^R ∈ CRL)
    {
```

```
        P(file_i) = 0;
        return;
    }
    else
    Del( file_i );
}
```

确定任务的配置文件在执行完成后是否保留需要分三种情况：

(1) 在任务就绪队列中有该任务。此时加大该任务在任务就绪队列中的优先级，因为该任务可以重用可重构资源，不需要再加载任务配置文件就能立即执行，可以节省重构时间。

(2) 该任务在预读取配置就绪队列中，且任务就绪队列中的可重构任务都已配置完成即都在CRL列表中。任务配置文件不进行替换，相当于已经把CPL中的任务配置文件加载完成。将该任务从CPL中删除。

(3) 任务既不在TRL而且也不在CPL中。这时把该任务的配置文件替换为其他任务。替换根据预读取配置文件调度策略进行。

4.2.3 动态可重构系统任务调度算法

由于动态可重构系统的动态重构特性拓展了系统的执行方式，因此动态可重构系统的任务调度算法也应进行改进以体现系统结构的变化和执行方式的多样化。本文提出的调度算法首先建立两个队列和一个列表，任务就绪队列TRL、预读取配置就绪队列CPL以及已配置文件列表CRL。

TRL队列保存就绪的软件和硬件任务，根据任务的优先级进行调度。优先级的计算使用ASAP调度算法并结合任务配置的情况做动态改变，以进一步提高调度的效率和可重构资源的利用率。首先通过系统模型CDFG图计算每个任务节点的ASAP值，包括软件节点和硬件节点。然后在系统执行过程中根据CRL中记录的文件配置状况加大相应任务的优先级。具体操作为，对新加入TRL中的可重构任务，如果在可重构单元中有已配置好的文件即该任务在CRL中，则提高该任务的优先级，这样做有助于配置重用和利用硬件执行的并行性加快任务的执行；对没有配置好的可重构就绪任务在预配置完成后，加大该任务在TRL中的优先级值，这样可以做到先配置完成先执行，能够隐藏部分还没有配置好的任务的重构时间。CPL保存的是预读取配置文件就绪队列，预读取配置算法使用该队列进行可重构任务配置文件的预先加载工作。在配置文件加载到可重构单元后将其加入已配置文件列表

CRL中，CRL保存可重构单元中能立即执行的任务，这些任务的配置文件也可进行重复利用以减少重构时间。任务调度的伪代码如下：

```
TaskScheduling()
{
    初始化TRL (TaskV_i = M − CalASAP(CDFG);)和CPL，CRL;
    ConPreAlm(C^R);
    while(V_i ∈ TRL ∩ V_i ≠ φ)
    {
        if(V_i ∈ C^S ∩ CPU空闲)
        V_i在CPU上执行
        elseif(V_i ∈ C^A)
        V_i在ASIC上执行
        elseif(V_i ∈ C^R ∩ V_i ∉ CRL)
        {
            V_i = FetchNext(TRL);
            continue;
        }
        else(V_i ∈ C^R ∩ V_i ∈ CRL)
        {
            V_i在可重构可重构单元」
            ChkCon(V_i);
        }
        ConPreAlm(C^R);
        更新TRL和CRL;
        (对新加入TRL的V_i
        if(V_i ∈ CRL ∩ TaskV_i < M − VRT_i)
        TaskV_i = TaskV_i + VRT_i;)
        V_i = MAX(TaskV_i);
    }
}
```

算法在执行时先初始化TRL、CPL队列和CRL列表，将就绪的软件和硬件任务放到TRL队列中，如果CPU资源空闲就执行其中优先级最高的软件任务；如果是ASIC任务，因为ASIC任务不需要重构且硬件具有的并行执行特性，所以该任务可以立即开始执行；对于用

可重构硬件执行的任务，在可重构资源空闲时根据CPL队列的优先级进行配置，配置完成后根据TRL队列的优先级执行，多个可重构硬件任务在时序允许的前提下也可以并发执行。

4.3 实验结果与分析

在得到软硬件划分结果后，需要对划分结果作评价，以验证其是否能适用于动态可重构系统的软硬件任务划分。首先利用TGFF工具[39]随机生成6种具有不同节点数的CDFG，每个节点都包含软件、ASIC、可重构单元的执行时间和重构时间、占用的ASIC和可重构单元面积等开销，同时确定系统约束条件和这些约束条件的关注度。可重构单元使用一维等宽布局模式，V_i^R的最大高度≤可重构单元(FPGA)的高，V_i^R的最大宽度≤可重构单元(FPGA)的宽。用可重构单元执行相同的任务通常比使用CPU执行更快，而使用ASIC实现的任务由于在设计硬件时就能充分考虑应用的计算特点，因此其执行速度比用可重构硬件实现更快，考虑参数设置的合理性，使每个任务的可重构硬件执行时间比软件执行时间快5～8倍，ASIC比可重构硬件执行快1.2～1.5倍。仿真程序的实验平台为Celeron2.8G，512M RAM。为了避免测试结果的偶然性，采用多次实验取平均值的方法。实验比较结果如下：

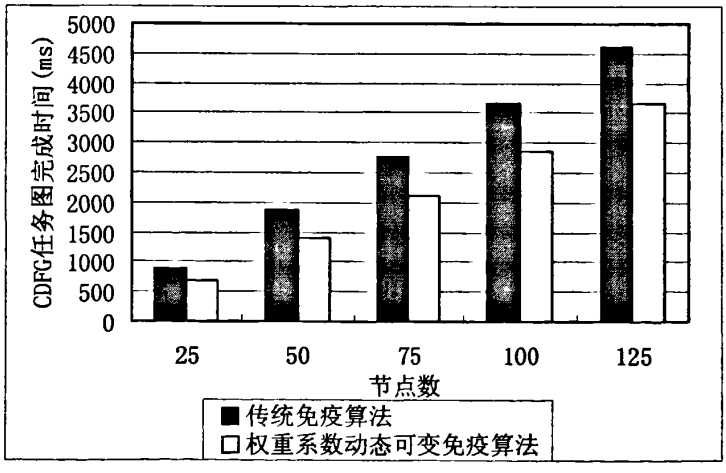

图4.3 传统免疫算法与权重系数动态可变免疫算法对比图

如图4.3所示，将传统的免疫算法与权重系数动态可变的免疫算法进行对比，可重构单元逻辑区域即DRU数取3。由于两者分别采用了不同的目标函数，所以单就算法运行时间的比较已经没有意义。在图中可以看到新算法的任务图完成时间要比传统算法的完成时间要短。这是因为传统的划分算法得出的划分结果并不恰当，它根本没有对动态可重构系统的特点加以考虑，尤其是没有把对系统性能影响极大的重构时间开销纳入到软硬件划分的过程中，在划分时把可重构单元等价于ASIC硬件来使用，背离了系统的设计要求，使系统

在运行调度时产生了很多不必要的重构延时并增加了可重构任务的等待概率，因此传统的免疫算法不能适用于动态可重构系统的任务划分。

在动态可重构系统中，多个任务可以被时分复用的映射到可重构单元上执行，这相当于增加了硬件的可使用面积，同时流水线和预读取技术进一步提高了系统的性能，并且当一个新的任务被划分到动态可重构单元执行时，它会和已经划分到可重构单元上的任务共同对后面任务的执行时间产生影响。在这些动态改变的情况下，划分算法也应该进行相应的变化来适应动态重构所带来的新问题。实验表明，采用权重系数动态可变的免疫划分算法能够获得比传统的免疫算法更好更贴近实际环境的软硬件划分结果，充分发挥可重构硬件的计算优势，从而加快目标系统的运行速度。

表4.1 两种算法性能对比

节点数	遗传算法		免疫算法	
	最优解值	运行时间(ms)	最优解值	运行时间(ms)
25	612.3	34	609.5	32
50	1305.3	356	1312.1	361
75	1956.2	584	1948.8	562
100	2309.7	812	2313.4	795
125	3123.4	1245	3117.6	1089

用传统的遗传算法与免疫算法进行对比，DRU数同样取3。表4.1中的实验结果表明：两种算法都能取得较好的划分结果，划分方案的最优解值相近。但由于免疫算法中加入了免疫算子，充分利用了从任务的基本特征信息中提取出的疫苗，使原来只依靠历史信息进行搜索的算法更贴近应用特征，也更容易收敛于最优解，从而较大的提高了算法的收敛速度，特别是在节点任务较多时效果更明显。采用权重系数动态可变的目标函数能够适应动态可重构系统的特点，但函数复杂度的增加会导致算法的收敛速度减慢，而免疫算法能加快收敛速度，因此把这两者结合起来用于动态可重构系统任务的软硬件划分是一种比较好的且更符合实际目标系统特征的方法。

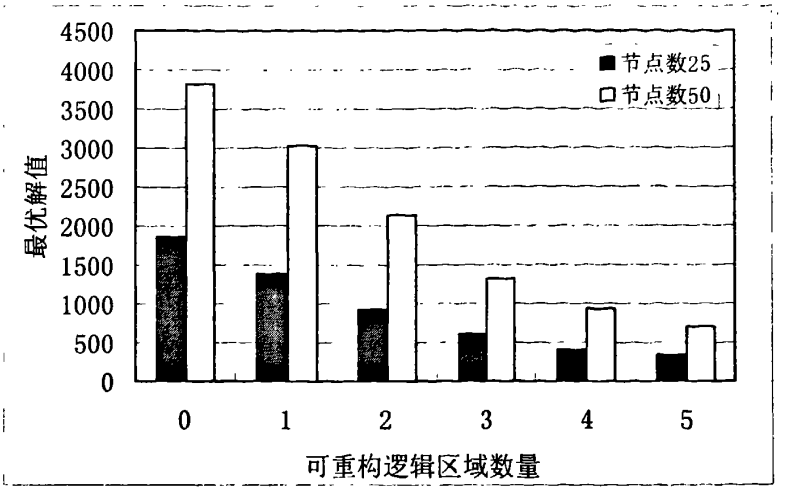

图4.4 可重构单元规模不同的情况下算法的最优解值比较

在可重构单元规模不同的情况下，算法对应节点数为25和50的系统最优解值比较如图4.4。可以看出，随着可重构硬件规模的增加即一维等宽的DRU单元数量增加，系统性能得到不断改善。这是因为可重构资源的增加，可以使更多的任务使用可重构单元实现，流水线和预读取技术也可以使更多的重构时间和任务执行时间重叠，并且等待重构概率也将会更低。图中DRU数量为0时表示系统使用软件或ASIC实现。随着DRU数量的增加，系统性能并不是成比例增长，这是因为系统中并不是所有的任务都适合使用可重构单元实现，且当可重构资源规模达到一定程度后，可以使用预读取配置算法实现资源的时分复用，不需要再增加资源的规模。可以看到如果可重构资源不断增加，最优解值的优化效果将会慢慢消失，最后趋于将所有任务都划分到可重构单元上执行的效果。

另外，把系统任务划分为更小的粒度在一定程度上也可相当于增加了可重构硬件的资源数量，因为在一维等宽的布局模式下，同样尺寸的可重构硬件可以划分出更多的DRU并保证足以容纳粒度更小的任务，使得重构的次数和重构等待时间都缩小。但是这样做也有一定的局限性，任务划分得过小会使系统任务数量增大，划分算法将耗费更多的计算时间，也使设计者难以辨识理解系统任务，对系统的评估也会造成较大困难。因此，在满足成本约束的条件下，其性能优化的效果不如直接增加可重构资源数量。

4.4 本章小结

动态可重构技术使嵌入式系统的执行方式发生了巨大的改变。本章在研究动态可重构特点和免疫算法特性的基础上，又考虑了流水线、预读取技术和可重用配置优化对系统性能的影响，提出采用权重系数动态可变的免疫算法实现动态可重构系统软硬件任务的划

分，并结合动态可重构调度算法做出评价。实验结果表明，算法较传统的划分方法更贴近动态可重构系统的执行环境，能有效实现软硬件任务的划分，降低重构时间的影响，提高系统性能。

第五章 总结与展望

软硬件协同设计广泛应用于嵌入式系统的设计中，其首要问题是软硬件任务的划分，划分的结果将对后面的设计实现及整个系统的性能产生直接影响。近年来，随着FPGA技术的发展，可重构计算模式在越来越多的领域中都显示出了极大的性能优势，使动态可重构系统成为了嵌入式系统发展的一个重要方向。

本文主要完成以下工作：

(1) 软硬件划分技术和可重构系统特点的研究。软硬件划分将系统任务分解成软件和硬件实现部分，确定系统的软硬件界面，把任务分配到相应的软件和硬件上，通过不断改进达到软硬件各部件之间优化配置。可重构系统相较传统的嵌入式系统具有很多新的特点，就动态可重构系统来说，对系统性能影响较大的因素有重构时间、可重构单元的资源模型等。为解决重构时间过长的问题，又提出了很多缩短重构时间的方法，其中较具代表性和实用性的有流水线、预读取技术和配置重用技术。本文认为这些问题在进行软硬件划分时都应该加以考虑，这样才能更符合系统运行的实际环境，使划分结果更具合理性。

(2) 提出了一种基于权重系数动态可变的免疫算法的软硬件划分算法。在建立动态可重构系统CDFG模型的基础上，提出通过动态改变开销权重系数来适应动态可重构系统的技术特征，从而把各个划分方案中不同的重构时间、重构等待概率和硬件使用面积等开销统一到评估体系中对系统划分结果做出评价，这样不但适应了动态可重构的特性，而且使划分结果的评估更客观真实。但是这将不可避免的增加划分算法的运行时间，因此本文采用免疫算法，通过使用疫苗中的先验知识，加快算法的收敛速度。基于权重系数动态可变的免疫算法能较好的适用于动态可重构系统的软硬件划分，充分发挥可重构单元的计算能力，提高系统的整体性能。

(3) 提出了对预读取和可重用配置优化的动态可重构任务调度算法。调度算法在实际应用中除了用来满足动态可重构系统的时序要求和评估软硬件划分结果外，还可以进一步缩短重构时间，提高系统性能和可重构资源利用率。本文提出的动态可重构调度算法包括预读取配置调度和对可重用配置进行优化的任务调度。通过使用TRL、CPL队列以及预读取技术，实现了重构时间和前驱任务节点执行时间的重叠，降低了重构延迟对系统性能的影响，并且通过任务优先级的动态改变和CRL列表实现了可重用配置的重复利用，减少了任务配置文件的加载次数，进一步降低了重构开销代价。

本文所给出的动态可重构系统软硬件任务划分方法虽然可以较有效的实现软硬件任务划分，并使划分结果更符合动态可重构系统的特点。但是也还存在着一些未解决的问题，

有待进一步研究：

(1) 算法的资源布局模型采用的是一维等宽模式，随着 FPGA 技术的发展，二维模式甚至二维尺寸可变模式由于其资源利用率高的优势，将成为今后动态可重构资源布局模型的主要方式，因此划分算法在二维资源模型中的应用还有待进一步的研究发展。

(2) 现代 EDA 软件在进行系统设计时，还没有把自动的软硬件划分方法结合起来，如何使 EDA 软件具有自动划分功能，尤其是针对动态可重构系统的划分，还有大量的工作要做。

(3) 动态可重构系统的操作系统是可重构技术研究的一个重要问题。如何将软硬件划分以及动态可重构任务调度算法同操作系统相结合，也是一个可以深入探讨的研究领域。

致 谢

感谢我的导师姚放吾教授在我研究生期间对我的悉心指导。在这三年中，姚老师在生活、科研和项目开发实践中都给予了我极大的帮助和关怀。姚老师渊博的知识、深厚的学术功底、严谨的治学态度和一丝不苟的科研精神深深的影响着我，将使我受益终生。

感谢912教研室的师兄师姐师弟师妹在我完成论文期间给我的指导和灵感，以及在这三年生活中对我的支持和鼓励，这些都给我留下了许多值得珍藏的美好回忆。

感谢我的父母和妻子，他们无私的爱是我前进的动力。

感谢所有关心和帮助过我的人们。

最后，特别感谢审阅本文的各位教授、专家，感谢你们在百忙之中抽出时间审阅本文并提出宝贵的意见。

参考文献：

[1] 桑楠. 嵌入式系统原理及应用开发技术. 北京航空航天大学出版社. 北京. 2002

[2] 张大波，吴迪，郝军，沙毅，冯建新. 嵌入式系统原理、设计与应用. 机械工业出版社. 北京. 2004

[3] Zhihui Xiong, Sikun Li, Jihua Chen, Dawei Wang. A platform-based SoC hardware/software co-design environment. Computer Supported Cooperative Work in Design. Proceedings. The 8th International Conference on Volume 2, 26-28 May 2004 443-448.

[4] Rabi Mahapatra. Hardware Software Codesign of Embedded System. CPSC689-602.

[5] Kent, K.B., Serra, M. Horspool, N. Hardware/software co-design for virtual machines Computers and Digital Techniques, IEE Proceedings -Volume 152, Issue 5, 9 Sept. 2005 537-548.

[6] 李晖. 嵌入式系统软硬件划分方法及协同设计研究. 南京邮电大学硕士学位论文. 2006年.

[7] Zhihui Xiong, Maojun Zhang, Sikun Li, Shaohua Liu, Yafei Chao. Virtual embedded operating system for hardware/software co-design. ASIC. ASICON 2005. 6th International Conference On Vol2, 24-0 Oct. 2005 939 - 943.

[8] Abdelhalim, M.B., Salama, A.E., Habib, S.E.D. Hardware Software Partitioning using Particle Swarm Optimization Technique. System-on-Chip for Real-Time Applications, The 6th International Workshop on Dec. 2006 189-194.

[9] Yan Sun, Xiaoqing Liu, McMillin, B. A Methodology for Structured Object-Oriented Elicitation and Analysis of Temporal Constraints in Hardware/Software Co-analysis and Co-design of Real-Time Systems. Computer Software and Applications Conference. COMPSAC '06. 30th Annual International Vol 1, 17-21 Sept. 2006 281-290.

[10] Bhattacharya, A., Konar, A., Das, S., Grosan, C., Abraham, A. Hardware Software Partitioning Problem in Embedded System Design Using Particle Swarm Optimization Algorithm. Complex, Intelligent and Software Intensive Systems. CISIS 2008. International Conference on 4-7 March 2008 171-176.

[11] Lysecky, R.; Vahid, F. A configurable logic architecture for dynamic

hardware/software partitioning. Design, Automation and Test in Europe Conference and Exhibition. Proceedings Vol1, 16-20 Feb. 2004 480 - 485.

[12] S Banerjee, N Dutt. Very Fast Simulated Annealing for HW-SW Partitioning. Technical Report CECS-TR-04-18.UC, Irvine.2004.

[13] 魏云龙. 动态可重构计算及演化硬件的研究与应用. 福州大学硕士学位论文. 2004.

[14] O.Diessel, H.EIGindy, M.Middendorf, H.Schmeck and B.Schmidt, "Dynamic scheduling of taskson partially reconfigurable FPGAS". In IEE Proc-Comput.Digit.Tech.Vol.14 7,No.3,May 2000.

[15] Zhu Mingcheng, Huang Qiang, The Initial Study on Dynamically Re-configurable Logic System Based on FPGA, Cottege of information Engineering, Shenzhen University.

[16] 康一梅等. 嵌入式软件设计. 机械工业出版社. 北京. 2007.06

[17] Rolf Ernst, Wei Ye, Thomas Benner and Jorg Henkel, Fast timing analysis for hardware /software co-design, In ICCD'93, 1993.

[18] R.Gupta and G. De Micheli. Hardware-Software Co-synthesis for Digital Systems. IEEE Design &Test, Vol.10, No.3, September 1993 29-41.

[19] Frank Vahid and Thuy DM Le. Towards a Model for Hardware and Software Functional Partitioning. Proceedings of the 4th International Workshop on Hardware/Software Co-Design (Codes/CASHE'96).

[20] Ta-Cheng Lin, Sadiq M. Sait , Walling R. Cyre. Buffer Size Driven Partitioning for HW/SW Co-Design. Proceedings of the International Conference on Computer Design 1998 Institute of Electrical and Electronics Engineers, Inc.

[21] X. Hu , G.. W. Greenwood, S. Ravichandran, G. Quan. A framework for User Assisted design space exploration 1999 ACM p414-419.

[22] Jorg Henkel. A Low Power Hardware/Software Partitioning Approach for Core-based Embedded Systems. 1999 Design Automation Conference.

[23] M. Meerwein, C. Baumgartner, Robert Bosch Gmbh. Linking Codesign and Reuse in Embed Systems Design. 2000 CODES.

[24] Daniel. Gajski, Frank Vahid 等著，边计年，吴为民等译. 嵌入式系统的描述与设计. 北京：机械工业出版社.2005.7.

[25] Saha, Proshanta (George Washington University); El-Ghazawi, Tarek, A

Methodology for Automating Co-Scheduling for Reconfigurable Computing Systems, Proceedings - Fifth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, MEMOCODE'07, 2007, p159-168.

[26] Vuletic, Miljan (Ecole Polytechnique Federale de Lausanne); Pozzi, Laura; Ienne, Paolo, Seamless hardware-software integration in reconfigurable computing systems, IEEE Design and Test of Computers, v 22, n 2, March/April, 2005, p102-113.

[27] 郭天天.嵌入式系统软硬件划分技术研究[D]. 博士学位. 国防科学技术大学.2006.

[28] 李涛. 动态重构系统若干关键问题的研究[D]. 博士学位. 南开大学 2007.

[29] Silva, Miguel L. (Faculdade de Engenharia, Universidade do Porto); Ferreira, Joao Canas, Support for partial run-time reconfiguration of platform FPGAs, Journal of Systems Architecture, v52, n12, December, 2006, p709-726.

[30] J.Villasenor, B.Schoner, K.-N,China, C.Zapata, H.J.Kim, C.Jones, S.Lansing, B.Mangione-Smith. Configurable Computing Solutions for Automatic Target Recongnition. IEEE Symposiumon FPGAs for Custom Computing Machines, 1996.pp.70-79.

[31] J. G. Eldredge, B. L. Hetchings. RRANN: the run-time reconfiguration artificial neuralnetwork. IEEE custom integrated circuits conference, San Diego, CA, 1994, 77-80.

[32] 齐骥，李曦，于海晨，胡楠，龚育昌，王立刚. 一种面向动态可重构计算的调度算法. 计算机研究与发展. 2007.44 (8):1439～1447.

[33] Vuletic, Miljan (Ecole Polytechnique Federale de Lausanne); Pozzi, Laura; Tenne, Paolo, Seamless hardware-software integration in reconfigurable computing systems, IEEE Design and Test of Computers, v 22, n 2, March/April, 2005, p102-113.

[34] S.Banerjee, E.Bozorgzadeh, N.Dutt. Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration. Proc. of the 42nd Design Automation Conference 2005, California, USA, Jun. 2005, 335-340.

[35] Shang, Li (Department of Electrical and Computer Engineering, Queen's University), Dick, Robert P., Jha, Niraj K., SLOPES: Hardware-software cosynthesis of low-power real-time distributed embedded systems with dynamically reconfigurable FPGAs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v 26,n 3, March, 2007, p508-525.

[36] 王凌. 智能优化算法及其应用. 清华大学出版社，施普林格出版社. 北京. 2001.

[37] 林飞卿,余传霖,何球藻. 医学基础免疫学. 上海. 上海医科大学出版社.

[38] 李德识，曹阳. 基于CDFG的SoC验证方法及其分割与搜索算法. 计算机工程. 2007.Vol.33 No.2.9-11.

[39] Keith Vallerio. Task Graphs for Free (TGFF v3.0).

[40] 侯慧，曹伟，王健，来金梅，童家榕. 动态可重构技术浅述. 半导体技术. 2008. Vol.33 No.7 553-557.

[41] Ebeling, C. Configurable Computing Platforms - Promises. Application-specific Systems, Architectures and Processors. ASAP '06. International Conference on Sept. 2006 3 - 4.

[42] K.Een Chehida, M.Auguin. HW/SW Partitioning Approach For Reconfigurable System Design [C] Proceedings of the 2002 international conference on compilers, architecture, and synthesis for embedded systems: 247-251.

附录:攻读硕士学位期间发表的学术论文

1. 姚放吾，卢昭材. 基于权重可变免疫算法的动态可重构任务划分，计算机技术与发展，2009.07
2. 姚放吾，曹木莲，卢昭材，丁富瞬. 51单片机与SD卡接口设计. 工业控制计算机. 2008. 0921. 60-61,68