MASTER THESIS XIDIAN UNIVERSITY

西安电子科技大学

硕士学位论文

基于C6000指令集可靠性评估方法的研究

作者姓名 曹玉芳 导师姓名、职称 郭宝龙教授

一级学科 控制科学与工程 二级学科 控制理论与控制工程

申请学位类别 工学硕士 提交学位论文日期 2014 年 12 月

西安电子科技大学
学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文若有不实之处，本人承担一切法律责任。

本人签名：曹玉芳　　日　　期：2014.12.23

西安电子科技大学
关于论文使用授权的说明

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

保密的学位论文在___年解密后适用本授权书。

本人签名：曹玉芳　　导师签名：郭宝龙

日　　期：2014.12.23　　日　　期：2014.12.23

学校代码 10701　　学　号 1204122050
分 类 号 TP39　　密　级 公开

西安电子科技大学

硕士学位论文

基于C6000指令集可靠性评估方法的研究

作者姓名:曹玉芳

一级学科:控制科学与工程

二级学科:控制理论与控制工程

学位类别:工学硕士

指导教师姓名、职称:郭宝龙教授

提交日期:2014 年 12 月

Research on Method of Reliability Evaluation Based on C6000 Instruction Set

A thesis submitted to
XIDIAN UNIVERSITY
in partial fulfillment of the requirements
for the degree of Master
in Control Science and Engineering

By
Cao Yufang
Supervisor: Prof. Guo Baolong
December 2014

摘要

航天和卫星技术的迅猛发展以及对超大规模集成电路的应用需求，使得对DSP 等集成芯片的依赖性越来越强，并且微处理器设计向低功耗和高性能目标的转向，电路小规模趋势的设计和供电电压的降低，使得微处理器受单粒子翻转的影响显著增强，软错误的发生更为频繁，而对于运行的航天器来说，任何错误的存在都可能导致巨大的灾难，软错误率反映了芯片出现故障或损坏的可能性。因此，对软错误的评估已经成为设计中必须考虑的重要问题。

针对 DSP 微处理器的单粒子翻转容易引起软错误的现象，为了降低软错误率以及减少对整体防护的开销，则需要对系统有针对性的进行防护。本文以 C6000 系列芯片为研究对象，研究了脆弱性评估的方法，并对研究中需要解决的关键问题进行了分析，最后对提出的评估方法进行了实验测试。主要的研究工作如下：

（1）提出了一种适用于 DSP C6000 指令集的可靠性评估方法，以静态汇编指令为研究对象，从单粒子翻转效应是否会对应用程序的输出结果产生影响方面对软错误率进行评估。通过对评估指标的分析，将对软错误率的评估转变为对脆弱性的评估，并指出评估中需要解决的关键问题。

（2）对当前的研究现状进行分析，并指出了当前评估方法的优点和缺点。基于软错误率与应用程序的紧密关系，以及当前研究方法中对应用程序可靠性评估的研究的缺乏性，根据对体系结构脆弱性指标的研究，提出了针对应用程序的脆弱性评估方法，并分析了其计算模型。

（3）在评估过程中，根据指令对输出结果的影响对指令进行分类，遍历汇编程序，对每种类型指令进行识别，通过建立指令链表和操作数的消费者-产生者链来完成对动态死指令的识别，并且依据每类指令的指令字二进制格式完成对 Un-ACE 位和 ACE 位的采集，结合计算模型完成对可靠性的评估，避免了传统评估方法中重复的注错和大量的仿真所带来的复杂度和巨大的开销。

（4）在对 DSP 的评估方法中，不同于以往的故障注入法，该方法不需要对微处理器的结构完全的了解，而是对 DSP C6000 指令集进行分析，对应用程序根据一定的划分规则进行模块划分，以模块为单位进行分析的，对于错误率高的模块便于有针对地进行防护，以降低模块的错误率以及防护过程中带来的资源开销。

通过对多组程序进行实验评估，验证了该评估方法的可行性，由于本文所提出的评估方法是针对指令集的，而不是局限于具体的体系结构，并且在实验过程中，操作简单，实验周期短，大大提高了评估的效率，因此具有良好的应用前景。

关 键 词：DSP 微处理器， 软错误， 单粒子翻转， 程序脆弱性因子， 可靠性评估

论文类型：应用基础研究类

ABSTRACT

With the rapid development of space and satellite technologies and the application requirements for Very Large Scale Integrated circuit (VLSI), the dependence on DSP and other integrated chips grows deeply. On the other hand, with the microprocessor design goals turning to low power and high performance, the trend of small-scale circuit design and the reduction of supply voltage make the microprocessor affected by Single event upset (SEU) more severely, and soft error occurs more frequently. As for the running of aerospace devices, the existence of any error may lead to a great disaster. The soft error rate reflects the possibility of failure or damage. Therefore, the evaluation of soft error should be an important issue to be considered in the design.

In view of the DSP microprocessor Single event upset causes the phenomenon of soft error easily. In order to reduce the soft error rate and the cost for the overall protection，targeted protection need to be done. Based on C6000 series chip as the research object, this paper studies the vulnerability evaluation method, and analyzes the key problems need to be solved in the study. Finally, the proposed evaluation method has carried on the experimental test. The main work can be summarized as follows:

(1) The thesis proposes an evaluation method applied to DSP C6000 instruction set. The research starts at static assembly instructions. Based on whether the output on the application program affected by Single event upset effects to evaluate soft error rate. Through the analysis of the assessment indicators, makes the soft error rate evaluation turn to vulnerability evaluation, and notes the key issues should to be addressed in the evaluation.

(2) The thesis analyzes the current research status, and points out the advantages and disadvantages of the current evaluation methods. Based on the close relationship between the soft error rate and application, and the lack of the current research on the application of reliability evaluation method, according to the study of the architecture vulnerability indicators, the vulnerability evaluation method is proposed for the application. And the calculation model is also analyzed.

(3) During the evaluation process, the paper classifies the instruction based on the output of the instruction, then traverses the assembler instructions and identifies each type instruction. The dynamic dead instruction can be identified by establishing the instruction list and operands consumer - producer chain. The collection of the Un-ACE and ACE-bits are competed based on the binary instruction word. Then the evaluation can be realized by combing with the calculation model. The method avoids the complexity and huge costs that cone from repeated injection and large number of simulations.

(4) In the evaluation method of DSP, different from the conventional fault injection, the method needn’t understanding of the structure of the microprocessor completely. And through analyzing the instruction set, the application can be divided into some modules according to certain rules. It is analyzed in each module. In order to reduce error rates and resource overhead comes from protection, the module whose error rate is high should be protected.

Through the multiple sets of experimental evaluation, the method is proved to be viable. The proposed evaluation method is aimed at instruction set. It isn’t limited to the specific architecture. And in the experiment, because of the simple operation and short cycles, the efficiency greatly improved. So it maybe applied widely.

Keywords: DSP microprocessor, Soft error, Single event upsets, Program Vulnerability Factor, Reliability evaluation

Type of Dissertation: Applied Basic Research

插图索引

表格索引

符号对照表

符号	符号名称
H	存储结构
b	存储结构的总位数
N	应用程序执行的总的时钟周期数
B_{ace}	存储结构的平均带宽
L_{ace}	ACE 位在存储结构的平均滞留时间
R	软件资源
I	指令总数目

缩略语对照表

缩略语	英文全称	中文对照
DSP	Digital Signal Processing	数字信号处理器
VLIW	Very Long Instruction Word	超长指令字
SEU	Single event upset	单粒子翻转
SDC	Silent data corruption	静态数据损坏
DUE	detected unrecoverable error	不可恢复错误
AVF	Architectural Vulnerability Factor	体系结构脆弱因子
PVF	Program Vulnerability Factor	程序脆弱因子
MTTF	Mean Time To Failure	平均失效时间
FIT	Failures In Time	失效率
IER	Intrinsic Error Rate	本征错误率
TVF	Time Vulnerability Factor	时序脆弱因子
ACE	Architectural Correct Execution	体系结构正确执行位
RTL	Register Transfer Level	平均失效时间
HAES	Hybrid AVF Evaluation Strategy	访存 AVF 评估策略
EPM	Error Propagation Model	错误传播模型
FDD	First-level dynamically dead instructions	第一级动态死指令
TDD	Transitively dynamically dead instructions	传递动态死指令

目录

第一章 绪论

1.1 引言

人类对于空间及外太空坚持不懈的探索促进了航天和卫星技术的迅速发展。近年来，航天和卫星技术呈现了新的发展特点，一是对信号在轨处理能力提高了更高的要求；二是卫星的发展趋向小型化、微型化、集成化方向。超大规模集成电路的应用需求也因此应运而生，其广泛应用使得上述发展成为可能。

对于空间电子仪器来说，它需要对光、电等多方面知识的全面使用，为了使得空间电子仪器能够满足空间环境的高标准的要求，则需要在多方面进行比较系统地研究，比如，可靠性评估问题。空间电子仪器作为航天器的重要部分，其发展必须适应航天技术的前沿，并为其提供强有力的支撑。“可编程逻辑器件、处理器以及存储器”这三类超大规模集成电路构成了大多数处理模块的核心结构，其中处理器主要包括通用的数字信号处理器 DSP、ARM 等，在航天器中有许多传感器并且这些传感器产生了大量的数据和动作，这就需要高速的 DSP 来处理这些大量的数据，因此 DSP 在航天应用中发挥着相当重要的作用。

就目前情况来讲，卫星技术在国民经济以及国防信息化建设中的突出作用已经不能忽视，这就对空间电子仪器的处理能力提出了更大的挑战，随着技术的进步，以卫星平台和载荷为典型代表的空间飞行器电子系统与 DSP 等超大规模集成电路的关系越来越紧密。

然而随着超大规模数字信号处理器器件在空间电子仪器的广泛应用，星载信号处理系统的任务要求日益复杂，功能和性能要求日益提高，加之复杂的外太空环境，比如宇宙射线、范艾伦辐射带等对航天器的影响，导致航天器发生故障和出现错误的几率成倍增长，其中由单粒子效应引起的故障和错误是一个非常重要的方面，对于运行中的航天器来说，任何错误的存在都可能导致重大的事故和巨大的损失。

例如，在上世纪时期的太阳质子事件，研究统计美国的卫星中随机存储器就出现了 200 多次单粒子翻转情况，中国的卫星也在上世纪 90 年代因为单粒子现象导致卫星出现故障，而不能够正常的运行；还有一些卫星也因为单粒子现象而不能继续使用[1]；就在前几年，我国就有颗卫星在轨道上不到一个月的时间，DSP芯片就发生单粒子翻转 60 余次……。

研究表明，大约百分之七十的故障都是由高能粒子导致的，其中百分之五十的错误都是由单粒子翻转引起的，其中出现故障次数[2]的统计如表 1.1 所示。

表 1.1 星载设备上出现的故障数据统计情况

故障类型	出现次数	故障比率/%
电子诱发电磁脉冲（EIEMP）	293	18.44
静电放电（ESD）	215	13.53
单粒子翻转（SEU）	621	39.08
其他	460	28.95
总计	1589	100

从我国 2006 年至 2010 年发射的若干颗卫星的测试情况来看，SRAM 型 FPGA、DSP 单粒子翻转已经多次引起空间电子仪器的功能故障。超大规模集成电路的单粒子故障已经是制约我国空间电子仪器性能提高的重要因素。空间任务、卫星、高能物理实验以及其他领域对使用抗辐射电路需求的驱动，使得对空间电子仪器超大规模集成电路的软错误防护方法的研究不断增加。而空间环境自身的辐射效应（尤其是单粒子效应）与超大规模集成电路间目前看来难以调和的矛盾成为设计者必须面对的问题。

单粒子翻转次数的增加将会导致单粒子效应引起的故障和错误增加，因此需要对单粒子翻转进行防护，以减少单粒子效应带来的错误和故障，对单粒子效应有选择的进行防护的前提是对系统进行软错误可靠性评估。

1.2 DSP处理器介绍

数字信号处理器 DSP（Digital Signal Processing）是对信号和图像实现实时处理的一类高性能的 CPU，从上世纪 80 年代初，第一片 DSP 的问世，其特有的稳定性、可重复性、可大规模集成等优点为 DSP 的发展带来了巨大的机遇，目前，DSP 已广泛应用于通信、航空航天、工业测量、控制、生物医学工程等重要领域。

本文主要是针对TI公司的C6000 系列芯片进行研究的。C6000 采用了超长指令字VLIW（Very Long Instruction Word）[3]的体系结构，在C6000 的CPU内设置多个并行操作的功能单元，指令是按照流水线的形式进行操作的，在一条指令执行时，另一条指令可以进行取指，相互之间不会产生干扰，最多可以 8 条指令同时执行操作，相比于串行操作的处理器而言，执行速率明显快很多，对于这种流水线的操作，大大提高了计算的速度。

DSP 内部设置有硬件乘法器来完成乘法操作，这是不同于其他处理器的关键因素，传统的微处理器使用的是冯·诺伊曼结构，数据和程序使用同一条总线来完成传输，在指令执行过程中只采用串行的操作，即指令只能依次顺序一条一条

执行，而不能同时进行多条指令的操作。对于 DSP 微处理器而言，使用的是哈佛结构，程序与数据总线是分开的，可以同时进行多个操作。

C6000 的指令集中的寻址方式全部采用的是间接寻址，寻址方式通常是指某一个 CPU 指令系统中，规定的寻找操作数所在地址的方式，或是通过什么方法找到操作数，寻址方式给出了 CPU 访问数据空间的方式，它与微处理器的硬件结构密切相关。在进行字节寻址时，可以充分使用存储器资源。该指令集中存在着两种寻址模式分别为循环寻址以及线性寻址，默认情况下都认为是线性寻址，它主要都是通过寄存器存放线性指针完成的。

下面以 C6713 芯片为例介绍 DSP 的内部结构，如图 1.1 所示描述了 C6713 的功能框图，该处理器主要由 CPU、存储器以及集成外设所组成。该处理器的主要特点是可以进行浮点运算。

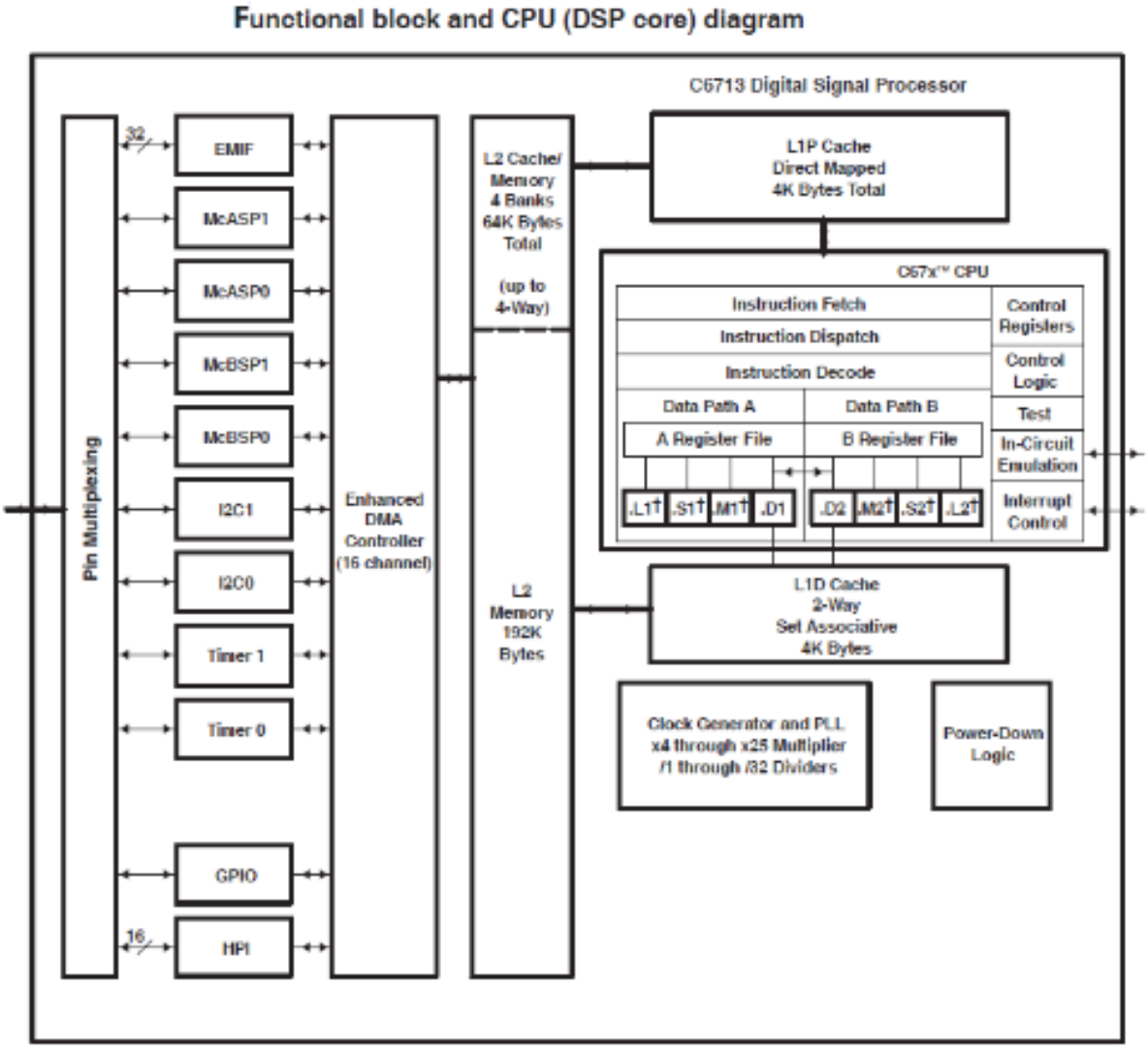


图 1.1 C6713 功能框图

中央处理器的主要功能是：C6713 的 CPU 是最新采用 Veloci TI 体系结构的 DSP 芯片。Veloci TI 主要采用超长指令字结构，该结构具有高性能的特点，在运行过程中程序可以并行执行，在处理器中由多个寄存器共同起作用，从而指令执行的效率很高，可以同步完成多个操作。

DSP 微处理器的运算、控制中心为 CPU，主要采用“流水线”的工作方式，大大提高了微处理器的速度，C6000 中所有指令的执行都是按照流水线的取指、译码以及执行这几个阶段进行的。

在 C6000 系列芯片的 CPU 中存在着两个数据通道，这两个通道内都存在有四个功能单元，以及 16 个寄存器，其中有两个乘法单元（.M）与六个逻辑单元相并行。汇编语言中的每条指令都必须在确定的单元中执行，其中与乘法有关的指令都在.M 单元内执行，.L 和.S 单元中大都执行与算术逻辑运算有关的指令，对于数据存储器地址的指令，则在.D 单元中执行。

C6000 的指令类型包含数据处理类型指令、数据传输类型指令、跳转指令以及空操作类型指令，每一条指令都有确定的执行单元，都有单独的指令字的二进制格式，存在着一一映射的关系。

DSP 为 CMOS 工艺器件，它依靠内部的存储器、寄存器、地址程序译码单元、乘加单元等共同完成用户定制的信号处理功能，DSP 通常由通用寄存器、程序和数据存储器、L1 程序 Cache、数据 Cache、算术逻辑单元、外设以及控制器组成。单粒子效应主要影响 DSP 的寄存器、L1 程序 Cache、数据 Cache、程序和数据存储器等存储型单元。

1.3 影响处理器的可靠性因素

单粒子翻转导致设备出现故障，主要表现为硬错误和软错误两种情况，硬错误指的是器件的永久性的损坏，而软错误不会导致器件本身损坏。目前大部分的研究是针对软错误进行的，软错误是影响处理器可靠性的主要因素。

1.3.1 软错误的概述

根据对摩尔定律的归纳，指出集成电路上的电路的数目，每隔十八个月就会增加一倍，其性能也同样增加了一倍。这也表明，半导体设备，尤其是微处理器的功能和性能将得到快速的发展，然而每次技术的成功进步，总是会带来一些新的问题。为了满足更多的用户对系统的高密度、高性能和低功耗的要求，则需要不断地降低芯片的尺寸以及供电电压，微处理器受辐射的影响越来越明显。研究表明，由单粒子翻转引起的软错误已经成为未来微处理器设计面临的问题之一。

单粒子翻转SEU（Single event upset）[4]是由太空环境中的高能粒子引起的，这些高能粒子包括封装材料中的α粒子、宇宙射线的中子等，当集成电路被高能

粒子所轰击，由于受库仑力的影响，高能粒子与硅晶格会出现电荷沉积现象，当这些电荷累积到一定数量时，有些逻辑器件，像SRAM中的逻辑单元中的存储位就会出现 0 或 1 的翻转[5]，图 1.2 为单粒子翻转的示意图。单粒子翻转不会导致器件出现永久的错误，因此称为瞬时错误或软错误。

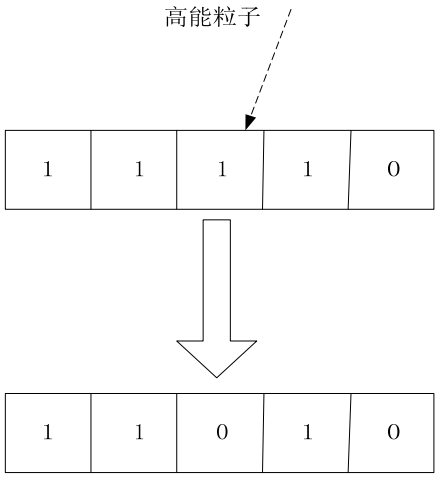


图 1.2 单粒子翻转

过去的研究中发现，集成的器件中出现的软错误主要是来源于三种情况。第一种主要是α粒子以及存在于包装中的钍导致的，第二种是来自于射线中的中子经过与硅核作用后出现的二次离子，第三种则是来源于射线中的低能中子束通过与同位素硼的作用。

单粒子翻转导致器件出现软错误的概率不仅与粒子的流通量有关，还与电路的特性密切相关，其中粒子流通量由物理环境所决定。例如在海拔 1.5km的地方，相对于海平面来说，宇宙射线产生的中子流通量比其高达 3 到 5 倍。中子流通量随海拔高度和纬度变化的情况分别如图 1.3 和图 1.4 所示[6]。电荷的存储数量、脆弱区域的截面面积以及电荷的收集效率等相关电路参数，是影响设备的软错误率的变量。由于设备尺寸的变小，每个设备中电荷越少，粒子轰击导致软错误发生的概率就有可能会越大，但是减少横截面积使粒子攻击发生在特定的设备上不太可能，因此每个锁存器或SRAM单元的软错误率在一定的海拔高度预计将不会改变或者稍微下降。然而在系统没有采取错误检测和校验机制的情况下，晶体管数量的增加，无疑将会导致芯片的错误率增加。

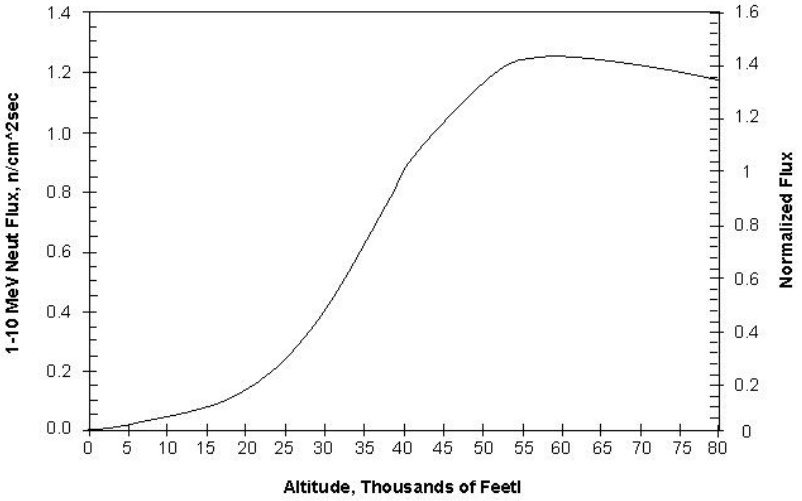


图 1.3 中子流通量随海拔高度变化的情况

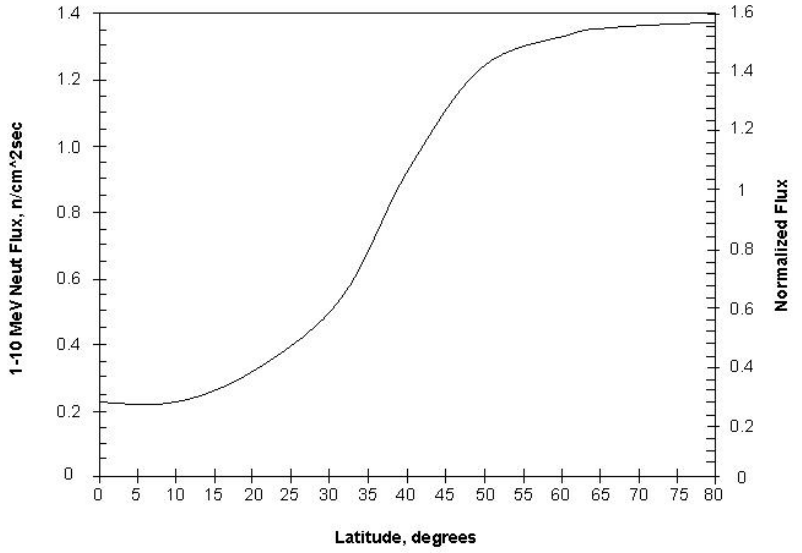


图 1.4 中子流通量随纬度变化的情况

1.3.2 软错误的分类

在系统操作中，不同的故障，在系统的影响上是不一样的，根据故障对系统的结果的影响，以及系统中是否有软错误检测和校验机制，可以把故障分为几类，如图 1.5 所示，并依次对每类故障进行了分析。

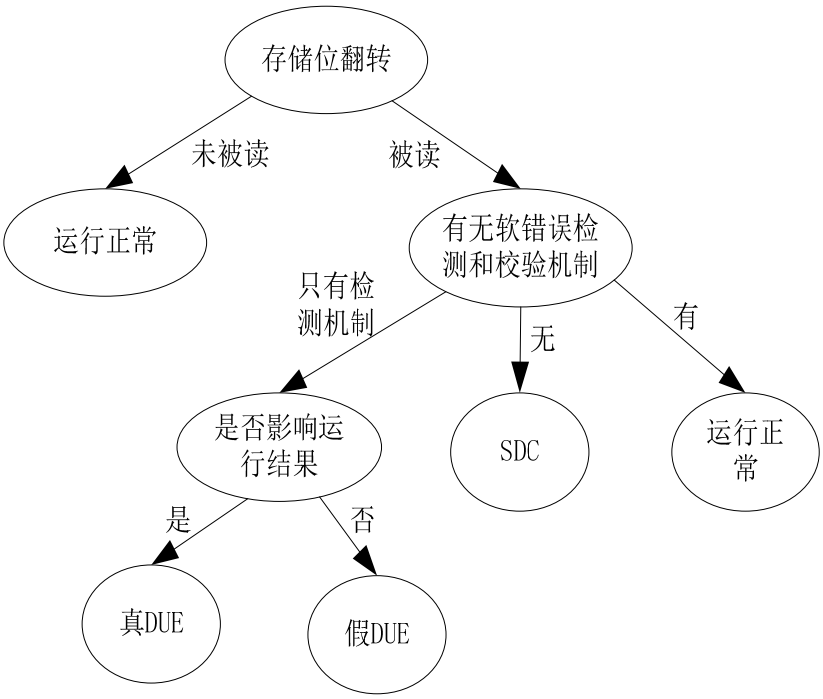


图 1.5 软错误的分类

良性错误：如果存储位发生了翻转，而在后续却没有被读，对系统不会造成影响，这种软错误称为良性错误，系统可以正常运行；

SDC故障：如果存储位发生了翻转，并且在后续系统执行中被读，然而却没有设置相应的软错误检测和校验机制，则单粒子的存储位翻转会导致系统执行结果出错，这种软错误称为SDC（Silent data corruption）故障[7]；

如果存储位发生了翻转，并且在后续系统执行中被读，但是电路中存在软错误检测和校验机制，则存储位即使发生了翻转，经过软错误检测以及校验后，能够使得程序正确的执行，进而系统可以正常运行；

DUE故障：如果存储位发生了翻转，并且在后续系统执行中被读，此时电路中只有软错误检测机制，而不存在校验机制，并且发生软错误的情况下，系统的输出结果出错，则认为该软错误为真DUE（True detected unrecoverable error），否则称这种软错误为假DUE[8]。

通过实验研究表明，所有的软错误并不都会使得系统的运行结果出错，微处理器体系结构固有特性能够屏蔽相当一部分的软错误[9][10]。例如，对于NOP指令来说，除了操作码字段以外，其他存储位发生翻转，程序的执行结果却不会受到影响。因此，Mukherjee等人[10]在对软错误导致应用程序输出结果出错的概率计算过程中，提出了体系结构脆弱因子AVF（Architectural Vulnerability Factor）的概念。由于不同的应用程序受软错误的影响不同，因此需要对存储部件有针对性的进行防护，以减少发生故障的概率，由于受到流水线机制、指令集的限制，在多数情况下，并不是所有的软错误都是能看得见的，即软错误对系统的影响不能够从系

统应用程序的输出结果直接看出来。

1.4 本文主要工作和结构安排

1.4.1 本文主要工作

为了对 DSP 处理器的单粒子翻转引起的软错误有针对性地进行防护，本文提出了一种针对 DSP 应用程序的脆弱性的计算方法，该方法从静态的汇编指令出发，根据指令执行结果对应用程序的输出结果是否产生影响，将指令分为 ACE 指令和 Un-ACE 指令，其中 ACE 指令是指指令的执行结果对应用程序的输出结果会有影响，而 Un-ACE 指令的执行结果则不会影响应用程序的输出结果。首先对 DSP 汇编程序进行模块划分，以模块为单位对指令进行分类，针对不同指令类型的指令数目进行统计，然后根据不同类型指令识别出应用程序中的 ACE 位，最后根据软错误程序脆弱因子 PVF（Program Vulnerability Factor）的计算公式，对 DSP 应用程序的脆弱性指标 PVF 进行评估。主要内容为：

首先，该可靠性评估方法适用于对 DSP 的设计评估，通过对汇编指令集的设计和行为分析，提出了一种不依赖于具体的体系结构的评估方法，克服了现有的只针对具体体系结构评估方法的缺陷。

其次，本文将指令行为作为研究的切入点，以应用程序中的指令为目标对象，从应用程序方面评估可靠性，而不是仅仅只针对位翻转方面进行研究的。本文主要对静态汇编指令集进行分析，通过对操作码的识别完成对 NOP 指令等的分类寻找，在对动态死指令识别时建立了指令链表和操作数的消费者-产生者链。针对不同类型的指令分析对应的 Un-ACE 位，完成 ACE 位的采集。

最后，该方法相比于传统的方法来说，实验时间短，工作效率高，资源开销少，降低了评估的局限性。传统的方法需要依据具体的体系结构实施故障注入，进行多次注错仿真，而本文提出的方法只需要对应用程序进行代码分析，是从对指令集层面进行评估的，不需要深入了解处理器的内部结构，对代码的分析在性能和时间上都优于传统评估方法，因此大大提高了评估效率。

1.4.2 本文结构安排

本文主要对 DSP 的应用程序进行了程序脆弱性评估研究，主要章节内容安排如下：

第一章介绍了论文的研究背景及意义，分析了 DSP 微处理器的结构，以及影响可靠性的主要因素即软错误，然后对论文的主要研究内容以及组织结构进行了梳理。

第二章对当前的评估方法进行了分析，首先对处理器的评估指标以及体系结构脆弱性因子的概念进行了介绍，然后分析了目前关于脆弱性因子评估的常用几

种计算方法，讨论了这些评估方法的不足之处。

第三章主要对本文提出的评估方法进行了分析，首先根据体系结构脆弱性指标引出本文的评估公式，并举例说明公式的计算方法。然后分析了评估方法的实质，并提出了需要解决的关键问题以及相应的解决方法，主要包含：（1）应用程序中 ACE 指令和 Un-ACE 指令的判断；（2）对不同类型指令的分类识别；（3）不同类型指令包含的 Un-ACE 位位数的分析。

第四章提出了可靠性评估的具体实现，通过和特定的指令集相结合，介绍了不同指令的识别方式，ACE 位位数的分析，最后对 DSP C6000 系列指令集的应用程序进行模块划分，对每个模块的脆弱性进行了评估和分析。

第五章全面概括总结了全文的主要工作，并对以后的研究方向进行了展望。

第二章 微处理器可靠性评估研究现状

由于微处理器在航天领域的推广，微处理器的软错误评估变得越来越重要，可靠性评估的研究进而也被越来越多的机构组织重视。本章开始先介绍评估微处理器可靠性的常用指标，然后指出脆弱性因子的计算是评估的关键，最后分析了当前评估可靠性的现状，并对几个著名的项目进行了比较。

2.1 处理器可靠性评估指标

平均失效时间 MTTF（Mean Time To Failure）和失效率 FIT（Failures In Time）被认为是当前系统软错误率评估的常用指标。系统平均发生一次失效的时间称为平均失效时间 MTTF，例如，如果系统每隔 3 年失效就产生一次，则系统的 MTTF 为 3 年。系统每运行10^9小时，失效平均发生的次数称为 FIT，即系统失效率，两者成反比关系：

$$MTTF = \frac{10^9}{24_{(hour)} \times 365_{(day)} \times FIT} \tag{2-1}$$

当MTTF为 1000 年时，大约相当于 114 FIT（10^9/（24*365*1000））。所说的零错误率对应的是零FIT失效率和无穷平均失效时间MTTF，设计者常用失效率FIT来描述SDC和DUE故障的概率[11]，因为失效率FIT可以叠加，而MTTF不能够叠加。

系统中各个存储部件的FIT_i的总和称为系统总的FIT_{system}，即：

$$FIT_{system} = \sum_i FIT_i \tag{2-2}$$

而对于某部件i来说，其本征错误率IER（Intrinsic Error Rate）和该部件的脆弱因子VF_i (Vulnerability Factor)的乘积称为该部件的失效率FIT_i[12]，FIT_i可以表示为：

$$FIT_i = IER_i \times VF_i \tag{2-3}$$

某部件中的存储位发生翻转的概率用来表征该部件的本征错误率IER_i，它受到具体的生产、工艺、电路、设计等因素的影响。部件的脆弱因子VF_i为时序脆弱因子 TVF（Time Vulnerability Factor）和体系结构脆弱因子 AVF（Architectural Vulnerability Factor）的乘积，即：

$$VF = TVF \times AVF \tag{2-4}$$

对于时序脆弱因子 TVF 来说，可以认为是部件运行过程中，部件中软错误发生的时间所占的比例，随着电路的特性而变化，例如在 RAM 单元中，该值为 1，而在锁存器和动态电路中，该值有所不同，约为 0.5。由单粒子翻转导致系统出现

的外部可见的错误的概率称为体系结构脆弱因子 AVF。因此，系统的总失效率可以表示为：

$$FIT_{system} = \sum_{i}(IER_i \times TVF_i \times AVF_i) \tag{2-5}$$

对于给定的结构，IER 和 TVF 可以当作常数来计算，因此，进行微处理器评估的可靠性研究，关键在于计算出体系结构脆弱因子 AVF。

图 2.1[13]表明了AVF对SDC FIT的影响。对图中的数据进行分析，可以发现，在 2005 年如果系统的AVF为 100%，那么为了达到其设定的目标，80%的存储位都需要被保护，而如果系统的AVF为 10%，从图中可以看出，完全达到要求，则不需要采取保护措施即可。用同样的方式可以对 2010 年进行分析，则为了达到目标，需要对 80%的存储位进行保护。

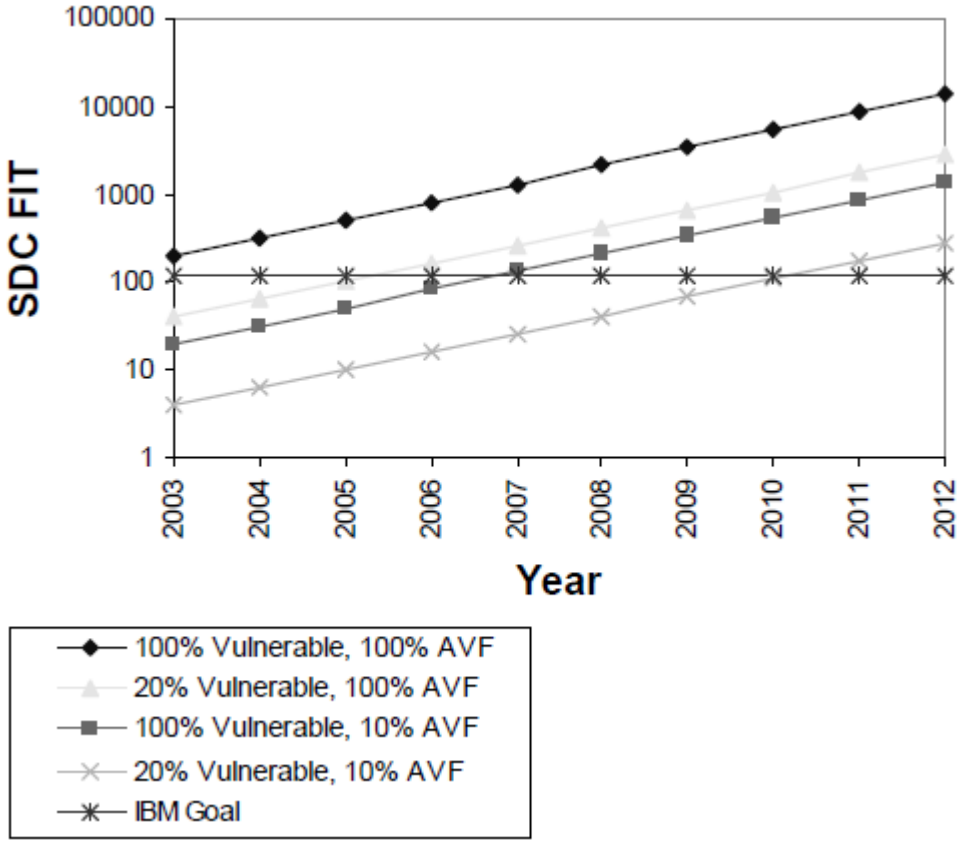


图 2.1 AVF 对 SDC FIT 的影响

根据 AVF 的定义，是指单粒子翻转引起应用程序输出结果出错的概率，在系统中，如果存在着错误检测以及校验机制，则系统能够正常的运行，系统不会出错，而如果系统中没有错误检测机制，同时也不存在错误校验机制，那么其 AVF 与该存储部件的功能有关。就拿分支预测位来说，该存储位发生单粒子翻转，应用程序的输出结果却不会受到影响，所以 AVF 的值为 0，而如果说发生反转的存

储位是指令计数器上的存储位，那么程序有可能会执行出错，甚至影响程序的结果，因此 AVF 为 100%。为了达到设计者的目标，减少器件的故障率，降低防护的开销，因此对脆弱性因子进行评估显得十分重要。

2.2 体系结构脆弱因子AVF

体系结构脆弱因子AVF是衡量单粒子翻转引起程序输出结果出错概率的指标，为了精确计算微处理器的AVF值，MuKherjee[10]等人引入了体系结构正确执行位ACE（Architectural Correct Execution）的概念，通过分析流过流水线上的ACE位在运行中所占的时间比例完成对AVF的评估。

根据软错误发生后，系统是否会出现故障或损坏，通常将体系结构状态位分为两类：1）影响体系结构正确执行的状态位，即 ACE 位；2）不影响体系结构正确执行的状态位，即 Un-ACE 位，在没有对系统进行防护的情况下，ACE 位发生单粒子翻转产生的软错误都会使程序的最终输出结果出错。在应用程序执行时，某个部件的AVF值可以表示为ACE位在该部件的停留时间所占整个部件工作时间的比重。假设存储位只有 ACE 位和 Un-ACE 位两种状态，必须为其中一种，那么设一个存储结构 H 的总位数为 b，应用程序执行的总的时钟周期数为 N，则该部件的 AVF 的计算公式为：

$$AVF_H = \frac{\sum_N ACE\ bits\ in\ H}{b \times N} \tag{2-6}$$

可见 AVF 的计算重点在于确定哪些位是 ACE 位和哪些位是 Un-ACE 位，然而在计算过程中，通常难以把握存储位是否处于 ACE 状态，因此，在对 ACE 位的识别过程中，通常先找出处于 Un-ACE 状态的位。主要思路是假设所有存储位都为 ACE 位，通过分析尽量识别出程序执行过程中的哪些位是 Un-ACE 位，则剩下的未识别出来的则认为都是 ACE 位。

2.3 AVF研究现状

目前，微处理器的体系结构脆弱性 AVF 评估的方法主要从仿真或者系统操作过程中收集信息，然后观察应用程序的运行结果，进行脆弱性因子的估计。评估中所运用的主要方法包括故障注入法和 ACE 位分析法。

2.3.1 故障注入法

在 20 世纪 70 年代初期人们首次提出故障注入方法，在那以后开始使用故障注入法来指导容错系统的设计，在这方面一直受到设计者的关注。80 年代中期，故障注入技术的研究才逐渐开始有所起色，90 年代后，各行各业与之相关的研究才逐渐流行，并进一步深入，也相应有一些应用投入到实践中，与此同时，在单

粒子效应的故障注入方面也进行了不少研究。

文献[14]中，对故障注入法的具体原理进行了总结，指出该方法主要是针对具体的目标系统，根据其体系结构选择对应的故障模型，通过对这些故障模型实施故障注入，然后对系统进行周期性观察和监视，最后将故障模型与无故障模型的结果进行对比。

故障注入是一个循环过程，主要包括故障模型的选择，对目标系统实施注入故障、对目标系统进行监视，最后对实验结果的分析，具体过程如图 2.2 所示。

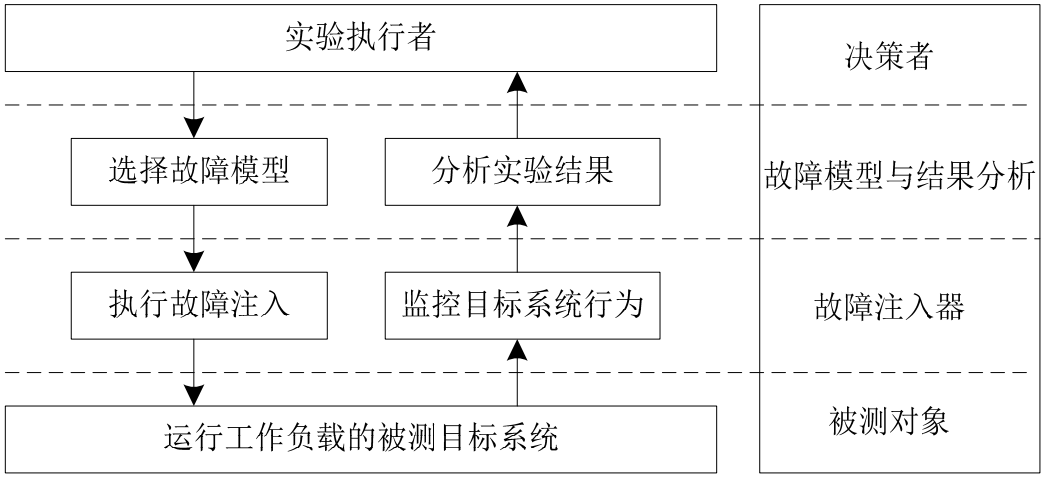


图 2.2 故障注入原理框图

故障注入有许多不同的分类方法，其中本文根据实验时故障注入的目标系统进行分类，包括基于模拟的故障注入方法和基于物理的故障注入方法。

基于模拟的故障注入方法可以在晶体管开关级、逻辑级、功能级等实施故障注入实验。晶体管开关级故障注入主要用于观察集成电路芯片的内部结构能发生的故障，逻辑级故障注入法主要就是采用 0、1 等这些与逻辑门相关的故障模型进行注错，观察故障注入的结果，然后通过和无故障情况下的比较，分析其实验结果。功能级故障注入的模型难以选定，在故障注入过程中需要分析系统的功能模块有针对性地实施故障注入。

基于物理的故障注入方法通常又可以分为硬件注入法和软件注入法。硬件故障注入一般为管脚级的故障注入[15]，硬件注入法产生的效果近似于软错误发生的场景，具有精度高、易于控制的优点，但是代价高，管脚级的故障注入容易对芯片造成损坏。软件注入法[16][17][18]注错的模拟实验原理，主要是对程序中的代码进行删除等操作的改变，软件故障注入法不能够直接读写流水线寄存器的内容。

目前，应用最广泛的一种故障注入评估方法就是基于仿真方法的软件故障注入，这种方法试验中处理器和注错均采用软件仿真进行，该方法的本质主要是将随机的位翻转引入到寄存器传输级RTL(Register Transfer Level)的部件模型中进行

研究，通过监视处理器的一定数量的仿真周期操作，分析位翻转产生的错误在处理器中是被覆盖了还是隐藏了，很多处理器均以该基本思想进行可靠性评估，只是根据不同处理器的结构，在实现过程中有所改进。E. W. Czeck[19]等人通过对IBM RT PC进行仿真注错，注错过程使用的是RTL部件模型，注错的时间间隔是每个周期，通过研究发现软错误中大约有 70%左右的被覆盖了，研究中还发现可靠性与测试程序中的指令类型以及处理器的执行机制之间存在着很大的关系；文献[20]中，通过对TRIP的 32 位精简指令集处理器仿真注错，使用的也是RTL部件模型，从评估的结果可以看出，屏蔽掉了 34%的软错误。

下面介绍两个故障注入评估方法的研究项目。

（1）Nicholas J. Wang等人通过对Verilog模型进行故障注入研究了类似于Alpha 21264 和AMD Athlon的现代处理器的可靠性[21]。项目主要对锁存器、寄存器等存储部件进行了研究，不仅对故障屏蔽的程度或瞬态故障作为软件可见错误被屏蔽的比例进行了估计，还对处理器的脆弱性部位进行了识别。研究过程中通过对Verilog模型多次执行故障注入，注错过程中一次使用大约三百个注错点，然后观察注错后的程序运行情况，同时分析没有实施故障注入的模型情况，比较得出处理器中程序的运行结果与存储位发生翻转的关系。观察在 10000 个监视周期内处理器是否出现错误的状态情况，没出现错误状态的情况存在两种可能性，一是错误本身被屏蔽了，不能够影响程序的正常执行，二是错误有可能在观察的时间段内还没有表现出来，因此无法从表面上看出，但是这种错误对程序可靠性评估的正确性具有潜在的影响。

（2）Giacinto等人以DLX处理器以及Alpha处理器为研究对象，分别分析了它们的可靠性[22]，其中这两种处理器模型分别是嵌入式以及高性能的处理器的主要例子。在实验过程中，通过选取两个注错模型进行故障注入，其中一个模型是对时序元件中的状态位进行随机的翻转，在翻转之前需要先随机选定一个时间周期，另一种模型主要是翻转其网表结点，仿真在RTL结构模型进行。

Giaclnto 等人对 CPU 的运行状态进行了定义，主要是和无故障注入时的状态进行区分，根据程序是否正常运行，这四种状态为 1）崩溃状态，2）不完全执行状态，3）数据错误状态，4）正确运行状态。通过对基准测试程序 SPEC2000 中的无损压缩程序进行测试来对处理器的存储部件进行评估。

研究表明基于模拟的故障注入方法操作方便，好实现，不需要设计者充分的了解所需要评估的目标系统，不过该方法也有其缺点。

第一，该方法需要通过大规模的仿真故障注入才能实现，评估性能低，在注错过程中，需要进行结构级的仿真，速度慢、周期时间长，影响了评估的效率。由于该方法对注错的空间的需求很大，那么可靠性分析时间特别长，在一定时间

段内难以实现，在不同的处理器中，软错误会呈现出不同的形式，因此不能使用同一个模型对所有的处理器进行评估。

第二，由于选取的注错点的随机性以及注错次数的有限性使得评估的精确性受到一定的限制，另外，为了提高软件注错的效率，在运行特定的周期后，将每次注错的结果与无故障模型的结果进行对照比较，而不是在应用程序结束后才进行观察每次注错的结果，这种情况下观察到的实验结果是不准确的，因为有些软错误并没有立刻表现出来，实际上却存在于系统中，而这种软错误的存在也会对系统的评估的准确性产生一定的影响。

第三，在用传统的方法对 DSP 处理器评估设计中，通过以特定的处理器结构进行注错研究，可靠性的评估需要多次注入不同的体系结构中来完成不同处理器的评估，由于注错目标是具体的体系结构，因此不具有通用性。

此外，当前的研究大多数都致力于对电路级层面上的软错误进行建模和检测[23][24][25]，然而随着近年来对电路级的软错误的深入研究，可以发现，很多错误在体系结构级就被屏蔽了，而没有影响执行结果，因此在一定程度上可能会降低硬件保护的成本，因此有必要在体系结构[26][27]层面上对处理器进行可靠性的评估。

2.3.2 ACE位分析法

ACE 位分析法主要是对存储位进行跟踪分析，观察存储位上发生单粒子翻转是否会导致系统运行结果出错。ACE 位分析法需要从微体系结构和体系结构层面上对处理器进行分析，记录 ACE 位状态所占的时间比。ACE 位分析法可以分为分析模型法以及性能模型法。

（1）分析模型法

分析模型法是在位流信息保持不变而且整个结构中没有复制的情况下，使用 Little′s Law[28]计算出一个结构中ACE位停留的平均数目，进而得出该结构的AVF值。

Little′s Law 可以用公式 $N = B \times L$ 来表示，其中 N 代表某个结构的平均位数，B 代表结构中每个周期的平均带宽，L 为每个单独的位流过该结构的平均延迟，将该方法扩展到 ACE 位分析法，可以得到该结构的 ACE 位平均数目等于每个 ACE 位在结构上的平均带宽 B_{ace} 与 ACE 位在结构中的平均滞留时间 L_{ace} 的乘积，则该结构的 AVF 计算公式为：

$$AVF_H = \frac{B_{ace} \times L_{ace}}{\text{结构中的总位数}} \tag{2-7}$$

比如，指令队列结构，其平均带宽 B_{ace} 表示每周期执行的指令数乘以每条指令中的 ACE 位位数，平均滞留时间 L_{ace} 表示指令序列中 ACE 位停留的时钟周期。对

于分析模型来说，主要使用于早前处理器不能够给出结构模型与性能模型时。其中文献[29]中的 softArch 工具则是采用上述的模型进行评估可靠性的。分析模型主要是对那些出错会导致应用程序结果输出错误的位进行追踪，出现的错误有可能是由前面的错误导致的，也有可能是当前存储位本身出现了错误。

（2）性能模型法

性能模型的主要思想就是识别流过流水线上的存储部件中的ACE位和Un-ACE位，对一个存储位来说，其AVF的值就是整个过程中处于ACE状态的时间区间。在判断ACE位时，可以认为存储位的所有生存周期区域内都是ACE位，然后确定其中的Un-ACE位，进而得到ACE位。Biswas等人在对存储部件进行脆弱性评估时就通过使用生命周期分析技术[30]。使用这种方法计算AVF的难点在于对Un-ACE状态位的确定。

生存期分析方法就是对流水线上的存储位的生存周期进行详细的划分，根据对存储器的操作，在程序执行过程中，一个寄存器有可能会被读取和写入多次，根据不同的操作，将存储器的每一位的生存期分为几种阶段，如 idle、fill-read、read-write 等，如图 2.3 所示。

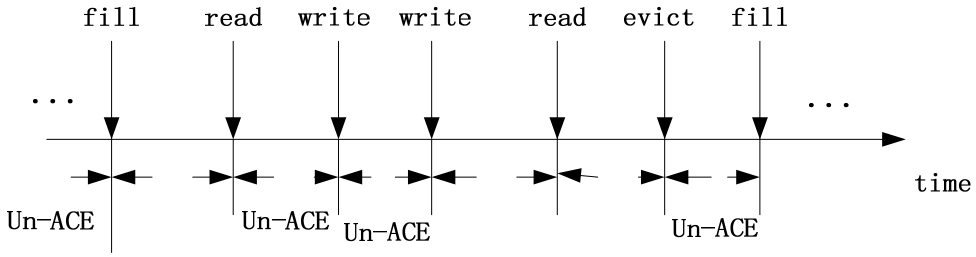


图 2.3 存储器生存周期

由于图 2.3 中显示的没有数据被读出，或写回到下一级存储层中，则 idle、fill-read、write-write 和 evict-fill 区间，即使存储位发生了单粒子翻转，但也不会影响程序的执行结果，因此，认为在该区间存储位处于 Un-ACE 状态，而处于 fill-read、read-read、write-read 以及 read-evict 阶段虽然没有数据读或写到处理器或其他存储结构，但不一定会导致程序执行结果出错，比如动态死指令中读出的数据，在后续执行过程中从未被使用，并且不会对输出结果产生影响，因此还需进一步判断是否为 ACE 状态。不同结构中生存期中 ACE 和 Un-ACE 的分类如表 2.1 所示。

表 2.1 生存期 ACE 和 Un-ACE 分类

处理器结构	生存期分析		
	ACE	Un-ACE	Unknown
Write-through data cache	Fill-to-read, read-to-read, write-to-read	idle, fill-to-write, fill-to-evict, read-to-write, read-to-evict, write-to-write, write-to-evict, evict-to-fill	fill-to-end, read-to-end, write-to-end
Write-back data cache	fill-to-read, read-to-read, write-to-read, write-to-evict, write-to-end, some of Un-ACE components can be conditionally ACE(see prose)	idle, fill-to-write, fill-to-evict, read-to-write, read-to-evict, write-to-write, evict-to-fill	fill-to-end, read-to-end
Data translation buffer	fill-to-read, read-to-read	idle, read-to-evict, evict-to-fill	fill-to-end, read-to-end
Store buffer	fill-to-read, fill-to-evict, fill-to-end, read-to-read, read-to-evict, read-to-end	idle, evict-to-fill	none

为了运用公式（2-6）计算一个结构的 AVF，下面的信息必须得到：

1）在该结构内，应用程序执行过程中，所有的 ACE 状态位的停留周期之和；

2）能够观察到的 ACE 位停留时间的总的执行周期；

3）硬件结构的总位数。

根据对性能模型的分析，我们可以得到上述的全部信息，特别地，如果是针对流水线上的指令信息进行的研究，则评估过程分为三个阶段进行：

1）记录指令在特定的硬件结构中的停留时间，主要对象是流水线上的指令，当指令执行结束，要脱离流水线之时（被提交或释放），需要对该硬件结构进行更新，比如使用停留周期等各种信息来完成；

2）如果指令提交了，则需要在下一个分析窗口来完成该指令的分析，判断该指令是否为动态死指令，或者判断有些位是否被逻辑屏蔽了；

3）根据 1 和 2 获得的信息，运用公式（2-6）计算出 AVF。

在使用此方法对微处理器评估时，需要对体系结构和微体系结构有很深入的了解，这样才不会造成 AVF 的计算大于实际值，相比于统计故障注入方法的不同

之处在于，ACE 性能模型分析方法的速度特别快，通过一次实验可以对很多处理器结构同时进行评估，并且可以运行上万个周期。

目前大部分的评估工具都是没有公开的，而只有AVF评估框架Sim-SODA[31]是开放的，该方法就是基于性能模型进行计算评估的。Sim-SODA是基于Alpha-21264[32]结构提出的，针对高性能超标量处理器的评估工具，它由美国佛罗里达大学的研究人员提出，实验中选用SPEC2000[33]中的测试程序对处理器进行评估的，它能够评估大部分基于Alpha-21264 的微处理器，该工具为了应付评估过程中的边缘效应，还引入了Cooldown技术，由于Alpha指令集与DSP指令集不同，因此在不能用Sim-SODA来完成对DSP微处理器的评估。

在现有的评估方法的基础上，成玉等人提出了一种改进的评估策略HAES（Hybrid AVF Evaluation Strategy）[34]，该评估策略主要是针对存储器等一些核心存储部件进行的研究，改进的主要思想是将访存操作分析和指令分析相结合来实现更精确的评估，主要过程为判断哪些存储位处于ACE状态，判断ACE位的同时，要统计出这些状态总共的执行时间也就是周期数，进而完成AVF的评估。相比于以前的分析方法中认为操作中存储单元只有两种状态：即ACE和Un-ACE状态，而该分析方法划分更细，认为还有一种不能够识别的状态，即未知（Unknown）的状态，这样以来，将会识别出更多Un-ACE状态的存储位，估算值更精确。该方法将评估策略与通用的模拟器结合在一起，评估框架如图 2.4 所示。

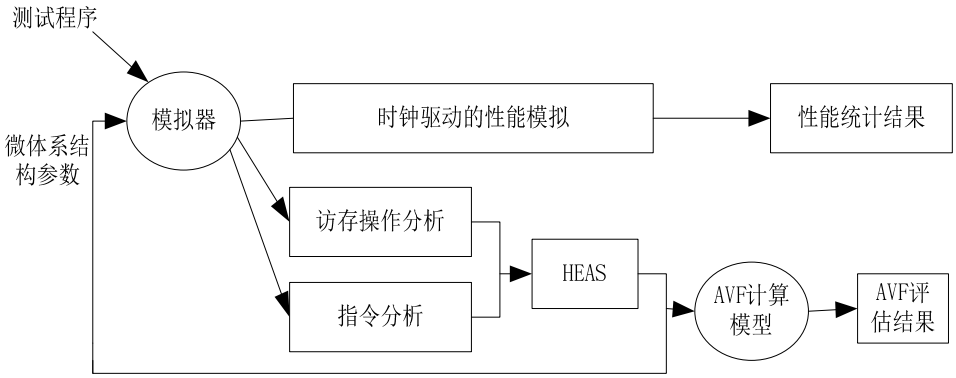


图 2.4 AVF 评估框架

文献[35]中提出了一种基于占有率的 AVF 计算方法，该方法把程序运行过程中，不同部件的占有率看作是 AVF 的上限值，指出 AVF 与占有率的关系为：

$$AVF \propto R \times \lambda \tag{2-8}$$

其中，R 为硬件结构的占有率，λ 表示的是软错误发生的概率。基于占有率的方法在评估过程中不需要在应用程序运行时对指令的停留时间进行统计，也不需要对指令中的ACE位以及Un-ACE位进行分类识别，该方法的缺点是在计算过程中没有对ACE位和Un-ACE位分类，因此得到的结果实际上会偏大，也就使得软错误

对程序输出结果的影响的估计过大，文献中还提出可以通过区分NOP指令的方法来提高精度。

研究人员还提出了一种AVF的计算方法即错误传播模型EPM（Error Propagation Model）方法，该模型是在处理器的各级存储单元以及流水线中，基于错误产生以及传播的规模的基础上建立的，模型建立的方法主要是运用概率论知识，对错误的产生和传播概率进行跟踪，来获取AVF的值。

当前，EPM评估方法已经被研究人员广泛用来评估体系结构级的可靠性[29][36][37]。但是，该方法中存在着一个难以克服的障碍，由于对错误的产生和传播的跟踪不能够十分精确，这与处理器的结构有关，结构越复杂，跟踪就越困难。目前该方法还不能对处理器的存储层面上进行评估，因此对该方法的使用受到了很大的制约。

如上所述，AVF 的度量和计算根据不同的体系结构需要从多个层面来进行，在此，通过选举几个典型的例子，分别从三个方面对这三种模型进行了比较，主要涵盖主要特点以及优缺点，具体如表 2.2 所示。

表 2.2　AVF 计算方法比较

分析方法	故障注入法[19][20][22]	分析模型法[29]	性能模型法[31][34][35]
主要特点	在处理器 RTL 结构模型中实施故障注入，观察分析对程序的执行结果的影响	利用数值分析，对故障的产生和传播进行统计	离线分析存储位的状态是 ACE 位或 Un-ACE 位，单线程
优点	可以对任意状态存储单元进行故障注入，实验精确度高	分析误差与可靠性的联系，得出系统可靠性与不同结构的关系	可以在早期进行分析、计算，具有指导意义
缺点	实验周期长，只有在实验结束后才能对实验结果进行分析	分析过程复杂，需要对所有的结构进行统计	需要大量的计算，复杂度高，且不能够全面分析 Un-ACE 位

以上计算软错误的方法都是从硬件层面来说的，主要是就软错误对系统可靠性影响方面定量分析的，表明了大多数的软错误都会被自动屏蔽，而不会影响系统的结果的正确输出。但是，这些方法都需要结合具体的硬件模拟器来实现，由于没有相应的硬件模拟器，实现起来开销大、复杂度高。而且文献[38][39]中指出这些方法在对可靠性评估方面不够精确，存在着一些不足。最重要的是这些方法

都没有与运行的应用程序相结合。但是在程序运行过程中，大量的计算以及控制流的影响，会产生错误传播，这与程序的运行行为以及内部结构紧密相关，因此软错误的特性使得计算过程中还不能忽略软件方面的影响，并且自身结构与软错误对应用程序的可靠性影响之间的关系，仅从硬件层次上是无法获得的，不能够用来指导软件实现的错误检测和恢复技术的设计。

目前，评估软错误对应用程序的可靠性影响的研究尚未成熟，可以使用的评估方法也不多。文献[40]中描述的 FEA 方法是一种在软错误条件下计算应用程序的可靠性上界的评估方法，但是 FEA 方法只考虑了软错误对程序控制流的影响，却忽略了程序数据受软错误影响的问题。文献[41]以软件模块为单位，针对数据错误在软件中传播情况进行了分析，但是基本模块的错误传播率需要通过故障注入法得到。本文根据现有的方法，结合软错误对应用程序的影响，提出了一种以模块为单位、面向程序层次的可靠性分析，通过在程序层次上对 ACE 位统计，分析 ACE 位在应用程序中占的比例来对微处理器的可靠性进行评估。

2.4 本章小结

本章首先介绍了微处理器的可靠性评估指标，然后分析影响可靠性评估指标的关键因素即脆弱性因子的值，讨论了脆弱性因子的两种常用的分析方法，针对不同的方法进行了分析，指出了对应方法的原理，并结合具体的例子进行了相应的说明，还指出了可靠性分析的关键是 ACE 位的识别，最后对比现有的研究方法，分别指出了现有方法中存在的优缺点。

第三章 微处理器可靠性评估方法的研究

本章首先对程序脆弱性因子 PVF（Program Vulnerability Factor）进行了分析，然后介绍了评估方法的基本思想，并对评估中的关键问题进行了分析。

3.1 PVF计算公式

ACE分析方法对AVF的评估需要模拟硬件处理器，这需要设计者能够了解软错误硬件结构的脆弱性以及考虑设计运行的特定的工作负载。AVF评估指标是对于硬件结构的可靠性而言的，它不能用来表示一个应用程序的脆弱性。Vilas Sridhara等人[42]在对AVF的描述研究的基础上提出了程序脆弱性因子PVF（Program Vulnerability Factor）的概念。本文对DSP的软错误可靠性评估研究是针对不同应用程序分析的，因此需要计算PVF的值来评估可靠性。

AVF 值能够精确地计算到每个硬件结构中的微体系结构位，该结构中的所有位的 AVF 值相加产生该硬件结构的 AVF 值，然后将处理器中所有结构的 AVF 值相加取其平均值，则得到整个处理器的 AVF 值，用这种方式计算时，把处理器看作是所有硬件结构的集合。相似地，可以把一个应用程序认为是所有软件资源的集合。

在对 PVF 的定义时需要对时间量给出一个体系结构级的定义，通常地，AVF 用时钟周期测量时间量，在一个时钟周期内发生的事件一般来说是分不开的，而时钟周期不是一个体系结构级概念，不能用来计算 PVF。PVF 的计算需要一些测量事件间的相对顺序和时间距离的量，在顺序编程模型中，一个体系结构级可见量使得这些测量是指令流。指令按照一定的顺序执行，并且操作之间的距离也可以由中间指令的数目给出。因此，PVF 的定义使用（动态）指令为单位时间段，这种选择将会随体系结构而变化。例如，某些架构如安腾在同一“时间”执行多个指令（指令束）。对于这些体系结构，用于 PVF 计算的时间段应是指令束。此外，一个指令（或束）内的一些操作相对于彼此也往往是有序的，例如，一个指令可以读取和写入相同的寄存器。虽然这些事件发生在相同的“时间”，事件通过该体系结构的排序定义必须被保留。

根据 AVF 定义，一个体系结构位的 PVF 值是该位处于 ACE 位状态的时间（在指令中）分数，整个软件资源的 PVF 可以看作是该资源中 ACE 位占的比重，对于特定的软件资源 R，该软件资源的大小为 B_R，资源中共有 I 条指令，则该软件资源的 PVF 可以归纳为：

$$PVF_R = \frac{\sum_I ACE\ bits\ in\ R}{B_R \times I} \tag{3-1}$$

整个应用程序的 PVF 可以通过每个软件资源的 PVF 值的加权平均来计算。

下面用一个实际的例子来说明运用公式(3-1)计算 PVF 的方法，选取了简单的一段程序，汇编代码如下：

```
1:  00000000  01904264        LDW.D1T1   A1, A2
2:  00000004  02080028        LDW.D1T1   A3, A2
3:  00000008  00004000        LDW.D1T1   A4, A2
4:  0000000c  028d0058        ADD.L1     A3, A3, A4
5:  00000010  01940264        SUB.L1     A1, A1, 1
6:  00000014  00006000  [A1]  B.S1       0x00000c
7:  00000018  008e7f09        STW.D2T2   A2, A3
8:  0000001c  00000000        NOP
```

该汇编代码对 A1、A3、A4 的值进行初始化，程序退出循环的条件为 A1 等于 0，保存 A3 最终的值并返回。在这段代码中，有 8 条静态指令，动态指令的数目由 A1 的初始值来决定，为了计算 A1 的 PVF 值，必须要确定寄存器 A1 发生的相关操作以及发生在这些操作间的动态指令。在代码执行的起始位置，指令行 1 的装入指令对寄存器 A1 有一个写的操作，在循环的每次迭代过程中，指令行 5 的减法操作会对 A1 执行一次读操作和一次写操作，虽然这两个操作发生在同一条动态指令内，由于读操作的结果是 A1 的值，先于写操作的结果，因此该体系结构语义定义读操作发生在写操作之前，指令行 6 的分支操作会导致最后对 A1 的读操作。

（1）如果设寄存器 A1 的初始值为 1，循环只执行一次，在这种情况下，对寄存器 A1 分析，发现在指令行 1 中执行写操作，在指令行 5 中先执行读操作，然后执行写操作，在指令行 6 中最后执行读操作，因此，在指令行 1 和指令行 5 之间以及指令行 5 和指令行 6 之间，寄存器 A1 是脆弱的。由于共有 8 条指令执行，在这部分代码中 A1 的 PVF 值可以被计算为：

$$PVF_{A1} = \frac{(5-1)+(6-5)}{8} = 62.5\% \tag{3-2}$$

（2）如果假设循环执行 100 次，并且寄存器 A1 在指令行 5 不间断地写入和读取，在指令行 6 的分支不间断的读取，分支的每一个实例在减法之后出现 1 条动态指令，随后的减法在分支后出现 2 条动态指令，因此，A1 的 PVF 计算变为：

$$PVF_{A1} = \frac{(5-1)+(6-5)+(99\times1)+(99\times2)}{305} = 99\% \tag{3-3}$$

计算这部分代码的整个软件资源的PVF需要计算每个软件资源的PVF的加权平均。例如，如果该体系结构中包含有 8 个可用的逻辑寄存器，每个寄存器包含32位，那么该寄存器结构的PVF可以表示为：

$$PVF_{ARF}=\frac{\sum_{i=1}^{8}32\times PVF_{Ai}}{8\times 32}=\frac{\sum_{I}ACE\ bits\ in\ ARF}{8\times 32\times I} \tag{3-4}$$

对于处理器的AVF可以认为是所有存储结构的AVF的平均值，同样，可以把应用程序的PVF就可以表示所有的软件资源的加权平均，则可由公式(3-1)计算得到。

3.2 ACE指令/Un-ACE指令

PVF主要表征应用程序受单粒子翻转影响的脆弱性指标，对该指标进行计算的关键问题是确定指令中的存储位的状态，存储位发生翻转是否会导致应用程序的输出结果出错与存储部件的指令类型相关[43]。

根据指令的执行结果是否会对应用程序的最终输出结果产生影响，将指令分为ACE指令和Un-ACE指令，其中把那些执行结果会对应用程序的输出结果产生影响的指令称为ACE指令，比如跳转指令B，而那些不会对应用程序的输出结果产生影响的指令称为Un-ACE指令。另外，考虑到所有的操作码位都为ACE位，那么在分类时要排除掉操作码位。图3.1给出了指令的分类情况。

Un-ACE指令不对输出结果产生影响主要分为两种情况：一是通过分析指令字或者指令自身所具有的功能就可以判断出其不会对输出结果产生影响，此种形式的指令认为是静态Un-ACE指令，这类指令有NOP指令和预取指令[44]等；第二种情况需要分析指令的上下执行顺序，即分析指令流的前后执行关系，进而分析对该指令的影响，此种形式的指令认为是动态死指令。还有一种比较特殊的指令即逻辑屏蔽指令，此类指令中包含有Un-ACE位，这些Un-ACE位主要是那些被屏蔽的存储位，但是此类指令属于ACE指令，由于有些ACE指令中存在着Un-ACE位，因此程序的输出结果不完全受ACE指令的影响。下面将分别具体的分析这几种指令的特点。

图 3.1 指令分类

3.2.1 静态Un-ACE指令

NOP 指令和预取指令本身的功能使其不会对输出结果产生影响，具体来讲，在指令操作过程中，该指令在所作用的区域内出现软错误，则对应用程序最终的执行结果不会产生影响。NOP 指令为空操作指令，NOP 指令当前存在于很多指令集中，像 ARM、DSP 等指令集中就都包含有 NOP 指令，这些指令的存在并不仅仅只是一种原因。例如，为了减少取指令时的内存访问次数，则指令需要按字节对齐，这时可以通过 NOP 指令的填充来完成，NOP 指令占有一个字节。NOP 指令也用于超长指令集中，有时还使用 NOP 指令产生一定的延迟。通过分析指令的二进制格式，可以发现 NOP 指令中只有操作码位决定着指令的功能，而其他位均不起作用。

对于 DSP 指令集中的 NOP 指令，其指令字的二进制格式如图 3.2 所示：

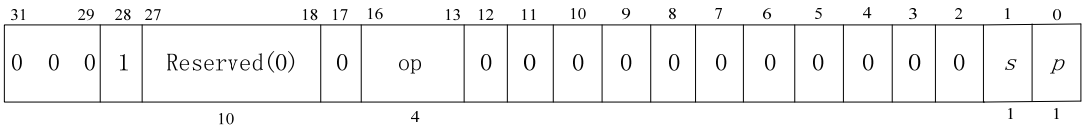


图 3.2 NOP 指令的二进制格式

对于预取指令而言，指令中包含有操作数，这类指令并不像 NOP 指令那样，仅仅用来填充一些字节，设计的原因是为了使处理器的性能效率更高，该指令存在于很多指令集中，比如 ARM、DSP 中都有。预取指令主要用来完成对数据的提取，在指令执行时直接在高速缓存中使用，而不需要先从主存中获取，然后再使用，减少了对主存的访问时间，大大提高了程序运行的效率。对于该类型的指令，如果在高速缓冲中的数据不是指令执行时所需要的数据，那么指令会转到对应的内存中去寻找，这种情况下指令还会继续的执行，而不会发生错误，只是在执行

过程中需要重新进行访问主存的操作。实际上，预取指令没有发挥实际的作用，不会对后面指令的执行造成影响。指令执行时内存地址发生错误可能是源操作数域段的存储位出现了单粒子翻转，也有可能是预取指令没有正确的执行，事实上上述情况都对程序的输出结果不会造成任何的影响。

静态 Un-ACE 指令中还包含有一些特殊的指令，对于这类指令来说，其中的有些存储位没有什么实际意义，而是为了以后指令的扩展等，称这些位是空闲位。空闲位存在的两个原因为：

（1）为了扩展指令集，指令在早期的设计中都比较精简，在随后的设计中会对指令集扩充，引入更多的指令以及增添一些寻址操作，为了不改变早前的设计，这时就可以使用指令中的空闲位来实现；

（2）对于精简指令集的体系结构处理器而言，指令中的指令字长都是一定的，但是由于指令类型不同，执行的操作也不一样，那么指令中所包含的位数不会完全相同，这时就通过指令中的空闲位来填充。

这种类型的静态 Un-ACE 指令虽然从功能上看会对程序的输出结果产生影响，指令中的源操作数在执行过程中不能够发生错误，但是即使指令中存在的空闲位出现了翻转，指令也会正常的执行而不受影响。

3.2.2 动态死指令

对于动态死指令，其执行与前后的指令有关，从流水线上进行判断，这类指令的执行不影响最后程序的输出结果，其执行结果在执行时没有起作用，这类指令不同于静态指令，主要是从指令本身无法判断其是否影响程序的执行，而必须结合前后的程序来进行分析。

动态死指令进一步划分为两类[45]：第一级动态死指令FDD（First-level dynamically dead instructions）和传递动态死指令TDD（Transitively dynamically dead instructions）。对于第一级动态死指令FDD，其指令的执行结果，在应用程序执行过程中，直接被后续指令所覆盖了，或者是程序运行结束了，也没有对指令的执行结果使用，具体表现为：

（1）指令向存储单元中保存数据，但是操作的值在被使用之前，会有新的数据向该存储单元中写入，则指令在这段时间内没有起作用，那么即使存储位出现翻转现象，也对结果没有影响；

（2）指令向存储单元中写入的数据在程序执行完也没有起作用，很明显，这类指令的存储位即使出现了翻转，那么由于在整个程序执行时也都不会影响到其他指令的操作。

传递动态死指令 TDD 是指指令的运行结果只会由其他的动态死指令所引用。该类型指令被认为是动态死指令的原因可以分析为，在指令运行时，如果有些处

理器中存在着分支预测功能，那么在分支过程中，就有可能导致预测结果出现错误，而对后续流水线上指令的执行造成影响，从而影响对应用程序可靠性的评估，而实际上在处理器检测到错误之时，处理器就会采取一定的措施对这些错误指令产生的影响进行消除，由此以来，这些错误就不会影响程序的正常运行，因此即使该指令中的存储位出现了翻转，那么应用程序的执行也不会受到影响。比较具有代表性的就是在跳转指令后边的一些指令，假设指令按顺序依次执行到跳转指令，而实际上流水线上已经有跳转指令后的 m 条指令在执行，如果跳转指令能够成功执行，则判定其后的 m 条指令均为动态死指令。因为在跳转指令执行之前，这 m 条指令的操作信息都要被清除掉，因此，认为在这个阶段即使一些相关的存储位出现了单粒子翻转的现象，那么也不会影响程序的正确执行。

由上可知，传递动态死指令不会对结果造成影响表现在指令没有正常的按照流水线的执行结束，而是在流水线操作尚未结束之前就从流水线上消失了。

3.2.3 逻辑屏蔽指令

逻辑屏蔽指令是指那些没有执行计算而其输出结果能够被决定的指令，在对逻辑屏蔽指令识别时通常扩展到对那些没有执行计算而其输出结果能够被决定的指令的识别[46]，下面对这类指令中的三种情况来分析：

（1）指令包含两个源寄存器 A 和 B，对于乘法指令以及 AND 指令，如果任意一个源寄存器的值为 0，则另一个寄存器中的值的大小对指令的执行结果不会产生影响，对于 OR 指令而言，如果其中一个寄存器中的值为 1，则另一个寄存器的取值也不会影响指令的执行结果。

（2）指令中包含 1 个源寄存器和一个立即数 IMM，如果源寄存器中值为 0，则立即数将会屏蔽指令对计算结果的影响，如果立即数 IMM 为 0，则源寄存器不影响计算结果，对于 OR 指令，如果源寄存器值为 0 或立即数 IMM 为 1，则相对应的立即数或者源寄存器的值不影响计算结果。

（3）指令为特定的 XOR 操作，当 A=B 时，指令的结果与 A、B 的值大小无关。

根据以上对逻辑屏蔽的扩展分析把逻辑屏蔽指令的情况总结如表 3.1 所示：

表 3.1 逻辑屏蔽指令

类型	操作	条件
I	MPY/MPYH/.../:A*B	A=0 或 B=0
	AND:A&B	A=0 或 B=0
	OR:A|B	A=1 或 B=1
II	MPY/MPYH/.../:A* IMM	A=0 or IMM=0
	AND:A & IMM	A=0 or IMM=0
	OR:A | IMM	A=0 or IMM=1
III	XOR:A^B	A=B

3.3 ACE位/Un-ACE位位数

指令在进行读、写等操作过程中，会对存储单元传输数据，在空间环境下，这些数据的存储位有可能会发生单粒子翻转，但是发生的翻转存储位却不影响程序的运行，称这些位为 Un-ACE 位。对于 Un-ACE 指令来讲，其操作数和操作结果域段的存储位并不一定都是 Un-ACE 位。比如，对于 FDD 指令，如果目的操作数域段的存储位出现了位翻转，数据将不被写入正确的目的地址，这种情况对后续指令执行的结果却可能会造成影响，在程序中这种情况却是不应该存在的。因此，指令中的 ACE 位和 Un-ACE 位所占的位数的情况取决于指令属于哪一种类型和指令的操作类型，不同类型指令所对应的 Un-ACE 位位数还依赖于指令的格式和寻址方式。

对于静态指令中的 NOP 指令存在的原因，是为了满足体系结构的要求，预取指令设计的目的在于提高整个系统的性能，这两种指令在执行过程中任何一个操作数存储位发生单粒子翻转，对程序的最终结果都不会产生影响。对于静态指令中的另外一种特殊指令而言，在其指令字中有些域段的存储位没有什么实际的意义，则这些存储位发生翻转不会对程序的执行结果造成影响。

由于第一级动态死指令 FDD 中的执行结果,在后续程序执行中直接就覆盖了，或者到程序结束都没有被使用，那么即使发生单粒子翻转也不会对最终结果造成影响，分析这类指令的关键在于分析操作结果写入的存储部件以及写入的位置，这与指令的操作类型有关系。

3.3.1 指令类型

大多数指令集完成的主要操作类型包括数据处理、数据传输以及分支跳转操

作等，对于这三种类型的操作不同指令集中会有相应的指令。对第一级动态死指令分析 Un-ACE 位时重点分析指令的写入地址，因为该地址中的值没有被引用过，在指令执行过程中，只要保证写入的地址正确，则程序执行的最终结果都不会受影响，而与指令写入该地址的内容无关。本文以 DSP 指令集为研究对象，下面对三种操作类型指令进行分析。

（1）数据处理类型指令包括：

1）算术运算指令，这类指令不仅包含加法、减法、乘法、比较等基本指令以外，还包含对这些指令扩展的情况，比如对于 DSP 指令集而言，加法指令 ADD，以及对其进行扩展后的 ADDK、ADD2 等指令都属于算术运算指令；

2）逻辑运算指令，主要是那些与逻辑操作相关的指令，如与指令，DSP 指令集中还存在一些与位操作相关指令也是逻辑运算指令；

3）移位操作指令，就是对存储位的左移或右移相关的指令。

对于数据处理类型指令，其指令执行的最终的目的都是对寄存器进行执行结果的写入，对于该类型指令，源操作数的一个或多个存储位发生单粒子翻转，而目的寄存器的标号只要正确，那么这些出错的存储位对指令的输出结果不产生影响。

（2）数据传输类型指令包括：

1）数据在两个寄存器间的传输；

2）数据在存储器与寄存器间的传输；

3）数据在两个存储器间的传输。

其中，对于将常数写入到寄存器这种情况属于第一种数据传输类型，同样将常数写入到存储器这种情况属于第二种数据传输类型，这两种数据传输类型比较常见，大部分的指令集中都有其代表指令，如 DSP 指令系统中的 MV 指令，它用于完成两个寄存器之间的数据传输，对于 STB 指令则是存储器或寄存器向存储器的数据传输，由此类指令进行扩展的指令也用来实现数据之间的传输，比如 MVK、MVC、STW 等等。对于目标地址为寄存器的情况，可以依据数据处理类型的所述方法进行分析。对于数据在两个寄存器之间传输的情况，源寄存器的值通常来说是不起作用的，同样，对于存储器向寄存器这种传输的情况，即使存储器的值发生了错误，程序也会正常运行而不会出现错误。

对于数据传输类型指令，如果目标地址是内存地址，那么只要保证写入的地址是正确的即可，而具体写入的数值是什么，不会影响指令的操作，因此认为该类型指令为 Un-ACE 指令，这个地址来源于寄存器中的数值，也可以是寄存器中的数值与立即数的计算结果。

（3）分支跳转类型指令：对于程序的执行，一般而言都是有序地进行的，实

际上执行过程中还会出现分支的情况。这种情况下，指令就会转到分支的目标地址执行，根据其执行的状况，将分支跳转指令分为两种：

1）无条件转移指令：到遇到无条件转移指令时，指令就会转向指定的目标地址，无条件的执行，不需要判断是否满足状态寄存器中的条件；

2）有条件转移指令：条件转移必须受到条件的约束，通过判断是否满足状态寄存器中的条件来确定是否发生转移，如果满足条件，指令会寻找到相应的目标地址而执行，如果不能够满足所要求的条件，则指令将按照指令原先的执行规则依次执行。程序中使用条件转移指令的目的是因为在执行过程中，执行的顺序并不是按照预先设定好的顺序一条一条的执行的，有可能会在执行时跳到其他的地方，这种情况下就要满足相应的条件。

3.3.2 指令格式

DSP 的 C6000 指令集中，指令都包含着 32 个存储位，不同指令其指令格式不一样，具体的跟其二进制代码相关。图 3.3 是功能单元的指令格式。

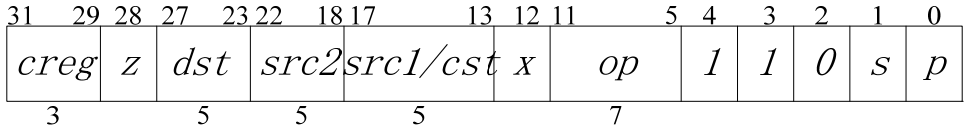


图 3.3 功能单元的指令格式

op 表示指令操作码，指明指令执行什么样的操作，指令执行的前提是需要对指令完成译码，也就是说需要先分析操作码，然后确定指令的执行行为，操作码不相同，则说明指令不是同一类型，可以通过观察其编码方式来判断指令的类型，它们之间是一一对应的关系。

creg 指定条件寄存器代码，根据条件寄存器中的特定位确定指令是否执行，DSP 指令集中，31 至 29 位之间的四位组成条件寄存器代码；z 指定指令执行的条件，如果在指令中 creg 为 0 的同时，z 也为 0，那么该指令为无条件执行的指令。

x 判断操作数 2 是否使用交叉通道。

src 表示源操作数域段，dst 表示目的操作数域段。在指令系统中，存储器中包含着很多存储单元，用来存放指令或数据，这些存储单元每个都有一个对应的地址编号，就是所说的地址码，可以通过地址寻找对应的操作数，地址码通常指定参与操作的操作数的地址，在指令中按照操作数的地址个数，可以对指令分类，比如，有一个操作数，则认为是一地址指令，主要使用的就是一地址指令以及二地址指令。

零地址指令就是说指令中不存在操作数，例如，NOP 指令和空闲指令，指令字中只有操作码。

一地址指令很明显只有一个显地址，指令中隐含着一个操作数，这个隐含的操作数通常放在一个特定的寄存器中，这个就寄存器在进行连续运算时，保存着多条指令的运算结果，称为累加寄存器。该类指令需要对主存访问两次，先取指令本身，然后取操作数。

二地址指令是指令中只有两个操作数，这两个操作数中有一个源操作数，一个目的操作数，操作数可以是内存地址，也可以是寄存器地址。

三地址指令也就是说指令中包含有三个操作数地址，其中两个用来指明操作数的来源地址即源操作数，第三个用来存放结果的单元地址，即目的操作数。

s 用来选择目的操作数是寄存器组 A 还是寄存器 B。

p 指定指令是否并行执行。

3.3.3 寻址方式

所谓寻址方式通常是指某一个CPU指令系统中，规定的寻找操作数所在地址的方式，或是通过什么方法找到操作数，寻址方式给出了CPU访问数据空间的方式，它与微处理器的硬件结构密切相关，TI C6000 系列DSP芯片全部采用间接寻址，在线性寻址时都使用寄存器对作为地址增加或减小的指针量，这些寄存器为A4-A7、B4-B7，由寻址模式寄存器AMR控制地址修改方式，在C6000 系列DSP中存在两种寻址模式分别为线性（默认）方式和循环方式[47]。图 3.4 给出了寻址模式寄存器AMR各个位域的定义，表 3.2 给出了由模式（mode）的 2 位数值所确定的选择。

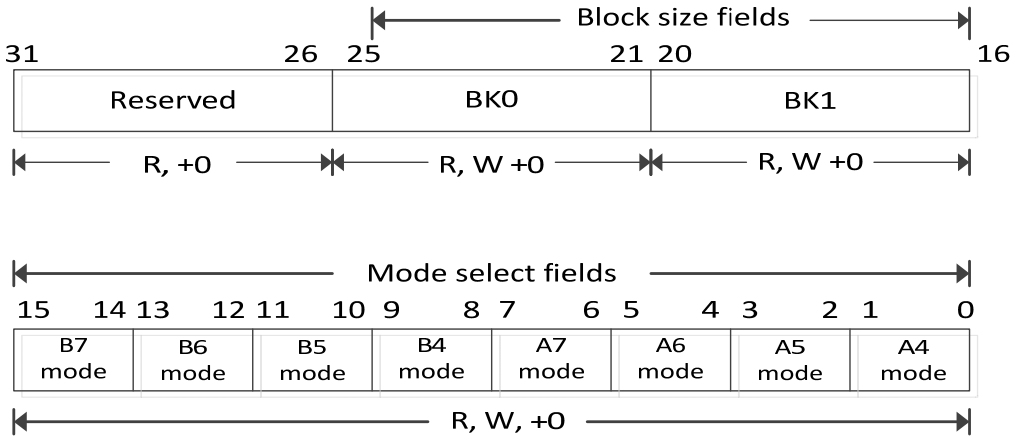


图 3.4 寻址模式寄存器 AMR 各个位域的定义

表 3.2 寻址模式寄存器模式选择字段编码

模式（Mode）	描述
00	线性寻址（复位后默认值）
01	循环寻址，使用 BK0 字段
10	循环寻址，使用 BK1 字段
11	保留

对于线性寻址而言，通过线性的增加或减少一定单位量来完成对基地址的改变。对于循环寻址，指令中的第 N 位是循环的块尺寸区间的结束位置，则不能够对第 N+1 的存储位进行借位或者进位，也就是说，这些位只能在 0~N 之间进行循环改变，而在 N 位以后的地址则不会改变，图 3.4 中可以看出，BK0 和 BK1 都占有 5 个字节，这些字节组成的数值用来表示块尺寸区间的大小，可以用公式：块尺寸=$2^{(N+1)}$来计算循环寻址的区间，单位是字节（Byte）。举例来说，如果 N 为 10000，用二进制表示，转换为十进制，则值为 16，那么用公式计算得到块尺寸大小为$2^{(16+1)}$=131072 字节。

Load/Store 类指令主要采用间接寻址的方式寻址，其间接地址的产生情况如表 3.3 所示。其中，ucst5 表示无符号的二进制数据偏移量为 5 位，offset 代表偏移地址，对于寄存器 B14 和 B15，则用 ucst15 来表示间接地址。

表 3.3 Load/Store 类间接地址的产生

寻址类型	不修改地址寄存器	先修改地址寄存器	后修改地址寄存器
寄存器间接寻址	*R	*++R *--R	*R++ *R--
寄存器相对寻址	*+R[ucst5] *-R[ucst5]	*++R[ucst5] *--R[ucst5]	*R++[ucst5] *R--[ucst5]
带 15 位常数偏移量的寄存器相对寻址	*+B14/B15[ucst15]		
基址+变址	*+R[offsetR] *-R[offsetR]	*++R[offsetR] *--R[offsetR]	*R++[offsetR] *R--[offsetR]

3.3.4 ACE位/Un-ACE位位数分析

通常指令执行什么样的操作由指令的操作码来决定，而有些指令系统中，不但有操作码，还存在着操作码附加段，那么指令执行的操作方式则不仅仅只受操

作码的影响，还与指令类型标识域段以及附加域段密切相关。而指令的寻址方式只能够给出获取地址的方法，通过该方法从而获得操作数，指令的寻址方式与指令的功能是没有联系的，即使指令的寻址方式不相同，但是指令却可以执行相同的操作。因此，指令字中的操作码域段、指令类型标识域段以及操作码的附加域段，这三者共同标识了指令执行的操作。如果指令执行过程中，执行的操作是与存储器相关的存储方式，并且操作码域段的位出现了错误，那么后面的指令域段就不能准确判断出指令执行的操作类型，从而无法准确判断出指令是 ACE 指令还是 Un-ACE 指令。因此对于静态指令以及动态死指令而言，指令中的 ACE 状态位，不仅包含操作码域段的存储位，还包含指令指令类型标识域段以及操作码附加域段的存储位。

对于 Un-ACE 指令而言，指令中哪些域段的存储位是 Un-ACE 位与指令的类型密切相关，不同类型的指令，其 Un-ACE 位位数不同。通过分析 NOP 指令的指令字发现，NOP 指令中只有操作码域段，而不存在源操作数以及目的操作数等内容，只有操作码域段决定了指令执行的操作类型，也就是说只有操作码域段的存储位发生单粒子翻转时，才会导致程序的执行结果出错，因此，认为 NOP 指令的操作码存储位为 ACE 位，而其他存储位都认为是 Un-ACE。

对于预取指令，由于其本身的功能使得该指令不会对程序的最终结果产生影响，而与执行过程中指令的操作数的正确与否没有关系，因此，操作数域段所占的存储位发生单粒子翻转，不会影响程序运行的正确性，所以预取指令中除了操作码为 ACE 位外，其他存储位都认为是 Un-ACE 位。

对于第一级动态死指令而言，其 Un-ACE 位的识别并不像 NOP 指令和预取指令那么简单。根据对第一级动态死指令的定义可以知道，该指令中的目的操作数中的数值，在整个应用程序执行过程中都没有被引用，或者在被引用之时就被后续执行的指令所产生的值覆盖了，对于该类 Un-ACE 指令来说，在操作过程中，如果数值被写入了正确的目的寄存器中，而数值的正确与否毫无关系，即便这个数值是不正确的，那么还没被引用就已经被覆盖了，因此在程序运行时不会起作用，因此，其 Un-ACE 位除了包含操作码域段的存储位外，还有可能包含着其他的存储位，比如那些表示操作结果域段的存储位，这和指令的类型相关。

对于数据处理类型指令，其指令执行的最终的目的都是对寄存器进行执行结果的写入，那么只要求写入的目的寄存器不能够出现错误，至于写入的值正确与否没有关系，因此产生写入数值的源操作数的存储位即使出现翻转，对程序的执行结果也不会造成影响。而源操作数的获得取决于指令的寻址方式，对于寄存器间接寻址来说，源操作数的地址来自于寄存器中的值，所以认为该类指令的 Un-ACE 位为源寄存器号域段中的存储位，对于基址和变址的寻址方式而言，由于

操作数所在的地址是两个源寄存器的值之和，因此，表示这两个源寄存器号域段中的存储位为 Un-ACE 位。

对于数据传输类型指令，根据数据写入的位置是寄存器或是内存，把指令分两种情况进行讨论：一是将数据写入到寄存器中，可以是寄存器之间的传输，也可以是内存向寄存器的写入；二是将数据写入到内存中，这种情况可以是内存到内存的写入，也可以是寄存器向内存的传输。

对于目的地址为寄存器的指令，目的寄存器标号域段必须正确，程序才能正确执行，如果指令中数据是寄存器之间的传输，那么表示立即数或源寄存器域段的存储位为 Un-ACE 位。如果数据是内存到寄存器传输的指令，即 Load 指令，对于 Load 指令寄存器中的值可以是源操作数的地址，立即数与寄存器中的值相加结果，以及寄存器中的值通过计算得到的值也可以作为源操作数的地址，无论该地址是如何得到的，表示源寄存器标号以及立即数的指令域段的存储位发生翻转，都不会影响程序的最终输出结果，那么这些存储位为 Un-ACE 位。

对于目的地址是内存地址的指令，如果该指令的运行结果在被访问之前就被覆盖了，只要保证写入的地址是正确的，那么不论写入的值正确与否都不会影响到后续指令的执行，从而不会对应用程序的结果产生影响，因此表示写入值指令域段的存储位认为是 Un-ACE 位，指令的寻址方式一般为寄存器间接寻址、基址和变址寻址以及寄存器相对寻址，因此认为此类指令中表示立即数域段和源寄存器标号域段的存储位为 Un-ACE 位。

对于那些带返回的分支跳转指令，指令操作的结果是将数据写入寄存器中，这类指令看起来和数据处理类型指令相似，但是该类指令被写入的数据是下一条指令的地址。如果写入的数据域段的存储位出现了翻转，则指令会转向错误的地址执行，也就改变了指令的执行方向，因此认为该数据域段的位是 ACE 位，并且指令的目标地址是通过源操作数得到，因此也认为源操作数域段的位是 ACE 位。

对于传递动态死指令，该指令是在第一类动态死指令基础上分析的，在执行过程中只有最后的结果，把中间过程得到的值都覆盖了，相应的存储单元会被清空，因此与该类指令相关的操作域段的存储位均认为是 Un-ACE 位。

对于逻辑屏蔽指令，由于指令中的那些被逻辑屏蔽的数据不论怎么改变，而执行的结果都已经被确定，因此，这些逻辑屏蔽位对结果不会造成影响，即认为是这些存储位是 Un-ACE 位。

3.4 本章小结

本章主要分析了本文所提出的评估应用程序的可靠性方法，并对评估过程中需要解决的关键问题进行了概括总结，包括对评估公式的介绍、应用程序中 ACE 指令和 Un-ACE 指令的确定以及 Un-ACE 位位数的判断等。其中 Un-ACE 指令的

分析是评估的重点，根据 Un-ACE 指令对输出结果不会产生影响的不同表现形式，将指令进行分类，然后进一步研究指令的类型和寻址方式进而完成对 Un-ACE 位位数的判断。

第四章 基于C6000 指令集程序的可靠性评估

本文提出的评估方法中，首先需要对指令进行遍历，依据指令的存储类型、指令之间的相邻操作关系，识别出所包含的 ACE 位，完成对 ACE 位的采集，最后根据评估模型对应用程序进行可靠性计算。评估框架如图 4.1 所示，其中指令的分类以及对 Un-ACE 位的识别是评估的关键问题。

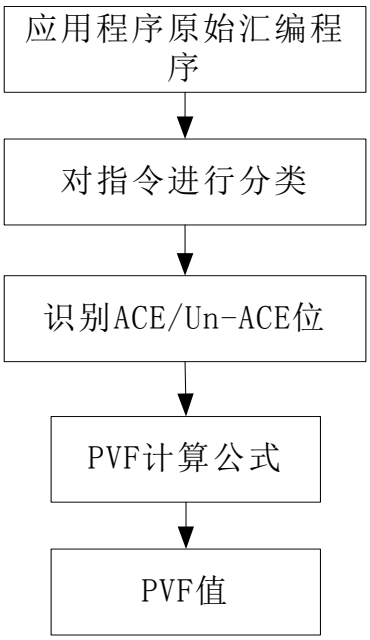


图 4.1 评估框架

本文的可靠性评估是针对 DSP 应用程序进行的，根据对前面的分析，评估的过程主要包括以下环节：

（1）对应用程序中指令进行分类，识别出每条指令的类型；

（2）根据指令的类型以及存储性质，判定指令中源操作数、操作码或目的操作数等域段的 ACE 位位数；

（3）根据指令的相邻操作，分析应用程序执行过程中的时间间隔，相当于指令的数目。

4.1 指令分类识别

根据第三章讨论可知，针对不同类型指令需要对汇编程序代码进行分类识别，对于 TMS320C6000 系列处理器，汇编程序代码格式如图 4.2 所示。其中，第一列表示指令的地址，第二列是机器码，第三列是操作码以及操作单元，第四列是操作数，其中，目的操作数在前，源操作数在后，中间用逗号隔开。

```
00000050    01904264          LDW.D1T1    *+A4[2],A3
00000054    02080028          MVK.S1      0x1000,A4
00000058    00004000          NOP         3
0000005c    028d0058          ADD.L1      8,A3,A5
00000060    01940264          LDW.D1T1    *+A5[0],A3
00000064    00006000          NOP         4
00000068    008e7f08          EXTU.S1     A3,19,31,A1
0000006c    80000691    [ A1] B.S1        0x000094
00000070    91940264 || [!A1] LDW.D1T1    *+A5[0],A3
00000074    900000aa    [!A1] MVK.S2      0x0001,B0
00000078    00004000          NOP         3
0000007c    008c8f78          AND.L1      A4,A3,A1
00000080    800000fa    [ A1] ZERO.L2     B0
```

图 4.2 汇编程序格式

在进行软错误评估时，根据需要对 DSP 工程目标文件（.out 文件）的反汇编文件进行分析，从汇编指令角度出发，根据一定的规则将汇编程序划分为若干个模块，各个模块之间的关系用拓扑图表示，以模块为单位，计算出每个模块的脆弱性，然后结合可靠性评估指标 FIT，对易出错的模块采取相应的防护措施。

针对 TI C6000 DSP 程序汇编代码，大致划分为四个层次：第一层为系统层；第二层为子系统层；第三层为函数层；第四层为模块层。图 4.3 所示为划分层次示意图。其中，系统层由多个子系统构成，子系统层由多个函数构成，函数层由多个模块构成。模块与模块之间的，函数与函数之间存在着一定的关系。

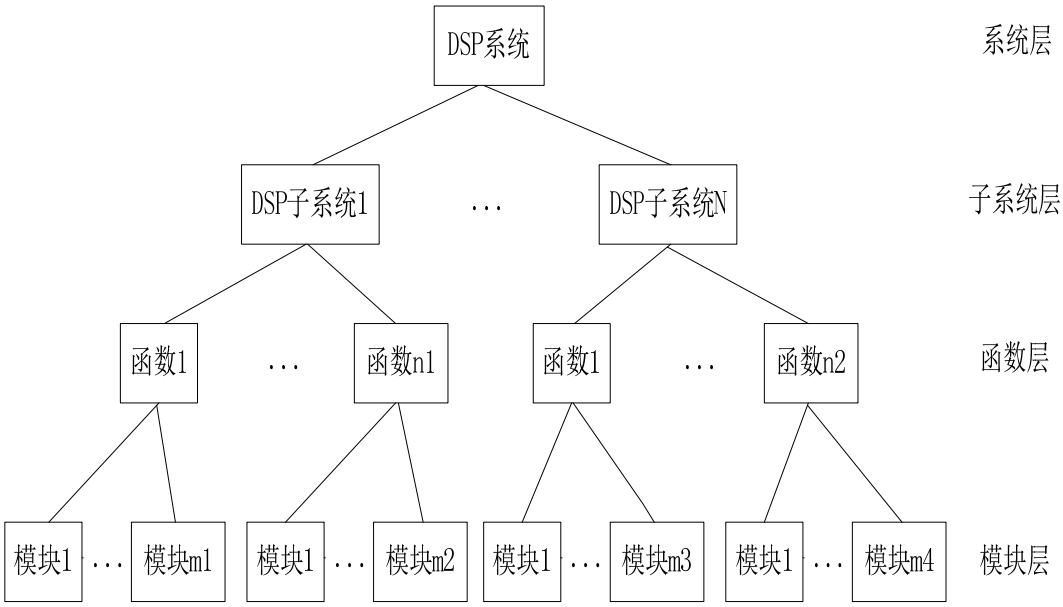


图 4.3 划分层次示意图

在分析每个模块的脆弱性时，需要对指令类型进行识别，其主要过程是：首先，遍历目标体系结构的汇编文件，通过对指令汇编代码中操作码的识别可以把 NOP 指令、预取指令以及跳转指令很容易识别出来，如果该指令是 NOP 指令或预取指令，则指令是 Un-ACE 指令，如果是跳转指令，则指令为 ACE 指令，对于逻辑屏蔽指令，主要是根据操作数的逻辑操作关系，来完成对该类指令识别的，逻辑屏蔽指令属于 ACE 指令，但是指令中包含着 Un-ACE 位。最后，在对动态死指令的识别时，需要对指令结果的后续使用情况进行准确的跟踪，这是指令识别过程中的一个难点。图 4.4 给出了一个模块的指令识别过程，指出了不同类型指令的识别顺序，在对指令进行识别过程中，需要统计出模块中每类指令的数目以及指令的总数目。

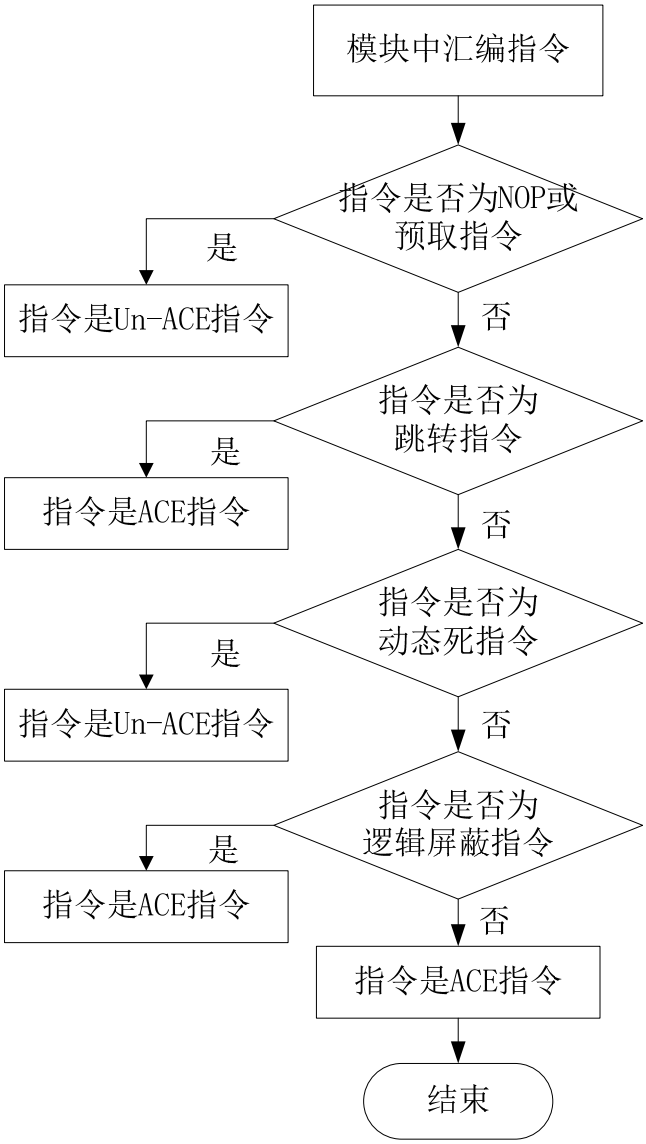


图 4.4 模块的指令识别过程

由于动态死指令中包含的两种类型指令的特点不同，因此在识别过程中需要分类进行。对于第一级动态死指令 FDD 的识别可通过识别属于第一级动态死指令两种情况，对于传递动态死指令需借助第一级动态死指令来识别。

第一级动态死指令的识别方法如下：

（1）当目的操作数是寄存器或者内存时，先对这些存储单元中的数值进行判断，检查这些数值被后续指令的引用情况，如果在后续执行过程中从未使用过这些值，则认为该指令为 FDD 指令，如果某条指令执行过程中只有这一个数值输出，则认为该指令也为 FDD 指令，另外，如果其他数值，也由产生这些存储单元中数值的指令产生，并且这些数值在后续执行过程中，也没有被使用过，则认为该产生指令为 FDD 指令；

（2）判断应用程序执行完毕时存储器中的数值，如果在后续执行过程中从未使用过这些值，则认为该指令为 FDD 指令，如果某条指令执行过程中只有这一个数值输出，则认为该指令也为 FDD 指令，另外，如果其他数值，也由产生这些存储器中数值的指令产生，并且这些数值在后续执行过程中，也没有被使用过，则认为该产生指令为 FDD 指令；

在对第一级动态死指令的识别过程中，不但需要利用指令链表，而且需要建立操作数的消费者-产生者链[48]来完成。所谓指令链表是指根据指令执行的先后顺序，建立的关于指令操作的链，在链表中保存着指令的操作码、目的地址、操作数等所有的相关信息，而操作数的消费者-产生者链是指对所有的存储器和寄存器单元，建立的关于目的操作数以及源操作数之间的关系的数据流图，指令链表与消费者-产生者链相对应，指令链中的源操作数被认为是消费者，而目的操作数则认为是产生者，为了清晰地说明操作数的利用情况，如图 4.5 所示是所建立的关于操作数的指令链表和消费者-产生者链。

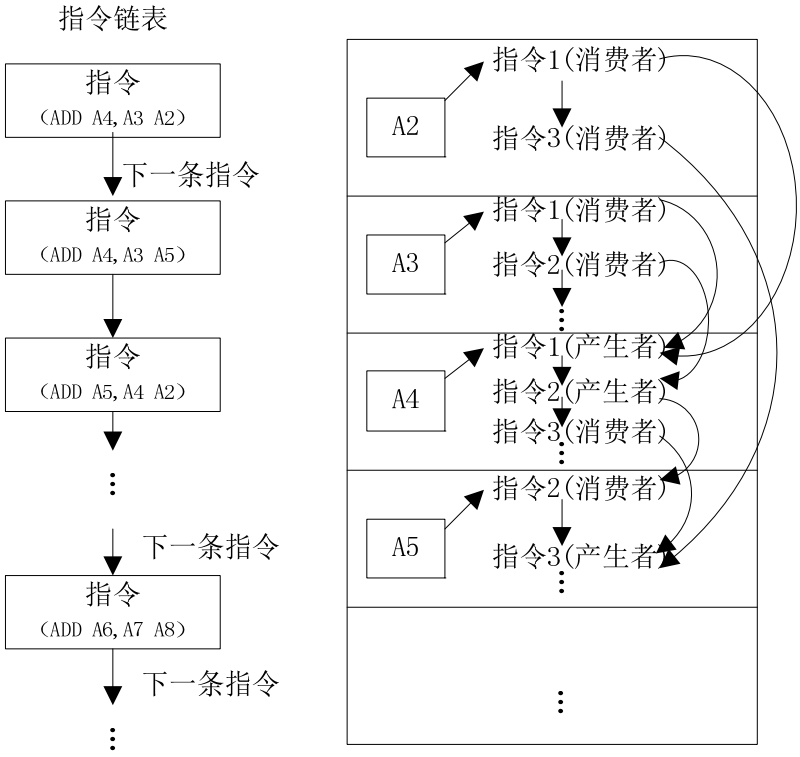


图 4.5 指令链表和操作数的消费者-产生者链

对一段应用程序的执行过程进行分析，从指令链表中可以把指令中的源操作数以及目的操作数标记出来，操作数的消费者-产生者链则可以给出该指令的各个操作数来源于那条指令的结果，一段程序结束后，需要更新指令链表以及消费者-产生者链。指令链表中的指令结点包含着目的地址、指令的操作码、指令执行的行号以及指令的执行结果等信息。

在指令链表中，相邻的结点存在着依赖关系，后继结点的源操作数如果是前驱结点的执行结果，则说明后继结点使用了前驱结点的执行结果。如果当一条指令链表执行完毕，然而某个结点的值却跟后续的指令没有关系，那么该结点将不起作用，认为该结点表示的指令为动态死指令，向前遍历得到该结点的前驱结点，如果前驱结点没有其他的后继结点，则认为前驱结点表示的指令也是动态死指令，同样如果该前驱结点的所有后继结点都没有在后续中起作用，则该前驱结点表示的指令为动态死指令。

从图 4.5 中可以看出，对于指令 1 和指令 2，其目的操作数都是寄存器 A4，也就是说都是 A4 的产生者，指令 1 执行完，指令 2 接着执行，指令 2 的结果直接覆盖了指令 1 的执行结果，因此说指令 1 为 FDD 指令。对 TDD 指令的识别过程要比 FDD 的难度大，通过对模块内的指令进行更深入的研究才能实现。再继续寻找寄存器 A4，可以发现，它是指令 3 中源操作数，也就是说指令 3 是寄存器 A4 的消费者，对分析与寄存器 A4 有关的所有指令，如果发现在指令 2 之后中出现了

ACE 指令，则认为指令 2 是 ACE 指令。如果指令 2 是 TDD 指令，则必须保证在指令 2 之后，所有源操作数是寄存器 A4 的指令是 FDD 或者 TDD 指令。假设模块的窗口是一定的，那么在指令 2 后有的指令包含源操作数寄存器 A4，但是却不能够确定那些指令的类别，则也认为指令 2 是 TDD 指令，这种情况下，与实际情况可能存在一定的差距。因此，模块大小的选择会对指令的分类造成一定的影响，模块选择的大时，对指令类别的分析会比较准确，但是反过来却带来了额外的时间开销以及分析的复杂度。

4.2 Un-ACE位位数的确定

在对指令分类检测完成之后，对应用程序进行脆弱性的评估，以 DSP C67XX 的体系结构为例，每条指令中包含 32 位，对不同类型指令中 Un-ACE 位位数的具体情况讨论如下。

（1）如果该指令是 NOP 指令，由于指令字中只有表示操作码的域段是 ACE 位，那么其 Un-ACE 位位数就是除去操作码剩下的 29 位；

（2）如果该指令是预取指令，同 NOP 指令相似，指令字中表示 ACE 位也只有操作码域段，那么其 Un-ACE 位位数就是除去操作码剩下的 29 位；

（3）如果该 Un-ACE 指令是动态死指令，通过对指令的操作类型和寻址方式进行分析。

对于数据处理类型指令：

1）如果是寄存器寻址方式，则表示源操作数的两个域段为 Un-ACE 位，因此这类指令的 Un-ACE 位数是 10 位；

2）对于立即数寻址来说，则表示立即数和源操作数的域段为 Un-ACE 位，这类指令的 Un-ACE 位位数是 19 位，对于移位指令，则是移位数和源操作数的两个域段为 Un-ACE 位，这类指令的 Un-ACE 位位数是 8 位；

对于数据传输类型的指令，通常分为两种情况：装载（Load）指令以及存储（Store）指令，两者表示的都是内存地址和寄存器之间的数据传递关系，装载指令是内存到目的寄存器的数据传输，存储则是源寄存器到内存的数据传输。

1）Load 指令中，内存地址通过两种方式得到，分别是寄存器中数值和立即数的和，或者和寄存器中数值的和，对于与立即数的和的情况，源寄存器和 offset 两个域段为 Un-ACE 位，这类指令的 Un-ACE 位位数是 9 位，对于两个寄存器值的和，则两个源寄存器域段为 Un-ACE 位，其 Un-ACE 位位数是 12 位；

2）Store 指令中，需要写入的内存地址的获取方式也包含两种，但其被写入的数据只来源于源寄存器，所以表示源寄存器域段的位为 Un-ACE 位，指令的 Un-ACE 位位数是 4 位。

（4）对于跳转类型指令，认为指令中的所有位都为 ACE 位，即 Un-ACE 位位数是 0 位。

（5）对于逻辑屏蔽指令，指令中被逻辑屏蔽的存储位认为是 Un-ACE 位，根据第三章对逻辑屏蔽指令情况的分析，可知指令中 Un-ACE 位主要来源于那些表示源寄存器域段和寄存器标号域段的存储位，或立即数域段的存储位，即指令的 Un-ACE 位位数是 6 位。

4.3 实验结果与分析

本节利用本文提出的评估框架进行应用程序的脆弱性评估，主要包括实验环境、测试结果与分析等。

4.3.1 实验环境

实验过程中，主机采用的是 Linux 操作系统，表 4.1 对主机的规格配置进行了说明，编译器的版本是实验的关键。

表 4.1 实验主机环境

项目	配置
CPU	Pentium4 3.0GHz
内存	2.00GB
硬盘	320GB，7200RPM，16MB Cache
操作系统	Ubuntu 12.04
编译器	gcc4.6.3

4.3.2 测试结果

本文选取了几个 DSP 工程目标文件（out 文件）作为测试程序，包含冒泡排序、傅里叶变换等，对这些测试程序进行反汇编，从汇编指令角度出发，根据一定的规则进行模块划分，然后计算各个模块的 PVF 值。其中一个模块从起始地址开始到终止地址结束，下面对应用程序中只有主函数 main 的测试程序 test_r.txt 进行模块划分，划分后 main 函数分为 8 个模块，表 4.2 列出了每个模块的起始地址、终止地址以及 PVF 评估结果，为了比较直观的分析，将每个模块的 PVF 值以图的形式表现出来，如图 4.6 所示。

表 4.2 test_r 中每个模块的评估数据

	测试程序	起始地址	终止地址	模块的 PVF 值
1	test_r			
2	main 函数	0x000004e0	0x000005fc	
3	模块 1	0x000004e0	0x0000055c	0.586397
4	模块 2	0x000005d8	0x000005e4	0.539062
5	模块 3	0x0000056c	0x000005ac	0.43125
6	模块 4	0x000005b0	0x000005c0	0.575
7	模块 5	0x000005c4	0x000005d4	0.332031
8	模块 6	0x00000560	0x00000568	0.6875
9	模块 7	0x000005e8	0x000005f0	0.489583
10	模块 8	0x000005f4	0x000005fc	0.125

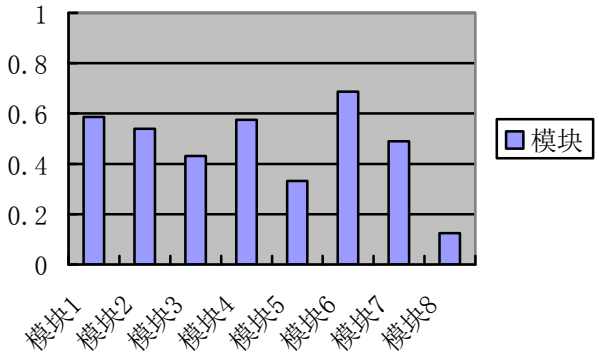
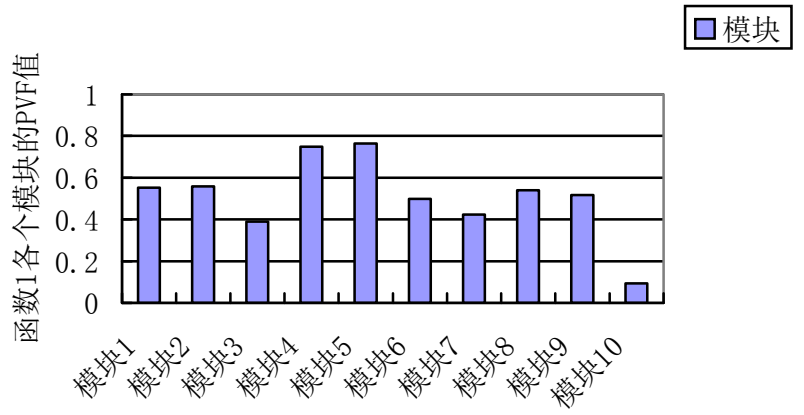
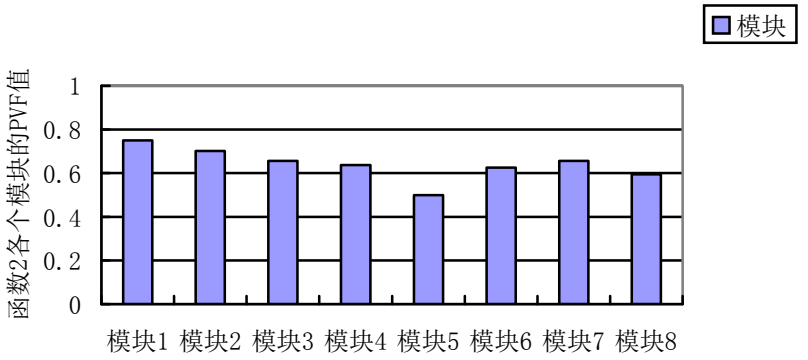


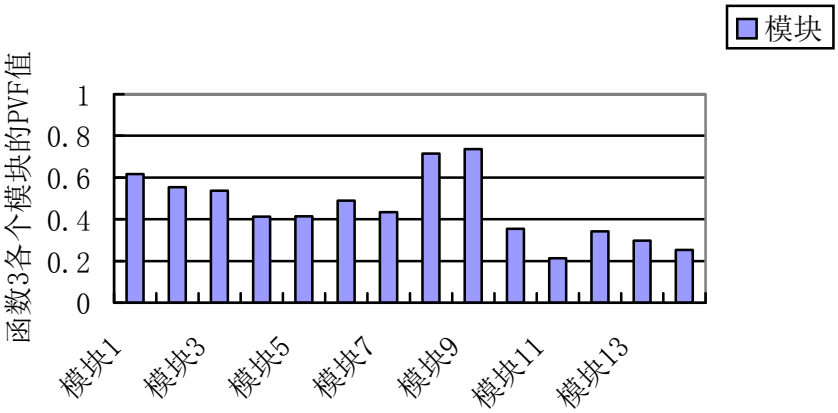
图 4.6 test_r 中每个模块的评估数据

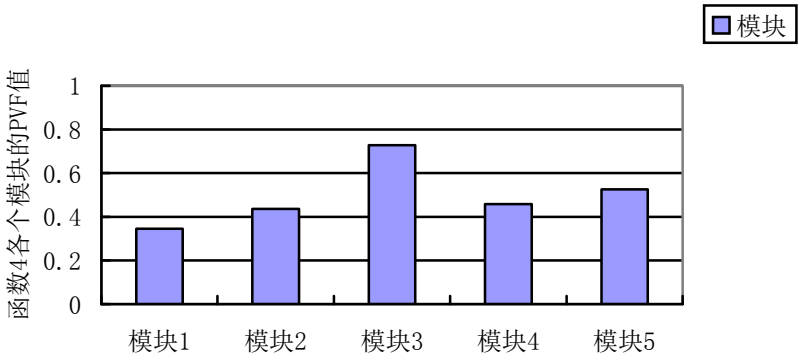
图 4.5 可以发现模块 8 对应的 PVF 值低，通过观察该模块的汇编指令，可以发现该汇编指令中，存在着比较多的 Un-ACE 指令，且还存在那些为了保持字节对齐而出现的 NOP 指令，这些指令对输出结果出错的影响小，因此得到的 PVF 值小。

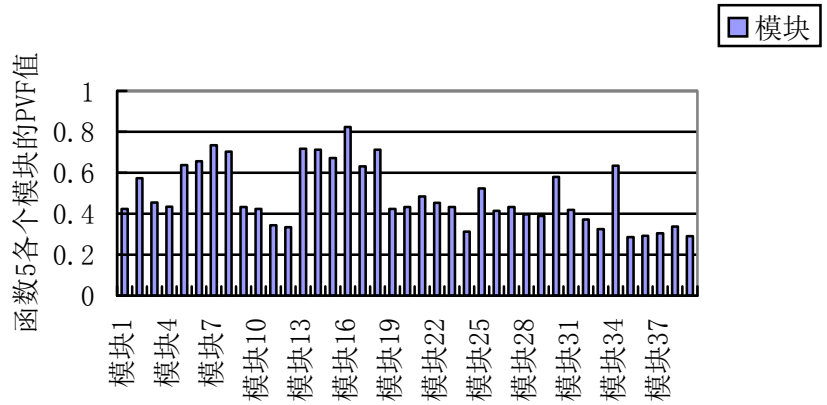
运用同样的方法，对傅里叶变换程序 fft.txt 的计算，在模块划分过程中将该程序划分为 12 个函数，每个函数中包含多个模块，对每个模块进行 PVF 评估，结果如图 4.7 所示。

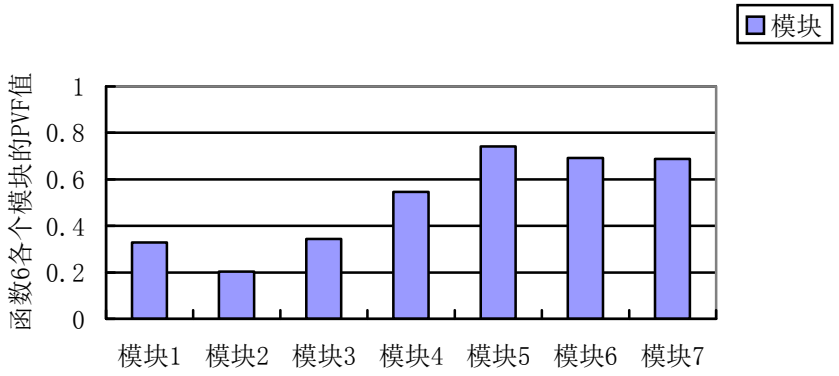

模块
函数1各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6
模块7
模块8
模块9
模块10

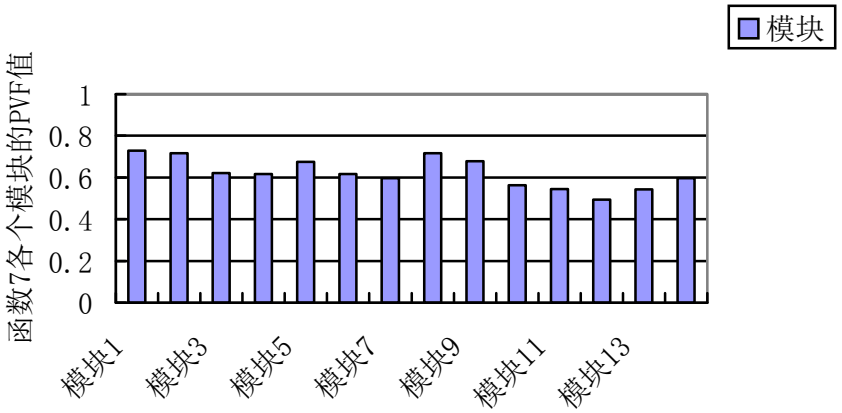

模块
函数2各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6
模块7
模块8

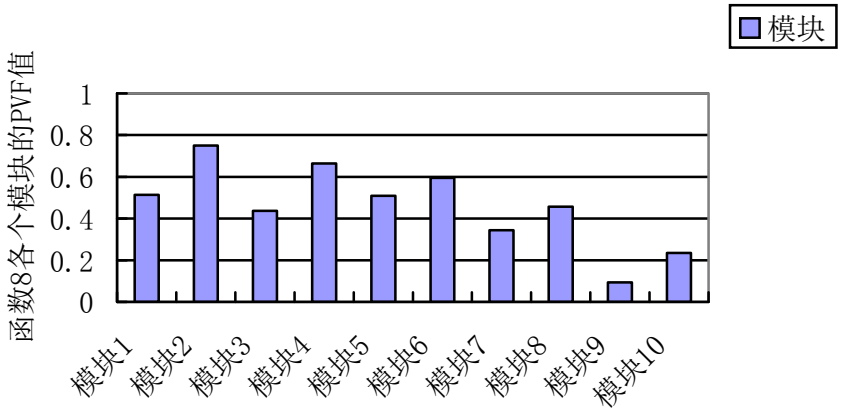

模块
函数3各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块3
模块5
模块7
模块9
模块11
模块13

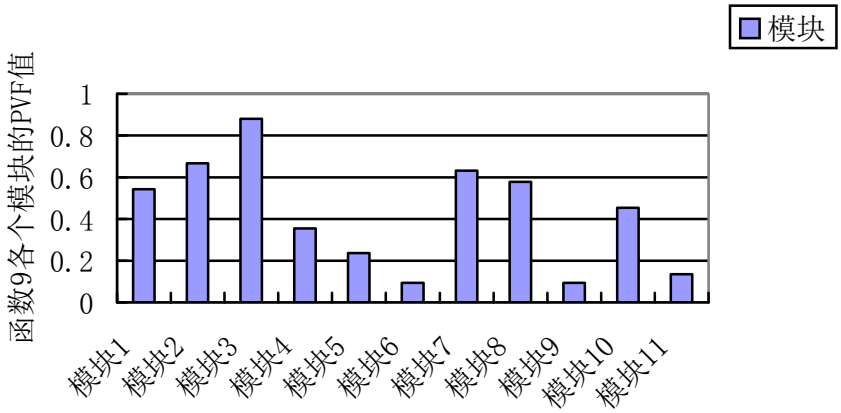

模块
函数4各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5

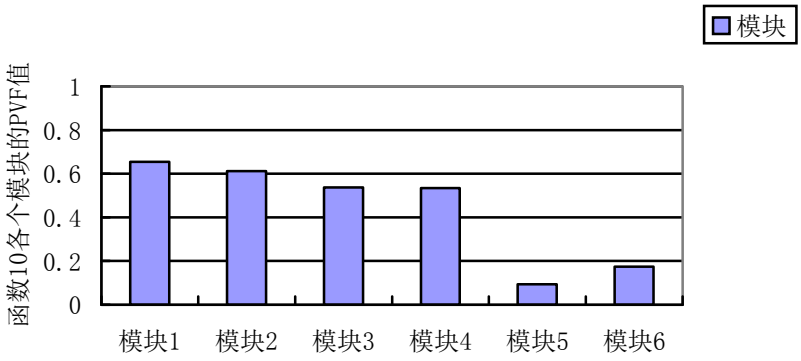

模块
函数5各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块4
模块7
模块10
模块13
模块16
模块19
模块22
模块25
模块28
模块31
模块34
模块37

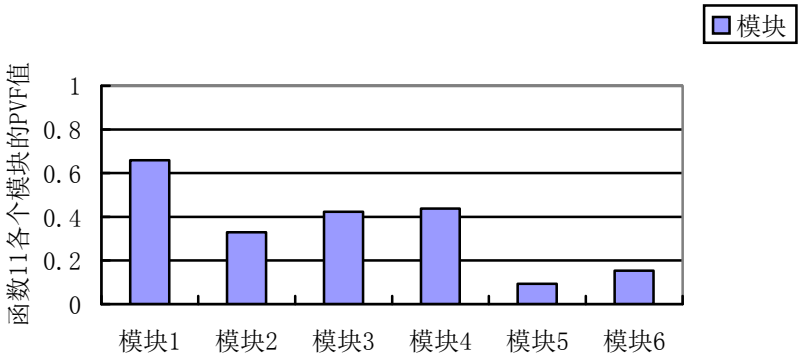

模块
函数6各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6
模块7


模块
函数7各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块3
模块5
模块7
模块9
模块11
模块13


模块
函数8各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6
模块7
模块8
模块9
模块10


模块
函数9各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6
模块7
模块8
模块9
模块10
模块11


模块
函数10各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6


模块
函数11各个模块的PVF值
1
0.8
0.6
0.4
0.2
0
模块1
模块2
模块3
模块4
模块5
模块6

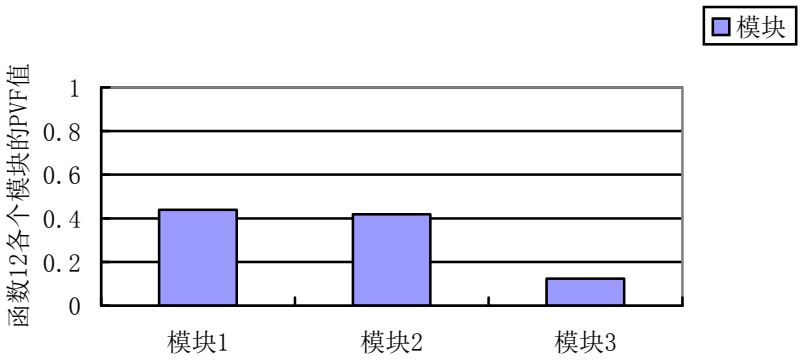


图 4.7 傅里叶变化测试程序中各个函数的 PVF 分布图

从图 4.7 中可以发现，函数 2 和函数 7 中各个模块的 PVF 都很高，在这些模块中 ACE 位所占的比重大，说明这些模块受单粒子翻转的影响严重，对这些程序需要进行重点防护。

除了上面的应用程序外，还选用了多组 DSP 的程序进行了实验，通过对多组测试程序的实验结果分析，可以发现在应用程序中，不同函数的程序脆弱性不同，每个函数中包含的模块数不相同，脆弱性因子和应用程序紧密相关，通过对脆弱性因子的分析可以看出应用程序中的哪些模块受单粒子效应的影响严重，通过对这些模块施加一定的防护措施，减小程序出错的概率。

根据程序的脆弱性，可以计算出模块的软错误率，软错误率值大的模块说明容易出错，根据错误率的值对容易出错的模块有针对性的进行防护。与传统方法相比，该方法计算周期短，不需要重复的注错，而该方法也存在着一定的不足之处，在划分过程中，模块大小的选择是关键问题，每个模块间的错误传播有可能会相互传播，错误传播[49]模型的选取也是评估整个系统的关键因素，还需要进一步研究。

4.4 本章小结

本章主要是对本文所提出的方法的具体实现进行了说明，并对评估过程中需要解决得关键问题给出了分析方法，指出了指令的分类识别过程以及 Un-ACE 位的位数，最后通过实验的方法指出了脆弱性评估的有效性。

第五章 总结与展望

5.1 全文工作总结

本文主要对 DSP 微处理器的可靠性进行了评估，随着 DSP 等微处理器在太空环境中的广泛使用以及微处理器的设计向低功耗和高性能的目标转向，电路小规模趋势的设计和供电电压的降低，使得微处理器受单粒子翻转的影响显著增强，软错误的发生更为频繁，软错误的问题带来了极大的危害，如果不能有效地采取相应的措施降低软错误率，微处理器有可能会出现故障或损坏，因此对软错误的评估显得十分重要。归纳起来，本文主要进行了以下几方面的工作：

（1）分析了目前处理器可靠性评估的方法，总结了传统方法中的不足，并从新的角度提出了针对应用程序的评估方法。传统的故障注入法需要通过大规模的仿真故障注入才能实现，对注错的空间需求的很大，可靠性分析时间特别长，在一定时间段内难以实现。由于选取的注错点的随机性以及注错次数的有限性使得评估的精确性受到一定的限制，为了提高软件注错的效率，在运行特定的周期后，将每次注错的结果与无故障模型的结果进行对照比较，而不是在应用程序结束后才进行观察每次注错的结果，这种情况下观察到的实验结果是不准确的，因为有些软错误并没有立刻表现出来，实际上却存在于系统中，而这种软错误的存在也会对系统的评估的准确性产生一定的影响。本文将指令行为作为研究的切入点，以汇编程序中的指令为目标对象，对应用程序进行代码分析，从对指令集层面进行评估的，不需要深入了解处理器的内部结构，对代码的分析在性能和时间上都优于传统评估方法，因此大大提高了评估效率。

（2）微处理器的脆弱性因子与应用程序紧密相关，本文通过对硬件结构的评估指标 AVF 的研究提出了一种基于程序静态分析的评估方法，并由此来衡量应用程序受软错误的影响。该方法不依赖于具体的硬件结构，克服了现有的只针对特定体系结构进行评估的缺陷，主要是对应用程序的评估。本文通过对汇编指令集的静态分析，对汇编应用程序进行模块划分，计算每个模块的脆弱性，在对脆弱性计算过程中，通过对操作码的识别完成对 NOP 指令等的分类寻找，在对动态死指令识别时建立了指令链表和操作数的消费者-产生者链。针对不同类型的指令分析对应的 Un-ACE 位，完成 ACE 位的采集。

（3）本文根据 DSP C6000 指令集的特点，分析了识别 Un-ACE 位的方法以及评估过程中需要的其他的量，重点分析 C6000 指令集中指令的二进制格式，操作类型、寻址方式等来完成关键信息的采集，并结合提出的评估方法，通过实际的例子对应用程序进行了可靠性计算的研究。

5.2 研究展望

目前对微处理器的可靠性评估研究应用在了设计上，研究者可以根据计算得到的脆弱性评估因子，有针对地进行软错误防护，达到设计者的要求、减少开销以满足所要求的性能指标，还可以通过实时追踪脆弱性因子的发展趋势，设计出更好的系统。而可靠性的评估的研究工作还需进一步的扩展，主要方向如下：

（1）多线程系统方面。当前基本所有的评估研究只针对了单线程的方面，实际上系统的可靠性受很多方面的影响，比如其工作负载等，因此，对多线程系统脆弱性的评估将称为以后的工作侧重点。

（2）间歇性故障的评估。间歇性故障是指故障出现之后，在一定时间内故障消失，然而之后在某一时间点又会出现，这种情况往往是系统出现故障的前期表现，然而现在大部分的可靠性评估都是对瞬态故障而言的，因此需要对间歇性故障的评估进一步探讨研究。

（3）应用方面的扩展。当前脆弱性评估的结果不仅用于微处理器的早期设计中，还用于对系统的防护方面的指导，以后还能够把脆弱性评估扩展到更多的方面，例如文献[50]中，通过与故障注入方相结合，来指导故障注入的注错位置，避免了仿真注错的盲目性，减少资源开销。

参考文献

[1] H. C. Koons, J. E. Mazur, R. S. Selesbiek, *et al*. The Impact of the Space Environment on Space System, American Defence Report (20000509112), 1999.

[2] 王长河．单粒子效应对卫星空间运行可靠性影响[J]．半导体情报．1998, 35(1)：1-8．

[3] Robert Baunla. Soft Errors in Commercial Semiconductor Technology: Overview and Sealing Trends. IEEE 2002 Reliability Physics Tutorial Notes, Reliability Fundamentals, pp.121_01.1-121_01.14, April 7, 2002.

[4] Tutus J L, Wheatley C F. Experimental Studies of Single-Event Gate Rupture and Burnout in Vertical Power Mosfets [J]. IEEE Transactions on Nuclear Science, 1996, 43(2): 533~545.

[5] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, *et al*. "IBM experiments in soft fails in computer electronics (1978 – 1994)," IBM Journal of Research and Development, pp. 3 – 18, Volume 40, Number 1, January 1996.

[6] Christian Boleat, Gerard Colas. Overview of Soft Errors Issues in Aerospace Systems. IOLTS 2005. page(s): 299-302.

[7] H. T. Nguyen, Y. Yagil, N. Seifert, e*t al.* Chip-level soft error estimation method. Device and Materials Reliability, IEEE Transactions. Volume:5. Pages:365-381.

[8] S. S. Mukherjee, Joel Emer, S. K. Reinhardt. The Soft Error Problem: Architectural Perspective. Proc. 1lth International Symposium on High-Performance Computer Architectural (HPCA). 2005: 243-247.

[9] Wang N J, Quek J, Rafacz T M, e*t al.* Characterizing the Effects of Transient Faults on High-Performance Processor Pipeline [C]// Proceedings of the International Conference on Dependable Systems and Networks (DSN). Piscataway, NJ, USA: IEEE, 2004: 61~70.

[10] Mukherjee S S, Weaver C, Emer J, e*t al.* A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor[C]// Proceedings of IEEE/ACM International Symposium on Microarchitecture (Micro'03). Washington, DC, USA: IEEE Computer Society, 2003: 29~40.

[11] Y. Tosaka, S. Satoh, K. Suzuki, e*t al.* Impact of Cosmic Ray Neutron Induced Soft Errors, on Advanced Submicron CMOS circuits. VLSI Symposium on VLSI Technology Digest of Technical Papers, 1996.

[12] N. Seifert, N. Tam. Timing Vulnerability Factors of Sequentials. IEEE Transactions on Device and Materials Reliability, Vol. 4, Nr. 3, P. 516-522, 2004.

[13] H. Ando, Y. Yoshida, A. Inoue, e*t al.* A 1.3 GHZ Fifth Generation SPARC64 Microprocessor.

International Solid-State Circuits Conference, 2003.

[14] 孙峻朝. 基于故障注入的容错机制测评技术的研究，哈尔滨工业大学博士学位论文, 1999.

[15] S. Chau. Fault injection boundary scan design for verification of fault to tolerant systems. Proc. ITC, pp. 667-682, 1994.

[16] J. Clark and D. Pradhan. Fault injection: a method for validating computer-system dependability. IEEE Computer, 28(6): 47-56, June 1995.

[17] V. Sieh, O. Tschache, F. Balbach. VERIFY: evaluation of reliability using VHDL-models with embedded fault descriptions[A]. Proc. PTCS-97[C]. pp. 32-36, 1997.

[18] J. Aidemark, J. Vinter, P. Folkesson, e*t al.* GOOFI: generic object-oriented fault injection tool[A]. Proc. DSN[C]. pp.83-88, 2001.

[19] E. W. Czeck and D. P. Siewiorek. Effects of transient gate level faults on Program behavior. FTCS. pp. 236-243, 1990.

[20] M. Rimen and J. Ohlsson. A study of the error behavior of a 32-bit RISC subjected to simulated transient fault injection. Proc. Int'1 Test Conf., pp. 696-704, 1992.

[21] WANG N J, QUEK J, RAFACZ T M, *et al.* Characterizing the effects of transient faults on a high-performance proeessor pipeline[A]. In Proceedings of the International Conference on Dependable Systems and Networks[C], 2004.

[22] Giacinto P Saggese, Nicholas J Wang, Zbigniew T Kalbarczyk, *et al.* An Experimental Study of Soft Errors in Microprocessors. IEEE MICRO 25:66, 30-39, IEEE Computer Soeiety, 2005.

[23] T. Karnik, P. Hazucha, and J. Patel. Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes. IEEE Transactions on Dependable and Secure Computing, l(2):128-143, June 2004.

[24] H. T. Nguyen and Y. Yagil. A Systematic Approach to SER Estimation and Solutions[A]. In Proceedings of the 4lst IEEE International Reliability Physics Symposium[C], 2003.

[25] P. Shivakumar M. Kistler, S Keckler *et al.* Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic[A]. In Proceeding of the 2002 International Conference on Dependable Systems and Networks [C], pages 389-398, June 2002.

[26] Joel Emer, Pritpal Ahuja, Eric Borch. Asim a performance model framework. Computer, Volume 35, Issue 2. Pages:68-76.

[27] C. Weaver, J. Emer, S. S. Mukherjee, *et al.* Techniques to Reduce the Soft Error Rate of a High-performance Microprocessor[A]. In Proceedings of the 3lst Annual International Symposium on Computer Architecture[C], 2004.

[28] E. D. Lazowska, J. Zahorjan, G. S. Graham, *et al.* Quantitative System Performance,”

Prentice-Hall, Inc, E140nglewood Cliffs, New Jersey, 1984.

[29] LI X, Adve S V, Bose P, *et al.* SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors[C]// Dependable Systems and Networks, 2005. DNS 2005. Proceedings. International Conference on. IEEE, 2005 496-505.

[30] Biswas A, Cheveresan R, Emer J, *et al.* Computing Architectural Vulnerability Factors for Address-based Structures [C]// Proceedings of the International Symposium on Computer Architecture (ISCA). New York, NY, USA: ACM, 2005: 532~543.

[31] Fu X, Li T, Fortes J A B. Sim-SODA: A Unified Framework for Architecture Level[C]// Proc of Workshop on Modeling, Benchmarking and Simulation, 2006.

[32] R. Desikan, D. Burger, S. W. Keckler, *et al.* Sim-alpha: A Validated, Execution-Driven Alpha 21264 Simulator, Technical Report, TR-01-23, Dept. of Computer Sciences, University of Texas at Austin, 2001.

[33] Y. Li, T. Li, T. Kahveci, *et al.* Workload Characterization of Bioinformatics Applications, In Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005.

[34] 成玉，马安国，蒋江等. 用于低开销容错设计的存储部件可靠性评估研究[J]. 电子与信息学报，2011，33(11)：2753-2758.

[35] 潘送军，陈传鹏. 基于占用率的体系结构脆弱因子在线计算方法. 计算机工程与科学，2014.

[36] Somani A K, Trivedi K S. A Cache Error Propagation Model [C]// Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems (PRDC). Piscataway, NJ, USA: IEEE, 1997: 15~21.

[37] Kim S, Somani A K. Area Efficient Architectures for Information Integrity in Cache Memories [C]// Proceedings of the International Symposium on Computer Architecture (ISCA). New York, NY, USA: ACM, 1999: 246~255.

[38] X. Li, S. V. Adve, P. Bose, *et al.* Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumption. Proc. of the 37th International Conference on Dependable Systems and Networks. 2007, 266–275

[39] S. S. Mukherjee. Architecture Design for Soft Errors. Burlington, MA, USA: Elsevier Inc., 2008

[40] A. Benso, S. D. Carlo, G. D. Natale, *et al.* Static Analysis of SEU Effects on Software Applications. Proc. of the International Test Conference. 2002, 500–508.

[41] V. Sridharan, D. R. Kaeli. Eliminating Micro-architectural Dependency from Architectural Vulnerability. Proc. of the International Symposium on High-Performance Computer Architecture. 2009, 117–128.

[42] V. Sridharan and D. R. Kaeli. The effect of input data on program vulnerability. In Selse'09: Proceedings of the 2009 Workshop on Silicon Errors in Logic and System Effects, Stanford, CA, USA, 2009.

[43] 周学海, 余洁, 李曦等. 基于指令行为的 Cache 可靠性评估研究. 计算机研究与发展.

[44] B. Fahs, S. Bose, M. Crum, *et al.* Performance Characterization of a Hardware Mechanism for Dynamic Optimization, Proceedings of the 34th Annual International Symposium on Microarchitecture (MICRO), pp. 16- 27, December 2001.

[45] J. A. Butts and G. Sohi. Dynamic Dead-Instruction Detection and Elimination, 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 199 -210, October 2002.

[46] J. J. Yi and D. J. Lilja. Improving Processor Performance by Simplifying and Bypassing Trivial Computations, In Proceedings of the IEEE International Conference on Computer Design, 2002.

[47] 李方慧, 王飞. TMS320C6000 系列 DSPs 原理与应用(第二版). 电子工业出版社, 2007.

[48] 成玉, 马安国, 张承义等. 微体系结构软错误易感性阶段特性研究[J]. 电子科技大学报, 2012.

[49] 李爱国, 洪炳镕, 王司. 基于错误传播分析的软件脆弱点识别方法研究. 计算机学报. 2007, 18(4): 808-820.

[50] XU X, LI M L. Understanding soft error propagation using efficient vulnerability-driven fault injection[C]// Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFTP International Conference on IEEE, 2012: 1-12.

致谢

时光飞逝，转眼间硕士学习生涯即将结束，在这两年的时光里，不论是科研方面还是为人处事方面都收益良多，借此论文完成之际，谨向所有给予我帮助和关怀的老师、同学、亲人和朋友表示最诚挚的谢意！

首先要衷心感谢我的导师郭宝龙教授，感谢他一直以来在学习和生活中对我的悉心指导和关怀。郭老师高深的学术造诣、精益求精的科研作风，对学科前沿的敏锐洞察和正确把握，对学生谆谆教导的态度，我由衷的感到敬佩，在学习和生活中，他鼓励我们迎难而上、积极创新，郭老师传授给我们的治学方法和教导我们的为人品格，将是我们人生中的宝贵财富。在此，我要再次对我的导师表达我最崇高的敬意和深深地谢意！

感谢智能控制与图像工程研究所给我提供的良好的环境，感谢闫允一老师、孟繁杰老师、吴宪祥老师、孙伟老师以及朱娟娟老师的指导和关怀。他们丰富的科研经验、对问题独到的见解和一丝不苟的科研态度是我以后学习工作的榜样。感谢实验室的张旭师兄、陈龙师兄、吴进福师兄、毛毅师姐等师兄师姐在生活上和学习上对我的关心和帮助。

感谢同年级的徐芳、朱新刚、贺元晨、桑林海、张西南、石强、白文杰、郑文玲、黄旭楠、高洪松、宁伟康博士以及赵丹博士等同学，他们陪伴我走过研究生这几年丰富多彩的生活，感谢实验室师弟师妹们，与他们的学术交流和讨论让我受益匪浅，他们每个人身上都有值得我学习的地方，感谢我的舍友徐芳、张艳倪、武冬平，我们一起度过了一段充实而短暂的时光，朝夕相处的研究生生活让我们结下了深厚的友谊。特别感谢寇宏达同学在研究生学习期间给我的关怀、爱护和帮助。

深深感谢我的父母和家人，感谢他们多年来对我含辛茹苦的养育和无微不至的关怀，感谢我的父母给予我的支持和鼓励让我在前进的道路上勇往直前，是我前进的不竭动力。

最后向在我成长道路上所有关心过和帮助过我的人表示衷心的感谢！

作者简介

1. 基本情况

女，河南许昌，1989 年 9 月出生，西安电子科技大学空间科学与技术学院控制理论与控制工程专业 2012 级硕士研究生。

2. 教育背景

2008.08~2012.07	安阳工学院电子信息与电气工程学院自动化专业，获工学学士学位
2012.08~	西安电子科技大学空间科学与技术学院控制理论与控制工程专业硕士研究生

3. 攻读硕士学位期间的研究成果

3.1 发表的学术论文

[1] Baolong Guo, Guochang Zhou, Yufang Cao. *etal*. Analysis of AVF evaluation methods for Microprocessor reliability.(Signal Processing Research(SPR), 2015，已录用)

3.2 发明专利和科研情况：

[1] 参与国家重点科研项目 XX 的设计，2013．11～至今，即将完成，本人完成 DSP 模块的软错误率计算工作，用以评估单粒子防护方法施加前后应用程序的错误率，便于有针对性地进行防护.

[2] 国家自然科学基金项目，基于局部不变特征和混合多示例学习的图像检索研究，2013．4～至今，即将完成，本人负责前期的资料收集整理工作，通过在 Matlab 平台下，对图像局部特征的提取，并结合多示例学习来完成图像检索.

西安电子科技大学

XIDIAN UNIVERSITY

地址：西安市太白南路2号

邮编：710071

网址：www.xidian.edu.cn