硕士学位论文

处理器中流水线和寄存器堆抗辐射加固技术研究

RESERARCH ON RADIATION HARDENED TECHNOLOGY OF PIPELINE AND REGISTER FILE IN PROCESSOR

王远刚

哈尔滨工业大学

2019 年 6 月

国内图书分类号：TN47　　学校代码：10213
国际图书分类号：621.3　　密级：公开

工学硕士学位论文

处理器中流水线和寄存器堆抗辐射加固技术研究

硕 士 研 究 生：王远刚
导　　师：肖立伊教授
申 请 学 位：工学硕士
学　　科：微电子学与固体电子学
所 在 单 位：航天学院
答 辩 日 期：2019 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index: TN47
U.D.C: 621.3

Dissertation for the Master Degree in Engineering

RESERARCH ON RADIATION HARDENED TECHNOLOGY OF PIPELINE AND REGISTER FILE IN PROCESSOR

Candidate： Wang Yuangang
Supervisor： Prof.Xiao Liyi
Academic Degree Applied for： Master of Engineering
Speciality： Microelectronics and Solid State Electronics
Affiliation： School of Astronautics
Date of Defence： June, 2019
Degree-Conferring-Institution： Harbin Institute of Technology

摘　要

随着集成电路和航天技术的快速发展，集成电路中的器件尺寸进入纳米量级，工艺的提升使晶体管中含有的掺杂粒子越来越少，导致晶体管携带的电荷量减少，因此集成电路容易受到空间粒子辐射出现软错误。而处理器是宇航应用中的核心控制部件，主要用于控制整个系统并对大量的数据进行处理，如果不加防护，容易受到辐射产生故障。随着工艺的提升，芯片上的晶体管密度升高，存储单元之间的间隔越来越小，辐射导致存储体多位翻转（Multiple Bits Upsets，MBU）的发生概率越来越高。而处理器对速度、面积和功耗开销的要求较高，因此在较小的开销条件下对处理器中流水线和寄存器堆进行抗辐射加固设计具有重要意义。

针对处理器中的流水线，本文提出了交错奇偶校验结合流水线重启和少量信号采用三模冗余的加固方案，基于 OR1200 处理器平台，本文对加固后流水线的功能进行验证并对性能进行评估。相对于未加固的处理器，使用该方法加固后的流水线，能够有效抵抗单粒子翻转，关键路径的延时开销为 7.66%，面积和功耗开销分别为 16.43%和 14.11%。本文构造一种能够纠正相邻 4 位内随机错误的错误纠正码对处理器中的寄存器堆加固，并采用触发异常的方式对寄存器堆刷新，基于 OR1200 处理器，经过综合评估，使用该方法加固寄存器堆不影响处理器的关键路径延时，加固后处理器的面积和功耗开销为 10.10%和 16.40%。为了评估寄存器堆的加固效果，本文提出一种多位故障注入模型并进行实现，测试结果表明加固后的寄存器堆可以纠正相邻 4 位内的随机错误。

关键词：交错奇偶校验；多位翻转；流水线；寄存器堆；错误纠正码

Abstract

With the rapid development of integrated circuits and aerospace technology, the size of devices in integrated circuits has entered the nanometer scale. The improvement of technology has made the transistors contain less and less doped particles, which result in a reduction in the amount of charge carried by the transistors, so the integrated circuit is susceptible to soft errors in the radiation of the spatial particles. The processor is the core control component in aerospace applications, which is mainly used to control the entire system and calculate a large amount of data. If it is not protected, it is susceptible to radiation failure. With the improvement of technology, the density of transistors on the chip increases, and the distance between the memory cells becomes increasingly smaller. The radiation lead to a rise in the probability of multiple bits upsets (MBU) of the memory. The processor has quite high requirements on speed, area and power consumption, so it is of great significance for the research of pipeline and register file radiation hardened design under the condition of low overhead.

Based on the OR1200 processor platform, this paper proposes a technique to protect most signals by using the interleaved parity codes combined with the pipeline restart and partial signals by using the Triple Modular Redundancy (TMR) for the pipeline in processor. This paper verifies the function and evaluates the performance of the hardened pipeline. Compared with the unhardened processor, the method of hardening pipeline proposed in this paper can effectively suppress Single Event Upset (SEU), and the critical path delay overhead of pipeline is 7.66%, and the area and power overhead are 16.43% and 14.11%, respectively. This paper proposes a kind of Error Correction Codes (ECC) that can correct random errors in the adjacent 4-bit to harden register file. And based on OR1200 processor, the register file is refreshed by triggering an exception. After comprehensive evaluation, the method of hardening register file proposed in this paper does not affect the critical path delay of the processor, and the area and power overhead are 10.10% and 16.40%. In order to evaluate the harden performance of the register file, this paper proposes a multi-bit fault injection model and implements it. The final results indicates that the hardened register file can correct random errors in the adjacent 4-bit.

Keywords: Interleaved parity; Multiple bits upsets; Pipeline; Register file; Error correction codes

目　录

第1章 绪　论

1.1 论文背景与意义

随着集成电路和航天技术的发展，IC 中的器件尺寸进入纳米量级，工艺技术的提升使得相同面积的 IC 中能够集成越来越多的应用[1,2]，随着晶体管体积的缩小和工作电压的降低，IC 中的晶体管受单粒子效应的影响显著提高[3-5]，另外 IC 中计算能力的提高也使得电路对可靠性的要求越来越高[6]。晶体管特征尺寸的不断缩小，使得一个晶体管中仅含有几十个掺杂粒子[7]，工艺波动和工作温度的波动增加了 IC 定时误差的可能性，晶体管携带的电荷量的减少，也使得电路更容易受到来自 IC 封装中宇宙射线和 α 粒子的影响，导致在集成电路中出现软错误，使整个系统失效。

空间环境与地面环境相比更为复杂，因为在空间环境中有大量的高能粒子，而航天应用中的各种器件围绕地球运动时，会受到空间中各种高能粒子的冲击辐射，因此会产生各种辐射效应[8-11]。当单个辐射粒子撞击组合逻辑的某个节点时，撞击所产生的载流子可能被晶体管的源级/漏级扩散区收集，导致节点上的电压扰动，这种瞬态错误可能表现为电脉冲的形式，这就是单粒子瞬变（Single Event Transient, SET）效应。另一种情况，当单个粒子撞击时序单元内部的节点，如果沉积的电荷量超过节点上的临界电荷则可以翻转存储元件的值，这就是单粒子翻转（Single Event Upset, SEU）效应[12,13]。

处理器是宇航应用中的核心控制部件，主要用于控制整个系统并对大量的数据进行处理，如果没有任何防护措施，则容易受到空间电磁波和高能粒子的冲击辐射。辐射效应使存储在存储单元（例如寄存器堆和程序计数器）中的数据发生翻转，导致处理器执行错误的指令或者重新执行指令，有时甚至会让内部的程序陷入死循环，最终可能会导致整个航天器无法正常工作[14]。由于航天器在围绕地球做运转，当发生故障时很难收回并进行维护修复，因此一旦在整个运行过程中由于辐射发生故障，可能整个航天器会因此报废，这将给航天事业的发展带来巨大的损失。所以，对处理器进行抗辐射加固设计是计算机系统设计的重要组成部分[15]。因此，在高可靠需求的应用环境（如宇航应用和核辐射环境）中需要对处理器抗辐射加固处理，以较小的延时、面积和功耗开销使电路能够抵抗软错误。

高能粒子轰击严重挑战了当今嵌入式处理器的可靠性，随着工艺技术的提升，晶体管的特征尺寸进入纳米量级，相同面积的芯片上能够集成越来越多的元器件，集成度的增高让存储单元之间的距离越来越小，距离的减小导致单个粒子辐射的能量可能对多个存储单元造成影响，相邻存储单元的状态因此发生改变，这就是多位翻转（Multiple Bits Upsets, MBU），MBU 会使存储体面临的可靠性问题日益严重[16-20]。存储结构被认为是处理器中多位错误的主要来源，这些包括高速缓存、寄存器堆和动态随机存取存储器（Dynamic Random Access Memory, DRAM），在文献[16]中，据报道由于中子碰撞，进入深亚微米的静态随机存取存储器（Static Random Access Memory , SRAM）中可能会出现高达 4 比特的翻转。据文献统计发现，在商用 130nm 工艺下的 SRAM，辐射效应引起的软错误中，SEU 发生的概率为 93%，MBU 发生的概率为 7%，而在 MBU 中，产生的大部分是两位翻转，少部分是三位翻转。文献[21]数据表明，Los Alamos Neutron Science Center 对于 90/65nm 下的 5/8Mbit 嵌入式 SRAM 测试芯片进行测试，采用中子束照射，与 90nm 工艺相比，65nm 工艺下的 SRAM 中，MBU 发生的概率显著增高，其中 SEU 发生的概率约为 50%，两位翻转发生的概率约为 20%，三位翻转发生的概率约为 8%，四位翻转发生的概率约为 18%，还有可能发生更多位数翻转，以上数据表明，随着工艺的提升，MBU 发生的概率越来越高。

1.2 流水线和寄存器堆加固技术研究现状

当面临可靠性问题时，已有多种不同的策略去避免、检测和恢复软错误[22]，解决方法适用于系统的各个层面，从工艺级、电路级到微架构级别。硬件冗余是常用的加固方法，这其中包含错误检测和校正（Error Detection And Correction, EDAC）、奇偶校验、三模冗余等低层次结构，又包括功能单元、协处理器、甚至利用多核/多线程架构上的多个硬件模块[25]。大多数硬件加固方法为解决瞬态故障提供了高效的解决方案。然而，在成本的限制下，这些技术在一些情况下是不可行的。

1.2.1 国外研究现状

硬件冗余可以用于检测错误，如果全部采用双模和三模冗余加固，具有较高面积和功耗开销的缺点。Protolan 等人在文献[26]提出一种基于奇偶校验检测的流水线保护机制，当寄存器发生翻转，则在流水线进入下一级之前暂停流水线，其

他阶段继续执行而没有性能损失。Ernest 等人提出针对时序错误可靠的流水线机制，通过额外的锁存器来对流水线级寄存器进行双倍采样，当检测到主锁存器有发生错误时，则从备份寄存器中恢复正确的值，每个错误需要一个时钟周期的开销。

Nicolaidis 等人提出了时间冗余的技术，可以检测单个时序错误、SET 和 SEU 错误。如图 1-1 所示，在每个需要保护的寄存器中添加备份寄存器，当 SEU 命中主寄存器和备份寄存器中的任何一个，比较器都能检测出错误，纠正错误时采用回滚的方式。文献[6]提出在主寄存器和备份寄存器中添加错误纠正码（Error Correction Codes, ECC），如图 1-2 所示，这样如果多个 SEU 发生在主寄存器和备份寄存器的相同位置，仍能检测到错误，但是图 1-1 所示方法无法检测。

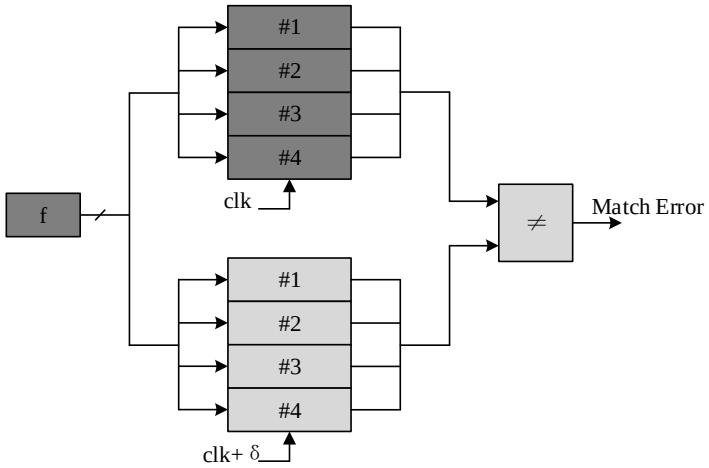


图 1-1 流水线中使用备份锁存技术

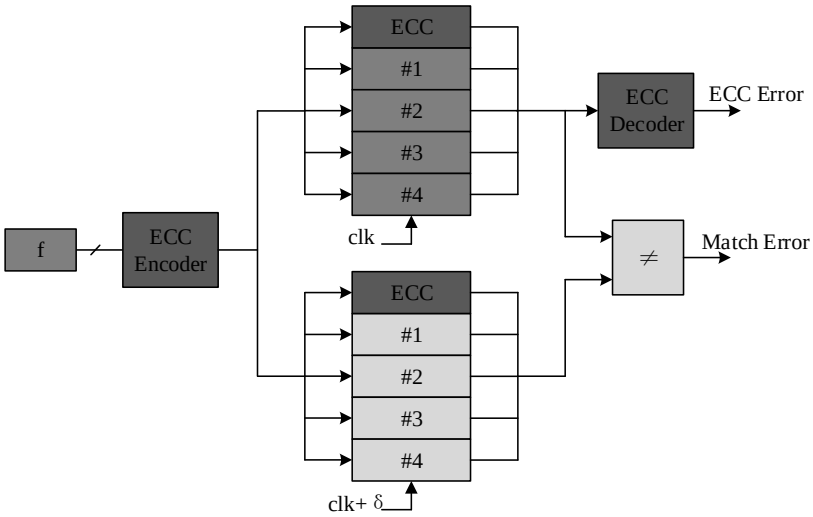


图 1-2 流水线中使用扩展 ECC 的备份锁存技术

在寄存器堆中防止 MBU 的一个有效方法是在寄存器堆中使用电路级保护的容错存储单元[27-29]。每个存储器单元都被保护免受粒子撞击，因此寄存器堆对于多个粒子撞击（即发生 MBU）变得高度稳定。然而，如果这种方法被用来保护整个寄存器堆，则会带来相当高的功耗和面积开销。

减少多位翻转现象最常用的方法是版图位交错技术。在版图上，将属于同一个字的各比特位分别放置到版图中不相邻的存储单元中，当单粒子引发多位翻转时，实际翻转的比特位在逻辑上属于不同的字，再结合其他一位故障修复技术，可以实现多位翻转的纠正，这种方法可以显著降低软错误率[26]。然而这种技术不适用于所有存储器[30,31]。因此，对于存储器阵列仍然需要复杂的错误检测和校正技术来应对 MBU。

由于版图位交错技术不适用于所有的存储器，在设计阶段实现的交错奇偶校验在多位翻转检测技术得到广泛应用[32]。这种技术的基本原理是将数据位在逻辑层次划分成多个组，划分组的个数经常与纠正能力和编码速度相关，每个组对应的数据彼此之间的距离相同，每一个组的数据由一个冗余位检测。

对所有数据位应用 ECC 是减少存储器阵列中软错误的常用方法。在这种方法中，存储器的每位数据都附加有 ECC 校验，并且在每次读取和写入操作期间，分别执行奇偶校验和纠错。最常见的纠错码是可以实现一位错误校正和两位错误检测（Single Error Correction-Double Error Detection, SEC-DED）的汉明码，这种方法现已广泛应用于商业处理器之中。文献[33]基于 SEC-DED 码，通过对奇偶校验向量中校正子的特殊排列设计，在没有增加冗余位数的前提下，提出一种能够纠正单个错误和相邻两位错误的 SEC-DAEC（Single Error Correction-Double Adjacent Error Correction）码。

因为寄存器堆处于处理器流水线中的关键位置，为了减小访问寄存器堆的时间开销，提高编译码电路的速度，Saizadalid L 等人通过增大冗余的方式，在文献[34]和[35]中分别提出了超快速的（16,8）SEC 码、（16,8）SEC-DED 码和（16,8）SEC-DAEC-DED 码。

1.2.2 国内研究现状

三模冗余（Triple Modular Redundancy, TMR）是一种现有已经成熟商用的加固技术，这种技术易于实现且可靠性较高而被广泛采用，而且由于不同的系统对可靠性的要求不同导致所使用的冗余的模数不同，基于此可以将冗余扩展到五模

或者七模，一般来说冗余的模数越多可靠性越高。对于三模冗余来说，如果其中一个单元发生故障，输出错误数据，那么表决器通过选取多数的原则，屏蔽少数故障单元的错误，从而得到正确的输出结果。

文献[36]三模冗余在高性能抗辐射 DSP 中的应用，基于国产高性能 DSP“魂芯”的架构，对片上的 SRAM 加固，主要通过对 SRAM 结合使用三模冗余和自刷新技术，可以对 SEU 进行有效抑制。

在数字通信中，为了提高数据传输的稳定性，EDAC 得到广泛应用。目前，为了提高微处理器的可靠性，一般都结合 EDAC 技术进行加固，而最近几年在国内通过编码方式对存储体加固成为研究的热点。

文献[37]使用系统级加固方法对处理器进行加固，主要包括错误检测与修复，常用错误检测方法包括奇偶校验，基于可靠性和性能的综合考虑。龙芯 X 采用最简单有效的检错纠错机制，对存储模块采用奇偶校验和 ECC 保护。

文献[38]针对于 SEU，对数字太敏 SoC 进行抗辐射加固研究，该文献中搭建了 Leon3 处理器平台，在此基础上设计了数字太敏 SoC 结构，并使用评测工具提取了其中的敏感单元，其综合考虑整体的性能，确定了相应的加固方案，其中针对于寄存器堆，其采用了分组汉明码，通过流水线重启对流水线进行纠错，针对于 Cache 存储体，该文献使用了循环冗余校验（Cyclic Redundancy Check， CRC），通过 Cache 强制不命中并读取内存的方式进行加固，另外对于普通的寄存器均采用 TMR 加固实现，在 FPGA 上予以验证得到良好的效果。

文献[10]搭建了 LEON3 处理器平台，并对外部存储器控制器进行加固，该文献使用了扩展的(45,32)BCH 码，该编码可以纠正存储体字上发生的任意两位错误，另外，该编码也能对三位错误进行检测，但是不能纠正。

文献[39]一种检测和校正存储双模的低冗余加固方法中，通过对传统的 SEC-DED 码进行适当的改进和优化，提出一种新特征结构的单错误校正-双错误检测-相邻双错误校正（Single Error Correction, Double Error Detection and Double Adjacent Error Correction, SEC-DED-DAEC）码，能够有效的对宇航用存储器抗辐射加固。

文献[40]搭建了 OR1200 处理器平台，使用混合双模冗余结合三模冗余方式对流水线进行加固，其中使用双模冗余检测流水线中发生的 SEU 类型错误，并通过重启流水线的方式对错误进行纠错。针对寄存器堆，文献[40]使用了二维奇偶校验码检测寄存器堆中发生的 MBU，并使用软件纠错的方式进行纠错。

1.2.3 国内外文献综述简析

目前，国内外在抗单粒子翻转软硬件加固方法的研究上已经比较成熟。硬件加固方法通常是在原有结构基础上增加冗余逻辑。硬件加固能有效校验和纠正软错误，但如果大面积使用冗余逻辑往往使得加固后面积过大，导致更多的功耗和生产成本。针对流水线，目前主要是使用双模冗余或结合 ECC 进行检测加固，这种方法加固后寄存器冗余量会增大一倍多，导致加固后的面积过大，相应的功耗和成本增大。

针对于流水线硬件冗余加固的缺点，可使用硬件对电路进行检错，检错电路可采用交错奇偶校验，一方面可以快速的检测错误，另外与双模冗余和三模冗余方式相比，减小了芯片的面积的同时降低了功耗，另外通过流水线重启的方式重新执行未完成的指令，完成对错误的纠正。

在国内外研究中，使用编码对存储体加固成为研究的一个热点。然而，传统的编码方案（例如汉明码、SEC-DED 码）大多只能纠正 1 位错误或纠正一位错误检测两位错误，已经不能满足 MBU 对存储体带来的影响，文献[39]提出的 SEC-DED-DAEC 码只能纠正一位错误或相邻两位错误并检测随机两位错误，文献[10]使用的扩展（45，32）BCH 码，仅能纠正两位错误，针对于三位错误只能进行检测，几种方法都难以应对因工艺进步导致的 MBU。传统的奇偶校验有着编码和校验速度快的优点，但只能检测奇数个错误，就此有人提出了交错奇偶校验的方法。为了使奇偶校验能够纠正 MBU，有人提出矩阵码的加固方法，但是当纠正能力增强的时候，存在冗余位过多的问题，面积和功耗开销较大。因此需要一种有着较小面积和功耗开销，能够抵抗多位翻转且冗余相对较低的加固方法。

1.3 本论文主要研究内容和论文结构

本文的主要研究内容为三方面，首先是交错奇偶校验加固流水线的技术，本文基于开源处理器 OpenRISC，搭建相关的开发平台，结合交错奇偶校验技术检测流水线中发生的 SEU 类型错误，并通过重启流水线的方式对故障进行处理。然后本文对低冗余矩阵码和寄存器堆加固技术进行研究，通过对现有文献的分析，构造了低冗余矩阵码，构造的低冗余矩阵码具有较高的纠正邻近错误的能力，同时编译码速度较快，在不影响编译码速度的前提下最大限度的降低了冗余度，比较适合寄存器堆的加固，同时本文采用触发异常的方式刷新寄存器堆，对寄存器堆中发生的 MBU 进行纠正。另外为了在寄存器传输级（Rigester Transfer Level, RTL）

评估抗多位翻转的加固效果，本文对多位故障注入技术进行研究，提出多位故障注入模型并进行实现，对使用低冗余矩阵码加固寄存器堆的处理器进行抗辐射能力评估。

全文由四章组成，安排如下：

第 1 章绪论，首先介绍了课题的研究背景和意义，然后简要介绍了国内外对流水线和寄存器堆的加固技术，并对现有的技术进行综述分析，然后介绍了本论文的研究内容和论文的组织结构。

第 2 章交错奇偶校验加固流水线设计，首先介绍了流水线的工作原理，然后对流水线的加固设计实现进行详细介绍，本文采用交错奇偶校验的方式对流水线中寄存器发生的 SEU 类型错误检错，然后通过触发异常的方式重启流水线，提出流水线加固的抽象结构，然后本章节验证了交错奇偶校验加固流水线的功能，并使用现有的抗辐射能力评测工具对加固后流水线的抗辐射能力进行评估，同时使用 DC 工具对其性能进行评估。

第 3 章低冗余矩阵码设计与寄存器堆加固，首先介绍了寄存器堆的工作原理，并对寄存器堆的读写过程进行分析，然后基于寄存器堆的结构和对性能的要求，本文构造了 32 位数据位的低冗余矩阵码，并详细描述了低冗余矩阵码编译码电路的构造过程，分析了所构造低冗余矩阵码的数据存放要求和可纠正的错误模式，并将构造的低冗余矩阵码与现有典型的编码和最新的 ECC 在性能上进行比较分析。基于构造的低冗余矩阵码，本文对寄存器堆加固，同时介绍了寄存器堆的纠错机制，最终对加固后的功能进行验证分析。

第 4 章多位故障注入设计与寄存器堆抗辐射加固能力评估，首先介绍了现有故障注入工具的工作流程，然后通过对 45nm SRAM 多位翻转错误图样的综合分析，提出多位故障注入的模型，详细介绍了多位故障注入的设计与实现，然后分别对设计的多位相邻连续错误和多位相邻随机错误的功能进行验证分析，本章最后对寄存器堆的抗辐射加固能力进行评估分析。

结论对本文的主要研究成果进行总结概括。

第2章 交错奇偶校验加固流水线设计

2.1 流水线工作原理

2.1.1 流水线简介

关于计算机中的流水线，是根据计算机中的指令特征，划分成几个不同的功能阶段，并设计成不同的功能单元，通过并行执行来加快执行速度。因此流水线的关键在于如何将模块拆分和并行，从直观上分析可知，指令的处理可以分为 5 步，分别是从指令存储器中取出指令、对指令进行译码生成相应的控制码、按照译码的结果执行，将执行的结果写入数据存储器或者回写到寄存器堆，简单描述就是：取指、译码、执行、访存和回写。如果仅有一个硬件单元，那么该单元则需要同时进行取指、译码、执行、访存、回写，假设五种操作需要的时间都是 T，那么处理一条完整的指令需要的时间为 5T，n 条指令需要的时间为 5nT，而如果采用五个硬件单元，则使用并行的方式在同一时间让各个单元分别做这五项工作中的一项，那么在访存的时候，可以对下一条指令进行执行，同时对再下一条指令进行译码，同时对再下一条指令进行取指。如图 2-1 所示为五级流水的示意图。

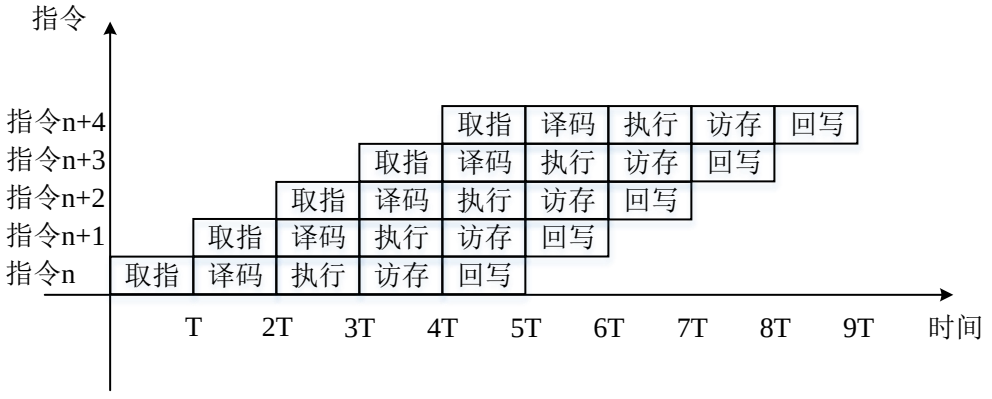


图 2-1 五级流水示意图

2.1.2 OR1200 流水线的工作过程

如图 2-2 所示是开源处理器 OR1200 的数据通路，由数据通路可知，OR1200 主要包含 GENPC 模块、指令存储器模块、IF 模块、RF 模块、OPERAND_MUX 模块、ALU 模块、SPRS 模块、LSU 模块、WB_MUX 模块、数据存储器模块和

EXCEPTION 模块，下面根据数据通路简要介绍各个模块的功能。

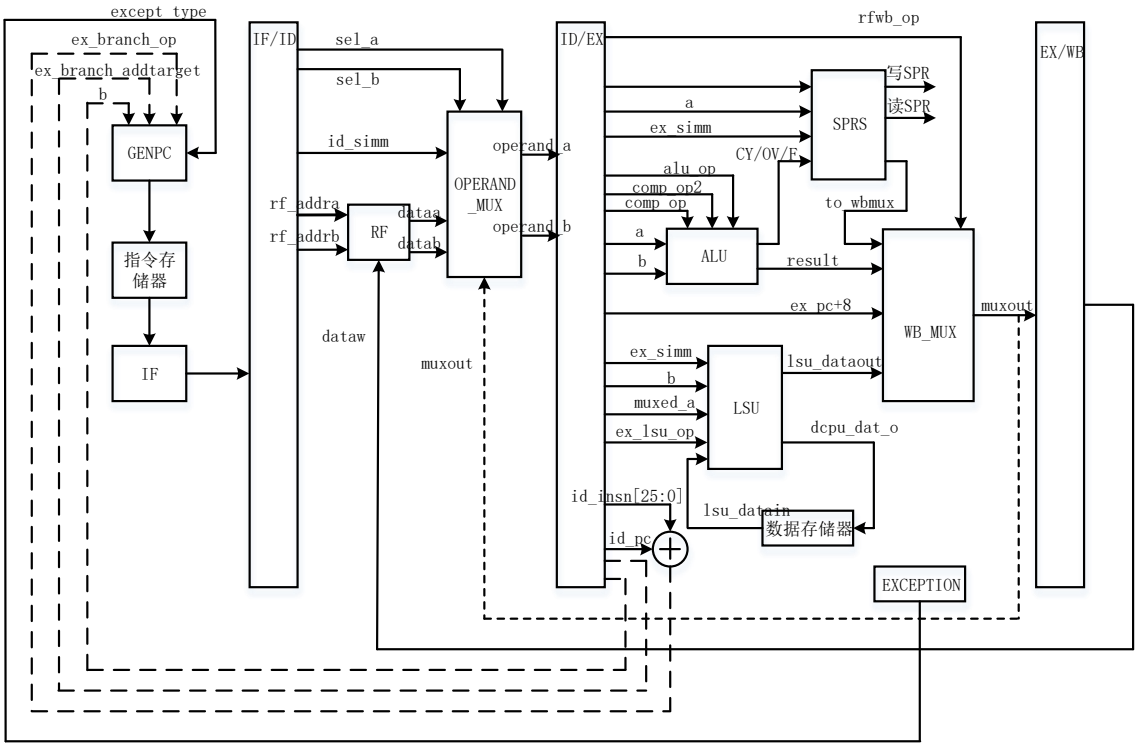


图 2-2 OR1200 数据通路图

GENPC 模块，主要是生成下一个 PC 地址，会根据跳转指令和异常处理类型寄存器的值计算程序计数器 PC 地址，然后将生成的 PC 地址传给指令存储器。

指令存储器模块存储需要执行程序的机器指令。

IF 模块，功能是从指令存储器中取出需要执行的指令，并输出寄存器堆的地址信号和读写信号。

RF 模块，为处理器的寄存器堆模块，存储指令执行过程的中间数据。

OPERAND_MUX 模块，解决流水线工作过程中的数据相关问题，从寄存器堆中读出的数据、扩展的立即数、执行阶段的输出结果和回写阶段的输出结果选择 ALU 所需要的操作数。

ALU 模块，根据输入的操作数和译码阶段生成运算控制码进行相应的运算，如算数运算、逻辑运算和移位操作等。

SPRS 模块，包含特殊寄存器，控制读写特殊寄存器。计算数据加载或者存储所需要的目的地址，存储数据时会将数据传送到 Wishbone 总线，加载数据会接收来自 Wishbone 总线上的数据。

WB_MUX 模块，会根据相应的操作码，从 SPRS 模块的输出结果、ALU 模块的输出结果、LSU 模块的输出结果或者跳转指令的最终计算结果中选择数据输出，

相当于一个多路选择器。

数据存储器模块，用于存储程序运行过程中所使用的数据或者输出的结果。

EXCEPTION 模块，异常处理模块主要用于异常处理，根据输入的异常信号，输出一系列的异常控制信号，包括异常类型和流水线各个阶段的清空信号等，该模块会设置 EPCR、EEAR、保存 SR、设置新的 SR 并控制跳转到异常处理程序。

以加法指令为例来介绍流水线的工作过程，加法指令 l.add 的用法为：l.add rD,rA,rB，代表的意思是将寄存器 rA 和 rB 中的存储的数据相加，计算的最终结果保存到寄存器 rD 中，同时将该指令的进位标志和溢出标志保存到特殊寄存器 SR 中。如图 2-3 所示表示加法指令的指令格式。其中 31~26 位表示指令的类型，例如 ox38 表示加法指令，25~21 位表示目标寄存器的地址，在回写阶段使用，20~16 位和 15~11 位分别表示源操作数 a 和 b 的地址，可以根据这两个地址从寄存器堆中取操作数，10 位表示保留位，9~8 位表示移位指令的操作类型，7~4 位为保留位，3~0 位表示具体的数据处理类指令类型。

31 26	25 21	20 16	15 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	rd	ra	rb	reserved	opcode 0x0	reserved	opcode 0x0
6 bits	5 bits	5 bits	5bits	1 bits	2 bits	4 bits	4bits

图 2-3 加法指令格式

以 l.add r4,r2,r3 指令为例，对应指令的二进制代码为 0xe0821800，其解释如表 2-1 所示，指令所完成的工作为：

表 2-1 l.add r4,r2,r3 指令的二进制数解释

数据字段	字段名称和含义	字段数值
31~26	opcode 0x38	0x38
25~21	rd	0x4
20~16	ra	0x2
15~11	rb	0x3
10	保留	0x0
9~8	opcode 0x0	0x0
7~4	保留	0x0
3~0	0x0	0x0

r4[31:0] = r2[31:0] + r3[31:0];

SR[CY] = carry;

SR[OV] = overflow;

取指阶段，该指令的组合逻辑输出，将读取寄存器 ra 的序号 0x02 给 rf_addra 信号，将读取寄存器 rb 的序号 0x03 给 rf_addrb 信号，将是否读取寄存器 ra 和寄存器 rb 的使能信号分别给 rf_rda 和 rf_rdb。该指令的时序输出，表示是否选择立即数作为 ALU 的操作数 b 赋给信号 sel_simm，该阶段得到转移指令的具体类型，并赋给 id_branch_op 信号，同时将取指阶段的指令 if_insn 赋给 id_insn 信号。l.add 取指阶段的主要工作如下：

(1) 为读取寄存器堆做准备，输出寄存器堆的地址信号和读使能信号；

(2) 判断指令是否需要选择立即数作为 ALU 的操作数 b；

(3) 指令进入译码阶段。

译码阶段组合逻辑输出如表 2-2 所示，

表 2-2 l.add 指令在译码阶段的组合逻辑输出

信号名称	信号作用	信号值
dataa	ra 中所存的数据	寄存器堆中地址 0x2 存储的数据
datab	rb 中所存的数据	寄存器堆中地址 0x3 存储的数据
id_branch_addrtarget	转移指令所用到的信号，跳转的目的地址	{{4{id_insn[25]},id_insn[25:0]}+id_pc[31:2]}
sel_a	选择 ALU 的操作数 a	OR1200_SEL_RF
sel_b	选择 ALU 的操作数 b	OR1200_SEL_RF
id_simm	指令需要的立即数	{{16'b0},id_insn[15:0]}
muxed_a	暂存 ALU 的操作数 a	寄存器堆中地址 0x2 存储的数据
muxed_b	暂存 ALU 的操作数 b	寄存器堆中地址 0x3 存储的数据
multicycle	指令执行是否需要多个周期	0
id_lsu_op	确定加载存储指令的操作类型	OR1200_LSU_NOP（空）
rfe	是否是异常返回指令	0

l.add 译码阶段组合逻辑的主要工作：

(1) 读出指令需要源寄存器的值，输出到 OPERAND_MUX 模块；

(2) 给出立即数的值到 OPENAND_MUX 模块；

(3) 给出 OPENAND_MUX 模块的操作数选择信号 sel_a 和 sel_b；

(4) OPENAND_MUX 模块根据输入的 sel_a 和 sel_b，从输入的数据中选择两个操作数 muxed_a 和 muxed_b；

(5) 根据指令的类型输出其余模块的控制信号；

译码阶段时序输出如表 2-3 所示

表 2-3 l.add 指令在译码阶段的时序逻辑输出

信号名称	信号作用	信号值
operand_a	ALU 的源操作数 a	寄存器堆中地址 0x02 中存的数据
operand_b	ALU 的源操作数 b	寄存器堆中地址 0x03 中存的数据
ex_simm	指令需要的立即数	{{16'b0},id_insn[15:0]}
ex_branch_a ddrtarget	转移指令用到的信号，跳转的目的地址	{{4{id_insn[25]},id_insn[25:0]}+id_pc[31:2]}
rf_addrw	回写寄存器堆的地址	0x04
alu_op	数据处理类指令的操作码	5'b00000 表示加法指令
alu_op2	具体数据处理指令操作码	4'b0000 表示没有立即数加法指令
ex_branch_op	处于执行阶段指令的转移操作类别	0x0
rfwb_op	选择要写入的目的寄存器	4'b0001
ex_insn	表示处于执行阶段的指令	0xe0821800

译码阶段时序逻辑的主要工作：

(1) OPENAND_MUX 模块将选择的操作数寄存到 operand_a 和 operand_b 中，并分别送到 ALU 的输入接口 a 和 b，从而使 ALU 得到操作数 a 和 b；

(2) CTRL 模块将操作码 alu_op、alu_op2 和 comp_op 送入 ALU 模块；

(3) CTRL 模块要写的目的寄存器地址寄存到 rf_addrw 中；

(4) 产生回写的控制码 rfwb_op，输出给 WB_MUX 模块，控制选择 ALU 的输出 RF 的输入写使能为 1；

l.add 执行阶段组合逻辑输出如表 2-4 所示

执行阶段组合逻辑的主要工作是：

(1) ALU 依据操作数和操作码计算结果；

(2) WB_MUX 依据 rfwb_op、各个模块输入的数据，从中选择一个数据输出到 muxout，muxout 的数据输出到 RF 的 dataw，作为 ALU 要写入的数据；

(3) 根据计算结果，准备修改特殊寄存器 SR 的 CY、OV、F 位。

l.add 执行阶时序逻辑输出如表 2-5 所示，l.add 执行阶段时序逻辑的主要工作：

(1) 将运算结果写入目的寄存器；

(2) 修改特殊寄存器 SR 的值。

表 2-4 l.add 指令在执行阶段的组合逻辑输出

信号名称	信号作用	信号值
result	ALU 的计算结果	两个寄存器相加的结果
muxout	要写入目的寄存器的值	ALU 的运算结果 result
flagforw	表示 ALU 计算得到的 flag 值	取决于计算结果
flag_we	表示是否写特殊寄存器 SR 的 F 位	1
cyforw	表示 ALU 的计算结果是否存在进位	取决于计算结果
cy_we	表示是否写特殊寄存器 SR 的 CY 位	1
ovforw	ALU 计算过程是否发生溢出标志	取决于计算结果
ov_we	表示是否写特殊寄存器 SR 的 OV 位	1
to_sr	该变量存储 SR 的新值，在下一个时	取决于计算结果

表 2-5 l.add 指令在执行阶段的时序逻辑输出

信号名称	信号作用	信号值
sr	特殊寄存器 SR	ALU 的输出结果 result
wb_insn	需要写入目的寄存器的序号	0x4
muxreg	保存指令执行的结果	上一时钟 ALU 的输出结果
muxed_vaild	表示 muxed 信号是否有效	1

2.2 流水线加固设计实现

2.2.1 交错奇偶校验原理

流水线工作的频率较高，因此对性能的要求比较高，因此在加固过程中需要使用相对快速的方法，目前常用的检错技术包括双模冗余和奇偶校验技术，而双模冗余的方法硬件冗余面积较大，会增大整体的功耗并显著增大整体的面积，而奇偶校验技术相对其他编码方法速度较快，但是当位数较多时也会影响编码的速度。交错奇偶校验技术则结合了两者的优点，在适当增大冗余的基础上降低编译码器的延时，比较适合流水线的检错过程。这种技术的基本原理是将数据位在逻辑层次划分成多个组，划分组的个数经常与纠正能力和编码速度相关，每个组对应的数据彼此之间的距离相同，每一个组的数据由一个冗余位检测，如图 2-4 所示是 16 位的数据划分成 4 个奇偶校验组的示意图，其中 c_0 冗余位是数据 d_0、d_4、d_8 和 d_{12} 数据的异或结果，其余 c 冗余位的计算方法与 c_0 类似。交错奇偶校验的目的

是将物理相邻的存储单元在逻辑上拆分开，以适当增大冗余的方法，加快电路的编码和译码速度。冗余位的获得如公式（2-1）所示。

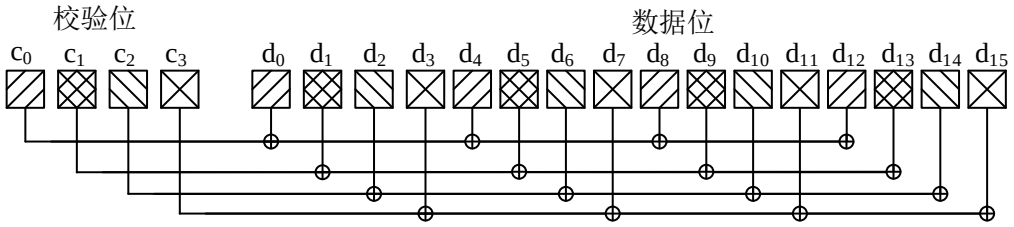


图 2-4 交错奇偶校验示意图

$$\begin{cases} c_0 = d_0 \oplus d_4 \oplus d_8 \oplus d_{12} \\ c_1 = d_1 \oplus d_5 \oplus d_9 \oplus d_{13} \\ c_2 = d_2 \oplus d_6 \oplus d_{10} \oplus d_{14} \\ c_3 = d_3 \oplus d_7 \oplus d_{11} \oplus d_{15} \end{cases} \tag{2-1}$$

2.2.2 异常处理过程

在 OR1200 处理器中，存在两种中断，一种是外部电路引起，一种是内部执行的指令引起，在该处理器中也将中断称为异常。OR1200 处理异常时一般包括存储必要寄存器的重要信息、清空流水线、针对不同的异常执行相应的异常处理程序然后重启流水线，当异常发生时，OR1200 会进行如下操作：

(1) 设置 EPCR 的值：为了解决处理器中的结构相关，处理器引入了延迟槽指令，需要判别当前处于执行阶段的是否为延迟槽指令，如果是存储前一条指令的地址，否则存储当前指令的地址，确保从未执行的指令重启流水线。

(2) 设置 EEAR 值：将正在访问的数据或者指令的有效地址 EA 保存到 EEAR 寄存器中。

(3) 设置 ESR 的值：保存当前状态寄存器 SR 的值，一般包括进位、溢出和 F 位等，在此过程中需要禁止中断，异常处理过程中再次发生中断。

(4) 将程序转移到相应的异常程序入口地址去执行异常处理程序，当异常处理程序结束时，执行 1.rfe 指令从异常处理程序中返回，恢复流水线的正常工作状态。

引入延迟槽指令是为了解决流水线中控制相关的问题，控制相关指流水线中的跳转指令或者其他需要改写 PC 的指令造成的相关。延迟槽指令一般是一条空指令，延迟槽后 PC 地址会进行跳转，如果发生异常时恰好是延迟槽指令，存储指令地址时只能存储前一条跳转指令的地址，如果存储当前延迟槽指令的地址，当执

行完异常处理程序后，重启流水线后会从延迟槽指令地址依次执行，使处理器发生混乱。

2.2.3 流水线加固结构

针对流水线中单粒子事件中的 SEU，对流水线中的寄存器采用交错奇偶校验的形式进行检错，加固的结构如图 2-5 所示。数据在进入下一级之前首先计算冗余位，冗余位的计算硬件电路如图 2-6 所示，是 8 位的信号通过树状形式进行异或，减少路径延时，在两级之间仅仅引入了三级异或门延时，对流水线的性能影响较小。在下一个时钟周期，重新计算冗余位的结果，并将结果与原冗余位结果对比，如果一致，则说明没有信号发生错误，否则说明寄存器遭受辐射发生 SEU，然后产生错误信号，流水线各级的错误信号进行或运算产生最终的错误信号，最终的错误信号输出给异常处理模块，然后通过异常处理的方式进行纠错。

异常处理的过程中，首先保存流水线中最后一级未执行完指令的地址，如果为延迟槽指令，则保存上一条指令的地址，然后清空流水线，并进行异常处理，异常处理完成后执行异常处理返回指令，该指令则会恢复发生错误时保存在 EPCR 中的地址加载到 PC 中，恢复到发生 SEU 之前的状态。

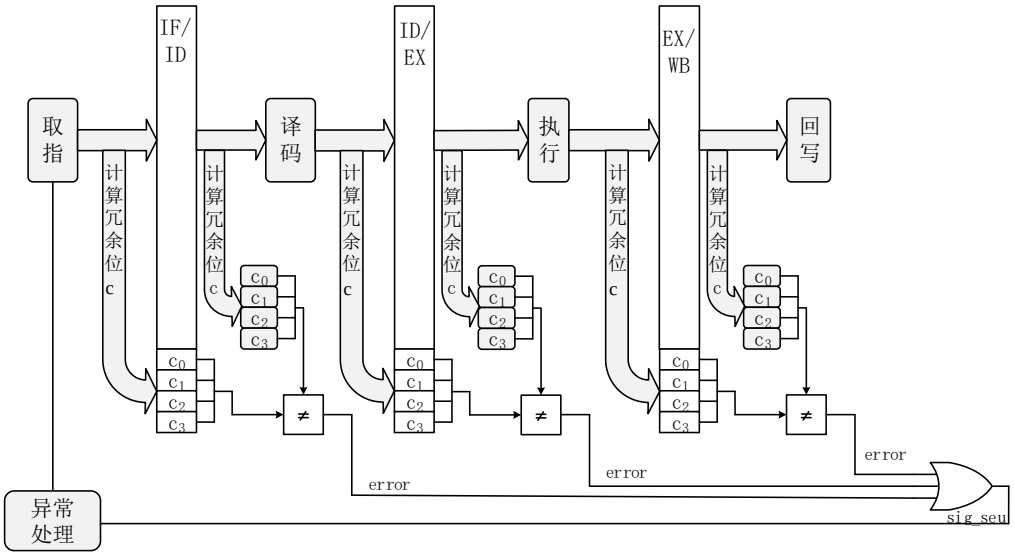


图 2-5 抗单粒子翻转流水线加固结构

然而流水线中并不是所有寄存器信号都适合使用交错奇偶校验的方式加固，数据位数小于 4 的信号和影响异常处理的信号，需要结合使用 TMR 进行加固。例如 OPERAND_MUX 模块中控制操作数 a 和 b 的保存信号 saved_a 和 saved_b，均

为 1 位的寄存器信号，需要结合使用 TMR 进行加固，以 save_d 信号为例，save_b 的加固代码如图 2-7 所示。

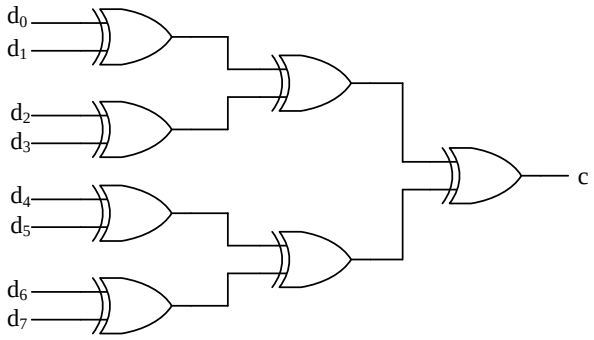


图 2-6 冗余位产生电路图

```
function tmr1;
    input       A,B,C;
    tmr1 = (A&B) | (A&C) | (B&C);
endfunction
always @(posedge clk or `OR1200_RST_EVENT rst) begin
    if (rst == `OR1200_RST_VALUE) begin
        saved_b_tmr1 <=    1'b0;
        saved_b_tmr2 <=    1'b0;
        saved_b_tmr3 <=    1'b0;
    end else if (!ex_freeze && id_freeze && !saved_b) begin
        saved_b_tmr1 <=    1'b1;
        saved_b_tmr2 <=    1'b1;
        saved_b_tmr3 <=    1'b1;
    end else if (!ex_freeze && !saved_b) begin
        saved_b_tmr1 <=    saved_b;
        saved_b_tmr2 <=    saved_b;
        saved_b_tmr3 <=    saved_b;
end
assign saved_b = tmr1(saved_b_tmr1,saved_b_tmr2,saved_b_tmr3);
```

图 2-7 TMR 加固 saved_b 寄存器信号代码

2.3 流水线加固功能验证

为了验证使用交错奇偶校验方法加固流水线的功能的正确性，以翻转 id_insn 的最高位为例，对该方法进行验证，如图 2-8 所示表示流水线发生 SEU 故障进入异常的波形图，步骤如下：

(1) 如标号①所示，l.jr r6 指令执行完成，延迟槽指令处于流水线执行阶段，该跳转指令的机器码是 0x44003000，延迟槽指令是一个空指 l.nop，机器码是 0x15000000；

(2) 如标号②所示，id_insn 的最高位注入 SEU 类型故障，即翻转最高位，数据由 0x14410000 变为 0x94410000；

(3) 如标号③所示，当数据发生改变时，重新计算的冗余位数据发生变化，由二进制 0000 变成 1000；

(4) 如标号④所示，当新冗余位 id_insn_rnew 与原冗余位 id_insn_r 数据不同时，则 id_insn_e 信号捕获错误，并传送给 or1200_ctrl 模块的 sig_ctrl_seu 信号，并输出给 or1200_except 模块；

(5) 如标号⑤所示，异常处理模块捕获异常并进入异常处理状态；

(6) 如标号⑥所示，由于是延迟槽指令，CPU 重启流水线需要从上一条跳转指令开始，因此保存上一条指令的地址 wb_pc 的值到 epcr 寄存器。

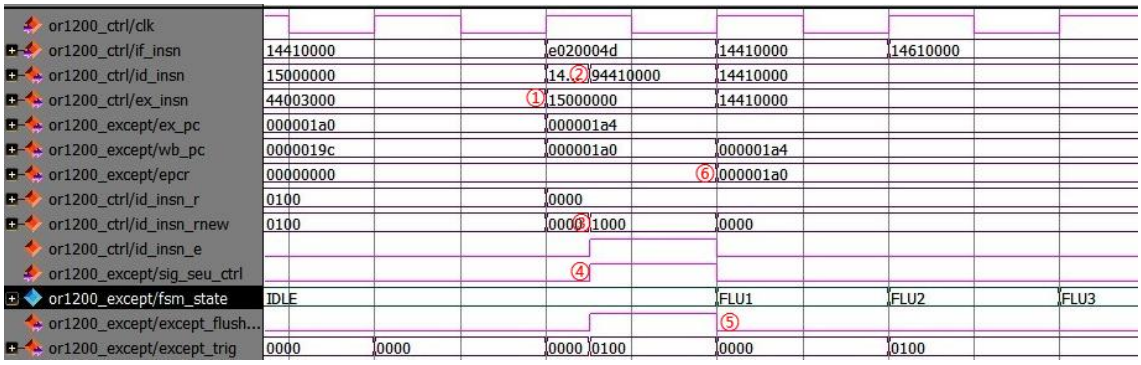


图 2-8 流水线发生 SEU 类型故障进入异常波形图

如图 2-9 所示流水线发生 SEU 故障退出异常的波形图，步骤如下：

(1) 如标号①所示，首先执行 l.rfe 指令，退出异常处理程序，l.rfe 指令的机器码是 0x24000000；

(2) 如标号②所示，退出 l.rfe 的执行阶段的第一个周期，CPU 会将 epcr 中保存的跳转指令的 PC 地址加载到 if_pc 寄存器中，并将指令地址输出给指令存储器；

(3) 如标号③所示，CPU 从指令存储器中取出发生 SEU 故障前的跳转指令 l.jr

r6 机器码是 0x44003000，并将指令加载到流水线的 if_insn,进入取指阶段，完成重启流水线的过程。

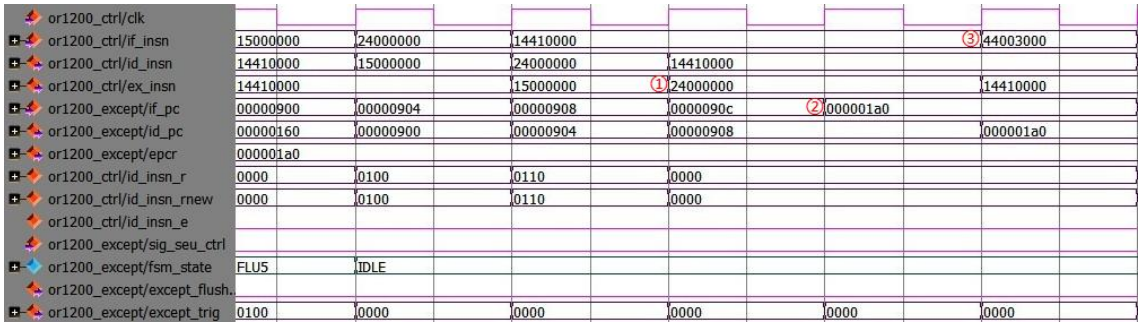


图 2-9 流水线发生 SEU 故障退出异常波形图

2.4 流水线抗辐射加固能力和性能评估

为了验证使用交错奇偶校验加固流水线的有效性，本文以冒泡排序为例，通过对比加固与未加固的节点错误率和系统错误率结果，完成对流水线加固效果的评估。设置的参数如下：

(1) 每位信号注入故障的次数 200 次，即每位信号注入的仿真次数 200 次，统计发生错误的次数；

(2) 每次故障注入运行一次完整仿真的时间 6500ns；

(3) 故障注入区域设置在 500~6000ns，故障注入区域表示，运行仿真时打翻节点的时刻所在的时间段，起始时刻保证故障不会在复位时刻使节点翻转，结束时刻保证节点翻转后，能够使错误有足够的时间传播到监视信号；

(4) 观察点的个数设置 10 个；

(5) 故障注入信号，为 CPU 模块中的所有寄存器变量；

(6) 监视信号为排序过程中使用的内存信号。

评估结果如图 2-10 所示，加固之前通过对节点错误率进行排序可知，错误率最高的信号，在注入 200 次 SEU 类型故障，其中有 179 次发生错误，节点错误率达到 89.5%，而系统错误率达到 7.31%，本文的系统错误率表示，芯片在空间辐射环境下，任意寄存器发生确定翻转后，整个系统发生错误的概率。而使用交错奇偶校验结合部分使用三模冗余的方法，仅有极个别信号发生错误，而节点错误率很低，最高仅为 1.5%，而系统错误率接近于 0。

```
time cost is 6801 seconds
Total simulate time is 6500 ns
Injected times per signal is 200
system fault rate is 7.31%
Reg Injected                    error-occured-times error-proportion
-----------------------------------------------------------------------
or1200_cpu/or1200_lsu/except_align              179         89.50%
or1200_cpu/or1200_except/extend_flush           178         89.00%
or1200_cpu/or1200_except/ex_exceptflags[2]      178         89.00%
or1200_cpu/or1200_except/ex_exceptflags[1]      178         89.00%
or1200_cpu/or1200_except/ex_exceptflags[0]      178         89.00%
or1200_cpu/or1200_ctrl/sig_trap                 178         89.00%
or1200_cpu/or1200_ctrl/sig_syscall              178         89.00%
or1200_cpu/or1200_ctrl/except_illegal           178         89.00%
or1200_cpu/or1200_except/id_exceptflags[2]      175         87.50%
or1200_cpu/or1200_except/id_exceptflags[1]      175         87.50%
or1200_cpu/or1200_except/id_exceptflags[0]      175         87.50%
or1200_cpu/or1200_ctrl/id_branch_op[1]          164         82.00%
or1200_cpu/or1200_ctrl/ex_branch_op[1]          163         81.50%
or1200_cpu/or1200_ctrl/id_insn[29]              159         79.50%
or1200_cpu/or1200_ctrl/id_insn[27]              159         79.50%
or1200_cpu/or1200_genpc/pcreg_default[9]        146         73.00%
or1200_cpu/or1200_genpc/pcreg_default[8]        146         73.00%
or1200_cpu/or1200_genpc/pcreg_default[12]       146         73.00%
```

a） 未加固节点错误率

```
time cost is 12022 seconds
Total simulate time is 6500 ns
Injected times per signal is 200
system fault rate is 0.00%
Reg Injected                    error-occured times error-proportion
-----------------------------------------------------------------------
or1200_cpu/or1200_operandmuxes/operand_a[10]   3           1.50%
or1200_cpu/or1200_ctrl/ex_simm[10]             3           1.50%
or1200_cpu/or1200_ctrl/id_insn[10]             2           1.00%
or1200_cpu/or1200_wbmux/muxreg_valid_tmr3      0           0.00%
or1200_cpu/or1200_wbmux/muxreg_valid_tmr2      0           0.00%
or1200_cpu/or1200_wbmux/muxreg_valid_tmr1      0           0.00%
or1200_cpu/or1200_wbmux/muxreg_r[3]            0           0.00%
or1200_cpu/or1200_wbmux/muxreg_r[2]            0           0.00%
or1200_cpu/or1200_wbmux/muxreg_r[1]            0           0.00%
or1200_cpu/or1200_wbmux/muxreg_r[0]            0           0.00%
or1200_cpu/or1200_wbmux/muxreg[9]              0           0.00%
or1200_cpu/or1200_wbmux/muxreg[8]              0           0.00%
or1200_cpu/or1200_wbmux/muxreg[7]              0           0.00%
or1200_cpu/or1200_wbmux/muxreg[6]              0           0.00%
or1200_cpu/or1200_wbmux/muxreg[5]              0           0.00%
or1200_cpu/or1200_wbmux/muxreg[4]              0           0.00%
or1200_cpu/or1200_wbmux/muxreg[31]             0           0.00%
or1200_cpu/or1200_wbmux/muxreg[30]             0           0.00%
```

b）加固后节点错误率

图 2-10 加固前后节点错误率对比

为了评估混合使用交错奇偶校验和三模冗余加固流水线的开销，基于 SMIC 65nm 的工艺库，本文使用 Design Compiler 综合工具得到加固前后处理器的面积和功耗对比数据，其中寄存器堆使用 memory complier 生成，生成寄存器堆的详细信息在第 3.5 小结介绍，综合时时序约束方面，除了时钟信号和复位信号外的所有输入端口，将输入延时设置为时钟周期的 40%，所有的输出信号设置输出延时与输入延时相同，为了评估使用混合交错奇偶校验和三模冗余对处理器关键路径的延时影响，本文分别综合了加固前后处理器所能达到最高时钟频率即最小时钟周期，在本文所设定的约束条件下，通过综合数据可知，未加固的处理器能够达到的最小时钟周期为 2.35ns，加固后的处理器能够达到的最小时钟周期为 2.53ns，经过分析可知，关键路径延时开销为 7.66%。为了评估流水线加固后的性能开销，综合时设定的最小时钟周期为 2.53ns，综合时采用相同的约束条件，其对比数据如表 2-6 所示，对流水线进行加固前面积为 39931.78μm^2，加固后变为 46492.06μm^2，面积开销为 16.43%，面积增加较小，功耗方面加固前为 3.8549mW，加固后变为 4.3988mW，功耗开销为 14.11%。

表 2-6 流水线加固前后处理器性能对比结果

对比项	未加固	混合交错奇偶校验和 TMR	开销
面积(μm^2)	39931.78	46492.06	16.43%
功耗(mW)	3.8549	4.3988	14.11%
最小时钟周期（ns）	2.35	2.53	7.66%

由数据可知，使用混合交错奇偶校验和三模冗余加固流水线的方法，对处理

器的关键路径延时影响较小，面积和功耗开销影响不大，适合处理器中流水线的加固。

2.5 本章小结

本章节主要对交错奇偶校验加固流水线进行详细描述，首先对流水线做了简要概述，并基于 OR1200 处理器的数据通路，简要概括了相关模块的基本功能，并以加法指令为例描述了流水线的工作过程。然后，对于流水线的加固设计进行详细描述，其中介绍了交错奇偶校验的工作原理，然后介绍了基于 OR1200 重启流水线时的异常处理过程，并给出流水线加固的抽象结构。为了验证本文设计方法的功能，本章节以翻转流水线中译码阶段的指令寄存器为例，分析整个故障检测和清空流水线并重启纠正的过程，验证功能设计的正确性。由于部分信号不适合该方法加固，因此部分信号采用经典的 TMR 加固，为了评估流水线加固的抗辐射能力，本文使用抗辐射能力评测工具，以冒泡排序为例评测对比流水线加固前后的节点错误率和系统错误率结果，完成对流水线加固效果的评估。结果显示使用混合交错奇偶校验和 TMR 加固后的流水线，节点错误率大幅度降低，仅有极个别信号有较低的节点错误率，整体的系统错误率接近于 0。另外本章节使用 DC 综合工具得到加固后的速度、面积和功耗信息，数据显示速度、面积和功耗开销较小，因此使用混合交错奇偶校验和 TMR 加固流水线的方法对处理器的性能影响较小，可以适用于大部分处理器的加固。

第3章 低冗余矩阵码设计与寄存器堆加固

3.1 寄存器堆工作原理

3.1.1 寄存器堆简介

寄存器堆用于暂存 CPU 运算过程中所需要的数据，因为处于流水线中，因此访问的速度和频率较高，所以寄存器堆常用 SRAM 来实现，而随着工艺技术的提升，单位面积上 SRAM 的存储容量变大，集成度的增高使寄存器堆中的存储单元距离减小，因此当发生单粒子效应时容易发生 MBU，因此对寄存器堆进行抗多位翻转加固具有重要的意义。

如图 3-1 所示是 OR1200 中寄存器堆的抽象模型，在 OR1200 中寄存器堆是两读一写的模型，在写数据的同时，能够同时读出数据，因此由两套读控制信号和一套写控制信号。

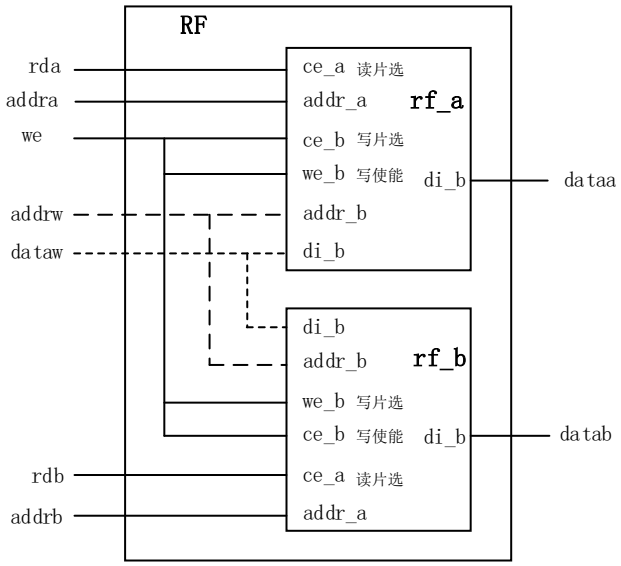


图 3-1 寄存器堆抽象模型

如表 3-1 所示，描述了寄存器堆各个信号的信号名、信号类型以及信号的具体功能。寄存器堆的读信号包括读使能信号、读地址信号、和数据输出信号，寄存器堆的写信号包括写使能信号、写片选信号、写地址信号和数据输入。

表 3-1 寄存器堆信号列表

信号名	信号类型	信号功能
rda	input	读数据 a 使能信号
addra	input	读数据 a 的地址
we	input	寄存器堆的写使能信号
addrw	input	寄存器堆的写地址
dataw	input	寄存器堆的写数据输入
rdb	input	读数据 b 使能信号
addrb	input	读数据 b 的地址
dataa	output	读数据 a 的数据输出
datab	output	读数据 b 的数据输出

3.1.2 寄存器堆的读写过程

如图 3-2 所示为寄存器堆的读过程，其详细过程如下：

(1) 在指令的取指阶段，给出读取寄存器堆的有效地址 addr 和读使能信号 rd_ce；

(2) 寄存器堆会根据使能信号选择是否保存地址，读出相应数据；

(3) 下一时钟周期，该指令的译码阶段，寄存器堆的输出端口会给出相应地址的数据 data_out，以 l.add r4,r2,r3 为例，寄存器堆同时读取 r2 和 r3 中的数据。

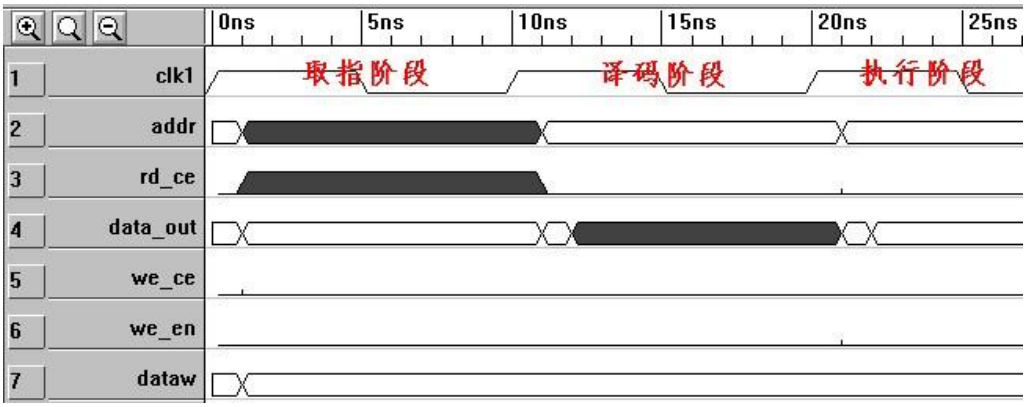


图 3-2 OR1200 寄存器读时序图

如图 3-3 所示为寄存器堆的写过程，其详细过程如下：

(1) 在指令的执行阶段，给出写寄存器堆的有效地址 addr、写片选信号 we_ce、写使能信号 we_en 和需要写入寄存器堆的数据 dataw；

(2) 寄存器堆会在时钟上升沿采取控制信号、地址和写入数据的值；

(3) 下一个时钟周期，寄存器堆会将需要写入的数据 dataw 写入相应的地址，以 l.add　r4,r2,r3 为例；则会将 r2 和 r3 所存储的数据相加写入 r4 寄存器。

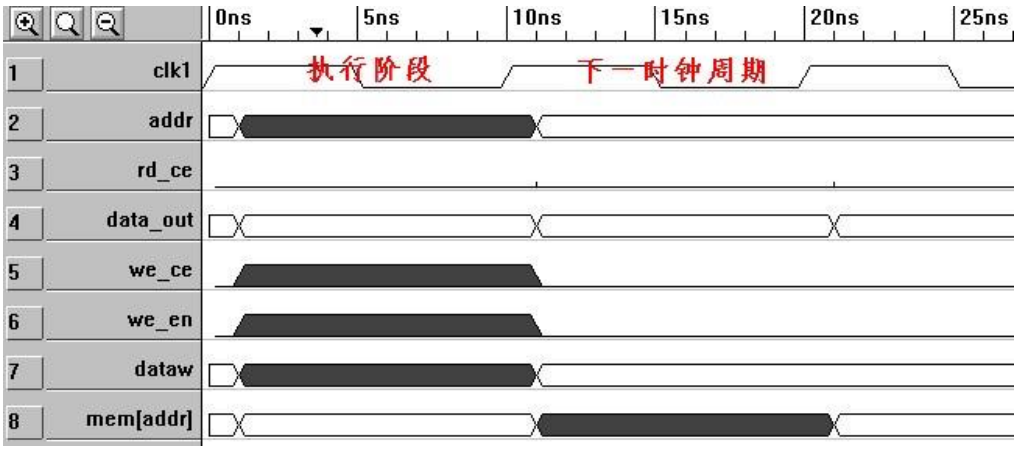


图 3-3 OR1200 写寄存器堆时序图

3.2 低冗余矩阵码设计

随着 CMOS 电路工艺尺寸的缩小，由于单个辐射粒子引起的 MBU 已经成为空间应用中存储器最具挑战的可靠性问题之一，在多数情况下，发生 MBU 的单元在位置上都是邻近的，通常使用 ECC 来保护存储器抵抗 MBU，最近几年，矩阵码由于译码简单而逐渐成为研究的热点[41]。为了抵抗内存中由辐射引起的 MBU，文献[42]提出一种针对存储体的二维修正码，其通过将数据划分成矩阵形式，然后在每行中应用低复杂度多位错误检测码检测数据的错误，然后在每列中使用奇偶校验码达到对错误数据的纠正。文献[41]提出一种基于矩阵编码的低冗余方案，其基于一种新的矩阵排列，将每一行汉明码和奇偶校验码与奇偶校验共享相结合，在具有相邻 4 位内随机错误的纠正能力前提下，降低了冗余位的个数，同时降低了面积和功耗的开销，但是该编码方案对数据位和冗余位的版图布局有要求，数据位和垂直校验位不能放在同一个存储体内，因此对应用有一定限制，而本文从新的角度构造一种低冗余矩阵码，相对于文献[41]所提出的编码，不仅降低了冗余位的位数，而且略微减小了编译码器的面积和延时开销，另外本文构造的编码对数据位和冗余位的版图布局没有要求，对应用没有限制。

构造矩阵码的第一步是划分需要保护数据的结构，为了提高纠正能力，需要将数据在逻辑上划分成二维矩阵结构的形式，然后将数据按照行或列进行依次排列，针对数据的每一行或每一列采用不同的 ECC 检测或纠正错误。根据数据的宽

度，一般为 16、32 或 64 等，因为数据位数恒定，因此设计矩阵时应当保持矩阵的行数与列数乘积不变，而矩阵的大小设定一般与纠正能力有关，另外设计矩阵的行数与列数一般是矩阵总位数的公约数。由编码理论可知，冗余位的个数与需要保护的数据位数和纠正能力有关，同时也和编译码器的复杂度有关，因此好的编码方案是按照需求在速度、面积、功耗、冗余位数和纠正能力达到折衷，总的校验位数一般随着纠正能力的增大而增多，本文在不影响编译码器延时的前提下，根据定位逻辑共享的原则最大限度降低冗余位数量。

由线性分组码的理论可知，随着保护数据位的增多和纠正能力的增大，相应的冗余位数量也会增大，而且编译码的复杂度也会显著增高，因此在硬件实现过程中所引起的问题就是硬件的面积增大、功耗增大而且编译码的延时显著上升，严重影响硬件电路的性能。本文所构造的低冗余矩阵码，通过将较长的数据块分割成较小的数据块，对各个小的数据块进行分别编译码，显著降低编译码的延时，同时提高编译码算法的纠正能力，另外通过将不同小的数据块共用定位校验位逻辑的方法，在不影响延时的前提下减少冗余位数量。

3.2.1 低冗余矩阵码编译码电路设计

以 32 位数据为例设计纠正能力为 4 的低冗余矩阵码，能够纠正相邻 4 位内随机的错误，首先将 32 位的数据划分成 8 行 4 列的矩阵形式，并且依次按照图 3-4 进行排列，在每一个逻辑行数据加一个水平校验位 c，水平校验位的数据用于检测是哪一个逻辑行的数据发生错误，水平校验位的描述如公式（3-1）所示，使用 c_0 校验位检测 d_0、d_8、d_{16} 和 d_{24} 数据位的错误，其余校验位以此类推。而对于 64 位的数据，需要划分成 8 行 8 列的矩阵形式，添加水平校验位 c，一个水平校验位需要检测 8 个数据。

水平 c 校验位只能确定错误数据所在的逻辑行，而无法定位数据的具体位置，因此需要额外增加校验位定位错误数据的具体位置，每一行有 4 个数据位和 1 个冗余位，因此需要三个校验位定位数据的错误位置，而从图 3-4 的逻辑布局可知，c_0 和 c_4 所检测的数据间隔两两之间都大于等于 4，因此可以共用定位校验位的逻辑，同理 c_1 和 c_5、c_2 和 c_6、c_3 和 c_7 也能共用校验位的逻辑，基于此可增加 12 位的 k 校验位就能定位发生错误数据的具体位置。64 位数据位的低冗余矩阵码与之类似，因为每一行的数据有 8 个数据位和 1 个冗余位，因此需要四个校验位定位错误数据的位置，基于此可以增加 16 位的 k 校验位来定位发生错误数据的位置。

d_0	d_8	d_{16}	d_{24}	c_0
d_1	d_9	d_{17}	d_{25}	c_1
d_2	d_{10}	d_{18}	d_{26}	c_2
d_3	d_{11}	d_{19}	d_{27}	c_3
d_4	d_{12}	d_{20}	d_{28}	c_4
d_5	d_{13}	d_{21}	d_{29}	c_5
d_6	d_{14}	d_{22}	d_{30}	c_6
d_7	d_{15}	d_{23}	d_{31}	c_7

图 3-4 32 位数据位低冗余矩阵码逻辑布局

$$\begin{cases} c_0 = d_0 \oplus d_8 \oplus d_{16} \oplus d_{24} \\ c_1 = d_1 \oplus d_9 \oplus d_{17} \oplus d_{25} \\ c_2 = d_2 \oplus d_{10} \oplus d_{18} \oplus d_{26} \\ c_3 = d_3 \oplus d_{11} \oplus d_{19} \oplus d_{27} \\ c_4 = d_4 \oplus d_{12} \oplus d_{20} \oplus d_{28} \\ c_5 = d_5 \oplus d_{13} \oplus d_{21} \oplus d_{29} \\ c_6 = d_6 \oplus d_{14} \oplus d_{22} \oplus d_{30} \\ c_7 = d_7 \oplus d_{15} \oplus d_{23} \oplus d_{31} \end{cases} \tag{3-1}$$

低冗余矩阵码数据译码时是以校验位的差异性来纠正错误数据，以 c_0' 表示新计算的校验位，sc_0 表示 c_0 与 c_0' 的异或，使用 $k_2k_1k_0$ 三个冗余位来定位数据的错误，以 sc_0 等于 1 发生错误为例，可以得到逻辑 1 行的 k 校正子的定位逻辑如表 3-2 所示，由 k 校正子的定位逻辑和共用定位校验位的逻辑可知，k 校验位校验逻辑如公式（3-2）所示。

表 3-2 32 位数据位的 k 校正子定位逻辑

k 校正子	c_0	d_0	d_8	d_{16}	d_{24}
sk_0	0	1	0	1	0
sk_1	0	0	1	1	0
sk_2	0	0	0	0	1

$$\begin{cases} k_0 = d_0 \oplus d_4 \oplus d_{16} \oplus d_{20} \\ k_1 = d_8 \oplus d_{12} \oplus d_{16} \oplus d_{20} \\ k_2 = d_{24} \oplus d_{28} \\ k_3 = d_1 \oplus d_5 \oplus d_{17} \oplus d_{21} \\ k_4 = d_9 \oplus d_{13} \oplus d_{17} \oplus d_{21} \\ k_5 = d_{25} \oplus d_{29} \\ k_6 = d_2 \oplus d_6 \oplus d_{18} \oplus d_{22} \\ k_7 = d_{10} \oplus d_{14} \oplus d_{18} \oplus d_{22} \\ k_8 = d_{26} \oplus d_{30} \\ k_9 = d_3 \oplus d_7 \oplus d_{19} \oplus d_{23} \\ k_{10} = d_{11} \oplus d_{15} \oplus d_{19} \oplus d_{23} \\ k_{11} = d_{27} \oplus d_{31} \end{cases} \tag{3-2}$$

由分析可知 k_2、k_5、k_8 和 k_{11} 检测的数据位数较少，在不影响路径延时的前提下，可以再次共用校验逻辑，以 k_2 和 k_5 为例，由于构造的矩阵码纠正能力为 4，因此同一个校验位检测的数据两两之间的间隔需要大于等于 4，否则可能会发生误纠，而 d_{24} 和 d_{25} 的间距为 1，不满足要求，为了扩大数据彼此间的间距，使用替换数据逻辑位置的方法，替换的规则为分别将数据与本组数据中列权重最小的数据替换，同时保证数据彼此之间的距离大于等于 4，将 d_{24} 和 d_{28} 的数据分别与 d_0 和 d_4 替换位置，d_0 和 d_4 所在的列（定位校正子为 001），d_{25} 和 d_{29} 的数据分别与 d_9 和 d_{13} 替换位置，d_9 和 d_{13} 所在的列（定位校正子为 010），不影响延时的前提下一个 k 校验位只能检测 4 位数据，因此使用一个 k 校验位去检测 d_0、d_4、d_9 和 d_{13} 数据的错误，这样同一校验位的数据彼此之间的间隔大于等于 4。

同理将 d_{26} 和 d_{30} 的数据分别与 d_2 和 d_6 替换位置，d_{27} 和 d_{31} 的数据分别与 d_{11} 和 d_{15} 替换位置，并使用一个 k 校验位去检测 d_2、d_6、d_{11} 和 d_{15} 数据的错误。因此可以减少两个冗余位，同时能够减少定位错误数据所使用 k 校验位的位数，重新构造的 k 校验位编码如公式（3-3）所示。加上水平校验位 c，对于 32 位的数据位需要添加 18 位的冗余位，为（50,32）的低冗余矩阵码。按照这个构造原则，对于 64 位数据位的低冗余矩阵码，通过本文所使用的共享校验位的原则可以减少三个 k 校验位，k 冗余位的数量变成 13 个，加上水平校验位 c 的数量 8 个，因此可以构造（85,64）的低冗余矩阵码。

$$\begin{cases} k_0 = d_{24} \oplus d_{28} \oplus d_{16} \oplus d_{20} \\ k_1 = d_8 \oplus d_{12} \oplus d_{16} \oplus d_{20} \\ k_2 = d_1 \oplus d_5 \oplus d_{17} \oplus d_{21} \\ k_3 = d_{25} \oplus d_{29} \oplus d_{17} \oplus d_{21} \\ k_4 = d_{26} \oplus d_{30} \oplus d_{18} \oplus d_{22} \\ k_5 = d_{10} \oplus d_{14} \oplus d_{18} \oplus d_{22} \\ k_6 = d_3 \oplus d_7 \oplus d_{19} \oplus d_{23} \\ k_7 = d_{27} \oplus d_{31} \oplus d_{19} \oplus d_{23} \\ k_8 = d_0 \oplus d_4 \oplus d_9 \oplus d_{13} \\ k_9 = d_2 \oplus d_6 \oplus d_{11} \oplus d_{15} \end{cases} \tag{3-3}$$

译码过程：

本文所构造的矩阵译码是根据校正子的差异性来判断错误模式继而将其纠正，详细的纠正过程如下所示。

计算水平 c 校验位的校正子 sc，具体的计算如公式（3-4）所示：

$$sc = c \oplus c' \tag{3-4}$$

式中 c ——存储的水平校验位；

c' ——读出的数据位重新计算的水平校验位；

计算定位 k 校验位的校正子 sk，具体的计算如公式（3-5）所示：

$$sk = k \oplus k' \tag{3-5}$$

式中 k ——存储的定位校验位；

k' ——读出的数据位重新计算的定位校验位

然后根据计算的 sc 来检测读出的数据是否发生错误，如果水平校正子 sc 的值全为 0，说明整体数据位没有发生错误，则直接读出的数据即为正确的数据，如果水平校正子 sc 中存在值为 1，说明相应的逻辑行有数据发生翻转，则根据定位校正子 sk 来具体定位错误的数据具体的位置，将数据进行译码得到正确的数据。如表 3-3 到表 3-10 所示为（50,32）低冗余矩阵码各逻辑行错误数据纠正表，可根据此表编写并行译码器，使译码器的速度和面积达到最优，其中表中?表示 0 或 1。

以其中的一组数据为例，可以得到相应的编译码电路，其中编码的电路图如图 3-5 所示，编码电路较为简单，仅引入了两级异或门延时，因此可知编码电路延时较小，速度较快。如图 3-6 所示为第 0 组数据所对应的译码电路，由译码电路可知，本文构造的编码方案各个路径长度平衡，合理的将延时均匀到各个路径，译码器的延时较小，由线性分组码的理论可知，译码器的延时高于编码器的延时，

表 3-3 逻辑行 0 数据纠正表

sc[0]	{sk[8],sk[1:0]}	错误数据
0	3'b???	逻辑行 0 无错误
1	3'b?01	d_{24}
	3'b?10	d_8
	3'b?11	d_{16}
	3'b100	d_0

表 3-4 逻辑行 4 数据纠正表

sc[4]	{sk[8],sk[1:0]}	错误数据
0	3'b???	逻辑行 4 无错误
1	3'b?01	d_{28}
	3'b?10	d_{12}
	3'b?11	d_{20}
	3'b100	d_4

表 3-5 逻辑行 1 数据纠正表

sc[1]	{sk[8],sk[3:2]}	错误数据
0	3'b???	逻辑行 1 无错误
1	3'b?01	d_1
	3'b?10	d_{25}
	3'b?11	d_{17}
	3'b100	d_9

表 3-6 逻辑行 5 数据纠正表

sc[5]	{sk[8],sk[3:2]}	错误数据
0	3'b???	逻辑行 5 无错误
1	3'b?01	d_5
	3'b?10	d_{29}
	3'b?11	d_{17}
	3'b100	d_{13}

表 3-7 逻辑行 2 数据纠正表

sc[2]	{sk[9],sk[5:4]}	错误数据
0	3'b???	逻辑行 2 无错误
1	3'b?01	d_{26}
	3'b?10	d_{10}
	3'b?11	d_{18}
	3'b100	d_2

表 3-8 逻辑行 6 数据纠正表

sc[6]	{sk[9],sk[5:4]}	错误数据
0	3'b???	逻辑行 6 无错误
1	3'b?01	d_{30}
	3'b?10	d_{14}
	3'b?11	d_{22}
	3'b100	d_6

表 3-9 逻辑行 3 数据纠正表

sc[3]	{sk[9],sk[7:6]}	错误数据
0	3'b???	逻辑行 3 无错误
1	3'b?01	d_3
	3'b?10	d_{27}
	3'b?11	d_{19}
	3'b100	d_{11}

表 3-10 逻辑行 7 数据纠正表

sc[7]	{sk[9],sk[7:6]}	错误数据
0	3'b???	逻辑行 7 无错误
1	3'b?01	d_7
	3'b?10	d_{31}
	3'b?11	d_{23}
	3'b100	d_{15}

而译码器的延时由校正子生成和校正子比较两部分组成，由本文所构造的编码方案可以看出，校正子生成电路仅需要三级异或门延时，而校正子比较逻辑较为简单，因此译码器的速度较快。

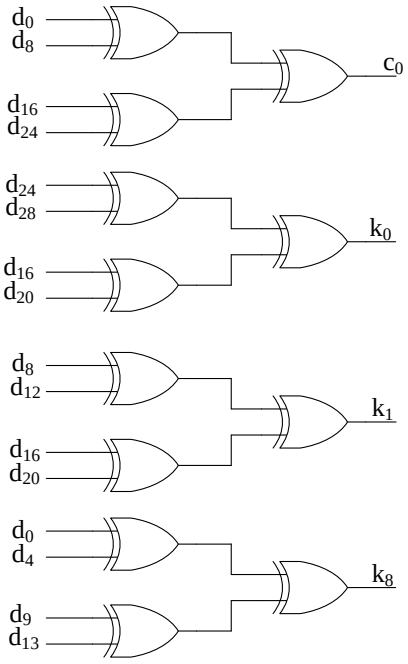


图 3-5 第一组数据编码电路

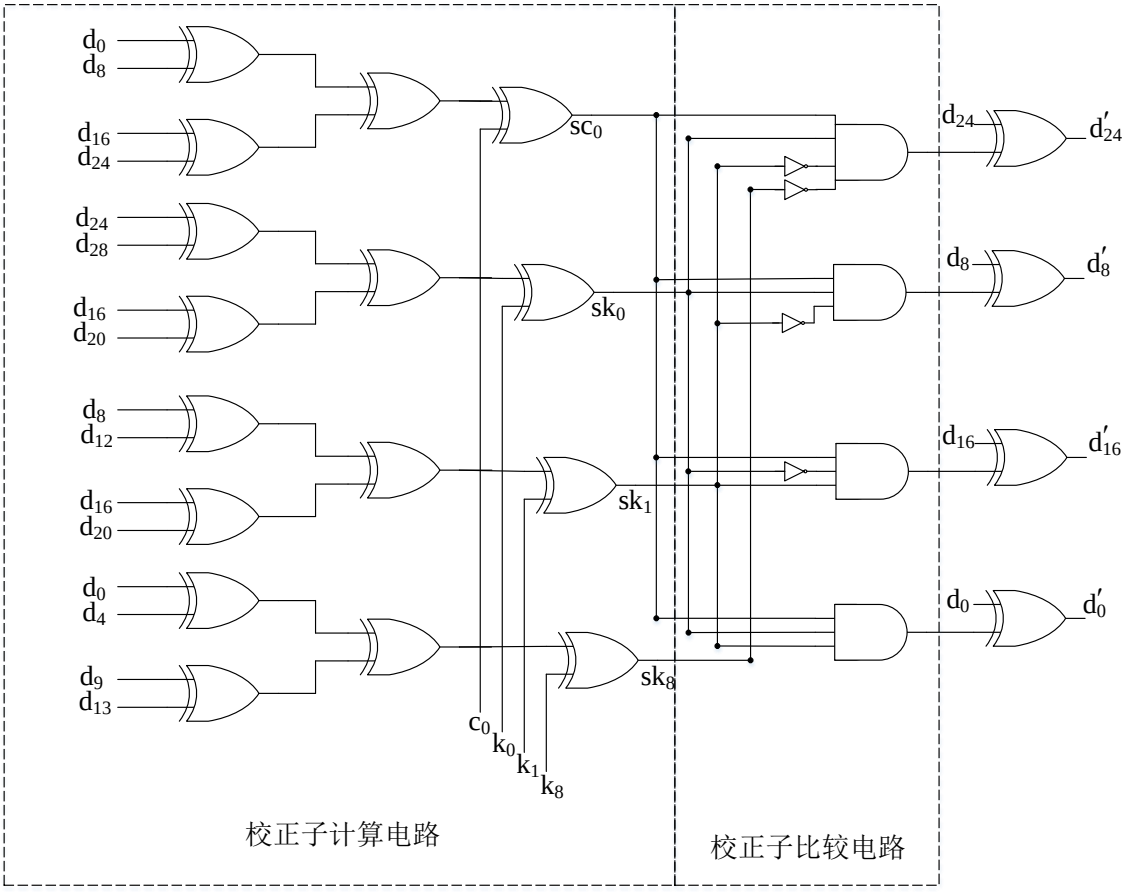


图 3-6 第一组数据译码电路

3.2.2 低冗余矩阵码纠错模式与数据布局

本文所构造的低冗余矩阵码可以纠正相邻 4 位内随机的错误，对于 32 位数据位的低冗余矩阵码，可以纠正的错误模式如图 3-7 所示，可以纠正错误模式包括任意一个单位数据的错误、相邻 4 位上的任意两位错误、相邻 4 位数据的任意三位错误和相邻的 4 位错误。

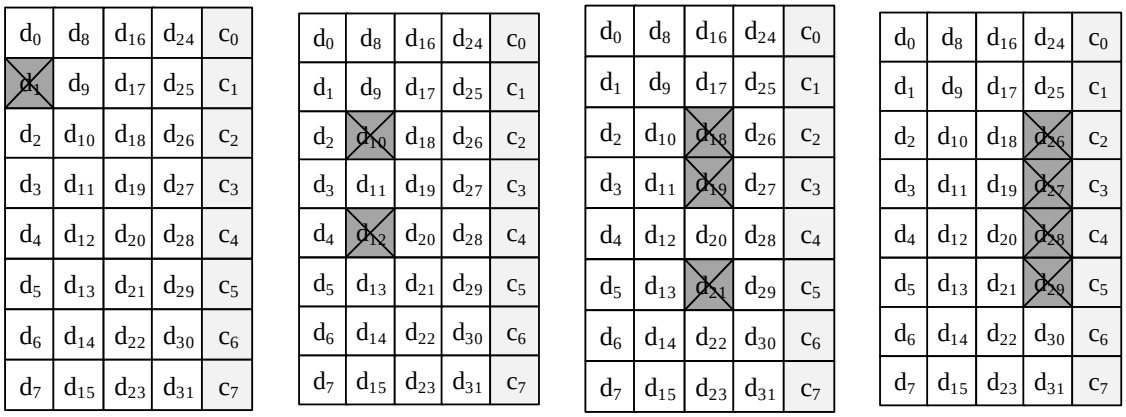


图 3-7 （50,32）低冗余矩阵码可纠正的错误模式

（50,32）低冗余矩阵码在数据排布上有一定要求，数据位和用于检测该数据的校验位不能在相邻的位置同时出错，否则错误会被掩盖或者被误纠，经过合理分析可知数据可按如图 3-8 所示存放数据，将 *c* 校验位存放到高 8 位，即第 49~42 位，数据位存储到第 41~10 位，然后将 *k* 校验位低 8 位放到第 9~2 位，*k* 校验位的高两位放到第 1~0 位，存储数据时只需将连线交织对存储器的版图布局没有要求，按照此种数据布局，可以纠正任意相邻 4 位内的随机错误。

49	.	.	.	42	41	.	.	.	10	9	.	.	.	2	1	0
校验位c[7:0]					数据位的d[31:0]					校验位k[7:0]					校验位k[9:8]	
8 bits					32 bits					8bits					2 bits	

图 3-8 （50,32）低冗余矩阵码数据布局

同样（85,64）低冗余矩阵码在数据排列上有一定的要求，其布局方式如图 3-9 所示。将 *c* 校验位存放到高 8 位，即第 84~77 位，数据位存储到第 76~13 位，将 *k* 校验位的低 12 位放到第 12~1 位，将 *k* 校验位的最高位放到第 0 位。

84	.	.	.	77	76	.	.	.	13	12	.	.	.	1	0
校验位c[7:0]					数据位d[63:0]					校验位k[11:0]					校验位k[12]
8 bits					64 bits					12bits					1 bits

图 3-9 （85,64）低冗余矩阵码数据排布图

3.2.3 低冗余矩阵码性能评估

为了评估低冗余矩阵码的整体性能，本文所构造的低冗余矩阵码分别和已经商用的经典汉明码和 SEC_DED 码，以及文献[41]提出的低冗余矩阵码 LRM_Liu 和文献[42]提出的二维修正码进行对比。本文设置相同的约束条件，使用 TSMC 65nm 的工艺库分别对几种编码算法进行综合，如表 3-11 所示，本文分别实现了几

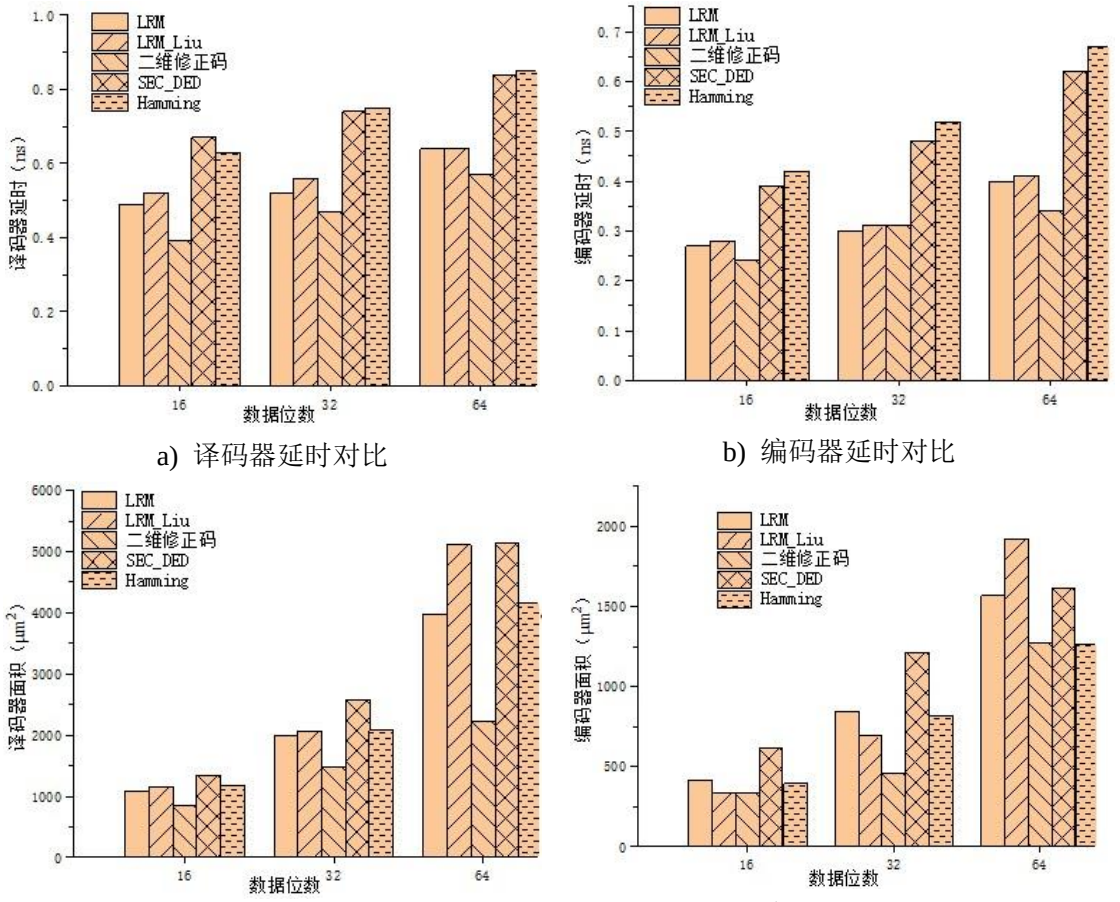
表 3-11 几种编码算法面积和延时对比数据(面积单位：μm^2，延时单位：ns)

数据位	编码名	校验位	面积优化				延时优化			
			编码器		译码器		编码器		译码器	
			面积	延时	面积	延时	面积	延时	面积	延时
16	LRM	14	121.60	0.57	247.20	1.02	414.40	0.27	1086.00	0.49
	LRM_Liu	16	129.60	0.62	289.60	1.06	339.60	0.28	1162.80	0.52
	二维修正码	16	115.20	0.36	211.20	0.58	332.40	0.24	844.00	0.39
	SEC_DED	6	178.80	0.71	322.80	1.86	612.80	0.39	1340.80	0.67
	Hamming	5	156.60	0.80	260.40	1.70	390.40	0.42	1166.00	0.63
32	LRM	18	280.79	0.57	478.00	1.21	842.89	0.30	1989.60	0.52
	LRM_Liu	20	302.40	0.63	528.00	1.17	690.80	0.31	2067.20	0.56
	二维修正码	24	259.20	0.48	446.40	0.83	460.00	0.31	1476.00	0.47
	SEC_DED	7	380.40	0.87	584.80	1.96	1213.60	0.48	2576.00	0.74
	Hamming	6	328.00	1.03	495.20	2.34	812.80	0.52	2075.60	0.75
64	LRM	21	628.80	0.80	987.60	1.65	1566.80	0.40	3976.40	0.64
	LRM_Liu	24	704.00	0.81	1116.00	1.78	1916.40	0.41	5108.40	0.64
	二维修正码	40	550.40	0.59	886.40	1.08	1267.60	0.34	2219.10	0.57
	SEC_DED	8	744.40	1.22	1163.60	3.21	1612.60	0.62	5128.80	0.84
	Hamming	7	674.80	1.31	958.80	2.72	1260.80	0.67	4163.40	0.85

种编码算法对常用的 16 位数据、32 位数据和 64 位数据的编译码电路，统计了校验位的个数，并从面积和延时优化两个方面，分别统计了各种编码算法编码器和译码器的面积和速度数据，其中面积的单位为 μm^2,延时的单位为 ns。

从冗余位的个数可知，Hamming 码和 SEC_DED 码冗余位数量最低，因为这两种码的纠正能力为 1，而 SEC_DED 码具有两位检错能力，因此冗余位略多于 Hamming 码，而二维修正码、LRM_Liu 和本文的 LRM 码的纠正能力均为 4，为相邻 4 位内的随机错误，因此冗余位数量都有增加。如表 3-11 所示针对于 16 位的数据，二维修正码和 LRM_Liu 的冗余位数量均为 16 位，而本文构造的 LRM 的冗余位数量为 14 位，针对于 32 位数据，二维修正码和 LRM_Liu 的冗余位数量分别为 24 位和 20 位，而 LRM 的冗余位数量为 18 位，针对于 64 位数据，二维修正码和 LRM_Liu 的冗余位数量分别为 40 位和 24 位，而 LRM 的冗余位数量为 21 位，其中二维修正码冗余位数量最高，而本文的 LRM 冗余位数量最低。而且随着数据位的增加，LRM 码与二维修正码和 LRM_Liu 码相比，冗余位的增长速度较慢。

如图 3-10 所示为几种编码在延时优化（即编译码器具有最小延时）的条件下，



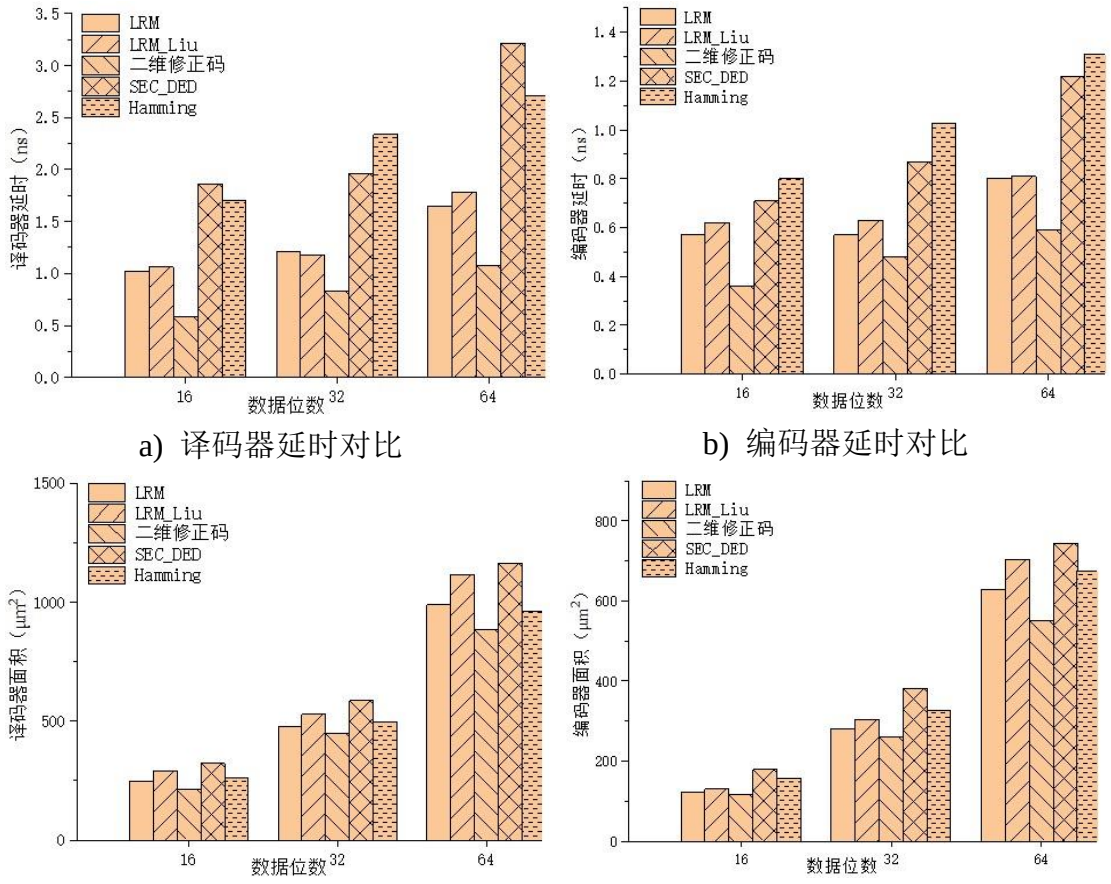
a) 译码器延时对比　　b) 编码器延时对比

c) 译码器面积对比　　d) 编码器面积对比

图 3-10 延时优化下的编译码电路延时和面积对比

编译码电路的延时和面积对比，其中子图 a）和 b）分别是编译码器的延时对比，子图 c）和 d）是编译码器的面积对比，对于延时对比来说，针对于相同数据位的编码，LRM、LRM_Liu 和二维修正码的编译码器延时均低于相同条件下 Hamming 码和 SEC_DED 码的延时，而对于 LRM、LRM_Liu 和二维修正码三种编码，整体来说 LRM 码的编译码器的延时略低于 LRM_Liu 码略高于二维修正码。对于面积对比来说，在译码器的面积上，LRM 码除了高于二维修正码的面积，均低于其余几种编码译码器的面积，而对于编码器来说，LRM 码编码器的面积处于几种编码的中等水平。

如图 3-11 所示为几种编码在面积优化（即编译码器具有最小面积）的条件下，编译码电路的延时和面积对比。与延时优化类似，针对于相同的数据位编码，LRM、LRM_Liu 和二维修正码的延时均低于 Hamming 码和 SEC_DED 码，整体来说，LRM 码的编译码器延时略低于 LRM_Liu 码，略高于二维修正码。在编译码器的面积方面，译码器中 LRM 码处于几种译码器的中等水平，编码器方面，LRM 码面积略高于二维修正码而略低于其余几种编码。



a) 译码器延时对比　　b) 编码器延时对比

c) 译码器面积对比　　d) 编码器面积对比

图 3-11 面积优化下的编译码电路延时和面积对比

综合来说，LRM 码由于提高了纠正能力，因此冗余位的数量高于 Hamming 码和 SEC_DED 码，而在相同的纠正能力前提下，LRM 码相对于 LRM_Liu 和二维修正码具有最低的冗余度。整体来说，LRM 码的延时除了略高于二维修正码，均低于其余几种编码，而编译码器的面积也处于中等或中上等水平。另外 LRM 码对版图的布局没有要求，相对于 LRM_Liu 码具有绝对的优势。因此 LRM 码适用于高纠正能力、低延时和低冗余的存储体加固。

3.3 寄存器堆加固

3.3.1 寄存器堆加固结构

处理器中的寄存器堆对读写速度的要求较高，因此在高性能的处理器中，对编译码的延时要求也较高，随着工艺的提升 MBU 在存储体中的发生概率显著上升，在加固设计过程中需要选择低延时高纠正能力的编解码算法。为了降低对处理器性能的影响，本文对寄存器堆采用（50,32）低冗余矩阵码进行加固，加固过程中只需要在数据的输入端口添加编码器，在数据的输出端口添加译码器，寄存器堆的加固抽象模型如图 3-12 所示，由于 OR1200 的寄存器堆是两读一写的结构，因此需要在输入端口添加一个编码器，在输出端口添加两个译码器。

对处理器来说编译码器的延时格外重要，编码电路相对译码电路简单，从表 3-11 可知，在 TSMC 65nm 工艺下译码器的延时仅为 0.52ns，译码器的速度较快，对处理器的性能影响较小，符合加固的需求。

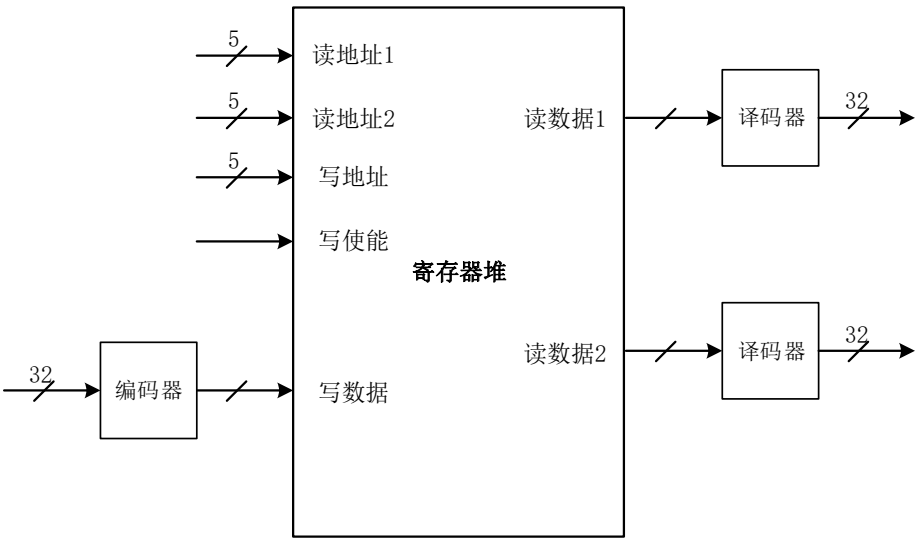


图 3-12 寄存器堆加固抽象模型

3.3.2 寄存器堆加固功能验证

为了验证寄存器堆加固功能的正确性，以翻转寄存器堆中的 r1 为例，对寄存器堆加固后的功能进行验证。如图 3-13 所示为寄存器堆的读取纠错过程，其详细纠错过程如下：

(1) 如标号①所示，机器码 0xd4011800 对应的指令是 l.sw 0x0(r1),r3，该指令表示 16 位立即数作符号扩展，与 r1 寄存器存储的值相加的结果作为有效地址，并将 r3 的数据写入存储器，因此在取值阶段需要读取寄存器 r1 的数据；

(2) 如标号②所示，取指阶段给寄存器堆读地址 addra 和使能信号 rf_ena；

(3) 如标号③所示，如上文描述（50,32）低冗余矩阵码布局上有要求，数据存放在第 41 到第 10 位，mem1_d 表示 mem[1][41:10]，将 mem[1]的数据翻转四位，mem1_d 由 0x00000008 变为 0x00003C08；

(4) 如标号④所示，水平译码校正子由 0b00000000 变成 0b00111100，译码器捕获错误定位错误数据所在的逻辑行，根据定位校正子 ski 值，对数据进行译码；

(5) 由标号⑤所示，读出的数据是 0x00000008，变成了翻转之前的数据。

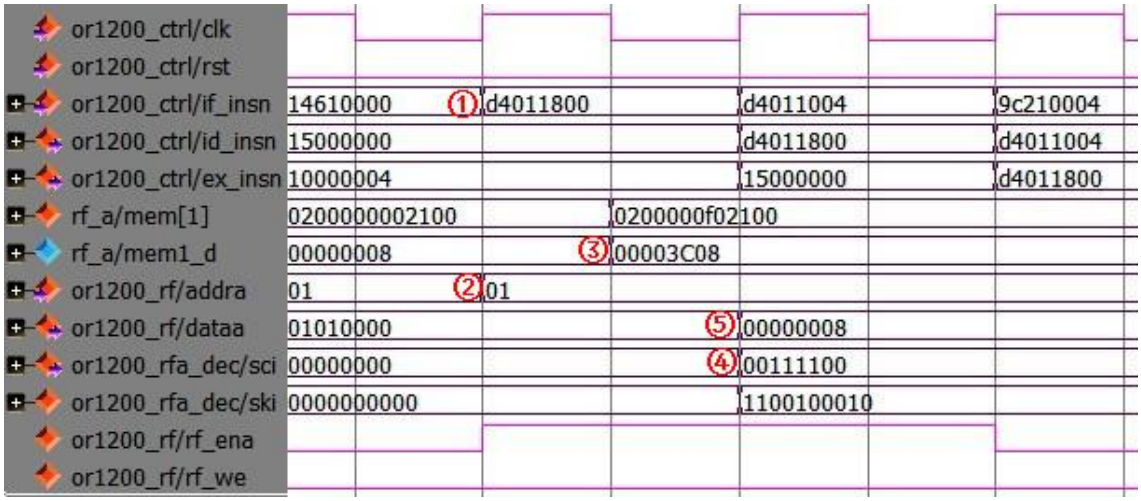


图 3-13 读取寄存器堆数据译码过程波形图

3.4 寄存器堆的纠错机制

本设计中，当寄存器堆中发生多位翻转，译码器能够确保读出正确的数据，而寄存器堆中的错误仍然存在，只能通过下次写入数据，寄存器堆中的错误才消除。为了防止累积错误，导致译码器的误纠，本文使用相应的纠错方法。

本文通过触发异常执行异常处理程序对寄存器堆进行纠错，当存储的数据发

生翻转时，译码器中的水平校正子 *sc* 不为零，使用控制电路记录发生错误时读取寄存器堆的地址，然后根据寄存器堆的地址进入不同的异常处理程序入口。如果发生错误的数据为 r8，则执行的异常处理程序如下：

```
l.addi   r8,r8,0x0
l.rfe
```

执行两条指令，第一条指令表示，读取寄存器 r8 中的数据，并将数据与 0 相加最终写回到 r8 寄存器中，因为错误的数据经过译码器可以纠正过来，因此只需要将正确的数据刷新到原寄存器就完成了数据纠正，第二条指令表示退出异常处理程序，返回到正常的程序。该方法不需要改动寄存器堆的读写控制结构，因此对处理器的性能影响较小，而且刷新仅需要 10 个时钟周期，刷新的时间相对较短，同时异常处理程序较小，因此可以将多个异常处理程序放到一个异常程序入口。

为了处理垂直方向不同字上有连续多位错误，可以根据垂直方向上连续发生多位错误的概率，决定是否需要连续刷新多行数据，有效避免处理器多次触发异常处理程序，同样以寄存器堆的 r8 为例，如果设置垂直方向上发生小于等于 2 位翻转的概率较大，则异常处理程序可以增加 r8 上下两行的数据，同时刷新 r7、r8 和 r9 的数据，异常处理程序如下所示：

```
l.addi   r7,r7,0x0
l.addi   r8,r8,0x0
l.addi   r9,r9,0x0
l.rfe
```

为了验证本小节纠错机制的正确性，下面以翻转 mem[1]中的数据为例，来分析寄存器堆的纠错过程，如图 3-14 所示为寄存器堆纠错的波形图。

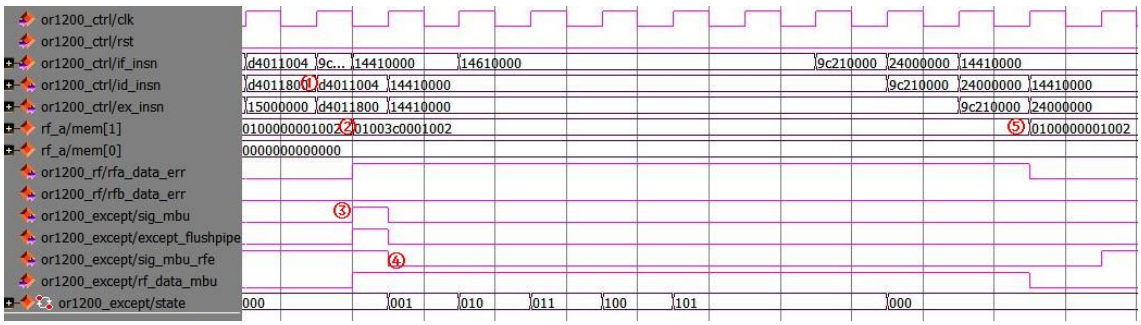


图 3-14 寄存器堆纠错波形图

(1) 如标号①所示，机器码 0xd4011004 表示的指令是 l.sw 0x4(r1),r2，该指令表示将 r1 中存储的数据与 0x4 相加作为有效地址，将 r2 中存储的数据保存到存储器，该指令的取值阶段需要读取 r1 的值；

(2) 如标号②所示，将 mem[1][33:30]数据全部翻转为 1，数据由 0x0100000001002 变成 0x01003c0001002，翻转了连续的四位数据；

(3) 如标号③所示，多位翻转信号 sig_mbu 有效，处理器进行异常；

(4) 如标号④所示，多位翻转屏蔽信号 sig_mbu_rfe 降为低电平，防止执行异常处理程序过程中再次触发异常，该信号在异常处理程序执行完成之后变为高电平；

(5) 如标号⑤所示，机器码 0x9c210000 表示 l.addi r1,r1,0x0 指令，该指令刷新寄存器 r1 的数据，指令执行完成之后，数据由翻转后的数据 0x01003c0001002 重新恢复为 0x0100000001002。

由图 3-14 所示的波形图可知，寄存器堆纠错机制功能正确，另外该方法不需要修改寄存器堆的读写控制结构，就能使用已有的指令去刷新寄存器堆的数据，且刷新一个寄存器仅需要 10 个时钟周期，对处理器的性能影响较小。为了验证寄存器堆整体的抗辐射能力，本文会在第 4.5 小节给出加固前后的评估数据。

3.5 寄存器堆加固性能评估

为了评估使用（50,32）低冗余矩阵码加固寄存器堆的性能开销，基于 SMIC 65nm 工艺库，本文使用 Design Complier 工具对设计代码进行综合，其中寄存器堆使用 memory complier 工具生成，因为加固前寄存器堆中每个寄存器的数据位宽为 32，加固后数据位宽为 50，因此分别生成两种位宽形式的 Verilog 模型和综合评估所需要的库。OR1200 处理器的寄存器堆在读取的同时需要进行写入，因此生成双端口能够同时进行读写的寄存器堆。如表 3-12 所示是通过 memory complier 生成寄存器堆的端口信号列表。

与综合流水线加固设定的综合时序约束一致，除了时钟信号和复位信号外的所有输入端口，设置输入延时为时钟周期的 40%，同样将所有的输出信号的输出延时设置为时钟周期的 40%，为了评估使用（50,32）低冗余矩阵码加固寄存器堆对处理器关键路径的延时开销，分别对加固前后的设计进行延时优化，然后分别测试加固前后的最高时钟频率（即最小时钟周期），经过综合可知，加固前后的最小时钟周期均为 2.35ns，经分析可知编译码电路的延时较小，经过编译码电路的路径不是该处理器的关键路径。

为了评估加固寄存器堆后寄存器堆和处理器整体的面积和功耗开销，将加固前后的电路综合设置相同的约束，设置时钟周期为 2.35ns。如表 3-13 所示为寄存

器堆加固后的面积和功耗对比数据，寄存器堆加固之前的面积为 9903.46μm^2，加固后变为 15623.28μm^2，面积增大了 54.12%，面积开销相对于双模冗余来说显著减小，而且纠正能力较高，加固前后，功耗由 0.311mW 变成 0.678mW，功耗增大了 118%，寄存器堆的功耗约增大一倍。

表 3-12 memory complier 生成寄存器堆端口信号列表

信号名	信号类型	信号功能
CLKA	input	读数据时钟信号
CENA	input	读数据使能信号，低电平有效
AA	input	读数据地址信号
CLKB	input	写数据时钟信号
CENB	input	写数据使能信号，低电平有效
AB	input	写数据地址信号
DB	input	数据输入信号
EMAA	input	读数据额外裕度调整信号，此信号为减慢寄存器堆访问，为寄存器堆读操作提供的额外延时，默认 3’b010
EMAB	input	写数据额外裕度调整信号，此信号为减慢寄存器堆访问，为寄存器堆写操作提供的额外延时，默认 3’b010
RST1N	input	复位信号，低电平有效
COLLDISN	output	读写冲突控制信号，当为 1’b1 时，读写地址相同时，保证写入数据正确，读写地址不同时，数据均正确，当为 1’b0 时，当读写地址相同时，无法保证写入数据正确，本文设置成 1’b1

表 3-13 寄存器堆加固前后面积和功耗对比数据

对比项	未加固	加固后	开销
面积(μm^2)	9903.46	15623.28	54.12%
功耗(mW)	0.311	0.678	118%

如表 3-14 所示寄存器堆加固前后处理器的综合数据，加固前处理器的面积为 42101.50μm^2，加固后变为 46354.68μm^2，面积增大了 10.10%，对于整个处理器来说面积开销比较小，加固前后，功耗由 4.2011mW 变成 4.8899mW，功耗增大了 16.40%，功耗开销相比面积开销略大，主要是因为编译码电路引入了部分异或门，

因此功耗的开销与面积开销略有差异，但是对整个处理器来说功耗开销没有显著增大。且该加固方法具有较高的纠正邻近错误的能力，同时对处理器的速度影响较小，因此该方法比较适合处理器中寄存器堆的加固。

表 3-14 寄存器堆加固前后处理器性能对比数据

对比项	未加固	加固后	开销
面积(μm^2)	42101.50	46354.68	10.10%
功耗(mW)	4.2011	4.8899	16.40%
最小时钟周期(ns)	2.35	2.35	0%

3.6 本章小结

本章节主要对低冗余矩阵码构造和寄存器堆加固进行详细介绍，首先是对寄存器堆的工作原理做了简要概述，讲解了 CPU 中寄存器堆的重要性以及寄存器堆的构成，另外讲述了寄存器堆的读写过程。然后详细介绍了针对于 32 位数据位的低冗余矩阵码的构造过程，以及可纠正错误的模式和数据布局，并以单组数据为例提出其编译码器的电路图，为了评估本文构造的低冗余矩阵码的性能，本文分别将该矩阵码与其他编码在校验位个数、编译码器的延时和面积进行对比，对比的编码包括商用经典的 Hamming 码和 SEC_DED 码，以及具有相同纠正能力的其余文献提出的二维修正码和低冗余矩阵码，对比结果显示，本文所构造的低冗余矩阵码适用于高纠正能力、低冗余和低延时的存储体加固。另外本章节提出抽象的寄存器堆加固模型，并进行设计和功能验证，同时提出寄存器堆的刷新机制，验证结果表明本文所提出的方法正确有效。最后本章节对寄存器堆的加固性能进行评估，基于 OR1200 平台，由于低冗余矩阵码的延时较小，通过综合数据表明，本文构造的低冗余矩阵码加固寄存器堆的方法不影响关键路径的延时，加固前后最小时钟周期均为 2.35ns，另外本章节对加固寄存器堆后处理器的面积和功耗进行评估，时钟周期为 2.35ns 下，使用该加固方法的面积开销为 10.10%，功耗开销为 16.40%，面积和功耗开销不大，因此本章节使用的加固方法适用于处理器中的寄存器堆加固。

第4章 多位故障注入设计与寄存器堆抗辐射能力评估

4.1 故障注入工作流程简介

故障注入工具是一款能够对软 IP 核（RTL 级）和固 IP 核（网表）进行抗辐射能力评估的工具，采用 perl 语言开发，结合 Gtk2-perl 开发包开发的工具。该工具采用模拟故障注入技术，结合现有的组合电路和时序电路的瞬态位翻转模型，底层调用仿真工具对设计文件进行评估，最后通过数据处理得到 IP 核内部各个节点的错误率，完成对设计的评估，以仅执行一次故障注入为例，故障注入的示意图如图 4-1 所示，详细工作过程如下：

(1) 首先运行仿真到 T_f 时刻，保存故障注入信号的正确值，为故障注入翻转信号做准备；

(2) 添加监视信号，此监视信号为顶层设计的输出信号或者用户关心的信号，运行仿真到结束时刻，保存监视信号在未注入故障的前提下，正确的输出结果报告，作为黄金结果；

(3) 再次启动仿真到 T_f 时刻，改变故障注入信号的值，模拟空间环境中寄存器发生的 SEU 或者组合电路发生的 SET；

(4) 添加与未注入故障时相同的监视信号，运行仿真到结束，保存故障注入后监视信号的变化报告；

(5) 数据分析，对比故障注入前后得到的两个报告，分析故障注入是否引起监视信号错误。

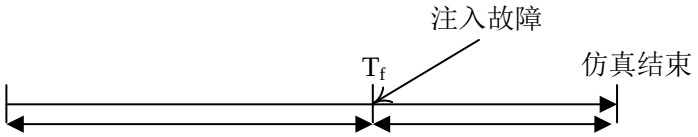


图 4-1 故障注入示意图

4.2 多位故障注入模型

辐射实验表明，在 45nm 工艺下，多位翻转的个数主要是 2-4 位，如图 4-2 所示 45nm SRAM 多位翻转错误图样，从图片中可以看出，发生错误的图样主要是水平方向、垂直方向和对角线方向。而 SRAM 中一般按字节读取数据，因此分析时只需看水平方向的错误类型，如果使用 X 表示数据位发生翻转，O 表示数据未发

生翻转，则错误类型一共可以分解为 OOX、OXO、XOO、OXX、XOX、XXO、XXX 七种。

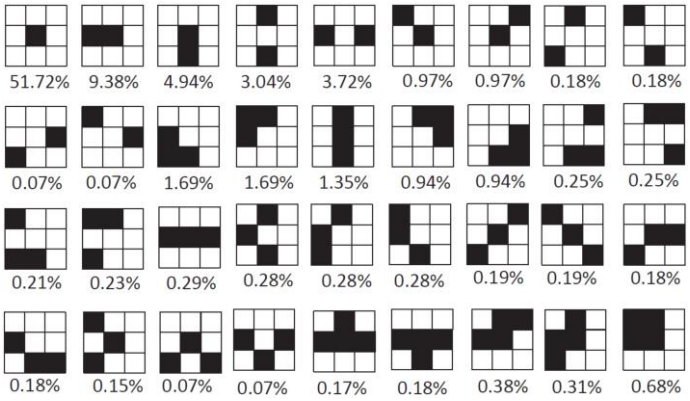


图 4-2 45nm SRAM 多位翻转错误图样

因此对于多位故障注入，可以创建两种多位翻转模型，如图 4-3 为多位翻转模型的示意图，假设存储体规模大小 6×6，抗多位翻转的编码的纠正能力为 3，在翻转数据的过程中，可以根据纠正能力设置窗口的长度，如图 4-3 所示，使翻转窗口依次按照图示顺序遍历整个存储体，可以有两种翻转模型，第一种可以直接翻转窗口内的所有位，检测添加到系统内的编译码电路以及纠错机制最终是否影响整个系统，此模型可以模拟连续翻转的多位故障，用于评估相邻连续翻转的编码方案的抗辐射能力，第二种可以设置参数，随机翻转窗口内的任意位，覆盖上述所介绍的所有错误类型，此模型可以模拟相邻多位内随机翻转故障。

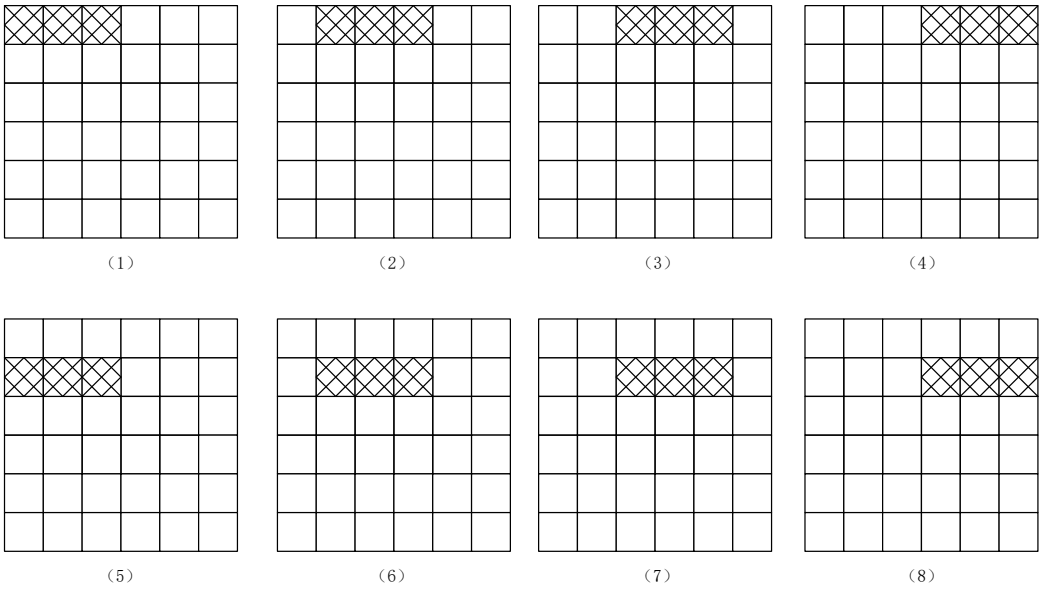


图 4-3 多位故障注入模型示意图

4.3 多位故障注入设计实现

多位故障注入是在现有的单位故障注入工具的基础上而设计的，多位故障注入的核心思想是如何将数据按照所需要的方式去翻转数据，本设计是根据辐射实验图样所抽象的模型设计相邻多位故障注入。多位故障注入与单位故障注入的主要区别是在翻转数据的个数和翻转数据的位置。

单位故障注入设计时，需要设置的故障注入参数包括每位信号故障注入的次数$times_per_sig、运行一次完整仿真总时间$simulation_end、观察点的个数 N、故障注入开始时间$inject_start 和故障注入结束时间$inject_end、故障注入类型等，部分参数所表示的含义如图 4-4 所示。

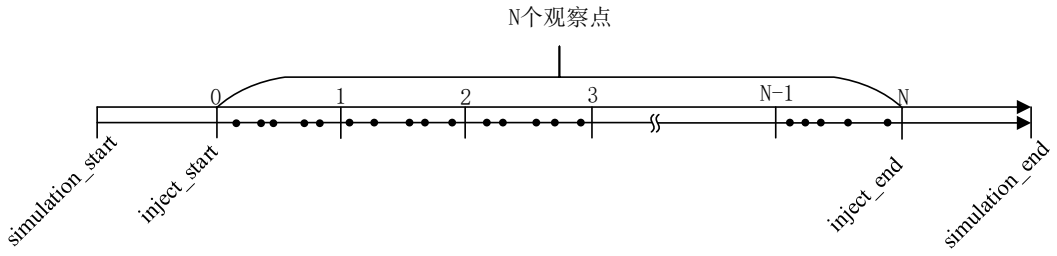


图 4-4 多观察点故障注入示意图

单位故障注入时，故障注入工具首先会根据观察点的个数 N 将故障注入区域$inject_start-$inject_end 平均分为 N 段，然后保存 N+1 个端点的仿真状态，然后进行故障注入的执行，执行故障注入之前首先需要生成未注入故障时的黄金模型，以每位信号注入 100 次、设置 20 个观察点为例，如图 4-4 所示则每个故障注入小区间会有 5 个故障注入点，这些故障注入点的时刻随机生成，然后需要保存未注入故障的黄金模型，故障注入工具会遍历各个故障注入点，为了加快仿真速度，遍历故障注入点时会从最近的仿真状态启动仿真，然后运行仿真到故障注入点，此时会保存故障注入信号在故障注入点的正确值，然后添加监视信号，再运行仿真到结束时刻，保存监视信号的信号变化报告[43]。然后将保存的各个故障注入时刻的故障注入信号的正确值全部翻转并保存到数组中，随后在进行故障注入时会依次遍历每个信号 100 次，在对应的故障注入时刻，把故障注入信号的值改变为翻转后的值，并添加监视信号，保存注入故障后监视信号的信号变化报告。最后进行数据处理，主要是将故障注入后的结果与黄金结果对比，统计信号的节点错误率和系统错误率。

对于多位故障注入来说，和单位故障注入的主要区别是翻转数据的个数和翻

转信号的位置，因此相对于单位故障注入，只需要根据相应的参数，确定遍历每个信号时需要翻转信号的个数和具体位置，因此可以生成一个多维数组，存储翻转数据和翻转位置的信息。对于多位故障注入需要解决的一个问题是如何根据信号名称判定信号是否相邻，本文则使用 Perl 的正则表达式来实现，对复杂的文本进行智能识别。

进行多位故障注入，首先需要在故障注入类型框架中添加 MBU 选项的按钮，设置使用 RadioButton，点击该按钮设置对应的 MBU 参数，如图 4-5 所示为 MBU 设置窗口。由多位故障注入模型可知，模型实现两种翻转方案，一种是直接翻转窗口内的所有位，一种是翻转窗口内的任意位，MBU 设置窗口中需要设置两个参数，第一个参数为窗口宽度设置，因为是多位故障注入，所以窗口宽度的最小值为 2，然后根据编码的纠正能力设置对应的数值，然后是设置翻转的方式，其中 Continuous 按钮表示翻转窗口内的所有位，Random 按钮表示随机翻转窗口内的任意位。

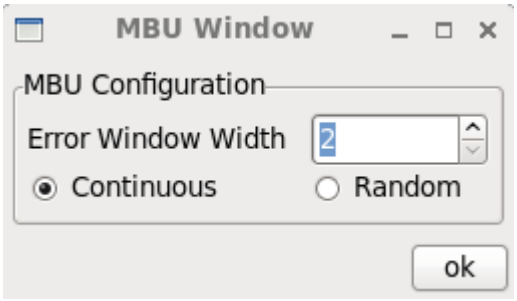


图 4-5 MBU 参数设置窗口

部分代码如图 4-6 所示，生成故障注入位置时需要遍历各个信号，假设设置的窗口宽度为 4，因此遍历信号时需要判断其相邻的 3 个信号，需要判断其后的几位信号是否和当前信号相邻。寄存器堆的大小 32×32，在设计代码中是一个二维数组模型，以 r0 寄存器为例其对应的数组是 mem[0]，mem[0]有 32 位的数据，经过数据分析后会差分成 32 个单位的信号，其信号名的格式为 mem[M][N]，M 表示对应的字，N 表示字下对应的位。以 mem[0][31]为例，其相邻的数据是 mem[0][30]、mem[0][29]和 mem[0][28]，判断相邻的条件满足两个，第一点是数据必须在相同的字上，上述的四个数据均在 mem[0]字，而 mem[0][0]和 mem[1][31]虽然在顺序上相邻但是在不同的字上应能判定出数据不相邻，第二点是字下对应的位，由于在信号提取时数据是从高位向低位排序，因此第 31 位相邻的其余三位是 30、29 和 28。而需要保存的是需要翻转数据的位置，因此对于 mem[0][31]数据对应的翻转位置，如果为连续翻转则需要保存的数据为 1111，因此在进行多位连续故障注入

```
1 for(my $k=0;$k<=$#reg_injected_basename;$k++){
2       if($inject_type eq 'mbu'){
3             chomp($reg_injected_basename[$k]);
4             my $start_signal = $reg_injected_basename[$k];
5             $start_signal =~ /(\w+.*)\[(\d+)\]$/;
6             my $sta_sig_name = $1;
7             my $sta_sig_num = $2;
8             if($mbu_inject_type){
9                   $ran_num_dec = $err_width_ran_max;
10            }else{
11                  $ran_num_dec = (int(rand($err_width_ran_max))+1);
12            }
13            $ran_num_bin = sprintf("%b",$ran_num_dec);
14            my $add0_len = $error_window_width-length($ran_num_bin);
15            $ran_num_bin = ('0' x $add0_len).$ran_num_bin;
17            my @ran_bin_char = split //,$ran_num_bin;
18
19            foreach (0..($error_window_width-1)){
20                  last if($k+$_==@reg_injected_basename);
21                  my $current_signal = $reg_injected_basename[$k+$_];
22                  $current_signal =~ /(\w+.*)\[(\d+)\]$/;
23                  if (($sta_sig_name eq $1) || (($sta_sig_num-$_)==$2)) {
24                  push @{$mbu_injected_position[$k][$i][$j]},$ran_bin_char[$_];
25                  }else{
26                        last;
27                  }
28            }
29      }
30}
```

图 4-6 多位故障注入部分代码

到 mem[0][31]时，就可以遍历已经保存的代表位置的四位数据，如果为 1 则翻转窗口内的数据，否则保持数据不变。在此基础上也可以实现相邻多位随机翻转，其实现方法与连续多位翻转不同，本文的实现方法代码如图 4-6 的第 8 到 17 行所示，根据设置的窗口宽度，假设为 4，得到其对应的 2 的指数次幂减一即为 15，然后生成 1~15 之间的随机数，将随机数转换为二进制数（4 位），如果对应位为 1 则表示对应的该数据需要翻转，如果对应的数据为 0 则表示对应的该位数据不需要翻转，以 X 表示翻转 O 表示不翻转，如表 4-1 所示该方法可以遍历所有的相邻四位内随机错误。

表 4-1 相邻四位随机错误

随机数	二进制数	错误类型
1	0001	OOOX
2	0010	OOXO
3	0011	OOXX
4	0100	OXOO
5	0101	OXOX
6	0110	OXXO
7	0111	OXXX
8	1000	XOOO
9	1001	XOOX
10	1010	XOXO
11	1011	XOXX
12	1100	XXOO
13	1101	XXOX
14	1110	XXXO
15	1111	XXXX

4.4 多位故障注入设计功能验证

由多位故障注入的设计方法可知，该设计能够用于评估相邻 N 位随机错误和连续错误的编译码电路，可以方便的验证设计电路功能是否正确，另外该方法能够评估使用 ECC 加固存储体的抗辐射能力，一方面能够方便的验证整体功能的正确性，指导设计人员进行相关设计和查找漏洞，另外给出一个评估抗辐射能力的定量指标，对比评估加固效果。

4.4.1 多位相邻连续故障注入功能验证

为了评估多位相邻连续故障注入功能的正确性，本文以构造的（50,32）低冗余矩阵码为例，通过理论分析和仿真数据结果对比验证多位故障注入的正确性。由于（50,32）低冗余矩阵码具有相邻 4 位的随机纠正能力，而对相邻 5 位数据的错误无法纠正，因此本文分别以翻转连续 4 位和 5 位数据为例对设计进行仿真。测试文件生成 100 个 32 位的随机数然后依次输入到编码器，通过翻转编码后的数据，验证输出结果是否正确，时钟周期为 4ns。设置的故障注入参数如下：

(1) 每个错误窗口的故障注入次数为 1000 次。
(2) 运行一次完整仿真的时间为 500ns。
(3) 故障注入的区域是 50~400ns，表示运行仿真时打翻存储节点所在的时间段。
(4) 观察点的个数设置为 20 个。
(5) 故障注入信号为编码后的所有存储信号 mem[49:0]。
(6) 监视信号为译码后的所有输出信号 cdk_out[49:0]。

经过测试可知，相邻 4 位连续翻转的节点错误率均为 0%，设计的低冗余矩阵码具有相邻 4 位内随机错误纠正能力，相邻 4 位内连续错误均能纠正，因此理论值也为 0%，说明相邻 4 位内连续故障注入正确。而相邻 5 位内连续翻转，经测试数据发现，只要测试数据中包含原数据位，节点错误率均为 100%，而如果相邻 5 位只含有冗余位，节点错误率均为 0%，因为冗余位的纠错与数据位略有不同，经理论分析，该编码的纠正能力为 4，因此针对于连续 5 位翻转，如果内部含有原数据位，编码无法进行纠正，因此错误率为 100%，因此相邻 5 位内连续翻转故障注入结果正确。经理论和测试数据对比可知，多位相邻连续故障注入功能正确。

4.4.2 多位相邻随机故障注入功能验证

评估多位相邻随机故障注入的功能，本文使用构造的（50,32）低冗余矩阵码，矩阵码的纠正能力为 4，因此通过对比相邻 4 位和 5 位随机错误的理论值和仿真数据，判定多位相邻随机故障注入的正确性，使用的测试程序和故障注入参数与多位相邻连续故障注入相同。

经过测试数据可知，相邻 4 位内随机翻转的节点错误率为 0%，（50,32）低冗余矩阵码的纠正能力为 4，因此相邻 4 位内随机翻转的节点错误率为 0%，说明相邻 4 位内随机翻转的故障注入功能正确。而相邻 5 位内随机翻转的故障注入数据如图 4-6 所示，其中相邻 5 位内随机翻转错误率最高为 28.7%，最低为 25%，整体

的错误率为 26%左右，由理论分析可知，当错误窗口为 5 时，则相邻随机多位翻转生成的随机数会在 1~31 之间，而该矩阵码的纠正能力为 4，则错误的数据间隔大于 4 时，不能纠正，因此不能纠正的数据二进制类型为 1???1，其中 1 表示翻转，0 表示不翻转，?表示 0 或 1，因此符合这种类型的数据有 8 个，因此理论错误率为 8 除以 31 等于 25.8%，因此理论值与测试数据相符，相邻 5 位内随机故障注入功能正确。经过理论和测试数据对比可知，多位相邻随机故障注入功能正确。

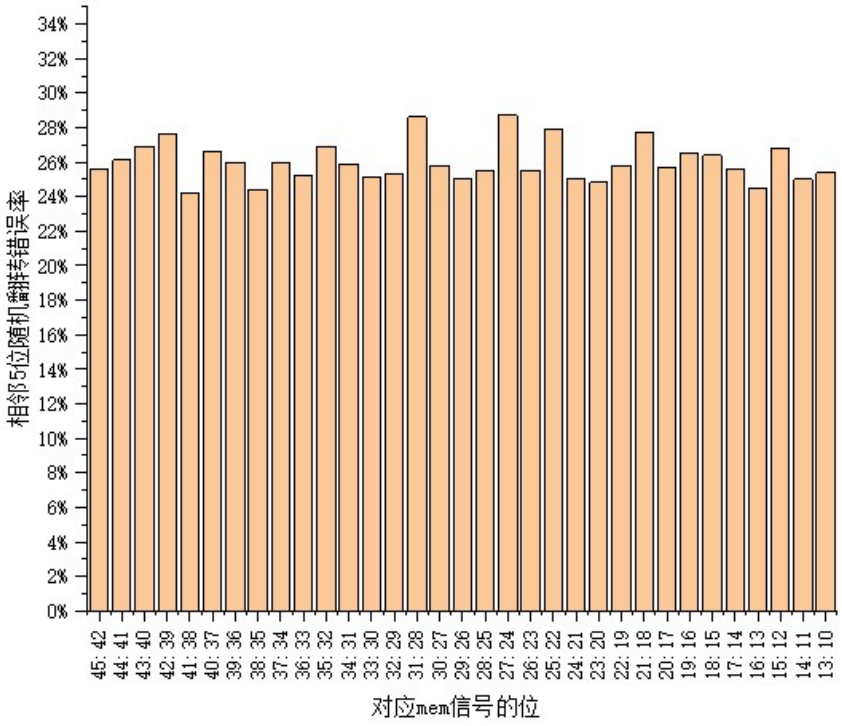
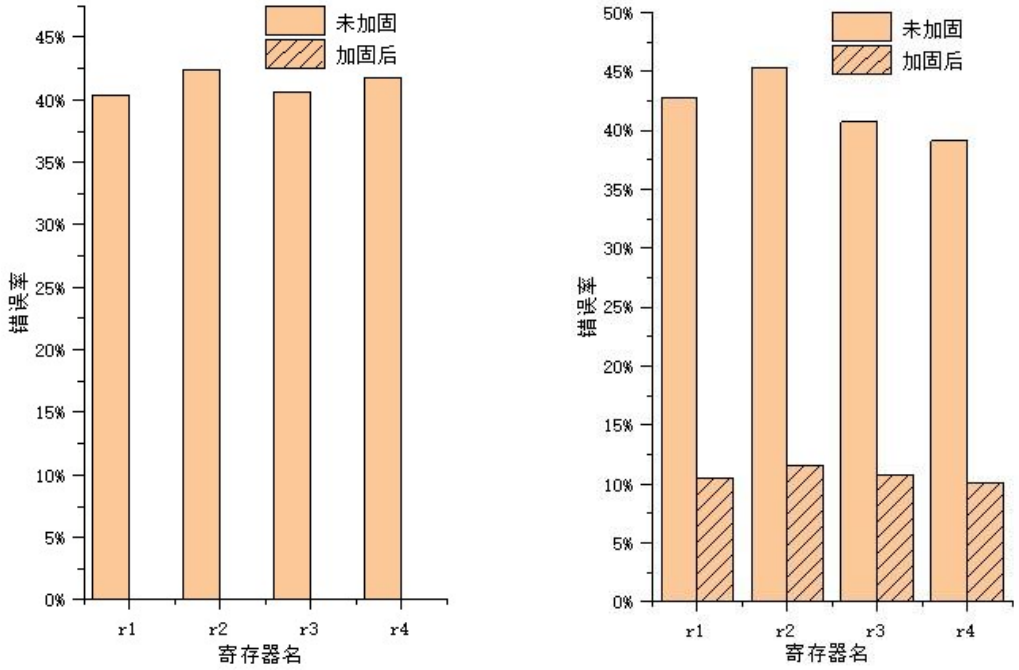


图 4-7 低冗余矩阵码相邻 5 位随机翻转错误率

4.5 寄存器堆抗辐射加固能力评估

本小节评估加固后寄存器堆的抗辐射能力，其中寄存器堆采用低冗余矩阵码加固，纠错过程使用刷新寄存器堆的方式，验证程序使用斐波那契数列，用到了内部的寄存器包括 r1、r2、r3 和 r4，因为寄存器堆的加固形式统一，因此以这四个寄存器组为例，对比寄存器堆加固前后的抗辐射能力。由本章的第 4 小结可知多位相邻故障注入正确，本小节则只测试窗口宽度为 4 和 5 的相邻随机多位错误，其对比数据如图 4-8 所示，图 a）为相邻 4 位内随机翻转错误率，右侧为相邻 5 位内随机翻转错误率。对于设置错误窗口宽度为 4 的相邻随机多位翻转，采用低冗余矩阵码和寄存器堆刷新加固之前的错误率在 40%以上，而加固之后错误率降为 0%，说明加固后的寄存器堆能够对相邻 4 位内随机错误进行有效抵抗。对于设置

错误窗口宽度为 5 的相邻随机多位翻转，加固之前的错误率为 40%左右，加固之后的错误率在 10%左右，因为本文设计的（50,32）低冗余矩阵码的纠正能力为 4，因此只能抵抗大部分错误，而无法消除所有的相邻 5 位内随机错误，然而错误率显著降低，综上所述，本文对于寄存器堆使用的加固方法能够有效抵抗错误数据间距小于等于 4 的多位翻转。



a) 相邻 4 位内随机错误　　b) 相邻 4 位内随机错误

图 4-8 寄存器堆加固前后多位翻转错误率

4.6 本章小结

本章首先介绍了故障注入工作的简要流程，然后通过 45nm SRAM 的多位翻转图样，提出抽象的多位翻转模型，经分析可知，多位翻转模型可以分为相邻连续多位翻转和相邻随机多位翻转模型。本章分析对比了多位故障注入和单位故障注入的主要区别，并在此基础上详细介绍了多位故障注入的设计实现过程，另外本章通过本文构造的低冗余矩阵码验证了多位故障注入设计功能的正确性，验证结果显示本文设计的方法测试数据与理论相符。最后本章对比多位随机故障注入前提下寄存器堆加固前后的数据，评估了寄存器堆对 MBU 的抗辐射能力。

结　论

随着航天技术和集成电路技术的发展，处理器作为航天器的控制核心，极易受到空间高能粒子的冲击辐射，因此对处理器的抗辐射加固具有重要的意义。本文对处理器中极易受到单粒子翻转影响的流水线和极易受到 MBU 影响的寄存器堆进行抗辐射加固技术研究，为了评估加固后寄存器堆的抗辐射能力，本文对多位故障注入技术进行研究，主要的研究成果如下：

(1) 完成了交错奇偶校验结合流水线重启加固流水线的设计，对于部分信号采用 TMR 加固技术，针对于处理器中的流水线，通过交错奇偶校验进行检测，并将错误信号连接到异常处理模块，当检测到错误时通过清空流水线，并进行流水线重启，恢复到单粒子翻转之前的状态。通过故障注入工具表明，经过本文所使用的混合交错奇偶校验和 TMR 技术加固的流水线，系统错误率接近于 0%。通过综合数据表明，该方法对处理器的速度、面积和功耗影响较小。

(2) 构造了一种新颖的能够纠正 4 位相邻随机错误的低冗余矩阵码，经过数据对比，该低冗余矩阵码相对于经典的 Hamming 码和 SEC_DED 码具有较低的延时，相同的纠正能力前提下，该码具有较低的冗余位数量，同时编解码器的面积较小。

(3) 使用低冗余矩阵码完成了寄存器堆的加固设计，本文通过低冗余矩阵码与处理器的异常处理，通过寄存器堆刷新的方式纠正发生在寄存器堆中的 MBU，基于 OR1200 处理器，综合数据表明，该方法不影响处理器的关键路径延时，同时使用该方法加固寄存器堆，对处理器的面积和功耗影响较小。

(4) 提出了相邻多位故障注入翻转模型，实现了相邻连续故障注入和相邻随机故障注入设计，可以方便的验证 ECC 编译码电路功能是否正确，另外该方法能够评估使用 ECC 加固存储体的抗辐射能力，方便设计人员查找漏洞。

对未来工作的展望：

(1) 可以结合电路加固方法，从寄存器单元进行设计，对处理器组合电路中的 SET 进行加固，进一步提高处理器的抗辐射能力。

(2) 补充多位翻转模型，可以实现随机位置固定位数的多位故障注入技术，可以用于评估多位随机错误类型的数据错误。

参考文献

[1] Pham D C, Aipperspach T, Boerstler D, et al. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor[J]. IEEE Journal of Solid-State Circuits, 2005, 41(1): 179-196.

[2] Mattson T G, Wijngaart R V D, Frumkin M. Programming the Intel 80-core network-on-a-chip terascale processor[C]. High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for. IEEE, 2008: 1-11.

[3] Henkel J, Bauer L, Dutt N, et al. Reliable on-chip systems in the nano-era: Lessons learnt and future trends[C]. ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, Tx, 2013: 1-10.

[4] González-Toral R, Reviriego P, Maestro J A, et al. A Fast Technique to Reduce Power Consumption on Linear Block Codes Used to Protect Registers[J]. IEEE Transactions on Device & Materials Reliability, 2018, 99(1): 189-196.

[5] Liu S, Reviriego P, Maestro J A. Efficient Majority Logic Fault Detection With Difference-Set Codes for Memory Applications[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2012, 20(1): 148-156.

[6] Bouajila A, Zeppenfeld J, Stechele W, et al. An architecture and an FPGA prototype of a reliable processor pipeline towards multiple soft- and timing errors[C]. IEEE, International Symposium on Design and Diagnostics of Electronic Circuits & Systems. IEEE, 2011: 225-230.

[7] Borkar S. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation[J]. IEEE Micro, 2005, 25(6):10-16.

[8] Johnston A H. Space Radiation Effects and Reliability Considerations for Micro- and Optoelectronic Devices[J]. IEEE Transactions on Device and Materials Reliability, 2010, 10(4): 449-459.

[9] Hubert G, Bourdarie S, Artola L, et al. Impact of the Solar Flares on the SER Dynamics on Micro and Nanometric Technologies[J]. IEEE Transactions on Nuclear Science, 2010, 57(6): 3127-3134.

[10] 阳建伟. 基于 LEON3 处理器外部存储器控制器加固设计[D]. 哈尔滨: 哈尔滨工业大学, 2012: 39-50.

[11] 刘瑞. 宇航处理器 Cache 系统的可靠性分析和加固研究[D]. 上海：上海交通大学, 2011: 9-13.

[12] Yan A, Liang H, Huang Z, et al. An SEU resilient, SET filterable and cost effective latch in presence of PVT variations[J]. Microelectronics Reliability, 2016, 63: 239-250.

[13] Huang Z , Liang H , Yan A , et al. High-performance, low-cost, and highly reliable radiation hardened latch design[J]. Electronics Letters, 2016, 52(2): 139-141.

[14] Bolchini C, Miele A, Santambrogio M D. TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs[C]. IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems. IEEE Computer Society, 2007: 87-95.

[15] 彭小燕, 彭飞, 刘凯俊, 等. 一种基于多处理器的星载计算机抗辐射加固设计方案[J]. 航天标准化, 2016(2): 1-7.

[16] Kim J, Hardavellas N, Mai K, et al. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding[C]. Ieee/acm International Symposium on Microarchitecture. IEEE, 2007: 197-209.

[17] Seifert N, Gill B, Zia V, et al. On the Scalability of Redundancy based SER

Mitigation Schemes[C]. IEEE International Conference on Integrated Circuit Design and Technology. IEEE, 2007:1-9.

[18] Ruckerbauer F X, Georgakos G. Soft Error Rates in 65nm SRAMs--Analysis of new Phenomena[C]. IEEE International On-Line Testing Symposium. IEEE Computer Society, 2007: 203-204.

[19] Fazeli M, Namazi A, Miremadi S G. Robust Register Caching: An Energy-Efficient Circuit-Level Technique to Combat Soft Errors in Embedded Processors[J]. IEEE Transactions on Device and Materials Reliability, 2010, 10(2): 208-221.

[20] Merelle T, Saigne F, Sagnes B, et al. Monte-Carlo simulations to quantify neutron-induced multiple bit upsets in advanced SRAMs[J]. IEEE Transactions on Nuclear Science, 2005, 52(5): 1538-1544.

[21] Georgakos G, Huber P, Ostermayr M, et al. Investigation of Increased Multi-Bit Failure Rate Due to Neutron Induced SEU in Advanced Embedded SRAMs[C]. IEEE Symposium on VLSI Circuits. IEEE, 2007: 80-81.

[22] Asadi G H, Sridharan V, Tahoori M B, et al. Balancing Performance and Reliability in the Memory Hierarchy[C]. IEEE International Symposium on Performance Analysis of Systems and Software. IEEE, 2005: 269-279.

[23] Austin T M. DIVA: a reliable substrate for deep submicron microarchitecture design[C]. MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture. IEEE, 1999: 196-207.

[24] Mottaghi M H, Zarandi H R. DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors[J]. Microprocessors and Microsystems, 2014, 38(1): 88-97.

[25] Liu W, Zhang W, Wang X, et al. Distributed sensor network-on-chip for performance optimization of soft-error-tolerant multiprocessor system-on-chip[J].

IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016, 24(4): 1546-1559.

[26] Abdelmajid B, Johannes Z, Walter S, et al. Multi-bit Soft- and Timing Error Detection for CPU Pipelines[J]. Proceedings of EDA Workshop, 2009, 15(3): 336-340.

[27] Baeg S, Wen S, Wong R. SRAM Interleaving Distance Selection With a Soft Error Failure Model[J]. IEEE Transactions on Nuclear Science, 2009, 56(4):2111–2118.

[28] Omana M, Rossi D, Metra C. Latch Susceptibility to Transient Faults and New Hardening Approach[J]. IEEE Transactions on Computers, 2009, 56(9): 1255-1268.

[29] Fazeli M, Patooghy A, Miremadi S G, et al. Feedback Redundancy: A Power Efficient SEU-Tolerant Latch Design for Deep Sub-Micron Technologies[C]. IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE Computer Society, 2007: 276-285.

[30] Ebrahimi M, Evans A, Tahoori M B, et al. Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales[C]. In Design, Automation and Test in Europe, 2014: 1-6.

[31] Naeimi H, Dehon A. Fault Secure Encoder and Decoder for NanoMemory Applications[J]. IEEE Transactions on Very Large Scale Integration Systems, 2009, 17(4): 473-486.

[32] Reviriego P, Maestro J A. Efficient error detection codes for multiple-bit upset correction in SRAMs with BICS[J]. Acm Transactions on Design Automation of Electronic Systems, 2009, 14(1): 1-10.

[33] Dutta A, Touba N A. Multiple Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DAEC Code[C]. IEEE VLSI Test Symmposium. IEEE Computer Society, 2007: 349-354.

[34] Saizadalid L, Gil P, Graciamoran J, et al. Ultrafast Single Error Correction Codes for Protecting Processor Registers[C]. Dependable Computing Conference. IEEE, 2015: 144-154.

[35] Saizadalid L, Gil P, Ruiz J C, et al. Ultrafast Error Correction Codes for Double Error Detection/Correction[C]. Dependable Computing Conference. IEEE, 2016: 108-119.

[36] 胡孔阳,胡海生,刘小明.三模冗余在高性能抗辐射 DSP 中的应用[J]. 微电子学与计算机, 2019, 36(03): 58-60.

[37] 杨旭, 范煜川, 范宝峡. 龙芯 X 微处理器抗辐照加固设计[J]. 中国科学:信息科学, 2015, 45(4): 501-512.

[38] 赵昌兵. 基于 FPGA 的数字太敏 SoC 抗 SEU 加固设计[D]. 哈尔滨: 哈尔滨工业大学, 2016: 10-36.

[39] 祝名,朱恒静,刘迎辉, 等.一种检测和校正存储器双错的低冗余加固方法[J]. 宇航学报, 2014,35(08): 924-930.

[40] 赤诚, 流水线与寄存器堆抗单粒子翻转加固研究与设计[D]. 哈尔滨: 哈尔滨工业大学, 2018: 22-46.

[41] Liu S , Xiao L , Li J , et al. Low redundancy matrix-based codes for adjacent error correction with parity sharing[C]. International Symposium on Quality Electronic Design (ISQED). IEEE, 2017: 76-80.

[42] Zhu M , Xiao L , Li S , et al. Efficient Two-Dimensional Error Codes for Multiple Bit Upsets Mitigation in Memory[C]. International Symposium on Defect and Fault Tolerance in VLSI Systems. IEEE, 2010:129-135.

[43] 李安龙, RTL 级故障注入工具开发与应用[D]. 哈尔滨: 哈尔滨工业大学, 2017: 12-18.

攻读学位期间发表的学术论文

[1] Liyi Xiao, Yuangang Wang, Zuqiang Zhang, Jiaqiang Li, Jie Li. Radiation Hardened Design of Pipeline and Register File in Processor[C]. The IEEE 13th International Conference on ASIC. (已投，待审).

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《处理器中流水线和寄存器堆抗辐射加固技术研究》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：王远刚　　日期：2019年 6月 21日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：王远刚　　日期：2019 年 6月 21日

导师签名：[illegible]　　日期：2019年 6月 21日

致　谢

研究生生涯转瞬即逝，回忆往昔历历在目，留下太多美好的回忆，我衷心的感谢硕士期间身边的老师、同学、实验室师兄师姐和远方的朋友。

首先，非常感谢我的导师肖立伊教授，感谢肖老师在我的硕士期间对我无微不至的关怀和教导，感谢老师在我硕士课题上提出的指导性意见，肖老师为人和蔼可亲，每次自己遇到问题，老师都给予悉心解答，肖老师不仅在学术上给予很多指导建议，在生活上也是，每当自己生病时，老师都是无微不至的关怀，每当自己在学习工作上有压力时，老师都能及时排忧解难提出指导性建议，减缓自己的压力，肖老师在同学心中不仅是学业上的老师，更像生活上的亲人，因此在整个硕士生涯，在实验室我都能感到家的温暖，在此衷心感谢我的导师肖老师，我也将永生铭记肖老师的恩情。

其次感谢王进祥老师、来逢昌老师、王永生老师和付方发老师，感谢他们对我课题提出的指导性意见，不仅丰富了自己的知识，而且在整个毕业设计中少走了一些弯路，他们治学严谨、精益求精的精神也时时熏陶着我。

感谢李家强师兄，给我的毕设课题提供了不少建议和思路，同时李家强师兄组织实验室活动，给整个团体带来不少欢乐，感谢强哥教会了我许多为人处世的道理。感谢李杰师兄，在整个硕士期间在技术上提供的各种帮助，每次问杰哥问题都是细心解答不厌其烦，感谢杰哥为实验室做的贡献，感谢曹学兵师兄、张荣生师兄、李林哲师兄、李洪辰师姐以及刘贺师兄对我生活和学业上的帮助。感谢张海龙师弟、李广鹏师弟和王志敏师弟为实验室所带来的快乐和拼搏奋斗的精神。

感谢我的室友张祖强、舒德刚和李钦鹤，每当自己生活遇到困惑时都能和你们畅谈人生，感谢你们为我的生活带来的欢乐，兄弟情永远在心中。

感谢自己的家人，感谢父母育我成人，感谢你们在我成长的环境中抵抗外界的压力为我遮风挡雨，你们不辞劳苦为孩子奉献，给我提供了精神的支柱，没有你们也就没有我今天的成绩。感谢哥哥和姐姐对我生活上的照顾，从小到大伴我成长，遇到困难互相扶持，有你们让我感觉生活满满的幸福感。

最后，对所有参加论文评审和答辩的各位专家、教授和老师的辛苦付出表示衷心的感谢。