分类号：TN492　　单位代码：10335
学　　号：21760268

浙江大学

硕士专业学位论文

中文论文题目：工业微控制器安全容错技术

英文论文题目：Security and Fault tolerance Technique for Industrial Microcontroller

申请人姓名：陈 群

指导教师：黄 凯

专业学位类别：工程硕士

专业学位领域：集成电路工程

所在学院：信息与电子工程学院

论文提交日期：2020 年 3 月

工业微控制器安全容错技术

论文作者签名：陈群

指导教师签名：

论文评阅人 1：汪楠 副教授 华东理工大学

评阅人 2：匿名评阅人

评阅人 3：匿名评阅人

评阅人 4：

评阅人 5：

答辩委员会主席：马琪 研究员 杭州电子科技大学

委员 1：罗小华 副教授 浙江大学

委员 2：张培勇 副教授 浙江大学

委员 3：

委员 4：

委员 5：

答辩日期：2020 年 3 月 14 日

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 浙江大学 或其他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谢意。

学位论文作者签名：陈群　　签字日期：2020 年 3 月 16 日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：陈群　　导师签名：

签字日期：2020 年 3 月 16 日　　签字日期：2020 年 3 月 16 日

致 谢

时光匆匆如流水，转眼我的学生时代就快结束了。回首两年半的研究生生活，渐渐懂得“生活不会总是一帆风顺”是个真理。在这期间，我时常感到迷茫与沮丧，有时会疑惑学习、生活与未来的意义在哪里。然而，每当我失去方向的时候，身边的亲人、师友总会及时给我支持和鼓励，让我能够继续坚持下去，在这里衷心的向你们表示感谢。

首先要感谢我的研究生导师黄凯老师。在您的带领下，我接触了有关数字集成电路的实际项目，在项目中快速学习和成长，并不断提升自身专业知识水平。在最后的毕业设计中，即便您的日常工作很忙，也会抽出时间指导我的科研工作。遇到问题无法解决时，您也总能给我提供许多指导性建议和帮助。在这里向您表示衷心的感谢和诚挚的敬意。

其次要感谢林威、何赟、熊东亮、蒋小文、王轲、赵通、易哲为、霍佳琦等几位师兄，在学业和工作上给予我的帮助和助攻，你们是我在学习道路上的榜样，我因有你们这些优秀的师兄而骄傲。感谢邵胜芒、田小波、陈子旋、郑昌立等与我同届的研友，闲暇时光里与你们一起吃吃喝喝，谈人生，谈理想的岁月让枯燥的研究生生活平添了许多色彩。感谢吴靖康、杜俊慧、陈思恒、井铭、江海天、万昕茗、沙天薏、余宏洲等几位师弟师妹，因你们的加入，我的研究生生活充满了青春与活力！

还要感谢我的家人，你们是我坚强的后盾，无论我遇到什么困难与挫折，你们总是陪伴我，给我鼓励，让我能够勇敢自信地去面对挫折。也感谢何赟师兄的陪伴，在对父母报喜不报忧的日子里，有你和我一起承担生活、学业中的烦恼与忧愁。

时光匆匆如流水，在岁月的长河里，感谢有过你们的出现。

摘 要

随着工业 4.0 时代的到来，工业微控制器在我国工业自动化发展中正扮演着越来越重要的角色。相比较一般的消费级应用，工业微控制器对可靠性、安全性、低成本及实时性上的要求更高。然而，不断更新的工艺节点以及不断发展的攻击技术对工业微控制器的可靠性和安全性带来严峻的挑战。就可靠性而言，目前常用的容错方法或占据较大面积，或性能开销大，实时性不足。就安全性而言，针对目前较为流行的功耗分析攻击抵御方案大都针对专门的硬件加解密模块，对与处理器相关的抗攻击研究不足，业界产品也较少。针对这一情况，本文在现有的基础上，针对微控制器的容错和处理器抗功耗分析攻击方面进行了相关研究。

在可靠性方面，本文通过对现有技术和业界产品进行调研，针对嵌入式处理器，提出了一种基于全硬件的 Lockstep 容错设计，该容错技术能够同时解决挂起和结果错误类型的错误，通过对处理器双模冗余并增加硬件容错模块，实现故障的实时检测和恢复，以及写通模式下的片上缓存容错。针对微控制器内的片上存储，分析了单比特错误与多比特错误发生概率，并分析常用检错码的漏检率等特性，最终设计并实现了基于汉明码和循环冗余校验码的纠一检多容错方案。

在安全性方面，本文研究了现有抵御功耗分析攻击的方案，并对业界有关抗攻击处理器产品进行了调研。基于容错的双核架构，提出了一种基于功耗隐藏的抗功耗分析攻击设计。通过在系统硬件层面，对主从处理器进行可控的随机延时插入，在保证处理器容错的同时，进行振幅维度及时间维度上的功耗隐藏，增加攻击者进行攻击的难度，从而提高微控制器的安全性。

最后，本文通过研究故障注入技术，搭建了基于仿真的故障注入平台，对处理器进行了容错测试，并与现有常见的处理器容错方案在面积，容错率，性能和故障恢复时间之间进行了对比，结果证明该方法在各项指标间进行了良好的折衷。针对安全性测试，以差分功耗分析攻击为例，搭建了差分功耗分析攻击平台，对处理器进行了攻击测试，证明了功耗隐藏设计的有效性。

关键词：微控制器；容错；锁步；功耗分析攻击；错误检测与纠正

Abstract

With the advent of the Industry 4.0 era, industrial microcontrollers are playing an increasingly important role in the development of industrial automation. Compared with general consumer products, industrial microcontrollers have higher requirements on reliability, security, area and real-time performance. However, with the scaling down of the characteristic size of circuit and the evolving of attack technologies, there are severe challenges to the reliability and security of industrial microcontrollers. From a reliability perspective, the fault tolerance currently used either occupy a large area, or have large performance overhead and insufficient real-time performance. From a security perspective, most of the researches on resisting power analysis attacks have focused on specific hardware encryption and decryption modules. However, there are few researches and products on processors. In view of this situation, this paper studies the fault-tolerance of the microcontroller and the anti-power analysis attack of the processor.

In terms of reliability, this paper studies the existing technology and industry products, and proposes a full hardware-based Lockstep fault-tolerant design for embedded processors. The propoesd fault-tolerant technology can simultaneously solve hang and silent data corruption errors. By replicating the processor and adding a hardware fault-tolerant module, both real-time fault detection and recovery, and on-chip cache fault tolerance in write-through mode are realized. As for the on-chip memory, the single-bit error and multi-bit error occurrence probabilities and the characteristics of the missed detection rate of common error detection codes are analyzed. Finally, a hamming code and cyclic redundancy check -based single error correction and multiple errors detection are finally designed and implemented.

In terms of security, this paper studies the existing solutions to resist power analysis attacks, and investigates the industry's anti-attack processor products. Based on the fault-tolerant dual-core architecture, an anti-power analysis attack design based on power hiding is proposed. By inserting the random delay in master and slave

processors at system hardware level, the power consumption in the amplitude and time dimensions is hidden. This method can improve the security of processor while ensuring the fault-tolerance of processor.

Finally, this paper studies fault injection technology, and builds a simulation-based fault injection platform to test the ability of fault-tolerance processor. Compared with existing common processor fault tolerance schemes, the proposed method provide a good tradeoff among area, fault-tolerance, performance, and fault recovery time. For security testing, a differential power analysis attack platform is built. The effectiveness of proposed power hidden technique is proved by experiment for embedded processor.

Keywords: Microcontroller; Lockstep; Fault tolerance; Power analysis attack; EDAC

目 次

摘 要.....2
Abstract.....III
目 次.....V
图目录.....VIII
表目录.....X
1 绪 论.....1
1.1 课题研究背景.....1
1.2 技术概述及国内外研究现状.....2
1.2.1 微控制器容错技术研究及现状.....2
1.2.2 微控制器安全性技术研究及现状.....5
1.3 研究意义与研究内容.....8
1.4 论文结构.....9
2 针对工业微控制器的可靠与安全设计.....11
2.1 微控制器结构.....11
2.1.1 嵌入式处理器.....11
2.1.2 总线.....13
2.2 微控制器容错方案设计.....14
2.2.1 软错误对微控制器的影响.....14
2.2.2 基于检查点的 LOCKSTEP 技术.....16
2.2.3 容错方案设计.....17
2.3 嵌入式处理器抗功耗分析攻击设计.....19
2.3.1 差分功耗分析攻击原理.....19
2.3.2 功耗隐藏技术研究.....21
2.3.3 抗功耗分析攻击方案设计.....23
2.4 本章小结.....24

3 硬件结构设计与实现.....26
3.1 基于全硬件的 LOCKSTEP 容错设计.....26
3.1.1 处理器故障检测设计.....26
3.1.2 处理器故障恢复设计.....31
3.1.3 处理器故障隔离设计.....33
3.1.4 仿真验证及逻辑综合.....41
3.2 基于双核容错的处理器抗功耗分析攻击设计.....45
3.2.1 失步运行设计.....46
3.2.2 同步比较设计.....49
3.2.3 仿真验证及逻辑综合.....50
3.3 存储器容错设计.....53
3.3.1 汉明码技术.....53
3.3.2 检错码技术.....55
3.3.3 检纠错设计.....58
3.3.4 仿真验证及逻辑综合.....63
3.4 本章小结.....65
4 可靠性与安全性测试.....67
4.1 容错性能测试.....67
4.1.1 故障注入技术分析.....67
4.1.2 仿真故障注入平台设计.....68
4.1.3 实验结果与对比分析.....70
4.2 抗功耗分析攻击性能测试.....75
4.2.1 DPA 攻击平台设计与实现.....76
4.2.2 实验结果.....80
4.3 本章小结.....83
5 总结与展望.....85
5.1 论文工作总结.....85

5.2 论文的局限与展望.....86
6 参考文献.....87

图目录

图 2.1 微控制器结构框图.....11
图 2.2 CK803 结构框图.....12
图 2.3 AHB 基本的读写时序图.....14
图 2.4 两种常用的 Lockstep 硬件架构图.....16
图 2.5 基于检查点的 Lockstep 技术容错流程图.....17
图 2.6 微控制整体容错方案框图.....19
图 2.7 处理器抗攻击方案及预期效果示意图.....24
图 3.1 单个容错模块的结构框图.....26
图 3.2 lc_error_raw 的同步电路示意图.....31
图 3.3 正确节点的状态保存时序图.....32
图 3.4 状态信息重置电路结构图.....33
图 3.5 处理器状态回滚而外部状态无法回滚造成的故障时序图.....34
图 3.6 写操作缓冲区结构图.....35
图 3.7 数据总线的写操作修改时序图.....35
图 3.8 数据总线的读操作修改时序图.....36
图 3.9 故障发生后存储器的状态回滚时序图.....37
图 3.10 系统总线的写操作修改时序图.....38
图 3.11 复位期间对缓存的无效操作时序图.....41
图 3.12 内部寄存器出错，处理器故障恢复时序图.....41
图 3.13 Cache 数据出错，处理器故障恢复时序图.....42
图 3.14 处理器系统总线上的写操作隔离及故障恢复时序图.....43
图 3.15 处理器数据总线上的写操作隔离及故障恢复时序图.....44
图 3.16 处理器 Cache 的写操作隔离及故障恢复时序图.....45
图 3.17 主从处理器同步与失步运行示意图.....46
图 3.18 安全容错模块结构框图.....47
图 3.19 随机延迟使能模块结构图.....48
图 3.20 主从处理器异步运行后的指令读取顺序不同.....48

图 3.21 程序编译后的.obj 文件格式 49
图 3.22 功耗隐藏使能后，主从处理器异步运行时序图.....51
图 3.23 写操作时，主从处理器同步运行时序图.....51
图 3.24 功耗隐藏使能期间发生故障后的恢复时序图.....52
图 3.25 检纠错模块结构图.....53
图 3.26 4 位信息码汉明校验分组示意图.....54
图 3.27 24 位信息码水平奇偶校验示意图.....56
图 3.28 检纠错编码过程示意图.....59
图 3.29 CRC12 对应的编码电路.....60
图 3.30 检纠错译码过程示意图.....61
图 3.31 错误纠正电路结构图.....62
图 3.32 CRC 译码电路.....62
图 3.33 无错误下编译码时序图.....64
图 3.34 单比特错误下编译码时序图.....64
图 3.35 多比特错误下编译码时序图.....65
图 4.1 故障注入平台结构框图.....69
图 4.2 sum_report 内容 70
图 4.3 HCL、CLC、TMR 三种容错方案指标对比 73
图 4.4 HLC 与 TMR 容错下处理器进行外设访问的性能对比.....75
图 4.5 功耗曲线收集流程图.....76
图 4.6 AES 加密算法流程图 77
图 4.7 out 文件格式 78
图 4.8 数据分析流程图.....79
图 4.9 功耗扰乱关闭时的 DPA 攻击结果.....81
图 4.10 功耗扰乱开启时的 DPA 攻击结果（80 条曲线） 82
图 4.11 功耗扰乱开启时的 DPA 攻击结果（2000 条曲线） 83

表目录

表 3.1 CK803 应对比及保存的内部寄存器列表.....27
表 3.2 CK803 应对比的外部接口信号列表.....28
表 3.3 CK803 应对比的缓存接口信号列表.....30
表 3.4 单个处理器与容错处理器综合后的指标对比.....45
表 3.5 单个处理器与安全容错处理器综合后的指标对比.....52
表 3.6 常用信息码对应的汉明码校验位长度.....54
表 3.7 校正子与错误位置关系.....55
表 3.8 常用检错码的检错能力比较.....58
表 3.9 检纠错模块综合后的指标.....65
表 4.1 单比特及多比特故障下的故障恢复情况.....71
表 4.2 各外设对应的测试用例说明.....74
表 4.3 汉明重量模型.....79

1 绪 论

1.1 课题研究背景

微控制器是一种集成电路芯片，它将 CPU，存储，I/O 以及其他多种外围电路集成到一块硅片上，构成了一种功能完善的微型计算机系统。由于其具有体积小、重量轻、性能好，成本低，可靠性高等优点，微控制器在工业控制中的运用越来越广泛。相比较面向消费类的微控制器产品，工业微控制器对于高可靠，高安全，低成本和实时性的需求更高。然而，随着科学技术的不断发展，工业微控制器的可靠性和安全性正面临着严峻的考验。

可靠性问题的来源主要来自工艺节点的降低和低功耗技术的发展。由于特征尺寸和电压阈值不断降低，半导体集成电路对电路串扰，大气中的辐射，封装材料衰变产生的高能粒子，极端温度，电磁干扰等因素也愈加敏感。与此同时，日益广泛的应用使得工业微控制器的工作环境更加复杂，遭遇恶劣条件的概率更大，因而受到干扰发生故障的可能性也就越高。有文献指出，计算机系统中，80%-90%的失效都是由于这种干扰导致的瞬态故障引起的[1]。瞬态故障是一种软错误，是由外界条件干扰导致半导体中随机、临时状态的改变或瞬变。相对于硬错误而言，软错误并不会造成永久性电路故障，能够通过复位，恢复受影响器件的功能。然而对于工业微控制器来说，运行过程中，任何一比特的错误都可能导致错误结果的输出或者整个系统的失效，这对工业应用来说，可能造成巨大的财产损失甚至人员伤亡。对于微控制器可靠性问题，早期主要是通过 CMOS（Complementary Metal Oxide Semiconductor，互补金属氧化物半导体）工艺解决，然而，一方面特殊工艺的成本高昂，违背了工业微控制器低成本需求；另一方面，当特征尺寸不断降低后，可靠性问题又会出现。因此，降低对工艺的依赖，从设计本身出发，提高微控制器的容错能力成为当前微控制器可靠性问题的研究热点。

安全性问题主要来自微控制器数据的泄露和篡改。为了保证微控制内的敏感信息不被泄露，通常会对这些信息进行加密。然而，近年来针对加密芯片的攻击越来越频繁，目前较为流行的一种攻击方式是功耗分析攻击，这是一种通过分析硬件运算时泄露的功耗信息获取密钥的攻击方式，由于它绕过对加密算法本身的

繁琐分析，直接着手功耗与数据间的相关性分析，因而能够快速准确地获取密钥。对于微控制器来说，如果需要进行软件加密，内部的处理器在执行加密算法时，实际功耗会随着处理器内部执行指令及数据的不同而有所区别。攻击者可以通过功耗与指令及数据间的相关性，结合统计数据分析获取密钥。在毫无保护的设备上，这种攻击方式能够以低廉的攻击成本快速获得密钥，进而获取敏感信息，这对工业应用来说，将带来灾难性的后果。因此，对于高安全需求的工业微控制器来说，对嵌入式处理器模块的进行抗功耗分析攻击具有重要的意义。

综上，由于工业微控制器在我国工业自动化，生产力发展道路上仍占据着重要作用，研究工业微控制器的可靠性与安全性技术，能够为工业应用带来更高的可靠性和安全性，促进我国的工业发展，与业界实际需求一致，具有重要的实践意义。

1.2 技术概述及国内外研究现状

1.2.1 微控制器容错技术研究及现状

容错设计的基本思想基于冗余[2]，通过增加冗余资源，检测当前是否出现故障并对故障进行一定程度的恢复。目前的冗余方式主要分为以下几类：时间冗余，硬件冗余，软件冗余和信息冗余。下面是对各类冗余技术及其在微控制器容错方面的相关介绍。

（一）时间冗余

时间冗余技术是指一个程序或运算在同一硬件功能模块中执行多次，并将每次执行获得的结果进行对比，从而检测错误，或对故障进行屏蔽和恢复。时间冗余可以通过硬件或软件的方式实现。例如，Patel[3]针对容易出错的算术逻辑单元，在硬件上将同一数据在同一运算单元上执行两次，然后将执行结果进行对比以检测故障。Asghari[4]在软件上通过一个 Manager Task 管理代码的冗余执行，先对软件进行两次运算，以检测错误，若比较不一致时，再进行第三次运算，以此选择正确结果输出。目前，有关时间冗余的研究有相当一部分研究集中在处理上。例如，文献[5]设计了基于时间冗余的 DoubleRun 处理器核，通过将程序以时间冗余的方式分段执行，确保其中无故障产生后再提交新产生的程序状态。文献[6]提

出在流水线中以两个冗余线程的方式两次取指，执行所有指令，并比较指令结果以检测处理器整个流水线中的故障。时间冗余的优点是不需要或只需少量地加入额外硬件电路就可以实现对故障的检测甚至恢复，然而，带来的缺点是系统的性能损失大，只适用于时间资源充裕的应用。

（二）硬件冗余

硬件冗余技术是指通过对功能单元进行多次复制，比较多个单元输出结果，从而检测故障，或对故障进行屏蔽和恢复。最早是由冯•诺依曼提出的三模冗余（Triple Modular Redundancy，TMR）[7]技术，该技术对功能单元进行三次复制，通过对比三个功能模块判定输出结果正误。相比较时间冗余，TMR 技术基本没有延迟，不存在性能损失。在一些超高可靠性需求设计中，针对多个功能模块可能同时出现故障的情况，还有 N 模冗余（N Modular Redundancy，NMR）技术。例如，Abdulhay[8]为应对两个模块同时出现错误的情况，提出了五模冗余。然而，TMR 和 NMR 技术的面积占用大，这在低成本要求的设计中令人难以接受。在一些低成本设计中，常用的是双模冗余（Double Modular Redundancy ，DMR），只对功能模块进行两次复制。例如，在电路级别，常通过比较时钟受延迟的影子锁存器和正常锁存器中的值检测时钟错误[9][10][11]，但通常只做检错，无法恢复。Sheikh[12]在电路进行双模冗余的同时，基于当前电路输出逻辑 0（逻辑 1）的概率对双模后的输出进行与操作（与非操作），能够在检测错误的同时，以最小的面积开销进行部分错误的恢复。在模块级别，例如处理器中，一种典型的容错结构是 CON-MON[13]，该结构在系统中采用双核，分别为主核和监控核，监控核的面积更小，功能更少，主要用于对主核执行的关键运算进行检查，能够检查主核发生的错误，但无法进行恢复。另一种是比较典型的应用是锁步（Lockstep）技术，该技术通过复制两个功能模块，使其成为自监控对，相比 CON-MON 能够更快检测出处理器发生的故障。基于检查点的 Lockstep 技术[14][15][16][17]由于在锁步的基础上，结合检查点（Checkpoint）技术，在软件上进行状态保存和状态恢复，能够在检测故障的同时，完成故障恢复，是目前处理器容错中一种主流的方法。但由于故障恢复的部分利用了软件方式进行，因而在容错实时性上还存在一定的不足。

（三）软件冗余

软件冗余技术是指通过在原有的功能软件上增加额外的软件代码，检测系统中发生的故障，通常应用于包含处理器的系统中。例如，文献[18]通过在程序中设置断言以及看门狗（Watch Dog Timer，WDT）对处理器中的控制流进行错误检测。由于软件冗余基于纯软件，无需更改硬件，可以在商用现货微控制器中使用，但多余的软件会增加代码量，影响正常程序的执行效率和系统的实时性。为了减少冗余代码带来的性能开销，许多研究学者对软件冗余的算法进行优化。例如，文献[19]提出在软件层面通过近似计算，以牺牲计算结果的精确度为代价来提升性能和降低功耗。文献[20]改进了常用的实时性容错 BCE 算法，进一步提高了容错系统的实时性。尽管，相关的优化算法较多，但与时间冗余一样，软件冗余难以从根本上解决对系统性能开销的影响。

（四）信息冗余

信息冗余技术多用于保护存储器，通过对数据进行编码实现故障检测。如常见的错误检测与纠正技术（Error Detection And Correction，EDAC）。常用的检纠错码包括汉明码，BCH 码，奇偶校验码等。信息冗余技术会对写入的数据按照一定的方式编码，得到的校验码和数据同时被写入存储区，当该数据被读出时再按相应的方式进行译码，根据译码结果完成错误的检测和纠正。

无论以上采用何种冗余方式，都会增加或面积或性能上的开销。面对不同的设计需求，业界所采用的容错方式不尽相同。法国的 iROC Technology 公司专为现今愈加复杂的微电子系统提供可靠性解决方案。其早期设计的一款名为 ROCS81 的 32 位 RISC 容错处理器[21]，兼容 SPARCV8 指令系统，采用基于时间冗余的触发器设计技术，对所有的触发器都增加一个冗余触发器，并在指令 Cache、数据 Cache、寄存器堆中采用了奇偶校验以及相关恢复机制，以检测和消除存储器中的软错误。随后，又提出了一种容错触发器[22]，通过在每一个触发器输出端添加翻转检测和局部错误纠错逻辑，实现单个触发器的容错，并在它们设计的 32 位 MIPS 微处理器中得到了使用。德国的 Infineon 公司是一家专注汽车电子的公司，其半导体器件以高可靠性著称。Infineon 推出的 AURIX 系列微控制器[23]，通过内嵌 3 个处理器，进行三核 Lockstep，并且为了尽量减少不同的核

发生一致性故障的情况，在电路，时序设计以及版图设计中进行了差异性处理，可达业界对于可靠性的最高要求。同样进行三核锁步设计的，还有 Arm 公司推出的基于 Cortex-R5 的三核 Lockstep 架构[24]，通过三个核同步运行相同指令，实时对比三核的输出结果，经多数表决后输出，能够在微秒内快速，自动地从故障中恢复。NXP 公司推出的 i.MX 8X 处理器[25]在工艺上使用了 FD-SOI 技术，此工艺对软错误有较高的抗扰能力。同时，对其中的 L2 缓存和 DDR3L 存储提供了 ECC（Error Correcting Code，错误纠正码）保护。TI 公司推出的新型 Hercules 安全微控制器平台[26]中采用了两个 ARM Cortex-R4F 核进行 Lockstep，并对存储器单元进行了 ECC 保护。Freescale 的 MPC5675K 系列微控制器[27]对于内部的处理器，存储器，以及桥间进行了指令级 Lockstep，并在内部 FLASH 及 SRAM（Static Random-Access Memory，静态随机存取存储器）上进行了 ECC 保护。

相比国外众多有关高可靠微控制器相关的业界产品，国内有关工业微控制器的可靠性研究起步晚，有关可靠性的研究主要集中在航天航空领域，国内的部分研究所通过集成电路抗辐射加固设计与工艺取得了许多的研究成果，也相继研制成功了一些加固指标接近国际标准的先进集成电路产品[28]。然而，有关工业微控制器的可靠性研究相对较少，一些国内高校如国防科技大学，厦门大学，北京交通大学等也做了相关研究，但是大多数研究仍停留在理论阶段，缺乏已被业界广泛使用的相关产品。

综上可知，业界对于微控制器的容错主要是对嵌入式处理器以及内部的存储器进行容错。嵌入式处理器的可靠保证了微控制器的控制可靠，而存储器的可靠保证了处理器所运行的代码、数据可靠。目前，通过 EDAC 技术提高存储器的可靠性已成为业界较为统一的解决方案，需要着重考虑的问题是如何针对不同的设计需求选择不同的检纠错码进行容错。对于嵌入式处理器来说，由于其复杂的功能和架构，采用冗余提高可靠性的同时，势必造成性能，面积，可操作性及复杂度上的损失。对于工业微控制器来说，可靠性方案在可实行的基础上，如何尽量保证工业微控制器在低成本和实时性上的要求，仍是亟待研究解决的问题。

1.2.2 微控制器安全性技术研究及现状

相比专门的硬件加密模块，针对处理器的功耗分析攻击更容易实现，因为通常硬件加密模块在设计时，针对特定的算法会考虑功耗掩盖的问题，而针对处理器抗攻击的设计相对较少。事实上，完全抵御功耗分析攻击非常困难，因为消除信息泄露几乎不可能。对于业界产品来说，只要能够在一定程度上增加攻击者的攻击难度，使得攻击成本远大于攻击带来的利润，那么抵御攻击的方案就是有效的。目前能够增加攻击难度，抵御功耗分析攻击的技术主要分为两类：隐藏技术和掩码技术，以下是这两类技术的介绍。

隐藏技术是通过一些方法切断被处理数据与硬件设备功耗消耗间的关系。由于进行功耗分析攻击时需要在多条功耗曲线对齐的基础上进行统计分析，因此，时间维度上，可以通过打乱各个功耗曲线攻击点在时间轴上的位置进行功耗隐藏；振幅维度上，可以通过减小功耗信噪比进行功耗隐藏。对于时间维度的隐藏来说，最常用的方法是插入伪操作，随机延时或乱序执行。例如，文献[29]通过在处理器的预取流水线前通过随机地将当前指令无效，完成指令间的随机延迟，从而进行时间维度上的功耗隐藏。文献[30]提出随机改变电路的时钟频率，使得各功耗曲线间对齐困难，从而增加攻击难度。对于振幅维度的隐藏来说，在硬件层面可以添加噪声引擎，例如杭州晟元微在其申请的专利[31]中，提出在系统内部集成一个功耗噪声单元，在系统执行加解密运算时，随机地打开或关闭功耗噪声单元，以在振幅维度上进行功耗隐藏。文献[32]通过分析特定加密算法，针对 AES（Advanced Encryption Standard，高级加密标准）及 DES（Data Encryption Standard，数据加密标准）提出了相关的功耗平衡算法，在多核系统中进行软件加解密时，让双核同时运行平衡后的算法，以达到振幅上的功耗平衡。文献[33]在文献[32]的基础上，提出在单核系统中，利用双带宽平衡算法也可对特定算法进行功耗平衡。文献[34]采用双栅预充电技术，使得电路的功耗消耗不依赖于被处理逻辑值。文献[35]在密码设备供电线路中插入了 RLC 滤波器，能够减小功耗的实际信噪比。

掩码技术是通过随机化中间值切断功耗信息与实际数据间的相关性。掩码技术的使用需要保证每一个中间值都处于被掩码状态，同时还需在结束计算时消除掩码。相比较隐藏技术，掩码技术的一个优点是在算法级实现，并且无需改变密

码设备的功耗消耗特性，因此，在硬件加密模块中的使用非常广泛。目前，有相当多有关掩码方案的技术研究。文献[36]讨论了用于 AES 算法的掩码技术，并给出了布尔掩码与算术掩码间的转换算法。文献[37]则讨论了用于 DES 的掩码方案。此外，针对处理器上的软件加密抗攻击，文献[38]针对 AES 算法，提出在 AES 软件实现上进行算法修改，对关键部分进行掩码操作。然而这种方法缺乏普适性，并且 S 盒部分掩码设计所消耗的时间甚至超过加密本身所需时间。文献[39][40]则提出在 RSIC-V 处理器指令执行级进行设计，对处理器内部的 ALU（Arithmetic and Logic Unit，算术逻辑单元）和寄存器进行掩码保护，能够减少处理器性能开销并具备通用性，然而相关的设计需要在处理器内部进行，实现较为复杂。

为了保护嵌入式处理器上运行的软件加密不被攻击，一些企业也采取了相关设计来保证工业微控制器的安全性。国外方面，Infineon 公司推出了 Integrity guard[41]技术，该方法使用了两个嵌入式处理器，同时运行实时对比，可以对故障攻击导致的错误状态进行检测及恢复。同时，在整个处理器及存储器上进行了加密处理，并且两处理器间具有不同的动态密钥，使得处理器在运行代码时，功耗信息与实际数据间的相关性几乎被完全切断。此外，ARM 公司针对智能卡应用推出了 SC300 嵌入式处理器[42]，其在 cortex-M 的架构上增加了有关安全的相关特性，能够抵御功耗分析攻击。然而，ARM 并没有对外公开具体采用的设计方式。意法半导体公司[43]在则在 ARM SC300 核的基础上，构建了高安全性的微控制器。

由于国内集成电路的发展起步晚，国内有关嵌入式处理器设计的公司并不多，能够在处理器级别进行功耗分析攻击防御的产品非常少。目前，已知的业界相关产品是杭州中天微系统有限公司的 CK802 系列 CPU[44]，它支持抵御物理攻击，适用于安全支付、智能卡等高安全领域应用。总体来看，相关的研究及产品与国外相比无论是种类还是技术上都还存在一定的差距。

综上可知，目前业界有关嵌入式处理器的防功耗攻击产品很少，一方面，国内外自主设计嵌入式处理器的公司较少，因此直接从处理器内部进行抗攻击设计的相关公司和产品并不多；另一方面，由于安全性需求，现有的国内外产品中，

所使用的抗攻击技术并不公开，难以进行技术上的讨论交流。因此，针对处理器的抗功耗分析攻击仍有很大的研究空间。

1.3 研究意义与研究内容

迄今为止，高可靠性，安全性，低成本以及实时性依旧是工业微控制器设计所需围绕的主题，尤其是国内的一些关键基础设施，如配电系统，经济金融，轨道交通等，一旦出现问题，其带来的损失难以估计。想要保证微控制器的可靠性与安全性，微控制器的核心嵌入式处理器必须具备安全容错能力。经历过中兴事件和华为制裁后的中国芯正面临着芯片自研道路上最关键的时刻。一方面，国产核的自主可控是众多高安全芯片的首要选择，另一方面，我国自主可控的国产核与国外的产品还有一定的差距，除了性能上的差异外，在可靠性和安全性上也还有待提高。国内的微控制器设计公司在追求高可靠高安全的同时，不应当只依赖国产核本身的特性，还应该在微控制器系统内增强微控制器的安全容错特性。对于微控制器设计公司来说，大部分 IP 来源于第三方购买，对 IP 内部细节的设计并不了解，而对于功能复杂的 IP，如嵌入式处理器，即便拥有软核，获得 RTL 代码，其内部复杂的设计也难以让微控制器设计公司在短时间内掌握并做有信心的安全容错修改。因此，对于他们来说，在系统级和软件级增强安全容错特性最为直接并更具操作性。考虑到实时性要求，在系统级硬件层面进行设计更为合适，并且不需要再对大量的软件进行重开发，降低了人力成本。

本研究基于以中天微 CK803 嵌入式处理器为核心的微控制器，从微控制器硬件系统层面，在安全容错，面积与实时性之间进行权衡，提出了基于全硬件的 Lockstep 处理器容错设计，并在此基础上，完成系统级的处理器抗攻击设计，以及存储器容错设计，最终完成微控制器的安全容错设计。主要的研究内容和创新点具体包括以下几个方面：

（1） 研究了软错误对微控制器的影响，以及现有的各类容错技术和业界产品，结合以 CK803 为核心的微控制器架构，制定了微控制器整体的安全容错方案。

（2） 研究了目前较为流行的基于检查点的 Lockstep 处理器容错技术，并针对

其不足，提出并实现了全硬件的 Lockstep 容错技术。该容错技术能够同时解决 Hang 和 SDC（Silent Data Corruption，结果错误）类型的错误，实现故障的实时检测和恢复，以及写通模式下的片上缓存容错。

（3） 研究功耗隐藏技术，并根据容错的双核架构，实现了在时间和振幅双重维度上的功耗隐藏设计，在支持处理器容错的同时，能够更好的抵御功耗分析攻击，增加了攻击者的攻击成本。

（4） 分析了半导体电路在不同环境中发生故障的概率以及几类检错码的漏检率等特性，最终选择使用汉明码和循环冗余校验码实现对存储器的检纠错，使其能在正常环境中纠错，恶意攻击环境中检错。

（5） 研究了故障注入技术及 DPA（Differential Power Analysis，差分功耗分析）攻击原理，搭建了用于可靠性测试的故障注入平台，以及用于安全性测试的 DPA 攻击平台，并针对测试结果进行了相关对比分析。

1.4 论文结构

本文主要研究工业微控制器的安全容错技术，全文可分为五章，章节安排如下：

第一章：绪论。阐述了工业微控制器可靠性和安全性面临的挑战，介绍了微控制器在容错，以及内部处理器在抵御功耗分析攻击方面国内外现有的技术，以及业界相关的设计产品，并根据以上情况确立了本文的研究内容。

第二章：针对工业微控制器的可靠与安全设计。首先介绍了微控制器结构，包括内部的总线协议以及处理器特性。其次，研究并分析了软错误对于微控制器的影响；根据研究内容对微控制器的容错设计提出了整体的方案，并根据容错架构，对处理器进行了功耗隐藏设计，以抵御功耗分析攻击。

第三章：硬件结构设计与实现。在整体方案的基础上，对微控制器的安全容错设计进行了详细的介绍。分别是处理器容错设计，处理器抗功耗分析设计，以及存储器 EDAC 设计，并对各设计模块进行了仿真验证和逻辑综合。

第四章：可靠性与安全性测试。研究了故障注入技术，并基于仿真故障注入方式，搭建了故障注入平台，测试并统计了处理器错误恢复能力，同时，对比分

析了本文提出的容错方案与目前常用处理器容错方法的各类指标；以 DPA 为例，研究了 DPA 攻击方法，搭建了 DPA 攻击平台，对比了功耗隐藏功能开启前后的攻击效果。

第五章：总结与展望。对本文工作进行总结，并对下一步工作的研究和发展方向进行展望

2 针对工业微控制器的可靠与安全设计

2.1 微控制器结构

本研究基于的微控制器结构如图 2.1 所示，该微控制器由嵌入式处理器 CK803，AMBA 2.0 总线，随机访问存储器（Random Access Memory，RAM），嵌入式闪存（Embedded Flash，EFLASH），只读存储器（Read-Only Memory，ROM），系统 IP，外围接口以及电源模拟模块组成。CK803 是整个微控制器的核心，负责整个系统的运算和控制。AHB（Advanced High performance Bus，高级高性能总线）总线用于处理器与存储器、系统 IP 等一些高速设备进行数据传输，外围接口等低速设备通过 APB（Advanced Peripheral Bus，高级外围总线）桥接至 AHB 与处理器进行数据交换。RAM 作为微控制器中的数据存储区，通过数据总线与 CK803 通信，EFLASH 和 ROM 作为微控制器中的代码存储区，通过指令总线与 CK803 通信，系统 IP 及外设通过系统总线与 CK803 通信。

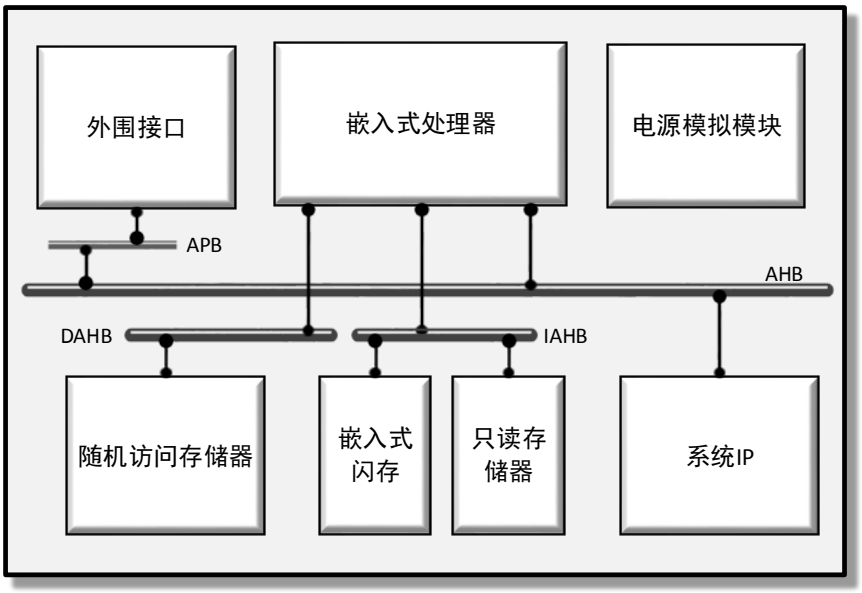


图 2.1 微控制器结构框图

2.1.1 嵌入式处理器

微控制器所采用的处理器为杭州中天微系统有限公司的 CK803 系列嵌入式处理器，它是面向控制领域的 32 位高能效嵌入式处理器核，采用 16/32 位混合

编码指令系统，内部采用 3 级流水线并提供多种可配置功能，包括硬件浮点单元、片上高速缓存、以及一些片上紧耦合 IP，如系统计时器，中断控制器等。此外，CK803 提供多个总线接口，包括指令总线、数据总线以及系统总线。其结构框图如图 2.2 所示。

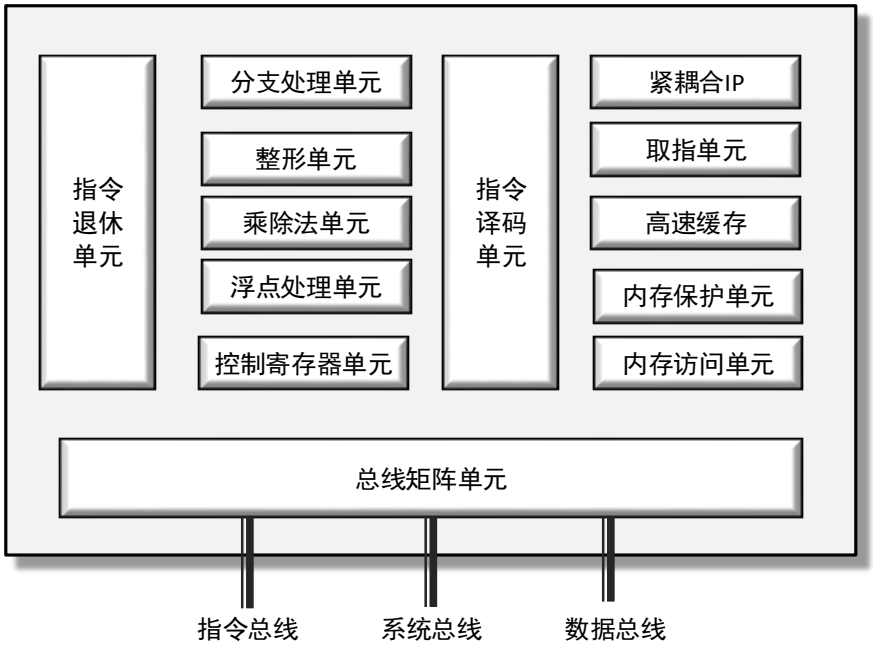


图 2.2 CK803 结构框图

CK803 的紧耦合 IP 包括系统计时器，矢量中断控制器以及片内高速缓存控制单元。系统计时器主要用于完成系统的计时功能，可用于低功耗唤醒。矢量中断控制器用于完成中断的收集、仲裁、硬件嵌套以及与处理的交互，具体的控制和状态有相关寄存器可配。CK803 的片上缓存大小硬件可配，映射方式为 4 路组相联模式，每个缓存行的大小为 16 个字节，高速缓存控制单元中主要用于配置高速缓存的工作方式。

CK803 的指令及数据存取由内存访问单元实现，当开启高速缓存时，内存访问单元会在系统存储器和高速缓存中读取相应的指令或数据。当内存访问单元从系统存储器中访问数据，而内存保护单元使能时，会对当前访问的合法性进行检查。读取的指令将进入译码单元进行译码，并由运算单元执行。最终，指令执行完毕后，视为退休。由于内部多级流水线的存在，读取后的指令会在多个周期后执行完毕，CK803 提供相关的通用观测信号 retire_pc 和 inst_retire 来指示当前退

休指令对应的 PC。PC 指程序计数器，也表示某一指令对应的地址。

CK803 上电复位后，默认从 0 地址开始读取指令，CK803 读取到 0 地址中的数据后，会将该数据作为下一个指令读取的地址。之后，CK803 会基于当前地址，按照具体的指令内容顺序往下执行，直到遇到跳转指令，CK803 才会打断当前取指顺序，跳转到指定地址执行指令，并在新地址处继续顺序往下执行。

2.1.2 总线

微控制器中采用的 AHB 总线，是 AMBA2.0 规范中的一个部分，是一种总线接口。AHB 的信号线主要有 HCLK，HTRANS[1:0]，HADDR[31:0]，HBURST[2:0]，HWDATA，HRDATA，HREADY 等。HCLK 是主机提供的时钟信号。HTRANS[1:0]是当前的传输类型，IDLE 表示当前时刻主机占有总线，但主机并不想进行数据数据传输，地址和数据都处于无效状态；BUSY 表示当前时刻正在进行突发传输，下一次突发无法立即发生；NONSEQ 可用于一次突发中最先进行的传输，也可用来表明当前突发方式为单一传输；SEQ 表示一次突发中除首个传输以外，余下的传输是连续的，且下一次的地址与所选择的突发方式有关。HADDR[31:0]是主机提供的地址。HBURST[2:0]是突发方式，CK803 总线端暂不支持 BURST 操作，固定为 SINGLE 进行单一传输。HWDATA 和 HRDATA 分别是写数据和读数据。HREADY 可以延长数据时间，为低电平时，当前地址，数据和控制信号都需保持，直到 HREADY 信号拉高才能进行下次数据传输。图 2.3 是 AHB 基本的读写时序图：

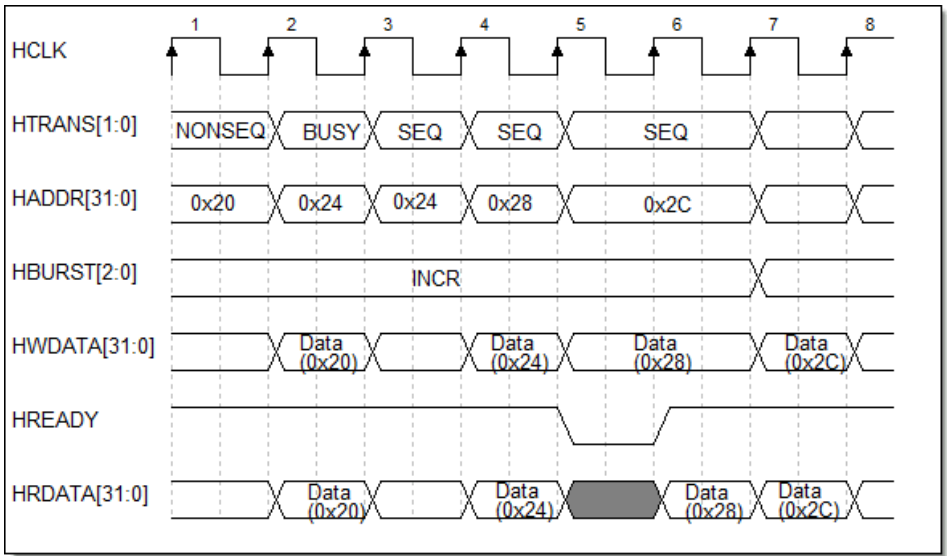


图 2.3 AHB 基本的读写时序图

（一）写操作

写操作时，在图 2.3 中 HCLK 的第一个上升沿时刻，主机将地址及控制信号发送给 AHB，这里的控制信号是指 HTRANS[1:0]，HBRUST[2:0]以及 HWRITE 等，在第二个时钟上升沿，从机采样地址和控制信息，同时主机在第三个时钟沿之前准备好 HWDATA，供从机采样。若 HREADY 为低电平，则保持当前数据、地址及控制信息不变，若为高电平，则说明此次传输成功，正常进行下次传输。地址和数据总是交叠传输，以保证高性能的操作。

（二）读操作

读操作时，在图 2.3 中 HCLK 的第一个上升沿时刻，主机将地址及控制信号发送给 AHB，在第二个时钟上升沿，从机会采样这些信息，然后发出相应的读数据，和写数据一样，读数据会在第三个时钟上升沿到来之前就准备好，AHB 会在第三个时钟上升沿采样 HRDATA。若 HREADY 为低电平，则保持当前地址和控制信号，直到 HREADY 拉高，再进行下一次的数据传输。

2.2 微控制器容错方案设计

2.2.1 软错误对微控制器的影响

随着集成电路工艺的发展，软错误已成为严重影响半导体器件可靠性的主要原因。在正常工作环境中，软错误出现的主要原因来自高能粒子，这些高能粒子

源于封装材料的放射性衰变以及宇宙辐射中最易到达地表的中子[45]，它们会穿透硅片衬底，制造额外的电子空穴对，造成电路的意外翻转，使得半导体器件失效。虽然，通过重写或复位等操作能够重置电路状态，但软错误的出现依旧会对微控制器产生严重的影响。

首先，考虑软错误出现在微控制器中的组合电路。组合电路的特点是不具有记忆性，输出信号随着输入信号的变化而即时变化。当组合电路受到干扰时，被干扰的电路会发生意外翻转；当干扰消失时，电路会恢复到原来正确的状态，表现出的故障形式类似于信号毛刺。此时，如果软错误的发生在一个周期内，并且错误状态没有被下一级寄存器采到，那么软错误不会在系统内传递。

其次，考虑软错误出现在微控制器中的时序电路。事实上，微控制器系统中的时序电路占据了芯片面积的 40-70%，包括系统中的各类寄存器，存储器和触发器等。由于时序电路具有记忆性，当时序电路发生意外翻转后，即便干扰消失，错误的状态依旧存在，并且错误将在系统中向后传递。其中，存储器中数据的意外翻转将导致处理器读取到的数据是错误的，这将导致错误的计算结果，如果该数据是指令，还可能导致处理器出现运行异常。

针对以上软错误对微控制器中的电路影响分析可知，由于错误发生位置不同，故障类型也不同。在此，可将软错误的故障类型分为以下三种：

- 屏蔽（benign）
- 挂起(hang)
- 错误结果（SDC）。

benign 类型的错误是指系统内部虽然发生错误，但是不在关键部位，并且当前的指令运行没有涉及到这部分电路，例如组合电路上发生了短暂的毛刺性翻转，或者一些无关紧要的寄存器发生翻转。通常这类错误对于系统的正常运行没有任何影响，可以不用处理。hang 类型的错误是指错误的发生可能位于系统内部的关键部位，并且错误已经严重到影响了系统的控制逻辑。这类错误的发生会导致处理器挂起，无法正常执行后面的软件程序，即便软件程序中存在故障恢复的相关程序，处理器也因为挂起而无法执行，这类错误一般需要通过硬件上的复位才能完成故障的恢复。SDC 类型的错误是指错误的发生可能位于系统内部关键部位，虽

然系统依旧可以运行，但执行结果错误。例如处理器进行一则加密运算，软错误的发生影响了内部某个数据寄存器，从而导致最终的加密结果与设想不符。

根据以上软错误对微控制器影响的分析可知，对于嵌入式处理器来说，软错误的发生会造成错误的输出结果或者直接产生运行中止；对于存储器来说，会导致存储器中的数据和代码出错，进而导致处理器的异常运行。微控制器的可靠性提高应着手以上问题进行解决。

2.2.2 基于检查点的 Lockstep 技术

由微控制器可靠性技术现状及业界产品可知，基于检查点的 Lockstep 技术是处理器容错中一种常用的方法，其技术硬件架构如图 2.4 所示。目前，常用的硬件架构有两种，图 2.4(a)中两处理器享有各自独立的存储器，并且两处理器都可以对外发起访问，对于系统架构中共用的部件，两处理器通过握手机制先后访问。图 2.4(b)中两处理器共享同一个存储器，以主从关系工作，享有相同的输入，主处理器可以对外发起访问，从处理器则不能。两图中的检查器用于实时比较两处理器接口输出数据是否相同，处理器内部状态的比较则通过软件完成。图 2.4(a)的好处在于当不需要实时对比查错时，两处理器可以解开绑定独立工作，并且由于其复制了存储单元，还能对存储器上的错误进行检测。而图 2.4(b)的好处则在于无需解决对共用部件的访问冲突，面积小，成本低。

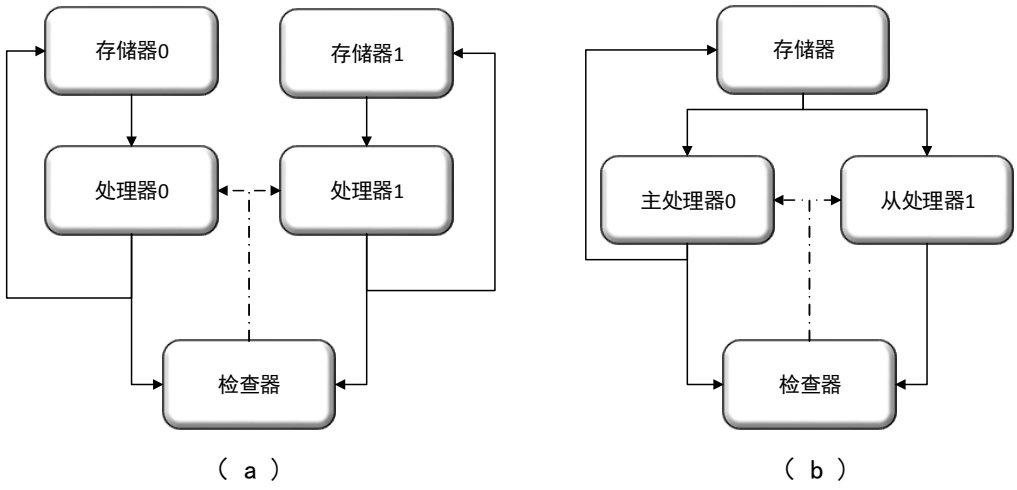


图 2.4 两种常用的 Lockstep 硬件架构图

整个容错的流程如图 2.5 所示，两处理器同时执行相同的软件程序，软件程

序中事先已设置好多个检查点。当软件执行到检查点时，检查并对比两处理器内部状态是否相同，若相同则进入状态保存中断服务程序，此刻处理器中的正确节点状态将被保存至可靠的内存中。保存的节点状态包括当前处理器内部的 PSR（Processor State Register，处理器状态寄存器），PC（Program Counter，程序计数器），GPR（General-Purpose Register，通用寄存器）等。若检查到两处理器内部状态不同，则进入状态恢复中断服务程序，将内存中保存的正确节点重写至处理器内部相关寄存器。此外，内部的硬件检查器会实时检查两处理器输出端口是否一致，当发现端口不一致时，也会进入状态恢复中断服务程序。

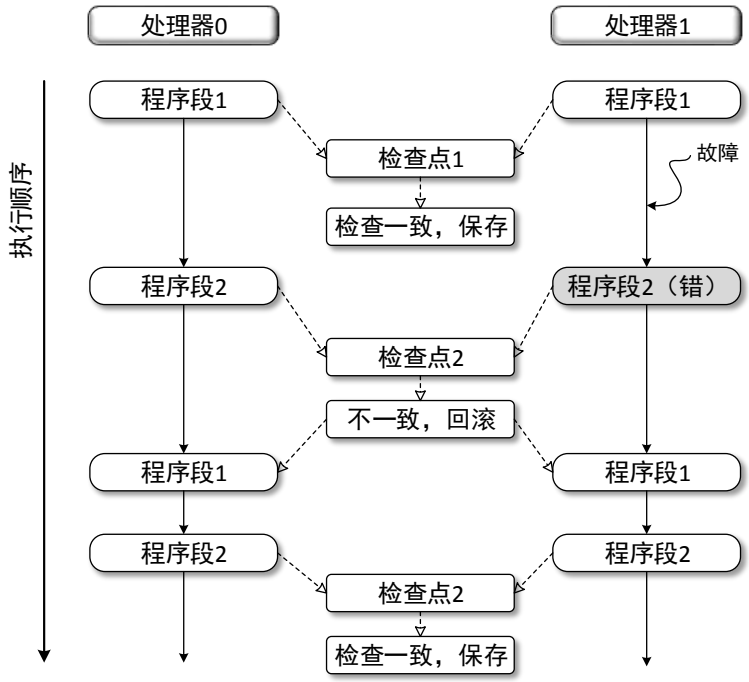


图 2.5 基于检查点的 Lockstep 技术容错流程图

2.2.3 容错方案设计

基于检查点的 Lockstep 技术通过结合软硬件实现故障的检测和恢复，使得容错方案在成本和性能间进行了平衡。然而，也存在一些不足。首先，以上方案对于 hang 类型的故障无法进行恢复，因为发生 hang 故障后，处理已经无法继续响应后面的恢复程序。其次，这类容错技术通常没有考虑到处理器内部的缓存容错，由于对处理器进行双模冗余的同时，内嵌缓存通常也进行了冗余，因此完全

可以利用这部分冗余对缓存进行有条件的容错。除此之外，根据研究背景可知，工业微控制器对于实时性的要求很高，以上的容错方式，无论是状态的保存还是恢复，都通过中断服务程序来完成，除了故障恢复响应慢以外，还需要在程序中间隔插入检查点，并不断进入状态保存中断，这将造成极大的性能开销，这也是众多高可靠微控制器设计公司宁愿采用三核 Lockstep 的原因之一。然而，面积的急剧增加带来的成本也不容忽视。因此，本研究考虑采用共用存储的低成本全硬件双核 Lockstep 架构，使用硬件实时对比状态一致性，并及时保存正确节点状态，减少状态保存带来的性能损失；使用硬件实现快速故障恢复，并解决 hang 类型故障无法恢复的问题；同时，通过硬件无效错误的缓存行解决写通工作模式下的缓存容错。在对处理器进行容错的同时，还应提高容错模块的自身可靠性。相比较嵌入式处理器，容错模块所占用的面积很小，因此可以直接使用 TMR 技术进行容错。

存储器的容错采用信息冗余。对于 ECC 技术来说，所需校正的错误数据个数越多所需要的存储空间越大，且带来的性能开销越高，难以做到单周期数据输出。事实上，由于本研究微控制器的数据位宽是 32 位，非恶意攻击情况下，出现多比特错误的概率很低。以临近空间下的辐射量计算，SRAM 的翻转概率在 1.2e-7 次/位·天[46]，那么 32 位数据中出现单比特错误和多比特错误的概率分别如下：

$$P_{mer} = 1 - ((1-p)^{32} + 32p(1-p)^{31}) \quad (2.1)$$

$$\approx 7.144\text{e-}12$$

$$P_{ser} = 32p(1-p)^{31} \quad (2.2)$$

$$\approx 3.84\text{e-}6$$

从计算结果可知，32 位数据出现多比特错误的概率远远低于出现单比特错误的概率。另外，有资料[47]显示，即便在 500mV 近阈值电压下，存储器中出现多比特错误的概率依然很低，仅有 4%。因此，对于 32 位数据位宽的存储器来说，在非恶意攻击情况下，采用 ECC 技术进行单比特纠正，已经可以大大增加

数据的可靠性。然而面对恶意攻击，存储中的数据翻转可能达到 10bit/32bit[48]，对于这种情况，简单的错误纠正技术很难满足需求，因此增加错误检测技术保证在多位数据出错时报警，以供系统及时进行下一步操作应对恶意攻击。

综上所述，本研究基于检查点的 Lockstep 架构，将软件状态保存恢复改为硬件实现，从而设计了基于全硬件的 Lockstep 容错模块，并对容错模块本身进行 TMR 可靠性处理。同时，采用单比特错误纠正码和多比特错误检测码对存储器进行容错。总体的容错方案如图 2.6 所示。

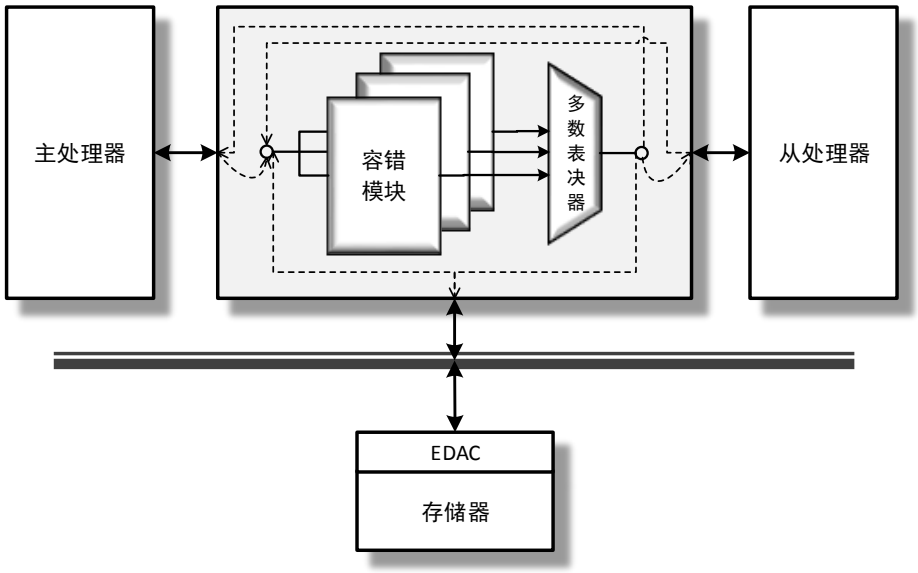


图 2.6 微控制整体容错方案框图

2.3 嵌入式处理器抗功耗分析攻击设计

功耗分析攻击主要包括简单功耗分析（Simple Power Analysis，SPA）和差分功耗分析，SPA 攻击的目标是仅通过少量的功耗曲线揭示出密钥或者与之相关的敏感信息，SPA 一般要求攻击者掌握攻击设备的设计细节或代码细节。DPA 攻击者无须了解关于被攻击设备的详细知识，它通过记录密码设备对大量不同数据分组进行加密或解密操作时的功耗曲线，并基于这些信息恢复出设备中的密钥。在实际的攻击测试中，DPA 的使用更为广泛，是最流行的功耗分析攻击方式。因此，本研究以 DPA 攻击方式作为研究对象，分析其原理并制定相应的抗攻击方案。

2.3.1 差分功耗分析攻击原理

DPA 主要分析固定时刻的功耗消耗与被处理数据之间的依赖关系。因此，DPA 攻击仅关注功耗曲线的数据依赖性。其具体的攻击步骤如下：

第一步：选择所执行算法的某个中间值。DPA 攻击的第一步是选择被攻击设备所执行密码算法的一个中间值，这个中间值必须是一个函数f(d, k)，其中，d 是已知的非常量数据，而 k 是密钥的一小部分。一般来说，d 可以直接是明文或是明文已经经过几轮计算后得到的中间密文。

第二步：测量功耗消耗。DPA 攻击的第二步是测量密码设备在加密 D 个不同明文时的功耗消耗。这 D 个明文可以直接或间接得到上一步中计算中间值所需的 d。将这些已知数值记作向量$\mathbf{d} = (d_1, \dots, d_D)$，其中，$d_i$ 表示第 i 次加密操作对应的已知非常量数据。在每一次设备运行期间，攻击者都会记录一条功耗曲线。对应于数据d_i 的功耗曲线记作$t_i' = (t_{i,1}, \dots, t_{i,T})$，其中，T 表该功耗曲线的长度。对于 D 个明文，攻击者分别测量一条功耗曲线，因此这些功耗曲线可以使用一个D × T 的矩阵 **T** 来表示。其中，要求矩阵 **T** 中的每一列$\boldsymbol{t}_j$ 的功耗消耗值必须是由相同的操作引起，即要求 D 条功耗曲线在时间轴上对齐。其中，对于上一步中所选中的中间值，其在功耗曲线中的位置，记作 ct。

第三步：计算假设中间值。DPA 攻击的第三步是对于每一个可能的 k 值，即对每一个可能的密钥的一小部分，计算对应的中间值。假设 k 共有 K 中可能，则将这些可能的一小段密钥记为向量$\mathbf{k} = (k_1, \dots, k_K)$。这样，根据已知向量 d 和可能的一段密钥 k，根据它们间的函数式 f，可以计算出中间值f(d, k)，其计算结果可写成一个矩阵大小为D × K 的矩阵 **V**。那么每一列都是一个可能的k_j对应的不同明文中间值。然而，设备中的真实密钥只是向量 **k** 中的其中一个元素，将该真实密钥的索引记作 ck，则该真实密钥为k_{ck}。

第四步：将中间值映射为功耗消耗值。DPA 攻击的第四步攻击是将上一步得到的中间值矩阵 **V** 映射为对应的功耗消耗值，具体的映射模型包括汉明重量或汉明距离模型等，这样获得的假设功耗消耗值矩阵记为 **H**。

第五步：比较假设功耗消耗值和实际的功耗曲线。DPA 攻击的最后一步是将矩阵 **H** 的每一列h_i 和矩阵 **T** 的每一列t_j 进行比较。为了确定数据间的线性关系，常用的方法是相关系数法，计算数据间的相关系数，最大相关系数对应的假

设密钥即为正确密钥。

2.3.2 功耗隐藏技术研究

掩码技术的实现通常需要对具体的运算单元进行重设计，因此，功耗隐藏技术相对来说更适合在硬件系统层面实现。因此本研究采用功耗隐藏技术实现处理器的抗攻击设计。如 1.2 章节所述，在功耗隐藏技术中，分为时间维度的功耗隐藏和振幅维度的功耗隐藏。以下是对两种隐藏技术效果量化的结果[49]。

对于振幅维度的功耗隐藏来说，首先，对一条功耗曲线上的某一点的总功耗组成进行分析，其功耗模型可由式 2.3 表示：

$$P_{total} = P_{exp} + P_{sw.noise} + P_{el.noise} + P_{const} \tag{2.3}$$

其中，P_{exp} 表示对攻击者有用的功耗，$P_{sw.noise}$ 表示转换噪声，指系统中攻击者不关心的其他部件工作带来的功耗，$P_{el.noise}$ 表示电子噪声，P_{const} 表示噪声的常量部分，这几个功耗组成相互独立。可进一步用信噪比的概念，对所需功耗曲线进行数量估算。在功耗分析攻击中，信号对应于P_{exp}，噪声分量由$P_{sw.noise}$ 和 $P_{el.noise}$ 的和给出，因此，其信噪比公式如式 2.4 所示：

$$\mathrm{SNR} = \frac{Var(P_{exp})}{Var(P_{sw.noise} + P_{el.noise})} \tag{2.4}$$

其中，Var 表示方差。则在 j 处，攻击者假设的功耗消耗值与设备实际的功耗消耗值间的相关性可用$\rho(H_j, P_{total})$ 表示，该相关性越高，所需的功耗迹越少。将式 2.3 和式 2.4 带入$\rho(H_j, P_{total})$ 的计算，化简可得式 2.5 的结果。

$$\begin{aligned}
\rho(H_j, P_{total}) &= \rho(H_j, P_{exp} + P_{sw.noise} + P_{el.noise} + P_{const}) \\
&= \rho(H_j, P_{exp} + P_{sw.noise} + P_{el.noise}) \\
&= \rho(H_j, P_{exp} + P_{noise}) \\
&= \frac{E\left(H_j \cdot (P_{exp} + P_{noise})\right) - \mathrm{E}(H_j) \cdot E(P_{exp} + P_{noise})}{\sqrt{Var(H_j) \cdot (\mathrm{Var}(P_{exp}) + Var(P_{noise}))}}
\end{aligned}$$

$$= \frac{E\left(H_j \cdot \left(P_{exp} + P_{noise}\right)\right) - \mathrm{E}(H_j) \cdot E(P_{exp} + P_{noise})}{\sqrt{Var(H_j)} \cdot \mathrm{Var}\left(P_{exp}\right)\sqrt{1 + \frac{Var(P_{noise})}{Var(P_{exp})}}}$$

$$= \frac{\rho(H_j, P_{exp})}{\sqrt{1 + \frac{1}{SNR}}} \qquad (2.5)$$

根据实施一次成功的 DPA 攻击所需要的功耗曲线数量 n 的经验法则，当$\left|\rho_{ck,ct}\right| \leq 0.2$且系统中较小的信噪比时，功耗曲线数量 n 与在中间结果攻击处的假设功耗消耗值与实际功耗消耗值间的相关系数$\rho_{ck,ct}$及信噪比 SNR 间的关系如式 2.6 所示，因此当信噪比减小一半时，所需要的功耗迹数量将增加 2 倍。

$$\mathrm{n} \approx \frac{28}{\rho_{\mathrm{ck,ct}}^2} \sim \frac{1}{SNR} \qquad (2.6)$$

对于时间维度功耗隐藏来说，由于随机延迟或乱序操作等原因，在各个功耗曲线中，对被攻击中间结果进行处理的位置有所不同。假设该位置为 ct，随机延迟通常会导致 ct 服从二项分布或均匀分布。用$\hat{p}$表示该分布中最高频率。同时，将此处对应的功耗消耗表示为$\hat{P}_{total}$，其具有如下的特性：$\hat{P}_{total}$以概率$\hat{p}$对应于被攻击中间结果的功耗消耗，即$\hat{P}_{total} = P_{total}$ 的概率为$\hat{p}$ ，而$\hat{P}_{total}$ 对应其他操作功耗消耗的概率为（1-$\hat{p}$ ）。使用P_{other} 来表示其他操作的功耗消耗，协方差 $\mathrm{Cov}(H_{ck}, \hat{P}_{total})$ 可用式 2.7 计算：

$$Cov\left(H_{ck}, \hat{P}_{total}\right) = \hat{p} \cdot Cov(H_{ck}, P_{total}) + (1 - \hat{p}) \cdot Cov(H_{ck}, P_{other}) \qquad (2.7)$$

由于$\hat{p}$为最大概率，所以在未受保护密码设备的 DPA 攻击中，H_{ck}和$\hat{P}_{total}$的相关性导致最高的相关系数。因此，可以使用相关系数$\rho\left(H_{ck}, \hat{P}_{total}\right)$ 来确定实施攻击所需要的功耗曲线数量，该数值可以基于$\rho(H_{ck}, P_{total})$ 计算，如式 2.8 所示：

$$\rho\left(H_{ck}, \hat{P}_{total}\right) = \frac{\hat{p} \cdot Cov(H_{ck}, P_{total}) + (1 - \hat{p}) \cdot Cov(H_{ck}, P_{other})}{\sqrt{Var(H_{ck}) \cdot Var(\hat{P}_{total})}}$$

$$= \frac{\hat{p} \cdot Cov(H_{ck}, P_{total})}{\sqrt{Var(H_{ck}) \cdot Var(\hat{P}_{total})}}$$

$$= \rho(H_{ck}, P_{total}) \cdot \hat{p} \cdot \sqrt{\frac{Var(P_{total})}{Var(\hat{P}_{total})}} \qquad (2.8)$$

上式中，$\rho(H_{ck}, P_{total})$确定了未受保护的密码设备中所需要的功耗迹。随机延时

或乱序操作的效果主要依赖于概率$\hat{p}$，而$\hat{p}$则会线性降低相关系数$\rho(H_{ck}, \hat{P}_{total})$。同样，根据式 2.6 所示关系式，若$\hat{p}$减半则相关系数减半，从而导致需要 4 倍数量的功耗迹。

2.3.3 抗功耗分析攻击方案设计

功耗隐藏技术的量化结果表明，对振幅维度隐藏来说，信噪比越低，所需的功耗曲线越多，带来的攻击难度越大。然而，信噪比的降低往往是通过增加系统噪声实现的，通常是增加额外的无用功耗，这在一定程度上增加了系统的整体功耗。对时间维度隐藏来说，运算过程的执行越随机，$\hat{p}$越小，所需的功耗曲线也就越多。以随机延时为例，随机延时插入的随机性越大，抵御攻击的效果越好。然而，随机性大的要求是延时范围大，显然，这会使得指令的执行速度大大降低，造成较大的性能开销。

考虑到以上问题，结合本研究所采用的双核容错方案，可以考虑对双核分别进行不同的随机延时插入，通过双核间的异步执行，同时进行振幅和时间维度上的功耗隐藏，平衡功耗消耗与性能开销。由于两处理器插入了随机延时，导致处理器运行不再同步，容错方案中实时对比状态一致性将导致故障产生的误判。为了保证抗功耗分析攻击的同时，容错功能依旧有效，需要修改故障检测方式。由于处理器在进行容错设计后可以对故障进行恢复，无论处理器内部是否出错，只要没有将错误传播至系统中，都可以进行故障恢复。因此，在功耗隐藏模式开启下，可以将状态的实时对比更改为在处理器写操作时对比。整体抗攻击方案及预期效果如图 2.7 所示。

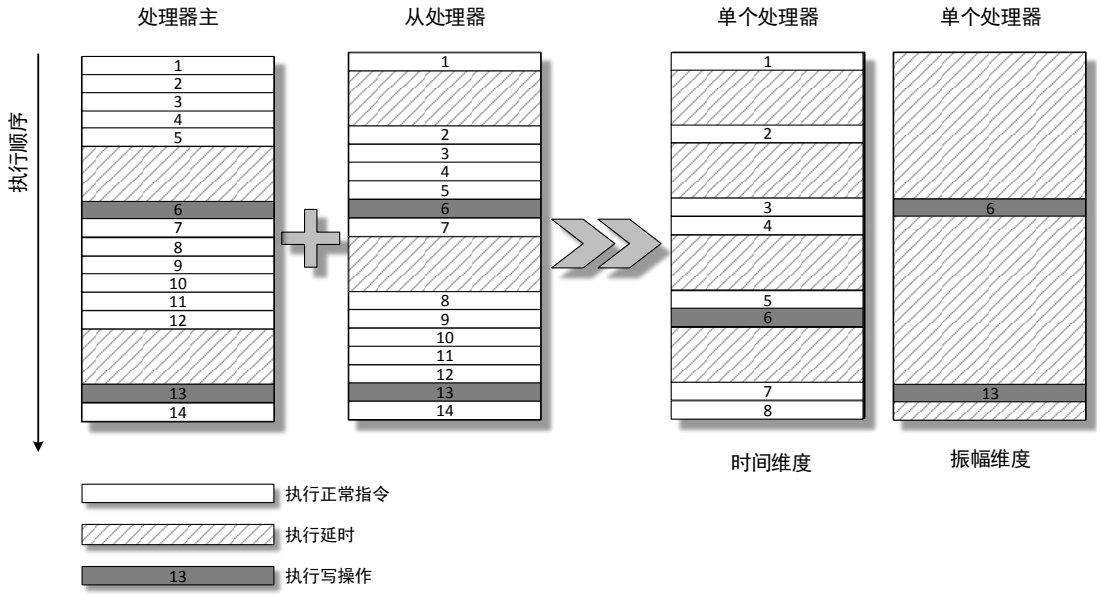


图 2.7 处理器抗攻击方案及预期效果示意图

图中左半部分是双核系统主处理器和从处理器执行指令运算。功耗隐藏模式开启后，分别在主处理器和从处理器中两处位置插入了延时，只从时间维度上看，若攻击者进行 DPA 攻击，则需要将插入的延时移除，将处理器指令的运行在时间轴上对齐。在这一段时间内共需要移除 4 段干扰，对应在单核处理器中的效果如右图所示。相比较双核系统，单个处理器因为插入了 4 段延时，程序执行效率大大降低，在这段时间内只执行到指令 8，而双核系统执行到指令 14。只从振幅维度上看，由于双核间的随机延迟插入，导致同一时刻，两处理器中执行的指令随机，这会造成在功耗振幅上的随机。对应在单核处理器中的效果如右图所示，除了写操作时刻上的功耗一定，其他时刻的功耗随机。虽然写操作的功耗没有在振幅上得到隐藏，但由于随机延迟的插入其在时间轴上依旧是随机的。

2.4 本章小结

本章首先分析了软错误对微控制器的影响，以及软错误在微控制器中表现出的故障形式。介绍了基于检查点的 Lockstep 技术，分析了其在执行效率，hang 类型故障无法恢复，以及不考虑缓存容错等方面的不足，提出了改进后的处理器容错方案。其次，分析了单比特错误和多比特错误在存储器中的出现概率，并据此选择纠一检多作为存储器的容错方案。最后，介绍了 DPA 的攻击原理，分析了

功耗隐藏技术效果的量化结果，并基于双核容错架构，设计了嵌入式处理器的抗功耗分析攻击方案，使得设计能够在容错的同时，在时间维度和振幅维度上进行功耗隐藏，增加微控制器的攻击成本。

3 硬件结构设计与实现

3.1 基于全硬件的 Lockstep 容错设计

基于全硬件的 Lockstep 容错模块主要包括三个部分：故障检测、故障恢复和故障隔离。单个容错模块的结构框图如图 3.1 所示。

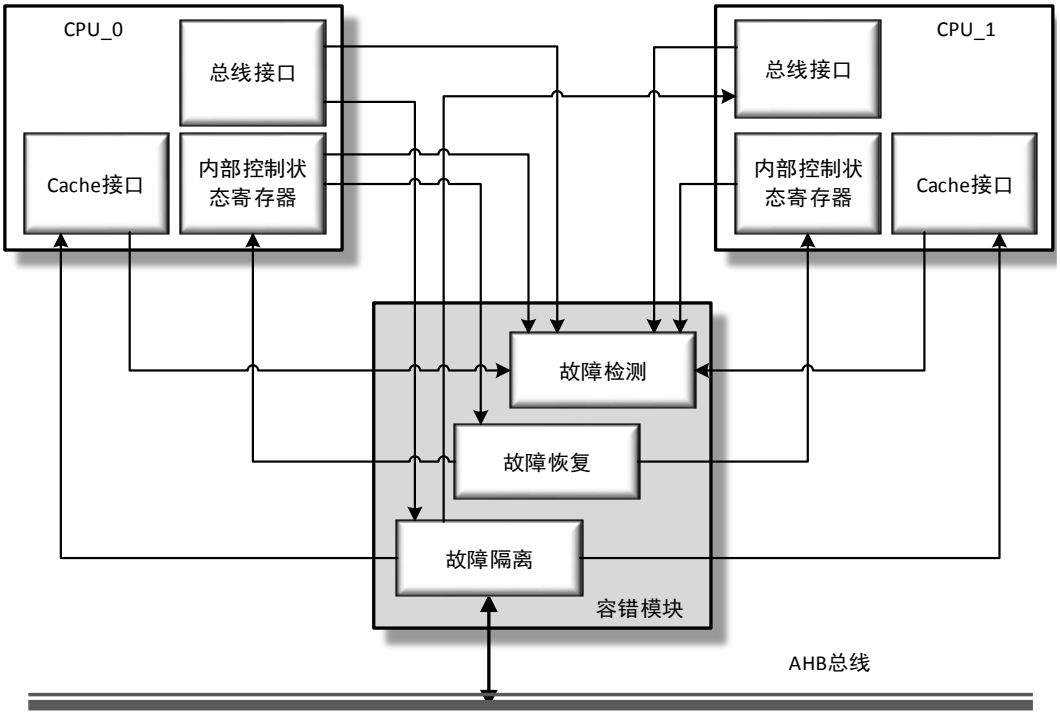


图 3.1 单个容错模块的结构框图

图中故障检测模块实时对比主从处理器中的内部控制状态寄存器，总线接口，以及 Cache 接口，一旦对比不一致，立即产生错误报警信号。故障恢复模块实时保存处理器内部控制状态寄存器中的值，一旦检测到错误报警信号，故障恢复模块将立即使处理器返回上一个正确节点，并将保存的状态重新置入到处理器内部控制状态寄存器中。处理器对外的总线接口信号将通过故障隔离模块后输出到系统中，一旦故障隔离模块检测到错误报警信号，将及时隔离总线上的错误写操作，并对已写入 Cache 的错误进行恢复。

3.1.1 处理器故障检测设计

故障检测是通过硬连线将主从处理器内部相关信号拉出并作对比。首先，是内部的控制状态寄存器信号。由于内部的寄存器信号除了用作故障检测，还将用于故障的恢复。因此，需要仔细分析处理器内部的寄存器结构，选择要比较和保存的寄存器。通过阅读 CK803 的用户手册可知，CK803 内部的控制及状态寄存器主要包括通用寄存器，控制寄存器和紧耦合 IP 相关的寄存器，如表 3.1 所示。

表 3.1 CK803 应对比及保存的内部寄存器列表

寄存器类别	寄存器名称	寄存器说明
通用寄存器	R0~R15，R28	通用寄存器 0~15 及通用寄存器 28
	USR SP	普通用户模式下的堆栈指针寄存器
	SPV SP	超级用户模式下的堆栈指针寄存器
控制寄存器	PSR	处理器状态寄存器
	PC	程序计数器
	VBR	向量基址寄存器
	EPSR	异常保留处理器状态寄存器
	EPC	异常保留程序计数器
	CHR	隐式操作寄存器
	CCR	高速缓存配置寄存器
	CAPR	可高缓和访问权限配置寄存器
	PACR	保护区控制寄存器
	PRSR	保护区选择寄存器
紧耦合 IP 片内高速缓存控制寄存器单元	CER	高速缓存使能寄存器
	CIR	高速缓存无效寄存器
	CRCR0~CRCR3	0~3 号可高缓区配置寄存器
	CPFCR	高速缓存性能分析控制寄存器
	CPFATR	高速缓存访问次数寄存器
	CPFMTR	高速缓存缺失次数寄存器

紧耦合 IP 矢量中断控制器	VIC_ISER	中断使能设置寄存器
	VIC_IWER	低功耗唤醒设置寄存器
	VIC_ICER	中断使能清除寄存器
	VIC_IWDR	低功耗唤醒清除寄存器
	VIC_ISPR	中断等待设置寄存器
	VIC_ICPR	中断等待清除寄存器
	VIC_IABR	中断响应状态寄存器
	VIC_IPR0~VIC_IPR7	中断优先级设置寄存器
	VIC_ISR	中断状态寄存器
	VIC_IPTR	中断优先级阈值寄存器
紧耦合 IP 系统计时器	CORET_CSR	计时器控制状态寄存器
	CORET_RVR	计时器回填值寄存器
	CORET_CVR	计时器当前值寄存器
	CORET_CALIB	计时器校准寄存器

以上寄存器信号需要在处理器 RTL 代码中找到相应的信号路径，并伸出至处理器外部，以供故障检测模块进行错误检测。由于处理器总线接口上出现软错误会导致故障蔓延至系统内，因此，处理器总线接口中的部分输出信号也需要进行对比以及时发现错误并做隔离。CK803 的应对比的外部接口信号如表 3.2 所示：

表 3.2 CK803 应对比的外部接口信号列表

寄存器类别	寄存器名称	寄存器说明
系统总线接口	HADDR	系统总线地址信号
	HBURST	系统总线突发传输指示信号
	HPROT	系统总线保护控制信号
	HSIZE	系统总线传输宽度指示信号
	HTRANS	系统总线传输类型指示信号
	HWDATA	系统总线写数据信号
	HWRITE	系统总线读写指示信号

指令总线接口	IBHADDR	指令总线地址信号
	IBHBURST	指令总线突发传输指示信号
	IBHPROT	指令总线保护控制信号
	IBHSIZE	指令总线传输宽度指示信号
	IBHTRANS	指令总线传输类型指示信号
	IBHWDATA	指令总线写数据信号
	IBHWRITE	指令总线读写指示信号
数据总线接口	DBHADDR	数据总线地址信号
	DBHBURST	数据总线突发传输指示信号
	DBHPROT	数据总线保护控制信号
	DBHSIZE	数据总线传输宽度指示信号
	DBHTRANS	数据总线传输类型指示信号
	DBHWDATA	数据总线写数据信号
	DBHWRITE	数据总线读写指示信号
其他	IU_PAD_INST_RETIRE	指令退休指示信号
	IU_PAD_RETIRE_PC	当前退休指令的 PC
	CTIM_PAD_INT_VLD	系统计时器中断信号

除了以上信号外，由于 CK803 内部包含紧耦合的高速缓存，一方面，如果处理器的错误来源于缓存，仅仅通过处理器的状态恢复重执行，并不能解决实际的故障；另一方面，当处理器进行故障恢复回滚到上一个状态时，缓存中的数据内容也必须回到上一个状态，保持数据的一致性。因此，为了对写通模式下的高速缓存进行以上的容错设计，需要对缓存接口上的输入输出信号进行监控对比。写通模式下涉及到的缓存存储区主要包括两种类型：数据存储区和标记存储区。本研究所使用的 CK803 配置了 4KB 容量的高速缓存，内部由 4 块数据存储宏单元和 1 块标记存储宏单元组成。所需相关的信号如表 3.3 所示：

表 3.3 CK803 应对比的缓存接口信号列表

寄存器类别	寄存器名称	寄存器说明
数据存储区	CACHE_DATA_ADDR	数据访问地址信号
	CACHE_DATA_WEN	数据读写指示信号
	CACHE_DATA_DIN	数据写数据信号
	CACHE_DATA0_CEN	数据存储区访问使能
	CACHE_DATA0_DOUT	数据存储区读数据信号
	CACHE_DATA1_CEN	数据存储区访问使能
	CACHE_DATA1_CEN	数据存储区读数据信号
	CACHE_DATA2_CEN	数据存储区访问使能
	CACHE_DATA2_DOUT	数据存储区读数据信号
	CACHE_DATA3_CEN	数据存储区访问使能
	CACHE_DATA3_CEN	数据存储区读数据信号
标记存储区	CACHE_TAG_ADDR	标记访问地址信号
	CACHE_TAG_WEN	标记读写指示信号
	CACHE_TAG_DIN	标记写数据信号
	CACHE_TAG_CEN	标记存储区访问使能
	CACHE_TAG_DOUT	标记存储区读数据信号

故障检测模块获得以上信号后，可以通过比较操作进行故障的检测，一旦存在对比结果的不一致，则拉高 lc_error_raw 信号。然而，由于故障的产生是随机的。以上信号的不一致可能出现在任意时刻，因此，lc_error_raw 信号实际上应当做异步信号处理，不能直接用于后续的电路设计中，否则可能因亚稳态的出现，将不确定状态传播下去。如图 3.2 所示，lc_error_raw 可通过两级同步，作为后续故障隔离及恢复的信号。

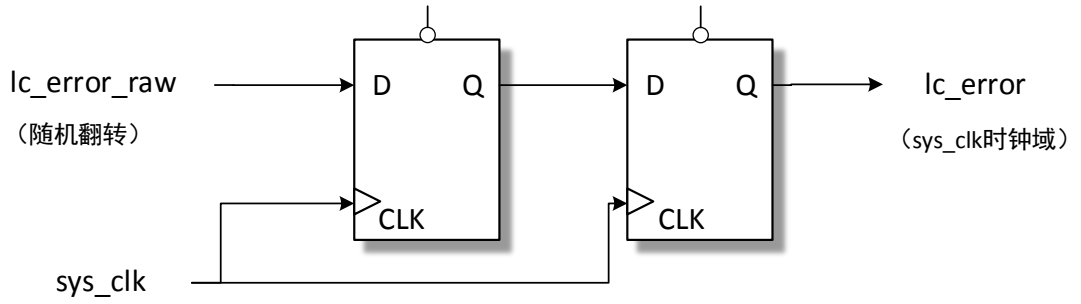


图 3.2 lc_error_raw 的同步电路示意图

3.1.2 处理器故障恢复设计

当 lc_error_raw 信号同步至 lc_error 上时，故障恢复模块将对当前主从处理器进行故障恢复。故障恢复主要分为两个部分，首先，需要将一个正确节点上的处理器状态信息保存至回滚缓冲区，这个正确节点，指的是在此之前，两处理器正常运行，没有因软错误发生而导致两处理器出现状态不一致时的某个执行点，而状态信息指的是处理器内部的寄存器值。其次，将回滚缓冲区中保存的正确节点上的状态信息置入主从处理器，使主从处理器从该节点重新执行指令。

为了对系统进行最快的故障恢复，正确节点选择出现状态不一致时的上一个指令周期。如图 3.3 所示，PC1 指令退休后，内部寄存器状态跳转为 S1，PC2 指令退休后，内部寄存器状态跳转为 S2。当执行 PC3 时检测到主从处理器失步，发生故障。此时，在 lc_error_raw 拉高前的一个时钟上升沿保存下来的正确节点状态为 PC2 和 S1，处理器故障恢复时应跳转至该状态。由于容错模块可用的故障检测信号 lc_error 相对实际检测到异常时延迟了两拍，而每一次指令正确退休后，当前处理器状态都将得到保存。所以，当 lc_error 拉高需要进行故障恢复时，回滚缓冲区中的内容已在第 6 个时钟上升沿更新为 PC3 和 S2。故障恢复时，处理器会跳转到已经发生错误的 PC3 和 S2 状态，这与预期结果并不相符。为了解决这一问题，可以对当前的处理器状态信息 inner_reg，及指令退休信号延迟两个周期，并用延迟后的状态信息 inner_reg_dff2，作为正确节点信息保存的来源。此时，当两个周期后 lc_error 拉高，需要进行故障恢复时，回滚缓冲区中所保存的正确节点仍为 PC2 和 S1，与实际应跳转状态一致。

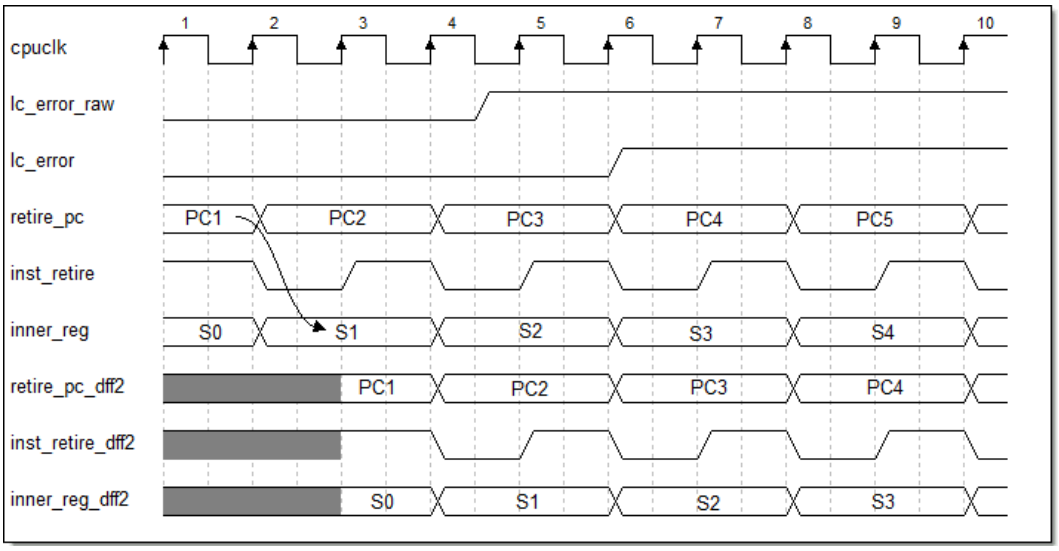


图 3.3 正确节点的状态保存时序图

处理器回滚需要将处理器内部的控制状态寄存器重置，并将程序计数器 PC 设置为当前保存的正确节点中的对应值。对如何在硬件上将 CK803 中的程序计数器设置为指定 PC，有以下几种方案：

（1） 在 CK803 的指令跳转逻辑中更改 PC 源。这种方法需要对处理器内部逻辑进行较多探索，并且需要在跳转前进行 flush 操作，冲刷掉当前流水线中的其他操作，否则会破坏处理器内部的时序，无法正常工作。此外，这种跳转过程不可控，主从处理器一旦失步，难以再同步。

（2） 可在需要进行 PC 回滚时，硬件插入跳转指令。这种方法跳转周期长，需要将近 20 个时钟周期才能跳转到指定 PC 下执行程序。并且，同前一种方法一样，这种跳转的过程不可控，两处理器一旦失步，难以再同步。

（3） 在出现故障后，可先对两处理器进行硬件复位，该信号经异步复位同步释放电路处理后释放，复位后的 CK803 会在复位信号释放后的下一个周期重新启动，从 0 地址取指。此时，需要在总线上将从 0 地址中读到的数据，换成当前正确节点中保存的 PC 值。CK803 会将这个数据作为下一条指令对应的取指地址。处理器可以在 12 个周期内跳转到该地址取指重新执行。该方法跳转过程固定，可控，能够方便的完成对失步处理器的再同步。并且由于采用了硬件复位，即便处理器发生了 hang 类型错误也能够进行顺利的恢复。

因此，选择方案三作为程序计数器回滚方法。处理器复位后，内部的寄存器状态信息也将被复位。此时，需要在处理器跳转到指定 PC 执行程序前，恢复内

部的寄存器状态。状态信息重置的电路结构如图 3.4 所示。将保存的正确节点上的状态信息作为多路复用器的输入，并用 lc_error_pulse2 作为选择信号。lc_error_pulse2 会在 lc_error 拉高后，在处理器从 0 地址读取到指定 PC 后，产生一个位宽的高电平脉冲，寄存器的状态将在此刻被重置。

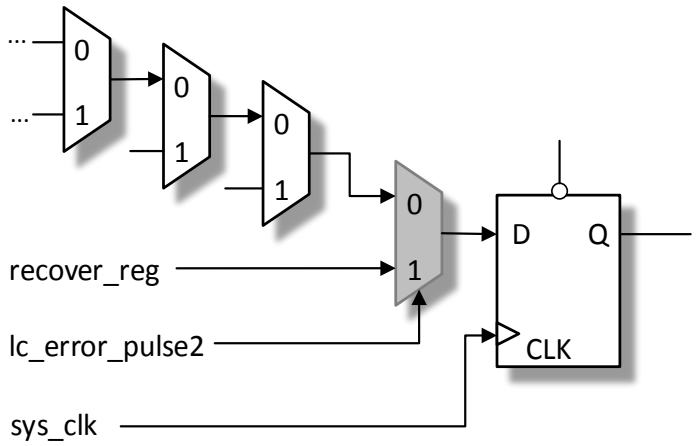


图 3.4 状态信息重置电路结构图

3.1.3 处理器故障隔离设计

故障恢复可以回滚主从处理器的状态，但无法回滚处理器外部的状态，这里的外部状态指的是外部的存储器，外围接口或系统 IP，以及处理器内部的缓存状态。考虑以下这种情况，如图 3.5 所示，假设指令 2 是对存储器的 A 地址进行读操作，读出结果为 old_a。指令 3 是对存储器的 A 地址进行写操作，写入的值为 new_a。指令 4 是比较操作，比较对象是 old_a 和 new_a。若指令 3 退休周期发生故障，根据前文可知，此时回滚缓冲区中保存的跳转 PC 是指令 2，处理器故障恢复后会跳转到指令 2 处重新执行。由于 lc_error 的迟滞性，故障恢复的相关操作会在指令 4 期间进行，而指令 3 对应的写操作会被正常执行。因此，当处理器复位后跳转到指令 2 处再次对 A 地址进行读取时，读到的值不再是 old_a，而是上次写操作中写入的 new_a。这将导致后面指令 4 的比较操作产生错误。这种情况不仅会发生在存储器中，同样会发生在缓存以及外围接口，系统 IP 的配置中。这主要是因为在故障发生时，没有及时地进行故障隔离，导致外部状态超前处理器回滚后的状态。因此，必须将故障隔离考虑到容错设计中。

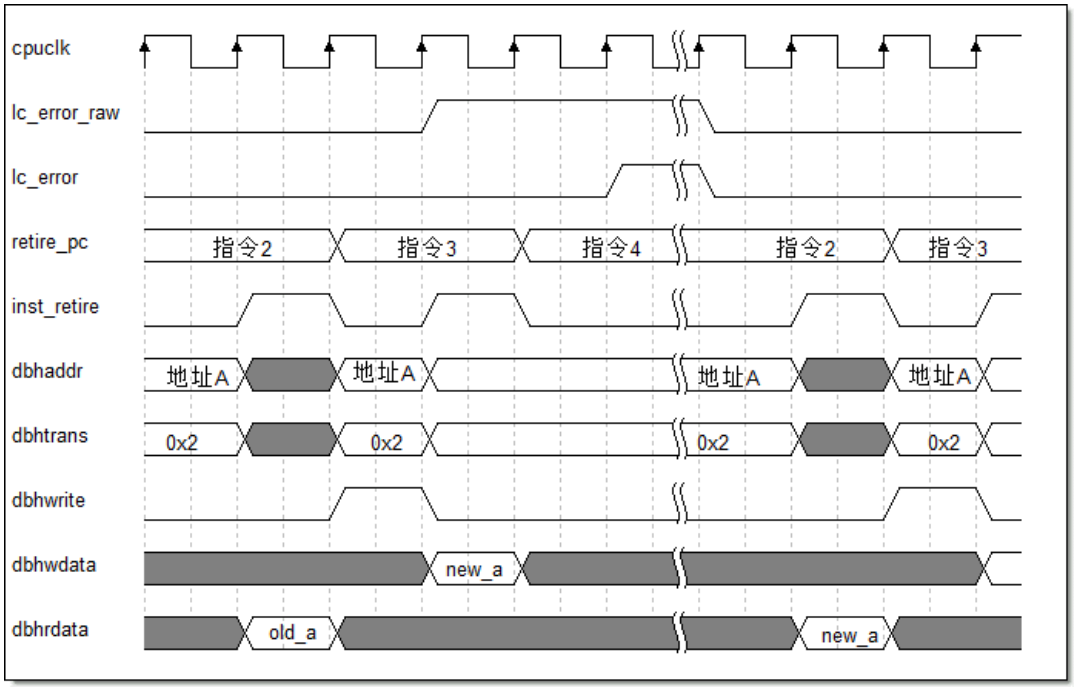


图 3.5 处理器状态回滚而外部状态无法回滚造成的故障时序图

由于存储器，外围接口，以及缓存在功能和需求方式上有所不同，故障隔离模块将分为三个部分分别进行设计。

（一）存储器

本文所研究的微控制器，其对于存储器的写操作，主要是在数据总线上进行。为了进行故障隔离，需要对数据总线的写操作进行修改。对于存储器来说，写操作中的数据是否被真实写入并不重要，重要的是当处理器再次访问该地址时，能够得到之前写入的值。针对这种特性，存储器的故障隔离可以通过建立一个写操作缓冲区完成。

如图 3.6 所示，写操作缓冲区主要由四个缓冲区组成，分别为写地址缓冲区，写数据缓冲区，PC 缓冲区，以及故障 PC 缓冲区。写地址缓冲区保存的是每次写操作对应的写地址，写数据缓冲区保存的是每次写操作对应的写数据，PC 缓冲区保存的是每次写操作对应的当前退休指令的 PC，故障 PC 缓冲区保存的是发生故障后至处理器被复位这段时间内执行过的指令的 PC。由于处理器发生故障后，最多可再执行两次写操作，即最多有不超过三个错误的写操作需要被隔离，因此每个缓冲区由 3 个寄存器组成。

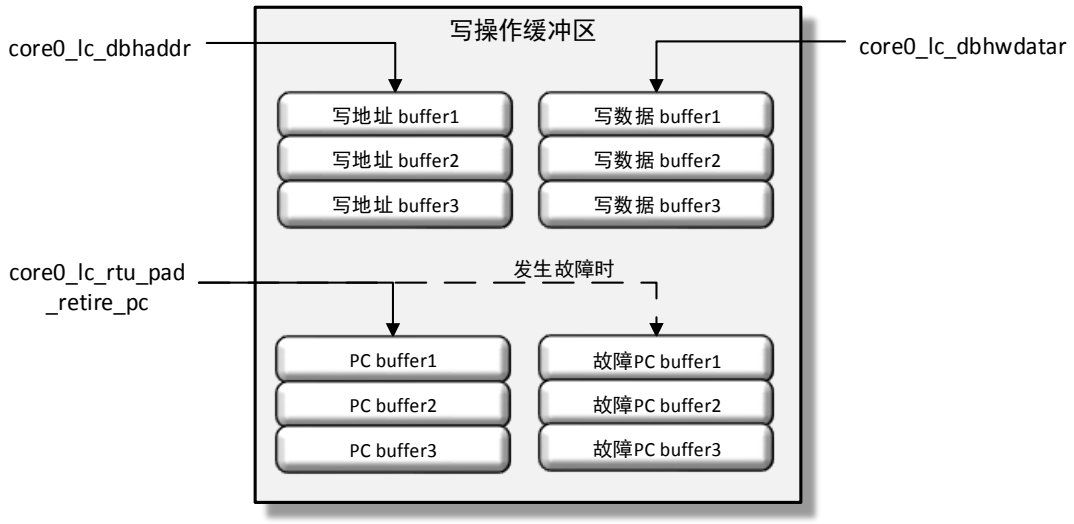


图 3.6 写操作缓冲区结构图

当处理器正常运行，没有发生故障，而写操作缓冲区未满时，相关的写操作时序修改如图 3.7 所示。图中主处理器数据总线 core0_lc_dbus 上发起了写操作 1，此时，写操作缓冲区将写操作 1 对应的地址，数据及其对应的 PC 保存到对应的缓冲区 WR_BUF 中，而经过故障隔离模块后的数据总线 lc_sys_dbus 实际上没有发起任何操作。此后的写操作 2，写操作 3 依次被保存在写操作缓冲区中。当发起写操作 4 时，由于缓冲区内的写操作已满，保存写操作 4 的同时最早存入的写操作 1 将被冲出，lc_sys_dbus 上实际执行的是写操作 1。此时，写操作 1 的实际执行时间至少延迟了 3 个周期，若此刻 lc_error 仍保持为低，说明 core0_lc_dbus 执行写操作 1 时，处理器运行正常，该写操作可靠，能够保证进行故障恢复时，回滚节点在当前写操作之后。

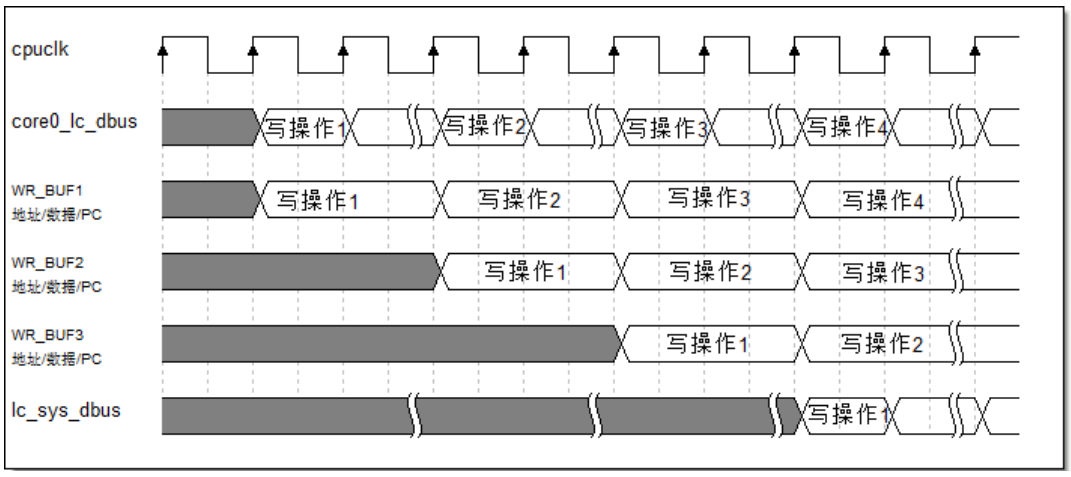


图 3.7 数据总线的写操作修改时序图

当处理器正常运行，没有发生故障，处理器发起读操作时，相关的读操作时序修改如图 3.8 所示。图中主处理器数据总线 core0_lc_dbus 上发起了读操作 1，读操作 2，读操作 3 和读操作 4，故障隔离模块不经修改的将这些读操作传递至对系统输出的数据总线 lc_sys_dbus 上。假设以上读操作的地址与之前的写操作 1，写操作 2，写操作 3 和写操作 4 一一对应。那么，通过操作地址的比较可知，读操作 1 在写操作缓冲区中没有对应的数据存储，因此，读数据为 lc_sys_dbus 上获得的读数据。读操作 2 在写操作缓冲区中有对应的数据存储，因此，读数据为写操作缓冲区中写操作 2 对应的写数据。读操作 3 和读操作 4 同理，其读数据分别为写操作 3 和写操作 4 的对应的写数据。如果发起的读操作与写操作缓冲区中的多个写操作地址相同，则读数据来自缓冲区中最近一次保存的写操作数据。

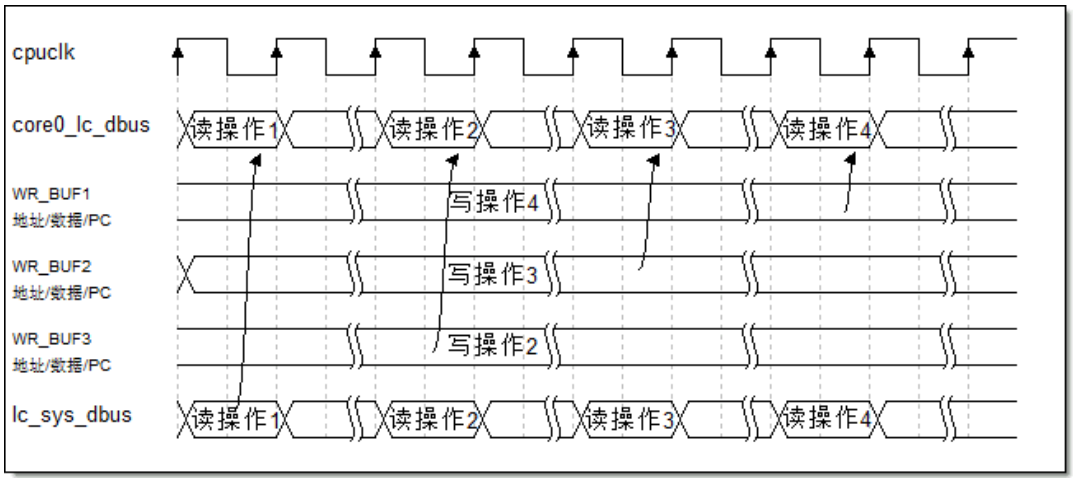


图 3.8 数据总线的读操作修改时序图

当处理器运行出现故障时，故障发生后存储器的状态回滚如图 3.9 所示。图中，执行 PC6 时出现了故障。当 lc_error_latch 拉高后，写操作缓冲区会将延迟两拍后的 PC 值保存至故障 PC 缓冲区中，这里保存的故障 PC 为 PC6 和 PC7。在处理器的故障恢复期间，将故障 PC 与缓冲区中保存的写操作对应 PC 进行比较，如果比较一致且故障 PC 不为 0，则将 PC 对应的写操作地址清零使该操作无效，故障 PC 缓冲区会在处理器复位后被清零。图中因 PC6 与写操作 6 对应的 PC 一致，PC7 与写操作 7 对应的 PC 一致，写操作 6 和写操作 7 在下一个周期被无效。无效后的写操作从缓冲区冲出后不会再被发送到对外的数据总线上，进行读操作时因与无效后的写操作地址不匹配，也不会将错误的数据返回给处理器。

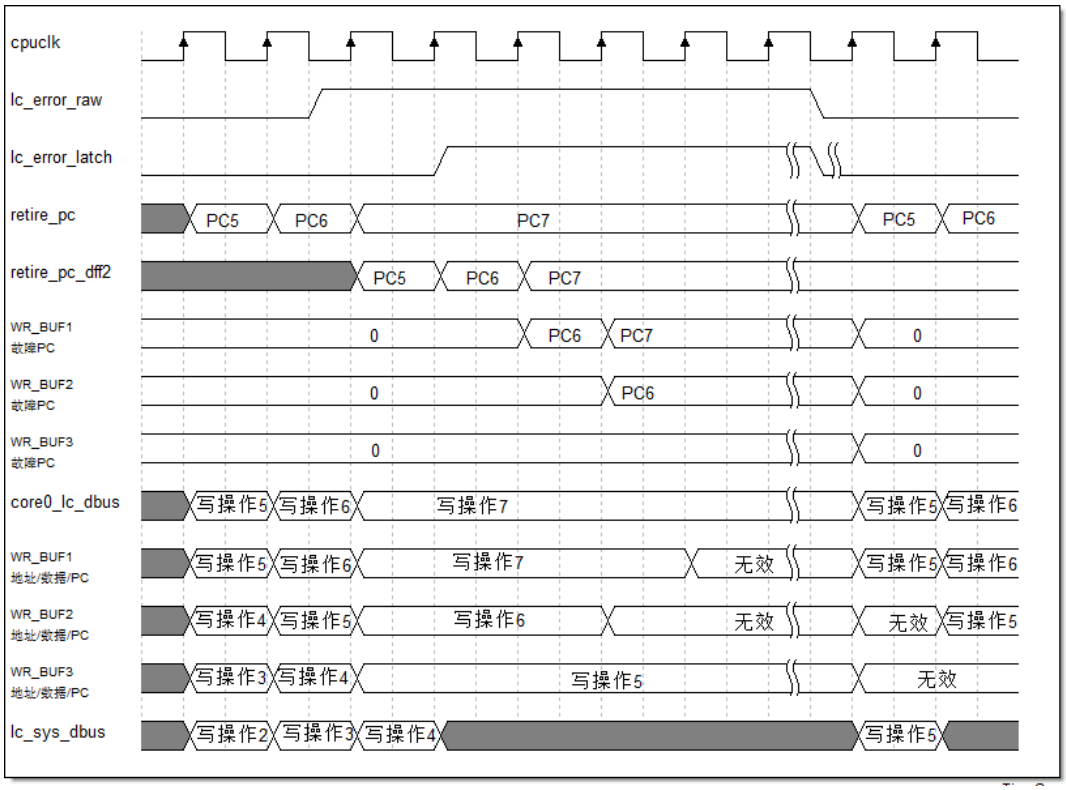


图 3.9 故障发生后存储器的状态回滚时序图

以上设计中，由于对数据总线写操作进行了缓冲，故障发生时，可以有效的隔离故障，完成存储器的状态回滚，并且整个过程不会延迟或影响到处理器对存储器的访问，因此，不会对性能造成任何损失。需要注意的点是，由于数据总线上的三个写操作被阻塞住，所以，除处理器以外的主机访问存储器时，会存在存储器与系统的状态不一致。例如 DMA（Direct Memory Access，直接内存存取）从存储器中搬运数据时，若搬运的地址范围正好包括写操作缓冲区中保存的写操作地址，那么存储中该地址对应的数据并不是最新数据，会造成错误发生。因此，需要在软件上保证，启用 DMA 搬运数据前应对存储器中一个无用地址连续进行 3 次写操作，以将保存在写操作缓冲区内的写操作冲出，更新存储器在这些地址上存储的数据。在下一次发起写操作时，当前存储的无用地址的写操作会被继续冲出，用于更新无用地址上存储的数据。但由于该地址无用，所以数据的更新不会对系统有任何影响。

（二）外围接口

本文所研究的微控制器，其对于外围接口以及系统 IP 的写操作，主要是在系统总线上进行的。为了进行故障隔离，需要对系统总线的写操作进行修改。处

理器对于外围接口以及系统 IP 的访问主要是用于控制这些 IP 的工作模式和状态，因此，与存储器不同，数据是否真的写入 IP 是非常重要的。例如，考虑这样一种情况，外围接口中的 I2C 发起了中断，处理器执行完相应的中断服务程序后准备跳出中断继续往下执行。跳出中断服务程序前需要对 I2C 进行写清中断，然而若此时采用存储器上的故障隔离方案，会导致实际写清操作没有完成，而处理器跳出中断后，因为中断没有被清除，会再次进入中断服务程序，造成实际执行情况与预想不符，从而导致系统错误。此外，对于外围接口和系统 IP 来说，与它相关的写操作往往是一些简短的控制信息，而非大量的数据信息，注重的是数据的实际写入，但对性能的要求并不高。针对这种特性，外围接口及系统 IP 的故障隔离可以通过延迟系统总线上的写操作完成。

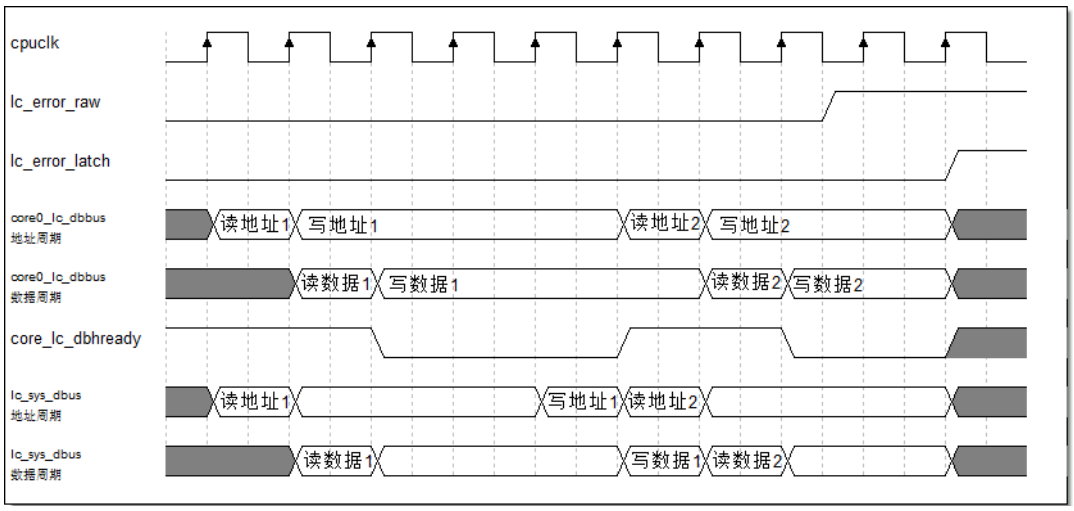


图 3.10 系统总线的写操作修改时序图

延迟操作的时序如图 3.10 所示，故障隔离模块对于读操作不进行修改，在写操作发生时，延迟三个周期将写操作传递至对系统输出的系统总线，在此期间，通过拉低 core0_lc_hready 信号，将处理器停在当前指令。处理器会等待这一信号为高时结束本次操作，然后继续执行后面的程序。若在写操作期间发生了故障，如图中对写地址 2 的写操作，故障发生在写数据周期，此时，虽然 lc_error 慢了两拍，但因为写操作得到了延迟，在三个周期后即将传递写操作时，lc_error 已经拉高，因此可以根据 lc_error 取消本次写操作。

（三）缓存

CK803 的片上缓存在写通模式下的工作原理如下：处理器发起读操作时，如

果要访问的数据在缓存中没有命中，处理器从外部存储读取该数据，并将该数据写入对应的数据存储区中，同时将其高位地址和数据有效写入到对应的标记存储区中。此后，当处理器再次对该地址进行读访问时，通过检索标记中的高位地址和数据有效位，可知缓存命中，此时可直接从缓存中读取数据而无需对存储器发起读操作。写操作下，若发生写缺失，在写通模式下，CK803 仅支持写不分配模式，即当处理器要访问的数据在缓存中没有命中时，不对缓存进行更新，只更新存储器中的数据。若写操作在缓存中命中，则将当前数据同时更新到缓存和存储器中。综上可知，处理器访问数据时，数据来源并不唯一，缓存中缺失数据时，处理器会从存储器中读取。因此，针对这一特性，可以通过对缓存中的错误数据或超前状态数据进行无效，完成缓存的故障隔离。

处理器检查缓存是否命中主要通过检索缓存标记中的高位地址以及数据有效位确定，CK803 缓存控制器中的 CIR 寄存器可以让整块缓存或指定的缓存行无效，其原理是通过对缓存标记存储区写 0 实现。对整块缓存无效，需要对每一个标记存储区内的块写 0，共需要 256 个周期。为了尽量减少无效缓存所带来的延时，只将缓存超前状态期间写入的数据无效即可。划分缓存超前状态期间的写操作比较复杂，因为一次访问可能贯穿两个指令周期。为了降低逻辑复杂度，每次状态回滚时无效N_{max}个缓存行数，其中，N_{max}为最多所需无效缓存行数。

假设处理器在状态 2 时发生故障，将回滚到状态 1，故障后最多允许执行两个写操作。因此，需要无效的缓存块最多包括状态 2 中发生的写操作加上额外两个写操作。对于缓存的写入存在两种情况，一种是发生写命中，此时缓存会在该指令期间对命中地址进行单个数据写入；另一种是发生读缺失，此时缓存会从存储器中连续读取缺失地址及其后连续三个地址的值并写入缓存（在 2 个字节间循环），在第一个指令期间缓存会完成第一个数据的写入，其余三个数据的写入则是在下一个指令期间完成。以下是不同情况下所需无效的缓存行个数：

（1） 若状态 1 和状态 2 都是写命中，则最多需要无效状态 2 中的 1 个写操作及额外两个写操作，即无效 3 个缓存行即可；

（2） 若状态 1 位写命中，状态 2 为读缺失，则最多需要无效状态 2 中的一个写操作及额外两个操作，即无效 3 个缓存行即可；

（3） 若状态 1 为读缺失，状态 2 为写命中，则最多需要无效状态 2 中 4 个操作及额外的两个写操作。然而，状态 2 中的 4 个写操作，是对读操作的缺失数据填写，并没有更新数据，所以，如果当前写操作在执行过程中没有发生错误，无需对其进行无效操作。因此，最多需要无效状态 2 中开始故障的一个写操作及额外两个写操作，即无效 3 个缓存行即可；

（4） 若状态 1 和状态 2 都是读缺失，同上，状态 2 中的 4 个写操作，是对读操作的缺失数据填写，并没有更新数据，所以，如果当前写操作执行过程中没有发生错误，无需对其进行无效操作。因此，最多需要无效状态 2 中开始故障的一个写操作及额外两个写操作，即无效 3 个缓存行即可。

（5） 最后还有一种特殊情况，即缓存中存储的数据发生错误，主从处理器从缓存中读到的数据不一致。假设缓存数据读出当周期正在进行一个写操作，并且后面紧跟两个写操作，则此时除了需要无效最后 3 个对缓存的写操作，还需要无效缓存中读出数据不一致的缓存行，即无效 4 个缓存行即可。

以上对所需无效地缓存行个数的分析，是针对单个处理器的。由于处理器发生故障后，主从处理器对于缓存的访问已经不同，若各自对各自缓存的 4 个缓存行无效，则会造成主从处理器缓存间的状态失步。为了进行缓存的故障隔离，并且保持主从处理器的缓存状态一致，可以将缓存的 8 个缓存行无效。这 8 个需要无效的缓存行其地址为：

（1） 当缓存中没有读数据错误时，主处理器缓存最后 4 个写操作地址，从处理器缓存最后 4 个写操作地址，共 8 个地址作为需要无效的缓存行地址

（2） 当缓存中出现读出数据错误时，读出数据错误的 1 个地址，主处理器缓存最后 3 个写操作地址，从处理器缓存最后 4 个写操作地址，共 8 个地址作为需要无效的缓存行地址

如图 3.11 所示，在处理器出现故障至复位跳转期间，存在一段等待的空闲周期，可以将这 8 个缓存行的无效操作在该空闲周期内完成。将标记存储区 SRAM 接口上的 CEN 拉低，同时，对这 8 个地址依次写 0，将对应的缓存行无效，从而完成对缓存的故障隔离。

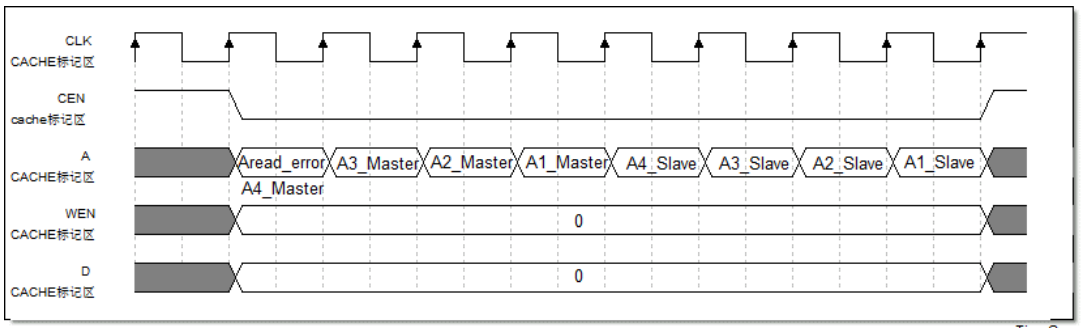


图 3.11 复位期间对缓存的无效操作时序图

3.1.4 仿真验证及逻辑综合

（一）仿真验证

为了验证以上容错功能点的正确执行，在 Testbench 中 force 相关信号，模拟处理器内部发生的故障。图 3.12 是在只有读操作的程序中，处理器内部寄存器发生错误时的故障恢复时序。容错模块在 PC 为 0x10000b90 的指令期间，检测到主从处理器内部的通用寄存器输出 reg_dout_1 不同，一个为 0x66，一个为 0x67。此时，lc_error_raw 立即拉高，并在两周期后，同步至 lc_error 信号中。在 lc_error 拉高的当周期，处理器的外部复位信号拉低，对主从处理器进行复位。复位信号经同步释放后的下一个时钟周期，主从处理器复位成功，开始从 0 地址读取跳转地址，此时，容错模块将正确节点上保存的 PC 值 0x10000b8e 替换至读数据总线。处理器获得 PC 值后，顺利跳转到 0x10000b8e 处继续取指执行。同时，复位释放后在 lc_error_pulse2 的设置下，主从处理器的 reg_dout_1 同时被还原成正确节点状态中保存的值 0x66。此后，主从处理器继续执行后面的程序，故障被恢复。

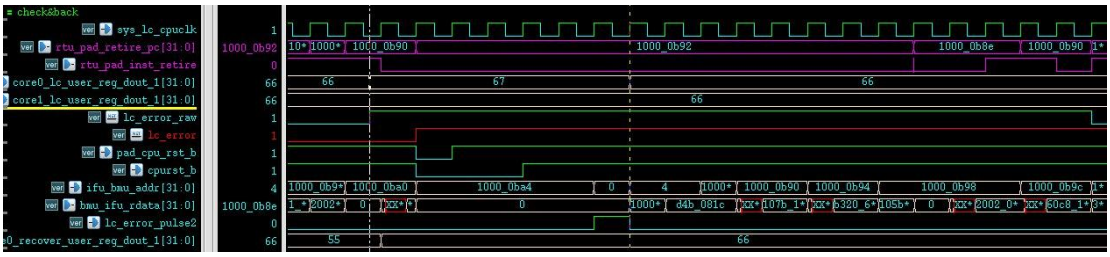


图 3.12 内部寄存器出错，处理器故障恢复时序图

图 3.13 是在只有读操作的程序中，处理器缓存读命中且缓存内发生数据错误时的故障恢复时序。第一个光标处，处理器读取 0x20028100 处数据，该地址

在缓存中命中，然而由于缓存数据出错，主从处理器缓存的输出 cache_data0_dout 不一致。此时，lc_error_raw 立即拉高，并在两周期后同步至 lc_error 信号中。随后根据 lc_error 信号进行了处理器的复位以及内部寄存器重置等故障恢复操作。与此同时，由于检测到主从处理器缓存接口的输出数据不一致，cache_dout_error 信号拉高，此时会将当前出现读错误的缓存地址 0x10 锁存到 cache_dout_addr 中。随后，在处理器复位期间，容错模块依次对读错误地址 0x10、主处理器缓存的最后 3 个写操作地址 0x14，0x18，0x1c 以及从处理器最后 4 个写操作地址 0x10，0x14，0x18，0x1c 对应的标记存储区清零。处理器复位后跳转到 0x10000b60 处重新取指执行，第二个光标处，处理器重新读取 0x20028100 处数据，此时错误的缓存数据已被无效，处理器通过总线从存储器中获取正确数据，故障顺利恢复。

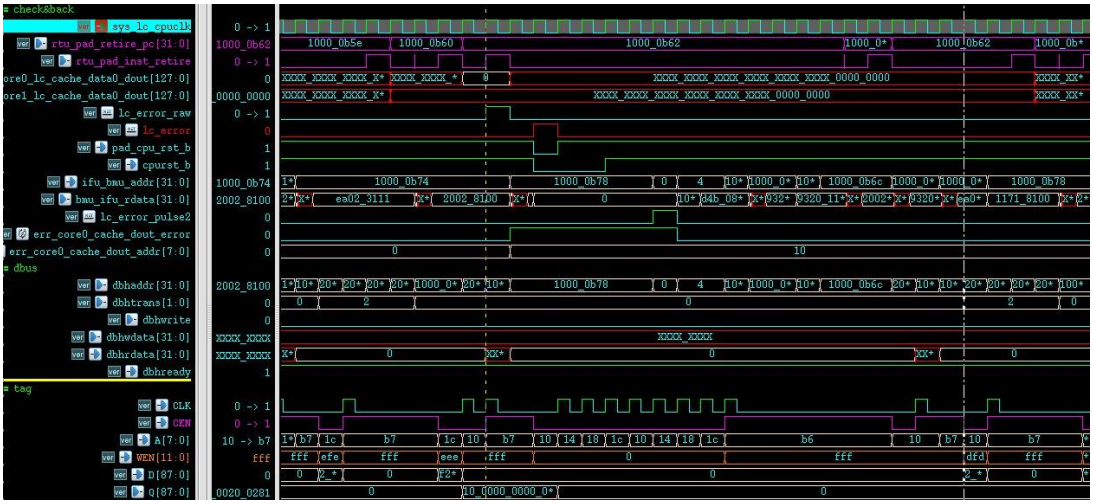


图 3.13 Cache 数据出错，处理器故障恢复时序图

当执行的程序涉及到对系统的写操作时，需要进行相应的故障隔离处理。首先是系统总线，如图 3.14 所示，处理器对外设地址 0x40017068 进行写操作时，在第一个光标处检测到主从处理器系统总线上的写数据 lc_hwdata 不一致，主处理器的写数据跳变成了 0x14。此时，由于容错模块对系统总线上的写操作进行了三个周期的延迟，当 lc_sys_bus 总线上要传递被延迟的写操作时，检测到 lc_error 信号已拉高，因此取消了本次写操作。写操作并没有在对外的系统总线上发送，因此该错误数据并未被写入系统中。当处理器完成故障恢复操作后，再次对外设地址 0x40017068 进行写操作,这次没有出现故障，写操作被延迟 3 个周期后，在第二个光标处，正常发送到对外的系统总线上，数据被写入外设。

图 3.14 处理器系统总线上的写操作隔离及故障恢复时序图

其次是数据总线，如图 3.15 所示，处理器对存储器地址 0x20028110 进行写操作时，在第一个光标处检测到主从处理器数据总线上的写数据 lc_dbhwdata 不一致，主处理器的写数据从 0x55 跳变成了 0x14。上一个正确执行完的指令 PC 为 0x10000ba8，该指令从 0x20028118 地址中读数据。本次写操作是处理器发起的第五次写操作。由于写操作缓冲区的存在，当前第五次写操作信息被存入 buf_wr_dbbus 中，对外的数据总线 lc_sys_dbus 上实际发送的是第二次写操作——对 0x20028104 地址写 0x22。由于 lc_error 还未拉高，后面的第六次及第七次写操作正常执行，但对外数据总线上实际执行的是第三次和第四次写操作。发生故障后，err_pc*保存了故障发生期间对应的 PC。在复位期间，写操作缓冲区中与故障 PC 缓冲区中存储的 0x20028110，0x20028114，0x20028118 一致的 PC 对应的写操作地址被清零。故障恢复后，处理器从 0x10000ba8 处取指重新执行，第二个光标处，重新发起对 0x20028118 中的读操作。此时，由于 0x20028118 被无效，在写操作缓冲区没有命中，读出的数据直接来自存储器为 0x0，而非错误操作期间写入缓冲区的 0x14，故障顺利恢复。

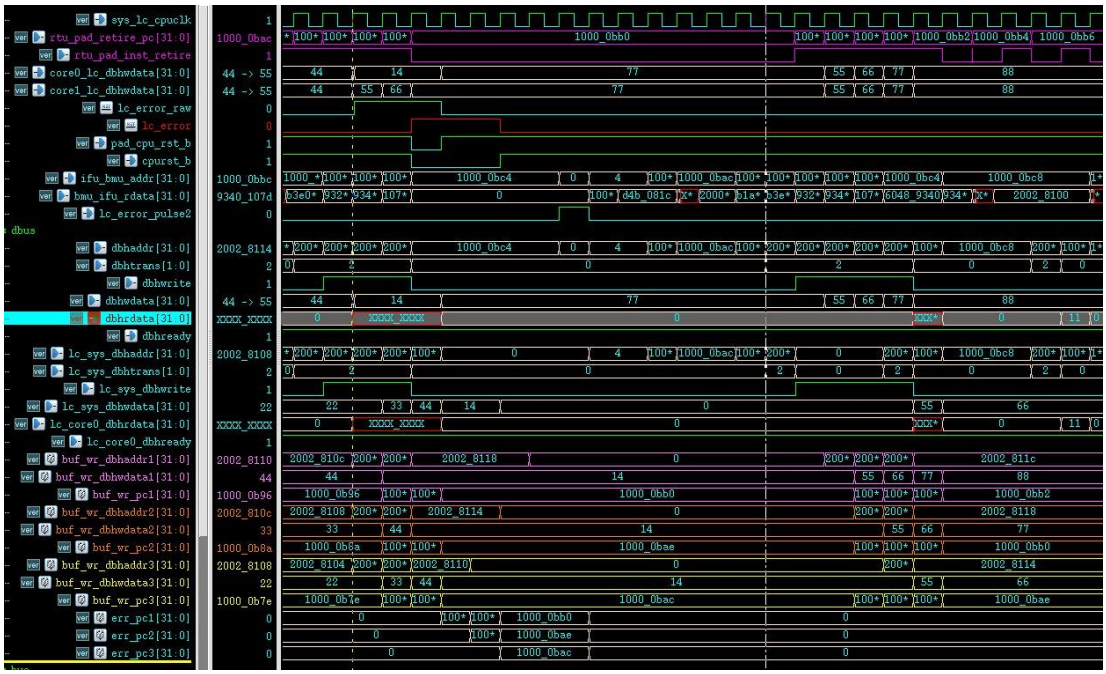


图 3.15 处理器数据总线上的写操作隔离及故障恢复时序图

最后是缓存，如图 3.16 所示，处理器进行读操作时，发生读缺失后，从存储器中连读四个数据并写入缓存。在第一个光标处，进行第一个写操作时，由于主从处理器内部状态寄存器不一致，检测到错误产生。此时，主从处理器缓存最近执行的 4 次写操作地址都得到了保存。在处理器故障恢复期间，8 个写操作地址对应的标记都被写入 0。复位成功后，处理器回滚到 0x10000b54 处取指执行，在第二个光标处重新进行读操作。此时，因为缓存已被无效，于是发生读缺失。处理器再次从外部存储中读取，故障顺利恢复。

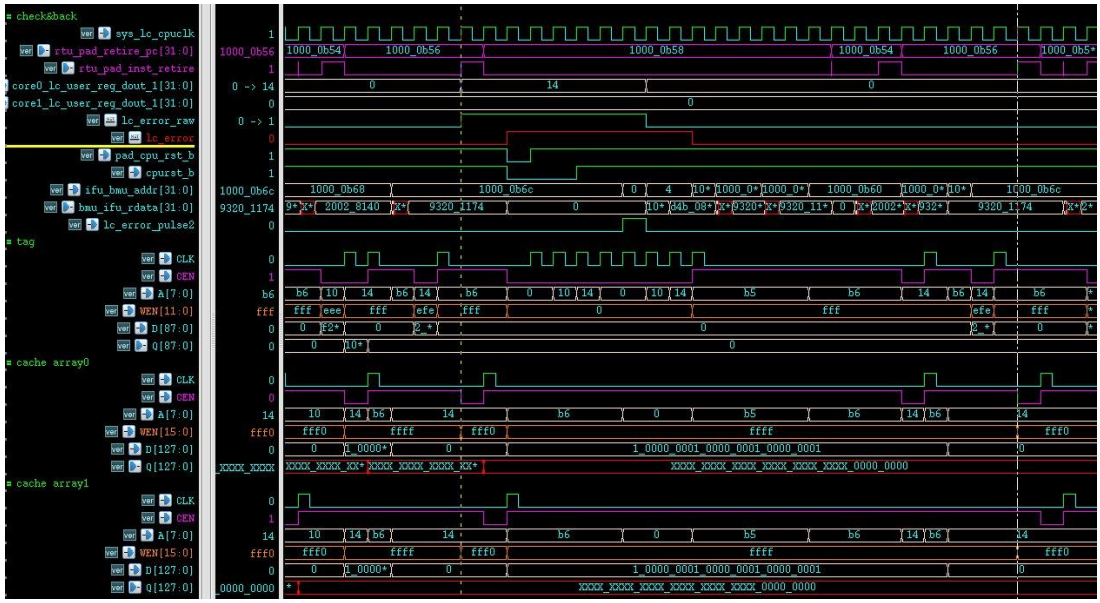


图 3.16 处理器 Cache 的写操作隔离及故障恢复时序图

（二）逻辑综合

为了对容错后的处理器进行指标评估，使用 Design Compiler 分别对单个处理器和容错处理器进行综合。采用中芯国际的 55nm 工艺以及电压为 1.08V，温度为-40℃的工艺角。单个处理器和容错处理器综合后的各项指标如表 3.4 所示。从表中数据可知，容错处理器的面积是单个处理器面积的 2.2 倍，而 TMR 技术通常在 3 倍以上，因此，本方案节省了较大的面积成本。在时序上，容错后的处理器时序有所下降，但下降幅度并不大，仍在可接受范围内。

表 3.4 单个处理器与容错处理器综合后的指标对比

指标	单个处理器	容错处理器
总面积（166M 时钟约束下）	271327.85um^2	599700.37um^2
等效门数（166M 时钟约束下）	242257	535447
功耗（166M 时钟约束下）	12.68mW	32.79mW
最大时钟频率	330M	300M

3.2 基于双核容错的处理器抗功耗分析攻击设计

基于双核容错处理器的抗功耗分析攻击设计分为同步运行和失步运行两个阶段。这里的同步、失步指的是主从处理器间的运行状态是否一致，若运行状态一致，称其为同步运行，若运行状态不一致，则称其为失步运行。如图 3.17 所

示，失步运行期间，从处理器随机插入延时，主处理器在写操作前随机插入延时以等待与从处理器的同步运行。同步运行期间，对主从处理器内部的控制状态寄存器以及部分接口信号作对比，进行故障检测。由于缓存攻击的存在，软件加密过程中开启缓存会增加被攻击的风险，因此，本研究在功耗隐藏模式开启时不考虑缓存的使能。

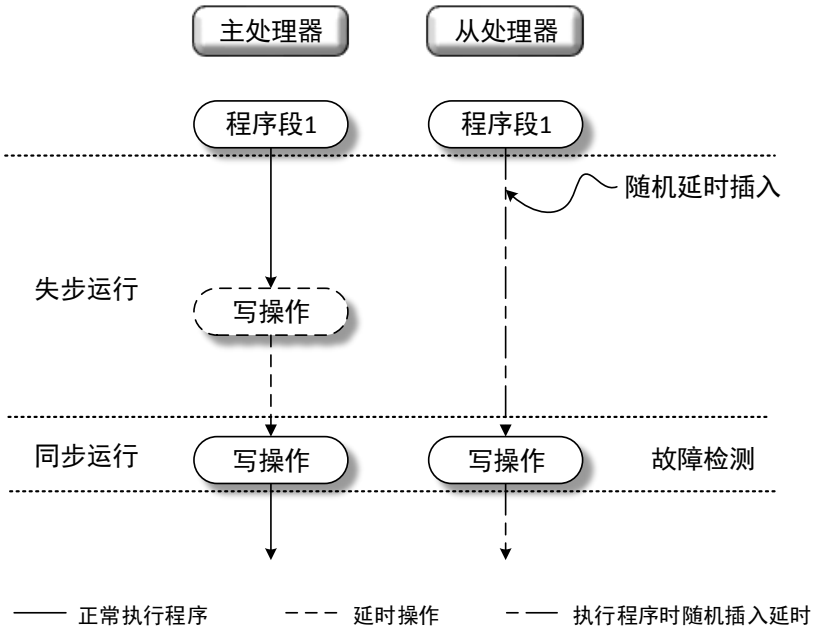


图 3.17 主从处理器同步与失步运行示意图

3.2.1 失步运行设计

由于微控制器的 Lockstep 容错架构为主从式运行，只有主处理器拥有对外的输入输出。从处理器的输入与主处理器一致，输出只用于和主处理器的输出作对比，以判断故障的发生。因此，一旦失步运行，从处理器就无法获得正确的输入。通过对 CK803 的输入接口分析可知，在进行加解密操作时，需要产生变化的信号主要是指令总线接口及数据总线接口，处理器需要从中获取指令和数据。因此，主从处理器失步运行后，需要额外的设计提供从处理器所需的指令以及数据。图 3.18 所示为整个安全容错模块的结构框图，除了用于容错设计的模块外，还包括提供主从处理器失步运行的相关设计。随机延迟使能模块用于产生随机延迟使能信号，从处理器在随机延迟使能信号为高时，进行延迟操作。读操作存储区由多组 FIFO 构成，用于保存主处理器读取指令和数据时对应的地址和数据。

当从处理器发起对指令和数据的访问时，将从读操作存储区中获得。

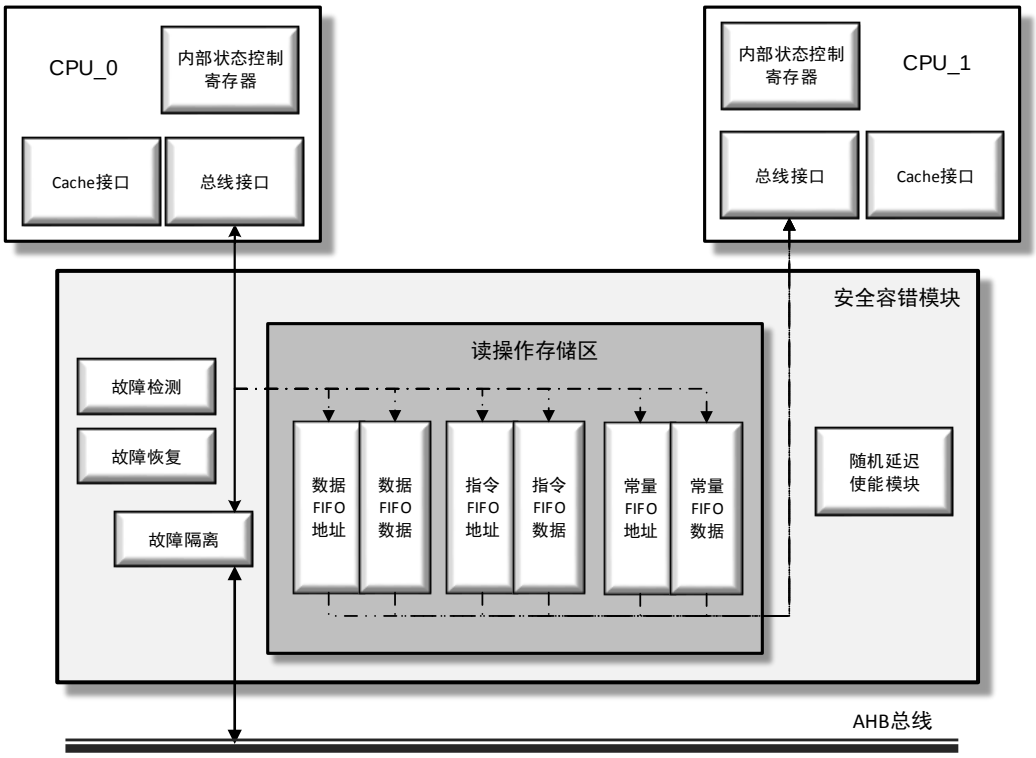


图 3.18 安全容错模块结构框图

随机延迟使能信号用于控制从处理器的随机延迟，当使能信号为高，从处理器插入延时，进入等待状态；当使能信号为低，从处理器恢复运行。如图 3.19 所示，随机延迟信号由真随机数、计数器和相关控制逻辑构成。从真随机数发生器获得的随机数经掩码后的数据为 rand_num。在功耗隐藏模式开启，即 lc_aten 为高时，对 rand_num 进行计数，当计数至 0 时，Tcore1_delay 拉高；然后再获取一次 rand_num 并对其计数，当计数至 0 时，Tcore1_delay 拉低。该过程反复进行以获得随机高低电平的 Tcore1_delay。延迟时间以及正常运行时间可以通过配置掩码值来控制，掩码后的随机数有效位宽越大，从处理器延迟插入的随机性越大，但同时也会带来更高的性能开销，因此，可以根据实际情况配置合理的掩码值。

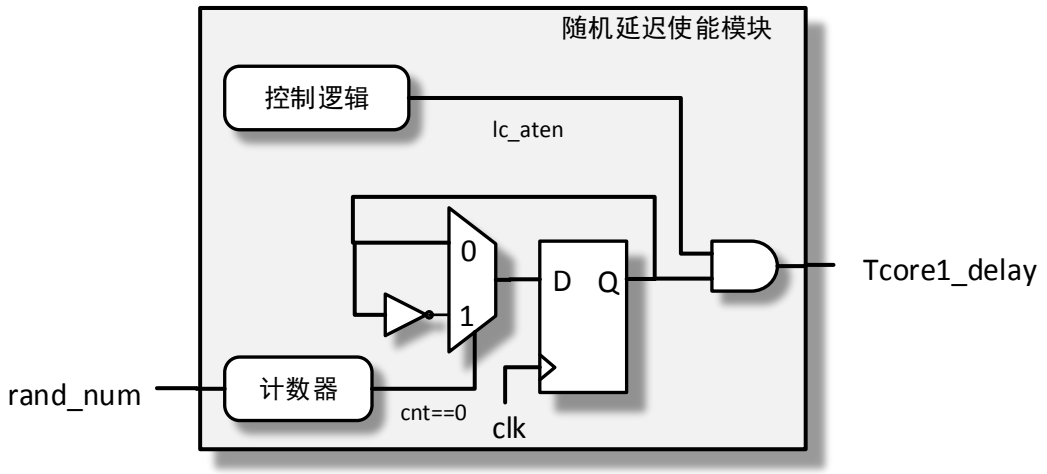


图 3.19 随机延迟使能模块结构图

读操作存储区中包含6个深度为16的FIFO,分为3组。第一组为数据FIFO，其中的2个FIFO用于存储读取的数据和对应的地址；第二组为指令FIFO，其中的2个FIFO用于存储读取的指令和对应的地址；第三组为常量FIFO，其中的2个FIFO用于存储读取的常量和对应的地址。由于主从处理器在正确的执行情况下,即便进行了随机延迟,处理器的执行顺序也会保持一致,因此,可以采用FIFO结构，将主处理器中所读取过的指令和数据写入到FIFO中，失步运行后从处理器读取的指令和数据从FIFO中获得。

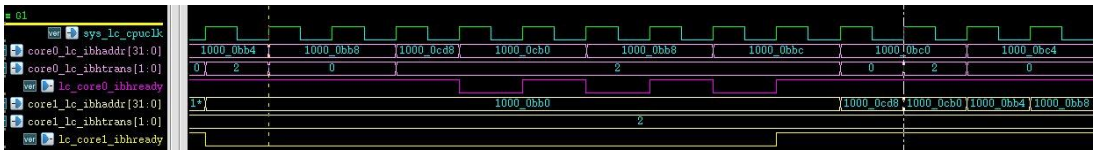


图 3.20 主从处理器异步运行后的指令读取顺序不同

然而，当两处理器异步运行后，指令总线上的指令读取顺序并不完全一致，如图 3.20 所示，主处理器 CORE0 在第一个光标处开始依次读取的四个指令地址分别是0x10000bb4，0x10000cd8，0x10000cb0，0x10000bb8；从处理器CORE1从第二个光标处开始依次读取的四个指令地址分别是0x10000cd8,0x10000cb0，0x10000bb4，0x10000bb8。图 3.21 为一段程序编译后的.obj 文件，程序中的一些常量会被编译至代码末尾,从图中可知0x10000cd8以及0x10000cb0属于代码段中的常量存储区。由于从处理器在0x10000bb0处取指阻塞，CK803为了提高性能，在正常指令读取出现阻塞时，会先去读取常量存储区指令，从而导致了指令

读取顺序的不一致，但正常指令与常量各自之间依旧会保持不变的读取顺序。因此，为了保持从处理器能够按照正确的顺序获取正常指令和常量，将常量和正常指令拆分为 2 组 FIFO 分别保存。

```
1460 10000c7c:       6506            cmpne           r1, r4
1461 10000c7e:       0803            bt              0x10000c84
1462 10000c80:       0405            br              0x10000c8a
1463 10000c82:       6c47            mov             r1, r1
1464
1465 10000c84 <TEST_ERROR>:
1466 10000c84:       6c47            mov             r1, r1
1467 10000c86:       10fd            lrw             r7, 0x100000d4
1468 10000c88:       781c            jmp             r7
1469
1470 10000c8a <SUCCESS>:
1471 10000c8a:       6c47            mov             r1, r1
1472 10000c8c:       ea89001c        lrw             r9, 0x100000c0
1473 10000c90:       6fe7            mov             r15, r9
1474 10000c92:       783c            rts
1475 10000c94:       e000f008        .long   0xe000f008
1476 10000c98:       e000f00c        .long   0xe000f00c
1477 10000c9c:       e000f004        .long   0xe000f004
1478 10000ca0:       e000f000        .long   0xe000f000
1479 10000ca4:       e000e100        .long   0xe000e100
1480 10000ca8:       0000000b        .long   0x0000000b
1481 10000cac:       e000e240        .long   0xe000e240
1482 10000cb0:       20020000        .long   0x20020000
1483 10000cb4:       40017060        .long   0x40017060
```

图 3.21 程序编译后的.obj 文件格式

当开启功耗隐藏模式后，主从处理器会从同步运行状态转变至失步运行状态。此时，从处理器在符合以下延迟条件时，进行相应的延迟操作：

（1） 随机延迟使能信号 Tcore1_delay 为高时；

（2） 从处理器的运行状态追上主处理器，读操作存储区中的 FIFO 为空时；

当以上条件不满足时，从处理器正常取指执行。从处理器的随机延迟会导致主从处理器间的状态差距越来越大，为了主从处理器够达到一致的执行状态，进行结果对比，主处理器也需要插入延迟。主处理器在开启功耗隐藏后符合以下条件时，进行延迟操作：

（1） 读操作存储区中的 FIFO 满时；

（2） 主处理器需要执行写操作时；

处理器的延迟通过拉低 AHB 总线上的 hready 实现，此时处理器会停在当前指令周期，直到 hready 为高时，才会认为当前指令完成，从而继续往下执行程序。

3.2.2 同步比较设计

功耗隐藏模式开启后，处理器的故障检测只在写操作期间进行，由于不考虑缓存的使能，对比的内容为处理器内部状态控制器和总线接口信号。

（一）内部控制状态寄存器对比

主从处理器内部寄存器的对比在主处理器发起写操作的指令周期进行。当检测到主处理器在数据总线上发起写操作，将数据总线的 hready 拉低，使得主处理器停在当前指令。当从处理器也执行到这一指令并发起写操作，此时，将数据总线的 hready 拉高，主处理器当前指令正常退休。在正常运行的情况下，该指令退休期间，主从处理器内部的寄存器状态应一致，此时对内部寄存器值作对比，如果对比一致，则认为当前无故障发生。

（二）总线接口对比

主从处理器插入随机延时后，两处理器的接口时序不再一致，难以直接对比。由于在加解密过程中主要涉及指令总线及数据总线上的数据读写。因此，只需在接口上保证读写数据的正确性即可。

（1） 指令 FIFO 中保存了主处理器每次读指令的地址和数据，从处理器从指令 FIFO 中取读数据的同时，也会读取地址，并将 FIFO 中的读地址与当前从处理器自身发起的读指令地址作对比，如果对比一致，则认为当前无故障发生。

（2） 数据 FIFO 中保存了主处理器每次读数据的地址和数据，从处理器从数据 FIFO 中读取数据的同时，也会读取地址，并将 FIFO 中的读地址与当前从处理器自身发起的读数据地址作对比，如果对比一致，则认为当前无故障发生。

（3） 在进行写操作时，将写操作期间主从处理器间的写地址和写数据作对比，如果对比一致，则认为当前无故障发生。

3.2.3 仿真验证及逻辑综合

（一）仿真验证

为了验证以上功能点的正确执行，编写了相关的测试用例。图 3.22 是开启功耗隐藏模式后，主从处理器的指令运行时序。从图中可以看出，功耗隐藏使能

lc_aten 拉高后，Tcore1_delay 在下一个周期拉高。两处理器在第一个光标处开始，指令的执行状态开始不一致，从处理器的状态因延迟的插入而停在 PC 为 0x10000b8c 的指令周期，直到 Tcore1_delay 拉低后，在第二个光标处，从处理器才恢复正常的取指执行过程。

图 3.22 功耗隐藏使能后，主从处理器异步运行时序图

当主处理器执行到写操作指令时，主从处理器需要进行状态同步。图 3.23 是主从处理器在写操作时的状态同步时序。图中，主处理器在第一个光标处发起了写操作，此时，该总线上的 hready 信号拉低，主处理器始终停在写操作指令期间。当从处理器状态追上后，也发起了对同一地址的写操作，主处理器的 hready 信号在第二个光标处拉高，主从处理器同时在当周期完成写操作的执行。此时，两处理器的内部状态一致。

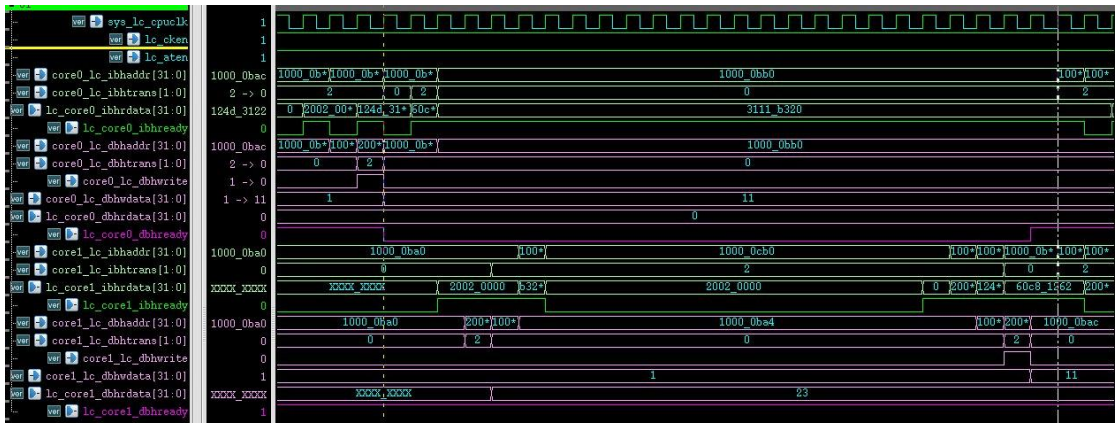


图 3.23 写操作时，主从处理器同步运行时序图

功耗隐藏模式开启时，处理器内部若发生故障，安全容错模块依旧能够保证故障的恢复。图 3.24 是故障的恢复时序。从处理器在图中第一个光标处追上主处理器的状态，并发起相同的写操作。故障检测模块在此时开始对主从处理器状态作对比，发现两处理器的 reg_dout_1 不同，一个为 0x14，一个为 0x44，此时

lc_error_raw 拉高。随后两个周期后 lc_error 拉高，并对两处理器进行复位。两处理器在第二个光标处，同时获取到上一个写操作执行时保存的正确节点状态中的 PC，0x10000bca，并跳转到该 PC 重新执行指令。同时，由于功耗隐藏模式的开启，故障恢复后的主从处理器又开始了异步运行。

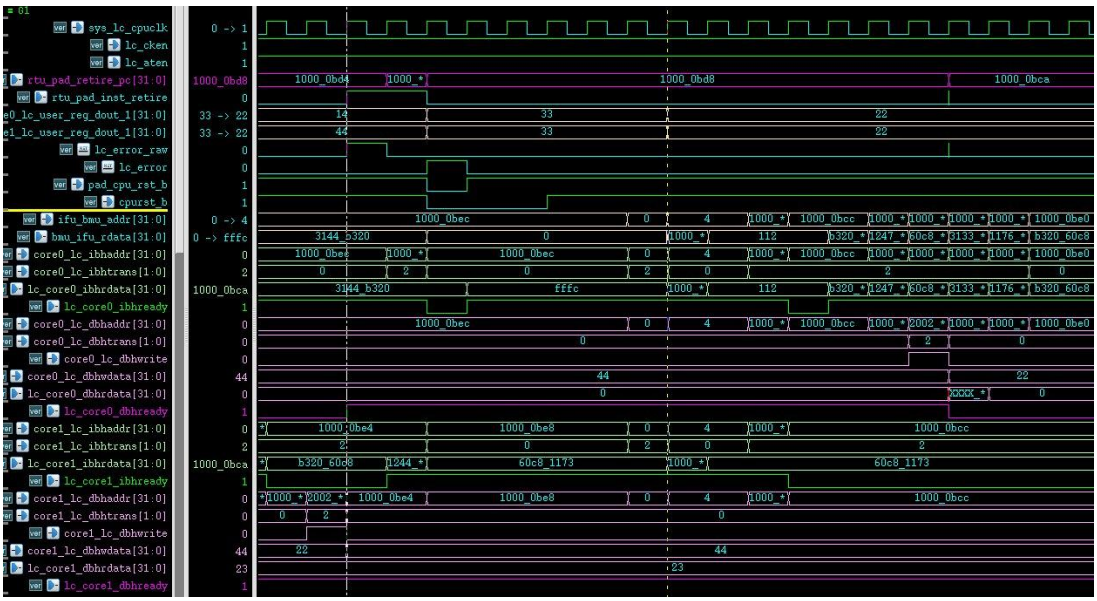


图 3.24 功耗隐藏使能期间发生故障后的恢复时序图

（二）逻辑综合

为了对安全容错后的处理器进行指标评估，使用 Design Compiler 分别对单个处理器和安全容错处理器进行综合。采用中芯国际的 55nm 工艺以及电压为 1.08V，温度为-40℃的工艺角。单个处理器和安全容错处理器综合后的各项指标如表 3.5 所示。从表中数据可知，安全容错处理器的面积是单个处理器面积的 2.4 倍，相比单独的容错模块，面积占用有所提高，这主要是由于功耗隐藏模块中包含 6 个 16x32 的 FIFO，但相比 TMR 技术仍具有一定的面积优势。在时序上，安全容错后的处理器时序有所下降，但下降幅度并不大，仍在可接受范围内。

表 3.5 单个处理器与安全容错处理器综合后的指标对比

指标	单个处理器	安全容错处理器
总面积（166M 时钟约束下）	271327.85um^2	650526.07um^2
等效门数（166M 时钟约束下）	242257	580826
功耗（166M 时钟约束下）	12.68mW	33.23mW
最大时钟频率	330M	305M

3.3 存储器容错设计

存储器的检纠错模块内嵌在存储器控制器接口与存储器宏单元之间，对输入数据和输出数据进行实时的编解码。如图 3.25 所示，检纠错模块主要分为两个部分：检纠错编码模块和检纠错译码模块。编码模块将从控制器接口中输入的写数据进行编码，并将编码后的数据写入存储器。译码模块将从宏单元中读出的数据进行译码，译码无误的数据返回至控制器接口。

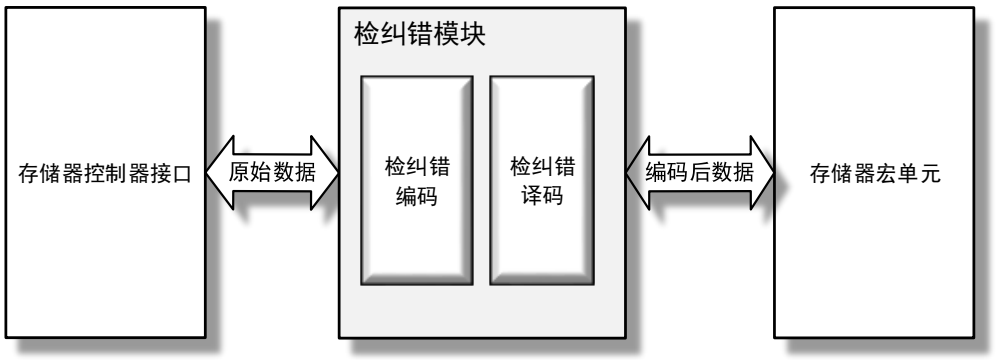


图 3.25 检纠错模块结构图

3.3.1 汉明码技术

汉明码是由 Richard Hamming 于 1950 年提出的一种线性分组码，目前广泛应用于存储及通信容错中。汉明码的最小汉明距为 3，能够纠正一位错误。其原理是对信息码按照一定的方式编码，并将获得的校验码附在信息中。进行校验时，通过确定当前信息码与校验码是否符合既定的运算关系，进而判断是否有错误发生。若发生错误，可以根据具体的错误形式，对错误位置进行定位和纠正。下面是对汉明码的编译码方法介绍。

首先需要根据信息码的长度确定校验码的长度。式 3.1 为汉明码信息位与校验位间的关系式。

$$n = k + r \leq 2^r - 1 \qquad (3.1)$$

其中 n 为信息码和校验码的总长度，k 为信息码的长度，r 为校验码的长度。由该公式可以确定指定信息码所需的校验码长度，常用的信息码所对应的汉明码校

验位长度如表 3.6 所示。对于本研究来说，由于片内存储上的数据位宽为 32 位，因此使用汉明码纠错，需要增加 6 位校验位。

表 3.6 常用信息码对应的汉明码校验位长度

信息码位数	校验码位数	信息码位数	校验码位数
1	2	27~57	6
2~4	3	58~120	7
5~11	4	121~247	8
12~26	5	248~502	9

其次，计算对应的校验值。汉明码需要对信息码和校验码按照一定的规则进行分组，校验值由其所在分组内的其他信息码决定，要求同一分组内 1 的个数固定为偶数（偶校验）或奇数（奇校验）。图 3.26 是一个长度为 4 的信息码，其 3 个校验位所对应的分组。将校验位插入信息码的第 2^i 次方位置，如第 1、2、4 位。第一个校验位 P_0 位于其后每隔 1 位的数据分组，第二个校验位 P_1 位于其后每隔 2 位的数据分组，第三个校验位 P_2 位于其后每隔 4 位的数据分组。以此类推，第 r 个校验位 P_{r-1} 位于其后每隔 2^{r-1} 位的数据分组。分好的组中，每组只包含一个校验码，其他都是信息码，该校验码的值可由偶（奇）校验计算获得。

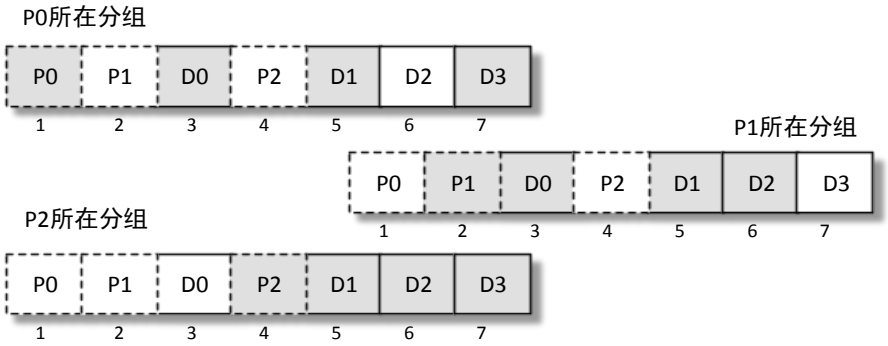


图 3.26 4 位信息码汉明校验分组示意图

编码后得到的校验码将和信息码一同送入存储器中保存，当再次读取该数据时，需要对该数据进行译码。译码的方式和编码一致，对各数据分组重新进行偶（奇）校验，并将校验结果与编码时保存的校验结果进行异或获得校正子。在数据没有出错的情况下，各组前后校验结果一致，校正子应为 0。若校正子不为 0，说明数据发生了错误。在只发生单比特错误的情况下，可以通过校正子的值定位

出错误位置。表 3.7 所示为一个（7,4）汉明校验的校正子与错误位置间的关系。若校正子为 1，则错误指向数据的第一位，即 P_0；若校正子为 2，则错误指向数据的第二位，即 P_1 得；若校正子为 3，则错误指向数据的第三位，即 D_0。以此类推，校正子的值表明了当前错误位置。

表 3.7 校正子与错误位置关系

校正子	错误位置	错误数据	校正子	错误位置	错误数据
000	无错误	无错误	100	4	P2
001	1	P0	101	5	D1
010	2	P1	110	6	D2
011	3	D0	111	7	D3

3.3.2 检错码技术

（一）奇偶校验码

在差错控制编码中，最常用的检错码是奇偶校验码。它通过增加 1 个校验位使得码字中 1 的个数保持为偶数（偶校验）或奇数（奇校验）。然而奇偶校验码对于错误的检出能力有限，它只能检测出奇数个错误，但对偶数个错误则无能为力。在本研究中，经过汉明编码后的存储数据位宽为 38，奇偶校验所需的校验位为 1，因此，共有 39 位数据。考虑在恶意攻击情况下，单个数据发生错误的概率 p_e 较高为 0.1 时，漏检概率 P 为：

$$\mathrm{P}=\sum_{i=1}^{19} p_e^{2i}(1-p_e)^{39-2i}\approx 0.48 \qquad (3.2)$$

（二）水平垂直奇偶校验码

水平垂直奇偶校验码是在奇偶校验码的基础上形成的。如图 3.27 所示，以 25 位信息码为例，在编码时，首先将信息码排列成 5x5 的矩阵形式，并对矩阵的每一行，每一列分别进行校验计算，右下角的校验位可对该行或该列或所有数据进行检验，最终获得了位宽为 11 的校验码。该校验码所占的校验位宽大，但是具有优良的检错性能，能够检测所有奇数个错误以及大部分偶数个错误。不能检测的错误情况是，发生偶数个错误且错误发生的情况使得矩阵中的每一行每一

列上的错误数量都为偶数个，例如，一个矩形上的 4 个点发生错误。在本研究中，经过汉明编码后的存储数据位宽为 38，若采用水平垂直奇偶校验，若不考虑多余的两位信息位，以 6x6 的矩阵形式展开，需要 13 位校验数据。现考虑这 36 个信息位经水平奇偶校验后的漏检率，同样地，假设当前单个数据发生错误的概率p_e为 0.1，无法检测到的错误的个数可能为 4 个，6 个，8 个…48 个，假设其对应的错误形式组合数为 E_4， E_6，E_8…E_{48}，则漏检概率可用式 3.3 计算：

$$P = E_4 p_e^4 (1-p_e)^{45} + E_6 p_e^6 (1-p_e)^{43} + \cdots + E_{48} p_e^{48} (1-p_e)^1 \quad (3.3)$$

其中的 E_4 可以计算出为$C_7^2 \times C_7^2$，第一项的结果为 3.8e-4；E_6 可以计算出为$C_7^3 \times C_7^3 \times C_3^2 \times 2$，第二项结果为 7.9e-5。因此，可粗略估计水平垂直奇偶校验的漏检率大于 4e-4。

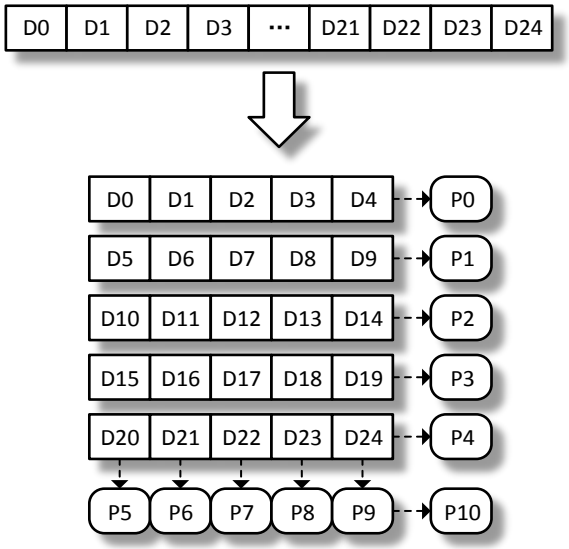


图 3.27 24 位信息码水平奇偶校验示意图

（三）恒比码

恒比码又称定比码或等比码，要求每组码中 0 和 1 的个数保持恒定的比例，在通信可靠性中常用的有五三恒比码，即每五个数据中，都由 3 个 1 和 2 个 0 组成。5 位的数据只能表示 0~9 十个数字。恒比码可以发现所有奇数个错误，及部分偶数个错误，对于偶数个错误中，若恰好为 0 错成 1,1 错成 0 的情况，恒比码无法检测。对于本研究所需检测的 38 位数据来说，若采用恒比码的思想，对 38 位数据重新编码，采用（45,27）恒比码，每一串信息位长 45，其中有 27 个 1，通过 1 在数据中的排列位置不同表示不同的信息即C_{45}^{27}，该值大于2^{38}，可以保证

覆盖所有信息码。无法检测的错误个数可能为 2 个，4 个，6 个，…44 个，但只有错误的数据中 1 和 0 的个数相等时，才无法检测，则其漏检率为：

$$\mathrm{P}=\sum_{i=1}^{18} C_{27}^{i} \times C_{18}^{i} \times p_{e}^{2i}(1-p_{e})^{45-2i} \approx 0.18 \qquad (3.4)$$

（四）群计数码

群计数码是对信息码中 1 的个数进行计数，并将最终计数值的二进制码作为校验位，添加到信息码中。例如，信息码为 1010001，信息码长度为 7，所以校验码的位宽应为 3。信息码中 1 的个数为 3，对应的二进制码为 11，所以最终的校验位为 011。群计数码可以校验所有奇数个错误，但和恒比码一样，对于恰好 0 错成 1,1 错成 0 的情况，群计数码无法检测。在本研究中，经过汉明码编码后的存储数据位宽为 38，采用群计数码进行检错需要 6 位校验码，无法检测错误个数可能为 2 个，4 个，6 个，…44 个，但只有错误的数据中 1 和 0 的个数相等时，才无法检测，则其漏检率为：

$$\mathrm{P}=\sum_{i=1}^{22} \frac{C_{2i}^{i}}{2^{2i}} \times C_{44}^{2i} \times p_{e}^{2i}(1-p_{e})^{44-2i} \approx 0.18 \qquad (3.5)$$

（五）循环冗余校验码

循环冗余校验码（Cyclic Redundancy Check，CRC）在数据存储及通信领域中被广泛使用。CRC 校验的基本思想是对信息码进行求余，并将获得的余数作为校验位附加到信息码中。对于 CRC 校验来说，最重要的是这个除数的选择，该除数也称为 CRC 码的生成多项式，经许多学者研究，整理并形成了一套常用生成多项式，包括 CRC4，CRC8，CRC12，CRC16 等等，CRC 生成多项式的最高次项越大，相应的检错能力也更强。以 CRC4 为例，CRC 的编解码过程如下：

（1） 通过查表获取 CRC4 的生成多项式：x^4+x^3+1，对应的二进制为：11001。

（2） 假设信息字段代码为：1011001，左移 4 位（由生成多项式最高次项决定）后为 10110010000。

（3） 通过多项式除法，用移位后的信息码除以多项式，这里的除法为模 2 除，不考虑借位，获得的余数 1010 即为校验码，编码后的数据为 10110011010。

（4） 对于 CRC 编码后的数据进行校验，将编码后数据 10110011010 除以生成

码 11001，如果能除尽，则正确，否则数据出错。

在本研究中，经过汉明码编码后的存储数据位宽为 38，若采用 CRC12 作为检错码，则需要 12 位校验码，其漏检率可以通过式 3.6 近似计算[50]：

$$\mathrm{P} = \frac{1}{2^{12}}(1-(1-p_e)^{50}) \approx 2.4e-4 \qquad (3.6)$$

根据以上对各类常用检错码的研究和分析，表 3.8 总结了以上常用检错码对 38 位数据位宽的信息码进行检错，所需校验位宽，漏检率等特性。从表中可知，群计数码和奇偶校验码的校验位宽及漏检率固定，无法进行进一步提高漏检率。恒比码的校验位宽随恒定比可变，且随着恒定比的提高，漏检率会随之降低。然而恒比码由于其编码规则限制，存在需要的禁用码，所以编码效率不高。漏检率最低的是水平垂直奇偶校验码和 CRC 校验码，虽然水平垂直奇偶校验的编码更为简单，性能开销更小。然而，由于本研究对于恶意攻击产生的多比特错误检测有较高的要求，因此采用漏检率更低的 CRC 循环冗余校验码作为错误检出码。

表 3.8 常用检错码的检错能力比较

编码方式	编解码性能	校验位宽	漏检率(P_e=0.1 时)
奇偶校验码	—	1 位	0.48
水平垂直奇偶校验码	最多存在 7 级异或	13 位	＞4e-4
恒比码	—	7 位	0.18
群计数码	—	6 位	0.18
CRC12 校验码	最多存在 26 级异或	12 位	2.4e-4

3.3.3 检纠错设计

为了确保所执行代码的正确性，使用汉明码对数据进行纠错编码，并使用 CRC 对编码后的数据再次进行校验编码。编码过程如图 3.28 所示：

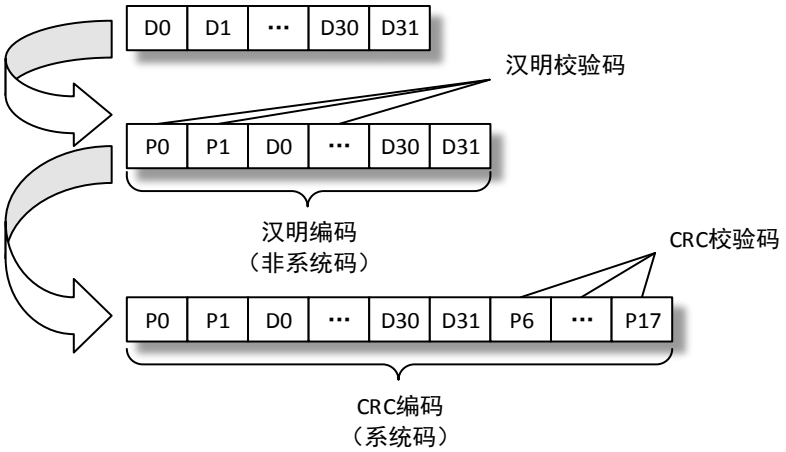


图 3.28 检纠错编码过程示意图

首先是对 32 位总线数据进行汉明编码。根据前面所介绍的汉明码编码方式，对于 32 位信息位 D[0]~D[31]，各个校验位可通过以下异或逻辑获得：

P[0]=D[0]⊕D[1]⊕D[3]⊕D[4]⊕D[6]⊕D[8]⊕D[10]⊕D[11]⊕D[13]⊕D[15]⊕D[17]⊕D[19]⊕D[21]⊕D[23]⊕D[25]⊕D[26⊕D[28]⊕D[30]；

P[1]=D[0]⊕D[2]⊕D[3]⊕D[5]⊕D[6]⊕D[9]⊕D[10]⊕D[12]⊕D[13]⊕D[16]⊕D[17]⊕D[20]⊕D[21]⊕D[24]⊕D[25]⊕D[27]⊕D[28]⊕D[31]；

P[2]=D[1]⊕D[2]⊕D[3]⊕D[7]⊕D[8]⊕D[9]⊕D[10]⊕D[14]⊕D[15]⊕D[16]⊕D[17]⊕D[22]⊕D[23]⊕D[24]⊕D[25]⊕D[29]⊕D[30]⊕D[31]；

P[3]=D[4]⊕D[5]⊕D[6]⊕D[7]⊕D[8]⊕D[9]⊕D[10]⊕D[18]⊕D[19]⊕D[20]⊕D[21]⊕D[22]⊕D[23]⊕D[24]⊕D[25]；

P[4]=D[11]⊕D[12]⊕D[13]⊕D[14]⊕D[15]⊕D[16]⊕D[17]⊕D[18]⊕D[19]⊕D[20]⊕D[21]⊕D[22]⊕D[23]⊕D[24]⊕D[25]；

P[5]=D[26]⊕D[27]⊕D[28]⊕D[29]⊕D[30]⊕D[31]；

将以上获得的 6 位校验码插入信息码的 2^i 位置，然后对所获得的 38 位数据进行 CRC 编码。本研究所采用的 CRC12 生成多项式[51]为：

$$x^{12}+x^{11}+x^3+x^2+x+1 \qquad (3.7)$$

由于 CRC 的编码过程实际是对信息位进行求余操作，因此可以通过一个除法电路实现。根据以上多项式，可知其对应的除法电路如下：

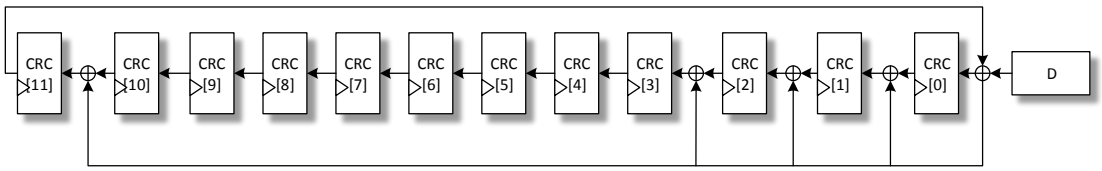


图 3.29 CRC12 对应的编码电路

以上电路是一个串行电路，对 38 位数据进行完全编码需要 38 个周期，显然不适合用于片上存储。因此，需要将其转化为并行电路，通过组合电路实现单周期输出。具体的方法是，根据串行电路算法，推导出 38 个周期后每个寄存器 Q 端的函数式。第一个时钟上升沿到来时（寄存器初始值为 0）：

CRC[0]=D[37]；CRC[1]=D[37]；CRC[2]=D[37]；……；CRC[11]=D[37]；

第二个时钟上升沿到来时：

CRC[0]=D[36]⊕D[37]；CRC[1]=D[36]；……；CRC[11]= D[36]⊕D[37]；

以此类推，第三十八个时钟上升沿到来时：

CRC[0]=D[0]⊕D[1]⊕D[2]⊕D[3]⊕D[4]⊕D[5]⊕D[6]⊕D[7]⊕D[8]⊕D[11]⊕D[12]⊕D[13]⊕D[14]⊕D[15]⊕D[16]⊕D[17]⊕D[22]⊕D[23]⊕D[24]⊕D[25]⊕D[26]⊕D[29]⊕D[30]⊕D[33]⊕D[34]⊕D[35]；

CRC[1]=D[0]⊕D[9]⊕D[11]⊕D[18]⊕D[22]⊕D[27]⊕D[29]⊕D[31]⊕D[33]⊕D[36]；

……；

CRC[10]=D[7]⊕D[9]⊕D[16]⊕D[20]⊕D[25]⊕D[27]⊕D[29]⊕D[31]⊕D[34]；

CRC[11]=D[0]⊕D[1]⊕D[2]⊕D[3]⊕D[4]⊕D[5]⊕D[6]⊕D[7]⊕D[10]⊕D[11]⊕D[12]⊕D[13]⊕D[14]⊕D[15]⊕D[16]⊕D[21]⊕D[22]⊕D[23]⊕D[24]⊕D[25]⊕D[28]⊕D[29]⊕D[32]⊕D[33]⊕D[34]；

由于涉及到的周期数较多，靠人工推导比较麻烦，并且可能出错。因此，可以编写相关的脚本文件帮助计算并给出逻辑表达式。

经过以上编码，32 位的信息位经检纠错编码后，增加了 18 位校验位。当编码后的数据从存储中读出时，需要对编码后的数据进行译码。译码过程如图 3.30 所示：

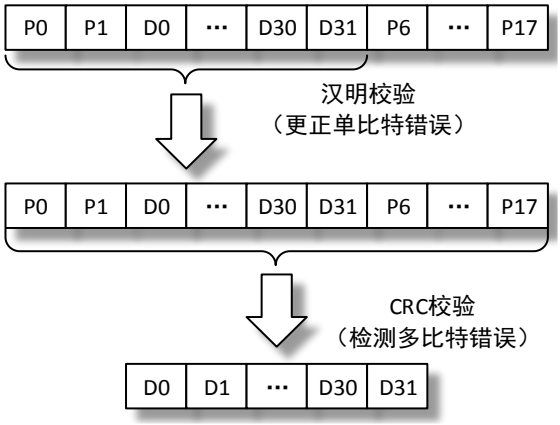


图 3.30 检纠错译码过程示意图

译码时，首先对 50 位数据中的前 38 位进行汉明译码。根据前面章节可知，译码时，需要对原 32 位信息位进行重校验。假设从存储中读出的数据为 D_NEW，重校验获得的校验值为 P_NEW：

P_NEW[0]=D_NEW[2]⊕D_NEW[4]⊕D_NEW[6]⊕D_NEW[8]⊕D_NEW[10]⊕D_NEW[12]⊕D_NEW[14]⊕D_NEW[16]⊕D_NEW[18]⊕D_NEW[20]⊕D_NEW[22]⊕D_NEW[24]⊕D_NEW[26]⊕D_NEW[28]⊕D_NEW[30]⊕D_NEW[32]⊕D_NEW[34]⊕D_NEW[36];

P_NEW[1]=D_NEW[2]⊕D_NEW[5]⊕D_NEW[6]⊕D_NEW[9]⊕D_NEW[10]⊕D_NEW[13]⊕D_NEW[14]⊕D_NEW[17]⊕D_NEW[18]⊕D_NEW[21]⊕D_NEW[22]⊕D_NEW[25]⊕D_NEW[26]⊕D_NEW[29]⊕D_NEW[30]⊕D_NEW[33]⊕D_NEW[34]⊕D_NEW[37];

P_NEW[2]=D_NEW[4]⊕D_NEW[5]⊕D_NEW[6]⊕D_NEW[11]⊕D_NEW[12]⊕D_NEW[13]⊕D_NEW[14]⊕D_NEW[19]⊕D_NEW[20]⊕D_NEW[21]⊕D_NEW[22]⊕D_NEW[27]⊕D_NEW[28]⊕D_NEW[29]⊕D_NEW[30]⊕D_NEW[35]⊕D_NEW[36]⊕D_NEW[37];

P_NEW[3]=D_NEW[8]⊕D_NEW[9]⊕D_NEW[10]⊕D_NEW[11]⊕D_NEW[12]⊕D_NEW[13]⊕D_NEW[14]⊕D_NEW[23]⊕D_NEW[24]⊕D_NEW[25]⊕D_NEW[26]⊕D_NEW[27]⊕D_NEW[28]⊕D_NEW[29]⊕D_NEW[30];

P_NEW[4]=D_NEW[16]⊕D_NEW[17]⊕D_NEW[18]⊕D_NEW[19]⊕D_NEW[20]⊕D_NEW[21]⊕D_NEW[22]⊕D_NEW[23]⊕D_NEW[24]⊕

D_NEW[25]⊕D_NEW[26]⊕D_NEW[27]⊕D_NEW[28]⊕D_NEW[29]⊕D_NEW[30]；

P_NEW[5]=D_NEW[32]⊕D_NEW[33]⊕D_NEW[34]⊕D_NEW[35]⊕D_NEW[36]⊕D_NEW[37]；

将 P_NEW 与读出数据中附带的 P 进行异或获得校正子 S：

S[0]=P_NEW[0]⊕P[0]；S[1]=P_NEW[1]⊕P[1]；S[2]=P_NEW[2]⊕P[2]；

S[3]=P_NEW[3]⊕P[3]；S[4]=P_NEW[4]⊕P[4]；S[5]=P_NEW[5]⊕P[5]；

如前所述，校正子 S 等于 0，则说明当前未发生错误；若 S 不等于 0，则说明存储中的前 38 位数据发生了错误，错误的位置即校正子的值。因此，错误纠正可以通过以下逻辑电路完成：

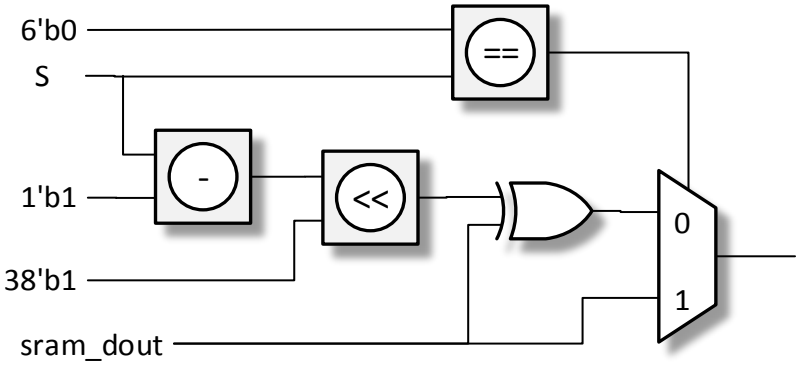


图 3.31 错误纠正电路结构图

因此，若读出数据中的前 38 位发生了单比特错误，可以被纠正过来。然而，若当前的微控制器处于恶意攻击下，存储中可能发生多比特翻转。此时，经汉明译码后，非但不能纠正错误，还可能产生更多错误。因此，接下来进行 CRC 的译码，对经校正过的数据进行错误检测。CRC 的译码过程如前所述，需对编码后的 50 位数据进行多项式除法。与编码电路有所区别，译码电路中异或门中的其中一端来自最后一级寄存器，而非输入与最后一级寄存器的异或值。译码电路如下：

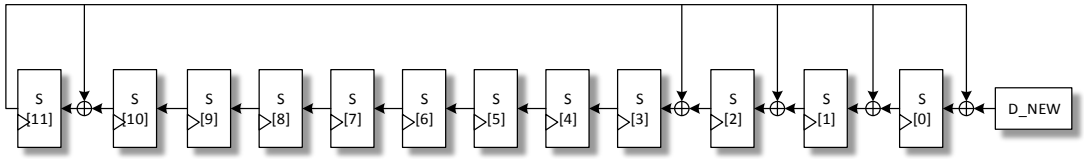


图 3.32 CRC 译码电路

经过 50 个周期后，所有数据都移入寄存器中时，电路中寄存器的内容即为校正子。若将此过程并行化，则第一个时钟上升沿到来时（寄存器初始值为 0）：

S[0]=D[37]；S[1]=0；S[2]=0；S[3]=0；S[4]=0；S[4]=0；……S[11]=0；

第二个时钟沿到来时：

S[0]=D[36]；S[1]=D[37]；S[2]=0；……S[11]=0；

……

第五十个时钟沿到来时：

S[0]= D_NEW[0] ⊕ D_NEW[12] ⊕ D_NEW[13] ⊕ D_NEW[14] ⊕ D_NEW[15] ⊕ D_NEW[16] ⊕ D_NEW[17] ⊕ D_NEW[18] ⊕ D_NEW[19] ⊕ D_NEW[20] ⊕ D_NEW[23] ⊕ D_NEW[24] ⊕ D_NEW[25] ⊕ D_NEW[26] ⊕ D_NEW[27] ⊕ D_NEW[28] ⊕ D_NEW[29] ⊕ D_NEW[34] ⊕ D_NEW[35] ⊕ D_NEW[36] ⊕ D_NEW[37] ⊕ D_NEW[38] ⊕ D_NEW[41] ⊕ D_NEW[42] ⊕ D_NEW[45] ⊕ D_NEW[46] ⊕ D_NEW[47]；

S[1]=D_NEW[1] ⊕ D_NEW[12] ⊕ D_NEW[21] ⊕ D_NEW[23] ⊕ D_NEW[30] ⊕ D_NEW[34] ⊕ D_NEW[39] ⊕ D_NEW[41] ⊕ D_NEW[43] ⊕ D_NEW[45] ⊕ D_NEW[48]；

……；

S[10]=D_NEW[10] ⊕ D_NEW[19] ⊕ D_NEW[21] ⊕ D_NEW[28] ⊕ D_NEW[32] ⊕ D_NEW[37] ⊕ D_NEW[39] ⊕ D_NEW[41] ⊕ D_NEW[43] ⊕ D_NEW[46]；

S[11]=D_NEW[11] ⊕ D_NEW[12] ⊕ D_NEW[13] ⊕ D_NEW[14] ⊕ D_NEW[14] ⊕ D_NEW[16] ⊕ D_NEW[17] ⊕ D_NEW[18] ⊕ D_NEW[19] ⊕ D_NEW[22] ⊕ D_NEW[23] ⊕ D_NEW[24] ⊕ D_NEW[25] ⊕ D_NEW[26] ⊕ D_NEW[27] ⊕ D_NEW[28] ⊕ D_NEW[33] ⊕ D_NEW[34] ⊕ D_NEW[35] ⊕ D_NEW[36] ⊕ D_NEW[37] ⊕ D_NEW[40] ⊕ D_NEW[41] ⊕ D_NEW[44] ⊕ D_NEW[45] ⊕ D_NEW[46]；

若生成的校正子 S 为 0，则说明本次读取数据没有发生错误，或者前 38 位数据中发生过单比特错误但已被纠正；若生成的校正子不为 0，则说明本次数据前 38 位数据中发生了多比特错误或者后 12 位数据中发生了错误，本次数据不可用。

3.3.4 仿真验证及逻辑综合

（一）仿真验证

为了验证存储器容错的检纠错功能，以片上 SRAM 为例，将检纠错模块嵌入至 SRAM 控制器接口，图 3.33 是正常无错误情况下的波形。图中在第一个光标处，处理器对 SRAM 的 0x20000000 地址发起写操作，写数据为 0x11。编码模块在当周期立即将写数据编码为 0x186f3c 并写入 SRAM。在第二个光标处，处理器对 0x20000000 地址发起读操作，从 SRAM 中读出的 0x186f3c 经译码模块后译码为 0x11 并在当周期返回至处理器。期间，错误中断 err_intr 信号始终为低。

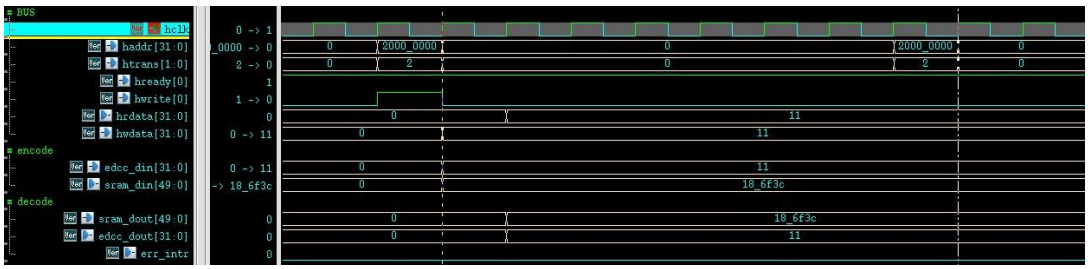


图 3.33 无错误下编译码时序图

为了模拟错误发生情况，在 Testbench 中对存储器的仿真模型进行赋值，图 3.34 是出现单比特错误下的编译码时序。同样地，在第一个光标处，处理器对 SRAM 的 0x20000000 地址发起写操作，写数据为 0x11。编码模块在当周期立即将写数据编码为 0x186f3c 并写入 SRAM。然后将仿真模型中存储的编码数据修改成 0x196f3c。在第二个光标处，处理器对 0x20000000 地址发起读操作，此时读出的数据为 0x196f3c，但经汉明纠错后，最终的译码结果仍为 0x11。期间，由于错误得到了纠正，错误中断 err_intr 信号仍保持为低。

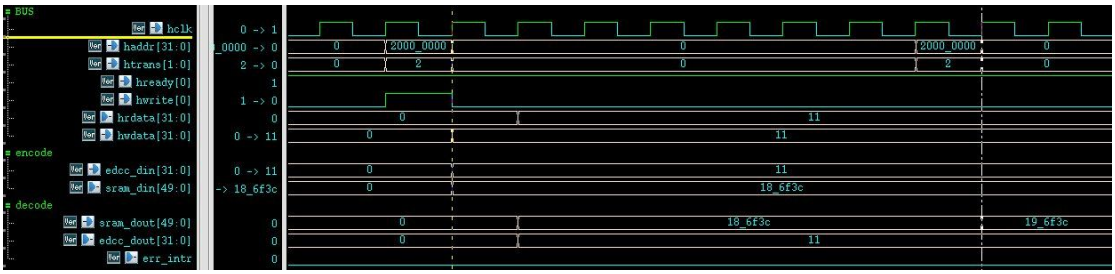


图 3.34 单比特错误下编译码时序图

图 3.35 是出现多比特错误下的编译码时序，与单比特错误仿真流程相同，只不过在仿真模型中将编码后的数据修改成了 0x750d3c，此时存储数据发生了多比特翻转。当处理器在第二个光标处对 0x20000000 地址发起读操作时，由于

单比特纠正后的数据仍检测到错误，错误中断 err_intr 拉高，并且输出数据因访问失败，修改为 0。

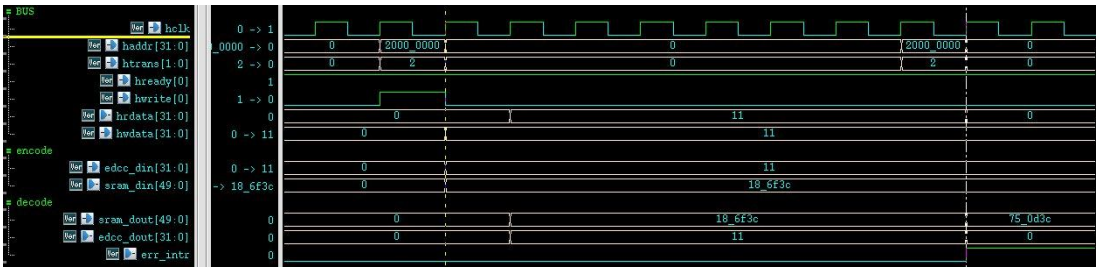


图 3.35 多比特错误下编译码时序图

（二）逻辑综合

为了对检纠错模块进行性能评估，使用 Design Compiler 对检纠错模块进行综合。采用中芯国际的 55nm 工艺以及电压为 1.08V，温度为-40℃的工艺角，检纠错模块综合后的各项指标如表 3.9 所示。从表中数据可知，检纠错模块的面积占用不多，在校验位相同的情况下，其等效门数仅为检二纠四 BCH[52]的 1/10。其次，关键路径延迟较小，最高支持 460M 的微控制器频率。对于本研究中的微控制器来说，能够轻松支持 100M 以下的单周期编译码输出。

表 3.9 检纠错模块综合后的指标

指标	检纠错模块
总面积（166M 时钟约束下）	$2133.04um^2$
等效门数（166M 时钟约束下）	1904
功耗（166M 时钟约束下）	0.268mW
最大时钟频率	460M

3.4 本章小结

本章详细介绍了微控制器安全容错技术的实现，首先是基于全硬件的 Lockstep 容错模块，容错模块主要分为故障检测，故障恢复以及故障隔离三个模块，故障检测将处理器内外部信号拉出进行实时对比和状态保存，并通过硬件复位和状态重置完成故障恢复，保证了多种类型故障的快速恢复。故障隔离模块分别针对存储器，外围接口或系统 IP 以及缓存设计了不同的隔离方案，高效隔离

了故障在系统内的传播，并解决了写通模式下的缓存容错。其次介绍了基于双核容错的处理器功耗隐藏模块，功耗隐藏模块通过在主从处理器间插入随机延时，实现时间和振幅维度上的功耗隐藏，并基于同步运行和异步运行的方式在完成功耗隐藏的同时保持容错功能。最后介绍了存储器检纠错模块的实现，介绍了各类检纠错码的编码方式，并通过对多种检错码进行漏检率分析，选择了 CRC 作为检错码。在介绍各个设计的同时，还编写了相关的用例，针对不同设计进行了相应的功能仿真，确保功能的正确性，并对具体设计完成逻辑综合，分析综合后的各项指标。

4 可靠性与安全性测试

4.1 容错性能测试

存储器的容错性能可以直接通过检纠错码本身的特性确定，但对于处理器来说，由于其功能结构的复杂性，难以通过简单的公式进行描述。仿真验证中典型错误的覆盖也是只针对单个功能点的容错说明，并不能对处理器的容错性能进行评估。因此，为了探究处理器容错方案的实际可用性，本研究通过搭建故障注入平台，向处理器中注入大量的随机错误，模拟真实环境中的故障发生，并根据最后的实验结果，结合目前常用处理器容错技术的各项性能指标，对本文提出的处理器容错技术进行综合评估。

4.1.1 故障注入技术分析

目前，对处理器进行故障注入的方法有很多，不同的方法在成本，时间，可控性及可观察性上都有所不同。从注入类型上主要可以分为硬件故障注入、软件故障注入以及仿真故障注入。

硬件故障注入是指通过一些物理上的攻击手段，对芯片实体进行故障注入。常用且成本较低的方法有电磁故障注入，电源及时钟抖动等方法，但是这类故障注入的可控性及可观察性不好，难以精确控制故障注入的位置和时间，不利于结果分析；一些高成本的故障注入方式如激光注入，聚焦离子束等，虽然能够精确控制故障注入位置，但其设备十分昂贵，并且可能对芯片造成损伤。此外，硬件故障注入的目标是实物，需要等样片流片回来或使用 FPGA，这就限制了硬件故障注入在设计初期进行容错评估上的使用。

软件故障注入是指程序在编译时或者运行时动态修改处理器内部的状态，从而模拟故障的产生。这种方式不需要任何的硬件设备，具有良好的可控性和可观察行，且成本低。然而，由于软件只能修改处理器对软件可见的寄存器资源，无法控制处理器内部不可见的电路，难以模拟实际的故障情况，因此，其对容错能力的评估往往与实际结果有着较大差距。

仿真故障注入是指通过仿真工具，在精确的时间，对处理器内部的信号进行

修改，从而模拟故障注入。这种故障注入方式具有良好的可控性和可观察性，并且由于可以访问 RTL 级中的所有信号，因此具有较为全面的故障覆盖情况。但其缺点是仿真带来的时间开销大，需要尽可能快速自动化地完成故障的注入以及故障恢复的自检，以减少测试时间。

4.1.2 仿真故障注入平台设计

考虑到测试的开销以及效果，本研究采用仿真故障注入对嵌入式处理器的容错性能进行测试。为了进行可靠的容错测试，故障注入点需要仔细考虑。根据前文所述，处理器内部的组合逻辑错误，若要产生显式的运行故障，一定会被寄存器采样并传递。而处理器接口上的信号出错，则会传递至系统寄存器以及缓存中。因此，本研究将处理器内部所有寄存器以及接口上的输出信号作为故障注入点。为了确保错误能在内部传递，提高有效错误率，每个故障注入的时间为两个系统时钟周期。故障的注入可以由 Verilog 的 force 语句完成，force 语句能够强制翻转任意路径的信号。一个简单的单比特故障注入方式为：在 Testbench 中，延迟随机时间，然后强制某一信号与 1 进行按位异或，两个周期后，释放该信号，则一个单比特故障注入完成。然而进行大量随机故障注入测试时，无法手动完成以上过程，需要借助自动化平台的完成。本研究所搭建的故障注入的平台如图 4.1 所示。

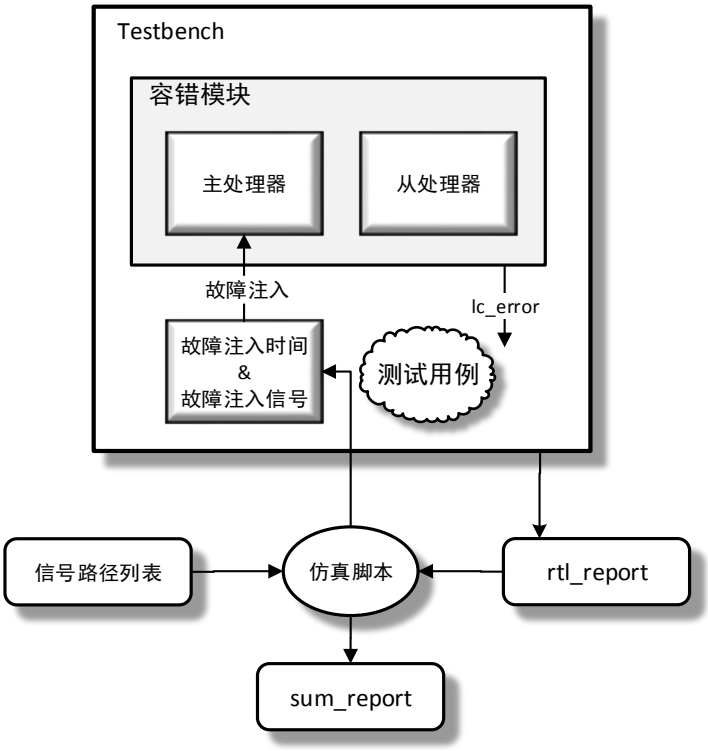


图 4.1 故障注入平台结构框图

首先，通过编写相关的脚本文件，对处理器 RTL 文件进行处理，将各个层级模块中的寄存器信号，总线接口的输出信号以及缓存接口提取出来，并将对应的信号路径写入到信号路径列表文本中。其次，在 Testbench 中，使用宏代替具体的延迟时间以及故障注入信号，当运行仿真脚本进行连续仿真执行时，在不同的随机种子下，每次仿真时都会生成随机的延迟时间，并在信号列表路径中随机地选择信号。随机时间和随机信号会以宏定义的方式传递到 Testbench 中，从而实现自动化的随机故障注入。

由于随机选择处理器内部信号进行故障注入，有些信号的强制翻转可能不会被传递并导致显式的故障出现，这种故障形式可认为是一种无效的故障注入，为了去除无效故障注入的影响，仿真过程中，Testbench 会监测 lc_error 信号，以判断每次故障注入是否有效，一旦检测到本次仿真 lc_error 拉高，则认为本次仿真故障注入有效；若直至程序顺利执行完毕也没有检测到 lc_error 拉高，则认为本次仿真故障注入无效。单次仿真的报告结果会生成到 rtl_report 中，仿真脚本最终会对每个仿真的 rtl_report 进行解析，将每个用例的测试结果，执行周期进行整理，写入至总的容错性能报告 sum_report。

4.1.3 实验结果与对比分析

（一）实验结果

本研究从CK803中提取到的内部寄存器以及外部输出接口信号将近2000个，以一次注入一个单比特错误为例，图 4.2 中是随机注入5000个错误的仿真结果。图中 BLOCK 栏为测试模块，该模块为全硬件 Lockstep 容错保护下的处理器模块。PATTERN 栏为测试用例，为了较为全面地对容错功能点进行覆盖，该测试用例中的指令操作包括读存储及外设，读缓存，写存储及外设，写缓存，数据运算及指令跳转。SEED 栏为每个测试用例的随机种子，不同的随机种子保证了每个用例的故障信号以及故障发生时间的随机性。RESULT 栏显示当前用例的测试结果，结果为 FAILED 表示处理器发生故障，并且故障没有得到恢复。PASSED 表示当前用例执行过程中检测到了故障的发生，但故障被成功恢复。WARNED 表示当前用例执行过程中没有检测到故障的发生，此次故障注入为无效注入。通过 RESULT 栏显示的结果可以计算出容错设计的容错率。SIM_SYCLE 栏表示单个测试用例所用时钟周期，结合测试结果可用来计算平均故障恢复时间。

	BLOCK	PATTERN	SEED	RESULT	SIM_CYCLE
1	BLOCK	PATTERN	SEED	RESULT	SIM_CYCLE
2	==========	==========	==========	==========	==========
3	LC_FT	lc_ft_test.rs	128557597	****WARNED****	2295
4	LC_FT	lc_ft_test.rs	471241413	****WARNED****	2295
5	LC_FT	lc_ft_test.rs	1469888884	****WARNED****	2295
6	LC_FT	lc_ft_test.rs	584003805	****WARNED****	2295
7	LC_FT	lc_ft_test.rs	463417553	****WARNED****	2295
8	LC_FT	lc_ft_test.rs	1267636019	passed	2323
9	LC_FT	lc_ft_test.rs	1412044586	passed	2315
10	LC_FT	lc_ft_test.rs	179763602	passed	2315
11	LC_FT	lc_ft_test.rs	788605551	passed	2316
12	+--4987 lines: LC_FT	lc_ft_test.rs	519023601	****WARNED****	
4999	LC_FT	lc_ft_test.rs	146221642	passed	2316
5000	LC_FT	lc_ft_test.rs	972722527	****WARNED****	2295
5001	LC_FT	lc_ft_test.rs	1304480867	passed	2314
5002	LC_FT	lc_ft_test.rs	412567524	passed	2315
5003	==========	==========	==========	==========	==========
5004	PATTERN_NUM	PASSED	FAILED	WARNED	SUM_CYCLES
5005	5000	1617	98	3285	12412085

图 4.2 sum_report 内容

从图中的结果可以看出，在5000次故障注入中，有超过60%的故障是无效故障，这些故障最终没有导致显式的错误。在有效错误中，故障顺利恢复的比例达到了94.3%。剩下没有正常恢复的用例，大部分是由于处理器及缓存仿真模型内有个别寄存器没有进行复位初始化，导致在一些特殊情况下，故障无法正确恢复。此外，在不发生错误正常执行下，测试用例需要2295个周期，当发生错误需要进行故障恢复时，则需要更长的时钟周期，例如，SEED为179763602的用例在出现故障时多使用了20个时钟周期。

除单次注入 1 个错误外，本研究对多个故障同时出现的情况也做了相关的故障测试，分别为单次注入 2 个错误，单次注入 5 个错误和单次注入 10 个错误。同样地，每种情况下进行 5000 次随机仿真，各种情况下的故障恢复率及故障恢复时间如表 4.1 所示。

表 4.1 单比特及多比特故障下的故障恢复情况

单次故障注入数	PASS	FAIL	WARN	故障恢复率	故障恢复时间
1	1617	98	3285	94.3%	22 个周期
2	2738	213	2049	92.7%	22 个周期
5	4039	397	564	91.0%	23 个周期
10	4331	556	113	88.6%	23 个周期

从实验结果看来，单次注入的错误越多，无效故障数越低，但故障恢复率也越低，当同时注入 10 个故障时，故障恢复率降低到 88%左右。这主要是部分无法恢复的故障随着故障注入数量的增多，出现的概率越大，从而导致了整体故障恢复率的降低。在故障恢复时间方面，无论单次注入多少故障，故障恢复时间始终保持在 20 多个周期左右，对于 25M 以上的微控制器来说，该故障响应速度在微秒以内，具有极高的实时性。

（二）对比分析

为了对本文提出的全硬件 Lockstep 处理器容错技术（以下简称 HLC）进行各项指标的综合评估，本文以现有常用的 TMR 处理器容错技术，以及基于检查点的 Lockstep 处理器容错技术（以下简称 CLC）作为对比，对三种处理器容错技术分别在面积、容错率、性能及故障恢复时间上进行评估。

为了进行合理的指标对比，本文实现了基于 CK803 的 TMR 容错技术，TMR 及本文提出的 HLC 容错的各项指标将通过上文的仿真故障注入平台测试得出。为了方便与 CLC 技术[17]的相关指标做对比，在进行指标测试时，采用相同的基准程序 Matrix，该程序用于计算两个 10x10 大小的整数矩阵之积。同时，为保证故障注入方式一致，进行故障注入测试时，只对处理器流水线寄存器以及内部控制状态寄存器进行单比特故障注入[17]。进行对比的 CLC 技术，其检查频率为每 300 个写操作设置一次检查点[17]。由于作为对比的 CLC 技术采用 Leon 处理器，

与本文采用的处理器不同，因此在各指标进行对比时，需要对各指标进行合理的定义。

- 面积

该指标用于表示各个容错方案的面积开销，由于不同处理器的初始面积不同，而三种容错方案都进行了硬件冗余，因此，为了合理体现各容错技术所带来的面积开销，在对相应设计进行逻辑综合后，分别计算处理器容错前后的面积比，并将该比值作为面积开销的评价指标，如式 4.1 所示，比值越高说明该容错方案带来的面积开销越大。

$$面积 = \frac{容错后面积}{容错前面积} \tag{4.1}$$

- 容错率

该指标用于表示各个容错方案的容错能力，为了与 CLC 技术[17]对比，在进行单比特故障注入后，分别计算容错处理器运行基准程序时未出现显式错误或错误被恢复，即程序被正确执行的占比，并将该比值作为容错能力的评价指标，如式 4.2 所示，比值越高说明该容错方案的容错能力越好。

$$容错率 = \frac{基准程序正确执行个数}{基准程序总执行个数} \tag{4.2}$$

- 性能

该指标用于表示各个容错方案无故障发生时对性能的影响，为了减小不同处理器计算能力不同的影响，分别计算处理器容错前后执行基准程序所需的时钟周期比，并将该比值作为性能开销的评价指标，如式 4.3 所示，比值越高说明该容错方案带来的性能开销越小。

$$性能 = \frac{容错前执行基准程序所需时钟周期}{容错后执行基准程序所需时钟周期} \tag{4.3}$$

- 故障恢复时间

故障恢复时间用于表示各个容错方案故障发生后进行故障恢复所需要的时间，由于不同的处理器执行指令的情况不同，为了便于比较，分别计算执行基准程序时，发生单个故障所需执行周期相比无故障发生所需执行周期多出的比值，

并将该比值作为故障恢复时间的评价指标，如式 4.4 所示，比值越高说明该容错方案所需故障恢复时间越长。

$$故障恢复时间 = \frac{单故障发生时执行周期 - 无故障发生时执行周期}{无故障发生时执行周期} \quad (4.4)$$

图 4.3 是这三种处理器容错方案的各项指标对比。为了便于观察分析，对各项指标的纵坐标进行了归一化处理，以每项指标中的最大值作为基准，将另外两个数值映射在[0,1]之间，进行比较。

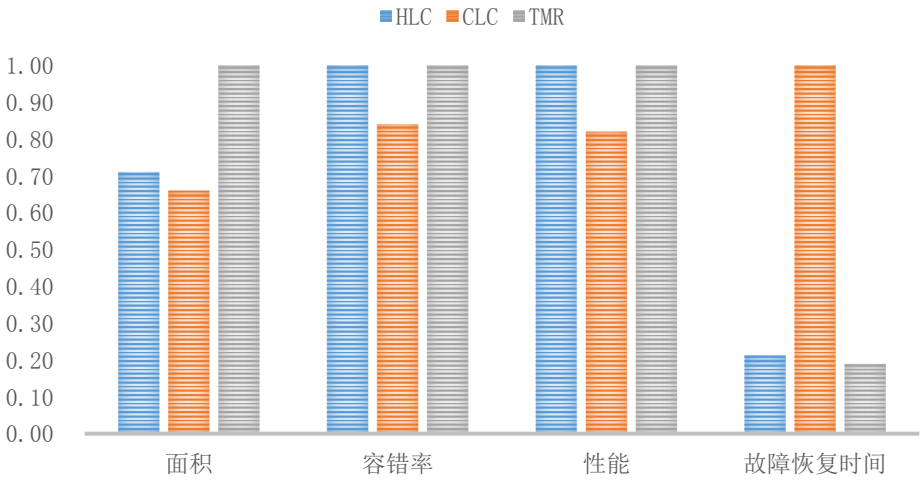


图 4.3 HCL、CLC、TMR 三种容错方案指标对比

从图中各个指标对比的情况可知，在面积开销上，由于 HLC 不需要对处理器进行三倍冗余，所以其面积开销远低于 TMR 所需面积开销；与 CLC 相比，由于增加了额外的故障恢复逻辑，HLC 面积开销虽略有增加，但增幅不大，约为 CLC 的 7%左右。在容错率上，由于 HLC 在硬件上对处理器内部所有控制状态寄存器进行监控和恢复，因此，其容错率与高可靠的 TMR 相当；与 CLC 相比，由于 CLC 无法实时检测故障，且只能恢复处理器内部对软件可见的寄存器，因此 HLC 的容错率高于 CLC，约为 CLC 的 119%，此外，由于 CLC 在硬件上只进行了简单的接口信号对比，没有对处理器的写操作进行隔离，因此 CLC 的实际故障恢复能力会更低。在性能上，当处理器只进行存储和缓存访问时，HLC 与无性能损失的 TMR 性能相当；与 CLC 相比，其性能损失远低于 CLC，这主要是由于 HLC 可以直接在硬件上实时获取所需的状态恢复节点，而 CLC 需要在程序正常运行阶段间或进入中断服务程序进行节点保存。在故障恢复时间上，由于

HLC 在进行故障恢复时，对处理器复位后，状态会回滚至上一个状态，而不是当前状态，因此，相比 TMR 其故障恢复时间略长；相比 CLC 来说，HLC 的故障恢复时间远小于 CLC 所需故障恢复时间，约为 CLC 的 20%。这主要是因为 HLC 是在硬件上进行故障恢复，CLC 是通过软件恢复状态节点，状态的恢复需要执行完回滚程序中的指令，并且还需从上一个正确节点执行到当前故障节点。

由于上述基准测试程序只涉及到存储及缓存的访问，而本文提出的 HLC 容错方案在对处理器进行故障隔离时，针对外设的访问隔离通过延迟操作实现，因此在进行写外设操作时，存在一定的性能损失。针对这一情况，本文以无性能损失的 TMR 容错技术作进一步的性能对比，以本微控制器中包含的几个主要外设作为测试对象，通过执行相同的测试用例，分别对比 HLC 与 TMR 两种容错方案下处理器访问外设时的性能大小，表 4.2 为各个外设对应的测试用例的具体说明。

表 4.2 各外设对应的测试用例说明

测试对象	测试用例说明
SPIS	配置使能 SPIS（Serial Peripheral Interface Slave，串行外设接口从设备），开启 DMA 数据搬运功能，与 SPI 主设备通信，发送和接收 10 个字节数据。
SPIM	配置使能 SPIM（Serial Peripheral Interface Master，串行外设接口主设备），开启 DMA 数据搬运功能，与 SPI 从设备通信，发送和接收 10 个字节数据。
I2C	配置使能 I2C（Inter-Integrated Circuit，集成电路总线），开启 7bit 地址、从机模式，与另一 I2C 主机通信，使其在标准模式下发送和接收 5 个字节数据。
UART	配置使能 UART（Universal Asynchronous Receiver/Transmitter，通用异步收发传输器），开启非自动流控、全双工模式，与另一 UART 通信，使其在波特率为 9600 bit/s 下发送和接收 5 个字节数据。

以上测试用例为各外设主要工作模式下的数据收发情况，其中包括 DMA 协助搬运数据的情况。同样以上文定义的性能指标进行计算，图 4.4 是 HLC 与 TMR

容错方案中处理器进行外设访问时的性能对比图。

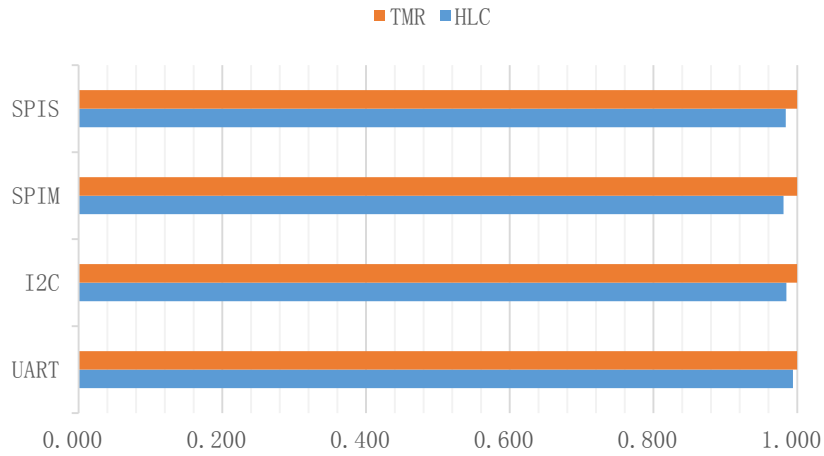


图 4.4 HLC 与 TMR 容错下处理器进行外设访问的性能对比

从图中对比情况可知，在进行有关外设的访问时，HLC 相比 TMR 存在一定的性能损失，其性能约为 TMR 的 98%左右。这主要是由于 HLC 容错方案中针对外设进行了故障隔离，处理器对外设进行写操作时存在延迟，因而降低了整个微控制器的运行效率。但对微控制器来说，外设的配置在设置完成后通常较少变更，进行大量数据传输时，通常使用 DMA 协助搬运数据，处理器无需直接对外设进行写操作，所以实际性能损失会更小，因此，HLC 中针对外设的故障隔离设计并不会影响微控制器的整体性能。

综上，本研究提出的基于全硬件的 Lockstep 技术相比基于检查点的 Lockstep 技术虽然增加了少许面积，但其在容错率，性能，故障恢复时间上具有十分明显的优势；相比 TMR 技术来说，在大幅降低面积开销的同时，在容错率、性能、故障恢复时间上与 TMR 近乎一致。因此，基于全硬件的 Lockstep 技术在具备优良的可靠性与实时性的同时，降低了面积成本，适合用于低成本，实时性要求高的工业微控制器。

4.2 抗功耗分析攻击性能测试

本研究对微控制器进行了功耗隐藏设计，能够在微控制器执行软件加密时，在时间维度及振幅维度上进行功耗隐藏，增加攻击者的攻击成本。为了证实抗功耗攻击的有效性，本研究搭建了 DPA 攻击平台，以软件 AES 加密为例，分别在

关闭及打开功耗隐藏的情况下，对处理器进行 DPA 攻击，并分析最终的攻击结果。

4.2.1 DPA 攻击平台设计与实现

DPA 攻击平台主要分为两个部分，功耗曲线收集模块和数据分析模块。功耗曲线收集模块用于获取处理器执行软件加密时，不同明文下相同时间段内的功耗曲线。数据分析模块用于将获取到的功耗曲线及对应明文进行相关的计算分析，猜测密钥。

（一）功耗曲线收集模块

功耗曲线的收集主要有两种方式，一种是将芯片的电源端口串接电阻，通过示波器测量电阻端口上的电压获取功耗曲线；另一种是通过 EDA 工具，对 RTL 或网表文件进行功耗仿真获取功耗曲线。相对于前一种方法，使用 EDA 工具进行功耗仿真，一方面可以在设计初期就进行抗攻击测试，提前进行抗攻击评估；另一方面仿真所得的功耗数据，噪声小、功耗分析范围可控，可以用更少的功耗曲线，更快分析出密钥。因此，本研究采用第二种方式，通过 EDA 工具进行功耗仿真。使用的 EDA 工具为 Prime Time PX，该仿真软件是 Synopsis 公司开发的一款用于对 RTL 或网表文件进行功耗仿真的工具。其中的 time_based 模式能够根据标准库文件中的功耗信息以及信号翻转波形文件等计算出当前设计运行过程中的瞬时功耗。整个功耗曲线收集的流程如图 4.5 所示：

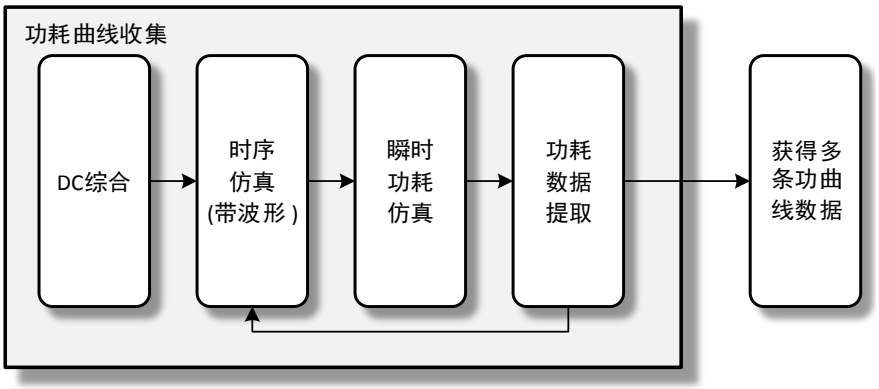


图 4.5 功耗曲线收集流程图

第一步：DC 综合。通过 DC 对设计好的文件进行综合，获得所需的网表 v 文件，时序反标 sdf 文件，设计约束 sdc 文件以及寄生电容 spef 文件。

第二步：时序仿真。这里以 128-AES 作为待攻击的软件加密算法，AES 的算法如图 4.6 所示，每一轮运算中包含字节替换，行移位，列混淆以及轮密钥加，这样的运算共进行十次。除了第十轮中的运算没有列混淆，其他操作十轮一致。为了加快密钥攻击进程，本研究将攻击点选为第一轮字节替换的输出处，并在测试用例中将密钥固定为{0x24, 0x15, 0x19, 0xf8, 0x39, 0x45,0x33, 0x46, 0x78, 0x35, 0x98, 0xbe, 0xb7, 0x5e, 0x29, 0xa9}，后续的攻击实验结果将以此作为正确结果对比。编译好的程序和数据将在 Testbench 中存入 EFLASH 和 SRAM 中。使用 VCS 进行时序仿真并开启波形记录，获得信号翻转 fsdb 文件。由于带波形的时序仿真耗费时间长，这里只记录攻击点附近的波形。

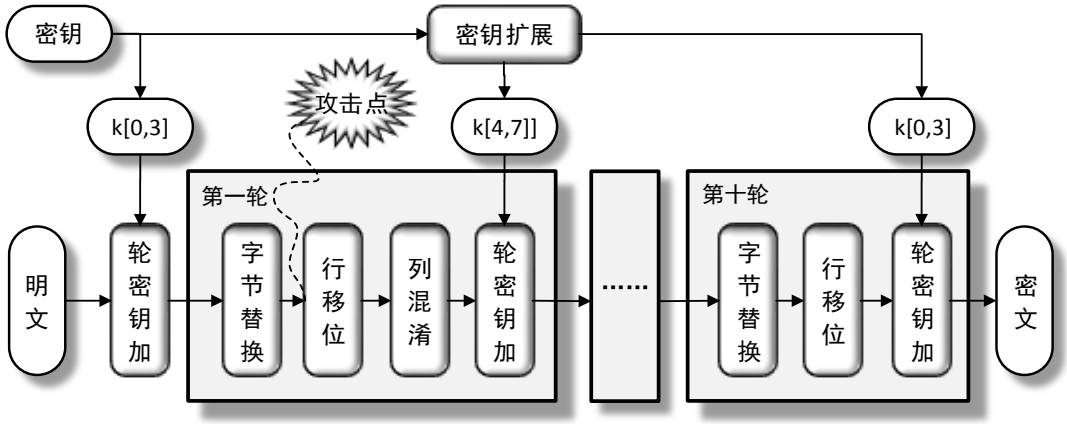


图 4.6 AES 加密算法流程图

第三步：瞬时功耗仿真。编写 Prime Time PX 的 tcl 文件，将以上获取的文件导入 Prime Time PX，进行基于 time_based 模式的功耗仿真。仿真结束后的瞬时功耗文件有两种形式一种是 fsdb 格式的波形文件，可以通过 nWave 打开查看具体的功耗波形及峰值；另一种为 out 格式的数据文件，内部以时间戳和功耗值的形式记录了每个采样点上设计的功耗大小。由于后续的数据分析需要获取具体的功耗曲线数值，因此将输出文件格式设定为 out 文件输出。

第四步：功耗数据提取。上一步中生成的 out 文件虽然包含功耗信息，但还不能直接用于后续的数据分析，需要进行预处理。图 4.7 是 out 文件的格式，每一条时间戳下都记录了整个微控制器下各个模块的瞬时功耗，为了加快攻击速度，本研究选取了处理器顶层 x_lc_core_top 的功耗信息作为攻击对象，降低了其他模块的干扰。对于 Prime Time PX 来说，若当前时刻模块的瞬时功耗与上一时刻

相同，不会列出当前时刻模块的功耗信息，在多次仿真中可能存在时间戳不一致的情况，而攻击时要求所有的功耗迹应在时间维度上对齐，因此，在提取 out 文件中有效功耗值的同时，还应补全没有列出的功耗值。

```
;! output_format 5.3
; header 116306022046
;
;                          PrimeTime-PX (TM)
;
;           Version J-2014.06-SP2 for RHEL32 -- Aug 27, 2014
;                Copyright (c) 2000-2014 Synopsys, Inc.
;                          ALL RIGHTS RESERVED
;
;
.time_resolution 0.1 ---------------------------------- 功耗曲线采样间隔，
.hier_separator /                                       单位ns
.index Pc(pp_root) 1 Pc ------------------------------
.index Pc(fuxi_top) 2 Pc ------------------------------ 整个电路
.index Pc(fuxi_top/x_core_domain) 3 Pc ---------------- 及各个模块对应的标号
.index Pc(fuxi_top/x_core_domain/x_lc_core_top) 4 Pc --
......
10140000 ---------------------------------------------- 时间戳
1  2.961e+01 ----------------------------------------
2  2.961e+01 ---------------------------------------- 各个模块在当前时间戳下
3  2.961e+01 ---------------------------------------- 对应的瞬时功耗
......
10140001 ---------------------------------------------- 时间戳
1  2.961e+01
......
; file closed at 1017496ns ----------------------------- 结束时间
; trailer 819505344
```

图 4.7 out 文件格式

（二）数据分析模块

数据分析模块基于 Matlab 实现，通过计算假设功耗与实际功耗间的相关性确定密钥。由于 AES 算法将数据划分成多个字节进行计算，因此，可以一个字节一个字节的攻击密钥，本研究以密钥的第一个字节为例进行攻击，数据分析的流程如图 4.8 所示：

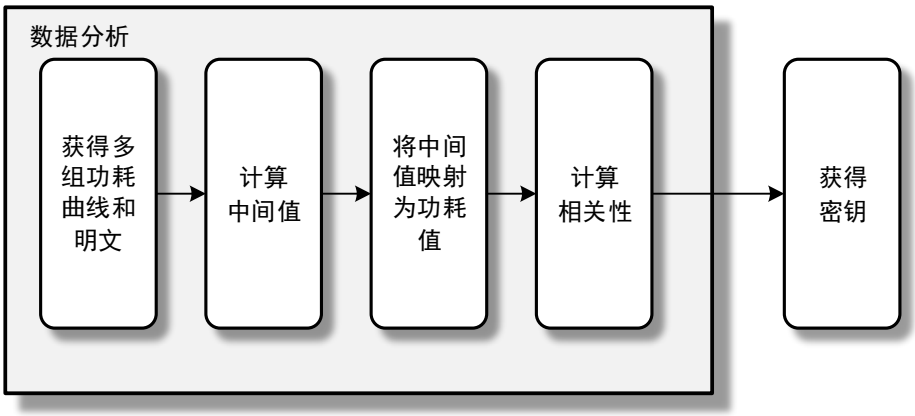


图 4.8 数据分析流程图

第一步：获取多组功耗曲线和明文。将功耗曲线收集模块中获取到的功耗曲线以及对应的明文读入 Matlab，假设功耗曲线和对应的明文有 M 个，采样点数为 T，则获得一个 MxT 的功耗曲线矩阵和 Mx1 的明文矩阵。

第二步：计算中间值。由于本研究选择的攻击点是 S-BOX 输出，假设明文为 pt，轮密钥为 k，经过 S_BOX 输出的函数为 sbox()，则根据 AES 加密算法，对应中间值为sbox(pt ^ k)，不同猜测密钥和明文组合下生成一个 Mx 256 的中间值矩阵。

第三步：将中间值映射为功耗值。由于汉明重量模型相比汉明距离模型更为简单，无需知道数据前后变化，是攻击者在无法获取网表信息时常用的选择，因此本研究选择汉明重量模型作为功耗映射模型。汉明重量模型的实质是计算操作数中比特 1 的个数，表 4.3 是以 3 比特数据为例列出了汉明重量模型下操作数与功耗值间的映射关系，映射后的数据为一个 Mx256 的假设功耗矩阵。

表 4.3 汉明重量模型

操作数	功耗值	操作数	功耗值
000	0	100	1
001	1	101	2
010	1	110	2
011	2	111	3

第四步：计算相关性。为了确定假设功耗矩阵与功耗曲线矩阵间的相关性，本研究采用相关系数法，计算假设功耗矩阵与功耗曲线矩阵列与列间的相关系数，

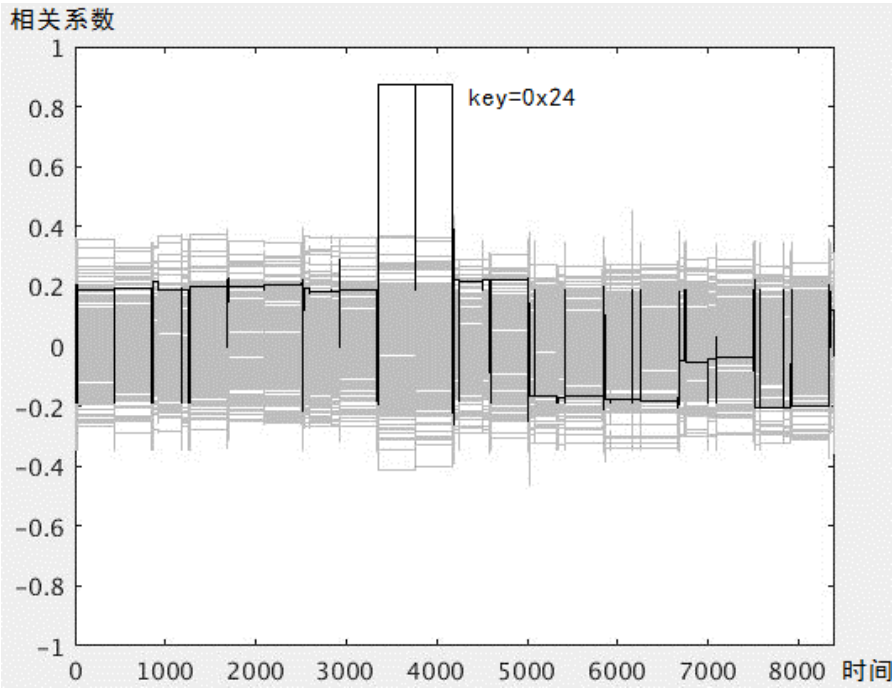
具有最大绝对值的相关系数所对应的密钥即为正确密钥。

4.2.2 实验结果

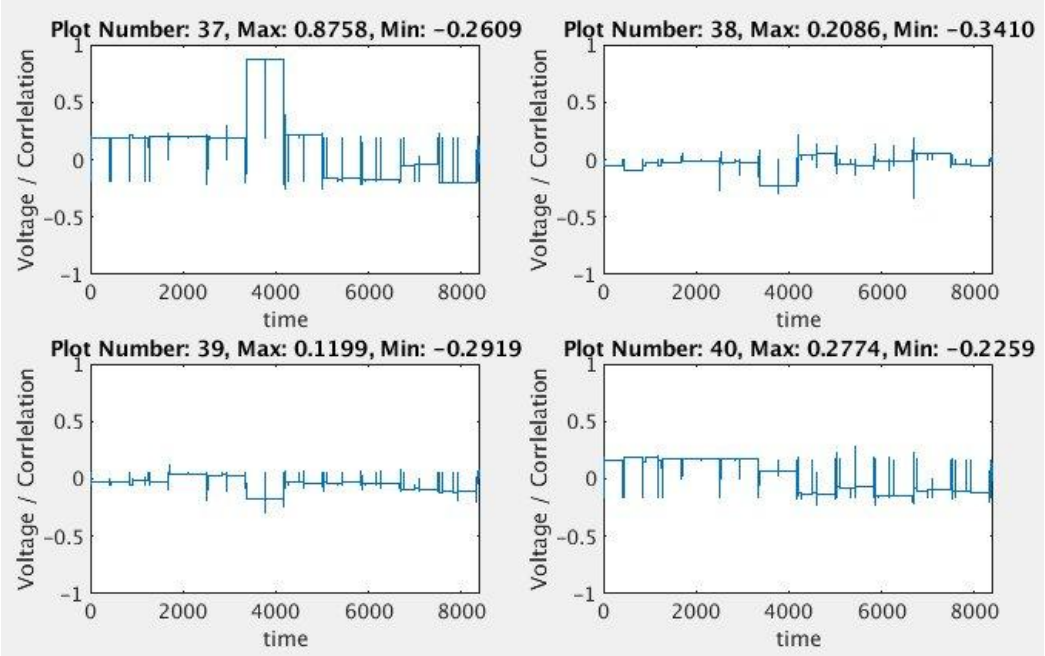
基于以上 DPA 攻击平台，对打开和关闭功耗隐藏模式分别进行 DPA 攻击测试。

（一）关闭功耗隐藏模式

主从处理器同步运行，执行 AES 加密运算。由于本研究为了加快攻击速度，只采集了攻击点附近的功耗曲线信息，并只对处理器顶层模块进行功耗采样，以排除微控制器中其他模块的干扰。因此，在采集 80 条功耗曲线时，DPA 攻击效果已经非常明显。图 4.9 是针对密钥第一个字节的攻击结果。从图中可以看出有明显的高于其他所有曲线的尖峰值出现，该尖峰所在曲线对应于第 37 条曲线，即对应于十六进制密钥 0x24，这与前文申明所使用的密钥一致，密钥猜测正确。



(a) 256 个猜测密钥对应的相关性曲线，其中 key=0x24 的曲线有明显尖峰

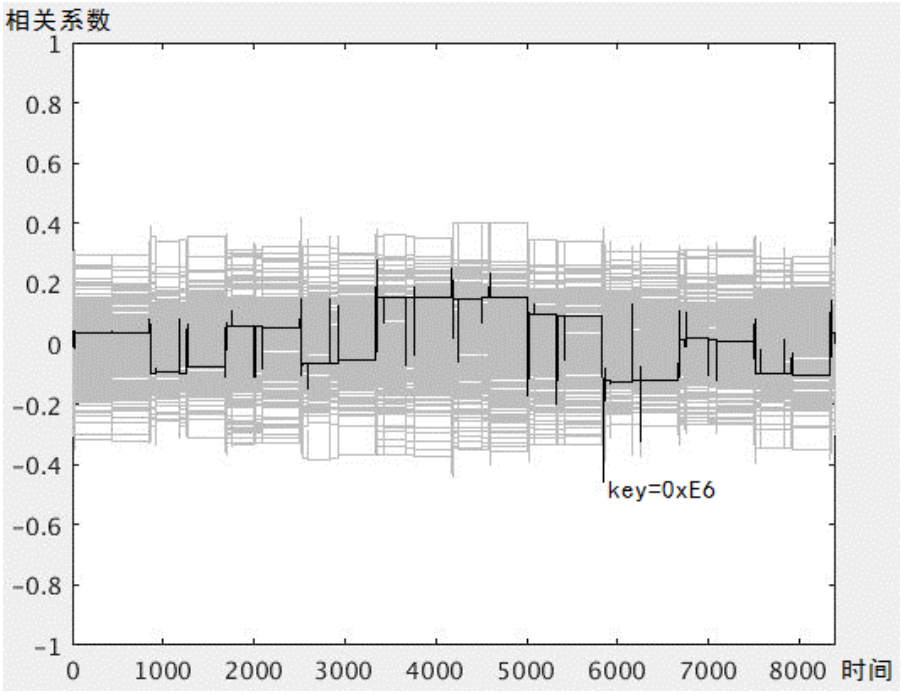


(b) 正确密钥附近的各个猜测密钥对应的相关性曲线

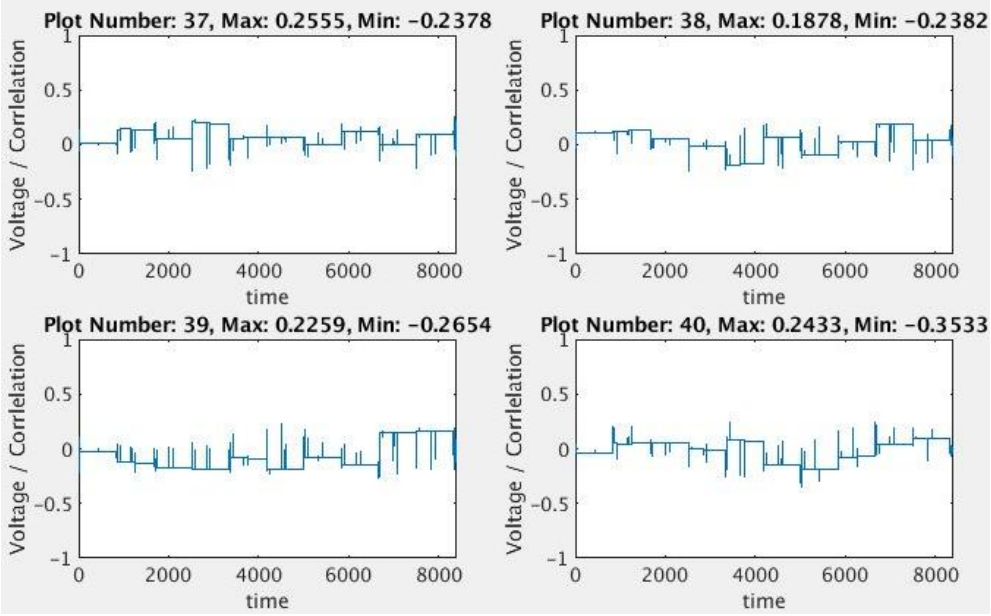
图 4.9 功耗扰乱关闭时的 DPA 攻击结果

（二）打开功耗隐藏模式

主从处理器异步运行，交错执行 AES 加密运算。再次进行以上步骤的 DPA 攻击，同样采集 80 条功耗曲线，攻击结果如图 4.10 所示。从图中可以看出，攻击平台猜测的密钥为 0xE6，与实际不符。对应密钥正确密钥 0x24 的第 37 条曲线相比较其他曲线并没有出现明显的尖峰，因此攻击失败。



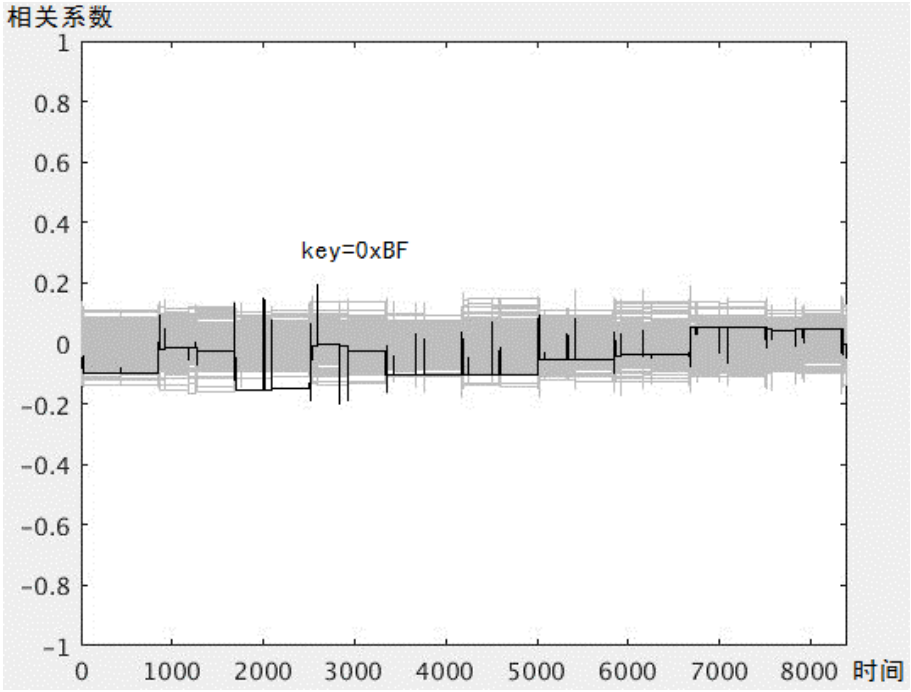
(a) 256 个猜测密钥对应的相关性曲线，其中显示 key=0xe6 的曲线相关性最高



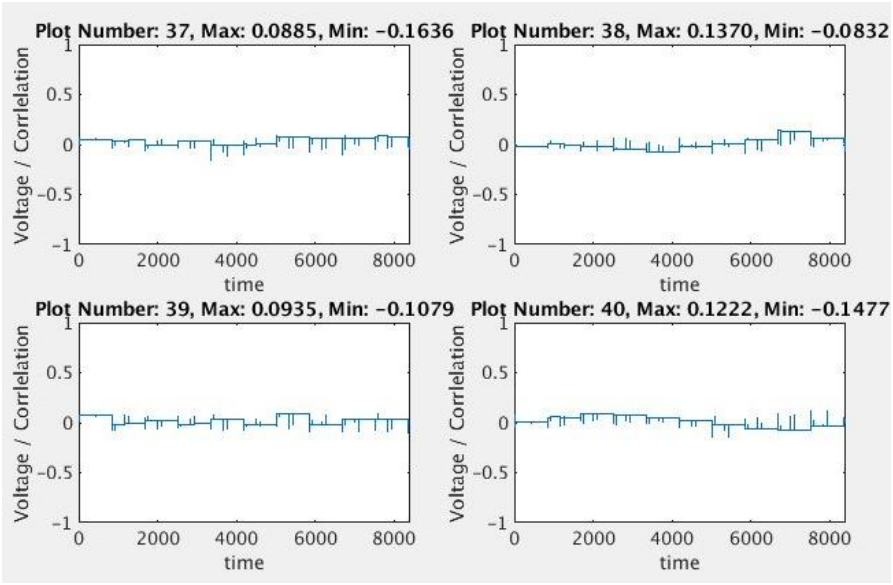
(b) 正确密钥附近的各个猜测密钥对应的相关性曲线

图 4.10 功耗扰乱开启时的 DPA 攻击结果（80 条曲线）

进一步增加功耗曲线的采样条数，图 4.11 是采样条数为 2000 条功耗曲线下的攻击结果。从图中可以看出，攻击平台猜测的密钥为 0xBF，与实际不符。对应正确密钥 0x24 的第 37 条曲线相比较其他曲线并没有出现明显的尖峰，因此攻击依旧失败。



(a) 256 个猜测密钥对应的相关性曲线，其中显示 key=0xBF 的曲线相关性最高



(b) 正确密钥附近的各个猜测密钥对应的相关性曲线

图 4.11 功耗扰乱开启时的 DPA 攻击结果（2000 条曲线）

因此，本研究所使用的处理器抗功耗分析攻击设计具有可行性，对于功耗分析攻击的抵御具有较好的效果，能够使得攻击变得更加困难，增加了微控制器的攻击成本，从而使得工业微控制器能够在低成本下获得更高的安全性。

4.3 本章小结

本章针对微控制器中的处理器进行了安全容错性能测试。首先，基于仿真故障注入的方式搭建了故障注入平台，该平台能够自动化地连续执行多次仿真，并对每次仿真注入随机故障。故障注入后，平台能检测无效的故障，并将最后的运行结果进行对比，输出最后的测试结果。从最后的实验结果来看，本研究提出的处理器容错方案与现有常见的处理器容错方案相比，在面积，容错，性能和故障恢复时间之间进行了良好的折衷。其次，基于 DPA 攻击原理搭建了基于仿真的 DPA 攻击平台，该平台能够自动化采集功耗曲线，完成对功耗信息的预处理，并对数据进行相关性分析。最后的实验结果证明，开启功耗隐藏模式后，处理器能够抵御 DPA，保护密钥不被窃取。

5 总结与展望

5.1 论文工作总结

本文主要对工业微控制器的安全容错技术进行了较为深入的研究，针对工业微控制器在安全性和可靠性上的需求，设计并实现了嵌入式处理器的容错和抗攻击技术，以及存储器的检纠错技术。具体概括如下：

（1） 介绍了工业微控制器安全容错技术的背景及研究意义，明确了对微控制器进行可靠性设计，以及对内部的处理器进行抗功耗分析攻击设计的必要性。分别调研了目前学术界和业界在高可靠微控制器，以及抗功耗分析处理器上采用的相关方案及产品，针对工业微控制器在低成本，实时性上的要求，从硬件层面出发，着手微控制器核心——内部处理器及存储器提出了相关的安全容错方案。

（2） 对基于检查点的 Lockstep 处理器容错技术进行了介绍，并针对现有方案在性能开销大，实时性不足，hang 类型故障无法解决以及不考虑 Cache 故障的情况下，提出使用硬件完成正确节点状态保存和故障恢复的全硬件 Lockstep 容错技术。在 3.1 节中对该技术方案的实现从故障检测，故障恢复及故障隔离三个方面进行了详细的介绍，同时，对设计进行了仿真验证和综合。最后，在 4.1 节中搭建了基于仿真的故障注入测试平台，并对几种常用处理器容错方案进行了对比，实验结果表明基于全硬件的 Lockstep 容错设计相比基于检查点的 Lockstep 容错设计在面积只增加 7%左右的情况下，性能提高了 22%，故障恢复时间缩短了 80%，与 TMR 技术在性能、容错率及实时性上的表现几乎一致。因此，本文提出的容错技术具有综合优势，适合用于低成本和高实时要求的工业微控制器。

（3） 介绍了 DPA 攻击原理，对功耗隐藏技术效果进行量化，并以此为基础，结合双核容错架构，提出了一种基于双核异步运行和同步运行的功耗隐藏方案。该方案能够在保持容错功能的同时，从时间维度和振幅维度对处理器泄露的功耗进行隐藏。在 3.2 节中详细介绍了异步运行和同步运行方案的实现，同时，对设计进行了仿真验证和综合。在 4.2 节中搭建了基于仿真的 DPA 攻击平台，对设计进行攻击实验，实验结果表明，在无防护的处理器仅需 80 条功耗曲线即可完成攻击的情况下，开启功耗隐藏模式后，采集 2000 条功耗曲线仍无法完成攻击，

证明了本文提出的处理器抗功耗分析攻击方案的有效性。

（4） 分析了在 32 位总线位宽系统中，不同情况下存储器单行数据发生单比特与多比特错误的情况，分析结果表明自然环境下，数据发生单比特错误的概率远远大于多比特错误概率；而人为恶意攻击情况下，出现多比特错误的概率则非常高，据此确定了纠一检多的容错方案。此外，通过分析常用检错码的漏检率，确定使用 CRC 作为多比特错误检测码。在 3.3 节中详细介绍了检纠错模块的实现，同时，进行了仿真验证和综合。综合的结果表明，相比一般的多比特纠错码，该检纠错模块在贴合实际需求的情况下，未引入过多的面积及性能上的开销，适合工业微控制器的存储容错。

5.2 论文的局限与展望

本文提出的微控制器安全容错技术在工程领域中具有较强的实用价值，虽然已取得一定的成果，但限于实验条件和个人精力，仍存在一些有待完善的方面：

（1） 本研究利用 Lockstep 进行处理器容错的同时，还实现了缓存的容错，但是目前对于缓存的容错只支持写通模式。由于写回模式中涉及到缓存与存储一致性问题，仅通过对错误缓存行的无效，可能导致最新数据缺失。因此，基于 Lockstep 对缓存进行全面容错，还有待进一步的研究。

（2） 本研究对于存储器的容错选择了检错码与纠错码结合进行容错，但对算法优化上没有进行过多研究。因此，下一步可以对该算法进行进一步优化，使得检纠错模块在延时或面积上具有更好的指标。

（3） 本研究对于微控制器的容错主要针对处理器容错和存储器容错，对于其他模块出现的错误无法进行检测和恢复。因此，对于微控制器在低成本，高实时下的全面容错还有待进一步的研究。

6 参考文献

[1] Karlsson J , Liden P , Dahlgren P , et al. Using heavy-ion radiation to validate fault-handling mechanisms[J]. IEEE Micro, 1994, 14(1):0-23.

[2] Kanekawa N, Ibe E H, Suga T, et al. Dependability in electronic systems: mitigation of hardware failures, soft errors, and electro-magnetic disturbances[M]. Springer Science & Business Media, 2010.

[3] Patel J H, Fung L Y. Concurrent error detection in ALU's by recomputing with shifted operands[J]. IEEE Trans. Computers, 1982, 31(7): 589-595.

[4] Asghari S A, Marvasti M B, Rahmani A M. Enhancing transient fault tolerance in embedded systems through an OS task level redundancy approach[J]. Future Generation Computer Systems, 2018, 87: 58-65.

[5] 刘光辉. 使用时间冗余保证处理器的可靠性[J]. 计算机工程与应用, 2011, 47(21): 17-22.

[6] Sundaramoorthy K, Purser Z, Rotenberg E. Slipstream processors: Improving both performance and fault tolerance[J]. ACM SIGPLAN Notices, 2000, 35(11): 257-268.

[7] Von Neumann J. Probabilistic logics and the synthesis of reliable organisms from unreliable components[J]. Automata studies, 1956, 34: 43-98.

[8] Abdulhay E, Elamaran V, Arunkumar N, et al. Fault-tolerant medical imaging system with quintuple modular redundancy (QMR) configurations[J]. Journal of Ambient Intelligence and Humanized Computing, 2018: 1-13.

[9] Ernst D, Kim N S, Das S, et al. Razor: A low-power pipeline based on circuit-level timing speculation[C]//Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2003: 7.

[10] Ernst D, Das S, Lee S, et al. Razor: circuit-level correction of timing errors for low-power operation[J]. IEEE Micro, 2004, 24(6): 10-20.

[11] Austin T, Blaauw D, Mudge T, et al. Making typical silicon matter with razor[J]. Computer, 2004, 37(3): 57-65.

[12] Sheikh A T, El-Maleh A H. Double Modular Redundancy (DMR) Based Fault

Tolerance Technique for Combinational Circuits[J]. Journal of Circuits, Systems and Computers, 2018, 27(06): 1850097.

[13] Gyorok G, Beszédes B. Duplicated control unit based embedded fault-masking systems[C]//2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY). IEEE, 2017: 000283-000288.

[14] Abate F, Sterpone L, Lisboa C A, et al. New techniques for improving the performance of the lockstep architecture for SEEs mitigation in FPGA embedded processors[J]. IEEE Transactions on Nuclear Science, 2009, 56(4): 1992-2000.

[15] de Oliveira Á B, Tambara L A, Kastensmidt F L. Exploring performance overhead versus soft error detection in lockstep dual-core ARM cortex-A9 processor embedded into Xilinx Zynq APSoC[C]//International Symposium on Applied Reconfigurable Computing. Springer, Cham, 2017: 189-201.

[16] Reorda M S, Violante M, Meinhardt C, et al. A low-cost SEE mitigation solution for soft-processors embedded in systems on programmable chips[C]//Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, 2009: 352-357.

[17] Violante M, Meinhardt C, Reis R, et al. A low-cost solution for deploying processor cores in harsh environments[J]. IEEE Transactions on Industrial Electronics, 2011, 58(7): 2617-2626.

[18] Azambuja J R, Altieri M, Becker J, et al. HETA: Hybrid error-detection technique using assertions[J]. IEEE Transactions on Nuclear Science, 2013, 60(4): 2805-2812.

[19] Aponte-Moreno A, Pedraza C, Restrepo-Calle F. Reducing Overheads in Software-based Fault Tolerant Systems using Approximate Computing[C]//2019 IEEE Latin American Test Symposium (LATS). IEEE, 2019: 1-6.

[20] Li D, Hu X. Redundant and fault-tolerant algorithms for real-time measurement and control systems for weapon equipment[J]. ISA transactions, 2017, 67: 398-406.

[21] Chardonnereau D, Keulen R, Nicolaidis M, et al. Fault tolerant 32-bit RISC processor: Implementation and radiation test results[C]. 2002.

[22]Valadimas S, Tsiatouhas Y, Arapoyanni A, et al. Single event upset tolerance in flip-flop based microprocessor cores[C]//2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2012: 79-84.

[23]Infineon. 32-bit AURIX™ Microcontroller based on TriCore™ [EB/OL]. https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/?redirId=41544.

[24]Iturbe X, Venu B, Ozer E, et al. A triple core lock-step (TCLS) ARM® Cortex®-R5 processor for safety-critical and ultra-reliable applications[C]//2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W). IEEE, 2016: 246-249.

[25]俞庆华. 恩智浦推出全新 i. MX 8X 处理器, 为工业应用带来更高的安全性, 可靠性和可扩展性[J]. 汽车零部件, 2017 (3): 34.

[26]丛秋波. 新型安全微控制器平台打造更高安全特性[J]. EDN CHINA 电子设计技术, 2011, 18(11): 18-18.

[27]Freescale. MPC567xK: Ultra-Reliable MPC567xK MCU for Automotive & Industrial Radar Applications[EB/OL]. https://www.nxp.com/design/development-boards/automotive-development-platforms/mpc56xx-mcu-platforms/mpc567xk-family-evaluation-board:MPC567xKEVB.

[28]包斌. 航空高可靠处理器容错设计与实现[D]. 国防科学技术大学, 2007.

[29]段凌霄, 孟建熠, 严晓浪. 基于随机延时的嵌入式 CPU 抗 DPA 硬件架构[J]. 计算机应用与软件, 2015, 32(10): 220-222.

[30]Yang S, Wolf W, Vijaykrishnan N, et al. Power attack resistant cryptosystem design: A dynamic voltage and frequency switching approach[C]//Design, Automation and Test in Europe. IEEE, 2005: 64-69.

[31]杭州晟元芯片技术有限公司.一种抗 DPA/SPA 攻击的系统和方法[P].中国:CN201210124489.4, 2012.10.03.

[32]Ambrose J A, Ragel R G, Parameswaran S, et al. Multiprocessor information

concealment architecture to prevent power analysis-based side channel attacks[J]. IET computers & digital techniques, 2011, 5(1): 1-15.

[33] Arora A, Ambrose J A, Peddersen J, et al. A double-width algorithmic balancing to prevent power analysis side channel attacks in aes[C]//2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2013: 76-83.

[34] Tiri K, Akmal M, Verbauwhede I. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards[C]//Proceedings of the 28th European solid-state circuits conference. IEEE, 2002: 403-406.

[35] Coron J S, Kocher P, Naccache D. Statistics and secret leakage[C]//International Conference on Financial Cryptography. Springer, Berlin, Heidelberg, 2000: 157-173.

[36] Messerges T S. Securing the AES finalists against power analysis attacks[C]//International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, 2000: 150-164.

[37] Akkar M L, Goubin L. A generic protection against high-order differential power analysis[C]//International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, 2003: 192-205.

[38] 徐佩, 傅鹂. 防止差分功耗分析攻击的软件掩码方案[J]. 计算机应用研究, 2016: 245-248.

[39] De Mulder E, Gummalla S, Hutter M. Protecting RISC-V against Side-Channel Attacks[C]//Proceedings of the 56th Annual Design Automation Conference 2019. ACM, 2019: 45.

[40] Werner M. Concealing Secrets in Embedded Processors Designs[C]//Smart Card Research and Advanced Applications: 15th International Conference, CARDIS 2016, Cannes, France, November 7–9, 2016, Revised Selected Papers. Springer, 2017, 10146: 89.

[41] Infineon.Integrity Guard – the revolutionary digital security technology from Infineon inspired by nature[EB/OL]. https://www.infineon.com/cms/en/product/

promopages/integrity-guard/?redirId=64328.

[42]ARM.SC300: Designed for High Performance, Embedded Security Applications [EB/OL].https://www.arm.com/products/silicon-ip-cpu/securcore/sc300.

[43]ST.Secure MCU with 32-bit ARM® SecurCore® SC300™ CPU, SWP, ISO, SPI and GPIO interfaces and high-density Flash memory[EB/OL].https://www.st.com/zh/secure-mcus/st33g768.html.

[44]平头哥. S802：具备抗物理攻击能力的超低功耗 32 位处理器[EB/OL]. https://www.t-head.cn/product/s802?spm=a2ouz.12987052.0.0.736148abKJ45Vx.

[45]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]//Proceedings of the conference on Design, Automation & Test in Europe. European Design and Automation Association, 2014: 30.

[46]张振力, 蔡明辉, 韩建伟, 等. 临近空间大气中子诱发电子器件单粒子翻转数值仿真研究[J]. 航天器环境工程, 2010 (4).

[47]Alameldeen A R, Wagner I, Chishti Z, et al. Energy-efficient cache design using variable-strength error-correcting codes[J]. ACM SIGARCH Computer Architecture News, 2011, 39(3): 461-472.

[48]Govindavajhala S, Appel A W. Using memory errors to attack a virtual machine[C]//IEEE Symposium on Security and Privacy. 2003, 5.

[49]Mangard S, Oswald E, Popp T. Power analysis attacks: Revealing the secrets of smart cards[M]. Springer Science & Business Media, 2008.

[50]李明军. CRC 的漏检率分析和铁路信号产品的安全性改进[J]. 2012 列车运行控制系统技术交流会, 2012: 91-94.

[51]尹亚兰, 刘晓然, 施未来. 对 JTIDS 中的 CRC 码的仿真及分析[J]. 通信技术, 2010 (1): 61-62.

[52]高玉红. 32 位 EMC 中 ECC 的设计研究[D]. 哈尔滨工业大学, 2010.