硕士学位论文

基于 ECC 电路的 SRAM 自检测修复设计与验证

DESIGN AND VERIFICATION OF SRAM SELF-DETECTION REPAIR BASED ON ECC CIRCUIT

朴英珲

哈尔滨工业大学

2019 年 6 月

国内图书分类号：TN492　　学校代码：10213
国际图书分类号：621.3　　密级：公开

工程硕士学位论文

基于 ECC 电路的 SRAM 自检测修复设计与验证

硕士研究生： 朴英珲
导　　师： 霍明学 教授
申请学位： 工程硕士
学　　科： 集成电路工程
所在单位： 航天学院
答辩日期： 2019 年 6 月
授予学位单位： 哈尔滨工业大学

Classified Index: TN492

U.D.C: 621.3

Dissertation for the Master Degree in Engineering

DESIGN AND VERIFICATION OF SRAM SELF-DETECTION REPAIR BASED ON ECC CIRCUIT

Candidate: Piao Yinghun

Supervisor: Prof. Huo Mingxue

Academic Degree Applied for: Master of Engineering

Speciality: Integrated Circuit Engineering

Affiliation: School of Astronautics

Date of Defence: June, 2019

Degree-Conferring-Institution: Harbin Institute of Technology

摘　要

在空间环境当中，大量的辐射粒子照射到存储器芯片，辐射粒子导致存储单元发生多位翻转（Multiple Bit Upset，MBU）的软错误，以及单粒子硬错误（Single Hard Error，SHE）。其中，硬错误不会被覆盖消除，这种硬错误的积累，对于纠错能力有限的存储器，势必会影响其数据的准确性。因此针对两种错误，提高存储器的自检测修复能力具有重要的现实意义。

本文深入研究了单粒子多位翻转和硬错误对存储器的影响，从系统级加固设计出发，基于低冗余矩阵码编码译码原理，实现了可纠正连续 4 位错误，加固 32 位数据的低冗余矩阵码电路，并对该编译码器进行了功能仿真验证和纠错模式分析。

研究了自检测区分存储单元内软错误和硬错误的方法。基于低冗余矩阵码电路，对存储器进行加固设计；通过二次检测的方法，采用有限状态机实现了状态控制器设计；从而实现了修复软错误的同时判断错误类型的功能。构建故障注入平台，验证了修复软错误探测硬错误功能的正确性。

研究了修复存储单元硬错误的方法。通过地址映射隔离硬错误的方法，设计实现了故障地址分析器，改进了状态控制器，并协调各个电路功能，实现了自检测修复软硬错误的 SRAM 整体结构。在 SMIC65nm 工艺条件下，对各个电路进行综合，并对整体 SRAM 结构进行了门级仿真验证。

本文最后基于 AHB 总线协议，利用总线功能模型对本文设计进行了系统级验证。为自检测修复软硬错误的 SRAM 结构设计了 AHB 总线接口，经验证结果表明，本文设计的自检测修复软硬错误的 SRAM 结构能够集成到系统中去应用。

关键词：低冗余矩阵码；SRAM 存储器；软错误；硬错误；AHB

Abstract

In the space environment, a large number of radiation particles irradiate the memory chip, which results in Multiple Bit Upset (MBU) soft errors and Single Hard Error（SHE）. Among them, hard errors will not be covered and eliminated. The accumulation of hard errors will inevitably affect the accuracy of data for memory with limited error correction ability. Therefore, in view of the two kinds of errors, it is of great practical significance to improve the self-detection and repair ability of memory.

In this paper, the influence of single event multi-bit flip and hard error on memory is studied in depth. Based on the coding and decoding principle of low redundancy matrix-based codes, the circuit of low redundancy matrix-based codes which can harden 32-bit data and correct 4 adjacent errors is realized. The function simulation of the circuit is carried out and the error correction mode is analyzed.

The method of judging type of error in memory cell by self-detection is studied. Based on the low redundancy matrix code circuit, the memory is hardened. The design of state controller is realized by finite state machine with the method of secondary detection. Thus, the function of repairing soft errors and judging the type of errors is realized. A fault injection platform is constructed to verify the correctness of the function of repairing soft errors and detecting hard errors.

The method of repairing memory cell hard error is studied. By means of address mapping to isolate hard errors, a fault address analysis is designed and implemented, the state controller is improved, the functions of each circuit are coordinated, and the SRAM architecture for self-detection and repair of soft and hard errors is realized. Under the condition of SMIC 65nm process, all circuits are synthesized, and the whole SRAM structure is verified by gate-level simulation.

Finally, based on the AHB bus protocol, the system-level verification of design is carried out by using the bus function model. The AHB bus interface is designed for the SRAM structure of self-detecting and repairing soft and hard errors. The results show that the SRAM structure designed in this paper can be integrated into the system for application.

Keywords: low redundancy matrix-based code, SRAM, soft error, hard error, AHB

目 录

摘 要 I
ABSTRACT II
第 1 章 绪 论 1
1.1 研究目的和意义 1
1.2 国内外研究现状 3
1.2.1 存储器加固 3
1.2.2 硬错误产生 6
1.3 论文研究内容及论文结构 7
第 2 章 低冗余矩阵码理论及实现 9
2.1 线性分组码基本理论 9
2.1.1 线性分组码原理 9
2.1.2 汉明码编码译码实现 11
2.2 低冗余矩阵码原理 13
2.2.1 低冗余矩阵码编码原理 14
2.2.2 低冗余矩阵码译码原理 16
2.3 低冗余矩阵码设计 18
2.3.1 编码器设计 18
2.3.2 译码器设计 20
2.4 功能仿真验证 21
2.5 本章小结 23
第 3 章 软错误修复硬错误探测的 SRAM 设计与验证 24
3.1 基于低冗余矩阵码的存储器加固设计 24
3.1.1 存储器模型 24
3.1.2 存储器加固 25
3.2 软错误修复与检测硬错误设计 27
3.2.1 软错误修复与硬错误检测 27
3.2.2 状态控制器设计 28
3.2.3 整体结构设计 31
3.3 故障注入平台及验证 33
3.3.1 软错误修复 33

3.3.2 硬错误探测 36
3.4 本章小结 37
第 4 章 软硬错误自修复的 SRAM 设计与系统级验证 38
4.1 软硬错误自修复设计 38
4.1.1 故障地址分析器 38
4.1.2 改进的状态控制器 42
4.1.3 软硬错误自修复的 SRAM 整体结构 44
4.1.4 综合与门级仿真 47
4.2 AHB 总线接口设计与系统级验证 51
4.2.1 AHB 总线接口设计 51
4.2.2 系统级验证 53
4.3 本章小结 56
结论 57
参考文献 58
攻读硕士学位期间发表的论文及其它成果 62
哈尔滨工业大学学位论文原创性声明和使用权限 63
致谢 64

第1章 绪　论

1.1 研究目的和意义

航天科技的水平通常代表着一个国家的综合实力。随着对高性能航天电子设备的需求日益迫切，越来越多的电子信息集成系统被应用在各类航天器上。不同于地面环境，航天器在轨运行时，会受到多种极端环境的影响。对于航天器上的电子设备而言，最主要的影响来自粒子辐照。空间中的辐射环境主要有地球辐射带、太阳和银河的宇宙线等，也与太阳活动及地磁活动有关，在不同轨道的辐射环境差别很大[1]。其中，对电子信息系统中集成电路芯片可靠性影响最大的是高能质子、重离子和电子。

集成电路芯片在空间辐射环境中受到粒子辐射时，入射的空间粒子在器件或集成电路中产生缺陷和电离损伤，会影响器件的性能，严重的可能会导致器件的功能失效或者毁损，这种现象称为单粒子效应（Single Event Effect，SEE）[2]。粒子撞击器件时积攒的能量，会使器件内产生大量的电子空穴对，这就会引起两种SEE 错误，软错误[3]和硬错误。软错误有单粒子翻转（Single Event Upset，SEU）和单粒子瞬态效应（Single Event Transient，SET）等[4-5]，像 SEU 通常是一个尖脉冲或者位翻转，在现代存储器当中，SEU 会使电路发生一个或多位的翻转现象，但这种错误是可以纠正或重写来恢复原有功能，使电路恢复正常。硬错误有单粒子烧毁（Single Event Burnout，SEB），单粒子栅击穿（Single Event Gate rupture，SEGR）和单粒子硬错误（Single Hard Error，SHE），这些硬错误是器件的永久性物理损伤，导致器件功能失效[5-7]，并且无法通过重写对其原有功能进行修复。

随着微电子制造工艺的快速发展，单片集成电路系统级芯片中所需的存储器容量变得越来越大。例如，在片上系统（System on Chip，SoC）中，存储器的面积占比一般为整个芯片面积的 60%以上；在通用微处理器中，高速缓存 Cache 所占的面积也达到了 30%以上[8]。这也加重了存储器在空间环境当中被粒子辐照的影响，逐渐升高的集成度会对辐射更加敏感。

因此一直以来，单粒子效应所引起的错误始终是存储器故障的主要因素之一。1994 年发射的“实践四号”卫星上安装了“静态单粒子翻转事件探测仪”，这是我国首次开展 SRAM 芯片空间单粒子效应研究，实验结果显示，1Mbit SRAM 在轨道上每天平均发生 3.5 个单粒子翻转[9]。在 2002 年到 2009 年之间，Alsat-1 卫星的辐射故障统计显示，7 年间单粒子翻转发生率为 3.67×10^{-7}SEU/位/天，其中 98.6%

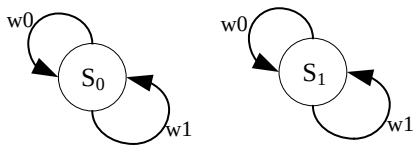
为一位错误（Single-bit Errors），1.21%为双位错误(Double-byte Errors)，总共发生了 247595 次单粒子翻转错误[10]。2011 年我国成功发射了第一个空间实验室“天宫一号”，然而在 2013 年与“神舟十号”飞船进行对接的前两个月，由于空间粒子辐射的影响，电子信息系统发生了多次单粒子翻转，导致舱内的泄压指令出现错误，泄压系统无法及时启动，严重威胁了飞船内宇航员的生命安全[11]。

随着单位面积上存储单元数量的增加，存储单元的临界电荷不断降低[12-13]。空间粒子辐射会造成多个存储器单元发生翻转，即发生存储器的多位翻转（Multiple Bit Upset，MBU）[3,14]。Eishi Ibe 等人研究了 180nm，130nm，90nm，65nm，45nm，32nm 和 22nm 等在不同工艺条件下存储器 MBU 特性，辐射实验与仿真研究显示，从 180nm 到 22nm，最大位数从 2 位增加到了 18 位，且 22nm 多位翻转比例为 180nm 的 8 倍左右，MBU 的发生概率明显增大；典型的为 65nm 时，MBU 的比例为 2.4%，具体的多位翻转如表 1-1 所示。

表 1-1 MBU 占 SEU 的百分比[3,13,15]

工艺尺寸（nm）	总数	多位翻转位数				
		2	3	4-8	>8	最大位数
180	0.5%	0.5%	0.0%	0.0%	<0.1%	2
130	1.2%	0.8%	0.2%	0.2%	<0.1%	7
90	1.9%	1.5%	0.2%	0.2%	<0.1%	7
65	2.4%	2.1%	0.1%	0.0%	<0.1%	10
45	2.3%	1.9%	0.2%	0.1%	<0.1%	16
32	3.1%	2.6%	0.2%	0.3%	<0.1%	16
22	3.9%	3.0%	0.3%	0.1%	<0.1%	18

对于空间辐射引起的 SRAM 存储器硬错误，在 20 世纪 90 年代时，由 Koga R 等人进行了 SRAM 存储器的重离子注入实验，首次发现并提出了“固定位”（stuck bit）错误[16-18]，固定位错误如图 1-1 所示。



存储单元固定为0　存储单元固定为1

图 1-1 固定位错误

从图 1-1 中可见，当存储单元固定为高（低）电平时，对存储器写入任何数值，都不会改变存储单元的数据。这种硬错误是由于单个高 LET 值的重离子在其轨迹附近沉淀的微计量失效而导致器件的永久性退化，这种现象与单粒子翻转所引起

的软错误不同，SRAM 存储器的存储状态会发生不可更改的现象[18]。一旦发生这类“固定位”的硬错误，即使进行刷新也无法修复存储单元的错误，若再次发生单粒子事件，就会发生错误积累的情况，对于纠错能力不足的存储器，纠错码将会失效。

为了保证航天器内集成电路芯片在空间辐射环境中正常工作，鉴于存储器在集成电路系统级芯片中占据了相当大的比重，提高芯片内存储器的可靠性，具有重要的意义。因此针对空间的恶劣环境导致的软错误与硬错误，根据两种错误的特点，设计出各自对应两种错误的自诊断并自修复的存储电路是航天器在空间辐射环境当中能够长期稳定工作的根本。

1.2 国内外研究现状

1.2.1 存储器加固

针对 SRAM 存储器软错误修复的研究有很多，对其单元进行加固设计的研究也很多，可根据加固的类型有版图级、电路级和系统级加固等等。

电路级加固是通过改变 SRAM 存储单元的电路结构来实现加固的。其中最典型的存储单元结构是双互锁存储单元（Dual Interlocked Storage Cell，DICE），其特点是添加了存储单元中相同逻辑状态的节点来进行内部冗余的加固[19]。其电路结构如图 1-2 所示：

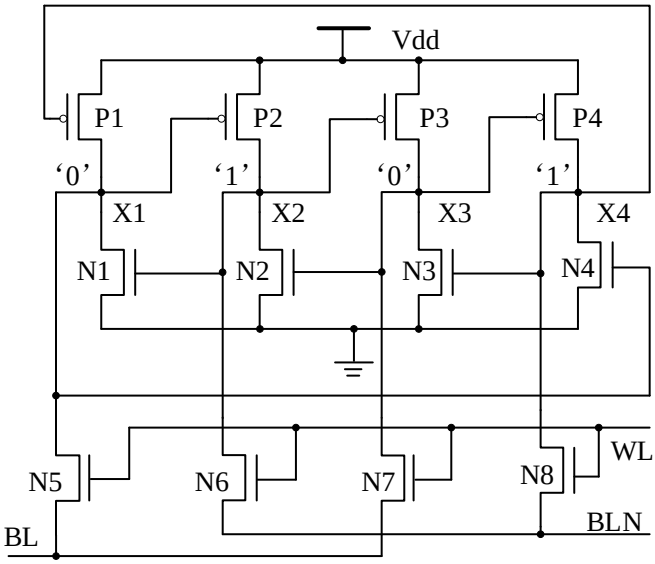


图 1-2 DICE 单元结构

从图 1-2 中可以看出，DICE 单元中，P 表示 PMOS 管，N 表示 NMOS 管，X 表示存储节点，X1 和 X3 为相同逻辑，X2 和 X4 为相同逻辑。N5 到 N8 为传输门，在存储阶段处于关闭状态。如果 DICE 单元存储的值为“1”时，从图中所示，X2 和 X4 节点为“1”，X1 和 X3 节点为“0”。此时，P2、P4、N1 和 N3 导通；P1、P3、N2 和 N4 截止，这 8 个 MOS 管形成 2 个反馈环，假如单粒子翻转使 DICE 的 X2 存储节点发生故障时，即从“1”变成了“0”，这时 P3 会导通，X3 节点可能会变成“1”，但因为存储单元的互锁结构，X1 和 X4 仍然是原来的状态，从而利用反馈使 X2 和 X3 进行修正[20-21]。

但是 DICE 单元并不能有效地抵抗单粒子多节点的翻转，因此 D´Alessio M，Ottavi M 等人以 DICE 存储单元为基础，提出了一种可以抵抗单粒子多节点翻转的 TDICE 存储单元结构，其详细电路如图 1-3 所示[22]。

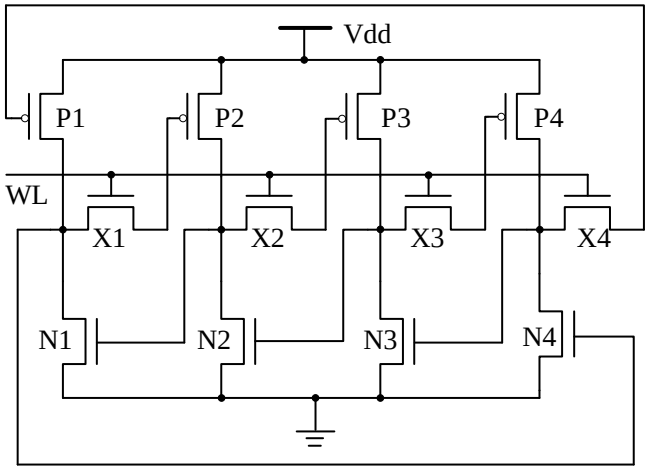


图 1-3 TDICE 单元结构[22]

从图 1-3 中可以看出，与 DICE 存储单元的不同之处在于把 NMOS 传输管添加到了存储器单元的反馈回路上。因此可通过传输管的开关特性控制 DICE 单元中的节点与节点间的反馈回路，从而使该存储单元在保持存储数据时，实现了每个节点相互独立的存储状态，这一设计使得该结构拥有了抵抗多节点翻转的能力[21-22]。

在系统级加固方面，针对 SRAM 存储器大多数应用的加固方法是错误修正码加固技术（Error Correction Codes，ECC）[23-25]。该加固方式不需要考虑版图与电路层面的问题，主要以存储的数据为主要对象，通过存储器添加一定的冗余单元，及在存储器的输入端与输出端，进行 ECCs 编码译码设计，从而实现对数据的加固。当存储器输入 k 位数据时，数据首先通过编码器编码得到 n 位码字之后，再存储到带冗余的存储器内。如果存储器内的数据发生错误时，则在读取数据时，可通

过译码器进行纠正，从而在译码器解码后恢复正确的 k 位数据，相关的存储器加固结构如图 1-4 所示。

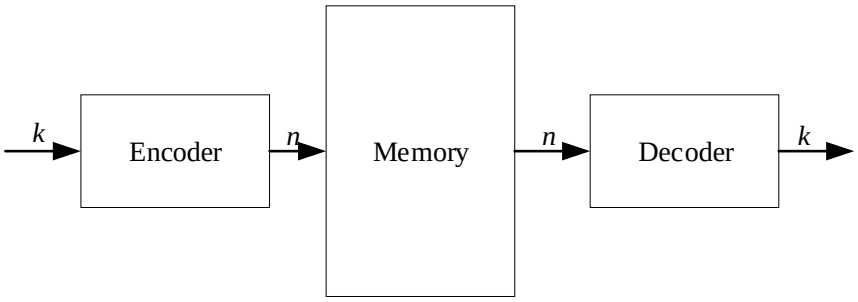


图 1-4 存储器加固

ECCs 加固设计中最常用的编码就是汉明码，它可以纠正一位错误[27]，汉明码是在原信息位的基础上，添加一定的监督位，得到码字之后，通过数据之间的相关算法计算数据中的错误位置，从而纠正一位错误。20 世纪 70 年代，Hsiao 在研究汉明码的基础上，添加一位校验码，提出了双错误探测码（Single Error Correction and Double Error Detection，SEC-DED），该纠错码能够在修正 1 位错误的同时，也可以检测到任意 2 位的错误[28]。在此基础上为了进一步提高检错能力，文献[29]提出既可纠正 1 位错误，检测随机 2 位错误，且还能检测到连续 3 位错误的 SEC-DED-TAED（Single Error Correction-Double Error Detection-Triple Adjacent Error Dectection）码，其特点在与构建校验矩阵时，使其能够在发生连续 3 位错误情形与其他两种情形产生不同的校正子，因此可以通过检测校正子的汉明重量来区分上述三种情形。

然而上述的纠错码都是在汉明码的基础上，增添了可以检测 2 位错误或连续 3 位错误的能力，并不能纠正多位，随着特征尺寸的减小，在空间强辐射环境当中，存储器多位翻转（Multiple Bit Upset，MBU）已经被检测出，因此需要纠错能力更加强大的 ECC 码来面对这种强辐射环境。在通信领域中，如 Reed-Somolon（RS）码[30-31]、Bose-Chaudhuri-Hocquenghem（BCH）[32]等都是纠错能力比较强的循环纠错码，但是这些码需要大量的冗余单元，且译码过程比较繁琐，可能需要多个时钟周期来运算，增加了相应的延迟，因此在高性能要求的领域中，不适合对存储器进行加固。

为了进一步提高纠错码的纠正能力及减少相应的冗余，研究者们研究出了矩阵码加固技术。Argyrides 等人研究出了把一个字排列成矩阵形式，并在相应的行添加 SEC-DED 码和相应的列分别计算奇偶校正码，该矩阵码可以纠正 2 位错误，具体的矩阵码结构如图 1-5 所示[33]。

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	C_1	C_2	C_3	C_4	C_5
X9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}	X_{16}	C_6	C_7	C_8	C_9	C_{10}
X_{17}	X_{18}	X_{19}	X_{20}	X_{21}	X_{22}	X_{23}	X_{24}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}
X_{25}	X_{26}	X_{27}	X_{28}	X_{29}	X_{30}	X_{31}	X_{32}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}
D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8					

图 1-5 文献[33]矩阵码形式

图 1-5 中，C_1 到 C_5 为前面 8 位数据的 SEC-DED 码校验码，D_1 到 D_8 为每列的奇偶校验码，冗余位为 28 位。利用故障注入实验对该方法进行了评价，与 Reed-Muller 码和汉明码进行了比较。结果表明该矩阵码技术性能优于汉明码，与 Reed-Muller 码的性能相当，且面积和功耗降低了 25%。在此基础上，文献[34]进一步增加了纠错能力，其将 32 位数据分成了 4 行 8 列的形式，对每行每列计算奇偶校验码，因此增加了 32 位冗余位，纠错能力为连续 4 位错误。由于矩阵码冗余位数比较多，存储器的面积也会相应的增加。为了在不降低纠正能力的基础上，降低冗余位数，文献[35][36]提出了低冗余矩阵码加固方案，该方案是把数据变成二维矩阵之后，用到了校验位的逻辑共享方法，隐藏掉了相关的校验位，从而降低了冗余位数，例如 32 位的数据，只需要添加 20 位的冗余位，就可纠正连续 4 位的错误，由于其加固方法的灵活性，可以用于任何宽度的数据加固。

1.2.2 硬错误产生

自 1991 年，研究人员首次在 SRAM 存储器单元中发现“固定位”错误之后，Dufour C 在对商用 SRAM 存储器做重离子注入试验也发现相应的固定为“0”或者“1”的单元错误[37]。

不同于单粒子翻转引起的软错误，这种错误是永久性损伤，存储单元受到辐照后，会一直保持“0”或“1”的值，这种错误一旦发生，即使存储器进行刷新操作，也无法把存储器内的错误消除。因此当包含硬错误的字因辐射粒子发生软错误时，就会导致一个字内同时存在软错误与硬错误，对于纠错能力有限的存储器，其纠错功能会受到影响，从而发生误纠的情形。C.Poivey，T.Carriere 等人对两种类型的 1Mbit 大小 SRAM 存储器进行粒子注入实验发现，单个粒子足以引起 SRAM 存储器单元发生“固定位”错误[38]。

2001 年，Koga 等人发现高密度的存储设备对重离子很敏感，高能重离子在其径迹附近所沉积的剂量达到一定水平时，会使硬错误的敏感度上升，继而发生存储器的“固定位”的硬错误[39-40]。Pellati P 等人对浮栅型存储器进行辐照时发现，辐照中的重离子会使浮栅型存储单元的阈值电压发生漂移，破坏读写功能或者使存储器的工作电流变异常[18,41]。文献[18]综述了高能粒子导致器件失效的机制：

LET 值足够高的重离子在入射到绝缘体的材料后，在其周围的原子发生电离，由于电离原子之间的库伦排斥力很大，从而导致化学键断裂，由于斥力的作用，电离的原子会分离运动，从而影响该材料的特性，严重的可能会使器件发生失效，其失效机制如图 1-6 所示。

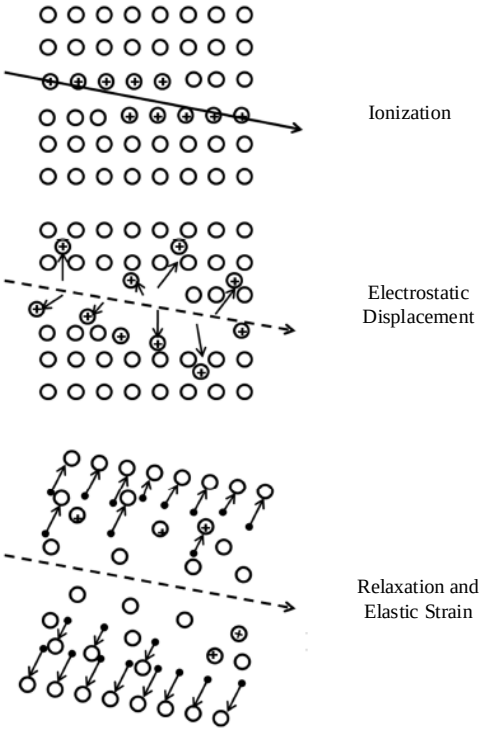


图 1-6 失效机制[18]

2014 年，Avner Haran，Joseph Barak 等人在 SRAM 中检测到重离子辐照引起的单粒子硬错误，并测量了其截面，同时讨论了 SHE 的产生机制，研究表明 SHE 可能与微剂量效应和通过栅极氧化物形成的导电路径有关，明确表明电离是导致 SRAM 发生“固定位”错误的机制[42]。2017 年，Anna B 等人基于实验测得 SHE 的截面，LET 阈值小于 7MeV·cm^2/mg[43]。

1.3 论文研究内容及论文结构

本课题的研究内容分为四个方面：（1）基于低冗余矩阵码电路，对 SRAM 存储器进行加固设计。重点研究线性分组码相关原理、汉明码编码译码原理、多位探测方法以及逻辑共享原则，研究低冗余矩阵码的构造方法，进行编码译码硬件电路设计，并对存储器进行加固。（2）研究自检测区分存储单元内软错误和硬错误的方法。重点研究二次检测技术，研究存储器回写的方法，设计相关的状态控

制器，实现自检测区分错误类型并修复存储器内软错误的功能；构建故障注入平台，验证该功能的正确性。（3）研究修复存储器内硬错误的方法。重点研究利用地址映射技术隔离硬错误的方法，设计映射硬错误地址到冗余地址的故障地址分析器；协调各个模块，实现可自检测修复存储器内软硬错误的 SRAM 存储器整体结构；基于 SMIC65 工艺条件下，对该 SRAM 存储器结构进行逻辑综合与门级仿真验证。（4）对自检测修复软硬错误的 SRAM 存储器结构电路进行系统级验证。为该 SRAM 存储器结构设计 AHB 接口，并基于 AHB 总线的 BFM 进行仿真验证，证明该 SRAM 结构能够集成到系统中使用。

论文的章节安排如下：

第 1 章绪论是先介绍了课题研究的意义和目的，并讲述了国内外研究现状，简单介绍了各种 ECC 纠错码及其优缺点，并介绍了导致 SRAM 存储器单元发生硬错误的产生原因及相关特征。

第 2 章主要介绍 ECC 基本理论、汉明码和低冗余矩阵码的理论基础，并基于相关理论，实现可纠正连续 4 位的（52，32）低冗余矩阵码编译码电路，并对该设计进行故障注入，验证功能是否正确。

第 3 章构建 SRAM 存储器的软错误自修复机制，实现检测区分软错误和硬错误并修复其存储器内软错误的方法，并对该软错误修复机制进行故障注入，验证加固效果。

第 4 章基于前章设计，添加适当冗余存储器，完成硬错误的自修复机制，并对整体设计添加 AHB 总线结构，构建系统级验证平台，对设计进行故障注入，完成系统级验证。

第2章 低冗余矩阵码理论及实现

本章介绍 ECC 基本理论，相关码字的构成以及校正子的纠错原理，研究矩阵码相关的汉明码，奇偶校验码，并实现（7，4）汉明码编码译码器。重点研究低冗余矩阵码的理论基础，多位错误探测方法及编码流程和纠错译码算法等等[35]，并结合已实现的（7，4）汉明码编码译码器，利用硬件语言设计（52，32）低冗余矩阵码的编译码电路，通过对电路进行故障注入，验证编码译码器的功能。

2.1 线性分组码基本理论

2.1.1 线性分组码原理

当分组码的信息位和监督位可通过线性关系进行联系时，可称为线性分组码，包括汉明码和循环码等。对于（n，k）线性分组码，码字为 n 位，原信息位为 k 位，n-k 为监督位宽度，可通过与码对应的生成矩阵，将 k 位信息位与 n-k 位监督位联系起来生成码字，该矩阵如下公式所示。

$$G=[I_k P]=\begin{bmatrix} 1 & 0 & \cdots & 0 & p_{1,1} & p_{1,2} & \cdots & p_{1,n-k} \\ 0 & 1 & \cdots & 0 & p_{2,1} & p_{2,2} & \cdots & p_{2,n-k} \\ \vdots & & & & \vdots & & & \\ 0 & 0 & \cdots & 1 & p_{k,1} & p_{k,2} & \cdots & p_{k,n-k} \end{bmatrix} \tag{2-1}$$

式中，I_k 为 k 阶单位矩阵；

P 为线性无关矩阵。

线性分组码的码字生成过程是通过对原数据进行 G 矩阵线性变化而得到的，即数据位乘以矩阵 G 而得到码字。码字中对应单位矩阵码列向量的数据位为原信息位，对应 P 矩阵列向量的数据位为监督位。若设信息位为 $a=[a_1,a_2,a_3,a_4,\ldots,a_k]$，则码字 C 的公式如下所示。

$$C=[C_1,C_2,C_3,\cdots,C_n]=[a_1,a_2,a_3,a_4,\cdots,a_k]\bullet G \tag{2-2}$$

通过公式（2-2）得到的码字 C 中可以看出，前 k 位为原信息位，后 n-k 位为通过矩阵生成的监督位，此生成码字的过程为编码流程。

在译码时，需要根据码字的奇偶校验矩阵 H，验证数据的正确性，奇偶校验矩阵 H 的具体算法如下所示。

$$H=[P^T I_{n-k}]=\begin{bmatrix} p_{1,1} & p_{2,1} & \cdots & p_{k,1} & 1 & 0 & \cdots & 0 \\ p_{1,2} & p_{2,2} & \cdots & p_{k,2} & 0 & 1 & \cdots & 0 \\ \vdots & & & & \vdots & & & \\ p_{1,k} & p_{2,k} & \cdots & p_{k,n-k} & 0 & 0 & \cdots & 1 \end{bmatrix} \quad (2\text{-}3)$$

式中，P^T 为矩阵 P 的转置。

结合码字的生成矩阵 G 和奇偶校验矩阵 H 可以得到码字 C 与校验矩阵 H 的关系，通过异或关系，码字乘以矩阵 H 的转置得到 0，具体公式如下所示。

$$C \bullet H^T=[C_1,C_2,C_3,\cdots,C_n] \bullet H^T = a \bullet G \bullet \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n-k} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n-k} \\ \vdots & & & \\ p_{k,1} & p_{k,2} & \cdots & p_{k,n-k} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{bmatrix} = 0 \quad (2\text{-}4)$$

根据上式，通过异或运算得到相应的校正子 S，P 矩阵对应的是 k 位信息位，单位矩阵对应的是监督位，当通过编码得到的码字 C 存储到存储器之后，若发生错误，设错误向量为 e，则从存储器读出的接收字为 $C' = C + e$，通过公式（2-4）计算可以得到校正子 S，具体公式如下所示。

$$S = C' \bullet H^T \quad (2\text{-}5)$$

由公式（2-5）可知，校正子反映了错误模式。因此进行译码操作时，先根据公式（2-5）计算接收字 C'的校正子判断是否发生错误，当校正子 S 为零向量时，该字是正确的。当校正子 S 为非零向量时，将其对应的错误向量 e 与接收字 C'进行异或操作即可纠正错误，继而得到正确的数据。

汉明码是线性分组码中最典型的纠错码，可以纠正一位错误，设汉明码的码字宽度 n，监督位宽度为 i（即 $i = n - k$），则需要满足以下关系式：

$$2^i - 1 \geq n \quad (2\text{-}6)$$

通过公式（2-6）可以得到汉明码中信息位宽度与监督位宽度的关系，如表 2-1 所示。

表 2-1 信息位宽度与监督位宽度的关系

信息位宽度 k	监督位宽度 i
1	2
2~4	3
5~11	4
12~26	5
27~57	6
58~120	7

根据表 2-1 选定合适的信息位与监督位数之后，可添加奇偶校验位来探测数据是否发生奇数个随机错误。对于 k 位数据 $a=[a_1,a_2,a_3,a_4,\ldots,a_k]$，奇偶校验位 h 可通过对数据内所有位进行异或而得到，具体公式如下：

$$h=a_1\oplus a_2\oplus a_3\bullet\bullet\bullet\oplus a_k \tag{2-7}$$

通过公式（2-7）可得到 h，当数据中发生奇数个错误时，对错误数据进行异或生成的 h"会与原数据中的奇偶校验位 h'不同，因此可以通过对原奇偶校验位 h' 和新生成的 h"进行异或得到奇偶校正子 S_h 来判断数据是否发生奇数个错误，具体公式如下：

$$S_h=h'\oplus h'' \tag{2-8}$$

当新生成的 h"与得到数据内的奇偶校验位 h'不同时，S_h 为“1”，即原本的数据发生了奇数个错误。若假定数据至多发生了 2 位错误时，可以根据奇偶校正子 S_h，和公式（2-5）得到校正子 S 来分出三种情况，第一：数据位发生错误，奇偶校验位 h 没有发生错误，这时奇偶校正子 S_h 为 1，数据校正子 S 不为 0；第二：奇偶校验位 h 发生错误，数据位没有发生错误；这时奇偶校正子 S_h 为 1，汉明码校正子 S 为 0；第三：奇偶校验位 h 发生错误和数据位发生 1 位错误或数据位发生 2 位错误；则奇偶校正子 S_h 为 0，汉明码校正子 S 不为 0。因此可以通过判断奇偶校正子 S_h 和汉明码校正子 S 来判断数据的错误情况，从而根据相应情形，采取相应的纠正策略。

2.1.2 汉明码编码译码实现

本小节将对（7，4）汉明码编码器和译码器进行设计。根据上一节汉明码的理论，首先生成校验矩阵，如图 2-1 所示。

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

图 2-1 （7，4）汉明码校验矩阵

根据图 2-1 中的矩阵、公式（2-3）、公式（2-4）和公式（2-7），添加奇偶校验位 h，设 a 为信息位，b 为校验位，监督位与信息位的关系如表 2-2 所示。

表 2-2 监督位与信息位的关系

监督位	信息位关系式
h	$a_1 \wedge a_2 \wedge a_3 \wedge a_4$
b_1	$a_1 \wedge a_3 \wedge a_4$
b_2	$a_1 \wedge a_2 \wedge a_3$
b_3	$a_2 \wedge a_3 \wedge a_4$

根据表 2-2 中监督位与信息位关系，校正子与数据位的关系如表 2-3。

表 2-3 校正子与数据位的关系

校正子	信息位关系式
S_1	$b_1 \wedge a_1 \wedge a_3 \wedge a_4$
S_2	$b_2 \wedge a_1 \wedge a_2 \wedge a_3$
S_3	$b_3 \wedge a_2 \wedge a_3 \wedge a_4$
S_h	$h \wedge a_1 \wedge a_2 \wedge a_3 \wedge a_4$

根据图 2-1，表 2-2 和表 2-3，校正子 S 与错误位置的关系如表 2-4 所示。

表 2-4 校正子与错码位置关系

$S_1\ S_2\ S_3$	错误位置
110	a_1
011	a_2
111	a_3
101	a_4
100	b_1
010	b_2
001	b_3

编码器的设计依据表 2-2 监督位与信息位的关系，利用组合逻辑得出监督位并与信息位一起输出。译码器首先主要负责计算校正子，根据表 2-3 中校正子与数据的关系，若 S 校正子为 0 时，直接输出数据；否则数据需要进行纠正。纠正时主要根据表 2-4 中校正子与错误位置关系，把相应错误位置数据进行翻转，并将修改的数据输出。编码器和译码器模块如图 2-2、2-3 所示。

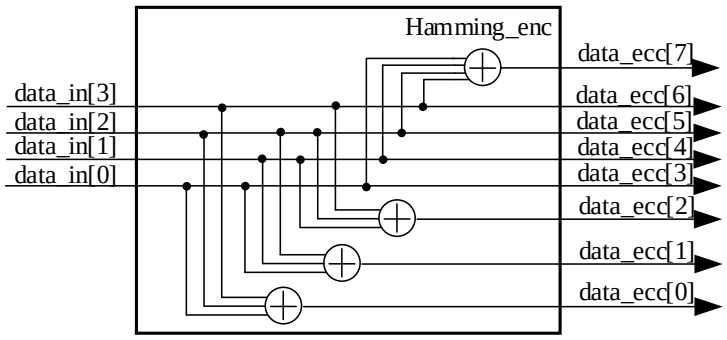


图 2-2 汉明码编码器

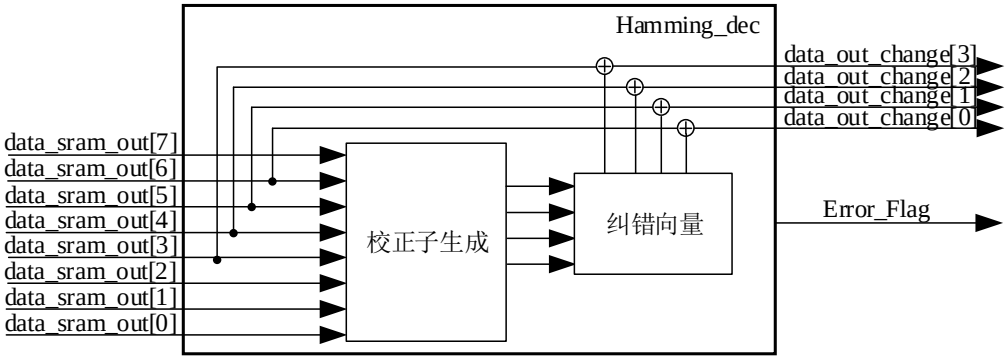


图 2-3 汉明码译码器

根据图 2-2 和图 2-3 中，汉明码编译码器的模块，其详细信号定义如表 2-5、2-6 所示。

表 2-5 汉明码编码器信号定义

信号	定义
data_in	输入数据（4 位）
data_ecc	编码数据（8 位）

表 2-6 汉明码译码器信号定义

信号	定义
data_sram_out	从存储器输出的数据（8 位）
data_out_change	校正数据（4 位）
Error_Flag	错误探测信号

2.2 低冗余矩阵码原理

根据上一节线性分组码的理论基础和汉明码的编码译码原理，以及所实现的汉明码的编码译码器，本节重点介绍可以纠正连续多位的低冗余矩阵码的构造规则以及相关原理。

2.2.1 低冗余矩阵码编码原理

根据汉明码奇偶校验位的探测方法，可以探测到数据位的奇数个错误位，但是不可确定哪些位置发生了错误，对于错误探测方法，还有一种可以探测连续位的方法，即利用 t 个校验位来探测 t 个相邻的错误，若数据为 k 位，该方法是将数据分成 k/t 个组合，把每个组合中对应的位进行异或得到相应的 t 个校验位，设校验位为 r，具体公式如下：

$$r_i = a_i \oplus a_{i+t} \oplus \bullet\bullet\bullet \oplus a_{i+(j-1)t} \tag{2-9}$$

式中，i 表示每组的对应位数；

j 表示对应的组。

t 个校验位就由公式（2-9）异或运算得到，并与原数据进行合并，得到编码数据。计算校正子时，根据接收数据的信息位，利用公式（2-9）重新计算校验位 r_n，并与接收数据内的校验位 r_m 相异或而得到校正子 S_r，从而通过 S_r 的值判断数据内发生的相邻错误，校正子 S_r 为：

$$S_r = r_m \oplus r_n \tag{2-10}$$

假设 S_r 的第 i 位为“1”，则表示上述每组中的第 i 位中，有奇数个错误，若假定只有两组时，就表明数据中 a_i 和 a_{i+t} 两位中有一位包含错误。当连续错误位数小于探测能力 t 时，可以根据 S_r 中 1 的个数来判断数据位中发生的连续错误位数。例如，数据位 16 位，把 16 位数据分为 4 组，通过公式（2-9）得到 4 位校验位；在译码时，计算校正子 S_r，假定数据中 a_1 到 a_4 发生错误，则 S_r 的值为“1111”，a_3 到 a_5 发生错误时，S_r 的值为“1011”，a_1，a_3 和 a_4 发生错误时，S_r 的值为“1011”。当连续错误的位数超过了纠错能力 t 时，有可能发生 S_r 变为全 0 的情况，即当 a_1 到 a_8 发生错误时，由于每组中对应的位数发生了偶数位错误，致使对应校验位 r 没有发生变化，对应 S_r 就变成了 0。因此，只要连续错误位数小于等于纠错能力，且每组相应的位数不同时发生错误时，该方式就可以探测到错误。虽然这种方法不能确定错误的具体位置，但是可以根据 S_r 的值，推测出哪些位置发生了错误，哪些位置没有发生错误。例如 S_r 为“1001”时，每组中第 2 位和第 3 位没有发生错误，发生错误的位置可能是每组中的第 1 位和第 4 位。

低冗余矩阵码的原理将探测数据的连续多位错误方法和原始矩阵码相结合的方式，从而利用较少的冗余达到纠正多位错误的目的。低冗余矩阵码的排列矩阵的方法与原始矩阵码排列方法相同，即第一步是将数据排列成 m 行 n 列的二维矩

阵形式，若设数据位为 k，则 m 乘以 n 等于 k，具体的矩阵形式如图 2-4 所示。

a_1	a_{1+m}	a_{1+2m}		$a_{1+m(n-1)}$
a_2	a_{2+m}	a_{2+2m}		$a_{2+m(n-1)}$
a_3	a_{3+m}	a_{3+2m}		$a_{3+m(n-1)}$
a_4	a_{4+m}	a_{4+2m}		$a_{4+m(n-1)}$
⋮				⋮
a_m	a_{2m}	a_{3m}	⋯	a_{mn}

图 2-4 低冗余矩阵码排列布局

根据图 2-4 中的排列布局，把数据排成相应矩阵之后，根据上一节中，汉明码编码方式，分别计算每一行的奇偶校验位 h 和监督位 b。添加相应的奇偶校验位和监督位之后，因为排列顺序是向下增大而不是向右增大的，若发生连续 m 位的数据发生错误时，可保证每一行只发生一位错误，满足汉明码的纠错能力。因此可以通过奇偶检验位和监督位，判断每一行的错误的位置，从而纠正连续 m 的数据。设监督位位数为 i，i 满足上述 2.1.1 节中公式（2-1）和表 2-1，相应的矩阵形式如图 2-5 所示。

a_1	a_{1+m}	a_{1+2m}	⋯	$a_{1+m(n-1)}$	h_1	b_1	b_2	⋯	b_i
a_2	a_{2+m}	a_{2+2m}		$a_{2+m(n-1)}$	h_2	b_{1+i}	b_{2+i}		b_{2i}
a_3	a_{3+m}	a_{3+2m}		$a_{3+m(n-1)}$	h_3	b_{1+2i}	b_{2+2i}		b_{3i}
a_4	a_{4+m}	a_{4+2m}		$a_{4+m(n-1)}$	h_4	b_{1+3i}	b_{2+3i}		b_{4i}
⋮					⋮				
a_m	a_{2m}	a_{3m}	⋯	a_{mn}	h_m	$b_{1+(m-1)i}$	$b_{2+(m-1)i}$	⋯	b_{mi}

图 2-5 生成校验位矩阵

得到奇偶校验位和监督位之后，生成低冗余矩阵码时，最关键的一步是逻辑共享原则，即把监督位利用上面所提到的多位探测方法进行隐藏，从而达到减少冗余位数的目的。在进行编码时，虽会计算相应的监督位，但不会成为最终的码字，因此不需要存储所有行的监督位，只需要存储根据监督位生成的垂直校验位 r。假设邻位纠错能力为 t，则对 m 行的监督位分成 m/t 个小组，根据公式（2-9），进行异或运算得到垂直校验位 r，具体公式如下所示。

$$r_j = b_j \oplus b_{j+t} \oplus b_{j+2t} \cdots \oplus b_{j+(m/t-1)t} \tag{2-11}$$

式中，r_j 表示第 j 行垂直校验位；

b_j 表示第 j 行汉明码校验位。

根据公式（2-11）可以得到 j 行垂直校验位，当发生相邻错误位数小于纠错能力时，可通过奇偶校验位判断，哪一行发生了错误，然后通过垂直校验位判断发

生错误行的哪一位发生了错误，从而可以确定错误的位置，具体的矩阵如图 2-6 所示。

a_1	a_{1+m}	a_{1+2m}	…	$a_{1+m(n-1)}$	h_1	b_1	b_2	…	b_i
a_2	a_{2+m}	a_{2+2m}		$a_{2+m(n-1)}$	h_2	b_{1+i}	b_{2+i}		b_{2i}
a_3	a_{3+m}	a_{3+2m}		$a_{3+m(n-1)}$	h_3	b_{1+2i}	b_{2+2i}		b_{3i}
a_4	a_{4+m}	a_{4+2m}		$a_{4+m(n-1)}$	h_4	b_{1+3i}	b_{2+3i}		b_{4i}
⋮					⋮				
a_m	a_{2m}	a_{3m}	…	a_{mn}	h_m	$b_{1+(m-1)i}$	$b_{2+(m-1)i}$	…	b_{mi}
						r_1	r_2	…	r_i
						r_{1+i}	r_{2+i}		r_{2i}
						⋮			
						$r_{1+(t-1)i}$	$r_{2+(t-1)i}$	…	r_{ti}

图 2-6 低冗余矩阵排列方法

根据图 2-6 得到低冗余矩阵排列的所有元素之后，可以隐藏生成的汉明码校验位，因此构成编码后码字 C 为{h，a，r}。t 为纠错能力，i 为每行汉明码校验位个数，通过矩阵可以看出数据位个数为 $m \cdot n$，汉明码校验位个数为 $m \cdot i$，奇偶校验位个数为 m，垂直校验位的个数为 $t \cdot i$，因此该码字的总长度为 $m+m \cdot n+ti$。

2.2.2 低冗余矩阵码译码原理

本章所介绍的低冗余矩阵码的译码方式主要是通过汉明码译码方法来进行的，根据 2.1 节中的汉明码校正子的计算方法，首先根据公式（2-8）中的奇偶校验码的校正子计算接收字中每一行的奇偶校验码 h 的校正子 S_h，设 h'为得到数据中的奇偶校验码，h''为根据得到数据生成的奇偶校验码，则奇偶校正子 S_h 的求解公式为：

$$S_h = h' \oplus h'' \tag{2-12}$$

根据 2.2.1 节，设数据为 m 行，因此奇偶校正子 S_h 位数也为 m，若数据中某一行的数据发生一位错误，则对应的奇偶校正子 S_h 会变成“1”，从而可以判断哪一行发生了错误。垂直校验位的校正子 S_r 也根据上述方法进行计算，主要是为了复原编码时隐藏掉的汉明码校正子，设 r'为得到数据的垂直校验位，r''为根据得到数据求得的垂直校验位，S_r 的求解公式为：

$$S_r = r' \oplus r'' \tag{2-13}$$

根据公式（2-12）和（2-13）得到接收字的相应校正子，该校正子可判断数据是否发生错误，即当两个校正子都为 0 时，表明数据没有发生错误，因此当数据

没有错误时，可直接截取数据位进行输出，从而避免纠错操作。若发生错误，则奇偶校验位的校正子和垂直校验位的校正子不为 0，由于不包含错误行的汉明码校正子为 0，因此得到的 S_r 中不为 0 的校正子即为包含错误行的汉明码校正子，设错误行的校正子为 S_{error}：

$$S_{error} = S_r \tag{2-14}$$

由公式（2-14）得到的汉明码校正子可以确定错误行里发生错误的位置，且通过公式（2-12）确定的错误行，就可以确定数据位中发生错误的位置。经过汉明码译码算法得到纠错向量，并与数据进行异或，从而得到纠正后的正确数据，其完整的纠错过程如图 2-7 所示。

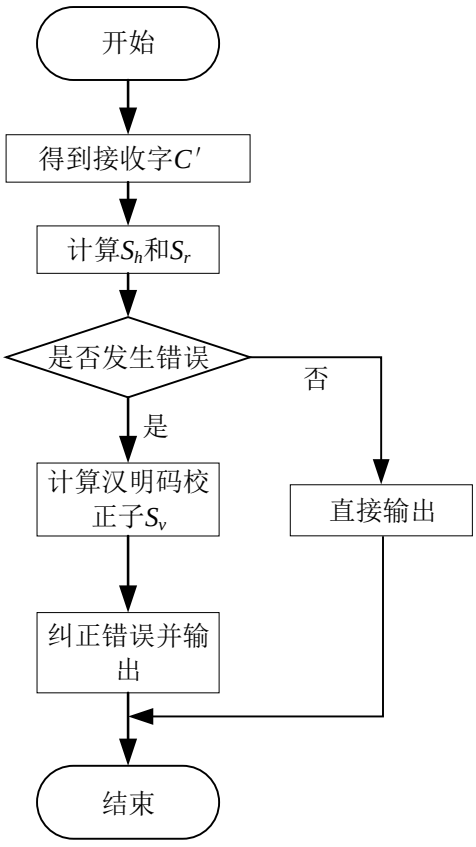


图 2-7 低冗余矩阵码译码算法流程

根据图 2-7 可知，译码的关键在于计算 S_h 和 S_r，并在发生错误时，求得隐藏掉的汉明码校正子 S_v，从而判断错误的位置。低冗余矩阵码在译码时，只要保证奇偶校验码 h 和垂直校验位 r 不同时发生错误，就不会发生数据的误纠现象。因为当两者同时发生错误，数据位没有发生错误时，S_h 和 S_r 都不为 0，从而对正确数据进行纠错操作，导致错误。为了避免同时发生错误，对存储器进行加固设计时，

对码字的顺序进行变更从而避免 h 和 r 同时发生错误，即调整为{h，a，r}的形式。

2.3 低冗余矩阵码设计

根据 2.2 节的低冗余矩阵码的编码与译码理论，基于 Verilog 硬件语言利用已实现的（7，4）汉明码编码译码器，设计实现 $n = 52$，$k = 32$ 的低冗余矩阵码硬件电路。

2.3.1 编码器设计

本文采用（n，k）低冗余矩阵码中，设 n 为 52 位，k 为 32 位，纠错能力 $t = 4$ 来实现低冗余矩阵码编码器。根据 2.2.1 小节中的低冗余矩阵码编码的排列方式，把 32 位数据排成 8 行 4 列的形式，以列为顺序进行排列，并添加 8 位奇偶校验位，24 位汉明码监督位和 12 位垂直校验位，低冗余矩阵码的排列形式如图 2-8 所示。

a_1	a_9	a_{17}	a_{25}	h_1	b_1	b_2	b_3
a_2	a_{10}	a_{18}	a_{26}	h_2	b_4	b_5	b_6
a_3	a_{11}	a_{19}	a_{27}	h_3	b_7	b_8	b_9
a_4	a_{12}	a_{20}	a_{28}	h_4	b_{10}	b_{11}	b_{12}
a_5	a_{13}	a_{21}	a_{29}	h_5	b_{13}	b_{14}	b_{15}
a_6	a_{14}	a_{22}	a_{30}	h_6	b_{16}	b_{17}	b_{18}
a_7	a_{15}	a_{23}	a_{31}	h_7	b_{19}	b_{20}	b_{21}
a_8	a_{16}	a_{24}	a_{32}	h_8	b_{22}	b_{23}	b_{24}
					r_1	r_2	r_3
					r_4	r_5	r_6
					r_7	r_8	r_9
					r_{10}	r_{11}	r_{12}

图 2-8 32 位低冗余矩阵码排列

图 2-8 中，h_1 为第一行的奇偶校验位，b_1-b_3 为第 1 行的数据 $a_1a_9a_{17}a_{25}$ 的 3 位汉明码监督位，r_1-r_3 为第 1 行和第 4 行的监督位相异或得到的 3 位垂直校验位，以此类推，得到所有的校验位和监督位。根据图 2-8 中，32 位低冗余矩阵码的排列方式，每一行都要算出校验码，由于每行的数据位为 4 位，因此采用的汉明码为（7，4）汉明码并添加奇偶校验位，具体实现已在 2.1.2 节中详细介绍。利用图 2-2 中的汉明码编码器，将每一行的 4 位数据分别进行汉明码编码，从而得到 3 位

监督位和奇偶校验位，详细的数据位与奇偶校验位和监督位的关系如下所示。

$$\begin{aligned}
&b_1 = a_1 \oplus a_{17} \oplus a_{25} \\
&b_2 = a_1 \oplus a_9 \oplus a_{17} \\
&b_3 = a_9 \oplus a_{17} \oplus a_{25} \\
&h_1 = a_1 \oplus a_9 \oplus a_{17} \oplus a_{25} \\
&\vdots \\
&b_{22} = a_8 \oplus a_{24} \oplus a_{32} \\
&b_{23} = a_8 \oplus a_{16} \oplus a_{24} \\
&b_{24} = a_{16} \oplus a_{24} \oplus a_{32} \\
&h_8 = a_8 \oplus a_{16} \oplus a_{24} \oplus a_{32}
\end{aligned} \tag{2-15}$$

根据公式（2-15）求得每行的奇偶校验位和监督位之后，结合 2.2.1 小节中垂直校验位的算法原理，由于纠错能力 t 为 4，把 8 行监督位分成 2 组，每隔四行进行异或得到垂直校验位 r。根据公式（2-11）得到表 2-7 的垂直校验位与汉明码监督位的详细关系。

表 2-7 垂直校验位和监督位的关系

垂直校验位	监督位
r_1	$b_1 \wedge b_{13}$
r_2	$b_2 \wedge b_{14}$
r_3	$b_3 \wedge b_{15}$
r_4	$b_4 \wedge b_{16}$
r_5	$b_5 \wedge b_{17}$
r_6	$b_6 \wedge b_{18}$
r_7	$b_7 \wedge b_{19}$
r_8	$b_8 \wedge b_{20}$
r_9	$b_9 \wedge b_{21}$
r_{10}	$b_{10} \wedge b_{22}$
r_{11}	$b_{11} \wedge b_{23}$
r_{12}	$b_{12} \wedge b_{24}$

根据表 2-6 得到垂直校验位后，隐藏每行生成的汉明码监督位，并组成{h_1-h_8,a_1-a_{32},r_1-r_{12}}的形式进行输出，低冗余矩阵码编码器详细结构如图 2-9 所示。输入数据{a_1-a_{32}}首先分成 8 行，每行 4 位信息位经过汉明码编码器编码之后得到汉明码监督位 b 和奇偶校验位 h，每两行汉明码监督位 b 相异或得到垂直校验位 r，最终构成低冗余矩阵码的码字{h_1-h_8,a_1-a_{32},r_1-r_{12}}而输出，从而实现低冗余矩阵码的编码设计。

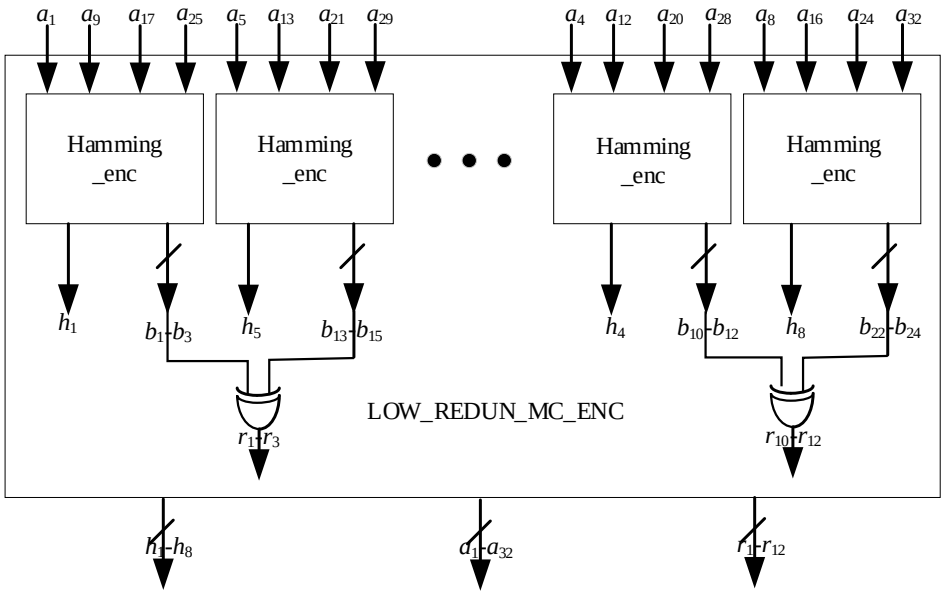


图 2-9 低冗余矩阵码编码器结构

2.3.2 译码器设计

由 2.2.2 节可知，低冗余矩阵码译码的关键在于生成相应的校正子以及恢复隐藏的汉明码校正子，从而判断接收字中是否发生错误。假设译码器接收的数据为 $\{h'_1\text{-}h'_8, a'_1\text{-}a'_{32}, r'_1\text{-}r'_{12}\}$，根据图 2-7 中译码的顺序，计算接收数据的奇偶检验位 h''、汉明码校验位 b''和垂直校验位 r''，具体公式如下：

$$
\begin{aligned}
b''_1 &= a'_1 \oplus a'_{17} \oplus a'_{25} \\
b''_2 &= a'_1 \oplus a'_9 \oplus a'_{17} \\
b''_3 &= a'_9 \oplus a'_{17} \oplus a'_{25} \\
h''_1 &= a'_1 \oplus a'_9 \oplus a'_{17} \oplus a'_{25} \\
&\vdots \\
b''_{22} &= a'_8 \oplus a'_{24} \oplus a'_{32} \\
b''_{23} &= a'_8 \oplus a'_{16} \oplus a'_{24} \\
b''_{24} &= a'_{16} \oplus a'_{24} \oplus a'_{32} \\
h'_8 &= a'_8 \oplus a'_{16} \oplus a'_{24} \oplus a'_{32} \\
r''_1 &= b''_1 \oplus b''_{13} \\
r''_2 &= b''_2 \oplus b''_{14} \\
&\vdots \\
r''_{12} &= b''_{12} \oplus b''_{24}
\end{aligned}
\qquad (2\text{-}16)
$$

根据接收数据算出的 h''，b''和 r''，算出数据的奇偶校正子 S_h 和垂直校正子 S_r，具体求解参考公式（2-12）和公式（2-13）。计算求得的 8 位奇偶校正子 S_h 和 12 位垂直校正子 S_r 中没有“1”时，设定纠正标志信号 correct_flag 为 0，表明数据无错，不需要进行纠正操作，直接输出数据$\{a'_1\text{-}a'_{32}\}$，否则根据 S_h 判断哪一行有错误。例如求得 S_h 为“00110000”，则表明矩阵中第 3 行和第 4 行有错误，再根据公式（2-14）算得错误行的汉明码校正子。例如垂直校正子 S_r 为“000_000_111_111”，可以推出第 3 行的汉明码校正子为“111”，第 4 行汉明码校正子为“111”，则通过表 2-2 里（7，4）汉明码中校正子与错误位置的关系推断出第 3 行的第 3 位和第 4 行的第 3 位发生错误，即 a_{19} 和 a_{20} 发生错误。判断出错误位置之后，根据（7，4）汉明码译码器和汉明码校正子，利用纠错向量对错误数据进行纠正之后进行输出，从而完成低冗余矩阵码的译码，详细的译码器模块如图 2-10 所示。

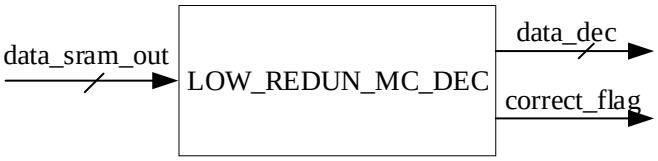


图 2-10 低冗余矩阵码译码器

根据图 2-10 中的译码器模块，详细的输入输出端口信号如表 2-8 所示。

表 2-8 低冗余矩阵码译码器输入输出

信息名称	输入输出	信号说明
data_sram_out	输入	译码器接收数据（52 位）
data_dec	输出	译码之后数据（32 位）
correct_flag	输出	纠正标志信号，高电平表示数据被纠正

2.4 功能仿真验证

根据 2.3 节利用 Verilog 硬件语言实现的（52，32）低冗余矩阵码的编码译码器，在 ModelSim 平台上进行仿真，并利用 force 语句对设计进行故障注入，验证设计是否正确。先对编码器进行仿真验证，对低冗余矩阵码编码器输入 data_in 输入数据 32’b00010001_00010001_00010001_00010001，仿真结果如图 2-11 所示。

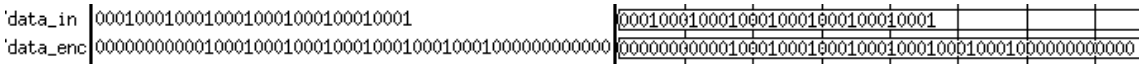


图 2-11 低冗余矩阵码编码波形

根据 2.3.1 中，对 32 位数据的编码运算之后按$\{h，a，r\}$输出的结果与图 2-11 中编码之后的数据 data_enc = 52’b00000000_0001000100010001000100010001000 1_000000000000 相同，前 8 位表示奇偶校验位，中间 32 位为数据位，后 12 位为

垂直校验位，表明实现的低冗余矩阵码编码器的功能正确。

验证编码器功能正确之后，验证译码器功能是否正确。根据编码器得到的结果，在译码器输入端 data_sram_out 输入相同的数据，根据输出结果是否与输入到编码器的数据相同来判断译码器是否正确译码，图 2-12 低冗余矩阵码译码器的波形。

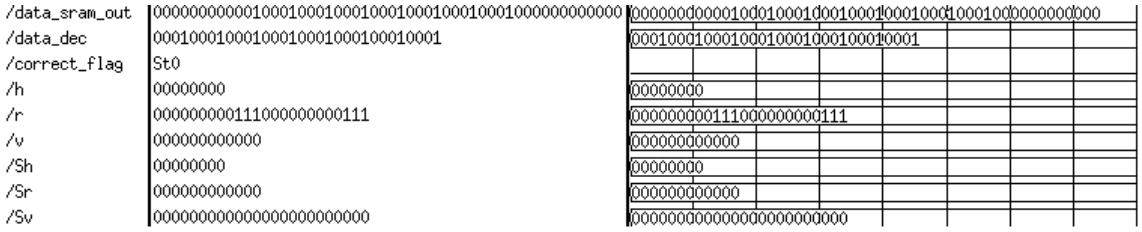
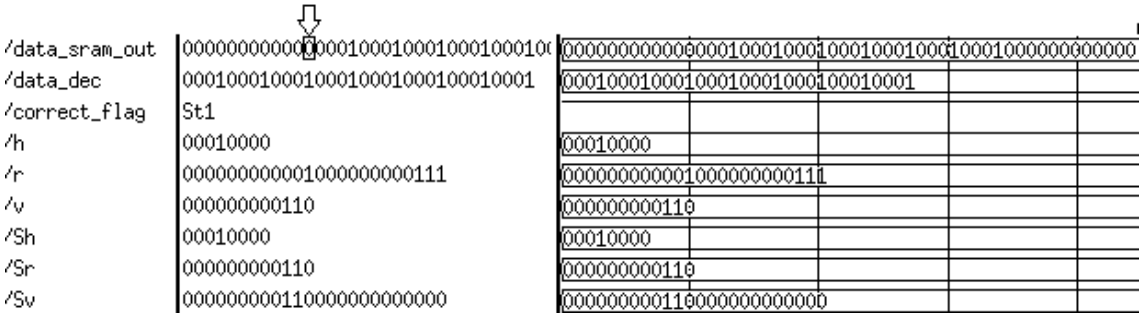


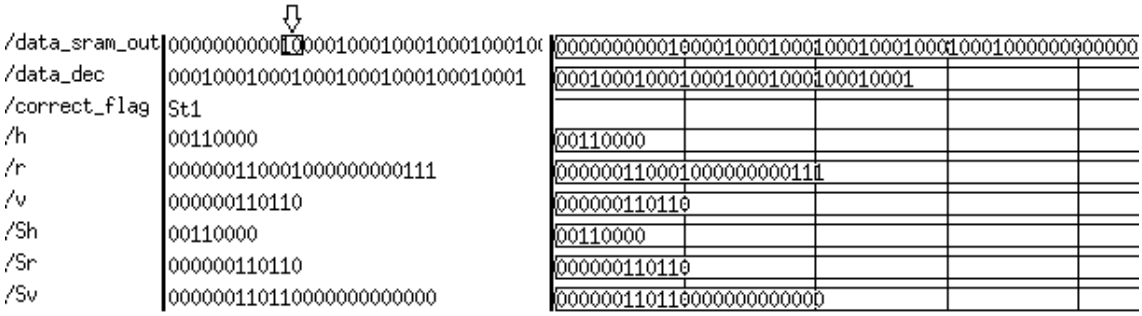
图 2-12 低冗余矩阵码译码波形

从图 2-12 中，可以看出根据接收数据得到的 h、v、r 的结果与公式（2-17）所得到的结果相同，此时没有注入错误，因此奇偶校正子 S_h 和垂直校正子 S_r 均为 0，因此纠正标志信号 correct_flag 为低电平。通过对比输出结果 data_dec 与编码器的输入 data_in 的值，可知该译码器正确译码。

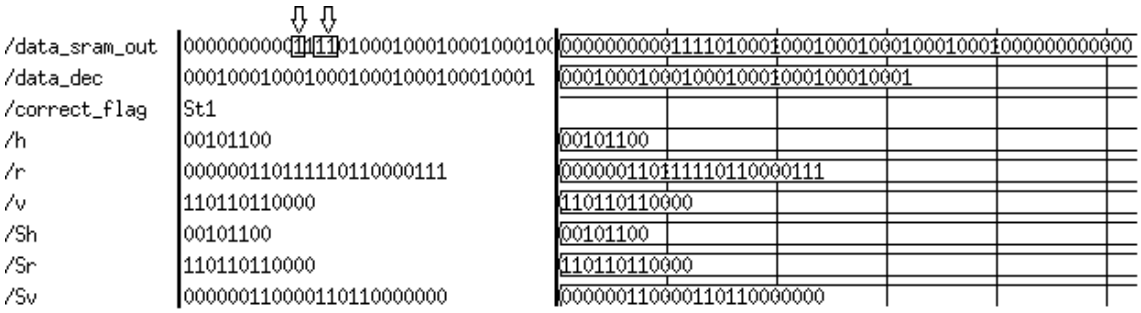
下面验证数据发生连续错误时，译码器的纠错能力。利用 force 语句和 release 语句来对译码器输入数据 data_sram_out 分别注入不同的连续错误，从而模拟相应位上翻转，详细的模拟仿真如图 2-13 所示。



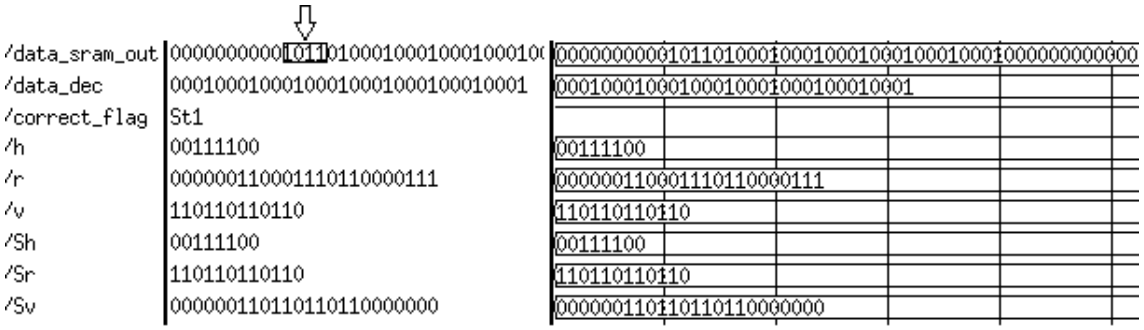
a) 注入单位错误



b) 注入连续两位错误



c) 4 个相邻位的不连续 3 位错误



d) 连续 4 位错误

图 2-13 译码器纠错仿真波形

图 2-13 中，箭头表示注入故障的位置，可以看出，连续 4 位数据内的任何一种故障形式，都可以被译码器纠正，且输出纠正标志信号 correct_flag = 1，同时 data_dec 输出正确数据。这就表明该（52，32）低冗余矩阵码译码器电路可以对单位错误、连续两位错误、连续三位错误、不连续的三位错误和连续四位错误进行纠正。

2.5 本章小结

本章首先介绍了线性分组码的基本理论，码字的构成和校正子的纠错原理，并重点介绍与汉明码相关的纠错原理，同时采用 Verilog 硬件语言实现了（7，4）汉明码的编码器和译码器电路。基于汉明码的理论分析，介绍了低冗余矩阵码的编码理论和译码理论。结合（7，4）汉明码编码译码电路，利用硬件语言实现了可以应对连续 4 位错误的（52，32）低冗余矩阵码，并在 ModelSim 仿真平台进行了验证。该电路可以正确纠错连续四位中不同类型的错误模式，从而验证了该低冗余矩阵码的编码器和译码器的正确性。

第3章 软错误修复硬错误探测的 SRAM 设计与验证

本章基于第 2 章实现的低冗余矩阵码电路，对 Memory-Compiler 生成的存储器进行加固，并添加自检测修复电路，使其能够在读取数据时，检测数据发生的是软错误还是不可修复的硬错误，并针对软错误进行修复。本章还将通过故障注入的方法，验证所设计的整体电路是否能够正确地区分错误类型，修复存储单元内的软错误，以及探测不可修复硬错误。

3.1 基于低冗余矩阵码的存储器加固设计

3.1.1 存储器模型

首先要生成存储器模型，并对其进行加固设计。本文采用 Memory-Compiler 工具生成 SMIC65nm 工艺的单端口 SRAM 存储器。设定宽度为 52 位，深度为 2k，频率为 125MHz，并设定 write-though 为 1，表明进行写操作时，数据存入当前地址之后，会输出当前地址的数据，详细的存储器模块及其主要输入输出如图 3-1。表 3-1 为详细的存储器输入输出端口说明。

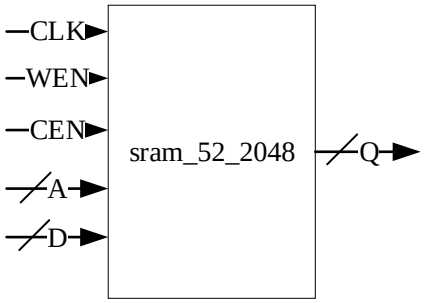


图 3-1 52x2k 存储器模型

表 3-1 存储器输入输出定义

信号名称	方向	信号说明
CLK	输入	时钟信号
WEN	输入	写使能，低电平有效
CEN	输入	片选信号，低电平有效
A	输入	地址信号（11 位）
D	输入	输入数据（52 位）
Q	输出	输出数据（52 位）

由图 3-1 和表 3-1 中，当片选信号为低电平，即 CEN = 0 时，才能对该存储器

进行操作。CLK 时钟上升沿来临时，若 WEN = 1，进行读操作，若 WEN = 0，进行写操作，该存储器模型的读写时序如图 3-2 和图 3-3 所示。

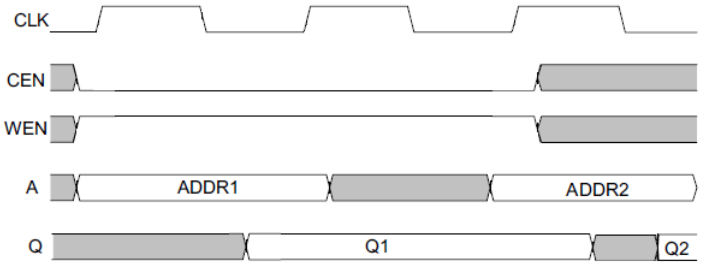


图 3-2 单端口 SRAM 存储器读时序

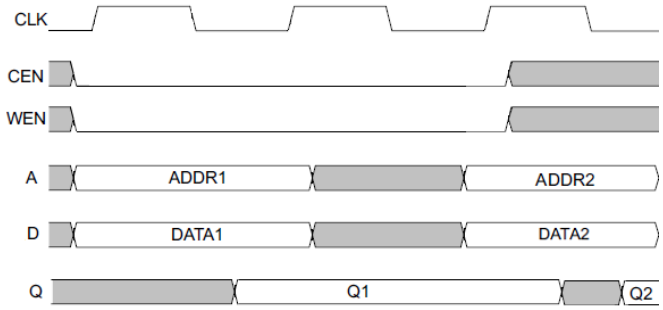


图 3-3 单端口 SRAM 存储器写时序

由图 3-2 和图 3-3 可知，存储器模型相关的详细信号操作如下，第一：存储器仅在 CLK 的上升沿接收数据；第二：存储器在片选信号 CEN 为 0 时工作；片选信号为 1 时，输出会保持上一个输出状态；第三：CEN 等于 0，WEN 等于 0 时，进行写操作，即把当前输入数据 D 存入到当前地址 A 之后，由于设 write-through 为 1，会输出已存入的数据；第四：CEN 为 0，WEN 为 1 时，输出端会输出存储在当前地址 A 的数据。

3.1.2 存储器加固

基于 Memory-Compiler 生成的存储器，在存储器的数据输入端 D 和数据输出端 Q 添加第 2 章设计的可纠正连续 4 位错误的低冗余矩阵码电路，使其对 32 位输入数据进行编码之后，把得到的 52 位码字存储到存储器内，并在输出时，存储在地址中的 52 位数据，先经过译码器译码之后，最后输出 32 位译码数据。因此即使存储器内某个位发生了错误，只要错误位数满足低冗余矩阵码的纠错能力，都会在译码纠正之后得到正确的数据，详细的加固设计如图 3-4 所示。其整体存储器加固设计的输入输出详细说明如表 3-2 所示。

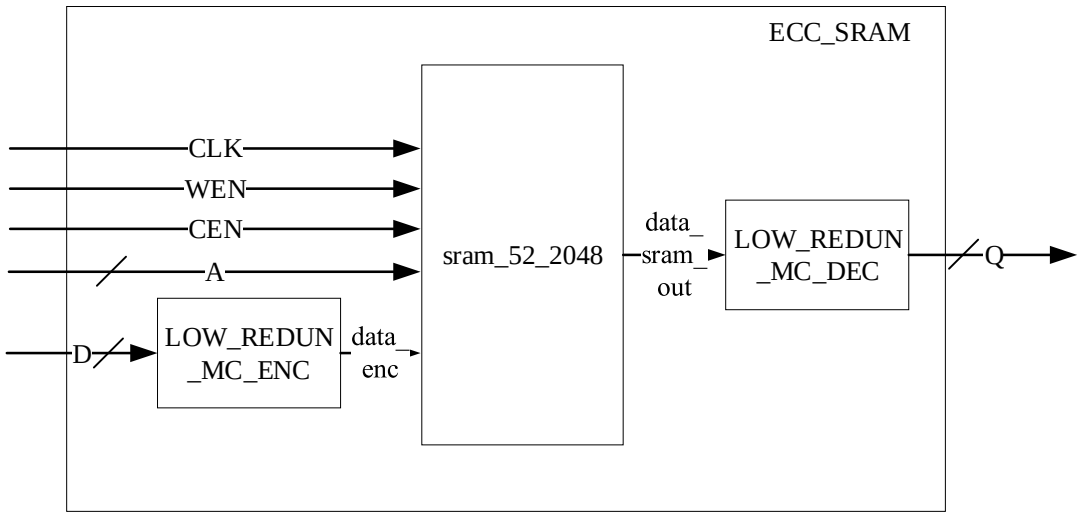


图 3-4 存储器加固设计

表 3-2 存储器加固输入输出信号定义

信号名称	方向	信号说明
CLK	输入	时钟信号
WEN	输入	写使能，低电平有效
CEN	输入	片选信号，低电平有效
A	输入	地址信号（11 位）
D	输入	输入数据（32 位）
Q	输出	输出数据（32 位）

该存储器实际上相当于宽度为 32 位，深度为 2K 的存储器。32 位数据经过低冗余矩阵码编码之后变成 52 位的码字 data_enc，当输出数据时，存储器输出数据 data_sram_out 经过低冗余矩阵码译码器译码之后变成 32 位数据 Q 输出，从而利用编码译码算法试下对存储数据的加固。因此当存储器内某地址发生错误时，可根据译码器的纠正能力使其恢复到正确数据，基于低冗余矩阵码加固的存储器的读写时序如图 3-5 和 3-6 所示。

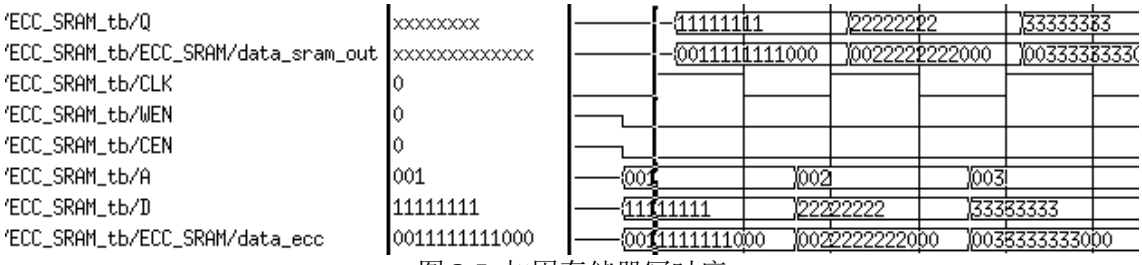


图 3-5 加固存储器写时序

图 3-5 为加固存储器的写时序，可以看出此时片选信号 CEN 为 0，WEN 为 0，当时钟上升沿到来后存储器进行写操作。具体的写操作过程如下，数据

32’h11111111 首先经过编码器编码成码字 52’h0011111111000，之后已完成编码的数据将存入到地址 11’h001。

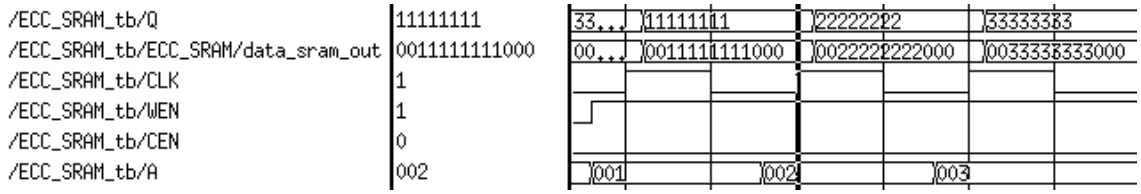


图 3-6 加固存储器读时序

图 3-6 中为加固存储器的读时序，此时 CEN 为 0，WEN 为 1，在时钟上升沿到来后存储器进行读操作。具体操作过程如下：存储器首先输出地址 11’h001 的数据 52’h0011111111000 到译码电路，译码电路经解码后得到 32 位数据 32’h11111111，从而完成数据输出。由于编码器与译码器都是通过组合逻辑来运算，用低冗余矩阵码加固的存储器的读写时序也满足单周期输出，不占用时钟周期。

3.2 软错误修复与检测硬错误设计

本节基于 3.1 节实现的利用低冗余矩阵码加固的存储器，设计自检测硬错误并修复软错误电路。该电路的特点是能够区分存储器内发生的错误类型，探测是否发生硬错误并且修复存储器内软错误。

3.2.1 软错误修复与硬错误检测

由文献[16][17]可知，重离子会导致存储器发生硬错误，从而引起“固定位”错误，此后，无论对该位置写入任何数值，其发生硬错误的“固定位”都会保持原来的数据。而正常因单粒子翻转引起的软错误是可以通过写回正确数据来修复原存储器内发生的错误的。

基于这一特点，若假定某个地址发生的错误满足 ECC 电路的纠错能力，当低冗余矩阵码译码器探测到错误时，采取写回操作，使其正确数据写回到原地址，之后再次读出该数据到译码器就可以验证所发生的错误是软错误还是硬错误，具体如下。若再次读出时没有发现错误，就表明该地址发生了可通过重写来消除的软错误，若发现错误仍然存在，则表明该地址存在不可修复的“固定位”错误，即探测到了硬错误。

综上，增加回写操作不仅能够区分出软错误和硬错误，还能在此过程中完成对存储单元内软错误的修复，为此本节设计了如图 3-7 所示的探测错误类型的 SRAM 结构。

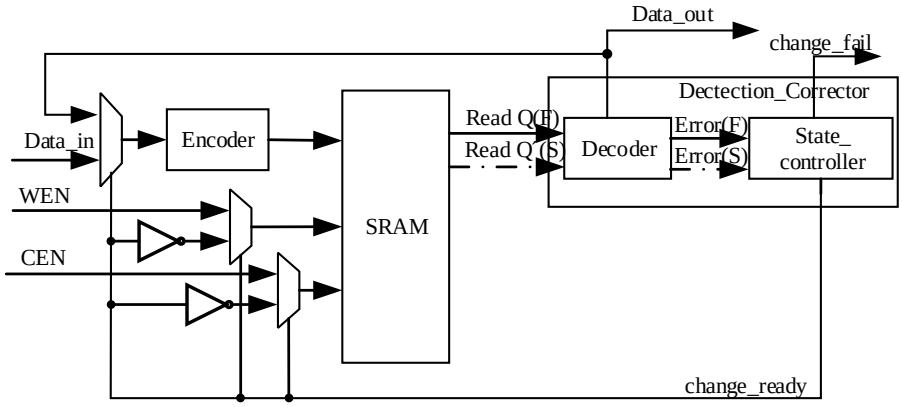


图 3-7 软错误修复与硬错误探测的 SRAM 结构

如图 3-7 所示，该结构包含编码器，SRAM 存储器与由译码器和状态控制器构成的探测纠正器。图中 F 为首次，S 为第二次，change_ready 为修正控制信号；在写入数据时，输入数据首先进入编码器，编码器对接收到的数据进行编码，并把生成的监督码一同存储在 SRAM 中；在读取数据时，首先通过译码器检测读取的数据 Q 是否有错误。如果没有错误，则直接输出数据 Q；当 Q 有错误时，译码器的纠正标志信号会告知状态控制器，此时状态控制器会将译码之后的数据重新送回到编码器输入端再次进行编码，同时控制存储器的读写信号，使编码后的数据写到原地址并再次读出（此时的读出数据标记为数据 Q′）。Q′通过译码器再次做出判断，如果无错则说明该地址之前存储的数据发生了软错误并且已经被修正；如果有错则说明该地址下的数据错误无法通过写回修正，译码器发出纠正标志信号，告知系统该地址有不可修正的“固定位”硬错误。例如地址 11’h001 的原数据为 52h’0011111111000，由于发生了单粒子翻转，该数据发生了连续 4 位的软错误，变成了错误数据 52’h0011e11111000。当上述结构读到地址 11’h001 时，低冗余矩阵码译码器接收到错误数据，并对其进行译码之后，纠正标志信号 correct_flag 置 1，告知状态控制器，该地址有错误。状态控制器将译码之后的数据发送回编码器再次编码，并将写使能发送到存储器，控制存储器把重新编码的数据 52h’0011111111000 再次写入当前地址中，并再次读出到译码器进行检测译码。由于软错误在此过程中被覆盖，所以再次输出的数据没有错误，从而正常输出正确数据。但是硬错误通过回写无法被覆盖，因此重新读出的数据仍将有错误，译码器会再次把纠正标志信号 corret_flag 置 1，告知系统该地址有硬错误。

3.2.2 状态控制器设计

通过 3.2.1 节的分析与讨论可知，探测修复架构的关键在于对存储器读写的控

制，即核心电路为控制整体修复和探测的状态控制器。

控制流程中存在有两个关键问题：第一，纠正数据的写回操作与存储器读写信号控制，本文需要采取多路选择器来控制输入数据以及写使能。即当译码器发生错误时，状态控制器发出修复控制信号 change_ready 到多路选择器，多路选择器使外部的输入数据和写使能信号被旁路，从而接收译码器发出的输入数据和状态控制器发出的写使能信号，相关的代码如图 3-8 所示。

```
assign   cn_change = ((change_ready)) ? ~change_ready : CEN;
assign   wr_change = ((change_ready)) ? ~change_ready : WEN;
assign   data_change = (change_ready) ? DATA_OUT :  DATA_IN;
```

图 3-8 存储器读写控制

wr_change 信号是实际连接到存储器写实使端的信号，cn_change 信号是实际连接到存储器片选端的信号，data_change 为连接到编码器输入端的输入数据信号，从而完成了对存储器读写的控制。即当译码器发现数据有错误时，控制存储器输入端，完成正确数据的重新写入。

第二个关键问题是输出数据的二次检测机制：译码器如何分辨该数据是第 1 次还是第 2 次探测，需要引入有限状态机来区分每一种状态，二次检测方法的具体流程如图 3-9 所示。

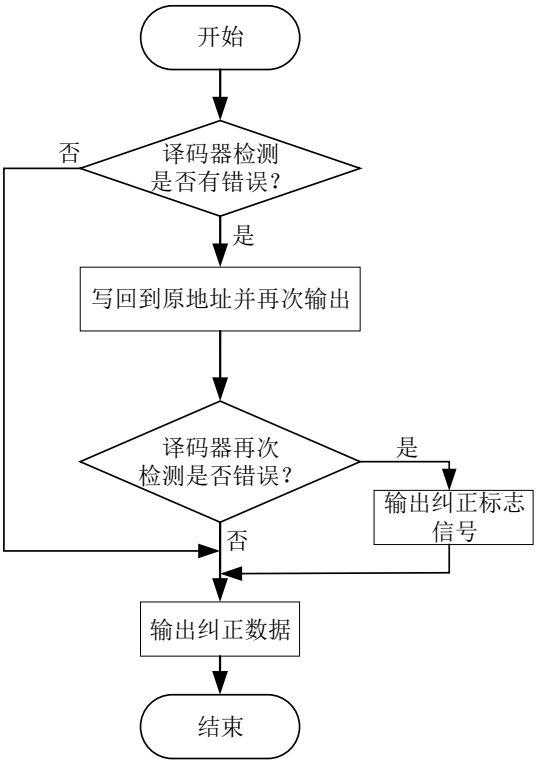


图 3-9 二次检测流程

基于图 3-9 中二次检测流程，设计相关的有限状态机，共设置 3 个状态，从而区分第一次与第二次，详细的状态控制器如图 3-10 所示。

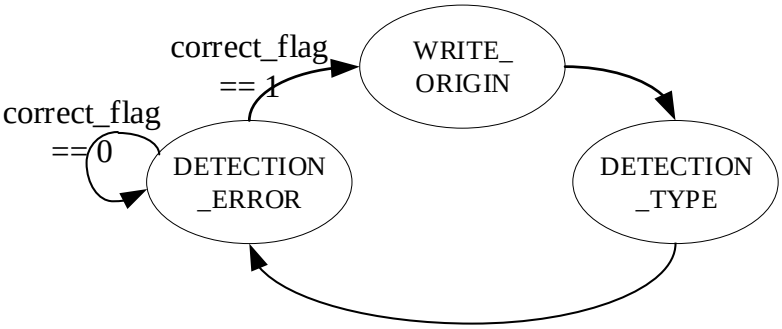


图 3-10 2 次检测状态转换图

分析图 3-10 可知，为了保证译码器认知是第二次检测，利用有限状态机分开各个状态，整个状态的主要变化依据的是低冗余矩阵码译码器发出的纠正标志信号 correct_flag：（1）DETECTION_ERROR 为初始状态，判断存储器第一次输出数据是否有误，即探测纠正标志信号 correct_flag 是否为 0；（2）探测纠正标志信号 correct_flag 为 0 时，回到初始状态 DETECTION_ERROR，再次检测下一个地址的数据；（3）当初次检测探测纠正标志信号 correct_flag 不为 0 时，当前状态先跳转到 WRITE_ORIGIN，即等待状态，change_ready 信号变成 1，外部数据输入和写使能 WEN 被旁路，从而控制存储器的读写，译码后数据送到编码器再次编码之后写回到原地址；（4）写入完毕之后，状态跳到 DETECTION_TYPE，存储器再次输出该地址数据，译码器再次译码之后输出纠正数据，如果纠正标志信号 correct_flag 为 0，表明存储器内的软错误已被修正，并输出正确数据。否则，纠正标志信号 correct_flag 连接到 change_fail 进行输出，表明在该地址探测到不可修复的硬错误。输出完毕之后，状态跳回初始状态 DETECTION_ERROR，再对下一个地址的数据进行检测。从而状态控制器的模块及输入输出如图 3-11 所示。状态控制器模块输入输出信号的详细定义如表 3-3 所示。

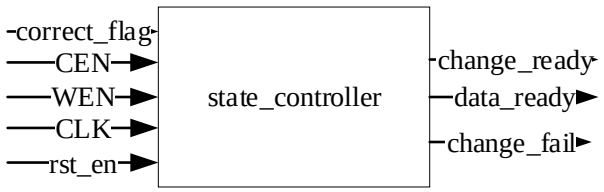


图 3-11 状态控制器模块设计

表 3-3 状态控制器信号定义

信号名称	方向	说明
CLK	输入	时钟信号
rst_en	输入	复位信号
CEN	输入	外部存储器使能信号，低电平有效
WEN	输入	外部存储器写实能信号，低电平有效
correct_flag	输入	纠正标志信号，高电平表示数据被纠正
change_ready	输出	多路选择器控制信号，高电平有效
data_ready	输出	数据准备信号，高电平表示数据有效
change_fail	输出	修复失败信号，高电平表示探测到硬错误

分析图 3-11 和表 3-3 可知，状态控制器只在存储器读操作时才会工作，即当 CEN = 0，WEN = 1 时，图 3-10 中的有限状态机才会启动。由于写回数据需要时间，添加了数据准备信号 data_ready，告知系统当前数据是否已经准备就绪。具体操作如下：没有错误时，数据准备信号 data_ready 为 1，当译码器发现错误时，数据准备信号 data_ready 变 0，表示当前数据需要修复，再次写回之后，data_ready 重新变为1，表示当前数据有效。在第二次检测时，如果探测到硬错误，则 change_fail 信号置 1，表示该地址有不可修复的硬错误。

3.2.3 整体结构设计

基于上一小节中的整体结构，结合已设计的状态控制器，Memory-Compiler 生成的 52 位 x2048 存储器模型，设计软错误修复与硬错误检测的 SRAM 整体结构，整体的软错误修复与硬错误探测 SRAM 模块框图如图 3-12 所示，其各输入输出信号的定义如表 3-4 所示。

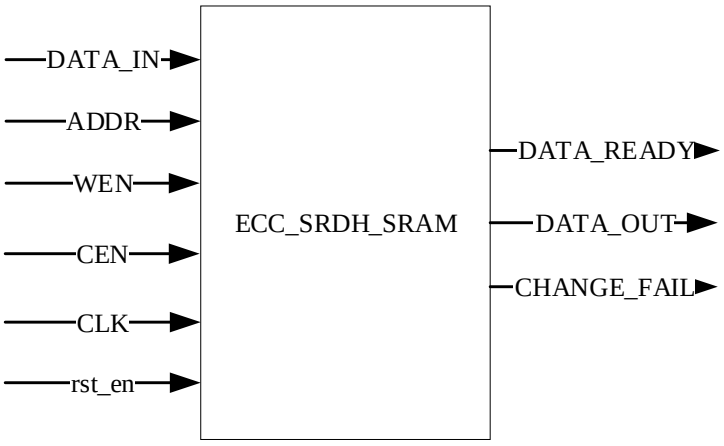
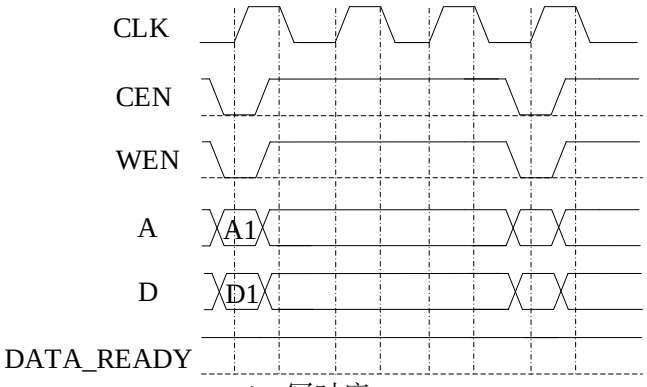


图 3-12 软错误修复与硬错误检测的 SRAM

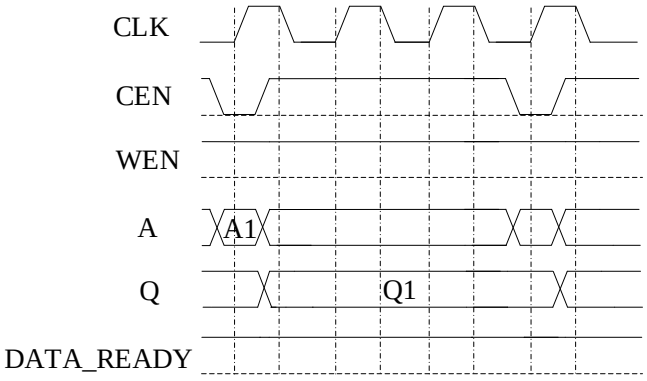
表 3-4 整体架构信号定义

信号名称	方向	信号说明
CLK	输入	时钟信号
rst_en	输入	复位信号
WEN	输入	写使能，低电平有效
CEN	输入	片选信号，低电平有效
ADDR	输入	地址信号（11 位）
DATA_IN	输入	输入数据（32 位）
DATA_OUT	输出	输出数据（32 位）
DATA_READY	输出	数据准备信号，高电平表示数据有效
CHANGE_FAIL	输出	修复失败信号，高电平表示探测到硬错误

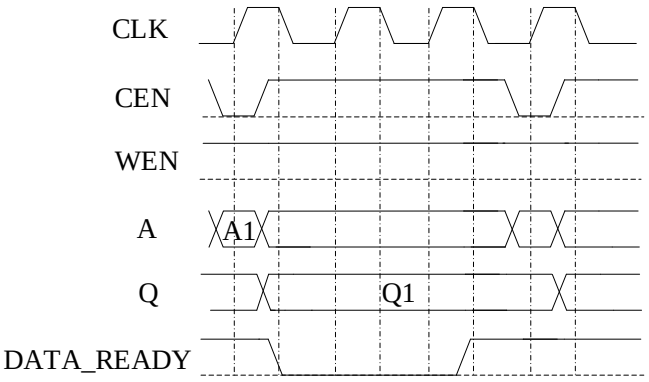
由于结构内部有自检测修复的状态控制器，因此相应的时序会产生变化。当译码器检测到错误时，写回再输出需要一定的周期时间，因此利用 DATA_READY 来表示输出数据的有效情况，该 SRAM 结构的详细的时序如图 3-13。



a) 写时序



b) 读时序（无错误）



c) 读时序（检测到错误）

图 3-13 各状态读写时序

第一种情形是存储器的写操作，可以从图 3-13.a 中看出，由于进行写操作时，不会启动译码器检测功能，因此写时序与正常存储器的时序相同。第二种情形是存储器在读操作时，如图 3-13.b，译码器未发现数据有错误，这时也不会启动修复策略，因此读时序也与正常存储器的读时序相同，且数据准备信号 DATA_READY 一直处于高电平逻辑状态。第三种情形是存储器在读操作时，译码器检测到输出数据有错误，如图 3-13.c，这时状态控制器控制存储器将纠正数据写回到原地址，存入数据需要一个周期的时间，再次读出需要一个周期的时间。因此总共需要两个周期，数据准备信号 DATA_READY 会在检测到错误时，拉低两个周期的时间，等数据修复完毕之后，拉高数据准备信号 DATA_READY 表示修复完成，且当前数据有效。

3.3 故障注入平台及验证

3.3.1 软错误修复

这里首先对本设计是否能够正确修复存储器内的软错误进行验证。建立故障注入平台，验证存储器的软错误是否被修复的思路是在 ModelSim 平台下，对存储器的所有位置先进行写操作仿真，保证存储器内所有地址都有数据之后，暂停仿真进度。然后提取存储器内的数据并对数据进行故障注入，将注入完故障的存储器数据载入到当前仿真的存储器之中，从而模拟存储单元发生了软错误。然后再次启动仿真，并对存储器的所有位置进行读操作，保证读完所有位置的数据之后，结束所有仿真。提取当前存储器内数据，并与做完写操作之后的存储器数据进行对比，判断是否相同。如果相同，则表明存储器的软错误数据修复正确。写操作

的仿真图如图 3-14 所示。

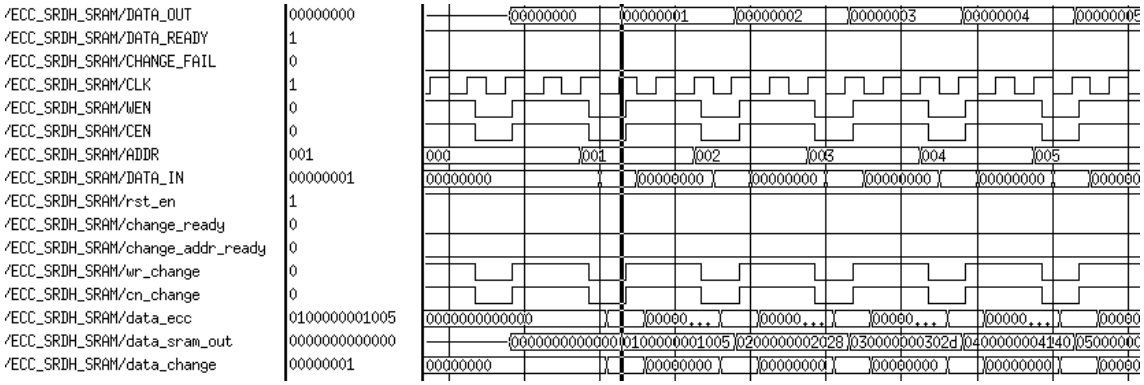


图 3-14 写操作仿真图

分析图 3-14 可知，每次 CLK 上升沿到来时，存储器会接收相应地址和输入数据，并进行存储，写操作完毕之后，暂停仿真进度，并提取存储器内数据，存储器内数据如图 3-15 所示。

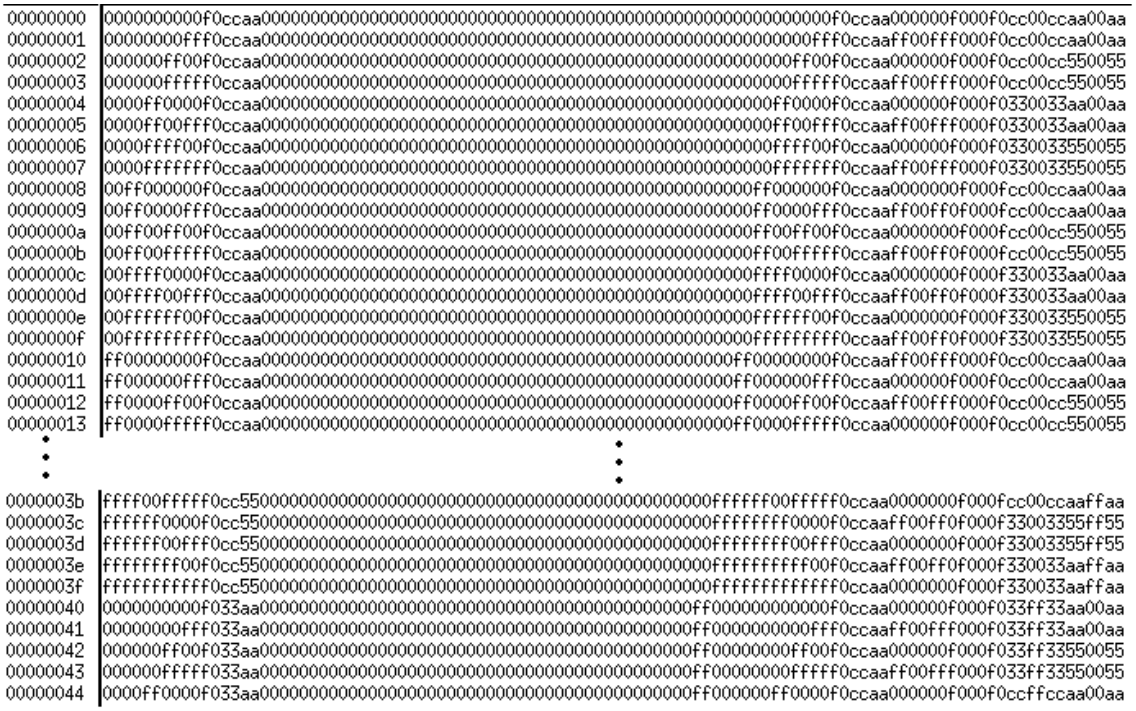


图 3-15 存储器内数据（写操作之后）

提取存储器内数据，并生成 before_fault.men 数据文件。然后直接对该数据文件进行故障注入，即修改 before_fault.men 文件内的数据，从而生成 after_faut.men 文件。即模拟出发生软错误的情形，注入故障之后的数据文件 after_faut.men 与原始数据文件 before_fault.men 的对比如图 3-18 所示。

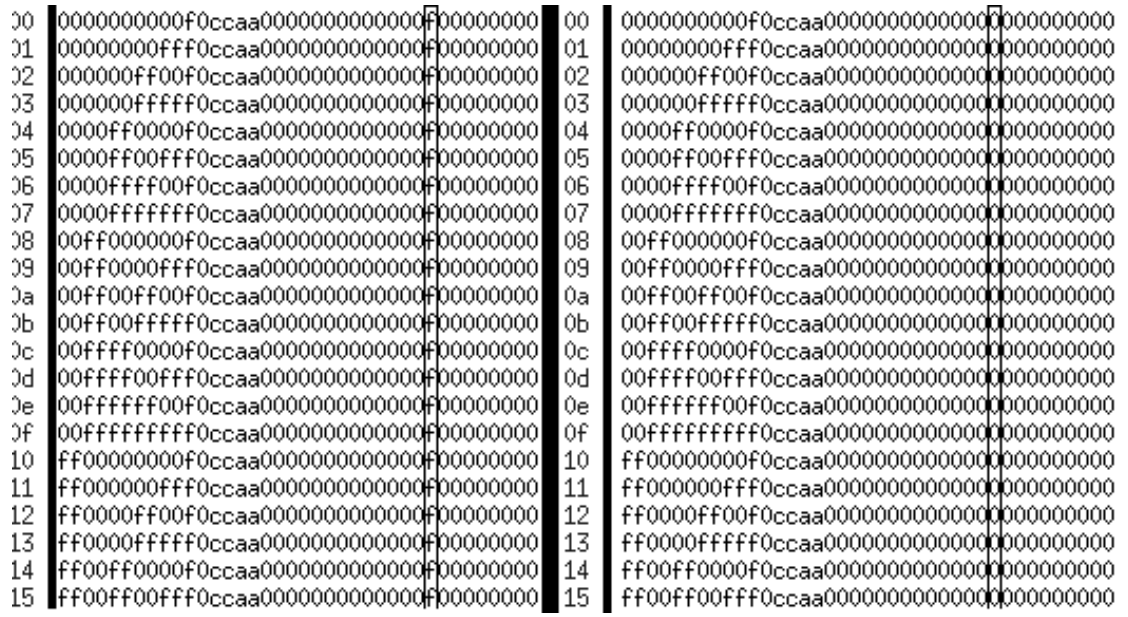


图 3-16 故障注入数据与原数据对比

从图 3-16 可以看出，对比原数据，存储器内的数据文件中第 30 列数据由 0 变成 f，即注入了软错误。把注入完故障的数据文件 after_faut.men 载入到当前仿真的存储器内，然后开启仿真，对每个存储器地址进行读操作，从而修复存储器内的软错误，读操作的仿真如图 3-17 所示。

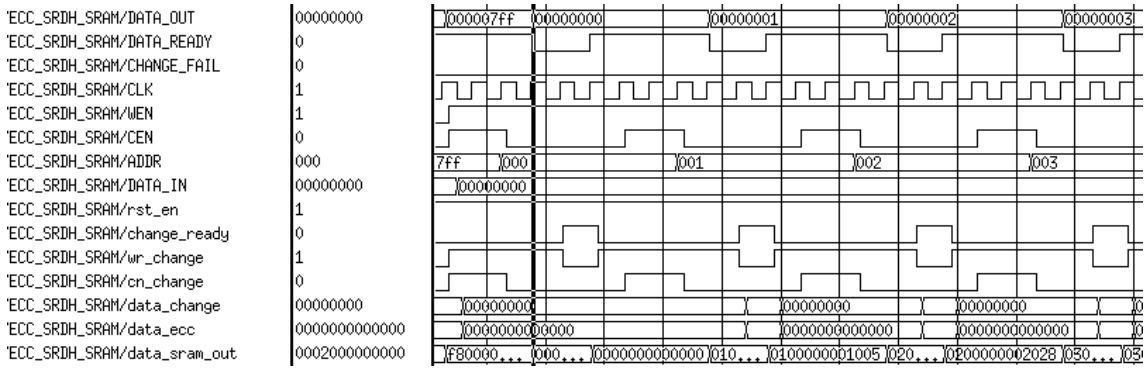


图 3-17 读操作仿真

可从图 3-17 的读操作仿真中看出，每次检测到存储器内的软错误时，先是拉低 DATA_READY 信号，然后 change_ready 置 1，使写实能有效，从而把纠正数据 data_change 写回到原地址当中，然后进行再次输出。在输出的过程中，译码器会再次检测数据是否有错误，因存储器内的软错误已被修复，数据有效信号 DATA_READY 已被置为高电平，从而完成对软错误的修复。对存储器的所有地址完成读操作之后，提取存储器内的数据，生成 after_repair.men 文件，与原数据 before_fault.men 进行对比，如图 3-18 所示。

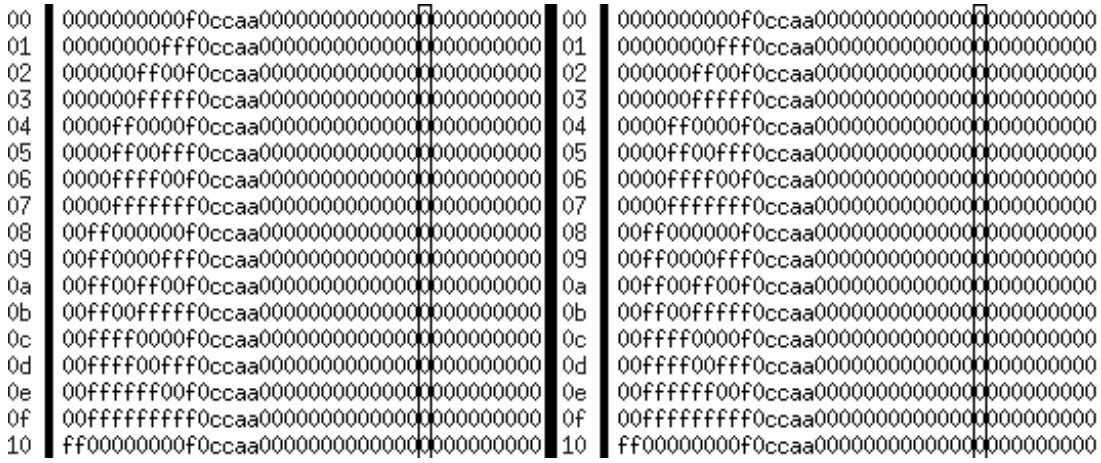


图 3-18 修复之后数据与原数据对比

可以从图 3-18 中看出，第 30 列故障数据 f 变成了正确数据 0，因此经过本设计的读操作，存储器内的数据从有故障的数据变成了没有故障的数据。从而完成了对存储器内软错误数据的修复，这也证明了该设计在错误位数满足纠正能力的情况下，可以修复存储器内发生的软错误，减少了传统加固方法因错误积累而无法进行纠正的风险。

3.3.2 硬错误探测

为了验证该设计是否能检测出不可修复的硬错误，需要本文模拟出硬错误的特性。硬错误的特性在于单元数据不变，即当地址一定时，一直输出错误数据。因此应在存储器进行读操作时，应让存储器的故障数据输出保持到第二次检测的时刻，从而模拟出存储器的单元不会通过写回覆盖而消除的状况。因此，本文利用 force 语句和 release 语句，控制故障数据的持续时间，从而完成硬错误的模拟，本文设计的硬错误注入平台如图 3-19 所示。

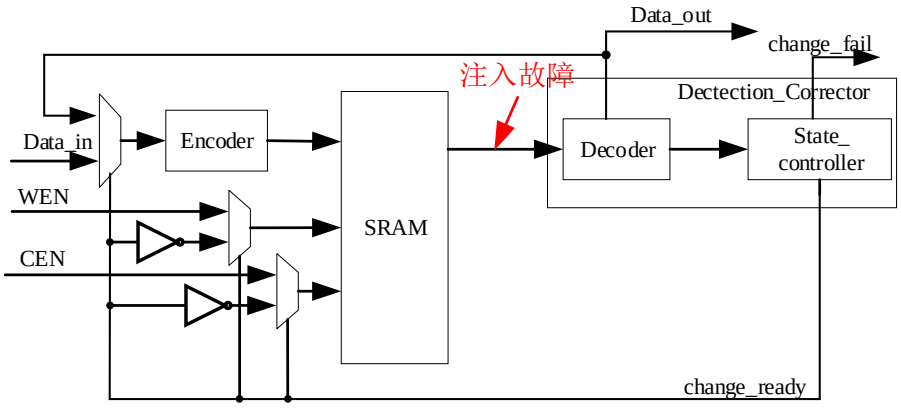


图 3-19 硬错误注入

如图 3-19，箭头表示的是故障注入点，即通过 force 语句对存储器输出数据进行更改，根据 3.2.3 小节中的时序图可知，硬错误需要维持 3 个周期的时间。例如设时钟周期时间为 8ns，当某地址数据开始输出的时间点开始，利用 force 语句先对存储器的输出数据 data_sram_out 进行强制变换（持续时间 24ns）。并在译码器第二次检测到输出数据 data_sram_out 有错误之后，利用 release 语句取消强制变化，从而完成硬错误的模拟，该仿真如图 3-20 所示。

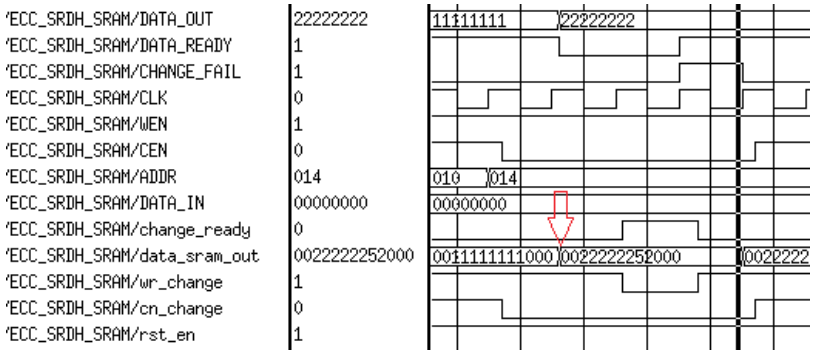


图 3-20 硬错误探测仿真

图 3-20 中箭头表示对存储器输出数据注入故障的起始时间点，并可以看出，该故障持续了 3 个周期，即 24ns 的时间。在上述操作下，译码器在第二次检测仍然收到了错误数据，译码器探测到不可修复的硬错误，硬错误信号 CHANGE_FAIL 置 1，表明该地址有不可修复的硬错误，需要采取相关措施。综上论文可知，本设计可以正确探测出存储器单元内发生的硬错误，并输出相关信号。

3.4 本章小结

本章采用第二章低冗余矩阵码编码器译码器硬件电路对 SMIC65nm 工艺的 Memory-Compier 工具生成的存储器进行了加固设计，并在此基础上添加了自检测错误类型的状态控制器，从而实现了修复软错误的同时探测硬错误的 SRAM 存储器整体加固设计结构。本章还构建了故障注入平台，并在 ModelSim 环境下，通过模拟注入软错误验证了所设计的 SRAM 存储器结构能够正确的修复存储单元内的软错误。硬错误模拟注入仿真也显示该设计可以探测存储单元内的硬错误，即本设计实现了对错误类型的分类以及对软错误的修复功能。

第4章 软硬错误自修复的 SRAM 设计与系统级验证

在前面研究成果的基础上，本章将添加修复硬错误功能，从而完成对硬错误的修复。由于软硬错误的修复都需要相应的时钟周期，为了满足系统时序的要求，本章将基于软硬错误修复的 SRAM 结构设计 AHB 接口，使用总线功能模型对具有软硬错误修复功能的 SRAM 进行系统级验证，从而证实该设计可以在系统中正常工作。

4.1 软硬错误自修复设计

由于硬错误无法通过写回操作来修正错误，因此在极端环境下，极易出现错误的累计，一旦一个字内发生错误的位数超过了纠错码电路本身的纠正能力，必将导致数据在读出时出现错误。本节将在硬错误探测机制的基础上，设计添加了硬错误修复模块，从而使得存储器能够在探测到硬错误时，采取相应的策略对其进行隔离，最终达到消除硬错误的目的。

由上述分析可知，硬错误具有不能被消除只能被隔离的特点，为在完成硬错误地址隔离的同时，保证存储单元的原有存储容量，需要在 SRAM 结构中预先设计添加适当的冗余字，此外，还需要设计故障地址分析器。SRAM 在硬错误修复模式下的工作方式如下：当 SRAM 存储结构探测到硬错误时，故障地址分析器存储当前硬错误地址，并将该地址映射到无故障的冗余地址上，之后利用第三章设计的状态控制器，控制存储器的读写，从而把正确的数据写到映射后的无故障冗余地址上，当下次再读到该硬错误地址的时候，故障地址分析器会把地址直接映射到相应的冗余地址，从而保证读出数据的正确性，最终达到修复硬错误的目的。

4.1.1 故障地址分析器

本节设计的故障地址分析器有以下六种功能：一是存储主存储器故障地址，即当译码器检测到硬错误时，能够存储当前硬错误地址；二是标记冗余地址，即对已使用的冗余地址进行标记，避免冗余地址的重复使用；三是通过地址比较判断当前地址是否是硬错误地址；四是映射故障地址到冗余地址；五是映射的冗余地址发生错误时，再次映射到新冗余地址；六是冗余地址全部使用完毕之后，再次探测到硬错误时，发出硬错误探测信号。

本节将通过设置冗余使用标志位来标记已使用的冗余地址；通过设置故障地

址存储器来存储硬错误地址；通过设置映射器来映射硬错误地址到相应的冗余地址。为此，本节设计了如图 4-1 所示的故障地址分析器。

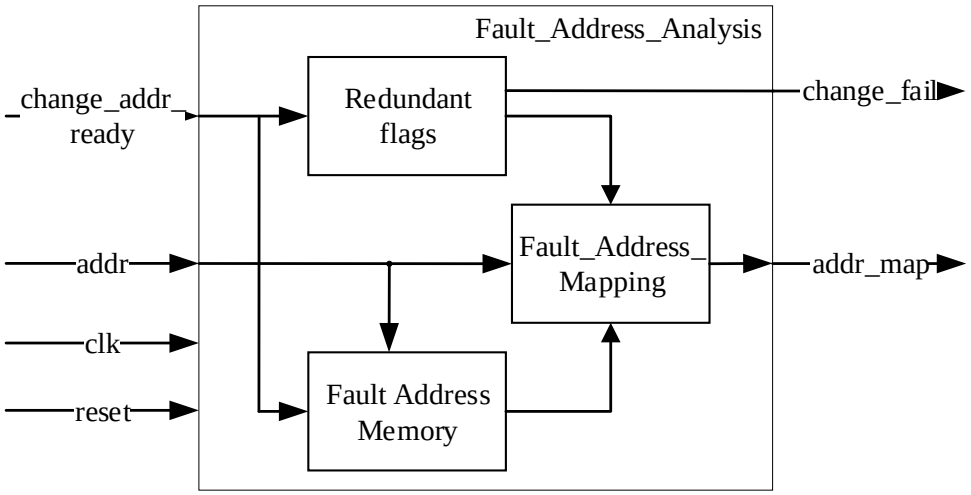


图 4-1 故障地址分析器结构

从图 4-1 中可以看出，故障地址分析器由冗余标志位、故障地址存储器和故障地址映射器构成。故障地址分析器输入输出信号的详细说明见表 4-1 所示。

表 4-1 故障地址分析器信号定义

信号名称	方向	说明
change_addr_ready	输入	隔离硬错误地址信号，高电平有效
addr	输入	输入地址
clk	输入	时钟信号
reset	输入	复位信号
change_fail	输入	硬错误检测信号，高电平有效
addr_map	输出	映射地址

故障地址分析器的具体工作方式如下：当译码器检测到硬错误时，隔离硬错误地址信号 change_addr_ready 被置为高电平，从而告知故障地址分析器需要隔离当前硬错误地址。故障地址存储器存储当前地址 addr，然后故障地址映射器会通过地址比较分析出当前地址 addr 为已存储的硬错误地址，从而把硬错误地址 addr 映射到对应的冗余地址 addr_map 并输出到存储器地址输入端，从而隔离掉了原地址。当下次地址端再次选到该硬错误地址时，由于已经存储了该硬错误地址，因此不需要再次进行存储，直接映射到对应的冗余地址当中。若映射的冗余地址输出数据有错误时，change_addr_ready 再次拉高，再次存储当前硬错误地址之后，映射器会把地址映射到新的冗余地址当中，从而隔离掉了冗余地址的硬错误地址。冗余地址标志位个数要与故障地址存储器的大小相同，也与添加的冗余地址的个

数相同。当冗余地址标志位全置 1 时，表示冗余存储器全部被使用完毕，这时再探测到硬错误时，已没有剩余的冗余地址去修复，硬错误检测信号 change_fail 变成高电平，从而告知系统探测到不可修复的硬错误。

有了故障地址分析器之后，地址比较次数会大大增高，为了减少比较次数，本文采用了地址分割与冗余使用标志位相结合的方法。如果添加的冗余地址较多，则故障地址存储器也会相应的增加，需要被比较的地址也会增加。利用地址分割的方式，把地址分割成为两个部分，即把故障地址存储器和冗余地址分割成为若干个小组合；这就相当于把主存储器分割成为若干个块，为每一个存储器块分配一个冗余地址小组。故障地址分析器进行地址比较时，先比较地址的前半部分，根据前半部分，映射到相应的小组合，然后跟选定小组合内的故障地址进行比较。冗余使用标志位表示该冗余使用与否，若该位为 1，则表示有故障地址已存储到相应的故障地址存储器当中。因此当地址进入到故障地址分析器时，先根据地址前半部分选定某个小组，然后只需要跟该组内冗余使用标志位为 1 的故障地址进行比较即可，例如假设地址为 5 位，则分割成为前两位和后三位，并存储后三位，具体方法如图 4-2 所示。

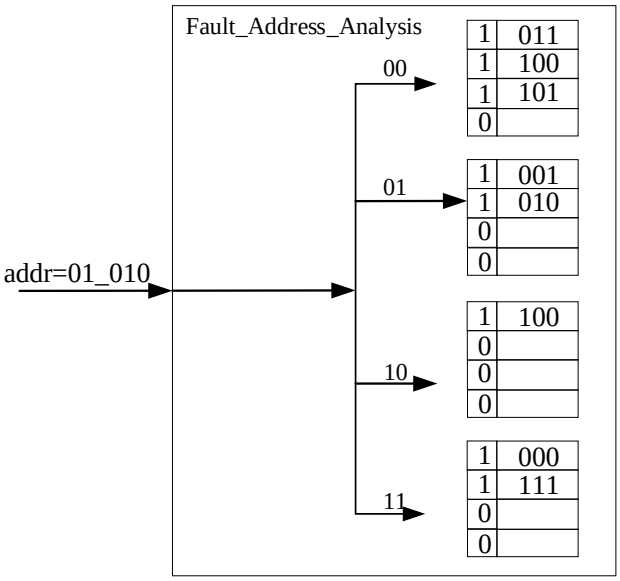


图 4-2 地址比较

在存储故障地址时，先把故障地址分割成为前后两个部分，图 4-2 中输入故障地址 01_010，分割成为 01 和 010，把故障地址存储器分割为四个小组，第一个小组代表的是前两位为 00 的小组，下一个是前两位为 01 的小组，依此类推。小组的个数与地址分割的前几位有关，例如分割为前两位，则有 4 个小组，分割为前四位，则有 16 个小组，每个小组的故障存储个数相同。

每个故障地址前面的标志位为冗余使用标志位，标志位为 1 表示该冗余已被使用。因地址 01_010 的前两位为 01，因此只会跟第二个小组的故障地址进行比较。且由于冗余使用标志位为 1 的地址当中才有故障地址，所以只跟 01 小组内的冗余使用标志位为 1 的故障地址进行比较，根据顺序只比较了 2 次，大大减少了比较次数。这就相当于把存储器分割成 4 个块，当每个存储器块内的故障数小于每个故障地址小组内的可存储故障地址个数时，该机制都可以进行存储。

经过地址比较之后，根据选中硬错误地址与否，输出相应的地址。例如由图 4-2 可知，地址为 5 位，则主存储器大小为 32，故障地址存储器中，可存储地址有 16 个，则整个存储器大小为 48 个。故障地址存储器会把相应比较信息传送给映射器，使其能够输出相应的映射地址。若没有相同的地址，则正常地址前添加 0 输出，即输出 0_01_010，从而选定了原地址。若通过地址比较选中了某个硬错误地址，其映射的冗余地址就是故障地址存储的位置，例如选中了 01010 地址，与第 2 个小组的第 2 个地址相匹配，则映射器会把当前地址映射到冗余地址的第 6 个地址，即输出的映射地址为 1_001_01，第 1 位为 1 表示该地址为冗余地址，中间三位表示选定的小组，后面 2 位表示小组中的位置。

由于冗余地址当中也有可能有错误，前面也提到当冗余地址检测到错误时，需要映射到新的冗余地址。为了解决这一问题，假设冗余地址 1_001_01 有错误，该地址原始地址为 01_010，就需要把该地址再次存储到故障地址存储器当中，从而让它映射到新的冗余地址当中，如图 4-3 所示。

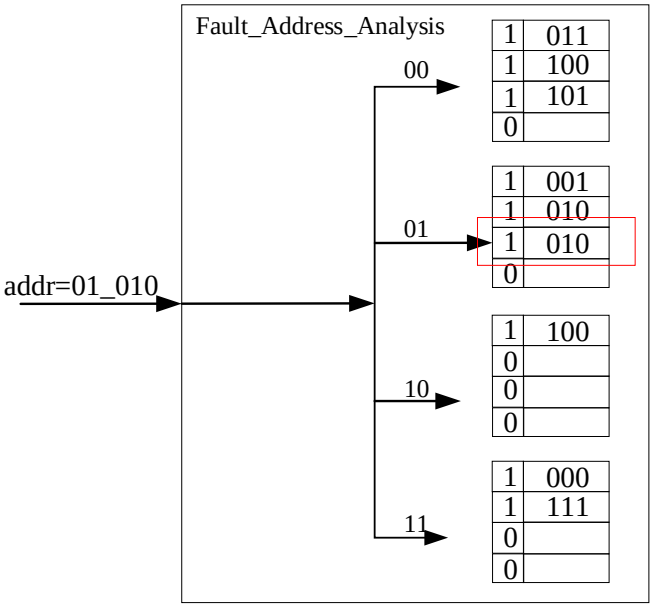


图 4-3 映射新冗余地址

根据图 4-3 所示，由于映射的冗余地址 1_001_01 有错误，则再次把硬错误地址 01_010 存储到故障存储器当中，冗余标志位置 1。让映射地址由原来的 1_001_01 变成了 1_001_10，等下一次再次选到该地址，直接映射到新的冗余地址，从而既隔离掉了有硬错误的原地址，又隔离掉了有硬错误的冗余地址。

4.1.2 改进的状态控制器

本节将对第三章提出的状态控制器进行改进，使其与故障地址分析器相协调，从而完成对硬错误的修复。当译码器第二次检测到错误时，即表明检测到的错误为硬错误，改进的状态控制器的目的是使其能够控制故障地址分析器，进而建立该硬错误地址与冗余地址的映射关系，最终通过写回操作，实现数据的正确存储，具体状态控制流程如图 4-4 所示。

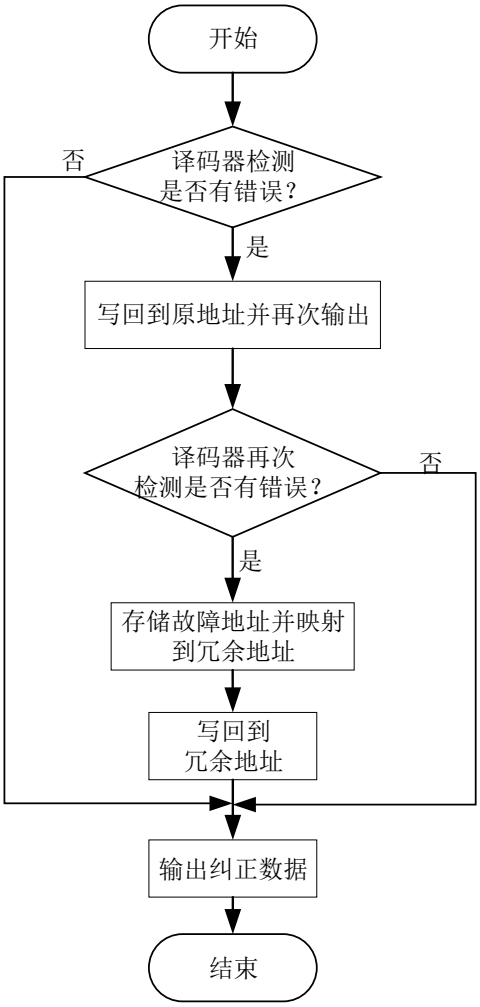


图 4-4 改进的状态控制流程

由图 4-4 可知当译码器检测到硬错误时，故障地址将被故障地址分析器记录并映射到冗余地址，之后利用写回操作，将正确数据写入到冗余地址并输出。基于图 4-4，本节改进了相关的有限状态机，改进后的状态机共设置 6 个状态，其详细的状态转变如图 4-5 所示。

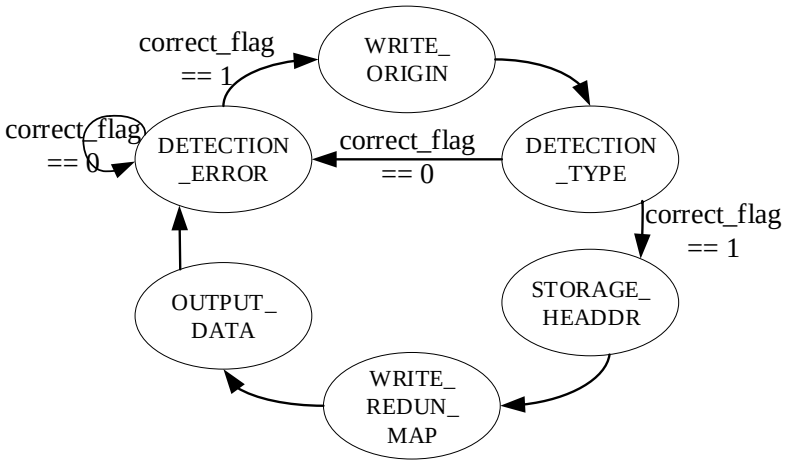


图 4-5 改进的状态转变

由图 4-5 中可知，改进后整个状态的变化主要还是依据原来低冗余矩阵码译码器所发出的纠正标志信号 correct_flag，具体转换过程如下：

（1）在第二次检测状态 DETECTION_TYPE 判断 correct_flag 是否为 0，若纠正标志信号 correct_flag 等于 0，则表示探测到的错误为软错误，状态机回到初始状态 DETECTION_ERROR，从而完成对软错误的修复；

（2）在 DETECTION_TYPE 状态下 correct_flag 为 1 时，表明当前地址为硬错误地址，change_addr_ready 置 1，从而告知故障地址分析器，需要隔离硬错误地址，状态机跳转到 STORAGE_HEADDR 状态；

（3）故障地址分析器存入当前硬错误地址之后，状态控制器的多路选择器控制信号 change_ready 变成 1，状态机跳到 WRITE_REDUN_MAP 状态；

（4）change_ready 控制存储器进行写操作，多路选择器会把纠正数据送到存储器的数据输入端，故障地址分析器会把映射的冗余地址送到存储器的地址输入端，从而把正确数据写到冗余地址当中，写入完毕之后，change_ready 置 0，状态跳到 OUTPUT_DATA；

（5）数据准备信号 data_ready 被置为高电平，同时冗余地址当中的数据被输出，从而完成对硬错误的修复，状态机回到原状态，修复工作结束。

根据上述工作流程，改进的状态控制器的输入输出端口也需要做出相应的变化，改进的状态控制器输入输出说明如表 4-2 所示。

表 4-2 状态控制器信号定义

信号名称	方向	说明
CLK	输入	时钟信号
rst_en	输入	复位信号
CEN	输入	外部存储器使能信号，低电平有效
WEN	输入	外部存储器写实能信号，低电平有效
correct_flag	输入	纠正标志信号，高电平表示数据被纠正
change_addr_ready	输出	隔离硬错误地址信号，高电平有效
change_ready	输出	多路选择器控制信号，高电平有效
data_ready	输出	数据准备信号，高电平表示数据有效

对比表 4-2 和表 3-3 可知，改进后的状态控制器添加了 change_addr_ready 输出信号，该信号将控制故障地址分析器存储硬错误地址。原探测硬错误信号 change_fail 转交给故障地址分析器控制，即当没有剩余冗余地址时，若再次探测到硬错误地址，change_fail 置 1，表示有不可修复的硬错误，也表明当前小组的冗余地址已被全部使用。

4.1.3 软硬错误自修复的 SRAM 整体结构

至此，本文已设计实现了 ECC 编码器、译码器、故障地址分析器、改进的状态控制器。Memory-Compiler 工具生成添加 16 个冗余地址的 52 位 x2064 存储器模型，与上述模块进行整合，同时协调各个模块的输入输出关系，本节实现了基于 ECC 电路的软硬错误自检测修复 SRAM 的整体结构设计，如图 4-6 所示。

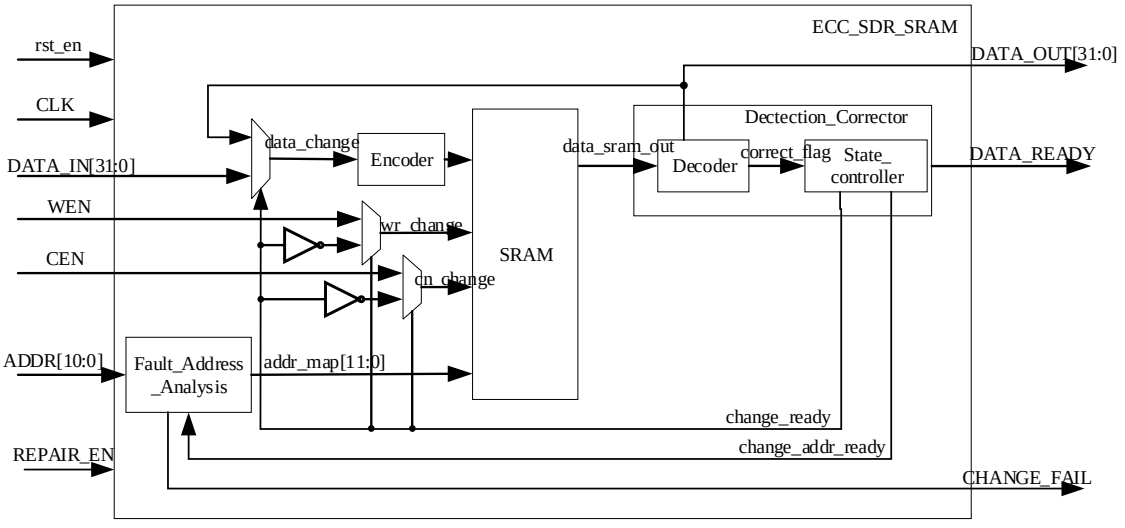


图 4-6 软硬错误自修复的 SRAM 结构

与图 3-7 对比可知，图 4-6 中的软硬错误自修复的 SRAM 结构添加了故障地

址分析器模块，该模块主要负责地址的分析与映射，从而隔离硬错误地址，并与状态控制器配合，完成对硬错误的修复，软硬错误自修复的 SRAM 结构的外部输入输出信号定义如表 4-3 所示。

表 4-3 软硬错误自修复的 SRAM 结构信号定义

信号名称	方向	说明
CLK	输入	时钟信号
rst_en	输入	复位信号
CEN	输入	外部存储器使能信号，低电平有效
WEN	输入	外部存储器写实能信号，低电平有效
ADDR	输入	外部地址输入（11 位）
DATA_IN	输入	外部数据输入（32 位）
REPAIR_EN	输入	检测修复使能信号，高电平有效
DATA_OUT	输出	数据输出（32 位）
CHANGE_FAIL	输出	不可修复硬错误探测标志信号，高电平有效
DATA_READY	输出	数据准备信号，高电平表示数据有效

由图 4-12 和表 4-3 可知，软硬错误自修复的 SRAM 结构添加了修复使能信号 REPAIR_EN，从而使系统可以通过该信号控制该存储器的检测修复功能。检测修复软错误和硬错误需要一定的时钟周期，因此在高速读写时，需要关闭检测修复功能，从而满足单周期输出。事实上，无论是开启还是关闭检测修复使能信号 REPAIR_EN，存储器的写时序都相同；当开启检测修复功能时，不同的错误类型会对读时序产生不同的影响，其影响如图 4-7 和图 4-8 所示。

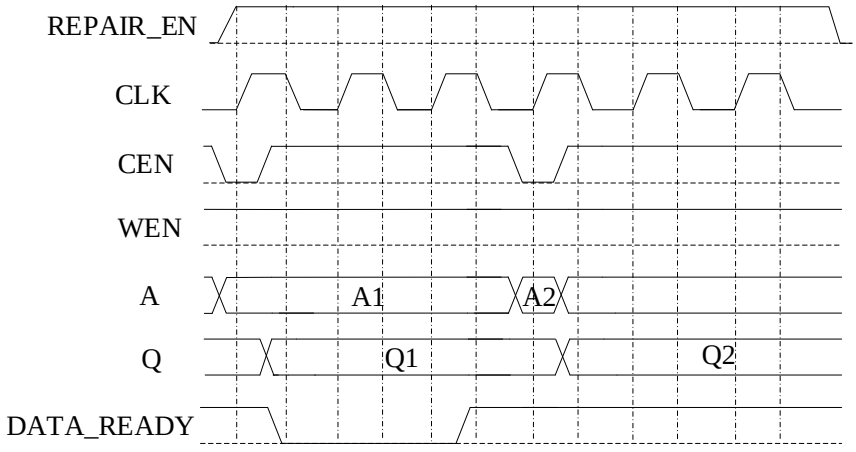


图 4-7 修复软错误时序（开启修复使能 REPAIR_EN）

从图 4-13 中可以看出，检测修复功能为开启状态。译码器第一次检测到错误时，DATA_READY 变为 0，存储器单元内软错误被修复并输出，DATA_READY 保持低电平两个周期之后变为高电平，表明软错误被修复。在进行下一个地址的

操作时，由于没有发现错误，即使开启了修复使能，也不会启动修复功能。

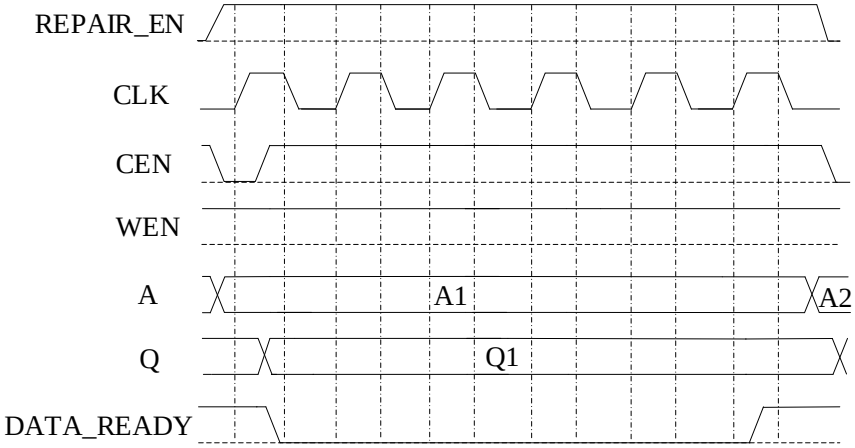


图 4-8 修复硬错误时序（开启修复使能 REPAIR_EN）

根据图 4-8 所示，当第二次检测发现该错误类型为硬错误，则存储当前故障地址，并映射到冗余地址之后，正确数据被写到该冗余地址并进行输出。从图 4-8 中可以看出，DATA_READY 保持五个周期的低电平之后，变回高电平，表明硬错误的修复工作完成，当前数据有效，并开始对下一个地址进行操作。

在读模式下，ECC 电路的作用是检测读出数据是否有错误，并纠正错误。状态控制器的作用是分辨错误类型、控制存储器的读写、控制故障地址分析器。故障地址分析器作用是隔离故障地址。表 4-4 为各情况下存储器发生错误时详细状态说明，可见错误情况有以下五种形式。

表 4-4 错误情况及其修正方法

情况	工作模式	故障原因	修正方法
1	读/写	地址为已存储故障地址	故障地址分析器把地址映射到冗余地址；
2	读	正常地址发生软错误	1.ECC 电路检测纠正； 2.纠正数据被写回到原地址；
3	读	冗余地址发生软错误	1.故障地址分析器映射；ECC 电路检测纠正； 2.纠正数据被写回到原冗余地址；
4	读	正常地址发生硬错误	1.ECC 电路检测纠正；写回到原地址； 2.状态控制器判断当前错误为硬错误； 3.故障地址分析器存储故障地址并映射； 4.状态控制器控制数据写到冗余地址；
5	读	冗余地址发生硬错误	1.故障地址分析器映射，ECC 电路检测纠正； 2.纠正数据被写回到原地址； 3.状态控制器判断当前错误为硬错误； 4.故障地址分析器存储故障地址并再映射； 5.状态控制器控制数据写到新冗余地址；

第一种情况，当前地址为硬错误地址时，故障地址分析器内已经存储了该硬错误地址，因此故障地址分析器直接把地址映射到相应的冗余地址。因此存储器进行写操作时，输入数据经编码器编码之后存入到该冗余地址当中；存储器进行读操作时，该冗余地址的数据经译码器译码之后输出。

第二种情况与第三章中所提到的修正方法相同，此处不再详细说明。

第三种情况，故障地址分析器把故障地址映射到冗余地址，ECC 电路检测到该冗余地址有软错误，状态控制器控制数据写到该冗余地址。

第四种情况，存储器输出数据经 ECC 电路的二次检测发现当前地址发生硬错误，状态控制器告知故障地址分析器需要存储当前硬错误地址；故障地址分析器存储当前地址，并把该硬错误地址映射到冗余地址；最后，状态控制器控制正确数据写回到映射的冗余地址当中。

第五种情况，故障地址分析器已把当前地址映射到冗余地址；该冗余地址的数据经 ECC 电路的二次检测发现当前冗余地址有硬错误，状态控制器告知故障地址分析器需要再次存储当前硬错误地址；故障地址分析器存储当前地址之后，再次映射到新冗余地址，状态控制器控制正确数据写回到映射的新冗余地址当中。

4.1.4 综合与门级仿真

本文采用 Memory-Compiler 生成 SMIC65nm 的数据宽度为 52 位、大小为 2064、频率为 125MHz，开启了 write-though 功能的单端口 SRAM 存储器，生成.lib 文件。采用中芯国际的 SMIC65nm 工艺，在 linux 环境下，利用 synopsys 公司的 Design Complier 工具，根据综合脚本对设计的电路进行综合，把存储器的.lib 文件转换成.db 文件之后，关联到综合时要使用的库上，综合时钟约束设置为 8ns，即时钟频率设为 125MHz，其综合报告以及面积报告如图 4-9 所示。

```
   Cell Internal Power  =   1.6929 mW   (96%)
   Net Switching Power  =  67.7818 uW    (4%)
                          ---------
Total Dynamic Power     =   1.7607 mW  (100%)

Cell Leakage Power      =   3.0895 uW
```

```
Combinational area:              2593.440014
Buf/Inv area:                     244.800009
Noncombinational area:           1728.000069
Macro/Black Box area:           74211.617188
Net Interconnect area:              0.000000

Total cell area:                78533.057271
Total area:                     78533.057271
```

a)功耗报告　　b) 面积报告

图 4-9 综合报告

根据图 4-15 中的功耗报告显示，本文设计的软硬错误自修复 SRAM 存储器的 Total Dynamic Power 为 1.6929mW，Cell Leakage Power 为 3.0895μW，面积报告显示，存储器面积约占全部面积的 94.50%。完成逻辑综合，提取综合后电路网表文

件.v 及延迟文件.sdf。在 ModelSim 平台下，导入利用 Memory-Compiler 生成存储器的.v 文件，SMIC65nm 工艺下的标准单元的.v 文件，综合后产生的网表文件，在仿真时，载入延迟文件.sdf 之后，进行综合后仿真，模拟注入软错误和硬错误，分别对各种情形进行了仿真，详细的波形如图 4-10 到图 4-15 所示。

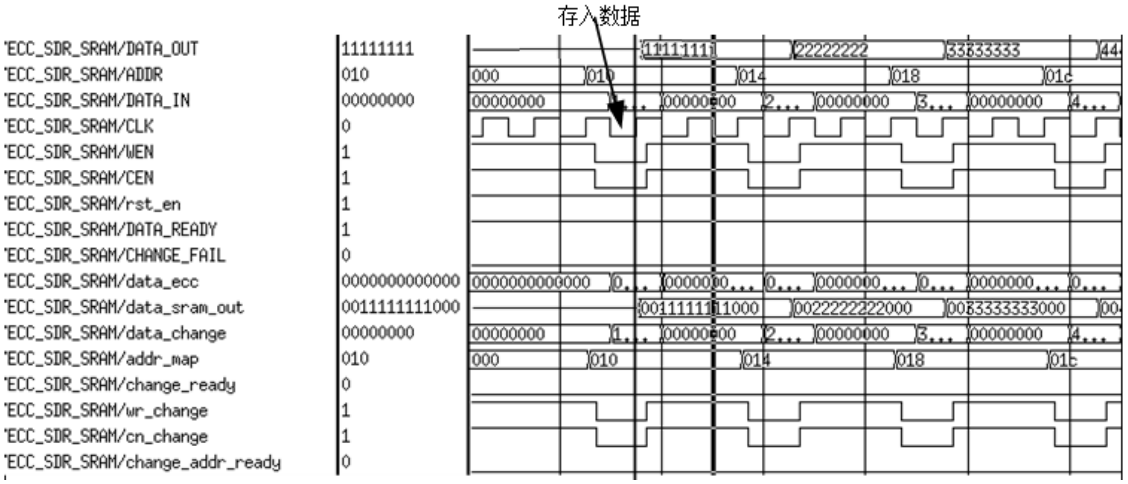


图 4-10 写操作

图 4-10 为写操作仿真结果，CLK 时钟上升沿来临时，WEN = 0，CEN = 0，表示当前为写操作状态。由于写操作时，译码器检测功能不会启动，多路选择器控制信号 change_ready 和硬错误隔离信号 change_addr_ready 一直保持 0，只会把当前数据 DATA_IN 经过编码器编码得到 data_ecc，之后把该数据输送到存储器的数据输入端。由于当前地址不是已存储的故障地址，ADDR 直接输入到存储器的地址输入端 addr_map，并把数据 data_ecc 存入到该地址当中，写操作完成。

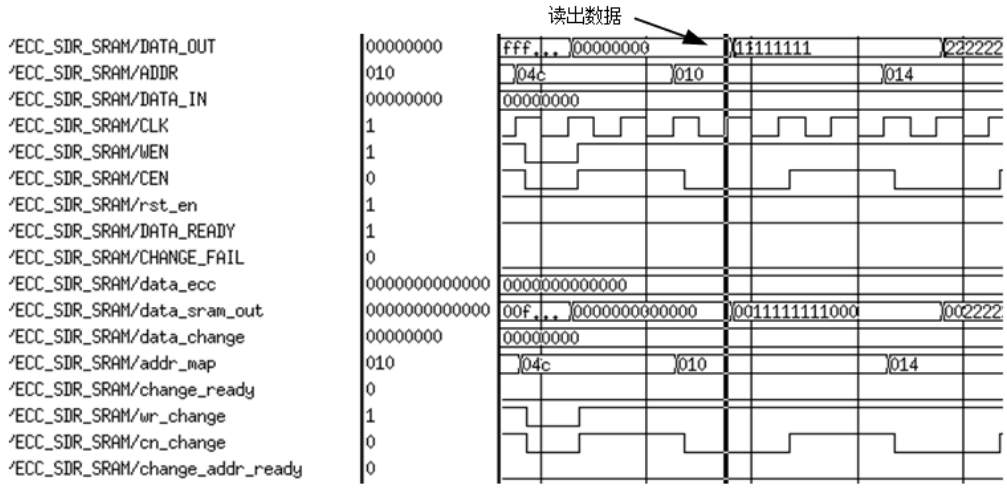


图 4-11 读操作（无错误）

图 4-11 为正常的读操作仿真结果，时钟上升沿来临时，WEN = 1，CEN = 0，

表示当前为读操作状态，由于译码器检测当前数据没有错误，因此状态控制器不会被启动。change_ready 和硬错误隔离信号 change_addr_ready 一直保持 0。因为当前地址不是已存储的故障地址，只会把当前地址 ADDR 中的数据经过译码器译码后，直接输出，在此过程中数据准备信号 DATA_READY 一直为 1，完成读操作。

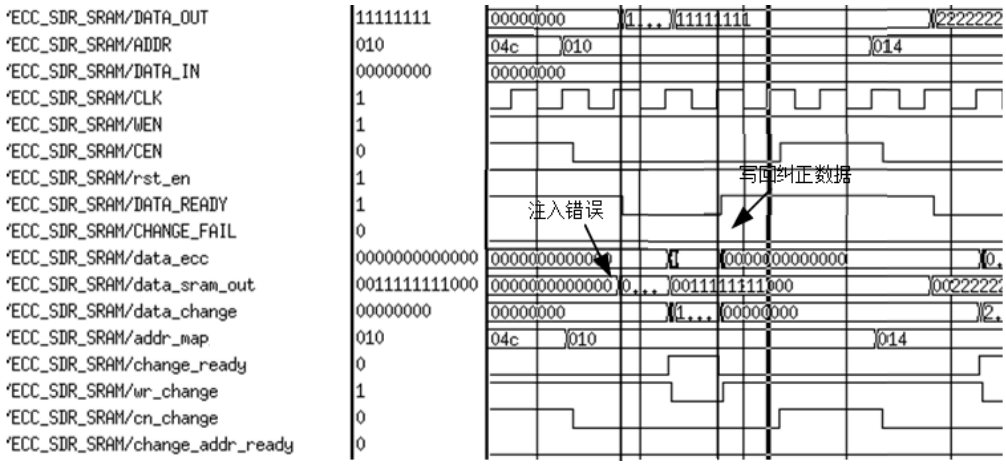


图 4-12 软错误修复

图 4-12 为软错误修复仿真结果，当外部选定 11’h010 地址时，对存储器输出数据 data_sram_out 注入错误，数据准备信号 DATA_READY 变成 0。一个周期之后，多路选择器控制信号 change_ready 变成 1，即存储器写使能有效，存储器输入数据 data_ecc 变成当前已被 ECC 电路纠正的数据，该数据通过写回机制重新写入到当前 12’h010 地址当中之后，change_ready 信号变成 0。当数据第二次输出时，译码器未检测到错误，表明存储器的软错误数据已被修复，数据准备信号 DATA_READY 变成 1，表明该数据有效，存储器开始等待下一个地址的操作。经上述分析易知，软错误的修复总共需要 2 个周期的延迟。

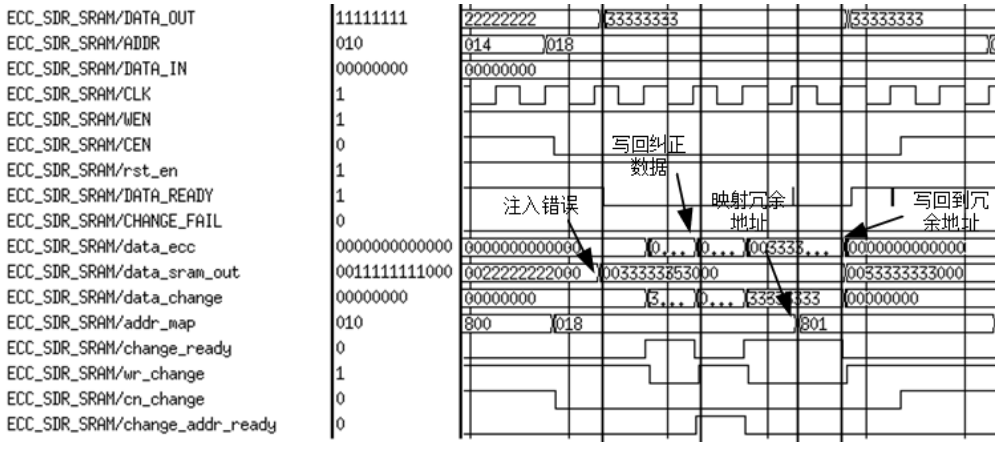


图 4-13 硬错误修复

图 4-13 为硬错误修复仿真结果，当外部读 11’h018 地址的数据时，对存储器输出数据 data_sram_out 上注入错误并持续 5 个周期。可以看到，注入错误之后，数据准备信号 DATA_READY 变成了 0，写回纠正数据后，再次输出的数据经译码器译码后发现错误仍然存在，这时硬错误隔离信号 change_addr_ready 变成 1，表示需要存储故障地址，当存储故障地址 11’h018 之后，change_addr_ready 变回 0。当前地址 addr_map 将在一个周期之后由 12’h018 变成映射冗余地址 12’h801，此时多路选择器控制信号 change_ready 为 1，即存储器将对映射的冗余地址 12’h801 写入纠正数据，分析图中时序可知，硬错误的修复需要 5 个周期的延迟时间

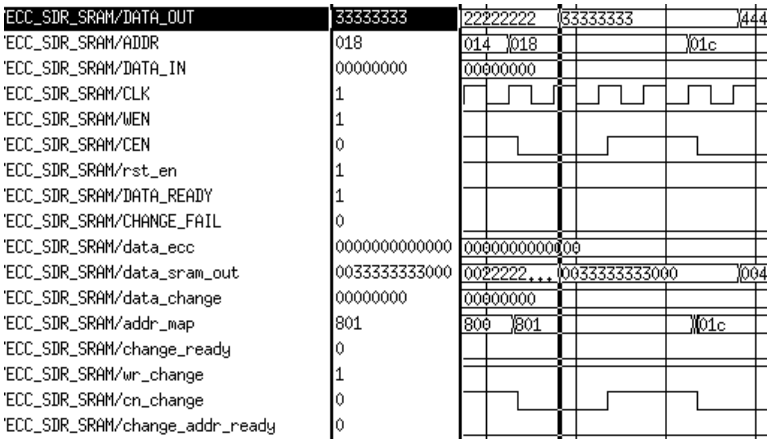


图 4-14 地址选到硬错误地址

图 4-14 为地址选到硬错误地址时的仿真结果，当存储器的地址 ADDR 选到已在图 4-13 中存储的故障地址 11’h018 时，映射地址 addr_map 直接把地址 11’h018 映射成冗余地址 12’h801 发送到存储器的地址输入端。存储器直接读取冗余地址 12’h801 中的数据进行输出，从而隔离掉了原地址当中的错误，并且没有时钟周期的延迟。

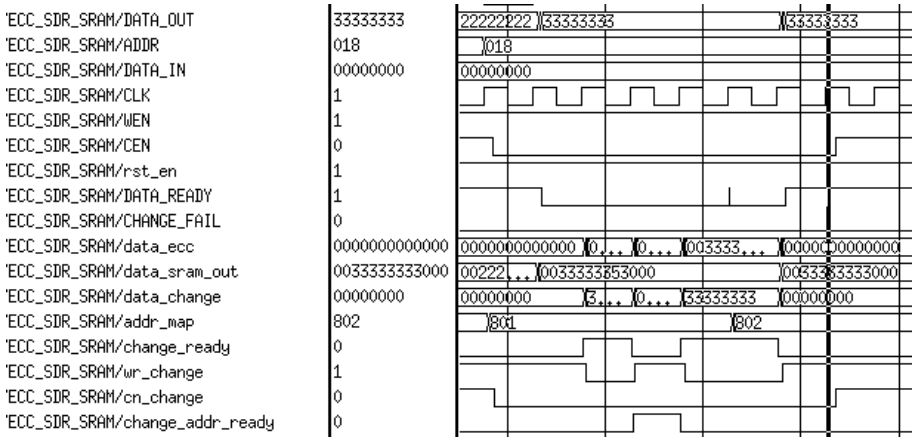


图 4-15 冗余地址硬错误修复

图 4-15 中为冗余地址硬错误修复仿真结果，原来的错误地址 11’h018 被故障地址分析器映射到冗余地址 12’h801，存储器输出的数据 data_sram_out 经译码器译码又检测到有硬错误，因此需要隔离掉已映射的冗余地址，硬错误隔离信号 change_addr_ready 变成 1。故障地址分析器把地址 11’h018 再次存储到故障地址存储器当中，然后旧冗余地址 12’h801 映射到新冗余地址 12’h802，把纠正数据写到该地址当中，之后存储器输出正确数据。从而完成对冗余地址硬错误的隔离，该修复措施与正常的硬错误修复一样，都需要 5 个周期的延迟，当下次再选到地址 11h’018 时，直接映射到新冗余地址 12’h802。

4.2 AHB 总线接口设计与系统级验证

存储器的使用在实际的系统应用中，应当要挂载到总线去运用，这样 SRAM 存储器可以与其他的模块通过总线进行通信。为此，本节基于 AHB 总线协议，为本文所提出的 SRAM 存储器设计了 AHB 接口，并使用总线功能模型（Bus Function Model，BFM）对连接了 AHB 接口的 SRAM 存储器结构进行仿真，从而验证文本设计可以集成到系统中去应用。

4.2.1 AHB 总线接口设计

AHB 总线是 ARM 公司提出的 AMBA2.0 规范当中新一代的满足高性能传输的总线结构，其采用了地址与数据分离的格式，可支持固定长、不定长猝发交易、分裂交易特性，且支持多个主设备的总线管理，总线规范中的 AHB 从设备的接口如图 4-16 所示[44]。

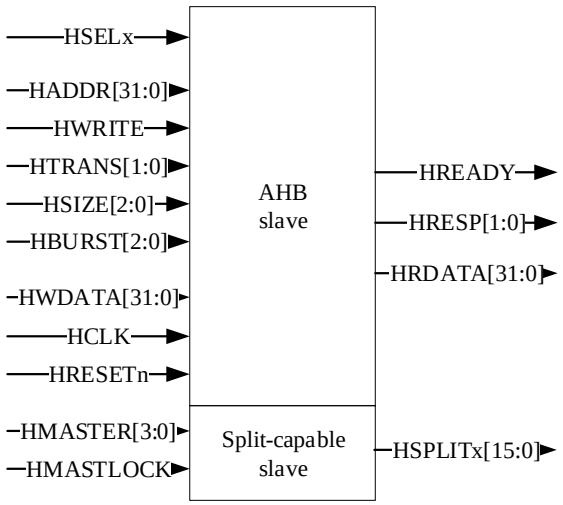


图 4-16 AHB 从设备接口

为了使本文设计的 SRAM 结构能够挂载到总线去应用，根据图 4-22 中，AHB 从设备的接口标准，设计连接 AHB 总线与 SRAM 结构的 AHB 接口，AHB 接口的详细设计及输入输出如图 4-17 所示。

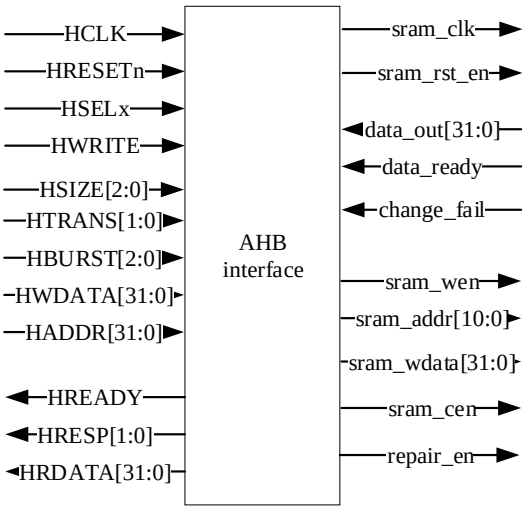


图 4-17 AHB 接口

如图 4-17 中 AHB 接口的设计，其左端为连接到 AHB 总线的 AHB 从设备的标准端口，该端口能够挂载到 AHB 总线进行数据通信；右端为连接到本文设计的 SRAM 存储器结构，通过 AHB 接口实现 AHB 总线与 SRAM 存储器结构的通信。存储器实际应用的大小为 2K，因此为存储器分配的地址范围是 0x000 到 0x7ff。在此基础上添加了 2 个特殊功能的寄存器，用于控制检测修复使能 repair_en，接收硬错误探测信号 change_fail，该寄存器的详细特征如表 4-5 所示。

表 4-5 特殊寄存器特征

寄存器名	分配地址	读写权限
REPAIR_EN_REG	0x800	可读可写
HARD_REG	0x804	可读可写

表 4-5 中，REPAIR_EN_REG 是控制存储器检测修复使能 repair_en 的寄存器，可通过写操作对该地址输入数据，来控制该存储器是否在进行读操作时，探测修复存储器内的软错误和硬错误，也可通过读该寄存器的数值来判断是否开启了探测修复功能。

HARD_REG 是表述硬错误探测个数的寄存器，接收 change_fail 信号，每接收到一次 change_fail 的高电平，该寄存器就利用计数器记录存储器内发生的不可修复的硬错误个数。根据 4.1.1 节，故障地址分析器比较地址时，采用地址分割的方式，因此把存储器分割成 4 个块，分别是（0x000，0x1ff），（0x200，0x3ff），（0x400，

0x5ff），（0x600，0x7ff）。因此 32 位的 HARD_REG 寄存器也分割成 4 个部分 HARD_REG1，HARD_REG2，HARD_REG3 和 HARD_REG4。硬错误探测个数寄存器与主存储器地址的对应关系如表 4-6 所示。

表 4-6 硬错误探测寄存器与主存储器地址对应关系

硬错误探测个数寄存器	主存储器地址
HARD_REG1[7:0]	0x000~0x1ff
HARD_REG2[7:0]	0x200~0x3ff
HARD_REG3[7:0]	0x400~0x5ff
HARD_REG4[7:0]	0x600~0x7ff

根据表 4-6，HARD_REG1[7:0]寄存器只记录 0x000~0x1ff 范围内发生的硬错误个数，HARD_REG2[7:0]寄存器记录 0x200~0x3ff 范围的硬错误，依次类推。每个部分寄存器分别表示在相应存储器地址内探测到的硬错误个数。例如 HARD_REG 寄存器的值为 32'h00000001，表示以下三种信息，第一：地址 0x000 到 0x5fff 未探测到不可修复的硬错误；第二：地址 0x600 到 0x7ff 中探测到了一个硬错误；第三：0x600 到 0x7ff 地址对应的冗余地址已使用完毕。

4.2.2 系统级验证

与本文 SRAM 结构相搭配的 AHB 接口设计完毕之后，基于 AMBA 总线的 VIP 模型，构建 AMBA 总线的系统级验证环境。AHB 接口与本文设计相结合，与总线功能模型 BFM 相连接，构建系统级的故障注入平台，详细结构如图 4-18 所示。

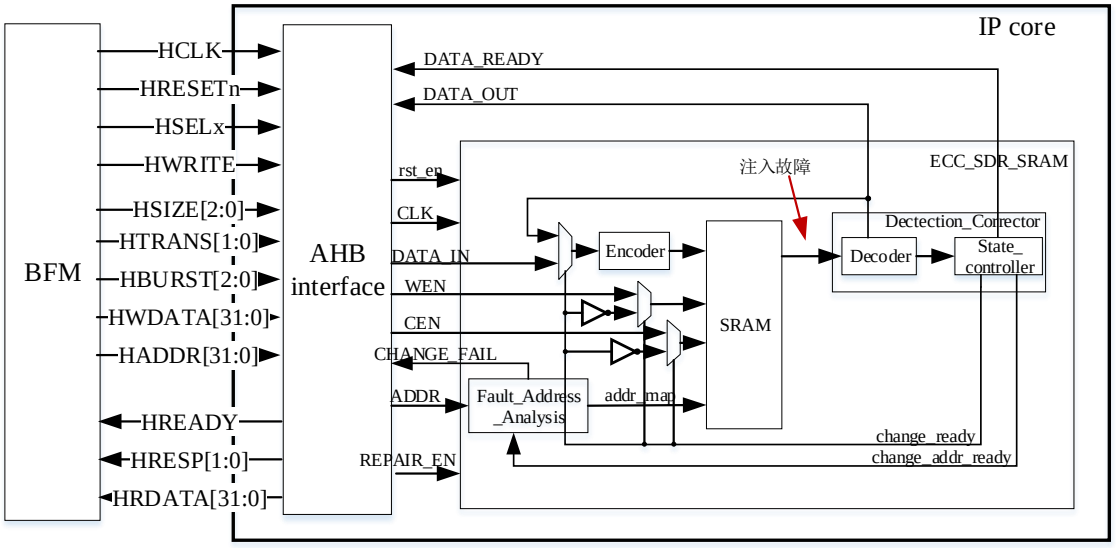


图 4-18 系统级验证环境

由图 4-18 可知，箭头表示的是注入故障点。在 ModelSim 平台上，基于 BFM 对 AHB 接口与 SRAM 结构相结合的 IP 核进行操作。在进行读操作时，采用 force 语句和 release 语句对存储器的输出端注入故障，分别查看波形，从而验证该设计是否正确修复存储器的软错误和硬错误。首先利用写操作对地址 0x800 写数据 32'h00000001，从而开启存储器的检测修复功能，然后对地址 0x000 到 0x014 分别写入数据 32'h10101010 到 32'hf0f0f0f0，写操作的仿真波形如图 4-19 所示。

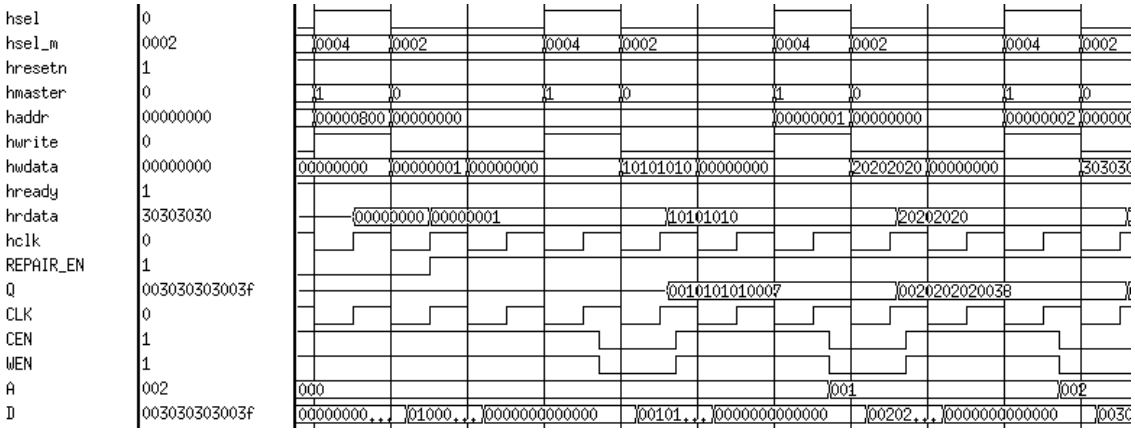


图 4-19 写操作

图 4-19 的仿真结果表明，每当片选使能 hsel 为 1、hwdata 为 1 时，存储器进行写操作，当地址 haddr 选到 0x800 时，输入数据 32'h00000001，可以看到 REPAIR_EN 置为 1，表示开启了存储器修复功能。这时不会对存储器进行操作，可以看出存储器的使能一直保持无效状态。然后对每个地址存入了相应的数据，写操作功能正确。下一个验证情形是对每个地址进行读操作，详细波形如图 4-20 所示。

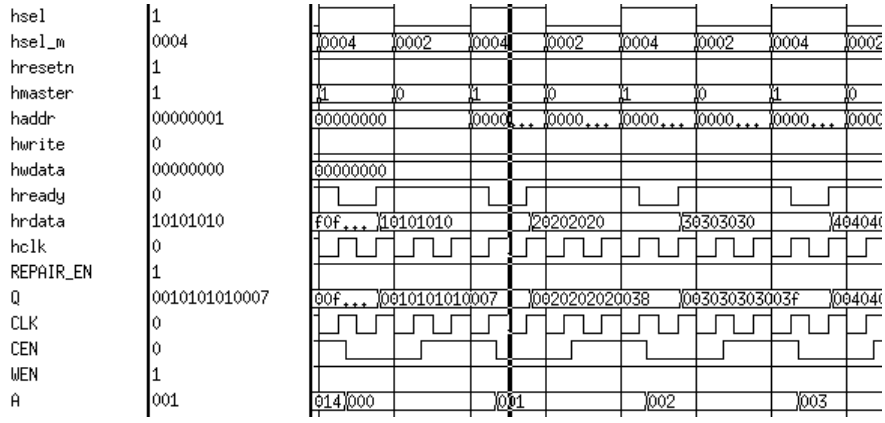


图 4-20 读操作

由图 4-20 可知，hwrite 一直保持 0，即表示存储器在执行读操作。由于存储器

本身需要一个周期的时间进行数据输出，因此每次选定某个地址时，hready 都会保持一个周期的低电平，从而完成存储器的读操作。对存储器进行读操作时，对存储器输出端注入软错误的波形如图 4-21 所示。

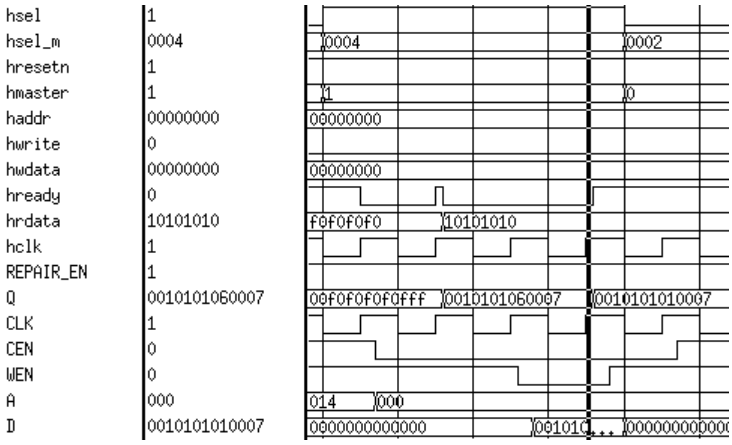


图 4-21 系统级验证软错误修复

对地址 0x000 注入软错误后，存储器输出数据 Q 变成 52’h0010101060007，SRAM 结构的软错误修复机制被启动，数据经过纠正之后被存储到原地址 0x000，存储器输出变回正确数据 52’h0010101010007，该数据通过译码器译码之后正确输出。总线会根据与 data_ready 相连的 hready 信号等待数据的修复，此操作过程比正常读操作多了两个周期。下面是硬错误的修复仿真结果，波形如图 4-22 所示。

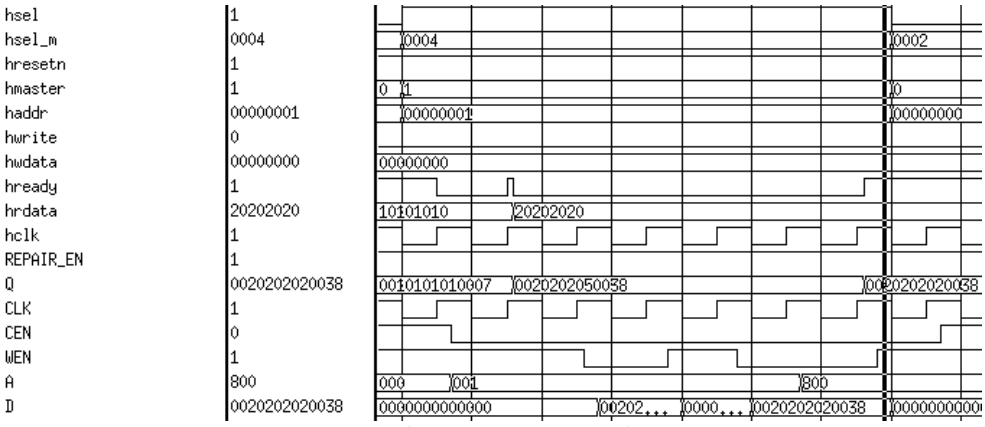


图 4-22 系统级验证硬错误修复

从图 4-22 中可以看出，读地址 0x001 时，注入了错误数据 52’h0020202050038，并在存储器地址 A 为 0x001 时，一直保持该错误数据。存储器地址 A 由 0x001 变成了冗余地址 0x800，并把正确数据 52’h00202020200038 写入该冗余地址，从而使存储器 Q 端输出正确的数据，最终完成硬错误的修复。这一操作过程比正常读操作增加 5 个时钟周期。

下一步为了验证冗余地址使用完毕时，该结构是否可以探测到硬错误，先对地址为（0x000，0x1ff）范围的数据注入 6 个硬错误，然后地址选到 0x804，查看硬错误个数寄存器 HARD_REG，相关波形如图 4-23 所示。

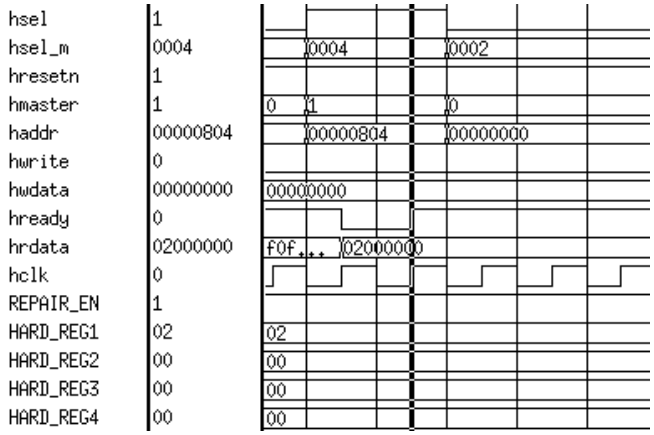


图 4-23 读硬错误个数

根据图 4-23 中当地址选到 0x804 时，读出的数据为 32’h02000000。HARD_REG1 为 8’h02，表明地址 0x000 到 0x1ff 对应的冗余地址已使用完，且又探测了 2 个硬错误；其他三个寄存器都为 0，表明其他地址没有探测到不可修复的硬错误。

至此，本文完成了所有软硬错误检测修复的系统级验证，通过系统级验证可以推断，开启修复使能 REPAIR_EN 之后，只有在发现存储器的错误时，才会启动修复机制；没有错误时，与正常的读写操作相同。此外，在修复硬错误之后，只要映射的冗余地址没有发生错误，再选到该地址进行读操作时，该 SRAM 存储器结构不会启动修复机制。

4.3 本章小结

本章在上一章设计的可探测硬错误的 SRAM 结构基础上，设计实现了可以判断硬错误地址的故障地址分析器，改进了状态控制器，并添加了适当的冗余存储器，从而完成了软硬错误自修复的 SRAM 结构。仿真结果表明，该 SRAM 结构能够在检测到软错误时，通过写回原地址进行修复；在检测到硬错误时，该结构通过写到冗余地址进行修复。在冗余地址使用完毕时，若再次探测到硬错误，该 SRAM 结构会输出硬错误探测信号。最后，本章基于 AHB 总线协议，对可修复软硬错误的 SRAM 存储器结构设计了 AHB 接口模块，并对其进行了系统级验证。结果表明，本文设计的可修复软硬错误的 SRAM 结构能够搭载到系统中去正常工作。

结　论

存储器芯片会在空间环境当中遭受到大量的辐射粒子，这些辐射粒子会导致芯片发生多位翻转的软错误和单粒子硬错误。本文主要根据两种错误的特点，对存储器进行加固，并修复存储器内发生的软错误和硬错误。主要研究内容总结如下。

（1）基于低冗余矩阵码的编译码原理，构建可纠正连续 4 位错误的（52，32）低冗余矩阵码为存储器的 ECC 电路。实现了该编码译码器硬件电路，利用注入故障的仿真方法，验证该电路功能的正确性。

（2）在实现（52，32）低冗余矩阵码编码译码器的基础上，设计了状态控制器，该控制器采用二次检测的方法区分软错误和硬错误类型。在故障注入平台上的仿真验证结果显示，搭载状态控制器的加固 SRAM 存储器可以修复软错误的同时探测到存储器单元内发生的硬错误，从而明确错误的类型。

（3）通过添加适当的冗余存储、改进状态控制器、设计故障地址分析器，建立硬错误地址的映射关系（映射到冗余地址）以及协调各模块的工作方式，提出了一种既能修复软错误又能隔离硬错误的 SRAM 存储器结构。在 SMIC65nm 工艺下，对该结构中各电路进行了逻辑综合并对整体 SRAM 结构完成了门级仿真，从而验证了本文所设计的 SRAM 存储器结构的正确性。

（4）对所提出的 SRAM 存储器结构设计了 AHB 接口，并根据总线功能模型，对该 SRAM 存储器结构进行了系统级验证。结果表明，该结构可以正确修复存储器内的软错误，并根据冗余地址用尽与否修复或探测硬错误。

综上，本文设计的 SRAM 存储器结构具备在线检测与修复功能。只要错误位数满足 ECC 电路的纠错能力，该 SRAM 结构就能够修复存储器单元内发生的软错误和硬错误，从而提高了 SRAM 存储器在空间环境应用的可靠性。

参考文献

[1] 陈伟，杨海亮，郭晓强等. 空间辐射物理及应用研究现状与挑战[J]. 科学通报，2017, 62(10):978-989.

[2] 韩郑生. 抗辐射集成电路概论[M]. 北京：清华大学出版社, 2011: 12-20.

[3] Ibe E, Taniguchi H, Yahagi Y, et al. Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule[J]. IEEE Transactions on Electron Devices, 2010, 57(7):1527-1538.

[4] Liu B, Du Y, Li Z, et al. Simulation Study of Node-State Transition Effect on the Single-Event Transient[J]. IEEE Transactions onDevice and Materials Reliability, 2015, 15(3):467-471.

[5] Huang P, Chen S, Chen J, et al. Simulation Study of Large-Scale Charge Sharing Mitigation Using Seamless Guard Band[J]. IEEE Transactions on Device and Materials Reliability, 2017, 17(1):176-183.

[6] 刘文平. 硅半导体器件辐射效应及加固技术[M]. 北京: 科学出版社，2013: 1-10.

[7] 王斌. 重离子辐照微纳级 SRAM 器件单粒子效应研究[D].北京: 中国科学院大学凝聚态物理博士学位论文，2017:16-20.

[8] Rajsuman R. Design and Test of Large Embedded Memories: an Overview[J]. IEEE Design and Test of Computers, 2001,18(3):16-27.

[9] 王丽君. 空间电子学的单粒子效应[J]. 空间电子技术, 1998(4): 1-5.

[10] Bentuotuo Y. A Real Time EDAC System for Applications Onboard Earth Observation Small Satellites[J]. IEEE Transactions on Aerospace & Electronic Systems, 2012, 48(1): 648-657.

[11] 刘海龙. 纳米工艺抗辐射加固集成电路设计研究[D]. 合肥: 合肥工业大学集成电路工程专业硕士论文, 2017: 1-5.

[12] T. Grasser, B. Kaczer, W. Goes, et al. The Paradigm Shift in Understanding the Bias Temperature Instability from Reaction-Diffusion to Switching Oxide Traps[J]. IEEE Transactions on Electron Devices. 2011, 58(11): 3652-3666.

[13] 闻昌. 基于 ECC 电路的软错误修复和诊断 NBTI 错误方法研究[D]. 哈尔滨:

哈尔滨工业大学集成电路工程硕士论文, 2016: 11-44.

[14] Gaspard N J, Jagannathan S, Diggins Z J, et al. Technology Scaling Comparison of Flip-Flop Heavy-Ion Single-Event Upset Cross Sections[J]. IEEE Transactions on Nuclear Science, 2013, 60(6): 4368-4373.

[15] 李家强. 基于正交拉丁码的存储器抗多位翻转设计[D]. 哈尔滨: 哈尔滨工业大学微电子学与固体电子学硕士学位论文, 2014: 1-45.

[16] Koga R, Crain W R, Lau D D, et al. On the suitability of non-hardened high density SRAMs for space applications[J]. IEEE Trans Nucl Sci, 1991, 38(6): 1507-1513.

[17] C. Dufour, P. Garnier, T. Carrière, J. Beaucour, R. Ecoffet, and M.Labrunée, Heavy ion induced single hard errors on submicronic memories[J], IEEE Trans. Nucl. Sci., 1992, 39(6): 1693–1697.

[18] 闫逸华, 范如玉, 郭晓强, 林东生, 郭红霞, 张凤, 祁陈伟. 电子元器件的微剂量效应研究进展[J]. 核技术, 2012, 35(9): 698-703.

[19] Calin T, Nicolaidis M, Velazco R. Upset hardened memory design for submicron CMOS technology[J]. IEEE Transactions on Nuclear Science, 1996, 43(6): 2874-2878.

[20] 郭瑞. 抗辐射 SRAM 单元及存储器设计[D]. 哈尔滨: 哈尔滨工业大学微电子学与固体电子学硕士学位论文, 2017: 19-22.

[21] 齐春华. CMOS 存储单元电路抗单粒子翻转加固设计研究[D]. 哈尔滨: 哈尔滨工业大学微电子学与固体电子学博士学位论文, 2018: 3-11.

[22] D'Alessio M, Ottavi M, Lombardi F. Design of a Nanometric CMOS Memory Cell for Hardening to a Single Event With a Multiple-Node Upset[J]. IEEE Transactions on Device and Materials Reliability, 2014, 14(1): 127-132.

[23] Blaum M, Goodman R, Mceliece R. The reliability of single-error protected computer memories[J]. Computers IEEE Transactions on, 1988, 37(1): 114-119.

[24] Saiz-Adalid L J, Reviriego P, Gil P, et al. MCU Tolerance in SRAMs Through Low-Redundancy Triple Adjacent Error Correction[J]. IEEE Transactions on Very Large Scale Integration Systems, 2015, 23(10): 2332-2336.

[25] Reviriego P, Pontarelli S, Evans A, et al. A Class of SEC-DED-DAEC Codes Derived From Orthogonal Latin Square Codes[J]. IEEE Transactions on Very

Large Scale Integration (VLSI) Systems, 2015, 23(5): 968-972.

[26] Mary Francy Joseph, Anith Mohan. Improved Architecture for Tag Matching in Cache Memory Coded with Error Correcting Codes[J]. Procedia Technology, 2016, 25: 544-551.

[27] Bota S A, Torrens G, Alorda B, et al. Cross-BIC architecture for single and multiple SEU detection enhancement in SRAM memories[C]// IEEE International On-line Testing Symposium. IEEE Computer Society, 2010: 141-146.

[28] Lin S, Daniel J, Costello J. 差错控制编码（原书第 2 版）[M]. 北京:机械工业出版社 2007:43-742.

[29] Sanchez-Macian A, Reviriego P, Maestro J A. Hamming SEC-DAED and Extended Hamming SEC-DED-TAED Codes Through Selective Shortening and Bit Placement[J]. IEEE Transactions on Device and Materials Reliability, 2014, 14(1):574-576.

[30] Neuberger G, Lima Fd, Carrol, et al. A Multiple Bit Upset Tolerant SRAM Memory[J]. ACM Transaction Design Automation of Electronic Systems, 2003, 8(4): 577-590.

[31] Namba K, Lombardi F. A single and adjacent symbol error-correcting parallel decoder for reed-solomon codes[J]. IEEE Transactions on Device & Materials Reliability, 2015, 15(1):75-81.

[32] Reviriego P, Argyrides C, Maestro J A, et al. Improving Memory Reliality Against Soft by Using Block Parity[J]. IEEE Transactions On Nuclear Science, 2011, 58(3):981-986.

[33] Argyrides C, Pradhan D K, Kocak T. Matrix Codes for Reliable and Cost Efficient Memory Chips[J]. IEEE Transactions on Very Large Scale Integration Systems, 2011, 19(3):420-428.

[34] Argyrides C A, Reviriego P, Pradhan D K, et al. Matrix-Based Codes for Adjacent Error Correction[J]. IEEE Transations on Nuclear Science, 2010, 57(4): 2106-2111.

[35] Liu S, Xiao L, Li J, et al. Low redundancy matrix-based codes for adjacent error correction with parity sharing[C]// 2017 18th International Symposium on Quality Electronic Design (ISQED). IEEE, 2017: 76-80.

[36] 柳姗姗. 基于错误纠正码的抗单粒子翻转存储器加固设计研究[D]. 哈尔滨: 哈尔滨工业大学微电子学与固体电子学博士学位论文, 2017: 23-44.

[37] Dufour C. Heavy ion induced single hard errors on submicronic memories (for space application) [J]. IEEE Trans Nucl Sci, 1992, 39(6): 1693-1697.

[38] C. Poivey, T. Carriere, J. Beaucour, and T. R. Oldham, Characterization of single hard errors (SHE) in 1M-bit SRAMs from single ion[J], IEEE Trans. Nucl. Sci., 1994, 41(6): 2235-2239.

[39] Koga R, Crain S, Crawford K, et al. Heavy ion induced hard errors in memory devices with sub-micron feature sizes[C]// Radiation and Its Effects on Components and Systems, 2001. 6th European Conference on. IEEE, 2001: 423-430.

[40] George J, Koga R, Crawford K, et al. SEE sensitivity trends in non-hardened high density SRAMs with sub-micron feature sizes[M]. IEEE Radiation Effects Data Workshop Rec., 2003: 83-88.

[41] Cellere G, Pellati P, Chimenton A, et al. Radiation effects on floating-gate memory cells[J]. IEEE Trans Nucl Sci, 2001, 48(6): 2222-2228.

[42] Avner Haran, Joseph Barak, David D, et al. Single Event Hard Errors in SRAM Under Heavy Ion Irradiation[J]. IEEE TRANSACTIONS ON NUCLEAR SCIENCE, 2014, 61(5): 2702-2710.

[43] Anna B. Boruzdina, Andrey V, et al. Microdose effects in SRAM cells under heavy ion irradiation[C]// 2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS), 2017: 1-3.

[44] 卞学愚. 基于 AHB 总线协议的 DMA 控制器设计[D]. 西安: 西安电子科技大学软件工程学科硕士学位论文, 2018: 7-14.

攻读硕士学位期间发表的论文及其它成果

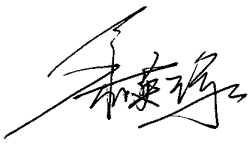
（一）发表的学术论文

[1] Yanqing Zhang, Yinghun Piao, Mingxue Huo, et al. Design and Verification of SRAM Self-Detection Repair Based on ECC and BISR Circuit[C]//2019 IEEE 26th International Symposium on the Physical and Failure Analysis of Integrated Circuits.（已收录）

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于 ECC 电路的 SRAM 自检测修复设计与验证》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：  日期：2019 年 6 月 21 日

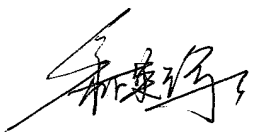
学位论文使用权限

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

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

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：  日期：2019 年 6 月 21 日

导师签名：  日期：2019 年 6 月 21 日

致　谢

两年匆匆而过，转眼毕业之时，内心思绪万千，久久不能平静。在我毕业论文即将完成之际，在此我向身边的各位老师、师兄师姐、同学和学弟致以最诚挚的谢意。

首先感谢我的导师霍明学教授，霍老师兢兢业业的治学态度和严谨细致的工作作风时时刻刻影响着我，激励我以认真的态度对待学业与工作。霍老师从我论文的选题到最后毕业论文的撰写，给予我宝贵的建议和巨大的帮助。霍老师待人富有热忱，谦虚谨慎，在生活上也对我们悉心照顾，无微不至。多少言语表达不尽我对霍老师的感激与敬意，能成为霍老师的学生，是我一生的财富。

特别感谢王天琦老师为我的论文倾注大量心血，在硕士期间，从小论文的撰写与审稿，到毕业论文的撰写，王老师都会尽全力提供宝贵的意见与帮助。促使我能不断地历练自己，完善自我。王老师精益求精的科研态度使我受益匪浅。

由衷的感谢齐春华老师为我论文提供帮助和审稿意见，是齐老师细心不耐其烦的指导，让我学到很多知识。齐老师耐心认真的科研态度始终是我学习的榜样。

感谢马国亮老师，刘超铭老师，张延清老师们的指导与建议。各位老师营造的和谐、积极、温馨的实验室氛围，给予我许多感动，使我有足够的动力和信心去完成我的学业。

感谢肖立伊老师，王进祥老师，来逢昌老师，王永生老师和付方发老师在我硕士学习期间在学业上的指导与提出的建议，是老师们的指正和批评让我能少走弯路，完善自己的设计方案。

非常感谢李杰师兄对我毕业设计思路上提出宝贵的意见；师兄的指导使我对设计有了正确方向；感谢柳姗姗师姐对我论文方案提供帮助和建议；感谢李何依师兄对我论文的建议；感谢我的六年同窗马建宁，我的室友冯开源，魏述琦和于斌对我学习和生活上的帮助与关心，感谢郭凯瑞和刘子靖为实验室做出的贡献。

对所有参加本论文答辩和评审的各位教授和老师表示最衷心的感谢！

最后感谢父母二十多年来对我的细心照料与默默支持，是家人的关怀和包容让我能踏踏实实地学习和进取，让我有足够的勇气去面对挑战。步入社会之后，我一定以最好的姿态回报父母对我的那份真挚的爱。